Adatbázis-kezelés

Az adatbázis-kezelés fogalmai

Civilizációnk fejlődése során egyre több tudást halmozott fel, közben egyre összetettebben működő társadalmat alakított ki. A kereskedelem, az adózás megjelenésével számtalan adat keletkezett, amelyeket gyakran táblázatos formában rögzítettek. Táblázatba foglalták az égitestek mozgására vonatkozó megfigyeléseket és az anyakönyvi eseményeket is.

Adót nem fizetett mindenki, a tudomány régen csak kevesek kiváltsága volt, de az egyházi, majd állami anyakönyvekbe gyakorlatilag mindenki belekerült. (Az Osztrák–Magyar Monarchiában 1895-ben tértek át állami anyakönyvezésre.) Születéskor, kereszteléskor rögzítették a dátumot, a gyermek és a szülők nevét. Bekerült a házasságkötés és a halálozás időpontja is. Ezek az anyakönyvek képezik az alapját a családfakutatásoknak. Minden rendelkezésre áll, mégsem könnyű néhány száz évre visszamenőleg összeállítani egy családfát. Nehézséget okoznak a névelírások és a papíralapú tárolás fizikai széttagoltsága.

Utóbbit megoldhatnánk, ha a különböző településeken, levéltárakban tárolt anyakönyvek tartalmát pontosan egyező szövegezéssel egy „könyvbe", azaz egyetlen szöveges állományba gyűjtenénk. Ezzel a keresési idő jelentősen csökkenne, de az adatrögzítési hibák nem tűnnének el. Egy elírt név továbbra is komoly fejtörést okozna, az egyező nevek szintén megnehezítenék a múlt megismerését. Jól látható, hogy nem elegendő pusztán egy új adathordozót használnunk, mert azzal ugyanazt csináljuk, csak gyorsabban, az igazi előrelépéshez szemléletet is kell váltanunk.

1. példa: A *diákok* adatbázis megtervezése

Nézzük meg egy példán lépésről lépésre, hogy miképpen alakíthatjuk át az egyszerű szöveges lejegyzést a hatékony adatfeldolgozás érdekében!

A szövegrészletben három diákról egy-egy mondat szerepel. Minden diákról ugyanazokat az ismereteket tartalmazzák a mondatok: hány éves, hol született, milyen nyelvet tanul, melyik iskolába jár, hány diák jár abba a suliba.

Nóra 13 éves, Budapesten született, angolul és németül tanul, a Piros iskolába jár, amely Budapesten található, és 652 diákja van.

***Pista** a neve annak az **angol** nyelvet tanuló, **11** éves diáknak, akinek születési helye **Sopron**, a **Piros** iskolába jár, amely **Budapest** városában található, és amelynek **652** diákja van.*

A Szolnokon született Bea 12 éves, a Kék iskolába jár, amely Szegeden található, és 541 diákja van.

Annyi mondatot írhatnánk, ahány diákot ismerünk. Milyen problémát rejt az adatrögzítésnek ez a módja? Az adatok közötti keresést nehezíti az eltérő megfogalmazás. Sokat segítene a rögzített szórend, hiszen tudnánk, hogy hova kell fókuszálni, ha például a budapesti születésűeket keressük. Néhány mondat után érezzük, hogy azokat a szavakat,

amelyek minden mondatban szerepelnek, fárasztó és felesleges leírni – régen sem tették, macskakörmözést használtak helyettük. Vegyük észre, hogy ezen ismétlődő szavak, kifejezések egy másik szóval alkotnak egy párt, például *név – Pista, születési hely – Sopron*. A szópárok egyik tagja egy tulajdonság, a másik pedig a tulajdonság adott diákhoz tartozó értéke. A második mondatban a tulajdonságot aláhúzással, az értéket félkövéren ki is emeltük.

A mondatok helyett táblázatot készíthetünk, ha az aláhúzott szavakat, kifejezéseket kiemeljük a fejlécbe, a félkövéren írtakat pedig a sorokon belül a megfelelő oszlopban tüntetjük fel.

név	**éves**	**nyelvek**	**születési hely**	**iskolanév**	**város**	**diákok száma**
Nóra	13	angol, német	Budapest	Piros	Budapest	652
Pista	11	angol	Sopron	Piros	Budapest	652
Bea	12		Szolnok	Kék	Szeged	541
…	…		…	…	…	…

Ezzel az áttekinthetőség sokat javult. Az első sorban a tulajdonságok elnevezése szerepel, ezeket **mezőnévnek** hívjuk. A mezőnév alatt a megfelelő tulajdonság egy-egy lehetséges értéke van megadva. A táblázat soraiban egy-egy diákhoz tartozó, tehát egymással összefüggésben – idegen szóval relációban – lévő **értékek** olvashatók. Az adatsorok neve szakszóval **rekord**. Ha egy táblázat minden oszlopában azonos szerepű értékek találhatók, az egyes sorok értékei pedig összefüggésben vannak egymással, **adattábláról** beszélünk.

A táblázatos forma logikus és áttekinthető. Érdemes azonban végiggondolni, hogy praktikusan választottuk-e meg a tartalmát. Vajon mennyire időtálló a tartalma? Néhány problémát könnyen felfedezhetünk:

- Nórából, Pistából még a Piros iskolában is sok lehet, hát még az országban. Segítene, de teljes mértékben még az sem oldaná meg a gondot, ha a vezetéknevüket ismernénk.
- Az idő múlásával változik az életkor, így szinte napról napra követnünk kellene, hogy kinek volt éppen születésnapja, kinek az életkorát kell módosítanunk.
- Az iskolaváltások megváltoztatják a létszámot, ezért ha Pistáék Szegedre költöznek, és ő a budapesti Piros iskola helyett a szegedi Kék iskolában folytatja tanulmányait, a 652 + 541 sorban kell módosítani a diákok számát. Hasonló probléma adódik akkor is, ha a nyolcadikosok helyét az elsősök veszik át szeptemberben.

E problémákat megoldhatjuk, ha

- rögzítjük a diákok oktatási azonosítóját is, hiszen az mindenkinek egyedi;
- nem az életkort, hanem a születési dátumot (esetleg évet) tartjuk nyilván;
- az egyes iskolákhoz tartozó diákok számát nem rögzítjük, mert – ha tényleg az ország összes diákjának adatait tároljuk – minden olyan esetben meg tudjuk számlálni, amikor szükség van rá.

A javított táblázat:

oktatási azonosító	név	nyelvek	születési dátum	születési hely	iskolanév	város
71234567890	Nóra	angol, német	2008.05.01.	Budapest	Piros	Budapest
71234564534	Pista	angol	2010.03.16.	Sopron	Piros	Budapest
71234553463	Bea		2009.06.12.	Szolnok	Kék	Szeged
…	…		…	…	…	…

Három problémát is kiküszöböltünk, de vajon tökéletes-e? Korántsem, hiszen ha az iskola a szomszéd településre költözne, akkor az összes oda járó diáknál meg kellene változtatni ezt a tulajdonságot. A fenti táblázat tehát nemcsak a diákokról tartalmaz adatokat, hanem az iskolákról is. Jobb, ha egy iskola települését, címét nem diákonként jegyezzük meg, hanem csak egyszer, az iskola nevéhez kapcsoltan. Rögzítsük külön táblázatban a diákok és az iskolák adatait!

Ahogy a diákoknál a név, önmagában az iskola neve sem határozza meg az iskolát, így az iskolához is adjunk hozzá egy egyedi értéket, azonosítót! Legyen ez az OM-azonosító! Ahhoz, hogy tudjuk, melyik diák melyik iskolába jár, rögzítsük a megfelelő OM-azonosítót a diák tulajdonságaként!

A nyelvek esetén érezhetünk még problémát, hiszen Nóra két nyelvet is tanul, Beánál pedig egy sem szerepel. Ha a diákokat tanult nyelv szerint keressük, akkor a többértékű mezők nehezíthetik a dolgunkat. Ha egy mezőnél több érték is szerepelhet, vagy előfordul üres érték, akkor javasolt külön táblázatban vagy táblázatokban tárolni ezeket az adatokat. Nézzük, hogy miképpen!

Diák táblázat

oktatási azonosító	név	születési dátum	születési hely	iskola
71234567890	Nóra	2008.05.01.	Budapest	789123
71234564534	Pista	2010.03.16.	Sopron	789123
71234553463	Bea	2009.06.12.	Szolnok	789225
…	…	…	…	…

Iskola táblázat

OM-azonosító	iskolanév	város
789123	Piros	Budapest
~~789123~~	~~Piros~~	~~Budapest~~
789225	Kék	Szeged
…	…	…

Nyelv táblázat

azonosító	név
1	angol
2	francia
3	német
…	…

Nyelvtanulás táblázat

azonosító	diák oktatási azonosító	nyelvazonosító
1	71234564534	3
2	71234567890	1
3	71234567890	3
…	…	…

Gondoljunk vissza a példamondatainkra! A „típusmondatot” könnyen átírtuk adattáblává, de a négy adattáblás végeredményig sok lépésből álló út vezetett. Az alábbiakban igyekszünk általánosabban megfogalmazni a lépéseket:

- Ne tároljunk olyan értéket, amely a többi tárolt adat alapján meghatározható (diákok száma)!
- Ne tároljunk olyan értéket, amely minden beavatkozás nélkül, automatikusan változik (életkor), cseréljük le olyanra, ami állandó!
- Minden adattáblában legyen egy azonosító szerepű tulajdonság (oktatási azonosító, OM-azonosító), akár egy sorszám (nyelv, nyelvtanulás), amely egyedi, ezáltal meghatározza a sor többi adatát!
- Ne legyen olyan oszlop, amelynek celláiba több értéket is bejegyzünk (nyelv), és lehetőség szerint kerüljük el az olyan oszlopokat is, ahol a cellákban (gyakran) nem szerepelnek értékek!

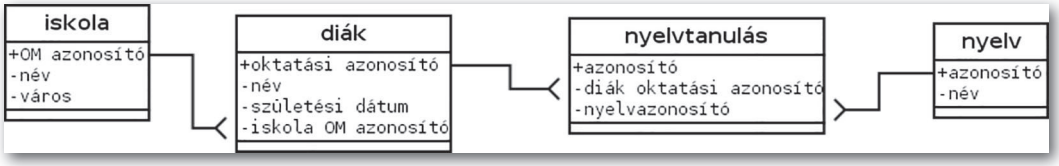
Ha valaki a fenti pontok szerint alakítja át a kiindulási táblázatát, akkor egy több adattáblából álló, jól használható struktúrához jut, amelyet **relációs adatmodellnek** is nevezik. (A reláció itt a soronként egymással kapcsolatban álló, összefüggő értékekre utal.) A szakemberek ezt az átalakítási folyamatot **normalizálásnak** nevezik. A normalizálás lépései a fenti tanácsok alapján kialakított struktúrát eredményezik. Mivel ebben a struktúrában az **adattáblák logikai kapcsolatban állnak egymással, adatbázisról** beszélhetünk. (Lásd a 10. évfolyamos tankönyvet.)

Az adatbázis általánosságban rendszerezetten tárolt adatok összessége, amely nemcsak relációs adatmodell alapján alkotható meg, hanem lehet hálós vagy hierarchikus is. (Olvassunk utána, mit jelent az adatbázis normalizálása!)

Néha előfordul, hogy praktikus okokból a fenti javaslatok egyikét-másikát figyelmen kívül hagyják, de mi ezt csak indokolt esetben tegyük. Az adatbázisok gyakran nem egy, nem négy, hanem több tucat táblából állnak, így nem könnyű jól használható normalizált adatbázist készíteni. A fenti példának nem az a célja, hogy profi szakemberré váljunk, hanem hogy könnyebben megértsük az általunk használt adatbázisok felépítését, és ezáltal hatékonyabban oldjuk meg a hozzájuk kapcsolódó feladatokat.

Relációs adatmodell kialakítása

A fenti táblázatokban az adatok csak példák, azt szolgálják, hogy könnyebben megértsük az egyes oszlopok szerepét. Aki rutinos az adatbázis-kezelésben, sokkal tömörebben is rögzítheti az adatbázis szerkezetét.



▶ A *diákok* adatbázis szerkezete (A rajz a Dia diagramszerkesztő programmal készült.)

A fenti ábra – mivel az adattáblák közötti kapcsolatokat is mutatja – az adatokat is tartalmazó változatnál több információt hordoz. Az azonosító szerepű tulajdonságokat a + karakterrel jeleztük. Ezt szokás **kulcsként** is nevezni. Látható, hogy az adattáblák közötti **kapcsolatok** a kulcsból indulnak, és a másik táblában, a kapcsolat túlsó végén ezek az elemek megjelennek egyszerű tulajdonságként. Utóbbit **idegen kulcsnak** nevezzük. A táblázatban az azonosító oszlopában minden érték pontosan egyszer szerepel, az idegen kulcs oszlopban pedig többször is előfordulhat. Ezt az összekötő vonal elágazásával jelezzük.

Vegyük észre, hogy az adattáblák kapcsolatának meghatározásánál azt kell eldöntenünk, hogy melyik azonosítóját kell beírnunk a másik táblába tulajdonságként.

Iskola–diák: Egy iskolába járhat-e több diák? Egy diák járhat-e több iskolába? Az első kérdésre igen a válasz, a másodikra nem. Ezért a diáktáblába kell felvennünk tulajdonságként az *iskola* tábla azonosítóját. A kapcsolatot megteremtő érték az egyik táblában csak egyszer fordulhat elő, a másikban többször is, ezért ezt **egy a többhöz kapcsolatnak** nevezzük.

Diák–nyelv: Egy diák tanulhat-e több nyelvet? Egy nyelvet tanulhat-e több diák? Mindkét kérdésre igennel válaszolunk. Ilyen, több a többhöz kapcsolat a relációs adatbázisban nem szerepelhet, ezért egy új táblát kell készítenünk, amelybe a két másik tábla azonosítóiból alkotott párokat jegyzünk be. Ezt a táblát **kapcsolótáblának** hívjuk.

Feladatok

1. Készítsünk adatbázis-szerkezetet a következő típusmondatokhoz!
   a) Svájcban a fizetőeszköz a frank, az ország államformája köztársaság, jelentősebb városai Zürich, Bern, Bázel.
   b) Az 1825-ben, Komáromban született Jókai Mór írta *A kőszívű ember fiai* és *Az arany ember* című regényeket is.
   c) Komáromi József Debrecenben született, jelenleg Szolnokon lakik, a következő e-mail-címeket használja: kjoco@komaromi.hu, komaromi.jozsef@munkahely.hu.
2. Milyen kapcsolatban állhatnak az alábbi táblák, melyikbe kell bejegyeznünk idegen kulcsként a másik azonosítóját?
   a) város és ország
   b) film és színész
   c) költő és költemény
   d) bolt és vásárló

Adatbázis a számítógépen

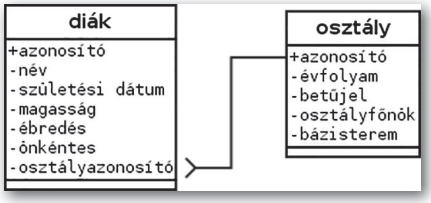
Az előző leckében megismertük, hogy egy adathalmaz tárolásához hogyan alakítunk ki **relációs adatmodellt.** Most megnézzük, milyen módon lehet a kialakított szerkezetben eltárolni az adatokat a számítógépen.

2. példa: Az *iskola* adatbázis megtervezése

Tekintsük az alábbi típusmondatot!

Nagy Nóra 2008. 05. 01-jén született, 1,51 m magas, minden tanítási napon 6:15-kor kel, vállal önkéntes segítői munkát, a 7. A osztályba jár, amelynek osztályfőnöke Virágh Zoltán, bázistermük a 231-es.

Ebből a képen látható szerkezetű adatbázist alkottuk.



▶ Az *iskola* adatbázis szerkezete

Próbáljuk megválaszolni az alábbi kérdéseket!

- Miért választottuk kétfelé az adatokat?
- Milyen szerepet tölt be a diáktáblában található osztályazonosító mező?
- Miért tároljuk két részletben az osztály megnevezésének évfolyam és betűjel részét?
- Miért nem az osztály megnevezése (7. A) azonosítja Csaba osztályát?
- Mikor döntenénk egy külön *tanár* vagy *terem* tábla létrehozása mellett? Hogyan befolyásolná a fenti rajzot?

Ha e szerint a relációs adatmodell szerint vetjük papírra az adatokat ténylegesen tartalmazó táblázatokat, akkor csak az oszlopok szélességére kell ügyelnünk. Ha számítógépes tároláson gondolkodunk, akkor nem hagyhatjuk figyelmen kívül **az adatok típusát** sem, hiszen az határozza meg, hogy milyen műveleteket végezhetünk rajtuk. A programozás során már megtapasztaltuk, hogy nem mindegy, szövegként vagy számként tárolunk-e egy adatot, és hogy számként egész vagy valós típust választunk-e.

Vegyük sorra, hogy az adatmodell egyes mezőihez milyen típust praktikus – vagy éppen kötelező – használni!

A diák *azonosítójaként* továbbra is az oktatási azonosítót használjuk. Ez minden esetben pontosan 11 számjegyből áll. Természetesnek tűnne, hogy szám típusúként tüntetjük fel, mégis szöveges típust kell alkalmazni. Gondoljunk arra, milyen műveletet végezhetünk ezzel az értékkel! Nincs olyan feladat, amelynél a számokon értelmezett matematikai műveleteket használhatnánk. Egy másik fontos észrevétel, hogy előfordulhat az ehhez hasonló azonosítók között olyan is, amely 0-s karakterrel kezdődik. (Az oktatási azonosító mindig 7-es számjeggyel kezdődik.) A vezető nullák tárolására szám esetén nincs mód. Tehát az oktatási azonosítót 11 karakteres szövegként adjuk meg.

A *név* tárolását egy mezőben végezzük. Ez nem magától értetődő, ugyanolyan jó megoldás lenne különválasztva tárolni a vezeték- és az utónevet. (Egyes esetekben még a családnevet megelőző dr., id., ifj. rövidítést is külön tárolják.) Ha nincs a felhasználás céljához tartozó indok a szétválasztásra, így sem követünk el hibát. Típusa szöveg. Ha az általunk használt programban szükséges megadnunk a hosszát, keressük meg az általunk ismert leghosszabb nevet, növeljük meg a hosszát az ötödével, és azt használjuk mezőhosszként! Más esetekben is érdemes hasonlóan túlbecsülni a méretet, hiszen előfordulhatnak az általunk ismertnél nagyobb helyet igénylő adatok.

A *születés dátumát* és az ébredés idejét természetesen dátum, illetve idő típusként rögzíthetjük.

A *magasság* méterben mért értéke nyilvánvalóan szám, mégpedig valós szám.

Az *önkéntesség* oszlopába igen vagy nem értéket lehet beírni, amelyet helyettesíthetünk az igaz/hamis szavakkal, de akár a 0 vagy 1/-1 értékekkel is. Ez logikai típus.

Nézzük meg, hogy az általunk tanult programozási nyelven az Igaz logikai érték és az 1 számérték egyenlő-e!

Az alábbi táblázatok alsó sorában feltüntettük, hogy milyen típust választhatunk a tároláshoz.

Diák tábla

azonosító	név	születési dátum	magasság	ébredés	önkéntes	osztály-azonosító
71234567890	Nagy Nóra	2008.05.01.	1,51	6:15	igen	23
…	…		…	…	…	…
szöveg (11)	szöveg (30)	dátum	valós szám	idő	logikai	egész szám

Osztály tábla

azonosító	évfolyam	betűjel	osztályfőnök	bázisterem
23	7	A	Virágh Zoltán	231
…	…		…	…
egész szám / számláló	egész szám	szöveg (1)	szöveg (50)	szöveg (10)

Próbáljunk válaszolni az alábbi kérdésekre!

- Miért tartunk fent nagyobb szélességű oszlopot az osztályfőnök neve számára, mint a diák neve számára?
- Miért szöveg típusú a bázisterem?
- Milyen széles oszlopot használnánk iskolánkban a bázisterem tárolására?

Típusok

A fenti példában csak az a néhány típus fordult elő, amely minden, középiskolában elterjedten használt programban elérhető, nincs is többre szükségünk a tanulmányaink során. E szoftverek mindegyike alkalmas arra, hogy az adatbázis-kezelési feladatokat – a programozáshoz hasonlóan – szöveges utasítások kiadásával oldjuk meg.

Az alábbi táblázat a parancssorban használt típusneveket tartalmazza.

	MS Access*	BASE	MySQL	PostgreSQL
szöveg	**varchar (n)** – *rövid szöveg*	varchar (n)	varchar (n)	character varying
egész szám	**integer** – *szám, hosszú egész*	integer	integer	integer
valós szám	**double** – *szám, dupla*	double	double	double precision
dátum	**date** – *dátum/idő*	date	date	date
idő	**time** – *dátum/idő*	time	time	time
logikai	**logical** – *igen/nem*	boolean	tinyint(1)	boolean
számláló	**counter** – *számláló*	integer identity	integer auto_ increment	serial

*A dőlt betűs típusneveket akkor használjuk, ha az adattáblát űrlappal készítjük.

Látható, hogy a típusnevekben alig van különbség. A később használt parancsok felépítésében sem találnánk túl sok eltérést a középiskolai tanulmányok szintjén előkerülő feladatokban.

Az MySQL és a PostgreSQL adatbázisai egy kiszolgálón találhatók, és a felhasználók sokféle programon keresztül, szövegesen megfogalmazott, SQL (Structured Query Language, azaz strukturált lekérdezőnyelv) nyelvű utasításokkal kezelik őket. Az SQL-nyelv alkalmas az adatbázis létrehozására, az adatok manipulálására, lekérdezésére és a hozzáférés szabályozására.

Az Access és a Base programok általában helyi gépen tárolt adatbázist használnak. A feladatok megoldásához nem szükséges parancsokat gépelnünk, de az adatlekérdező és -manipuláló műveleteket SQL-nyelven is megadhatjuk.

Ebben a könyvben az Access program segítségével mutatjuk be a feladatok megoldását, mert a legtöbb esetben egy jól áttekinthető felületen, kevés gépeléssel, a lényegre koncentrálva dolgozhatunk. A megoldást szemléltető képernyőképeket is ebben a programban készítettük. (A Microsoft Access program 2019-es verzióját használtuk, de ez alig tér el a 2013-as és 2016-os verzióktól.)

Adatbázis létrehozása

Hozzuk létre az általunk használt adatbázis-kezelő rendszerben az *iskola* adatbázist!

A számítógépen létrehozott adatbázisba minden esetben beleértjük az adattáblákat – az azokat alkotó mezők nevével, típusával, a kulcs megjelölésével –, a köztük lévő kapcsolatokat és a bennük tárolt adatokat is.

Az Access programban az adatbázis állományában tároljuk

- az adatfelvitelhez készített felületeket (űrlapokat),
- az adatok lekérdezését és manipulálását biztosító utasításokat (lekérdezéseket),
- az esztétikus megjelenítést, nyomtatást segítő felületeket (jelentéseket).

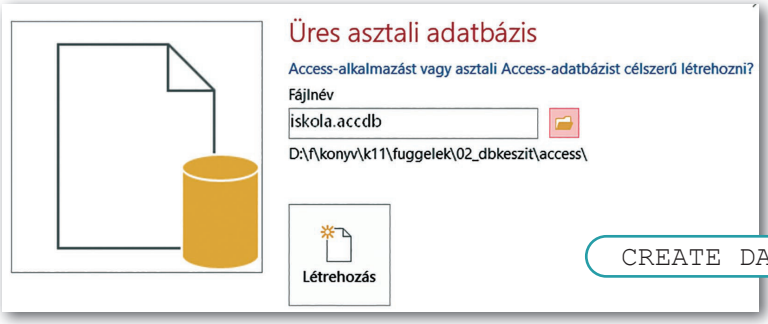
Az alábbiakban az *iskola* adatbázis létrehozását mutatjuk be. A grafikus felületen elvégzett műveletek mellett feltüntetjük az SQL-nyelvi megfelelőt is, amely a típusok nevének megfelelő módosításával a többi programban is használható.

Az Access program, az irodai programcsomag többi elemével szemben, új adatbázis készítésekor azonnal kikényszeríti a mentést. Így a későbbiekben magát az adatbázist nem kell rendszeresen mentenünk, az adatvesztés veszélye kisebb, mint például a táblázatkezelőben.

A tankönyvben a táblaneveket és a mezőneveket, valamint a lekérdezések, a jelentések és az űrlapok nevét az egyszerűbb kezelhetőség érdekében egybeírva, néhol ékezetek nélkül adjuk meg.

3. példa: Az *iskola* adatbázis létrehozása

Hozzuk létre az adatbázist!




```
CREATE DATABASE iskola;
```

▶ Az *iskola* adatbázis létrehozása

Hozzuk létre a *diák* és az *osztály* táblákat!

Ne feledkezzünk meg a kulcs beállításáról sem! Az egyes mezők esetén figyeljük meg a mezők listája alatt az általános tulajdonságokat, ahol a szöveges mező hosszát, valamint a szám valós vagy egész voltát állíthatjuk be.


```
CREATE TABLE diak (
  azonosito VARCHAR(11) PRIMARY
  KEY,
  nev VARCHAR(30),
  szuldat DATE,
  magassag DOUBLE,
  ebredes TIME,
  onkentes LOGICAL,
  osztalyazonosito INTEGER);
```

▶ A *diák* tábla létrehozása

A táblatervező nézetében az elsődleges kulcsot utólag is megjelölhetjük az eszköztár vagy a helyi menü segítségével:


```
ALTER TABLE diak ADD PRIMARY KEY(azonosito);
```

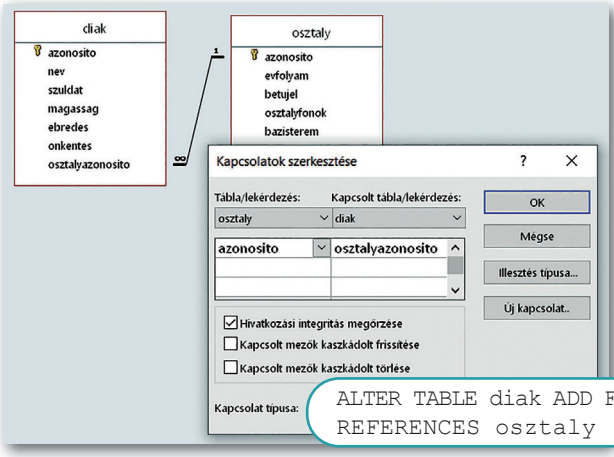
► Azonosító létrehozása a *diák* táblában

Az *osztály* táblában az elsődleges kulcs egy sorszám, amelyet számláló típusként kell rögzítenünk:


```
CREATE TABLE osztaly (
  azonosito INTEGER
  AUTO_INCREMENT PRIMARY KEY,
  evfolyam INTEGER,
  betujel VARCHAR(1),
  osztalyfonok VARCHAR(50),
  bazisterem VARCHAR(10));
```

► Az *osztály* tábla létrehozása

A táblák közötti kapcsolatot az *Adatbáziseszközök* rész *Kapcsolatok* pontját választva tudjuk beállítani. A hivatkozási integritás megőrzése biztosítja, hogy csak olyan osztályba rögzítsünk diákot, amelyik létezik is.




```
ALTER TABLE diak ADD FOREIGN KEY (osztalyazonosito)
REFERENCES osztaly (azonosito);
```

► Idegen kulcs létrehozása a *diák* táblában

Töltsük fel adatokkal a táblákat! Vigyázzunk arra, hogy a mintán látható 23-as azonosítójú osztályt csak a beszúró parancs segítségével lehet rögzíteni, a kézi adatfelvitel során a sorszámozás az 1-es értékkel kezdődik.

osztaly

azonosito	evfolyam	betujel	osztalyfonok	bazisterem	Hozzáadás
23	7	A	Virágh Zoltán	213	
24					
(Új)	0				

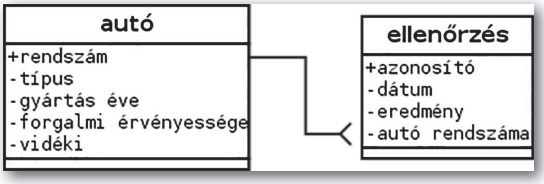
```
INSERT INTO osztaly (azonosito, evfolyam, betujel, osztalyfonok, bazisterem)
VALUES (23, 7, ,A', ,Virágh Zoltán', ,213');
INSERT INTO diak (azonosito, nev, szuldat, magassag, ebredes, onkentes,
  osztalyazonosito)
VALUES (,71234567890', ,Nagy Nóra', ,2008-05-01', 1.51, ,6:15:00', 1, 23);
```

▶ Adatbevitel az *osztály* táblába

Feladatok

Vizsgáljuk meg a következő típusmondatot: Az ABC-123 rendszámú, Kia típusú, 2013-ban gyártott autót, amelyet Budapesten használnak, 2018. 10. 08-án és 2020. 12. 13-án ellenőrizték, mindkétszer megfelelt.

