## Adatbázis fogalma

Adatbázison köznapi értelemben valamely rendezett, valamilyen szisztéma szerint tárolt adatokat értünk, melyek nem feltétlenül számítógépen kerülnek tárolásra.

Képzeljük el, hogy egy céghez naponta átlagban 20 levél érkezik. A cég irattárosa kellő adattárolási tapasztalat híján a leveleket az irattár ajtajára vágott lyukon keresztül bedobja. Elképzelhető, hogy pár év eltelte után milyen reménytelen vállalkozás egy levelet megtalálni az irattárban.

Ez az adathalmaz nem tekinthető adatbázisnak, ahhoz hogy adatbázis legyen nem elegendő a nagyszámú adat. Az adathalmaz csak akkor válik adatbázissá, ha az valamilyen rend szerint épül fel, mely lehetővé teszi az adatok értelmes kezelését. Természetesen ugyanazon adathalmazból többféle rendszerezés alapján alakíthatunk ki adatbázist.

Pl.: egy könyvtárban a könyveket rendezhetnénk a könyvek mérete vagy akár a szerző vagy szerzők testsúlya alapján. Ez már egy rendszert ad az adatok tárolásához. Íly módon minden könyv helye meghatározott. De bizonyára nehéz helyzetben lennénk, ha szerző és cím alapján próbálnánk meg előkeresni egy könyvet. Az adatok tárolásába bevitt rendszernek alkalmasnak kell lennie a leggyakrabban előforduló igények hatékony kielégítésére. Az adatbázisok mellé egy adatbáziskezelő rendszer (DBMS) is járul, mely az adatbázis vagy adatbázisok üzemeltetését biztosítja. Hagyományos adatbázis esetén ez a kezelő személyzet intelligenciájának része, elektronikus adatbázisok esetén pedig valamilyen szoftver.

## 1.2. Történelmi áttekintés

Azóta rendelkezünk adatbázisokkal, mióta írásban vagyunk képesek rögzíteni adatokat. Ez az ókorban történhetett akár kőtáblákra vagy papirusz tekercsekre. Az adatbázisok fejlettebb formái később a kartoték rendszerek lettek, melyek a számítógépek megjelenéséig az alapvető adatbázis rendszerek voltak.

60’ években CODASYL (**CO**nference on **DA**ta **SY**stem **L**anguages) konferenciák:
1968 oktoberében megalakult a DBTG (**D**ata**B**ase **T**est **G**roup), és kiadásra került 1971 áprilisában a DBTG REPORT.

DBTG REPORT:
Mik a gondok!
Az új adatkezelés alapelvei.
Módszertani javaslat az elvek érvényesítésére.

**Programozási nyelvek generációi:**

| | | |
|---|---|---|
| 1. generációs nyelvek | Gépi kódú programozás | 8C C0 8E D8 |
| 2. generációs nyelvek | Assembler nyelvek | MOV AX,ES<br>MOV DS,AX |
| 3. generációs nyelvek | Imperatív nyelvek: Algol, FORTRAN, C, Pascal, Basic | var i, s:integer;<br>s := 0;<br>for i:=1 to 10 do s := s + i; |
| 4. generációs nyelvek | | |
| 5. generációs nyelvek | Logikai programozási nyelvek: Prolog, Lisp | (+ (cadr p1) (cadr p2) (/ 750 2)) |
| Objektum orientált nyelvek | SmallTalk, C++, Java, Visual Basic | class Complex {<br>protected:<br>double real, img;<br>public:<br>Complex();<br>~Complex();<br>Complex& operator+(Complex& a);<br>} |

**Adatbáziskezelők és célja**

Manapság nem elégszünk meg egy adatbázissal, mely az adatokat rendszerezve tárolja, hanem az adatok kezeléséhez szükséges eszközöket is az adatbázis mellé képzeljük. Az így kialakult program rendszert adatbázis kezelő rendszernek (DBMS Database Management System) nevezzük. Egy DBMS egyszerűbb és gyorsabb megoldást kínál az űrlapokon alapuló alkalmazások kidolgozásában, az adatbázis adatokon alapuló jelentések készítésében. A DBMS-ek megváltoztatták a végfelhasználók adatnyerési lehetőségeit az egyszerű lekérdezési nyelvek bevezetésével.