A fenti mondathoz az adatbázis alábbi relációs adatmodelljét készítettük.



▶ A *taxi* adatbázis szerkezete

Oldjuk meg a következő feladatokat!

a) Milyen típusú és méretű mezőket kell használnunk az egyes táblákban?
b) Helyes-e az ábrán a kapcsolat iránya? Miért?
c) Hozzuk létre az adatbázist az általunk használt eszköz által biztosított grafikus felületen!
d) Állítsuk be az idegen kulcsot is!
e) Melyik táblába kell bevinnünk az első rekordot? Válaszunkat indokoljuk!
f) Vigyünk fel legalább két rekordot az *autó* táblába, és legalább öt rekordot az *ellenőrzés* táblába!
g) Töröljük az utoljára felvitt ellenőrzést, és vegyünk fel egy újabbat! Milyen azonosítót kapott az új rekord?
h) Próbáljunk meg felvenni olyan rendszámot az *ellenőrzés* táblába, amely nem szerepel az autó rekordjai között! Értelmezzük a hibaüzenetet!

Adatok importálása

Az előző leckében az *iskola* adatbázist mi hoztuk létre, és mi rögzítettünk benne néhány adatot. Célunk annyi volt, hogy megismerjük a különböző típusú adatok bevitelének módját és az adatbevitel sorrendjét. A legtöbb adatbázisba a valóságban is kézi adatbevitellel vagy legalábbis emberi közreműködéssel rögzítik az adatokat. Egy webáruház óriási kínálatát bizony árucikkenként viszik fel az alkalmazottak, és a vásárlók is minden megrendeléssel újabb és újabb rekorddal bővítik a táblák tartalmát. A tanulás során azonban nem hagyatkozhatunk kézi adatfelvitelre, mert nagyon sok időbe telne, mire akkora rekordszámú adattáblákat állítanánk elő, amelyeknél indokolt az adatbázis-kezelő használata.

A későbbiekben általában készen kapjuk az adattáblák feltöltéséhez szükséges adathalmazt, ritkábban pedig mi magunk állítjuk elő.

Adatbázis feltöltése szövegfájlból

4. példa: A *város* adatbázis megtervezése

Készítsük el az alábbi típusmondathoz tartozó adatbázist!

Hódmezővásárhely megyei jogú város, területe 487,98 km², népessége 43 311 fő, Csongrád-Csanád megyében fekszik, amelynek székhelye Szeged.

Természetesen átalakíthatjuk táblázatos formára a mondatot, de talán nem olyan bonyolult néhány kérdés megválaszolása után a relációs modell azonnali felrajzolása sem.

- Szerepel-e olyan adat, amely automatikusan, más esemény hatása nélkül megváltozik?
- Szerepel-e olyan adat, amely a többi adatból következik?

Mindkettőre nem a válasz.

- Hány „dologról” szól a mondat?

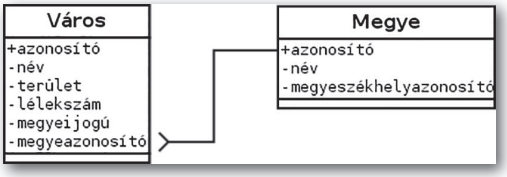
A városról és a megyéről szól. Tehát két táblát kell felrajzolnunk.

- Vajon a városnál kell-e felvennünk a megye azonosítóját idegen kulcsként, vagy a megyénél a városét?

Ehhez két kérdést kell feltennünk:

- Tartozhat-e egy megyéhez több város?
- Tartozhat-e egy város több megyéhez?

Ha csak az egyik kérdésre válaszolhatunk nemmel (itt a másodikra), akkor abba a táblába *(város)* kell bejegyeznünk a másik azonosítóját.



► A *városok* adatbázis szerkezete

A fentiekhez a képen látható adatbázis-szerkezetet választottuk:
Ha csak a rajzot látjuk, a két adattáblát összekötő vonal elágazásáról leolvasható, hogy egy megyéhez több város tartozik.

Határozzuk meg, hogy milyen típusúak az egyes mezők!

Az *azonosítók* típusa legyen **számláló,** mivel nincs olyan elterjedten használt tulajdonság, amely azonosítaná a várost vagy a megyét. (Ha a megyenév lett volna a kulcs, sok helyen kellett volna értéket módosítani, amikor 2020. június 4-én Csongrád megye nevét Csongrád-Csanádra változtatták.) A *nevek* legyenek **szövegesek,** méretüket pedig határozzuk meg a rögzíteni kívánt adatok alapján! A *lélekszám* és a *terület* **szám** típusú legyen, előbbi **egész,** utóbbi **valós.** A *megyeijogú* mezőben azt kell tárolnunk, hogy az adott település megyei jogú-e, vagy sem, így **logikai** típust kell választanunk. Az idegen kulcs típusát a másik tábla azonosítójának típusa határozza meg, azzal egyezőnek kell lennie.

Oldjuk meg az alábbi feladatokat a már megismert adatbázis-kezelő rendszerben!

- Hozzuk létre az adatbázist!
- Készítsük el a két táblát! (Ez grafikus felületen is megtehető, nem fontos SQL-utasításként.)
- Adjuk meg az idegen kulcsot! (Határozzuk meg a táblák közötti kapcsolatot!)
- Vigyünk be egy-egy adatsort a táblákba a mondatban olvasható adatok alapján!
- Vigyük be még egy megye és a megye néhány városának adatait!

Az adatbázist létrehozó SQL-utasításokat a források között `varosok.sql` néven helyeztük el.

A táblák létrehozása után válaszoljuk meg a következő kérdéseket!

- Miért a megye rögzítésével kellett kezdenünk?
- Mikor rögzíthetjük a *megye* táblában a *megyeszékhelyazonosító* értékét?
- Vajon megadhatjuk-e egy újabb idegen kulcsként a *megyeszékhelyazonosító* mezőt? Indokoljuk meg a választ! (Ha nem tudunk dönteni, próbáljuk ki, milyen hatása lesz, ha megadjuk.)

A végső cél természetesen az összes város és megye sémában szereplő adatainak rögzítése. Mivel háromszáznál is több város van, ezért az adatok összegyűjtése és begépelése időigényes. Ha az adattáblák tartalma szövegfájlban rendelkezésre áll, akkor az néhány lépésben beemelhető az adatbázisba.

A szükséges adathalmazt a Wikipédián megtaláljuk, így csak a megfelelő adatok táblázatba rendezése igényel munkát. Ezt végigcsinálhatjuk a következő példa alapján, vagy használhatjuk a források `varos.txt` és `megye.txt` állományait!

5. példa: Egy Wikipédia-oldal táblázattá alakítása

Látogassunk el a *https://hu.wikipedia.org/wiki/Magyarország_városai* lapra! (Ha éppen nem elérhető, használjuk a fejezethez tartozó állományok közül a `Magyarország városai – Wikipédia.htm` fájlt!)

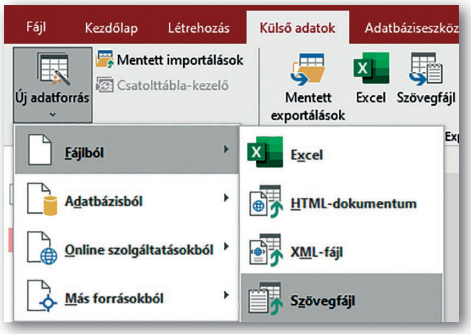
Az adatforrás két táblázattá alakításához alkalmazzuk a táblázatkezelés kapcsán tanultakat!

Egy lehetséges megoldás:

1. Másoljuk át a forrásban található táblázatot egy munkafüzet forrásmunkalapjára!
2. Töröljük a felesleges oszlopokat (név, típus, megye, lélekszám, terület maradjon)!
3. Vegyük fel az *azonosító* és a *megyeijogú* oszlopokat!
4. Az azonosító oszlopát töltsük fel számokkal!
5. A *megyeijogú* oszlopban megfelelő szűrés (szövegszűrő) után kézzel jegyezzük be az értékeket! (Használhatunk sorba rendezést vagy arra alkalmas függvényt is.)

6. A megyéket a *megye* oszlop adatainak felhasználásával vagy a megyeszékhelyre való szűréssel is előállíthatjuk. Ellenőrizzük, szerepel-e minden megye!
7. A városoknál a megyeazonosítókat határozzuk meg alkalmas függvénnyel!
8. A városokra és a megyékre vonatkozó adatokat másoljuk át két külön szövegfájlba (`varos.txt`, `megye.txt`)!

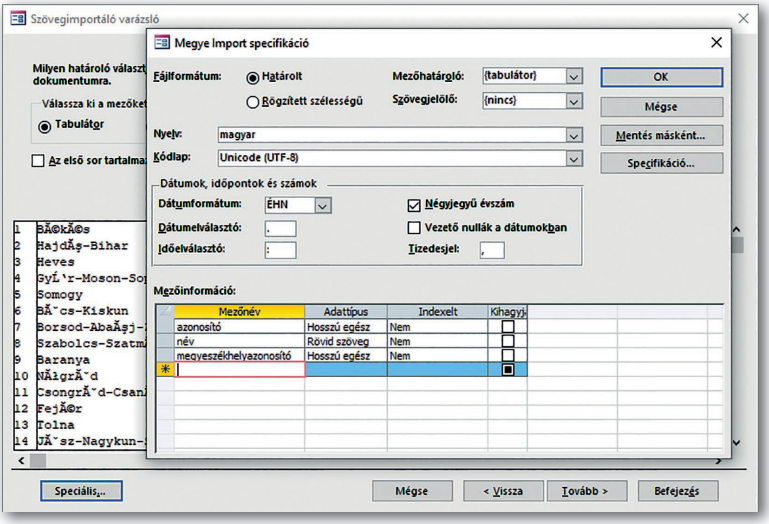
Az általunk előállított vagy a készen kapott szövegfájlok tartalmát helyezzük el a megfelelő adattáblákban! Ezt a műveletet **importálásnak** nevezzük.



► Importálás szövegfájlból

Az importálás során az alábbi lépéseket hajtsuk végre, de előtte ellenőrizzük, hogy a mezők sorrendje a fájlban és az adattáblában egyezik-e.

- Válasszuk a *Külső adatok* > *Új adatforrás* > *Fájlból* > *Szövegfájl* menüpontot, majd válasszuk ki a `megye.txt` fájlt!
- Mivel a tábla már létezik, ezért használjuk a *Rekordok másolatának hozzáfűzése* lehetőséget a megfelelő táblát választva!
- A megjelenő párbeszédablakban a *Speciális* gomb segítségével állítsuk be, hogy milyen mezőhatárolót használunk – ez általában tabulátor –, és adjuk meg a kódlapot, amely szinte mindig UTF-8.



► Az importált szövegfájl tulajdonságainak beállítása

Importáljuk a `varos.txt` állomány adatait a *varos* táblába!
Az importálást követően nézzük meg a táblák tartalmát! Fontos, hogy a szöveges adatok ékezethelyesek legyenek, elférjenek a megfelelő mezőben, a terület értékeinél pedig a tizedesjegyek szerepeljenek.

A *városok* adatbázis tartalmát ritkán bővítjük, hiszen csak akkor kell rekordot hozzáadnunk, ha újabb települést nyilvánítanak várossá. (Nézzünk utána, mik a várossá nyilvá-

nítás feltételei!) Ennek ellenére sem változtathatatlan adathalmazról beszélünk, hiszen a lélekszám nem állandó. A valósan használt adatbázisról az adatok frissítését követően vagy rendszeres időközönként készítsünk biztonsági mentést! Az Access program esetén ez az adatbázist tartalmazó fájlról készített másolat, a legtöbb adatbázis-kezelő rendszer esetén pedig az adatbázis exportja. Az exportálás eredménye egy olyan szöveges fájl, amely azokat a parancsokat tartalmazza, amelyekkel az adatbázis szerkezete és adatai később helyreállíthatók. A `varosok_export.sql` fájlban található meg a *városok* adatbázis MySQL-ben készült változatának mentése. Ezt a fájlt akkor is érdemes végignéznünk és megbeszélnünk, ha nem foglalkozunk mélyebben az adatbázis-kezeléssel.

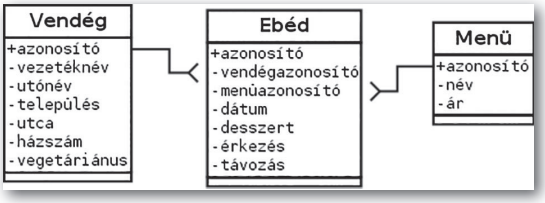
Importálás másképp

6. példa: Munka az *étkezés* adatbázissal

Az *étkezés* adatbázisban egy étterem menüjére előfizető vendégek adatait tároljuk. Nyilvántartják a számlázáshoz szükséges adatokat (név, cím), valamint azt, hogy a menü vegetáriánus-e. Mindennap négyféle menüből választhatnak a vendégek, ezek ára állandó. Eltároljuk azt is, hogy a vendégek melyik nap melyik menüt választották, kértek-e aznap desszertet, mikor érkeztek és távoztak. (Csak a 2020. év októberének adatait tartalmazza a hozzáférhető adathalmaz.)

Az adatbázis feltöltéséhez szükséges adatokat a `vendeg.txt`, `menu.txt`, `ebed.txt`, UTF-8 kódolású, tabulátorral tagolt szövegfájlok tárolják, amelyek első sora nem tartalmazza a mezőneveket.

Ehhez a képen látható adatbázis-szerkezetet használjuk.



► Az *étkezés* adatbázis szerkezete

A táblákat nem szükséges előre elkészítenünk, létrehozhatjuk az importálás során is. Ehhez a *Rekordok másolatának hozzáfűzése* helyett a *Forrásadatok importálása új táblába* pontot kell választanunk. A *Speciális* gombra kattintást követően a korábban megismertek mellett a mezőinformációs részben meg kell adnunk a mezőneveket. Az *Importálás varázsló* későbbi lépéseiben az azonosító (kulcs) beállítását szükséges még elvégeznünk. Ha már mindhárom táblát importáltuk, az idegen kulcsot a hivatkozási integritást megőrző kapcsolat megadásával állíthatjuk be.

Feladatok

Készítsük el a *javítás* adatbázist az alábbi leírásnak megfelelően! Az adatbázis feltöltéséhez szükséges adatokat az `ugyfel.txt`, `munka.txt`, `tipus.txt` nevű, UTF-8 kódolású, tabulátorral tagolt szövegfájlok tárolják, amelyek első sora tartalmazza a mezőneveket.

a) A mezőnevek tartalmát felhasználva rajzoljuk meg a *javítás* adatbázis szerkezetét!
b) Értelmezzük az egyes mezők szerepét, határozzuk meg a típusukat!
c) Milyen típust választottunk az ügyfél *irszám* mezőjének? Indokoljuk a választ!
d) Importáljuk a három szövegfájl tartalmát az adatbázisba! (A varázsló második lépésében ne felejtsük el megjelölni, hogy az első sor tartalmazza a mezőneveket!)
e) Állítsuk be az idegen kulcsokat a kapcsolatok létrehozásával! (A mezőnevek segítik a kapcsolatok meghatározását.)

Információ kinyerése az adattáblákból

Táblázatkezelés során általában egyetlen munkalap egy téglalap alakú tartományából nyertünk ki adatokat. Ennek során rendezhettük az adatokat, kereshettünk a munkalapon, vagy éppen a szűrésnek valamelyik változatát is segítségül hívhattuk.

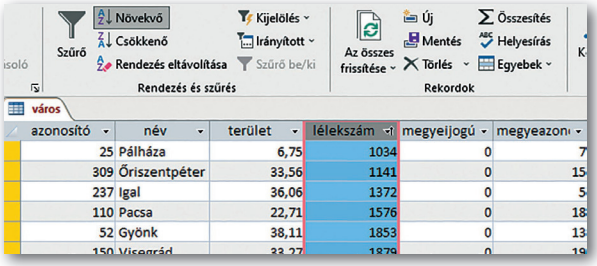
7. példa: Válaszok más szoftverekben is használt módszerekkel

Idézzük fel, hogy táblázatkezelő segítségével milyen kérdésekre tudnánk válaszolni ezekkel a műveletekkel! Ugyanezeket a problémákat az adatbázis-kezelőben is megoldhatjuk.

A már korábban elkészített, de a források között is megtalálható *városok* adatbázist használva adjuk meg, hogy

- melyik a legkisebb lélekszámú város;
- mekkora Komló területe;
- mely települések indulhatnak azon a pályázaton, amelyet 10 és 30 ezer fő közötti lélekszámú, legalább 250 km^2 területű városoknak írtak ki!

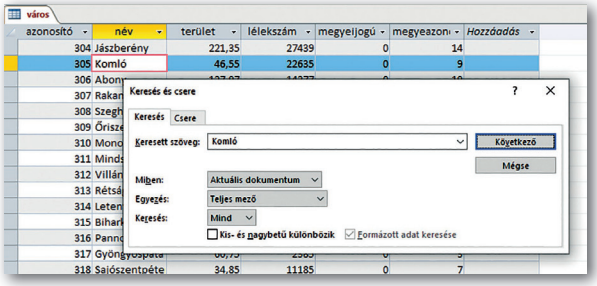
Rendezés



azonosító	név	terület	lélekszám	megyeijogú	megyeazoni
25	Pálháza	6,75	1034	0	7
309	Őriszentpéter	33,56	1141	0	15
237	Igal	36,06	1372	0	5
110	Pacsa	22,71	1576	0	18
52	Gyönk	38,11	1853	0	13
150	Visegrád	33,27	1879	0	19

A legkisebb lélekszámú várost a lélekszám oszlopára beállított növekvő rendezéssel lehet a legegyszerűbben meghatározni. A megjelenítés sorrendjét mindig a legutolsó rendezés szabja meg. Rendezés nélküli táblában az elsődleges kulcs a meghatározó.

Keresés



A keresés szóról szinte minden számítógép-használónak a `Ctrl + F` billentyűkombináció jut eszébe. A megnyitott táblában itt is használhatjuk.

Példaként keressük meg Komló adatait!

A keresés során figyeljünk a *hatókör* és az *egyezés* megfelelő beállítására! Próbáljuk megtalálni az „aba”, majd a „91” összes előfordulását a teljes mezővel egyezésben, majd bármely részében! Teljes egyezésnél mindkét esetben csak egy rekordot találunk, a mező bármely részét figyelve többet. Tehát az ilyen keresés nem más, mint a megadott karaktersorozat keresése – akár egy szám részeként.

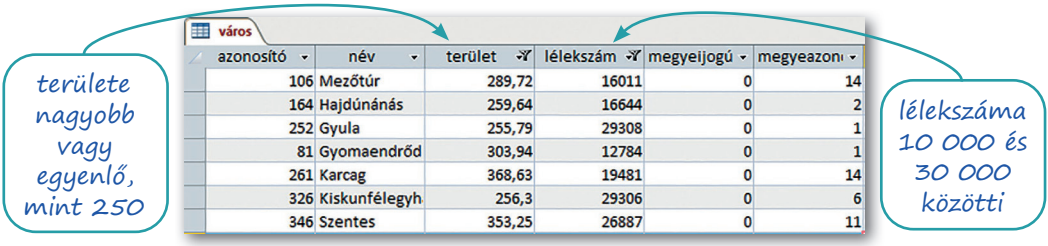
Keresés során a beírt szöveget tartalmazó rekordokon végig tudunk lépdelni, de az összes ilyen adatsort nem látjuk egyszerre. A keresésnél csak egy értéket adhatunk meg. Ha több feltételt is meg akarunk adni, ahhoz a szűrés funkcióra van szükségünk.

Szűrés

A szűréssel elérhetjük, hogy egyetlen táblának csak azokat a sorait jelenítse meg a program, amelyek az általunk meghatározott feltételnek megfelelnek. A táblázatkezelő helyben szűréséhez hasonlóan csak **ÉS** műveletet tartalmazó logikai kifejezést alakíthatunk ki szűrési feltételként.

Egy pályázaton csak azok a városok indulhatnak, amelyek lélekszáma 10 és 30 ezer fő között van, területük pedig legalább 250 km^2. Szűrjük ki ezeket a városokat!

Ehhez meg kell nyitni a táblát, és a megfelelő mezőnévre kattintva, a számszűrőknél beállítani a kívánt feltételt. A 10 és 30 ezer fő közötti lélekszámot megadhatjuk két részletben vagy intervallumszűréssel is. Érdemes megjegyezni, hogy az *„a és b érték között”* az adatbázis-kezelés során olyan értékekre teljesül, amelyekre igaz, hogy ≥*a* és ≤*b,* tehát a matematikában használt [*a;b*] zárt intervallumnak felel meg.



► Szűrési feltétel összeállítása

Hiába végzünk el egy szűrést, a táblát bezárva, majd ismét megnyitva az összes rekord megjelenik. Tehát a szűrés – a kereséshez hasonlóan – csak gyors tájékozódásra szolgál, a kapott adathalmazt áttekinthetjük vagy lemásolhatjuk, de – például újabb rekordok rögzítését követően – nem tudjuk ismét alkalmazni.

8. példa: Szövegből logikai kifejezés

Oldjuk meg a következő feladatokat rendezéssel, kereséssel vagy szűréssel!

1. Melyik a legkisebb területű város?
2. Melyik város neve az utolsó, és melyik az első az ábécérendben?
3. Mely városok területe nem haladja meg a 10 km^2-t? Hány ilyen város van?
4. Mely megyei jogú városok lélekszáma legfeljebb 100 ezer fő?
5. Mely 20 ezer főnél népesebb városoknak kisebb a területük 100 km^2-nél?
6. Mely 8 ezer főnél kevesebb lakosú városok területe 25 és 50 km^2 közötti? Hány ilyen város van?
7. Mely városokra nem teljesül, hogy népességük 10 ezer és 50 ezer fő között van?

Elég könnyű feladatokról van szó. Sokan úgy gondolják, hogy sikerült mindet megoldani. Nézzük meg alaposabban! Írjuk fel a feltételekben megfogalmazott logikai kifejezéseket a matematikában és a programozásban megszokott formában!

területe nem haladja meg a 10 km^2-t	terület ≤ 10
megyei jogú ÉS lélekszáma legfeljebb 10 ezer	megyei jogú = Igaz ÉS lélekszám ≤ 10000
100 km^2-nél kisebb ÉS 20 ezer főnél népesebb	terület < 100 ÉS lélekszám > 20000

Ha leellenőrizzük az utolsó sorban foglaltakat a szoftverben, akkor azt látjuk, hogy bizony ott szerepel az egyenlőség is. Ezen adatoknál nem tűnik fel a hiba, de érezzük, hogy adott esetben komoly gond lehet abból, ha csupán egy település is elesik egy pályázattól amiatt, mert egy egyenlőségjel lemaradt.

Szükségünk van egy eszközre, amelynek használata során a feltételt teljes pontossággal meg tudjuk fogalmazni. Ezt az eszközt lekérdezésnek nevezzük.

Lekérdezés

A lekérdezés megoldja az imént használt műveletekkel kapcsolatban felismert problémákat, tehát

- megváltozott tartalmú táblára is tudjuk alkalmazni,
- a szűrési kifejezés pontosan megadható,
- a szűrési kifejezésben nem csak az **ÉS** logikai művelet használható.

Lekérdezéssel a rendezési, keresési, szűrési feladatok megoldhatók. Az eddig megismert eszközöket természetesen továbbra is használhatjuk, de legyünk tudatában korlátaiknak.

A későbbiekben látni fogjuk, hogy a lekérdezésben nem csak egy táblát használhatunk, illetve lekérdezés segítségével számításokat is végezhetünk.

A lekérdezés legegyszerűbb formáját a szűréssel tekinthetjük egyenértékűnek: egy adattáblára vonatkozik, és a szűrési feltételnek megfelelő rekordok összes mezőjének értékét megjeleníti.

Ha az előző kérdésekre akarunk választ kapni, akkor a

```
SELECT *
FROM város
WHERE [feltétel]
```

formát kell használnunk. Ebben az alakban a félkövér karakterrel írtak mindig szerepelnek, az utolsó elemet kell helyettesítenünk a keresési feltétellel. A `SELECT *` azt jelenti, hogy a rekord minden mezőjét meg kell jeleníteni, a `FROM város` pedig megadja, hogy a *város* táblával dolgozunk.

9. példa: Feltételek megfogalmazása a lekérdezésekben

Nézzük meg, mit írjunk az állandó rész mögé a 8. példában szereplő, sorszámozott feladatok esetén:

```
3.… terület <= 10
4.… megyeijogú = Igaz AND lélekszám <= 100000
5.… lélekszám > 20000 AND terület < 100
6.… lélekszám < 8000 AND terület >= 25 AND terület <= 50
7.… NOT(lélekszám>= 10000 AND lélekszám <= 50000)
```

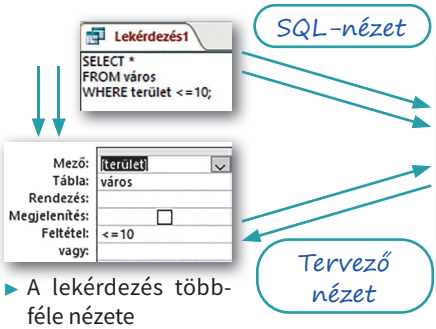
vagy – ha végiggondoljuk (akár egy számegyenes segítségével ábrázoljuk):

```
    … lélekszám < 10000 OR lélekszám > 50000
```

Ha mindez világos, akkor nézzük meg, miképpen használhatjuk az Access programban! Természetesen ugyanabban az adatbázisban dolgozunk, mint ahol a szűréseket végeztük. A *Létrehozás* > *Lekérdezéstervező* eszközt használva megnyílik egy *Lekérdezés1* nevű objek-

tum. A megjelenő párbeszédablakot most még hagyjuk figyelmen kívül, és a *Lekérdezés1* helyi menüjéből válasszuk az *SQL-nézet* pontot!

A párbeszédablakba írjuk be 3. feladat feltételt is tartalmazó parancsát, majd kattintsunk a *Lekérdezéstervezés* > *Futtatás* elemére!

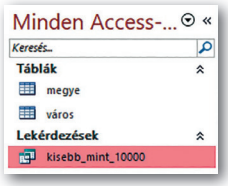


▸ A lekérdezés többféle nézete

▸ Adatlap nézet

A kapott lista egyezik a szűrés eredményével. Az Access programban a lekérdezésnek ezt az állapotát *Adatlap nézetnek* nevezzük. A lekérdezések helyi menüjében szabadon váltogathatunk a különböző nézetek között. Válasszuk ki a *Tervező nézetet!*

Tervező nézetben a lekérdezés ablakának alsó – táblázatos – részén, amelyet **lekérdezőrácsnak** hívunk, a mező sorában az a név látszik, amelyre a feltételt megfogalmaztuk, a feltétel sorában szerepel a relációs jel és a viszonyítási érték. Ez a két elem együttesen adja meg a korábban felírt feltételt.



A lekérdezést elmenthetjük, ha a nevére jobb gombbal kattintunk. Az elmentett lekérdezések az adatbázis ablakának bal oldalán megjelennek, nevükre duplán kattintva bármikor végrehajthatjuk azokat. A végrehajtással egyenértékű a *helyi menü* > *Megnyitás* lehetősége. A lekérdezés módosításához pedig a *helyi menü* > *Tervező nézet* pontját kell használnunk.

Bár minden lekérdezést megfogalmazhatunk SQL-nyelven, a későbbiekben elsősorban a lekérdezőrácson – *Tervező nézetben* – készítjük el a lekérdezéseket.

Feladatok

Próbáljuk megoldani az alábbi feladatokat lekérdezés készítésével és annak használatát mellőzve is! Nem kell megijednünk, ha csak az egyik módon járunk sikerrel, előfordul, hogy nem lehetséges, vagy még nincs minden szükséges ismeret a birtokunkban.

Használjuk az *étkezés* adatbázis tábláit!

a) Jelenítsük meg a vendégeket ábécérendben!
b) Adjuk meg az alsóvárosi lakosokat!
c) Beszélgettünk egy Ádám utónevű felsővárosi férfival. Ki lehetett ő?
d) Listázzuk ki azokat, akik valamilyen „út”-on laknak!
e) Mikor érkezett az első vendég 2020. 10. 07-én?
f) Mikor távozott az utolsó vendég 2020. 10. 07-én?
g) Hány vendég volt 2020. 10. 07-én?
h) Melyik menüt választotta Varga Tamara, amikor első ízben ott ebédelt?

Logikai műveletek a lekérdezésekben

Az előző leckében láttuk, hogyan tudjuk a szűrésnél megfogalmazott feltételeket a lekérdezésekben használni. Most már nem foglalkozunk a szűréssel, a problémákat lekérdezéssel oldjuk meg a *városok* adatbázisban.

10. példa: Relációs jelek a lekérdezésben

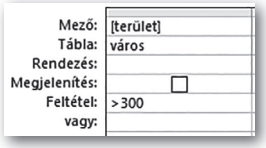
Az új ismeretek alapján könnyen tudunk lekérdezést készíteni az alábbi kérdésekhez. Az elkészült lekérdezéseket mentsük is el a zárójelben megadott néven!

- Mely városok területe nagyobb 300 km^2-nél? *(terület)*
- Mely városok lélekszáma haladja meg a 100 ezer főt? *(lélekszám)*

Íme a megoldások SQL-parancsként és lekérdezőrácson:

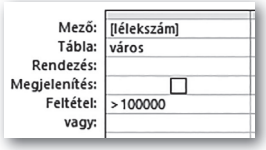
terület:

```
SELECT *
FROM város
WHERE terület > 300;
```



lélekszám:

```
SELECT *
FROM város
WHERE lélekszám > 100000;
```



Az ÉS művelet

11. példa: Az ÉS művelet többféle megfogalmazásban

Nézzük meg azt a feladatot, amelyben az előző két feltétel egyaránt szerepel!

Melyek a 300 km^2-nél nagyobb területű, ugyanakkor 100 ezer főnél népesebb városok? *(összetett1)*

A korábban megoldott szűrési feladatok szövegében a feltételeket az **és,** valamint a **vagy** szavak kapcsolták. Ezeket a szavakat, pontosabban az SQL-ben használt megfelelőjüket, az **AND** és **OR** logikai műveleteket szolgai módon be lehetett írni a lekérdezésbe. A feladat szövegében ezúttal ezt a szerepet az **ugyanakkor** szó tölti be, ilyen logikai művelet azonban nincs.

Meg kell szoknunk, hogy a mindennapokban megfogalmazott problémákhoz nem lehet tükörfordítással feltételt megadnunk. A bennük szereplő hétköznapi szófordulatok, megfogalmazások tartalmát meg kell értenünk, majd átfogalmaznunk úgy, hogy az elemi feltételeket az **ÉS, VAGY, NEM** szavak kapcsolják össze. Ezek az SQL-nyelvben használt logikai műveletek is, ezért az átfogalmazással általában a lekérdezésben használt feltételt is megalkottuk. Jelen esetben az **ugyanakkor** szó egyidejűséget fejez ki, az első feltétel a másodikkal egyszerre teljesül, ezért az **és** szóra cserélhetjük. Ha gondolkodunk egy keveset, akkor a feladatot másképp is meg tudjuk fogalmazni anélkül, hogy jelentésén változtatnánk.

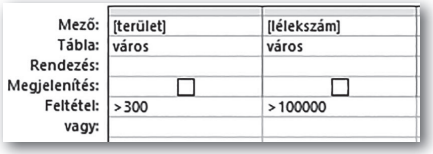
- Melyek a 300 km^2-nél nagyobb területű, valamint 100 ezer főnél népesebb városok?
- Melyek azok a városok, amelyek 300 km^2-nél nagyobbak, ráadásul 100 ezer főnél népesebbek is?

Keressünk még néhány, ezzel tartalmilag egyező megfogalmazást!

Gépeljük be az SQL-parancsot, és nézzük meg a *Tervező nézetet,* majd értelmezzük a látottakat!

összetett1:

```
SELECT *
FROM város
WHERE terület > 300
      AND lélekszám > 100000;
```



Lényegében annyi történt, hogy a *terület* és *népesség* lekérdezésekhez tartozó rácsokat összeillesztettük. A lekérdezőrács első sorában azok a mezőnevek szerepelnek, amelyekre feltételt fogalmazunk meg, alattuk pedig ugyanabban a sorban a relációs jelek és az értékek.

Ha két vagy több feltételt az ÉS művelettel kapcsolunk össze, akkor a lekérdezőrácson ugyanabba a sorba kerülnek.

Vizsgáljuk meg a *terület, lélekszám* és az *összetett1* lekérdezések eredményhalmazát! A *terület* és a *lélekszám* listák közös elemei alkotják az *összetett1* listáját, tehát ez a másik kettő metszete. Az adatbázis-kezelés során visszagondolhatunk a matematikából tanult halmazokra. Az adattábla teljes tartalmát tekintjük az alaphalmaznak, a feltételek pedig ennek részhalmazait határozzák meg. Egy-egy összetett problémához tartozó eredményhalmazt úgy is elképzelhetünk, mint a feltételek adta részhalmazokon végzett halmazműveletek eredményét. Nem kell minden esetben ilyen „hivatalosan” fogalmaznunk, de érdemes tudnunk, hogy a matematikából szerzett tudásunkat itt is alkalmazhatjuk.

A VAGY művelet

12. példa: A VAGY művelet alkalmazása

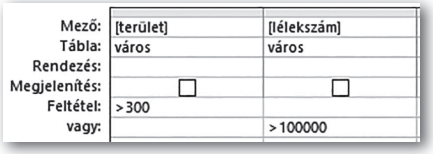
Nézzünk egy másik összetett feladatot!

Jelenítsük meg közös listában azokat a városokat, amelyek 300 km^2-nél nagyobb területűek, és azokat is, amelyek 100 ezer főnél nagyobb népességűek! *(összetett2)*

A fenti mondatban olvasott *és* szó ne tévesszen meg senkit, úgy kell tekintenünk, hogy vannak a 300 km^2-nél nagyobb területű városok, és ettől függetlenül vannak azok, amelyek 100 ezer főnél nagyobb népességűek. Nem elég csak az egyik halmaz városait megjeleníteni, hozzá kell venni még azokat is, amelyek a másik halmazban vannak. Nyilván azoknak nem kell duplán megjelenniük, amelyek mindkettőben szerepelnek. Ez pontosan megfelel a két halmaz uniójának. Az unió azon elemeket tartalmazza, amelyek legalább az egyik halmazban szerepelnek, tehát az egyik vagy a másik feltétel igaz rájuk, vagyis a két feltételt a **VAGY** művelettel, az SQL-parancsban pedig az **OR**-ral kapcsoljuk össze.

összetett2:

```
SELECT *
FROM város
WHERE terület > 300
      OR lélekszám > 100000;
```



A lekérdezés a lekérdezőrácson alig valamiben tér el az előzőtől, de figyelmesen megnézve a képet, észrevehetjük, hogy a feltételt megadó két érték nem ugyanabban a sorban van.

Ha két feltétel között **VAGY** kapcsolat van, akkor azokat a lekérdezőrácson külön sorba kell írnunk.

13. példa: A megfogalmazás egyértelműsége

Listázzuk ki azokat a városokat, amelyek területe meghaladja a 100 km²-t, és megyei jogúak, vagy lélekszámuk nagyobb 40 ezer főnél! *(összetett3)*

Egyesek úgy érezhetik, hogy a mondat megfogalmazása nem egyértelmű. Nem tudják eldönteni, hogy a területre vonatkozó feltétel csak a megyei jogú városokra vonatkozik-e, vagy a 40 ezer főnél népesebbekre is. Ez inkább a hétköznapi nyelven, szóban megfogalmazott feladatoknál fordulhat elő.

Az alábbi ábrák a lekérdezőrácson mutatják be a kétféle értelmezést. Határozzuk meg, hogy melyik melyik értelmezéshez tartozik!

Mező:	terület	megyeijogú	lélekszám
Tábla:	város	város	város
Rendezés:			
Megjelenítés:	☑	☑	☑
Feltétel:	>100	Igaz	
vagy:			>40000

► A feladat egyik értelmezése

Mező:	terület	megyeijogú	lélekszám
Tábla:	város	város	város
Rendezés:			
Megjelenítés:	☑	☑	☑
Feltétel:	>100	Igaz	
vagy:	>100		>40000

► A feladat másik értelmezése

Azt kell figyelnünk, hogy mit írtunk egy sorba, és mit külön. Ha ezeknek a lekérdezéseknek megnézzük az SQL-nézetbeli alakját, láthatjuk, hogy az Access híven követi azt, hogy amit egy sorba írunk, azok közé **AND** kerül, a külön sorokat pedig az **OR** határolja. (A parancssoros leírásból a felesleges zárójelek jelentős részét eltávolítottuk.)

```
összetett3a:
SELECT terület, megyeijogú, lélekszám
FROM város
WHERE (terület>100) AND (megyeijogú=True* OR lélekszám>40000);
összetett3b:
SELECT terület, megyeijogú, lélekszám
FROM város
WHERE (terület>100 AND megyeijogú=True)
  OR (terület>100 AND lélekszám>40000);
```

*A lekérdezőrácson feltétlenül ki kell írnunk az Igaz értéket, SQL-nézetben a True elhagyható, mivel a mező értéke önmagában is az igaz/hamis logikai érték.

Az első esetben a területre vonatkozó feltétel csak a megyei jogú városokra vonatkozik. A másik esetben a megyei jogúakra és a 40 ezer főnél népesebb városokra is igaz, hogy nagyobbak 100 km²-nél.

Ha megnézzük az összetett lekérdezések futtatásának eredményét, akkor azt látjuk, hogy az első kettőnél minden mező értéke megjelent, az utóbbiaknál pedig csak azok, amelyekre feltételt fogalmaztunk meg. A lekérdezőrácsra felvehetjük azokat a mezőket is,

amelyekre nem vonatkozik feltétel. A mezők láthatóságát a lekérdezőrács *Megjelenítés* sorában található jelölőnégyzetekkel szabályozhatjuk.

14. példa: A „között” alkalmazása

Adjuk meg az 50 és 100 ezer fő közötti népességű megyei jogú városokat! *(között)*

Vannak, akik az 50 ezer és 100 ezer közötti lélekszámú városok közé sorolják azt is, amelyik éppen 50 ezer lakosú, míg mások nem. Meg kell állapodnunk, hogy a továbbiakban a „között” kifejezést hallva az egyenlőséget megengedjük az intervallum mindkét végén.

Mező:	név	lélekszám	lélekszám	megyeijogú
Tábla:	város	város	város	város
Rendezés:				
Megjelenítés:	☑	☑	☐	☑
Feltétel:		>=50000	<=100000	Igaz
vagy:				

▶ Összeállított lekérdezés (A *lélekszám* mező kétszer is szerepel. Mivel ugyanazt a tartalmat jelenítik meg, ezért az egyiknél a megjelenítést kikapcsoltuk.)

Mező:	név	lélekszám	megyeijogú
Tábla:	város	város	város
Rendezés:			
Megjelenítés:	☑	☑	☑
Feltétel:		>=50000 And <=100000	Igaz
vagy:			

▶ Átalakított lekérdezés

Megoldásunk az első ábrán látható. Ha elmentjük a megoldást, majd *Tervező nézetben* megnyitjuk, nem ugyanezt kapjuk vissza, hanem a második ábrán látható tartalmat. Érdemes megjegyezni, hogy a lélekszámra vonatkozó feltételt – amely egy intervallumot ad meg – ebben az egyszerűbb formában is bejegyezhetjük, tömörebbé, áttekinthetőbbé téve megoldásunkat. Ha valaki biztosan nem akar hibázni, használhatja az ábrán látható megoldást is.

Mező:	név	lélekszám	megyeijogú
Tábla:	város	város	város
Rendezés:			
Megjelenítés:	☑	☑	☑
Feltétel:		Between 50000 And 100000	Igaz
vagy:			

▶ Az intervallumra szűrés új megoldása

Tagadás

15. példa: A tagadás használata

Határozzuk meg azokat a megyei jogú városokat, amelyek népessége nem éri el az 50 ezer főt, vagy meghaladja a 100 ezret! *(szélsőséges)*

- A három képen látható megoldás egyenértékű. Aki megjegyezte, hogy a VAGY kapcsolatot a feltételek külön sorba írásával fejezzük ki, az első képen látható formát használja. Nem szabad megfeledkezni azonban arról, hogy a megyei jogú tulajdonságot mindkét sorban fel kell tüntetni, hiszen az mindkét létszámkorlát esetén érvényes!

Mező:	név	lélekszám	megyeijogú
Tábla:	város	város	város
Rendezés:			
Megjelenítés:	☑	☑	☑
Feltétel:		<50000	Igaz
vagy:		>100000	Igaz

Mező:	név	lélekszám	megyeijogú
Tábla:	város	város	város
Rendezés:			
Megjelenítés:	☑	☑	☑
Feltétel:		<50000 Or >100000	Igaz
vagy:			

Mező:	név	lélekszám	megyeijogú
Tábla:	város	város	város
Rendezés:			
Megjelenítés:	☑	☑	☑
Feltétel:		Not (>=50000 And <=100000)	Igaz
vagy:			

- A második megoldáshoz az előző feladat második megoldása adja az ötletet. Ez nemcsak tömörebb leírást tesz lehetővé, hanem kevesebb hibalehetőséget rejt magában.
- A harmadik megoldást azok alkalmazzák, akik felismerik, hogy a lélekszámra vonatkozó feltétel tagadása az előző feladatban megfogalmazott kívánalomnak.

Jegyezzük meg: egy feltételt legegyszerűbben mindig az elé írt **NOT** szóval tagadhatunk.

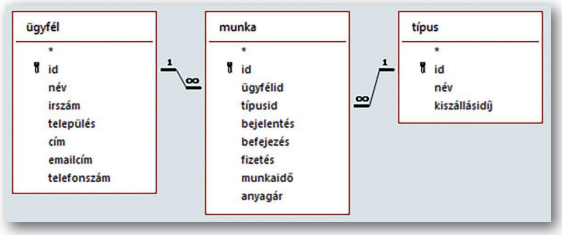
Feladatok

Oldjuk meg a következő feladatokat a *városok* adatbázist használva, lekérdezőrács segítségével!

a) Adjuk meg a 100 km^2-nél kisebb területű megyei jogú városokat! *(f5a)*
b) Listázzuk ki a 150 és 250 km^2 közötti területű nem megyei jogú városokat! *(f5b)*
c) Határozzuk meg, mely városok azok, amelyek területe 150 és 250 km^2 között van, és népessége 50 ezer főnél kevesebb! *(f5c)*
d) Adjuk meg azokat a városokat, amelyek területe nem 150 és 250 km^2 közötti, de népességük 50 ezer főnél nagyobb! *(f5d)*
e) Soroljuk fel azokat a városokat, amelyek 300 km^2-nél nagyobb területűek, vagy népességük meghaladja a 100 ezer főt! *(f5e)*
f) Soroljuk fel azokat a megyei jogú városokat, amelyek 300 km^2-nél nagyobb területűek, vagy népességük meghaladja a 100 ezer főt! *(f5f)*

A szöveg mezőtípus

Az eddig készített lekérdezéseinkben a számokra vonatkozó feltételek mellett adott szöveges értékkel egyezést vizsgáltunk. Egy-egy probléma megoldásához ennél többre lehet szükségünk. Használnunk kell a kisebb/nagyobb relációt szövegek esetén is, szükséges a szövegrész előállítása, és gyakori igény a név szerinti sorrend felállítása.



► A *javítás* adatbázis szerkezete

Ebben a leckében a példák a *javítás* adatbázishoz tartoznak. Az alábbi kép az adatbázis szerkezetét adja meg. (A szerkezet bemutatására az Access program *Adatbáziseszközök* > *Kapcsolatok* pontján látható ábrát használjuk.)

16. példa: A szöveggel való egyezés

Az adatbázisban az ügyfélként feltüntetett személyek a lakásukban előforduló különböző típusú hibákat javíttatják meg. A javítás folyamatához tartozó adatokat a *munka* nevű kapcsolótáblában rögzítették, így a hibabejelentést, a munka befejezésének és a számla kifizetésének dátumát, a ráfordított munkaórák számát és a felhasznált anyagok árát.

A tanultak alapján az alábbi feladatokat meg tudjuk oldani lekérdezés segítségével:

- Adjuk meg a felsővárosi ügyfelek nevét! *(felsőváros)*
- Határozzuk meg azokat az ügyfeleket, akik nem Felsővárosban laknak! *(nemfelsőváros)*

Az első feladat megoldása nagyon könnyű volt, csak a település nevét kellett beírnunk. A begépelt szöveget a program automatikusan kiegészítette az ábrán is látható idézőjelekkel. *SQL-nézetben* feltétlen be kell gépelnünk az idézőjeleket, a lekérdezőrácson csak akkor szükséges, ha a beírt szöveg egyezik egy mező nevével, különben az Access mezőnévként értelmezi.

A második feladatot sokan a **NOT** logikai művelet segítségével oldják meg, mert a szöveg alapján ez tűnik magától értetődőnek. Átváltva *Adatlap* nézetbe, majd vissza *Tervező nézetbe,* azt tapasztaljuk, hogy a program átalakította a feltételt, a **NOT** helyére a <>, azaz a nem egyenlő jel került.

Mező:	név	település
Tábla:	ügyfél	ügyfél
Rendezés:		
Megjelenítés:	☑	☑
Feltétel:		"Felsőváros"
vagy:		

► A felsővárosiakat listázó lekérdezés

Mező:	név	település
Tábla:	ügyfél	ügyfél
Rendezés:		
Megjelenítés:	☑	☑
Feltétel:		Not "Felsőváros"
vagy:		

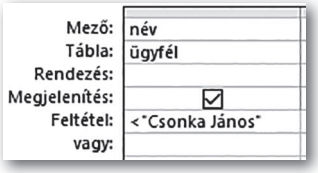
► A felsővárosi lakhelyet tagadó lekérdezés

Mező:	név	település
Tábla:	ügyfél	ügyfél
Rendezés:		
Megjelenítés:	☑	☑
Feltétel:		<>"Felsőváros"
vagy:		

► Felsővárossal nem egyező településnév

17. példa: A szöveg típussal kapcsolatos feltételek

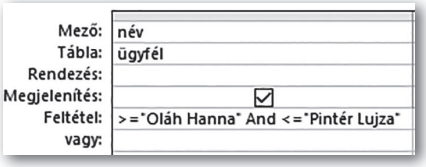
- A javítást végző cég ünnepi üdvözletet küld az ügyfeleinek. A leveleket ábécérendben küldik ki. Kiknek küldték el az üdvözletet Csonka János előtt? *(CsonkaJános)*
- Az ügyintézők között felosztották az ügyfeleket, ábécérendben. Istvánnál Oláh Hanna az első, Pintér Lujza pedig az utolsó. Kik tartoznak Istvánhoz? *(OláhPintér)*
- Évára bízták a számlák ellenőrzését az Oláh és a Pintér közötti vezetéknevű ügyfelek esetén. Adjuk meg az Évához tartozók névsorát! *(OP)*

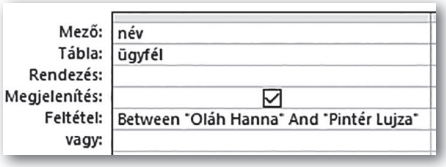


A kisebb és nagyobb reláció értelmezett a szöveg típus esetén is: az a név „kisebb”, amelyik az ábécében előrébb áll. Így könnyű dolgunk van, csak a megfelelő relációs jelet kell a név elé írnunk. Nézzük meg, kik előzik meg Csonka Jánost! Meglepő lehet, hogy Czirok Péter is közöttük van. A programozásban azt láttuk, hogy stringeknél az első eltérő karakter alapján dől el, melyik a kisebb, de a helyesírási szabályzat szerint az első eltérő betű betűrendben elfoglalt helye dönt. Jelen esetben a C betű előrébb van a Cs betűnél, ezért a Czirok is a Csonkánál.

Feladat

Olvassuk el, hogy mik a betűrendbe állítás szabályai (https://helyesiras.mta.hu/helyesiras/default/akh12#F2_4)!

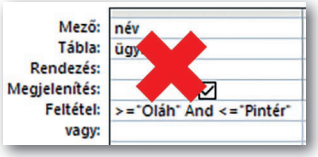




Mivel Oláh Hanna és Pintér Lujza is az eredményhalmaz része, ezért a ≥ és a ≤ relációs jeleket kell használnunk. A nevek rendezett sorozatában felfoghatjuk úgy is, mint egy zárt intervallumot, ezért a BETWEEN operátort is használhatjuk a feltétel megadásakor.

Az *OláhPintér* lekérdezés SQL-nyelven:

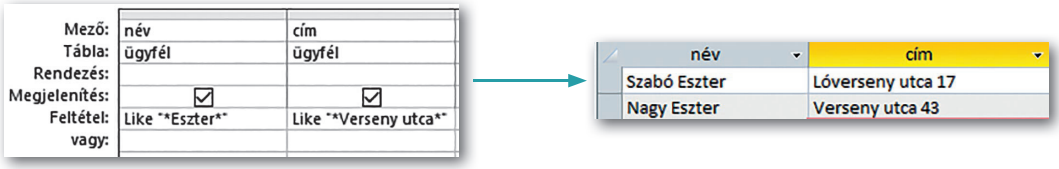
```
SELECT név
FROM ügyfél
WHERE név BETWEEN ”Oláh Anna” AND ”Pintér Lujza”;
```



Az Oláh és Pintér vezetéknevűekre vonatkozó feladat látszólag egyezik az előzővel – bár kevésbé pontosan adja meg a feltételt. Ha a megoldást a teljes nevet tartalmazóhoz hasonlóan állítjuk elő, akkor láthatóan nem az elvárt eredményhalmazt kapjuk meg, hiszen Pintér vezetéknevű egyáltalán nem szerepel a listában. Ha az ábrán látható feltételt nem a teljes névre, hanem csupán a vezetéknévre fogalmaznánk meg, a helyes eredmény állna elő. Ez a feladat így – bár megoldható az adatbázis-kezelő programmal – túlmutat a tankönyv tárgyalási szintjén.

18. példa: A mintaillesztés

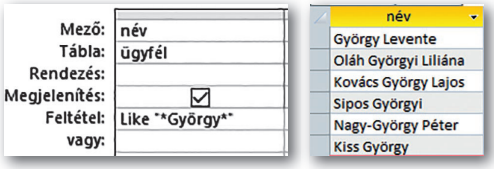
- A telefonos ügyfélszolgálatos helyét néhány percre egy másik munkatárs vette át. A legutolsó hívásból csak annyira emlékezett, hogy Eszter volt a Verseny utcából. Ki lehetett ő? *(Eszter)*
- A cég e-mailben köszönti az ügyfeleit a névnapjukon. Listázzuk ki a György utónevű ügyfelek nevét, e-mail-címét! *(György)*
- A cégnél rendszeresen ellenőrzik az ügyfelek adatait. Most azok vannak soron, akiknek a vezetékneve C-vel kezdődik. Adjuk meg az érintett ügyfelek nevét! *(Cbetű)*

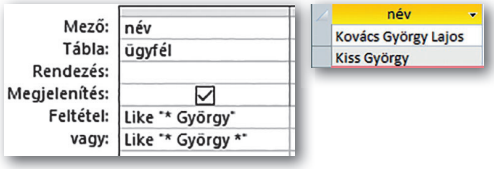


Ha belepillantunk az *ügyfél* táblába, azt látjuk, hogy az utónevet a vezetéknévvel együtt tároljuk a név mezőben, és az utca megnevezése sem önmagában áll. Ha lehetséges, hogy a keresett karaktersorozat csak része az adott mező értékének, akkor a nem ismert részek helyére a * karaktert írjuk. Ha úgy gondoljuk, hogy a keresett karaktersor bárhol állhat a mezőn belül, akkor elé és mögé is beírjuk a * karaktert. Ha beírjuk a feltételhez a helyettesítő karaktert tartalmazó szöveget, az Access program a feltételt automatikusan kiegészíti a **LIKE** operátorral. A fenti ábrát megnézve a lekérdezés nem az elvárt eredményt adja, mivel a „Verseny utca” a „Lóverseny utcára” is illeszkedik. Ezzel egyúttal azt is megtapasztaltuk, hogy a program szövegre vonatkozó feltételek vizsgálatakor nem különbözteti meg a kis- és nagybetűket.

Ha tudjuk, hogy a karaktersorozat a mező (kifejezés) elején helyezkedik el, akkor csak mögé kell írnunk a * karaktert. Az eredményhalmazt megvizsgálva azt hihetjük, hogy immár helyes megoldást adtunk. A lista tartalmilag megegyezik az elvárttal, de a következő feladat rávilágít arra, hogy átgondoltabbnak kell lennünk.

Az előző feladatban olvasott figyelmeztetés miatt érezzük, hogy a megoldás a György utónevűek megkeresésére a `Like "*György*"` feltétellel nem lesz alkalmas, mégis tanulságos ezzel kezdeni a próbálkozást. Számítottunk ugyan felesleges értékekre, de ilyen sokra nem. A sok felesleges sort többféle hiba okozza. Hiba, hogy megkaptuk azokat a neveket, amelyeknél a György vezetéknévként a név mező elején található. Szintén hiba, hogy megkaptuk a Györgyi nevűeket is.





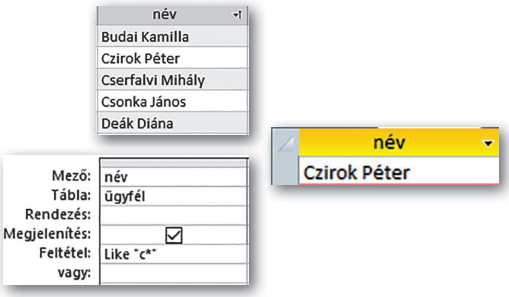
Nézzük, hogy a kívánalomnak megfelelő rekordokban miképpen fordul elő a György karaktersorozat! Lehet a végén, mint utolsó utónév. Ekkor szóköz áll előtte, és azelőtt bármi szerepelhet. Lehet, hogy egy közbülső utónév, ekkor viszont előtte és utána is szóköz áll, és az előtt és azt követően is bármely szöveg szerepelhet.

A *György* lekérdezés SQL-nyelven:

```
SELECT név
FROM ügyfél
WHERE név LIKE "* György" OR név LIKE "* György *";
```

Visszatérve az első feladatra, kijelenthetjük, hogy a helyes kimenet ellenére bizony hibás megoldást adtunk másodjára is. A szöveg nem szólt arról, hogy az Eszter vezeték- vagy utónév-e. Ha mindkettőt meg kell engednünk, akkor az ábrákon látható megoldás a helyes.

Ha a névre vonatkozó egyes eseteket soronként jegyezzük be, akkor vigyázzunk arra, hogy az utcára vonatkozó feltétel minden sorban szerepeljen! Az egysoros megfogalmazás nemcsak tömörebb, de kevesebb hibalehetőséget is rejt.



A három feladatból a C-vel kezdődő vezetéknevűek kigyűjtése a legegyszerűbb. Nem a megoldás nehézsége, hanem az eredményhalmazból levont tanulság a fontos. Az Access programban nem karakterekből, hanem betűkből álló szöveges kifejezésekkel dolgozunk. Ez a megoldási elvet nem befolyásolja, de ha meglepő eredményt kapunk, gondoljunk arra, hogy a magyar nyelvben két-, sőt háromjegyű betűk is vannak.

A **LIKE** operátor segítségével egy szöveg típusú kifejezést – általában egy mező értékét – összevethetünk egy mintával, amelyben a * tetszőleges számú, a ? pedig csak egyetlen karakter helyettesítésére szolgál. (Az Access programban a karakter szó helyett helyesebb betűt használnunk. Az SQL-nyelv szabványnak megfelelő változatai a * helyett a % jelet, a ? helyett pedig az _ jelet használják.)

Feladatok

Oldjuk meg a feladatokat a *javítás* adatbázist használva, lekérdezőrács segítségével!

a) Adjuk meg azokat, akiknek az irányítószáma 3000 és 4000 között van! *(f6a)*
b) Listázzuk ki azokat, akiknek .hu-ra végződik az e-mail-címük! *(f6b)*
c) Soroljuk fel azokat az ügyfeleket, akiknek az utóneve *B*-vel kezdődik! *(f6c)*
d) Határozzuk meg azokat az ügyfeleket, akiknek a nevében van *a* és *e* karakter is! *(f6d)*
e) Adjuk meg azokat az ügyfeleket, akiknek a nevében nincs *a* és *e* karakter! *(f6e)*
f) Határozzuk meg azokat, akiknek az e-mail-címében van pont karakter a @ karakter előtt! *(f6f)*
g) Adjuk meg azokat, akiknek legalább két utónevük van! *(f6g)*
h) Listázzuk ki azokat, akiknek csak egy utónevük van! *(f6h)*

A dátum és az idő típus

Igen egyszerű dolgunk van, ha az adattábla szöveg típusú mezőjét töltjük fel értékekkel. Ha a mező beállításait nem módosítottuk, pontosan a begépelt szöveg jelenik meg mindenkinek a képernyőjén. Egész számoknál ugyanezt tapasztaljuk. Valós számnál már az operációs rendszer területi beállításaitól függően tizedesjelként pontot vagy vesszőt mutat, és így is kell begépelnünk. A dátum megjelenítése és bevitele sokféleképpen történhet. Az Access programban akkor tudjuk megszokott módon kezelni a dátumokat, ha az operációs rendszerben az éééé.hh.nn. formátum van beállítva. Ellenőrizzük ezt a beállítást, és szükség esetén módosítsuk! (Ellenőrizhető a *Vezérlőpult* > *Régió* > *További beállítások* > *Dátum* > *Dátumformátumok* > *Rövid dátum* pontjában. Magyar nyelvű Windows 10 esetén az éééé. HH.nn. a helyes beállítás.)

Tanultuk, hogy a táblázatkezelő a dátumokat egész számként, a napon belüli időt pedig törtként tárolja. Ez az adatbázis-kezelőben is így történik. Táblázatkezelőben 1900.01.01. a kezdő dátum, amely az 1 értéket kapja. Adatbázis-kezelőben beírhatjuk a mohácsi vész vagy Szent István koronázásának dátumát is, jól fogja megjeleníteni. A 0 értéknek az 1899.12.31. felel meg, a korábbi dátumokat negatív értékként kezeli. A kezelés csak technikai értelemben helyes, tudnunk kell, hogy a naptárreformokat nem veszi figyelembe.

19. példa: A dátummal kapcsolatos feltételekről

A következő feladatokat az *étkezés* adatbázis *ebéd* tábláját használva oldjuk meg!

- Adjuk meg a 2020. 10. 06-án fogyasztott ebédek adatait! *(október6)*
- Határozzuk meg, mely napokon érkezett vendég 11:40 előtt! *(korán)*
- Listázzuk ki, mely napokon nem volt 11:40 előtt érkező vendég! *(nemvolt)*

Mező:	azonosító	személyazonosító	menüazonosító	dátum	desszert	érkezés	távozás
Tábla:	ebéd	ebéd	ebéd	ebéd	ebéd	ebéd	ebéd
Rendezés:							
Megjelenítés:	☑	☑	☑	☑	☑	☑	☑
Feltétel:				#2020.10.06.#			
vagy:							

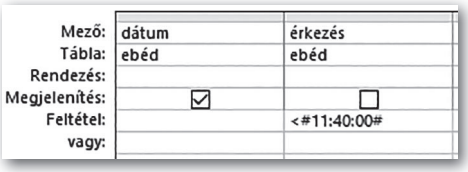
► Az *október6* lekérdezés a mezők egyenkénti feltüntetésével

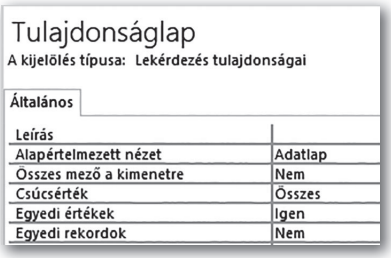
Mező:	ebéd.*	dátum
Tábla:	ebéd	ebéd
Rendezés:		
Megjelenítés:	☑	☐
Feltétel:		#2020.10.06.#
vagy:		

► Az *október6* lekérdezés a * karakter felhasználásával

Az *október6* lekérdezésnél az összes mező megjelenítését kétféleképpen is megoldhatjuk. Kiválaszthatjuk egyenként az összes mezőt, de használhatjuk a * karaktert is – jelen esetben az *ebéd*.* formában. Egy-egy feladat megoldása során értelmezés kérdése, hogy az azonosító mező része-e az ebéd jellemzőinek. Technikai szempontból mindenképpen, ezért nem követünk el hibát a megjelenítésével. Feltételként a 2020.10.06. értéket kell begépelnünk. Az Access program az adott mezőből továbblépve automatikusan kiegészíti a # karakterekből álló párral, jelezve, hogy dátum típusú adatról van szó. (A legtöbb adatbázis-kezelő esetén a dátumokat is idézőjelek közé írjuk.) Ha nem ez történik, akkor valószínűleg a dá-

tumot gépeltük el, vagy az adott mező nem dátum típusú. Ha az egyszerűbb megoldást választjuk, akkor a feltételt megadó *dátum* oszlop megjelenítését kapcsoljuk ki, ugyanis ez a * karakter miatt mindenképpen látható lesz.



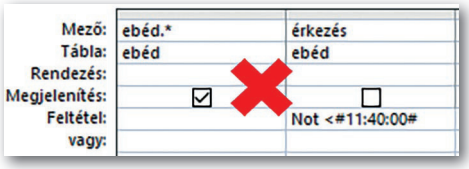


A *korán* lekérdezésnél a feltételhez beírt időpontot (például 11:40) a program kiegészíti óra:perc:másodperc alakra, és a # karakterrel jelzi, hogy idő típusú adatról van szó. A feladat nem kéri, hogy az érkezés időpontja is jelenjen meg, ezért kapcsoljuk ki a megjelenítését! Az eredményhalmazt megvizsgálva azt látjuk, hogy egyes napok többször is megjelennek. A többszörös megjelenés jelen esetben számunka semmilyen hasznos információt nem hordoz, ezért kapcsoljuk ki! Erre a *Lekérdezéstervezés* > *Megjelenítés, elrejtés* > *Tulajdonságlap* > *Egyedi értékek* ponton van lehetőség, az *Igen* érték beállításával.

A *korán* lekérdezés SQL-ben:

```
SELECT DISTINCT dátum
FROM ebéd
WHERE érkezés<#11:40:0#;
```

A DISTINCT kulcsszó az eredményhalmaz ismétlődéseit kiszűri, minden sor pontosan egyszer jelenik meg.

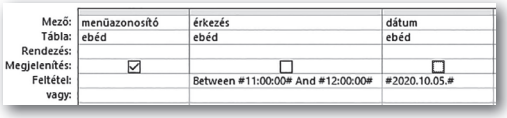


Sokan azt gondolják, hogy a *nemvolt* lekérdezést megkapjuk, ha a *korán* lekérdezés feltételét tagadjuk. A lekérdezést végrehajtva azt látjuk, hogy sok olyan nap megjelenik, amely az előző listának is része volt. Vegyük észre, hogy ezzel a feltétellel azokat a napokat adjuk meg, ahol volt 11:40-kor vagy azt követően érkező vendég. Ezeken a napokon érkezhettek néhányan korábban is. Az adatbázis-kezelés sokak számára azt jelenti, hogy megadjuk a rögzített értékek egy részhalmazát, ebben a kérdésben viszont éppen olyan értékeket várunk, amelyeket nem rögzítettek. Ez a feladat nehéz, jelenlegi tudásunkkal még nem tudjuk megoldani, később is csak arra tudunk válaszolni, hogy azon napok közül, amelyeken volt vendég, mely napokon nem volt 11:40 előtt érkező.

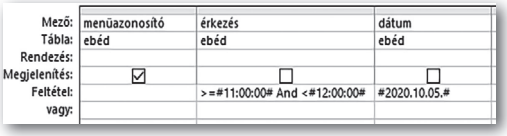
20. példa: Az időintervallumok kezelése

- Listázzuk ki, hogy milyen menüket szolgáltak ki 11:00 és 12:00 között 2020. 10. 05-én! Mindegyik menü betűjele pontosan egyszer jelenjen meg! *(11és12között)*
- Adjuk meg, hogy milyen menüket szolgáltak ki a nap 12. órájában 2020. 10. 05-én! Mindegyik menü betűjele pontosan egyszer jelenjen meg! *(óra12)*

A *11és12között* lekérdezést elkészítve azt látjuk, hogy az adott intervallumon belül több A, B és C menüt adtak ki. Ezek többszöri felsorolása felesleges, a *korán* lekérdezésnél látott módon, az egyedi értékek beállításával az ismétlődő megjelenést elkerülhetjük.

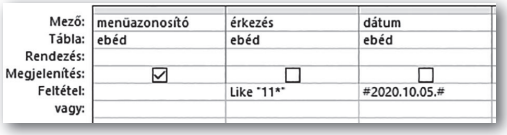


Mező:	menüazonosító	érkezés	dátum
Tábla:	ebéd	ebéd	ebéd
Rendezés:			
Megjelenítés:	☑	☐	☐
Feltétel:		Between #11:00:00# And #12:00:00#	#2020.10.05.#
vagy:			



Mező:	menüazonosító	érkezés	dátum
Tábla:	ebéd	ebéd	ebéd
Rendezés:			
Megjelenítés:	☑	☐	☐
Feltétel:		>=#11:00:00# And <#12:00:00#	#2020.10.05.#
vagy:			

Az *óra12* lekérdezésnél érdemes tisztázni, hogy a nap első órája 0 óra 0 perc 0 másodperckor kezdődik, a 01:00:00 pedig már a második óra kezdete. Tehát a nap 12. órájához a 11:00:00 hozzátartozik, de a 12:00:00 már nem. A képen látható megoldásban 11 óránál szerepel az egyenlőségjel, 12 óránál már nem!



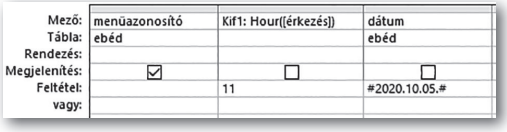
Mező:	menüazonosító	érkezés	dátum
Tábla:	ebéd	ebéd	ebéd
Rendezés:			
Megjelenítés:	☑	☐	☐
Feltétel:		Like "11*"	#2020.10.05.#
vagy:			

Gondoljuk végig, hogy mely időpontok tartoznak ide! Mindegyik a 11-es számmal kezdődik. Ha az előző anyagrésznél figyeltünk, akkor megkísérelhetjük a mintaillesztést az *érkezés* mezőre, azaz egy idő típusú adatra alkalmazni. 11 a kezdete a mezőnek, mögötte bármi szerepelhet. Az Access program a LIKE operátor használatakor automatikusan szöveggé konvertálja a dátum és idő típusú adatokat, ezért tudjuk használni a mintaillesztést. Időpontoknál jól gondoljuk meg a használatát, mert – ha a második óra lett volna a kérdés – az 1* feltétel illeszkedett volna az 1:15:00-s és a 11:15:00-s értékre is. Az ilyen problémákat elkerülhetjük, ha az elválasztó karaktereket is beírjuk a keresőkifejezésbe.

Dátum- és időfüggvények

A mintaillesztéshez vezető ötletet más módon is felhasználhatjuk. A 11 nem pusztán két karakter, amely a mezőben előre került, hanem az időpont óráját jelenti. Valójában az *érkezés* mező órájára kellene feltételül szabni, hogy az értéke 11 legyen. Táblázatkezelés kapcsán tanultunk olyan függvényeket, amelyek az időpont óráját, percét, másodpercét – `Óra()`, `Percek()`, `Mperc()` – határozzák meg. Ilyen szerepű függvények az adatbázis-kezelő programban is léteznek: **Hour()**, **Minute()**, **Second()**. E függvények paramétere dátum/idő típusú érték.

Az *óra12* feladatnak az ábrán látható lekérdezés is helyes megoldása. Ha végiggondoljuk, hogy milyen problémákat lehet az időpontokra vonatkozóan megfogalmazni, érezzük, hogy a relációs jelek segítségével megadott feltételek a legáltalánosabban használhatók. Természetesen a másik két megoldási módot is érdemes ismerni.



Mező:	menüazonosító	Kif1: Hour([érkezés])	dátum
Tábla:	ebéd		ebéd
Rendezés:			
Megjelenítés:	☑	☐	☐
Feltétel:		11	#2020.10.05.#
vagy:			

Az *Óra12* lekérdezés SQL-nyelvű megfelelője:

```
SELECT DISTINCT menüazonosító
FROM ebéd
WHERE Hour(érkezés)=11 AND dátum=#10/5/2020#;
```

Az SQL-nyelvű leírásban a dátum megadására csodálkozhatunk rá. A # határolók mellett a hónap, nap, év sorrend és a / karakter mint határoló jelenthet újdonságot.

Ahogy az időadatok óráját, percét, másodpercét meg tudjuk határozni megfelelő függvények segítségével, úgy a dátumokat is felbonthatjuk év, hónap, nap számértékekre. Ehhez a táblázatkezelőben használt Év(), Hónap(), Nap() függvényekkel egyezően működő **Year()**, **Month()**, **Day()** függvényekre van szükségünk.

Feladatok

Oldjuk meg a következő feladatokat a *javítás* adatbázist használva, lekérdezőrács segítségével!

a) Listázzuk ki a 2017. augusztus 15. és szeptember 15. között bejelentett munkák adatait! *(f7a)*
b) Határozzuk meg a 2017 decemberében kezdődő tél hónapjaiban befejezett munkák adatait! A megoldáshoz használjuk a BETWEEN operátort! *(f7b)*
c) Kérdezzük le a 2017 novemberében bejelentett munkákat mintaillesztés segítségével! *(f7c)*
d) Kérdezzük le a 2017 júniusában kifizetett munkákat dátumfüggvények segítségével! *(f7d)*
e) Adjuk meg a korábbi évek valamelyikében, a feladatmegoldás napján kifizetett munkákat! A megoldáshoz használjunk függvényt! *(f7d)*

Rendezés

Az eddig megismert lekérdezésekben elsősorban a szűrésre koncentráltunk, az eredmény megjelenési sorrendje közömbös volt számunkra. A valóságban gyakran nem elégszünk meg ennyivel, igenis tudni akarjuk, hogy az elérhető buszok közül melyik ér oda a leghamarabb, melyik szállodai szoba a legolcsóbb, vagy ki nyerte a versenyt. Ahhoz, hogy a kérdésekre válaszolni tudjunk, a rendezéssel kell megismerkednünk.

21. példa: Az egy- és a többkulcsú rendezés

Az alábbi feladatok az *étkezés* adatbázisra vonatkoznak.

- Listázzuk ki a vendégek adatait vezetéknevük szerint rendezve! *(vezetéknév)*
- Adjuk meg a vendégek adatait vezetéknév, azon belül utónév szerint rendezve! *(név)*
- Listázzuk ki a Nagyfaluban lakók nevét, utcáját és házszámát utca, azon belül házszám szerint rendezve! *(cím)*
- Adjuk meg az összes vendég vezeték- és utónevét, települését – ebben a sorrendben feltüntetve az adatokat – település, azon belül név szerint rendezetten! *(összetett)*

Mező:	vendég.*	vezetéknév	keresztnév
Tábla:	vendég	vendég	vendég
Rendezés:		Növekvő	Növekvő
Megjelenítés:	☑	☐	☐
Feltétel:			
vagy:			

54	Balog	Noel	Felsőváros	Verseny utca	57
32	Balogh	Adél	Alsóváros	Pellérdi út	21
146	Balogh	Eszter	Felsőváros	Görgey Artúr u	4
35	Balogh	Hunor	Nagyfalu	Stiglicfogdosó	32
83	Balogh	Levente	Nagyfalu	Kút utca	3
53	Barna	Panna	Alsóváros	Mátyás Flórián	23

▶ Teljes név szerint rendezve

Korábban tanultuk, hogyan lehet a *vezetéknév* feladatot lekérdezés nélkül megoldani, de mindenképpen ez az első lépés a rendezés tanulása során. Beállítása rendkívül egyszerű, a lekérdezőrács *Rendezés* sorában kell meghatározni a rendezés irányát.

Mező:	vendég.*	vezetéknév
Tábla:	vendég	vendég
Rendezés:		Növekvő
Megjelenítés:	☑	☐
Feltétel:		
vagy:		

51	Balog	Lilien	Kisfalu	József utca	17
146	Balogh	Eszter	Felsőváros	Görgey Artúr u	4
32	Balogh	Adél	Alsóváros	Pellérdi út	21
83	Balogh	Levente	Nagyfalu	Kút utca	3
35	Balogh	Hunor	Nagyfalu	Stiglicfogdosó	32
53	Barna	Panna	Alsóváros	Mátyás Flórián	23

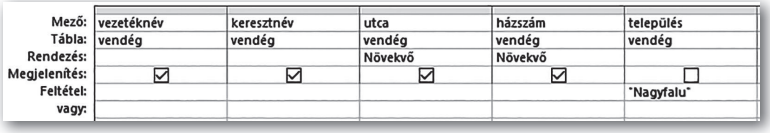
▶ Vezetéknév szerint rendezve

A *vezetéknév* lekérdezés SQL-nyelvű megfogalmazása:

```
SELECT *
FROM vendég
ORDER BY vezetéknév;
```

(Az **ORDER BY** vezetéknév **ASC**; forma lenne a teljes, de az ASC [ascending] elhagyható, mert a növekvő rendezés az alapértelmezett.)

A lekérdezés futtatását követően nézzük meg az eredményhalmaz rekordjainak sorrendjét! Azt találjuk, hogy a Balogh vezetéknevűek között a sorrend gyakorlatilag véletlenszerű. A *név* lekérdezés hétköznapi szóhasználattal azt kéri, hogy a *vezetéknév,* azon belül pedig az *utónév* döntsön a sorrendben. Ezt **többkulcsú rendezésnek** hívjuk. Ha két rekord első kulcs szerinti értéke egyezik, sorrendjüket a második kulcshoz tartozó érték fogja megszabni. A lekérdezőrácson a rendezésre kijelöltek közül az első kulcs oszlopának meg kell előznie a második kulcs oszlopát. A *név* lekérdezést futtatva azt látjuk, hogy a Balogh vezetéknevűek az utónevük szerinti sorrendben szerepelnek.



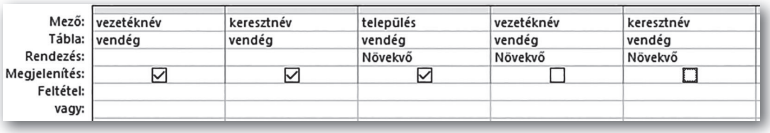
Mező:	vezetéknév	keresztnév	utca	házszám	település
Tábla:	vendég	vendég	vendég	vendég	vendég
Rendezés:			Növekvő	Növekvő	
Megjelenítés:	☑	☑	☑	☑	☐
Feltétel:					"Nagyfalu"
vagy:					

Horváth	Boglárka	Lahti utca	44
Lengyel	Zsófia	Mécses utca	46
Nagy	Eszter	Mécses utca	8
Tóth	Letícia	Örs utca	31

A *név* lekérdezés SQL-nyelvű megfogalmazása:

```
SELECT *
FROM vendég
ORDER BY vezetéknév, utónév;
```

A *cím* lekérdezés elkészítése semmivel sem nehezebb, mint a *név* lekérdezésé. A lényeget az eredményhalmaz sorrendjének vizsgálatánál látjuk. A Mécses utcában a 46-os házszám alatt lakó megelőzi a 8-as szám alattit. A sorrend nem az elvárásaink szerint alakul. Nézzük meg, hogy mi az oka! A tábla *Tervező nézetében* látjuk a szerkezetet és azt, hogy a házszám szöveg típusú mező. Idézzük fel a tanultakat! Két szöveg típusú érték sorrendjét az első eltérő karakter sorrendje dönti el. Ezért a szövegként tárolt 46 előrébb kerül a listában, mint a 8. Ennek a tulajdonságnak azért választottuk a szöveg típust, mert a *házszám* mezőben tároltuk a 2/B értéket is. Ahhoz, hogy a házszámokat szám típusú értékként tárolhassuk – a helyes rendezés érdekében –, az egyéb karaktereket le kell választani róla. Erre szolgálnak egy-egy űrlapon az *épület,* illetve *lépcsőház* mezők.



Mező:	vezetéknév	keresztnév	település	vezetéknév	keresztnév
Tábla:	vendég	vendég	vendég	vendég	vendég
Rendezés:			Növekvő	Növekvő	Növekvő
Megjelenítés:	☑	☑	☑	☐	☐
Feltétel:					
vagy:					

Ahogy a *cím* lekérdezésben, úgy sok más esetben sem tulajdonítunk jelentőséget a mezők megjelenési sorrendjének, az *összetett* lekérdezés viszont kifejezetten előírja azt. Az oszlopok megjelenítési sorrendje – *vezetéknév, utónév, település* – és a rendezési előírás sorrendje – *település, vezetéknév, utónév* – ellentmondóak. Ilyen esetben vegyük fel a rendezési előírásnak megfelelő sorrendben az érintett mezőket, állítsuk be a rendezést, de ne jelenítsük meg azokat, amelyek már előbb szerepeltek a megjelenítési sorrendben. A fenti ábrán egy lehetséges megoldás látszik. Ha valaki SQL-nyelven oldja meg a feladatot, nem kell ilyen „trükkhöz" folyamodnia:

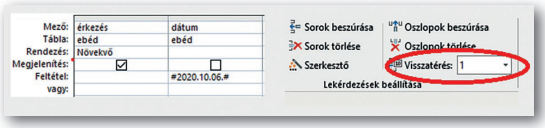
```
SELECT vezetéknév, utónév, település
FROM vendég
ORDER BY település, vezetéknév, utónév;
```

SQL-nyelven egyéb egyszerűsítési lehetőség is van, a rendezést megadhatjuk a megjelenített oszlop sorszámával is:

```
SELECT vezetéknév, utónév, település
FROM vendég
ORDER BY 3, 1, 2;
```

22. példa: A sorrendileg első rekordok megjelenítése

- Hánykor érkezett a legelső vendég 2020. 10. 06-án? *(legelső)*
- Hánykor távozott az utolsó vendég 2020. 10. 06-án? *(legutolsó)*
- Melyik nap kértek először desszertet? *(desszert)*
- Melyik volt az első öt olyan nap, amikor desszertet kértek? *(desszert5)*



A legelső kérdésre az előzőleg megoldott feladatok alapján könnyen tudnánk válaszolni. Növekvő sorrendbe állítjuk az érkezési időket, és leolvassuk az első értéket az *érkezések* oszlopából. A sok érték megjelenése azonban zavaró, hiszen nem az összes, hanem csak a legelső válaszol a kérdésre. A lekérdezőrácsban kialakított feltétellel együtt megjelölhetjük, hogy az előállt lista elejéről hány elemet jelenítsen meg. Ezúttal az 1 értéket írjuk a lekérdezés visszatérési tulajdonságához. Ez egyenértékű a lekérdezés *Tulajdonságlap* > *Csúcsérték* beállítással.

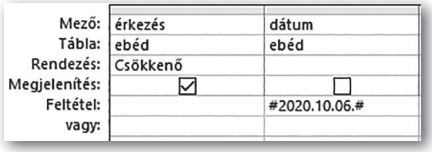
A *legelső* lekérdezés SQL-nyelven:

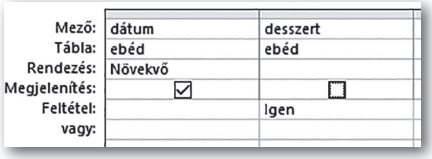
```
SELECT TOP 1 érkezés
FROM ebéd
WHERE dátum =#10/6/2020# ORDER BY érkezés;
```

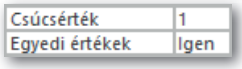
A legutolsó távozó vendég esetén megint a sorba rendezés a kulcs, csak a lista utolsó elemét kell választanunk. Mivel az adatbázis-kezelő programok csak az első néhány elem megjelenítésére képesek, ezért meg kell fordítanunk a rendezés irányát a megoldáshoz. A lekérdezéshez tartozó visszatérési értékek számát az ábrán nem jelöltük.

A *legutolsó* lekérdezés SQL-nyelven:

```
SELECT TOP 1 érkezés
FROM ebéd
WHERE dátum =#10/6/2020# ORDER BY érkezés DESC;
```







A *desszert* lekérdezés futtatása nem várt eredményt hoz. Hiába állítottuk a visszatérési értékek számát 1-re, vég nélkül sorolja a 2020.10.01. értéket. A desszertrendelés egyáltalán nem különleges, ezért már a legelső napon is sok vendég megtette. Az Access program nem tud választani az egyező értékek közül, így az összeset megjeleníti.

Ez nem jelenti azt, hogy a feladatot ne tudnánk megoldani jelenlegi tudásunkkal. Korábban megtanultuk kiszűrni az ismétlődéseket. A *Tulajdonságlapon* az egyedi értékek beállítása eltávolította az eredményhalmaz ismétlődő rekordjait, ha most ezt alkalmazzuk, az elvárt értéket, egyetlen dátumot kapunk. Elgondolkodhatunk azon, hogy vajon először az ismétlődéseket távolítja-e el, azután korlátozza a megjelenített sorok számát, vagy fordítva. Ezt a *desszert5* lekérdezést elkészítve magunk is megválaszolhatjuk.

A **TOP n** módosító segítségével a megjelenített rekordok számát az első *n* darabra korlátozzuk. A program nem rangsorolja a kulcskifejezés szerint egyező értékeket, így az *n*. rekorddal egyezőek mind megjelennek. (Az SQL-szabvány nem ismeri a TOP módosítót, abban a hasonló szerepű LIMIT található meg, amit a lekérdezés végén kell megadni.)

Feladatok

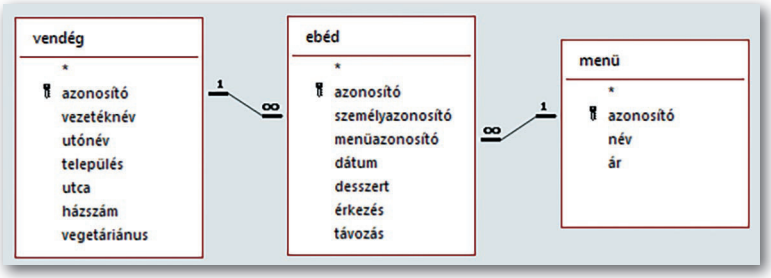
Oldjuk meg a feladatokat a *városok* adatbázist használva, lekérdezőrács segítségével!

a) Listázzuk ki a megyei jogú városokat betűrendben! *(f8a)*
b) Adjuk meg azon városok nevét és lélekszámát, amelyek nevében a város vagy a falu karaktersorozat szerepel! *(f8b)*
c) Határozzuk meg a három legnagyobb területű várost! *(f8c)*
d) Melyik a legkisebb népességű város azon nem megyei jogú városok közül, amelyek területe a 200 km^2-t meghaladja? *(f8d)*

Adatok több táblából

Az eddigi feladatok megoldásához elegendő volt egyetlen tábla adataival dolgozni. Sokszor olyan problémákat oldunk meg, amelyekhez egyszerre több tábla adatait kell használnunk.

Amikor a feladatmegoldás során használt táblákat létrehoztuk, meghatároztuk a közöttük lévő kapcsolatokat is. Ha felhasználjuk ezeket a kapcsolatokat, akkor az egymással összefüggésben álló táblákat tekinthetjük úgy is, mint egyetlen – olykor gigantikus méretű – **virtuális táblát.** Ez a tábla fizikailag nem létezik, de egy, több táblát használó lekérdezéssel el tudjuk készíteni, ahogy az alábbi ábrán látható is.



► Az *étkezés* adatbázis szerkezete

Mező:	vezetéknév	utónév	település	utca	házszám	vegetáriánus	dátum	desszert	érkezés	távozás	név	ár
Tábla:	vendég	vendég	vendég	vendég	vendég	vendég	ebéd	ebéd	ebéd	ebéd	menü	menü
Rendezés:	Növekvő	Növekvő										
Megjelenítés:	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
Feltétel:												
vagy:												

vezetéknév	utónév	település	utca	házszám	vegetáriánu	dátum	desszert	érkezés	távozás	név	ár
Bakos	Gergely	Felsőváros	Rakéta utca	27	☐	2020.10.12.	☐	12:10:05	12:24:04	húsimádó	1410
Bakos	Gergely	Felsőváros	Rakéta utca	27	☐	2020.10.22.	☐	11:42:14	11:56:02	tésztás	1290
Bakos	Gergely	Felsőváros	Rakéta utca	27	☐	2020.10.20.	☐	12:34:21	12:47:12	tésztás	1290
Bakos	Olivér	Nagyfalu	Rezeda dűlő	39	☐	2020.10.15.	☐	13:28:34	13:41:31	diétás	1380
Bakos	Olivér	Nagyfalu	Rezeda dűlő	39	☐	2020.10.02.	☑	13:21:01	13:34:42	húsimádó	1410
Bakos	Olivér	Nagyfalu	Rezeda dűlő	39	☐	2020.10.20.	☐	13:35:09	13:48:06	diétás	1380

► Az *étkezés* adatbázis összetartozó adatai

Az ábrán látható, összetartozó adatokat előállító lekérdezés SQL-nyelven:

```
SELECT vendég.vezetéknév, vendég.utónév, vendég.település, vendég.utca,
    vendég.házszám, vendég.vegetáriánus,
    ebéd.dátum, ebéd.desszert, ebéd.érkezés, ebéd.távozás,
    menü.név, menü.ár
FROM vendég, ebéd, menü
WHERE vendég.azonosító=ebéd.vendégazonosító
    AND ebéd.menüazonosító=menü.azonosító
ORDER BY vendég.vezetéknév, vendég.utónév
```

(A tábla nevét csak akkor kötelező a mezőnév elé írni, ha a felhasznált táblákban az adott mezőnév több helyen is előfordul.)

Ha visszagondolunk a fejezet elején tanultakra, akkor eszünkbe juthat, hogy ez a táblázat a normalizálás előtti állapotot mutatja. Nagyon nagy a redundancia, hiszen a nevek, címek sokszor szerepelnek, és számtalan sorban olvasható a menü neve és ára is. Innen emeltük ki a vendégek és a menük adatait egy-egy külön táblába. Vegyük észre, hogy a táblának

pontosan annyi sora van, mint az *ebéd* táblának, és oszlopainak száma az egyes táblák oszlopszámának összege.

A lekérdezések elkészítéséhez nemcsak táblákat, hanem a lekérdezések eredményeként előálló virtuális táblákat is használhatunk. A későbbiekben ezt – a táblákat egyesítő – lekérdezést használjuk majd fel. Ezt megtehetjük, ha meg akarjuk tudni, hogy Bakos Gergely mely napokon milyen menüt választott, azonban felesleges, ha Bakos Olivér címére vagyunk kíváncsiak. Utóbbi kapcsán miért dolgoznánk több mint 4000 rekorddal, ha ehhez elegendő kevesebb mint 200 vendég adatait átnézni?

Tehát, ha egy táblát felhasználva nem tudunk válaszolni egy kérdésre, akkor használjuk fel pontosan csak azokat, amelyek a megoldáshoz szükségesek, de ne többet. Érdemes néhány példán keresztül megismerni, hogy mire kell figyelni a táblák kiválasztásánál.

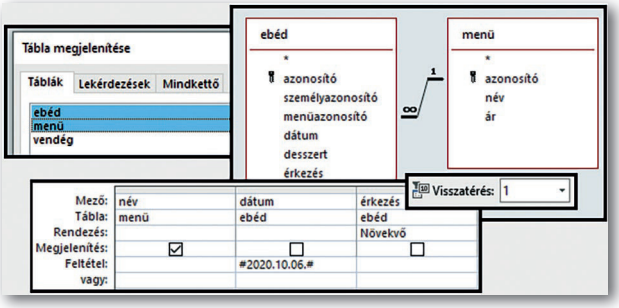
23. példa: **Több tábla használata**

- Határozzuk meg, hogy Bakos Gergely melyik nap milyen menüt választott! A lista legyen dátum szerint rendezett! *(Bakos)*
- Adjuk meg, hogy 2020. október 6-án melyik menüt szolgálták fel az elsőnek érkező vendégnek! *(első)*
- Listázzuk ki azokat a vegetáriánusokat, akik nem csak vegetáriánus menüt rendeltek! *(vegetáriánus)*

A *Bakos* lekérdezésben a négy szükséges mező három táblában található, ezért mindhármat használnunk kell. Amikor a lekérdezéskészítés első lépéseként kiválasztjuk a táblákat, a beemeltek között megjeleníti azokat a kapcsolatokat, amelyeket az adatbázis létrehozása során meghatároztunk. A feltétel beírását és a rendezés megadását követően készen is van a lekérdezés.

Mező:	dátum	név	vezetéknév	utónév
Tábla:	ebéd	menü	vendég	vendég
Rendezés:	Növekvő			
Megjelenítés:	☑	☑	☐	☐
Feltétel:			"Bakos"	"Gergely"
vagy:				

Az *első* lekérdezésben a menü *nevét* kell megjeleníteni, a *dátum* mezőre állítunk be feltételt, a sorrendet pedig az *érkezés* mező szabja meg. Az *ebéd* és a *menü* táblára van szükségünk, amelyek között kapcsolat van. Az eredményhalmaznak csak az első elemét kell megjeleníteni. Az ábra négy része ezeket a beállításokat mind mutatja. Ennek eredménye egyetlen szó, mely szerint diétás menüt szolgáltak fel.



A *vegetáriánus* lekérdezésben a neveket kell megjelenítenünk a *vendég* táblából, a feltétel pedig a *vendég* és a *menü* tábla mezőire vonatkozik. Látszólag csak két táblát igényel, a *vendég* és a *menü* táblákat. Ha ezt a kettőt vesszük fel, és a lekérdezőrács tartalmát az ábra szerint alakítjuk, akkor az eredményhalmaz 60 rekordot tartalmaz. Több mint 4000 ebédelésből nem olyan sok ez. Jelenítsük meg a menük nevét is, és rendezzük betűrendbe a neveket! Furcsának tűnik, hogy minden vegetáriánus vendéghez mindhárom nem vegetáriánus menü hozzátartozik. Mivel két olyan táblát használtunk, melyek között nem volt kapcsolat, ezért a 20 vegetáriánushoz mind a 3 lehetséges menüt hozzárendelte, ezért kaptunk eredményül 60 sort.

Mező:	vezetéknév	utónév	vegetáriánus	név
Tábla:	vendég	vendég	vendég	menü
Rendezés:				
Megjelenítés:	☑	☑	☐	☐
Feltétel:			Igaz	<>"vegetáriánus"
vagy:				

Vegyük fel az *ebéd* táblát is! A lekérdezőrácson nem módosítva 62 rekordot kapunk, de egyetlen esetben sem szerepel a húsimádó menü. Itt pontosan azok az ebédek szerepelnek, ahol a vegetáriánusok nem vegetáriánus menüt rendeltek.

Ahhoz, hogy pontosan a kérdésre válaszoljunk, állítsuk be az egyedi értékek megjelenítését a *Tulajdonságlapon!* Láthatjuk, hogy 20 személyből 19 fő élt ilyen választással.

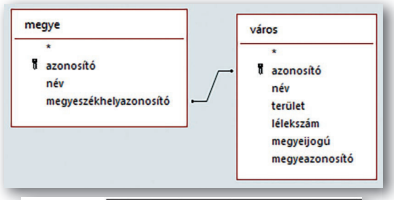
Ha egy adatbázis több táblából áll, a lekérdezést úgy építjük fel, hogy kiválasztjuk azokat a mezőket, amelyeket meg kell jelenítenünk, és azokat, amelyekre feltétel vonatkozik. A lekérdezésbe beemeljük azokat a táblákat, amelyekben szerepel kiválasztott mező. Ha az így kiválasztott táblák között nincs kapcsolat, akkor kiválasztjuk azokat a táblákat is, amelyek segítségével a kapcsolat megteremthető.

Egyedi kapcsolatok

Az eddigi lekérdezésekben azokat a táblák közötti kapcsolatokat használtuk, amelyeket az adatbázis készítése során határoztunk meg. Előfordulhatnak olyan esetek is, amikor nem erre vagy nem csak erre van szükség. A *városok* adatbázis kapcsán erre látunk példát.

24. példa: Előre nem megadott kapcsolatok használata

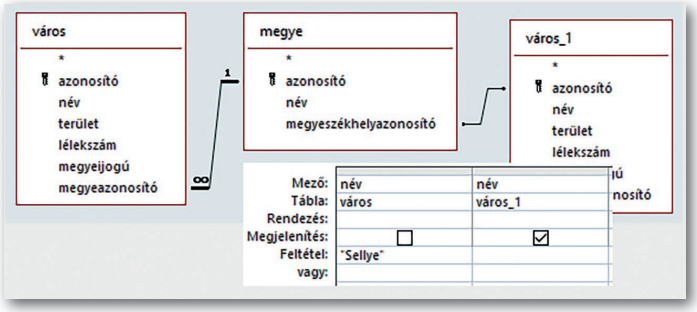
- Melyik város Heves megye székhelye? *(Heves)*
- Mi a székhelye annak a megyének, ahol Sellye található? *(Sellye)*



Mező:	név	név
Tábla:	megye	város
Rendezés:		
Megjelenítés:	☐	☑
Feltétel:	"Heves"	
vagy:		

A *Heves* lekérdezésben egy város nevére vagyunk kíváncsiak, és a megye nevét ismerjük. Természetesen mindkét táblát használnunk kell. A kérdést nem tudjuk megválaszolni az adatbázisban adott táblák közötti kapcsolattal, mert az a városhoz rendel megyét, nekünk pedig a megyéhez kell város. A *megye* táblában a *megyeszékhelyazonosító* tulajdonság idegenkulcs-szerepű, most az általa megadott kapcsolatra van szükségünk. A táblák választásakor automatikusan bekerülő kapcsolatot törölnünk kell, és az ábrán láthatót létrehozni.

Azt meg tudjuk mondani, hogy Sellye melyik megyében van. Ha ismerjük Sellye megyéjét, akkor – az előző lekérdezés alapján – képesek vagyunk meghatározni, hogy mi a megyeszékhelye. Aki gondolja, el is készítheti a megyét meghatározó lekérdezést, majd azt az ábra szerint a *város* tábla újbóli felvételével kiegészítve meg tudja alkotni a *Sellye* lekérdezést. Ebben a lekérdezésben sem csupán az adatbázisban megtalálható eredeti kapcsolatot használtuk, hanem egy újat is.



A *Sellye* lekérdezés SQL-nyelvű megoldása:

```
SELECT város_1.név
FROM város, megye, város AS város_1
WHERE város.megyeazonosító=megye.azonosító
    AND megye.megyeszékhelyazonosító=város_1.azonosító
    AND város.név="Sellye";
```

Feladatok

Oldjuk meg a feladatokat a *javítás* adatbázist használva, lekérdezőrács segítségével!

a) Határozzuk meg, mikor dolgoztak a Pellérdi úton! *(f9a)*
b) Listázzuk ki, hogy ki, mikor és milyen típusú problémát javíttatott 2019-ben! *(f9b)*
c) Határozzuk meg, ki volt az első ügyfele a cégnek! *(f9c)*
d) Adjuk meg, milyen típusú munkához tartozott a legnagyobb értékű javítás! *(f9b)*

Számított értékek

Emlékezzünk vissza az adatbázis-tervezés bemutatása során használt, diákokról szóló típusmondatra és táblázatra! Ott arra jutottunk, hogy nem tároljuk el a diákok korát, hiszen az folyamatosan változik, helyette a születési évet (dátumot) rögzítjük, mert abból mindig ki tudjuk számítani a kort. Szinte alig van olyan adatbázis, amelyben nem tudunk számítást igénylő problémát kitűzni.

25. példa: Matematikai műveletek használata

A *városok* adatbázisban a népsűrűséget, az *étkezés* adatbázisban az adott napon az ebédért fizetett összeget, a *javítás* adatbázisban a munkadíjat kell kiszámítanunk. A számítások során sokszor csak a matematikából ismert műveleteket kell használnunk, de előfordulhat, hogy az itt tanult függvényeket is alkalmaznunk kell.

- Határozzuk meg a *városok* adatbázisban a Tolna megyei városok népsűrűségét! A városokat népsűrűség szerint csökkenően jelenítsük meg! *(Tolna)*
- Az *étkezés* adatbázis adatait használva listázzuk ki azok nevét, akik kevesebb mint 12 perc alatt végeztek az ebéddel! Írassuk ki az érkezési és távozási időt, majd értelmezzük az eredményt! *(kisebb12)*
- A *javítás* adatbázis adatai alapján határozzuk meg a 2018-as év számláinak végösszegét! Egy munkaóra díja 3500 Ft. A befizető neve, címe és a végösszeg jelenjen meg! *(fizetendő)*
- A *javítás* adatbázis adatai alapján adjuk meg, kik voltak azok, akik – bár befejezték náluk a munkát – csak a befejezést követő évben fizettek! Az ügyfél azonosítója és neve jelenjen meg! *(jövőre)*

város.név	Kif1
Szekszárd	330,234732031575
Dombóvár	229,599898063201
Nagymányok	206,647940074906
Bonyhád	177,945661214306
Tolna	155,522724074856
Paks	120,865784008307
Simontornya	113,479160508424
Bátaszék	95,5167531854648
Dunaföldvár	75,7763417698797
Tamási	70,6208128628852
Gyönk	48,6224088165836

A *Tolna* lekérdezés elkészítésénél a népsűrűség meghatározása jelenti az újdonságot. A népsűrűség nem más, mint két mező értékének hányadosa: *lélekszám/terület*. Ezt a kifejezést begépelhetjük, vagy a lekérdezőrácson a mező sorában a *helyi menü* > *Szerkesztés*… pontját választva ki is kattintgathatjuk. Utóbbi megoldásnak a hasonló bonyolultságú képleteknél csak annyi előnye van, hogy a gépelési hibáktól megvéd bennünket.

A lekérdezést futtatva az ábrán látható eredményt kapjuk. A képet alaposan megnézve néhány szokatlan dolog tűnik fel. A népsűrűség oszlopának fejében a *Kif1* szöveg olvasható, a népsűrűség értékénél indokolatlanul sok tizedesjegy van, valamint a városok oszlopa felett a *város.név* látható. Azt szoktuk meg, hogy a lekérdezések eredményét megnézve az eredménytáblázat első sorában a mezőneveket olvashatjuk. A népsűrűség nem egyetlen mezőt jelöl, hanem kettőnek a hányadosát. Ilyen esetben az Access program az oszlopot automatikusan elnevezi. A népsűrűség oszlopa a *Kif1* nevet kapta. Az első oszlopban a mező neve azért egészült ki a tábla nevével, mert a felhasznált táblák közül legalább

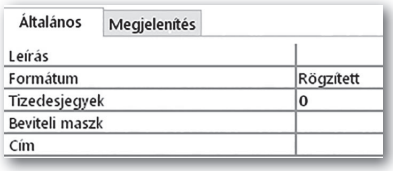
Mező:	város: név	népsűrűség: [lélekszám]/[terület]	név
Tábla:	város		megye
Rendezés:		Csökkenő	
Megjelenítés:	☑	☑	☐
Feltétel:			"Tolna"
vagy:			

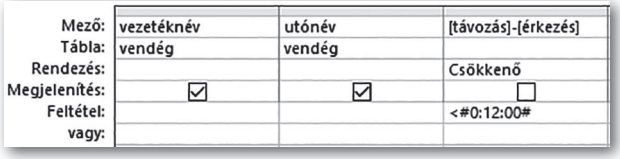
kettő tartalmazott *név* nevű mezőt, így a tábla neve tette egyértelművé, hogy melyikkel dolgozunk. Ha visszaváltunk a lekérdezőrácsra, akkor a *Kif1* szöveget ott is megtaláljuk: *Kif1*: [*lélekszám*]/[*terület*]. Egy oszlopot mi magunk is elnevezhetünk, ha a mező sorban a mezőnév vagy kifejezés elé beírjuk a kívánt feliratot, attól kettősponttal elválasztva.

A tizedesjegyek számának csökkentése is egyszerű feladat. A lekérdezőrácson ki kell választani az érintett mezőt, majd megjeleníteni a *Tulajdonságlapját*. A *Formátum* és a *Tizedesjegyek* sorában kell az ábra által megjelenített beállításokat megtennünk.

A *Tolna* lekérdezés SQL-nyelvű megoldása (a kerekítést az SQL-lekérdezésben kerekítő függvénnyel lehet előállítani, míg az Accessben készített megoldás formázással jelenítette meg):

```
SELECT város.név AS város, Round(lélekszám/terület, 0) AS népsűrűség
FROM város, megye
WHERE város.megyeazonosító=megye.azonosító
    AND megye.név="Tolna"
ORDER BY 2 DESC;
```





A *kisebb12* lekérdezésben két időértéket tartalmazó mező különbségét kell meghatározni. Gondoljunk vissza arra, amikor a dátum és idő típusú adatokkal ismerkedtünk! Valójában számértékekről van szó, csak a megjelenítési formátum az idő. A dátumot az egészrész adja, a napon belüli időpontot pedig a törtrész. Ellenőrzésképpen jelenítsük meg a különbség értékét is! Nem meglepő, hogy nemnegatív, egynél kisebb valós számokat látunk. Az előző lekérdezésnél látható módszerrel megpróbálhatjuk az időformátumra alakítást, de erre itt nincs mód a *Tulajdonságlap* segítségével. Jelenítsük meg az érkezés és a távozás időpontját is! Az a meglepő, hogy az érintett rekordok mindegyikében pontosan 12 percig tartó ebédeket látunk, tehát egyik sem teljesíti a feltételt. Nos, ez igaz, az ok a számítógép számábrázolásában keresendő. Mivel a 12 perc pontosan 120-ad része az 1 nap időtartamnak, ezért szorozzuk meg 120-szal a különbség értékét! Azt fogjuk látni, hogy nem 1 egészet kapunk eredményül, hanem annál egy nagyon kicsivel kisebb számot, az eltérést a 15. tizedesjegyben találjuk.

Természetesen nem várjuk el, hogy mindenki ilyen szintű alapossággal vizsgáljon meg minden feladatot, de fontos, hogy bemutassuk, bizony az elvileg helyes megoldással is kaphatunk hibás eredményt. Az idő óra, perc, másodperc részből áll. Ezek egészek. Ha ezeket használjuk, nem jelenik meg a valós számok ábrázolásából adódó probléma. Nézzük azt a feltételt, amely erre épül:

```
Hour([távozás])*3600+Minute([távozás])*60+Second([távozás]) -
    (Hour([érkezés])*3600+Minute([érkezés])*60+Second([érkezés])) < 720
```

Mező:	név	település	cím	Év: Year([befejezés])	fizetendő: [anyagár]+[kiszállásidíj]+3500*[munkaidő]
Tábla:	ügyfél	ügyfél	ügyfél		
Rendezés:					
Megjelenítés:	☑	☑	☑	☐	☑
Feltétel:				2018	

Ha ezt a feltételt használjuk, a helyes eredményt kapjuk, mert az eredménylista üres lesz.

A *fizetendő* lekérdezés azt mutatja, hogy a számításhoz szükséges mezőket lehet, hogy több táblából kell összegyűjtenünk. A *fizetendő* oszlopában szereplő számok pénzösszegek. Állítsuk be a mező megjelenítését tartalmának megfelelően, azaz jelenjen meg a pénznem is! Ezt a *Tolna* lekérdezés mintájára könnyen megoldhatjuk.

Mező:	id	név	Kif1: Year([befejezés])
Tábla:	ügyfél	ügyfél	
Rendezés:			
Megjelenítés:	☑	☑	☐
Feltétel:			<Year([fizetés])
vagy:			

Mező:	id	név	eltérés: Year([fizetés])-Year([befejezés])
Tábla:	ügyfél	ügyfél	
Rendezés:			
Megjelenítés:	☑	☑	☐
Feltétel:			1
vagy:			

A *jövőre* feladathoz a fenti ábrákon látható két lekérdezést készítettük. Futtatásuk eredménye ugyanaz a lista. Vajon mindkettő helyes?

Az első esetben a feltétel szerint a fizetés éve későbbi, mint a befejezés idejének éve. A második megoldás azt mondja, hogy a fizetés évének és a befejezés évének különbsége pontosan 1. Vegyük észre, hogy az első megoldás megadhat olyan ügyfelet is, aki elvégeztette a munkát 2019-ben, de már 2021 volt, amikor fizetett, tehát a lista elvileg tartalmazhat olyan adatsort is, amely nem felel meg a leírásnak. A helyes megoldáshoz egy számítás vezetett. Ettől eltérő megoldást is adhatunk, de azt nem tudjuk elkerülni, hogy számítást végezzünk.

Feladatok

Oldjuk meg a következő feladatokat lekérdezőrács segítségével!

a) A *javítás* adatbázis adatai alapján határozzuk meg, hogy a 2018 márciusában és áprilisában bejelentett munkák esetén a bejelentést követően hány nappal fejezték be a munkát! A befizető neve, címe és az eltelt napok száma jelenjen meg! *(f10a)*

b) A *javítás* adatbázis adatai alapján adjuk meg, mely munkáknál fordult elő, hogy az anyagköltség a duplája volt a kiszállási díjnak! *(f10b)*

c) Az *étkezés* adatbázis adatait használva listázzuk ki, hogy 2020. 10. 08-án kik ebédeltek a leghosszabb ideig! A három legtöbb időt eltöltő nevét adjuk meg! *(f10c)*

d) Az *étkezés* adatbázis adataiból dolgozva adjuk meg, hogy az egyes vendégek menynyit fizettek 2020. 10. 08-án! A desszert aznap 500 Ft-ba került. Ügyeljünk arra, hogy az érték helyes legyen a desszertet fogyasztó és nem fogyasztó vendégek esetén is! A vendég neve és a számla összege jelenjen meg! *(f10d)*

Aggregáló függvények

Az eddig szerzett tudásunkkal már sok problémát meg tudunk oldani, de több, a maga természetességében megfogalmazott feladattal nem boldogulunk. Azt bármely nap esetén meg tudjuk válaszolni, hogy mikor érkezett az első vendég, és mikor távozott a legutolsó, de ehhez két lekérdezésre van szükségünk. Hát még akkor milyen nehéz helyzetben lennénk, ha ezt az étterem minden munkanapjára meg kellene adnunk! Ez utóbbihoz először még a dátumokat is ki kellene gyűjtenünk, ráadásul egy újabb dátum bejegyzését nem követné a megoldás.

26. példa: Az aggregáló függvények használata

Az étkezés adatbázist használva oldjuk meg az alábbi feladatokat!

- Adjuk meg egyetlen lekérdezéssel, hogy 2020. 10. 08-án mikor érkezett az első vendég, és mikor távozott a legutolsó! *(tólig)*
- Listázzuk ki, hogy az egyes napokon mikor érkezett az első vendég, és mikor távozott a legutolsó! *(tólignaponta)*
- Határozzuk meg, hogy az egyes napokon hányan kértek desszertet! *(desszertnaponta)*
- Adjuk meg, hogy mely napokon kért legalább 100 fő desszertet! *(desszert100)*

Mező:	Nyitás: érkezés	Zárás: távozás	dátum
Tábla:	ebéd	ebéd	ebéd
Összesítés:	Min	Max	Where
Rendezés:			
Megjelenítés:	☑	☑	☐
Feltétel:			#2020.10.08.#
vagy:			

A *tólig* feladatot két lekérdezéssel meg tudjuk oldani, egyikben az *érkezés* szerint növekvően, a másikban a *távozás* szerint csökkenően rendezve a rekordokat. Akár azt is meg tudjuk adni, hogy melyik vendég volt az. E két érték megadását egyszerűbben is megoldhatjuk. Az egyik oszlop legkisebb és egy másik legnagyobb értékét határozzuk meg az adott dátumra szűrés mellett! A lekérdezőrácson jelenítsük meg az érkezés, távozás és dátum mezőket, majd kattintsunk a *Lekérdezéstervezés* > *Megjelenítés/elrejtés* > *Összesítés* gombra! Ennek hatására egy új, *Összesítés* nevű sor jelenik meg a rácson. Mivel az érkezés legkisebb értékére van szükségünk, az összesítés sorának lenyíló listájából válasszuk ki a **Min**, a távozásnál pedig a **Max** értékeket! A *dátum* oszlopában a **Where** elem szerepeljen. A futtatás eredménye egyetlen értékpár lesz, az október 8-i adatok.

A *tólig* feladat SQL-nyelvű megoldása:

```
SELECT Min(érkezés) AS nyitás, Max(távozás) AS zárás
FROM ebéd
WHERE dátum=#10/08/2020#;
```

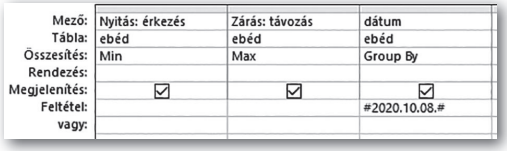
Mező:	Nyitás: érkezés	Zárás: távozás	dátum
Tábla:	ebéd	ebéd	ebéd
Összesítés:	Min	Max	Group By
Rendezés:			
Megjelenítés:	☑	☑	☑
Feltétel:			
vagy:			

A *tólignaponta* feladattal látszólag nehéz megbirkózni, hiszen az előző megoldás kibővítéséhez mindenképpen ismernünk kellene a dátumok listáját. Készítsünk másolatot a *tólig* lekérdezésről, majd módosítsuk azt! Töröljük a dátumot a feltételből, majd az *Összesítés* sorában a **Where** szót cseréljük a **Group By** kifejezésre! A lekérdezést futtatva pontosan a kívánt listát kapjuk. A **Group By** választásakor az történik, hogy a program az adott oszlopban szereplő mező vagy kifejezés szerint

csoportokat képez az egyéb feltételeknek megfelelő rekordokból. Ezt követően a program minden ilyen csoportban meghatározza és megjeleníti a minimális és a maximális értéket. Figyeljünk arra, hogy ekkor a *dátum* oszlop megjelenítését be kell kapcsolnunk.

A *tólignaponta* feladat SQL-nyelvű megoldása:

```
SELECT Min(érkezés) AS nyitás, Max(távozás) AS zárás, dátum
FROM ebéd
GROUP BY dátum;
```

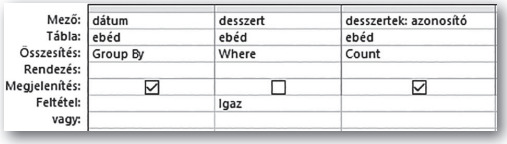


Mező:	Nyitás: érkezés	Zárás: távozás	dátum
Tábla:	ebéd	ebéd	ebéd
Összesítés:	Min	Max	Group By
Rendezés:			
Megjelenítés:	☑	☑	☑
Feltétel:			#2020.10.08.#
vagy:			

Megnézve az így kapott listát, az egyik sorban megtaláljuk a választ a *tólig* feladat kérdésére. Mi lenne, ha csak ezt listáznánk ki? Készítsünk egy másolatot ezúttal a *tólignaponta* lekérdezésből *tóligmásképp* néven! Egészítsük ki ezt a lekérdezést a dátumra vonatkozó feltétellel! Lefuttatva a lekérdezést, a megjelenített rekord egyezik azzal, amit a *tólig*nál láttunk. Vajon eltér-e, és ha igen, miben a *tólig* és *tóligmásképp?* Amikor a dátum a **WHERE** alatt szerepel, a program kigyűjti a feltételnek megfelelő rekordokat, majd azokra alkalmazza a minimum és a maximum meghatározását. Ha a **GROUP BY** oszlopába írjuk a feltételt, akkor először – ebben a feladatban – elvégzi a csoportosítást, tehát minden dátumra megállapítja a minimum- és a maximumértékét, majd ebből az eredményhalmazból szűr. Ez a második megoldás erőforráspazarló, hiszen így minden dátumra el kell végeznie a számítást, míg az első esetben csak az adott dátumhoz tartozó rekordokra.

A *tóligmásképp* lekérdezés SQL-nyelven:

```
SELECT Min(érkezés) AS nyitás, Max(távozás) AS zárás
FROM ebéd
GROUP BY dátum
HAVING dátum=#10/08/2020#;
```

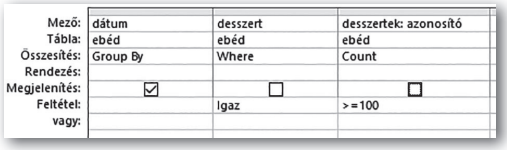


Mező:	dátum	desszert	desszertek: azonosító
Tábla:	ebéd	ebéd	ebéd
Összesítés:	Group By	Where	Count
Rendezés:			
Megjelenítés:	☑	☐	☑
Feltétel:		Igaz	
vagy:			

A *desszertnaponta* feladat megoldásánál már tudjuk, hogy az összesítés bekapcsolása a lényeges pont. Természetesen itt is dátum szerint csoportosítunk, a desszertre vonatkozó feltételnél pedig tudatosan a WHERE-t használjuk. Gyakran elkövetik azt a hibát, hogy valamelyik feltételként felhasznált oszlopnál választják ki a számlálást (COUNT) az *Összesítés* sorban. Ilyenkor a feltétel nem a tábla adataira, hanem az eredménytáblára vonatkozik, ezért céljainktól eltérő eredményt kapunk. A COUNT használata során minden olyan rekordot figyelembe vesz, ahol a választott mező nem üres. Mivel a legtöbb mező minden rekordban ki van töltve, ezért látszólag bármelyiket választhatjuk. Biztosan nem hibázunk, ha olyan mezőt választunk, amely sohasem üres. Az azonosító szerepű mező mindig megfelel ennek a kritériumnak.

A *desszertnaponta* lekérdezés SQL-nyelven:

```
SELECT dátum, Count(azonosító)
FROM ebéd
WHERE desszert
GROUP BY dátum;
```



Mező:	dátum	desszert	desszertek: azonosító
Tábla:	ebéd	ebéd	ebéd
Összesítés:	Group By	Where	Count
Rendezés:			
Megjelenítés:	☑	☐	☐
Feltétel:		Igaz	>=100
vagy:			

A *desszert100* feladat megoldása nem más, mint a *desszertnaponta* módosítása azzal, hogy a darabszámra vonatkozó feltételt megfogalmazzuk. Lényeges, hogy erre itt nem a WHERE-t kell használnunk, mivel a feltétel a számlálás eredményére vonatkozik.

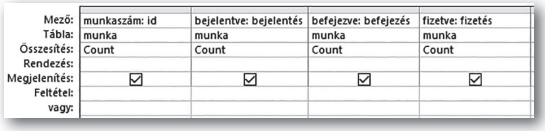
A *desszert100* lekérdezés SQL-nyelven:

```
SELECT dátum, Count(azonosító)
FROM ebéd
WHERE desszert
GROUP BY dátum
HAVING Count(azonosító)>=100;
```

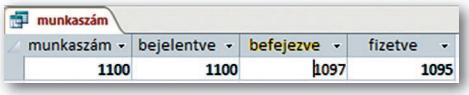
27. példa: A Null értékű mezők hatása az eredményre

A *javítás* adatbázist használva oldjuk meg az alábbi feladatokat!

- Adjuk meg, hogy hány munka adatai szerepelnek az adatbázisban! *(munkaszám)*
- Listázzuk ki, hogy az egyes években mennyi bevételt könyvelhetett el a cég, ha 5000 Ft-os óradíjjal dolgoztak! Csak azokat a munkákat vegyük figyelembe, amelyeket már ki is fizettek! *(bevételévente)*
- Határozzuk meg, mennyi volt típusonként a javítások átlagos anyagára! *(típusátlag)*



Mező:	munkaszám: id	bejelentve: bejelentés	befejezve: befejezés	fizetve: fizetés
Tábla:	munka	munka	munka	munka
Összesítés:	Count	Count	Count	Count
Rendezés:				
Megjelenítés:	☑	☑	☑	☑
Feltétel:				
vagy:				



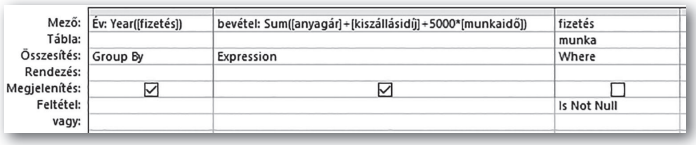
munkaszám

munkaszám	bejelentve	befejezve	fizetve
1100	1100	1097	1095

A *munkaszám* lekérdezés első ránézésre a legegyszerűbb feladatok egyike. Valóban az, de nem tanulság nélküli. Az azonosító *(id)* megszámlálása az adatbázisba bejegyzett munkák számát adja. Nézzük meg, hogy milyen eredményt kapunk, ha nem az *id*-t, hanem a *bejelentés,* a *befejezés,* valamint a *fizetés* mezőkre használjuk a COUNT függvényt! Azt látjuk, hogy a *bejelentve* mezőnél a darabszám egyezik a munkaszámmal. Ez nem meglepő, mert a munkát bejelentéskor veszik fel. A befejezett munkák száma természetesen kevesebb az összesnél, és néhány munka még a befejezettek közül sincs kifizetve. Azon mezőkbe, amelyek értéke nem ismert, nem kerül semmi, pontosabban egy úgynevezett NULL értéket vesznek fel.

> A **NULL** értékű mezőket az Access nem veszi figyelembe a számlálásnál.

A *bevételévente* lekérdezés csak annyi újdonságot hordoz, hogy a csoportosítást és az összegzést (SUM) is számított érték alapján végezzük, valamint több táblából használjuk az adatokat a számításhoz.



Mező:	Év: Year([fizetés])	bevétel: Sum([anyagár]+[kiszállásidíj]+5000*[munkaidő])	fizetés
Tábla:			munka
Összesítés:	Group By	Expression	Where
Rendezés:			
Megjelenítés:	☑	☑	☐
Feltétel:			Is Not Null
vagy:			

A *bevételévente* lekérdezés SQL-nyelven:

```
SELECT Year(fizetés) AS év,
    Sum(anyagár+kiszállásidíj+5000*munkaidő) AS bevétel
FROM munka, típus
WHERE munka.típusid=típus.id
    AND fizetés is Not NULL
GROUP BY Year(fizetés);
```

Mező:	név	átlagár: anyagár
Tábla:	típus	munka
Összesítés:	Group By	Avg
Rendezés:		
Megjelenítés:	☑	☑
Feltétel:		
vagy:		

A *típusátlag* lekérdezésben az ÁTLAG függvény (AVG) alkalmazására láttunk példát.

A **MIN()**, **MAX()**, **COUNT()**, **SUM()**, **AVG()** függvényeket **aggregáló** (csoportosító) függvényeknek nevezzük, mert ezeket általában a rekordok meghatározott csoportjaira alkalmazzuk.

Feladatok

Oldjuk meg a feladatokat a *városok* adatbázist használva, lekérdezőrács segítségével!

a) Adjuk meg, milyen lélekszámú a legkisebb magyar város! *(f11a)*
b) Határozzuk meg, hogy hány fő él városban Magyarországon! *(f11b)*
c) Listázzuk ki a megyei jogú városok átlagos lélekszámát! *(f11c)*
d) Határozzuk meg, hogy hány város található Pest megyében! *(f11d)*
e) Listázzuk ki, hogy az egyes megyékben hány város van! A sorrendet a városok száma határozza meg! *(f11e)*

Segédlekérdezések

Valaha, amikor a mai adatbázisok tábláit még nem a számítógép háttértára őrizte, hanem papírra nyomtatottan használtuk őket, a legtöbb kérdésre több lépésben adtuk meg a választ. Ha meg akartuk határozni, hogy mi a székhelye annak a megyének, ahol Sarkad található, a következő utat jártuk be. Először megkerestük a települések között Sarkadot, leolvastuk a megye nevét, tehát már tudtuk, hogy Békés megye székhelyét kell megtalálnunk. Aztán a megyék táblázatában Békés megyénél megnéztük a megyeszékhely nevét. Tehát két lépés kellett ahhoz, hogy megkapjuk Békéscsabát. Valószínűleg mindenki tud példát mondani arra, amikor a kívánt információhoz – a fentihez hasonlóan – többlépéses út vezet. Sok olyan feladat van, amelyet ma, az adatbázisok korában is így oldunk meg, mert számunkra ez a megoldás természetes útja, vagy másképp nem is lehetséges.

Ha egy lekérdezés elkészítése során más lekérdezéseket is készítünk, és azokat fel is használjuk, **allekérdezésről** beszélünk. A külön elmentett lekérdezést **segédlekérdezésnek** nevezzük.

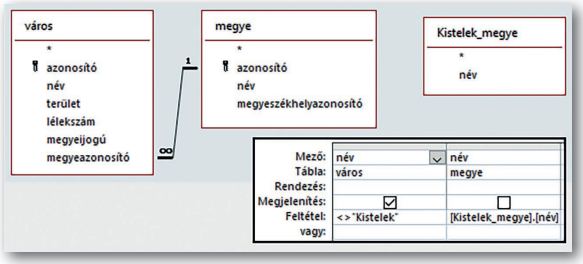
Egyes adatbázis-kezelő rendszerekben segédlekérdezések helyett úgynevezett *nézet* táblákat használhatunk. Ahogy a segédlekérdezés, úgy a *nézet* tábla is mindig az általa használt táblák aktuális állapotával dolgozik. Ezekből további lekérdezések készíthetők.

A segédlekérdezés és az allekérdezés gyakran egymás alternatívái. A segédlekérdezésekből felépülő lekérdezések általában könnyebben áttekinthetők, mint az allekérdezést használó megoldások.

28. példa: Segédlekérdezéssel megoldható problémák

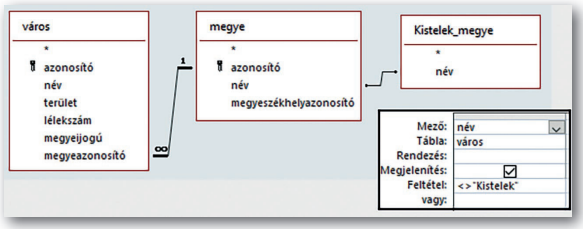
A *városok* adatbázist használva oldjuk meg az alábbi feladatokat!

- Készítsünk lekérdezést, amely megadja, mely városok találhatók ugyanabban a megyében, mint Kistelek! Kistelek nevét ne jelenítsük meg! *(Kistelek)*
- Határozzuk meg, hogy mely városok lélekszáma kisebb, mint Visegrádé! *(Visegrádnálkisebb)*
- Adjuk meg, hány város területe kisebb az átlagosnál! *(átlagnálkisebb)*
- Listázzuk ki, mely megyékben van ugyanannyi város, mint Hevesben! *(Heves)*

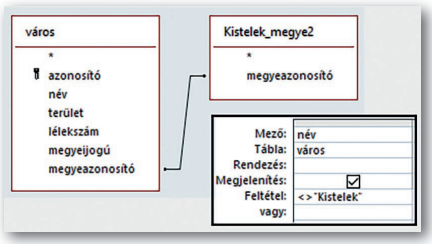


A *Kistelek* feladatra több, egymással egyenértékű megoldást mutatunk. Az első változat illeszkedik leginkább a természetes emberi gondolkodásmódhoz. Először elkészítettük a *Kistelek_megye* lekérdezést, amely két tábla felhasználásával Kistelek megyéjének nevét adja meg. Az ábrán látható, hogy a válaszadáshoz készített lekérdezésbe nem csupán a két táblát, hanem a *Kistelek_megye* lekérdezést is felvettük mint segédlekérdezést. Két feltételt adtunk meg. Az egyikkel Kistelek megyéjének nevére szűrünk. Ha a feltétel sorában egy lekérdezés vagy tábla egy mező-

jét adjuk meg, akkor a tábla/lekérdezés és a mező nevét is szögletes zárójelbe kell írnunk, és a kettő közé pontot tennünk. A másik szűrési feltétellel pedig kizárjuk a megjelenítendő városok közül Kisteleket.



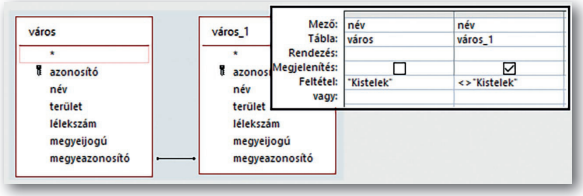
Az előző megoldás jól követhető és logikus, azonban érdemes egy kicsit módosítani, hogy egyszerűbbé váljon. Vegyük észre, hogy a megye nevére vonatkozó feltétel pontosan a *Kistelek_megye* lekérdezés és a *megye* tábla közötti kapcsolat. Ezt grafikusan is megjeleníthetjük a mellékelt ábrának megfelelően.

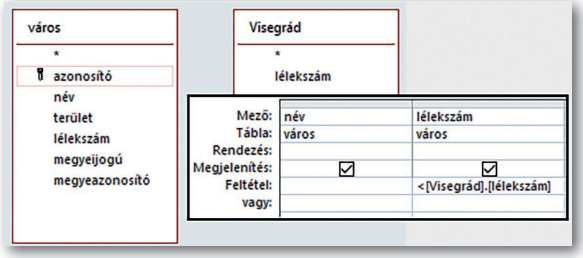


Ha alaposan megnézzük az előző megoldásokat, észrevehetjük, hogy a megye neve nem is lényeges a feladat szempontjából. A lényeg, hogy ugyanabban a megyében legyen a város. A megyét a *város* táblában található *megyeazonosító* meghatározza, tehát a megoldás egyszerűsíthető a következő módon: adjuk meg azokat a városokat, amelyek *megyeazonosítója* ugyanaz, mint Kisteleké! Ezt a megoldást természetesen olyan lekérdezéssel kell előkészítenünk, amely a megye neve helyett a megye azonosítóját adja. Ez a *Kistelek_megye2* lekérdezés, amely csak egyetlen táblát használ. Maga a lekérdezés az ábrán látható formára egyszerűsödik.

A *Kistelek* lekérdezés harmadik megoldásának SQL-nyelvű változata, amely segédlekérdezés helyett allekérdezést használ:

```
SELECT név
FROM város, (SELECT megyeazonosito FROM város WHERE név=”Kis-
  telek”) AS Kistelek_megye
WHERE város.megyeazonosito=Kistelek_megye.megyeazonosito
  AND név<>”Kistelek”;
```



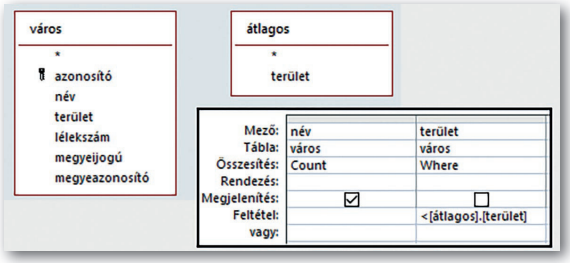


Az előző három megoldás mindegyike egy lekérdezéssel készítette elő a kérdés megválaszolását. Létezik olyan megoldás is, amely nem igényel pluszlekérdezést, de a *város* tábla többszöri felhasználását igen. Ez a legközvetlenebb megoldás, hasonlóval már találkoztunk korábban is, de ezt a formát kevesen szokták használni logikai összetettsége miatt.

A *Visegrádnálkisebb* lekérdezéshez szükségünk van egy Visegrád lélekszámát meghatározó lekérdezésre. Ezt használjuk fel a következő lépésben.

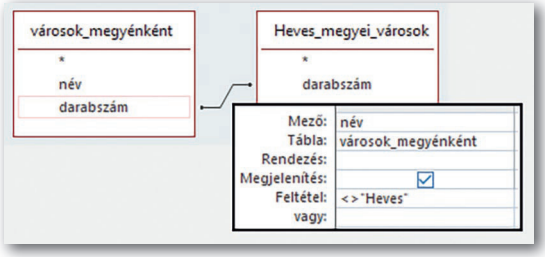
Itt nincs mód összekapcsolni egy másik mezővel, hiszen nem egyenlő, hanem kisebb reláció van a keresett városok és a meghatározott szám között. Az ábrán látható feltételt begépelhetjük, ekkor vigyáznunk kell, hogy az összes zárójel a helyén legyen, vagy a *helyi menü* > *Szerkesztés* pontját választva is megalkothatjuk.

Az *átlagnálkisebb* lekérdezés elkészítéséhez szükségünk van a települések átlagos területére. Nyilván ehhez az értékhez kell viszonyítani az egyes települések területét. A tábla és az átlag értékét adó lekérdezés között nincs kapcsolat, a lekérdezőrácson tudjuk beállítani a szükséges feltételt. Mivel nem a kimenet eredményére szűrünk, ezért az összesítés sorában a WHERE-t kell beállítanunk.



Az *átlagnálkisebb* lekérdezés megoldásának SQL-nyelvű változata, amely segédlekérdezés helyett allekérdezést használ:

```
SELECT Count(név)
FROM város
WHERE terület<(SELECT AVG(terület) FROM város);
```



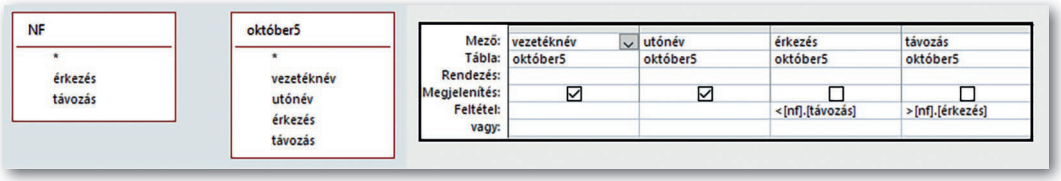
A *Heves* lekérdezés előkészítéséhez két lekérdezést is praktikus létrehoznunk. Az egyikben meghatározzuk a Heves megyei városok számát, a másikban pedig azt, hogy az egyes megyékben hány város van. Mivel az egyező darabszám a kérdés, ezért ezt a két darabszám mező közötti kapcsolat megadásával is beállíthatjuk.

29. példa: Táblák (és lekérdezések) között többszörös kapcsolat

Az *étkezés* adatbázist használva oldjuk meg az alábbi feladatokat!

- Készítsünk lekérdezést, amely megadja, kikkel találkozhatott ebéd közben Nagy Ferenc 2020. 10. 05-én! *(találkozás)*
- Határozzuk meg, hogy mely napokon hányan ették ugyanazt a menüt, és választottak ugyanúgy desszertet, mint Németh Sára! *(egyezően)*
- Az előzőhöz hasonlóan határozzuk meg, hogy mely napokon hányan ették ugyanazt a menüt, és választottak ugyanúgy desszertet, mint Németh Sára! Németh Sárát ne számítsuk közéjük! *(nélkül)*

A *találkozás* lekérdezés elkészítéséhez úgy jutunk közelebb, ha tudjuk Nagy Ferenc 2020. 10. 05-i érkezési és távozási időpontját. Ezt az *NF* lekérdezés adja meg. Az *október5* lekérdezés kilistázza az összes, 2020. 10. 05-én ott ebédelő érkezési és távozási adatait. A két

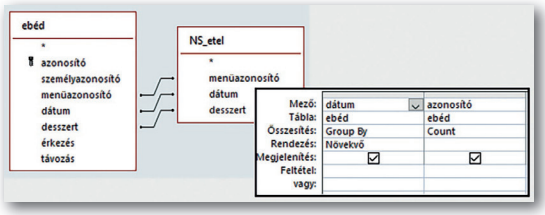


segédlekérdezés segített megszabadulni a probléma szempontjából érdektelen adatoktól, így már koncentrálhatunk csak a lényegre.

Gondoljuk végig az ábra segítségével, hogy miképpen helyezkedhetnek el az egyes vendégek jelenléti intervallumai Nagy Ferenc érkezési és távozási időpontjához képest!

A piros és kék vonalak ábrázolják két személy jelenlétét egy eseményen. A vonal kezdete az érkezés időpontját, a vége a távozását jelzi. Ha a két vonal legalább részben fedésben van, akkor találkoztak. Találkozásra csak akkor kerül sor, ha a vonalak helyzete egyezik a fenti négy egyikével. Miképpen tudjuk leírni a találkozást az érkezési és távozási időpontokkal? Mikor találkozott a piros a kékkel? Ehhez szükséges, hogy a piros hamarabb érkezzen, mint ahogy a kék távozik, de kell az is, hogy a piros később távozzon, mint ahogy a kék érkezik. Ennek a két feltételnek együtt kell teljesülnie. Ez a találkozással kapcsolatos feltétel legegyszerűbb megfogalmazása.

A fenti meggondolás alapján megfogalmazhatjuk a két segédlekérdezésen alapuló, megoldást adó feltételt. Ez a megoldás több más, intervallumok fedésén alapuló probléma leküzdéséhez is utat mutathat.

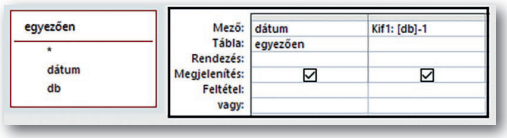


Az *egyezően* lekérdezéshez csak egy segédlekérdezést készítettünk. Az *NS_étel* lekérdezés megadja, hogy melyik napon melyik menüt ette Németh Sára, és azt is, hogy fogyasztott-e hozzá desszertet. A főlekérdezésben a segédlekérdezés kimeneti mezőit használjuk fel. Mivel ugyanakkor és ugyanazt kell enniük a keresett személyeknek, mint Németh Sárának, ezért a megfelelő mezők egyezésének teljesülnie kell. Ezt beállíthatjuk a lekérdezőrácson, de megadhatjuk az *ebéd* tábla és az *NS_étel* lekérdezés közötti kapcsolatok segítségével is. Itt az utóbbit választottuk. A rács pusztán azt írja le, hogy az egyes napok hányszor fordulnak elő. Fontos megjegyezni, hogy ez a lekérdezés olyan eredményt ad, amelyben Németh Sára is szerepel.

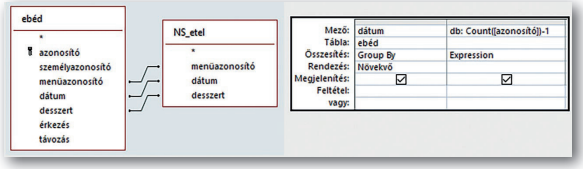
Az *egyezően* lekérdezés megoldásának SQL-nyelvű változata, amely segédlekérdezés helyett allekérdezést használ:

```
SELECT ebéd.dátum, Count(azonosító)
FROM ebéd,
    (SELECT menüazonosító, dátum, desszert
          FROM vendég, ebéd
          WHERE vendég.azonosító=ebéd.személyazonosító
                AND vezetéknév= ”Németh”
                AND utónév= ”Sára”
    ) AS NS_étel,
WHERE ebéd.dátum=NS_étel.dátum
    AND ebéd.menüazonosító=NS_étel.menüazonosító
    AND ebéd.desszert=NS_étel.desszert
GROUP BY ebéd.dátum;
```

A *nélkül* lekérdezést látszólag nagyon egyszerű elkészíteni, hiszen egyszerűen le kell vonni egyet az előző eredmény minden sorából. Ezt a matematikai műveletet nem alkalmazhatjuk közvetlenül a lekérdezőrácson.



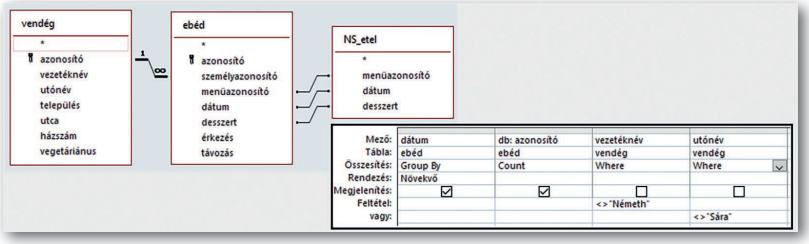
A lekérdezés elkészítésének legegyszerűbb módja, ha az előbb elkészített *egyezően* lekérdezést mint segédlekérdezést használjuk. Ekkor csak annyi a teendőnk, hogy minden ott kapott számértékből kivonunk egyet.



A másik lehetőség az *egyezően* lekérdezés módosításával adódik, de ehhez a mező sorában szerepelő kifejezést, amely a COUNT függvénynél eggyel kisebbet ad eredményül, nekünk kell begépelnünk. Ha az *Összesítés* gomb be van kapcsolva, és a mező sorában nem mezőnév szerepel, az *Összesítés* sorában általában az *Expression* szót kell kiválasztanunk a listából.

A harmadik lehetőség is egészen kézenfekvőnek tűnik: a megszámláltak közül ki kell zárnunk Németh Sárát. Ehhez az *egyezően* lekérdezést kell kibővítenünk a *vendég* tábla felvételével. A lekérdezőrácson kell úgy módosítanunk, hogy Németh Sárát ne számlálja. A feltétel megfogalmazásához vissza kell nyúlni a matematikai logikában tanultakhoz, mely szerint a Nem(A és B) kifejezés egyenértékű a Nem(A) vagy Nem(B) kifejezéssel.

Érdekes volt megtapasztalni, hogy egyetlen apró kis változtatás a feladatban jelentős módosítást igényel a megoldásban.



Feladatok

Oldjuk meg a feladatokat a *javítás* adatbázist használva, lekérdezőrács segítségével!

a) Adjuk meg, kik laknak ugyanazon a településen, mint Balogh Attila! *(f12a)*
b) Listázzuk ki, hogy kik fordultak ugyanolyan típusú problémával a céghez, mint Kovács Emma 2019-ben! *(f12b)*
c) Határozzuk meg, hány számlán szerepelt az átlag kétszeresénél nagyobb anyagköltség 2019-ben! *(f12c)*
d) Adjuk meg, hogy melyek azok a hibatípusok, amelyek a villannyal kapcsolatos problémáknál többször fordulnak elő! *(f12d)*
e) Listázzuk ki, hogy mely hibatípusok átlagos időszükséglete nagyobb, mint a klímával kapcsolatos hibák javítási idejének átlaga! *(f12e)*

Mi van, ha nincs?

Az eddig tárgyalt feladatok mindegyike arra vonatkozott, amit az adatbázis tábláiban közvetlenül megtaláltunk. Ritkábban bár, de előfordulnak olyan kérdések is, amelyek pontosan arra vonatkoznak, ami nem szerepel. Az *étkezés* adatbázisban rögzítették azt, hogy ki ebédelt 2020. 10. 05-én, de nem szerepel, hogy ki nem ebédelt akkor. Természetesen tudunk válaszolni erre a kérdésre, de csak azzal a megszorítással, hogy az adatbázisban szereplő személyek közül ki nem ebédelt. Gondoljunk vissza arra, hogy mit tanultunk a halmazokról matematikából! Akármiről is beszéltünk, mindig volt egy alaphalmaz. Bármely halmaz került szóba, az ennek az alaphalmaznak egy részhalmaza volt. Értelmeztük a halmaz kiegészítő halmazát, azaz komplementerét. Itt az összes vendég számít alaphalmaznak. Az ebédelők a részhalmaz, amelyet tárolunk, és ennek komplementerére vagyunk kíváncsiak. Tehát adatbázis-kezelésből ezen komplementerhalmaz előállítását kell megismernünk.

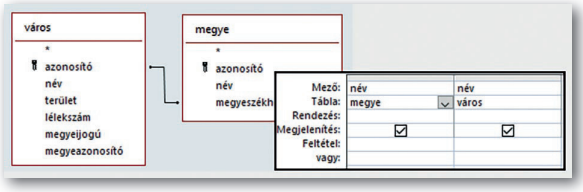
A legnagyobb nehézséget nem az jelenti majd, hogy miképpen készítjük a komplementerhalmazt, hanem annak felismerése, hogy mikor van rá szükségünk.

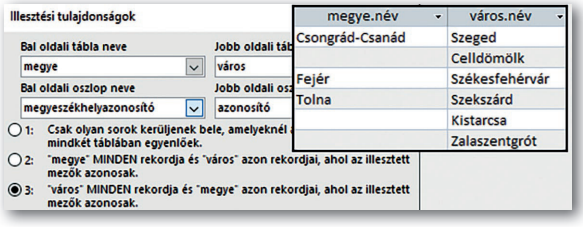
30. példa: Hiányzó adat keresése

A *városok* adatbázist használva oldjuk meg az alábbi feladatokat!

- Adjuk meg azokat a városokat, amelyek nem megyeszékhelyek! *(nemszékhely)*
- Melyek azok a megyék, amelyekben nincs 50 ezer főnél népesebb város? *(nincsnagy)*

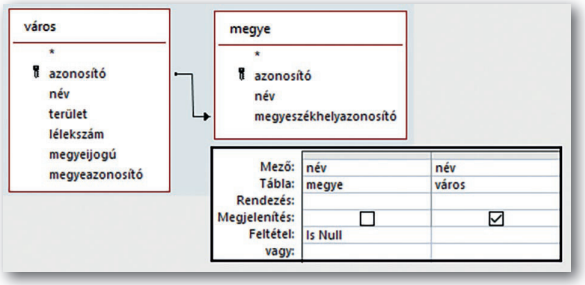
A *nemszékhely* lekérdezés kapcsán azt nézzük meg, ami van, és nem azt, ami nincs. A *megye* táblában a *megyeszékhelyazonosító* adja meg azt a *várost,* amely a megyeszékhely. Ha a *megye* és a *város* táblák közötti kapcsolatot közöttük hozzuk létre, akkor nem minden városhoz kapcsolódik megye, és az eredményhalmazba 19 rekord tartozik majd.





Ezt a *megyeszékhely* lekérdezést fogjuk majd átalakítani a cél érdekében. Kattintsunk duplán a két táblát összekötő, kapcsolatot jelző vonalra! A megjelenő párbeszédablakban a két tábla közötti kapcsolatot három különböző módon állíthatjuk be. Ha nem módosítjuk az alapértelmezést (1-es), csak azok a rekordok jelennek meg, amelyekben a két összekapcsolt mezőnek egyezik az értéke. Ha a 3-ast választjuk, akkor jelen esetben úgy hozza létre a kapcsolatot, ha a *város* tábla minden elemét felsorolja a *város.név* oszlopban, mellé pedig akkor írja be a megye nevét, ha az a megyeszékhely, egyébként a mező üresen marad. A lekérdezés eredménye látszik a képen is.

A *nemszékhely* lekérdezés végleges formáját a következő oldali ábra mutatja. A kapcsolati rajzból leolvasható, hogy a *város* tábla minden eleméhez próbál értéket társítani a *megye* táblából. Fentebb láttuk, hogy a megyeszékhelyek esetén olvasható a megye



neve, egyébként a mező üres. Az üres mezőkre az **Is Null** kifejezéssel tudunk szűrni, a nem üresekre pedig az **is Not Null** kifejezéssel. Most az előbbit használjuk a feladat megoldása érdekében.

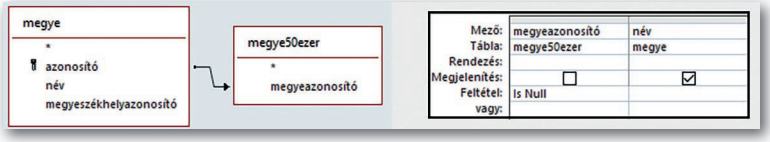
A *nemszékhely* lekérdezés SQL-nyelvi alakja:

```
SELECT város.név
FROM város LEFT JOIN megye ON megye.megyeszékhelyazonosító =
  város.azonosító
WHERE megye.név Is Null;
```

A *nemszékhely* lekérdezést elkészíthetjük allekérdezéssel is:

```
SELECT név
FROM város
WHERE azonosító NOT IN (SELECT megyeszékhelyazonosító FROM megye);
```

Az allekérdezéses megoldás fejezi ki a legszemléletesebb módon a bevezetőben ismertetett gondolatmenetet. Úgy fordíthatjuk hétköznapi nyelvre, hogy kilistázzuk azon városok nevét, amelyeknek az *azonosítója* nem szerepel a *megye* táblában tárolt *megyeszékhely-azonosítók* között.



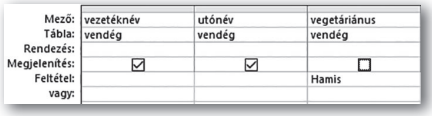
A *nincsnagy* lekérdezést közvetlenül, két táblát használva nem tudjuk elkészíteni. Abból csak azt tudnánk meghatározni, hogy melyikben van legfeljebb 50 ezer fős város. Az viszont nem jelenti azt, hogy nincs nagyobb város. Viszont ha elkészítjük az 50 ezer főnél népesebb városok megyéinek listáját, azt segédlekérdezésként fel tudjuk használni a *nemszékhely* lekérdezésnél megismert formában.

31. példa: Összetett lekérdezések

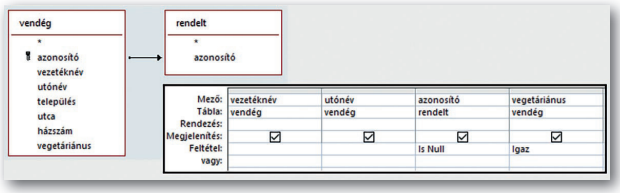
Az *étkezés* adatbázist használva oldjuk meg az alábbi feladatokat! Vajon melyik igényli a most szerzett ismereteket, és melyiket tudtuk volna már korábban megoldani?

- Készítsünk lekérdezést, amely megadja, hogy kik nem vegetáriánusok! *(nemvegetáriánus)*
- Határozzuk meg, hogy a vegetáriánusok közül ki nem rendelt vegetáriánustól különböző menüt! *(nemrendelt)*
- Adjuk meg lekérdezés segítségével, hogy mely napokon fogyott legfeljebb 80 desszert! *(max80)*

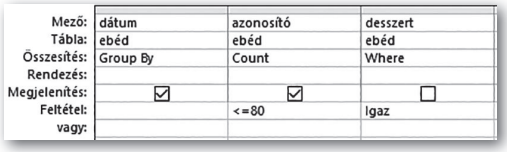
Alapos végiggondolás után a legtöbben azt mondják, hogy a *nemvegetáriánus* és *max80* lekérdezések azok, amelyeket korábban is meg tudtunk volna oldani. Lássuk, valóban így van-e!



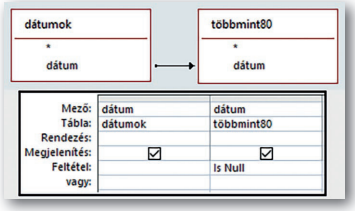
A *nemvegetáriánus* lekérdezéshez valóban nem kell pluszismeret, mivel a *vendég* tábla tartalmazza ezt a tulajdonságot. Így, ha a *vegetáriánus* értéke hamis, az illetőt meg kell jeleníteni. Mikor lett volna szükségünk a frissen tanultakra? Ha a vegetáriánusokat – alapvetően ezt a tulajdonságot – egy külön tábla tartalmazza, akkor a probléma hasonló lenne a *nemszékhely* feladathoz.



A *nemrendelt* lekérdezésnél – az előzőleg alkalmazott módszer szerint – elkészítjük azt a lekérdezést, amely megadja, kik azok, akik vegetáriánusként rendeltek nem vegetáriánus menüt. Ezt a lekérdezést mint segédlekérdezést felhasználjuk a megoldásban, mégpedig a *vendég* táblához illesztve. A *vendég* táblából mindenki szerepel, a *rendelt* lekérdezésből pedig azok, akiknél a megadott kapcsolatnak megfelelően van vele egyező érték. Ahol nincsenek megjelenített értékek, azokat kell megadnia a lekérdezésnek. Ezen rekordokra az **Is Null** feltétellel szűrünk a rácson – ahogyan az ábra is mutatja.



A *max80* lekérdezés elkészítése nem látszik túlzottan bonyolultnak. Egyetlen tábla segítségével válaszolhatunk: naponkénti csoportosításban megszámláljuk a desszertet fogyasztókat. Ha számuk nem haladja meg a 80 főt, akkor megjelenítjük. Mi ezzel a probléma? Látszólag semmi. Próbáljunk ki valamit! Jegyezzünk be egy újabb rekordot az *ebéd* táblába 2020. 11. 02-ára! Az 1-es azonosítójú személy C menüt rendelt, és nem kért desszertet. Az érkezés és távozás idejét állítsuk be tetszőlegesen! Futtassuk le az előző lekérdezést! Az eredmény 2020. 11. 02-át nem tartalmazza, pedig aznap csak ezt az egy ebédet fogyasztották, és így biztosan kevesebben fogyasztottak desszertet 80 főnél. Miért nem szerepel ez a dátum? Mert megszámlálni csak azt lehet, ami volt. Aznap pedig nem volt desszertet fogyasztó.



Hogyan oldható meg ez a probléma? Két segédlekérdezést kell készítenünk. Az egyiknek meg kell adnia az összes dátumot *(dátumok)*, a másiknak *(többmint80)* pedig felsorolnia, hogy mely napokon volt több mint 80 fő desszertet fogyasztó. Végül a *dátumok* lekérdezésből – a megszokott módon – meg kell jelenítenünk azokat a napokat, amelyek nem szerepelnek a *többmint80* lekérdezés kimenetében.

Feladatok

Oldjuk meg a következő feladatokat lekérdezőrács segítségével! Van-e olyan, amelyet a most tanult ismeretek nélkül is meg lehet oldani?

a) A *javítás* adatbázisban tároltak alapján adjuk meg, kik laknak ugyanazon a településen, mint Balogh Attila! *(f13a)*
b) A *javítás* adatbázis alapján adjuk meg, milyen típusú javításokat végeztetett Katona Lara, amiket Mezei Fanni nem! *(f13b)*
c) Az *étkezés* adatbázis használatával adjuk meg, kik azok, akik soha nem ettek húsimádó menüt! *(f13c)*
d) Az *étkezés* adatbázis alapján állapítsuk meg, milyen menüt nem választott soha Barna Panna! *(f13d)*
e) Az *étkezés* adatbázisban tároltak alapján adjuk meg, kikkel nem találkozott Nagy Dávid 2020. 10. 20-án ebéd közben annak ellenére, hogy aznap ők is ott ebédeltek! *(f13e)*

Jelentések

A lekérdezések fő célja, hogy a tárolt adatok alapján információhoz jussunk. Nem érdekes, hogy milyen formában jelenik meg, hanem csak a tartalma. Azonban van, amikor fontos, hogy az előállított adatokat könnyen áttekinthetően, a lényeget kiemelve jelenítsük meg. Az eszközt, amellyel ezt a célt elérhetjük, jelentésnek nevezzük.

A **jelentés** az adatbázisban tárolt adatoknak és az azokból kinyerhető információknak az áttekinthető megjelenítésére szolgál – gyakran nyomtatott formában.

Jelentést készíthetünk táblából vagy lekérdezésből. Jelentés készülhet több táblából is, ha az Adatbáziseszközök Kapcsolatok lapján megadtuk a köztük lévő kapcsolatokat.

32. példa: Egyszerű jelentés egy tábla alapján

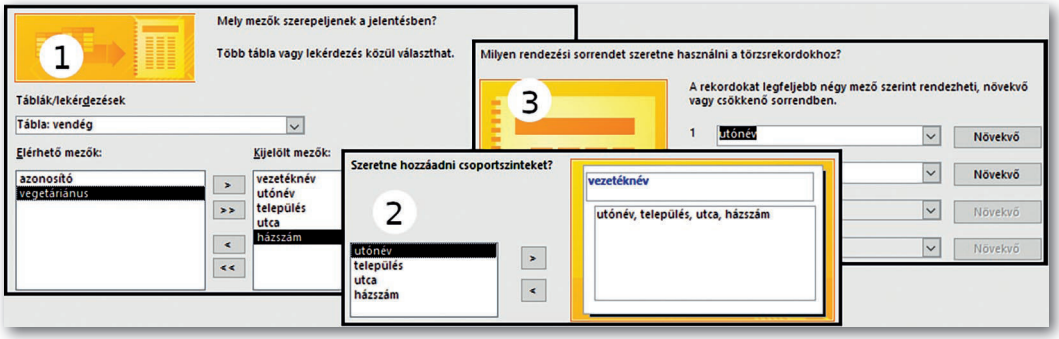
Készítsünk jelentést az *étkezés* adatbázis *vendég* táblájából az ábrán látható minta alapján! A személyeket csoportosítsuk vezetéknév szerint, azon belül pedig rendezzük az utónevük alapján! Jelenítsük meg a címüket is! Az elkészítés során ügyeljünk arra, hogy minden adat teljes egészében látható legyen! A fejrész tartalma ékezethelyesen jelenjen meg! Törekedjünk a mintán látható formai jellemzők kialakítására is! A kész jelentést őrizzük meg fájlba nyomtatva is! *(vendéglista)*

A vendégek listája betűrendben

vezetéknév	utónév	település	utca	házszám
Bakos				
	Gergely	Felsőváros	Rakéta utca	27
	Olivér	Nagyfalu	Rezeda dűlő	39
Balog				
	Lilien	Kisfalu	József utca	17
	Noel	Felsőváros	Verseny utca	57
Balogh				
	Adél	Alsóváros	Pellérdi út	21
	Eszter	Felsőváros	Görgey Artúr utca	4
	Hunor	Nagyfalu	Stiglicfogdosó	32
	Levente	Nagyfalu	Kút utca	3

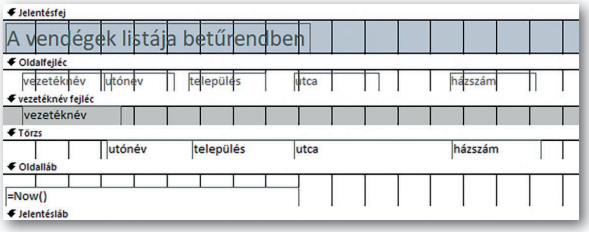
A folyamatot a *Létrehozás* > *Jelentések* > *Jelentés* varázsló elemre kattintva indíthatjuk el.

A *vendéglista* jelentés egyetlen adattáblából készül.



A varázslóban megtett lépések közül a fontosabbak láthatók az ábrán. (1) Kiválasztjuk a táblát, és megjelöljük azokat a mezőket, amelyeket felhasználunk. (2) Megadjuk, hogy mely mező vagy mezők szerint csoportosítunk. (3) Beállítjuk a sorba rendezés kulcsait és irányát. Ezeket nagyon könnyen megtehetjük a minta alapján. A folyamat további lépéseinél az elrendezésről, a grafikai és tipográfiai beállításokról döntsünk belátásunk szerint!

A program által előállított végeredmény gyakran elmarad elvárásainktól. Elképzelhető, hogy az egyes adatok távolsága a soron belül jóval nagyobb lesz, más adatok viszont nem is látszanak teljes egészében. A formai beállításokat az *Elrendezési* vagy a *Tervező*



nézetben módosíthatjuk. Ha még nem vagyunk rutinosak, az *Elrendezési nézetet* válasszuk, abban jobban látjuk változtatásaink hatását. A *Tervező nézet* – amellyel akár létre is hozhatjuk a jelentést – segít áttekinteni annak szerkezetét, amely az alábbi ábrán látható.

A jelentés részei:
Jelentésfej: A jelentésben egyszer, a legelején jelenik meg, tartalma állandó.
Oldalfejléc: Minden oldal tetején olvasható, általában a jelentés oszlopainak azonosítását szolgálja.
Csoportfejléc: Minden csoport kezdetén megtalálható, tartalma a csoportot alkotó tulajdonság értéke.
Törzs: A jelentésbe felvett tulajdonságokat tartalmazza – a csoportfejléc adatait kivéve.
Csoportlábléc: Általában a csoporthoz tartozó összegzést, a rekordszámot, a számokat tartalmazó mezők összegét, átlagát, minimumát, maximumát tartalmazza. (Ez az ábrán nem látható, mert ebben a feladatban üres.)
Oldalláb: Minden oldal alján olvasható szöveg, jellemzően dátum, időpont, oldalszám szerepel benne.
Jelentésláb: A jelentésben egyszer, a végén olvasható, tartalma gyakran összegzés.

Váltsunk át a *Tervező nézetre,* és vessük össze az itt látható képet a mintaként adottal!

- Állítsuk be az egyes mezők méretét úgy, hogy a tárolt adatok elférjenek bennük, de ne legyenek túl szélesek! Figyeljünk arra, hogy a fejlécben az oszlopnevek méretét is módosítsuk!
- Változtassuk meg az egyes mezők helyét úgy, hogy ne legyen köztük túl nagy távolság! A pozíció beállítását kövesse az oszlopnevek helye is!
- Írjuk át az oszlopnevek tartalmát a mintának megfelelően! Ha nem áll rendelkezésre minta, legyen tartalmilag kifejező és ékezethelyes!
- Módosítsuk a jelentésfejet a minta szerint! Ha nincs előírva a tartalma, akkor válasszunk rövid, kifejező címet!

33. példa: Jelentés több tábla felhasználásával

Több táblát felhasználva is készülhet jelentés, ha a táblák kapcsolatát az adatbázisban megadtuk.

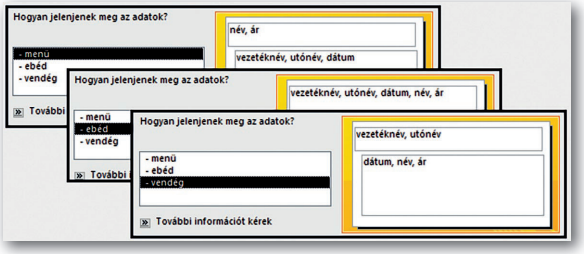
Készítsünk jelentést az *étkezés* adatbázis tábláiból a következő oldali ábrán látható minta alapján!

- Jelenítsük meg, hogy melyik napon ki milyen menüt evett!
- Az adatokat csoportosítsuk dátum szerint, a személyeket jelenítsük meg vezetéknév, azon belül utónév szerinti sorrendben!
- Tüntessük fel a menü nevét és árát is! A számértéket állítsuk be nulla tizedesjegy pontosságra, mögötte jelenítsük meg a pénznemet is!
- Minden napra adjuk meg az aznapi ebédelők számát és az ebédből származó bevételt!

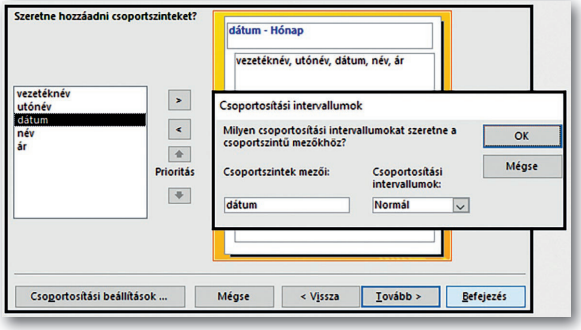
Nézzük, hogy milyen újdonságokat tartalmaz ez a feladat!

Az (1) lépésben egymás után több táblát kell felvennünk, és táblánként a megfelelő mezőket választanunk.

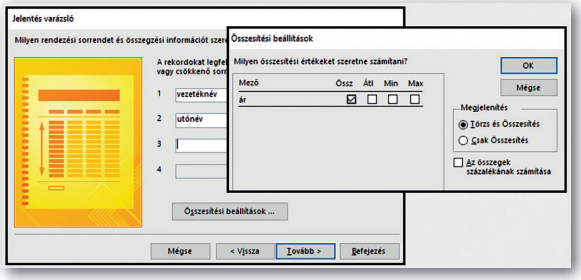
A (2), csoportképzést megadó lépés előtt a felhasznált táblákhoz és az adatbázis szerkezetéhez illeszkedő beállítások – nézetek – közül választhatunk. Az ábrán is jól látható a táblák kapcsolata: ha a menü oldaláról szemléljük, akkor egy menühöz több ebéd tartozik, ezért a menüt választva a program automatikusan megjelöli a csoportosítás alapjának. Ugyanígy történik a vendég választásakor, mivel egy vendéghez szintén több ebéd tartozhat. Ha az ebédet jelöljük meg, akkor nem kínál fel automatikusan *csoportszintet,* mert egy ebédhez pontosan egy vendég és pontosan egy menü tartozik. Mivel nekünk *dátum* szerint kell csoportosítanunk, ezért e lehetőségek közül válasszuk az ebédet, majd a következő lépésben a *dátumot* jelöljük meg a csoportosítás alapjának! Ha dátum típusú a *csoportszint,* az Access alapértelmezésben havi csoportosítást állít be. Ezt a *Csoportosítási beállítások* gombra kattintva céljainknak megfelelően tudjuk módosítani. A napi csoportosításhoz válasszuk a *Normál* beállítási lehetőséget! Szöveg típusú adatnál a teljes tartalom helyett kezdőbetű(k) szerint is csoportosíthatunk.



▶ Előzetes csoportbeállítás

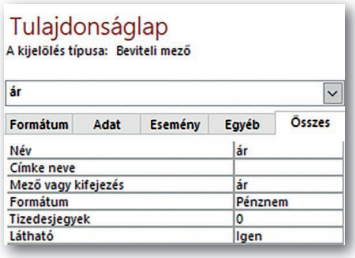


▶ Csoportosítási beállítások



▶ Összesítési beállítások

A (3), rendezési lépésben mód van arra is, hogy a *vezetéknév* és *utónév* szerinti rendezés beállításával együtt az adott csoportra vonatkozóan összesítést készítsünk. Ennél a jelentésnél az *ár* mező összegzését végezzük el.



Az elkészült jelentést az előző feladatnál látottak alapján módosíthatjuk úgy, hogy a mezők elhelyezése és a fejek beállítása megfeleljen a mintának. *Tervező nézetben* törölhetjük az összesítő sor felesleges elemeit is. Ezek után a jelentés egyedüli hiányossága, hogy az *ár* mező formailag eltér a mintától.

Tervező nézetben kattintsunk a *Törzs* részben az *ár* mezőre, majd jelenítsük meg a *Tulajdonságlapját,* azután állítsuk be a *Formátumot* és a *Tizedesjegyek* számát! Ugyanezt tegyük meg a *dátum láblécben* az összegző függvénnyel!

Ezt a feladatot úgy is megoldhattuk volna, hogy készítünk egy lekérdezést, amely az összes szükséges mező értékét tartalmazza, majd ezt felhasználva hozzuk létre a jelentést.

Ha a teljes adathalmaz megadott mezőinek kell a jelentésben szerepelniük, általában nincs szükségünk arra, hogy lekérdezést készítsünk. Ha nem minden adatot használunk fel, vagy meg kell jelenítenünk számított értéket is, akkor lekérdezéssel készítsük elő a jelentést!

- Készítsük el az első jelentést úgy, hogy csak az alsóvárosiak szerepeljenek!
- Készítsük el a második jelentést úgy, hogy csak október első hét napját vegyük figyelembe!

Feladatok

Készítsük el a következő jelentéseket a *javítás* adatbázist használva! Ha szükséges, lekérdezéssel készítsük elő a megoldást! A jelentéseket formailag magunk tervezzük meg!

a) Jelenítsük meg a felsővárosi lakosok nevét és e-mail-címét nevük kezdőbetűje szerinti csoportosításban! *(f14a)*

b) A munkavégzés települése és azon belüli helye szerinti csoportosításban jelenítsük meg a munka típusát és befejezésének dátumát! *(f14b)*

c) A munka típusa szerinti csoportosításban jelenítsük meg a munkavégzés helyét, dátumát, a munkaórák számát és az anyagköltséget! Csoportonként jelenjen meg a számértékek átlaga! Állítsuk be a pénznem formátumot az anyagköltségek megjelenítéséhez! *(f14c)*

Adatbevitel, űrlapok

Az adatbázis-kezelés gyakorlati részének elején létrehoztunk adatbázist, és vittünk be adatokat. Tisztáztuk, hogy nemcsak az rögzít adatokat, akinek ez a munkája, hanem az is, aki egyszerű használója egy ilyen rendszernek. Az adatbevitel során közvetlenül az adattáblába írtunk, de az egyrészt nem biztonságos, másrészt nem különösebben felhasználóbarát, elég csupán az idegen kulcs értékének bejegyzésére gondolnunk. Még kevésbé tarthatjuk megfelelőnek, hogy az egyszerű felhasználó, aki egy webáruházból rendel, ugyanilyen adattáblákat töltsön ki.

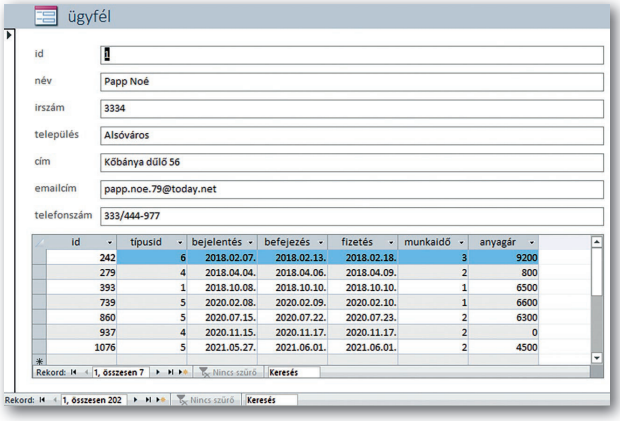
Azt a jól átlátható, könnyen kezelhető felületet, amelynek segítségével adatokat viheténk be, vagy adatokat kérdezhetünk le, **űrlapnak** nevezzük.

Az űrlapok készítését nem tekintjük mindenki számára szükséges tudásnak, használatukat viszont igen, ezért érdemes megismerni, mi van egy működő űrlap hátterében. Példáink a *javítás* adatbázishoz készültek.

34. példa: Űrlap generálása egy táblához

Készítsünk űrlapot az *ügyfél* táblához!

Az űrlapkészítés legegyszerűbb módja, hogy kiválasztjuk a megfelelő táblát, és rákattintunk a *Létrehozás* > *Űrlapok* > *Űrlap* elemre. Ekkor az Access automatikusan generál egy űrlapot, amelynek segítségével végiglépdelhetünk a tábla rekordjain, módosíthatjuk az aktuális rekordot, vagy éppen újat vehetünk fel. Mivel a *munka* táblában idegen kulcsként szerepel az ügyfél azonosítója, ezért segédűrlapként mindig megjelenik a *munka* tábla aktuális ügyfélhez tartozó része is.



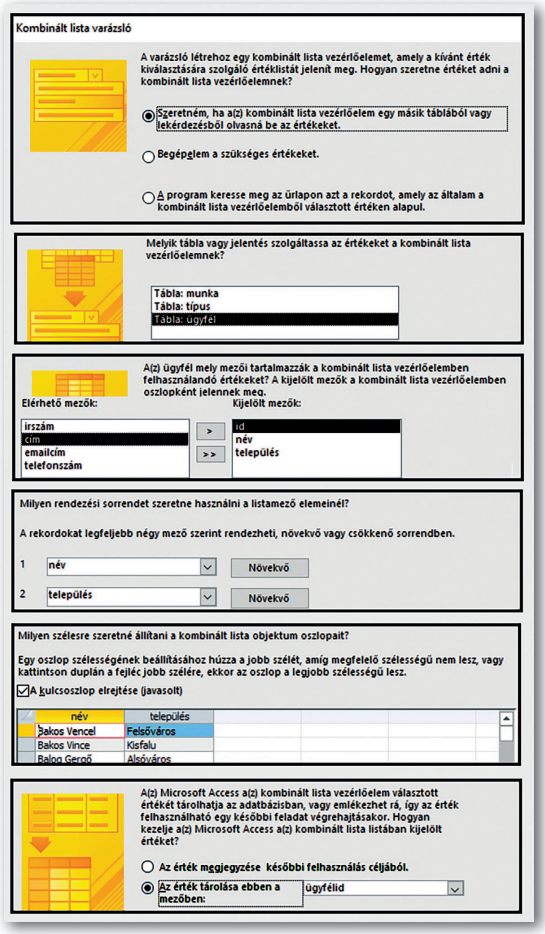
A kép alsó széléről leolvasható, hogy az *ügyfél* tábla első adatsorát látjuk a 202 rekordból. A táblázatos segédűrlap mutatja, hogy Papp Noé milyen munkákat végeztetett el. A munkának sajnos csak a *típusazonosítója* és nem a típusa szerepel, ezért nehezen használható. Az automatikusan generált űrlapoknál ilyen gyakran előfordul.

Nagyon fontos, hogy az adatbázisba ne kerüljön be hibás érték. Ezt természetesen nem lehet teljes mértékben megakadályozni, de elvárható, hogy az űrlap segítse a pontos adatrögzítést. Ennek első lépcsőfoka, hogy figyelmeztessen a nyilvánvalóan érvénytelen értékekre. Az ügyfél adatainál például az irányítószámnak 4 karakterből kell állnia. A képen látható *beviteli maszk*, az *érvényességi szabály* és az *érvényesítési*

szöveg erre vonatkozó beállítás. Ezt nemcsak az űrlap szintjén, hanem az adattábla mezőinél is be lehet állítani. Ilyen egyszerűen megfogalmazható feltétel a településnél nincs, ott egy, a létező települések nevét tartalmazó legördülő listával biztosíthatnánk, hogy csak létező település szerepeljen, és a név helyes legyen.

35. példa: Űrlap generálása segédűrlappal

Készítsünk űrlapot a *munka* táblához!



Az előző módszerrel előállított űrlapon már nem szerepel segédűrlap, és nem csupán a típus, hanem az ügyfél is azonosítóval található meg rajta. Ez az űrlaptartalom már a használhatóságot is megkérdőjelezi, mivel az ügyfeleket azonosító alapján ismerni lehetetlen. Felhasználói szemszögből az lenne az ideális, ha egy legördülő listából választhatnánk ki ezeket az értékeket. *Tervező nézetben* az *Űrlaptervezés* > *Vezérlők* > *Beviteli lista* elemére kattintva az űrlapon kattintással megadhatjuk a lista megjelenési helyét. A további lépéseket az ábrák mutatják.

Lényeges, hogy az *ügyfél* táblából dolgozunk, az ügyfélazonosító értékeként annak az *id* mezőjét fogjuk felhasználni. A megfelelő ügyfél kiválasztásához ismernünk kell a nevét és települését. Lehetséges, hogy még ez sem elég, ha az adatbázisban tárolt két Farkas Boglárka ugyanabban a városban lakik. A választást megkönnyíti, ha a listában ismert szempont szerint rendezzük az ügyfeleket.

Ha az ügyfelek beviteli listáját előállítottuk, akkor az *ügyfélazonosító* beviteli mezőre már nem lesz szükségünk, törölhető.

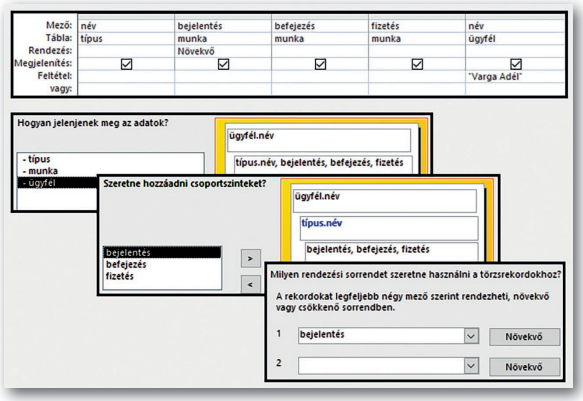
Ugyanezzel a módszerrel készíthetünk egy listát a típusazonosító helyett is, és akkor nem szükséges a munkafajták kódját ismerni az adatrögzítéshez.

36. példa: Űrlap segítségével generált jelentés

Készítsünk űrlapot, amellyel jelentést állíthatunk elő az űrlapon megadott személy által végeztetett munkákról!

Először ne is törődjünk azzal, hogy az űrlapon meg kell adnunk egy személyt! Készítsünk egy lekérdezést, amelybe írjuk be egy, az *ügyfél* táblában szereplő személy nevét, és listázzuk ki a végzett munka típusát és annak időadatait!

Az elkészült lekérdezésből készítsünk a lekérdezés minden mezőjét tartalmazó jelentést! A csoportosítást végezzük az ügyfelek, azon belül a munkatípus szerint, az adatok sorrendjét pedig a bejelentés dátuma határozza meg! Az elkészült jelentést a tanult módon formázhatjuk.

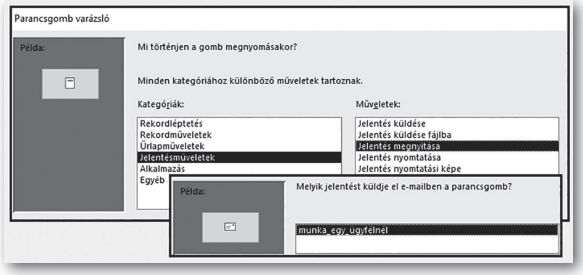


Ha a jelentést futtatjuk, akkor az automatikusan végrehajtja a lekérdezést, az abból kapott adatokból pedig elkészül a jelentés. Arra van szükségünk, hogy az űrlapról kezdeményezzük a jelentés futtatását, és a lekérdezés a feltételként szereplő nevet az űrlapból vegye át.

37. példa: Űrlap készítése *Tervező nézetben*

Készítsünk egy űrlapot *Tervező nézetben!* Az üres űrlapon helyezzünk el egy beviteli mezőt. A beviteli mező *Tulajdonságlapján* a nevét módosítsuk ügyfélnévre! Az űrlapon helyezzünk el egy gombot, amelyet az ábrán látható módon állítsunk be!

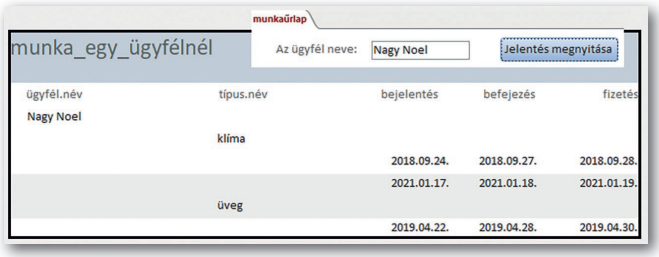
Ekkor a gombra kattintással megnyitja a képen látható jelentést. Ha kipróbáljuk, akkor még mindig nem az űrlapba írt névvel dolgozik, a futtatás eredményét az nem befolyásolja. Egyetlen feladatunk van: a lekérdezésben lecserélni a konkrét nevet az űrlap általunk elnevezett mezőjére.



Tervező nézetben töröljük ki a lekérdezésben szereplő nevet, majd a *helyi menü* > *Szerkesztés* > *Kifejezéselemek* > *javítás.accdb* > *Űrlapok* > *Betöltött űrlapok* > *munkaűrlap* > *ügyfélnév* választásával teremtsük meg a kapcsolatot az űrlappal!

Mező:	név	bejelentés	befejezés	fizetés	név
Tábla:	típus	munka	munka	munka	ügyfél
Rendezés:		Növekvő			
Megjelenítés:	☑	☑	☑	☑	☑
Feltétel:					[Űrlapok]![munkaűrlap]![ügyfélnév]
vagy:					

Ezután az űrlapba tetszőleges nevet bejegyezve megjeleníthetjük a kívánt adatokat. Fontos megemlíteni, hogy az elkészült megoldás csak mintát ad hasonló feladatok megoldásához, de nem tökéletes, hiszen ha két ügyfél neve egyezik, akkor az általuk végeztetett munkákat nem különíti el.



Feladatok

Készítsük el a következő űrlapokat az *étkezés* adatbázist használva! Ha szükséges, készítsünk lekérdezést! Az elkészített objektumokat formailag tetszés szerint tervezzük meg!

a) Készítsünk űrlapot, amely lehetőséget ad egy új ételmenü rögzítésére! Állítsuk be, hogy az *ár* csak 1000 és 2500 Ft közötti érték lehessen! *(f15a)*
b) Készítsünk űrlapot, amely egy új vendég rögzítésére alkalmas! Próbáljuk beállítani, hogy településként csak az eddig is rögzített két várost és két falut fogadja el! *(f15b)*
c) Készítsünk űrlapot, amely egy új ebédet rögzít! A vendéget és a menüt egy-egy legördülő listából válasszuk! *(f15c)*
d) Készítsünk űrlapot, amelyen szerepel egy *vezetéknév* nevű beviteli mező! Az űrlapon található gomb megnyomására fusson le egy lekérdezés, amely az űrlapba írt vezetéknevű személyek adatait listázza ki! *(f15d)*

Ami a választó lekérdezésen túl van

Ha adatbázisokkal dolgozunk, bármilyen felületen használjuk is őket, szinte mindig adatokat kérdezünk le. Egyesek feltehetik a kérdést: mi mást lehetne? Szó esett már arról, hogy az adattáblákat új rekordokkal bővíthetjük. Arról nem beszéltünk, hogy módosíthatunk az adatbázison, és törölhetünk is belőle. Ezeket a műveleteket **adatmanipulációs utasításoknak** nevezzük.

Próbáljuk ki az adatmanipulációs utasításokat a *javítás* adatbázist használva!

Törlő lekérdezés

Fontos tudni, hogy törlő lekérdezésnél általában csak egy táblából törölhetünk adatot. (Ha a kaszkádolt törlést beállítottuk a kapcsolat megadásakor, akkor azok a rekordok is eltűnnek a többi táblából, ahol az azonosító értéke idegen kulcsként szerepel.) Ha a tábla kapcsolatainál a hivatkozási integritás be van kapcsolva, akkor az a rekord nem törölhető, amelyre más táblából idegen kulcs mutat. A törlés mindig a teljes rekordot eltávolítja.

38. példa: Rekordok törlése

- Nagy Alex felsővárosi lakos tévesen került az adatbázisba. Soha egyetlen problémát nem jelentett be. Adatainak törlését kéri az *ügyfél* táblából. Készítsünk lekérdezést, amely eltávolítja az ő adatait tartalmazó rekordot! *(Alex)*
- Az adatvédelmi szabályozás miatt – bizonyos feltételek teljesülése esetén – lehet kérni a velünk kapcsolatban keletkezett adatok törlését. Töröljük a *munka* táblából a Katona Lara által rendelt munkákat! *(Lara)*
- Az *ügyfél* táblából töröljük azokat a személyeket, akiknél még nem végeztek munkát! *(nemszerepel)*

Azt javasoljuk, hogy törlő lekérdezés esetén először vegyük fel a szükséges táblákat, mintha adatlekérdezést szeretnénk készíteni. Listázzuk ki a feltételnek megfelelő rekordokat, nézzük meg az érintett rekordok körét! Ha meggyőződtünk arról, hogy valóban azok szerepelnek, amelyeket törölni akarunk, akkor állítsuk át a lekérdezés típusát törlőre, és futtassuk le! Fontos az óvatosság, mert ez a művelet nem vonható vissza.

Mező:	ügyfél.*	név
Tábla:	ügyfél	ügyfél
Törlés:	From	Where
Feltétel:		"Nagy Alex"
vagy:		

Az *Alex* lekérdezés egyetlen táblát, az *ügyfél* táblát érinti. A lekérdezőrácson a *Törlés* jelenti az újdonságot a választó lekérdezéshez képest. A FROM segítségével adjuk meg a táblát, amelyből törlünk, a WHERE segítségével pedig a feltételt!

Az *Alex* lekérdezés SQL-nyelvű megfelelője:

```
DELETE * FROM ügyfél WHERE név="Nagy Alex";
```

A *Lara* lekérdezésben már két tábla szerepel. A követendő módszer most is ugyanaz. A *munka* táblánál beállítjuk a FROM-ot, az *ügyfél* táblában fellelhető névre a WHERE-t, megadjuk a feltételt, és készen is vagyunk.

A *Lara* lekérdezés SQL-nyelvű megfelelője:

```
DELETE munka.* FROM ügyfél INNER JOIN munka ON ügyfél.id =
  munka.ügyfélid WHERE ügyfél.név="Katona Lara";
```

A *nemszerepel* lekérdezés a lekérdezőrácson nem kattintgatható ki. Az ilyen lekérdezések SQL-nyelven allekérdezést használva elkészíthetők. Ekkor a főlekérdezésben csak az a tábla szerepel, amelyből törölni szeretnénk – a feltételben a kapcsolatot megadó mezőre szűrünk.

A *nemszerepel* lekérdezés SQL-nyelvű megfelelője:

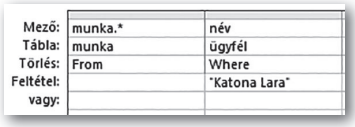
```
DELETE *
FROM ügyfél
WHERE id NOT IN (SELECT ügyfélid FROM munka);
```

Frissítő lekérdezés

Frissítő lekérdezésnél egy tábla néhány mezőjének értékét módosítjuk. Ha a módosítani kívánt mező egy másik táblában idegen kulcsként szerepel, akkor nem változtatható meg az értéke, amennyiben a kapcsolatnál a hivatkozási integritás be van kapcsolva. Ha a kaszkádolt frissítés be van állítva a kapcsolatnál, ez az érték a másik táblában is megváltozik.

39. példa: Adatok módosítása

- Egy rendelettel megszüntették a 3333-as irányítószámot, helyette a 3330-ast kell használni. Végezzük el ezt a módosítást! *(irszám)*
- Szintén a szabályozás változása miatt a *munka* táblában az adóval növelt árat kell feltüntetni az *anyagár* oszlopban a klímával kapcsolatos szereléseknél. Végezzük el ezt a módosítást! *(klíma)*

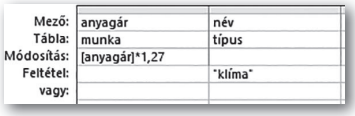


A frissítő lekérdezésnél is követhetjük a törlő lekérdezésnél leírt tanácsot: először választó lekérdezéssel ellenőrizzük az érintett rekordokat. Az ellenőrzés után alakítsuk frissítő lekérdezéssé a választót, majd futtassuk le! A megfelelő körültekintés azért fontos, mert ez a művelet sem vonható vissza.

Az *irszám* lekérdezés egyetlen táblát érint. A rácson a *Módosítás* sor jelenti az újdonságot, oda a változtatás értékét vagy képletét kell beírnunk.

Az *irszám* lekérdezés SQL-nyelvű megfelelője:

```
UPDATE ügyfél SET irszám =”3330” WHERE irszám=”3333”;
```



A *klíma* lekérdezés kéttáblás. Az anyagár mező új értékét egy képlettel tudjuk meghatározni, a korábbi érték 1,27-szorosával. Természetesen nemcsak az adott tábla, hanem akár más táblák mezőit is felhasználhatjuk a feltételben.

A *klíma* lekérdezés SQL-nyelvű megfelelője:

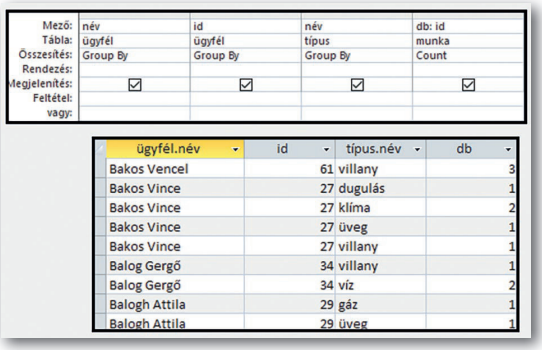
```
UPDATE típus INNER JOIN munka ON típus.id = munka.típusid SET
  munka.anyagár = [anyagár]*1.27 WHERE típus.név=”klíma”;
```

Keresztáblás lekérdezés

A keresztáblás lekérdezés nem része a klasszikus adatbázis-kezelési eszközöknek, sokkal inkább kimutatáskészítési eszköz, amely az Access program által is biztosított.

40. példa: Kétdimenziós kimutatás

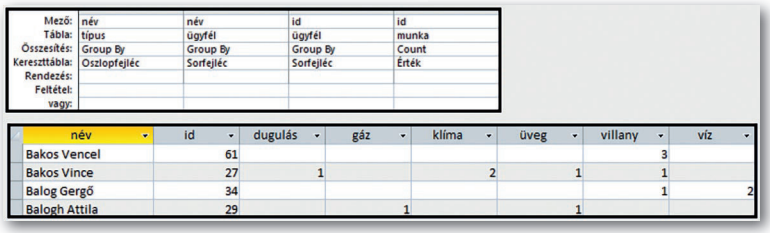
Határozzuk meg, hogy melyik ügyfél hány alkalommal végeztette el az egyes munkákat! *(stat)*



Mező:	név	id	név	db: id
Tábla:	ügyfél	ügyfél	típus	munka
Összesítés:	Group By	Group By	Group By	Count
Rendezés:				
Megjelenítés:	☑	☑	☑	☑
Feltétel:				
vagy:				

ügyfél.név	id	típus.név	db
Bakos Vencel	61	villany	3
Bakos Vince	27	dugulás	1
Bakos Vince	27	klíma	2
Bakos Vince	27	üveg	1
Bakos Vince	27	villany	1
Balog Gergő	34	villany	1
Balog Gergő	34	víz	2
Balogh Attila	29	gáz	1
Balogh Attila	29	üveg	1

Ezt a feladatot egyszerű lekérdezéssel is megoldhatjuk. Csoportosítanunk kell az ügyfelek és a munkatípusok szerint, majd megszámláltatni az elvégzett munkákat. Az ügyfelenkénti számláláshoz mindenképpen szükségünk van az ügyfél azonosítójára – vagy pontosabb címadataira –, különben az azonos nevű személyeket egyként kezelné.



Mező:	név	név	id	id
Tábla:	típus	ügyfél	ügyfél	munka
Összesítés:	Group By	Group By	Group By	Count
Kereszttábla:	Oszlopfejléc	Sorfejléc	Sorfejléc	Érték
Rendezés:				
Feltétel:				
vagy:				

név	id	dugulás	gáz	klíma	üveg	villany	víz
Bakos Vencel	61					3	
Bakos Vince	27	1		2	1	1	
Balog Gergő	34					1	2
Balogh Attila	29		1		1		

A kapott lista nehezen áttekinthető, hosszabb ideig tart meghatározni, hogy kik hányféle munkát végeztettek, melyik munkából kinél fordult elő több is.

Ennek a problémának áttekinthető megoldását kapjuk kereszttáblás lekérdezéssel. A végeredmény természetesen ugyanaz, de a meghatározott értékeket egy kétdimenziós táblázatban helyezi el. Ha az előző választó lekérdezést átalakítjuk – a lekérdezés típusát megváltoztatva – kereszttáblás lekérdezéssé, akkor a táblázat sor- és oszlopfejlécét kell megadnunk. Az adatok elemszáma alapján a sorok fejlécébe az ügyfelek neve kerüljön, az oszlopok fejlécébe pedig a munkák típusa. A lekérdezés futtatásának eredménye az ábráról leolvasható.

Feladatok

Készítsük el a következő lekérdezéseket az *étkezés* adatbázist használva!

a) Készítsünk lekérdezést, amely a menük árát 10 százalékkal megemeli! *(f16a)*
b) Egészítsük ki az *ebéd* táblát egy *eltöltött* nevű, idő típusú mezővel! Készítsünk lekérdezést, amely az *eltöltött* mező értékét kitölti az *érkezés* és *távozás* mezőben tárolt adatok alapján! *(f16b)*
c) Készítsünk lekérdezést, amely a nagyfalui Kőrös utcát Tisza utcára cseréli! *(f16c)*
d) Készítsünk lekérdezést, amely kitörli a Fekete Kamilla által fogyasztott ebédek adatait! *(f16d)*
e) Készítsünk lekérdezést, amely kitörli Fekete Kamilla adatait! (f16e)
f) Készítsünk kereszttáblás lekérdezést, amely megmutatja, hogy az egyes vendégek külön-külön hányszor választották az egyes menüket! *(f16f)*

Vegyes feladatok

Készítsük el a következő lekérdezéseket a *város* adatbázist használva!

a) Soroljuk fel ábécérendben a Heves megyei városokat! *(f17a)*
b) Listázzuk ki azokat a városokat, melyek nevében van *e* betű! *(f17b)*
c) Adjuk meg azokat a városokat, melyek nevében nincs *e* betű! *(f17c)*
d) Határozzuk meg azokat a városokat, melyek nevében a magánhangzók közül csak *e* betű van! *(f17d)*
e) Adjuk meg, melyik megyében található a legkisebb területű város! *(f17e)*
f) Listázzuk ki azokat a megyei jogú városokat, amelyek nem megyeszékhelyek! *(f17f)*
g) Határozzuk meg, hogy hány város népesebb 75 ezer főnél! *(f17g)*
h) Adjuk meg, hogy hány fő lakik a legkisebb és a legnagyobb lélekszámú Fejér megyei városokban! *(f17h)*
i) Készítsünk lekérdezést, amely megadja a megyeszékhelyek átlagos lélekszámát! *(f17i)*
j) Adjuk meg azokat a megyéket, amelyekben több megyei jogú város is található! *(f17j)*
k) Listázzuk ki az Egernél népesebb városokat! *(f17k)*
l) Melyik megyében van ugyanannyi város, mint Hevesben? *(f17l)*
m) Soroljuk fel annak a megyének a városait, amelyben a legkevesebb város található! *(f17m)*