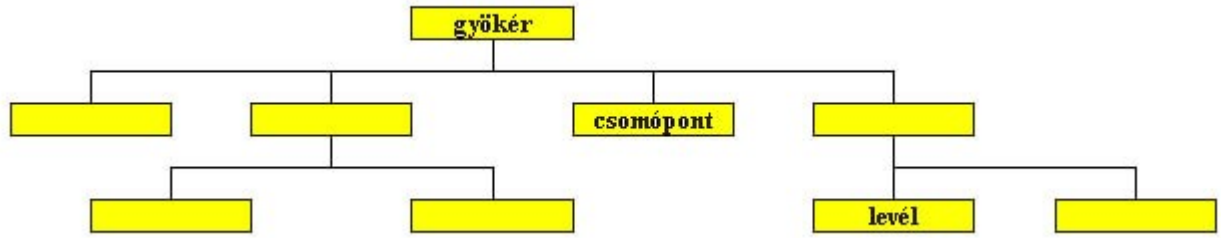
Az adatbáziskezelők az operációs rendszerekhez hasonlóan számos eszközt szolgáltatnak a gyakran előforduló problémák megoldására, de míg az operációs rendszerek a perifériák kezelésére és a fájlok használatára adnak lehetőséget, addig a DBMS-ek eszközei egy magasabb szintű absztrakcióra épülve az adatok logikai szintű elérését támogatják a rekordokon belüli bájt poziciók helyett, továbbá eszközöket tartalmaznak az adatok űrlap szintű kezelésére illetve jelentések és menük készítésére.

**Különböző adatbázis modellek**

Az adatbáziskezelők fejlődése során többfajta logikai modell alakult ki, melyek főként az adatok közötti kapcsolatok tárolásában térnek el egymástól. A három alapvető modell a **hierarchikus**, a **háló** és a **relációs modell**. Ezek közül manapság a DOS/Windows 3.x/Windows 95/NT illetve UNIX operációs rendszerekben kizárólag a relációs modellre épülő adatbáziskezelőket használnak.

### 1.4.1. Hierarchikus adatbázis modell

A hierarchikus modell volt a legelső az adatbáziskezelőkben és egyben a leginkább korlátozott. Például az IBM IMS adatbáziskezelő rendszer alkalmazta ezt a modellt. A neve is utal rá, hogy az adatokat egy hierarchiában kell elrendezni. Ezt egy fa szerkezettel tehetjük szemléletessé.




*Hierarchikus adatmodell*

Az adatbázis több egymástól független fából állhat. A fa csomópontjaiban és leveleiben helyezkednek el az adatok. A közöttük levő kapcsolat, szülő gyermek kapcsolatnak felel meg. Így csak 1:n típusú kapcsolatok képezhetők le segítségével. Az 1:n kapcsolat azt jelenti, hogy az adatszerkezet egyik típusú adata a hierarchiában alatta elhelyezkedő egy vagy több más adattal áll kapcsolatban.

A hierarchikus modell természetéből adódóan nem ábrázolhatunk benne n:m típusú kapcsolatokat (lásd a háló modellt). Emellett további hátránya, hogy az adatok elérése csak egyféle sorrendben lehetséges, a tárolt hierarchiának megfelelő sorrendben.

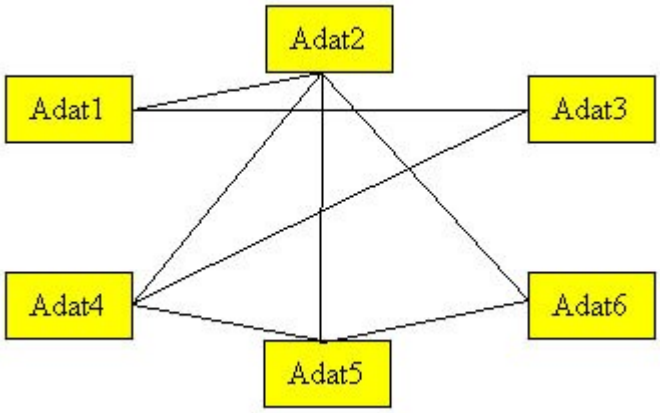
A hierarchikus adatmodell alkalmazására a legkézenfekvőbb példa a családfa. De a főnök-beosztott viszonyok vagy egy iskola szerkezete is leírható ebben a modellben. Az iskola esetén többféle hierarchia is felépíthető. Egyrészt az iskola több osztályra bomlik és az osztályok tanulókból állnak. Másrészt az iskolát az igazgató vezeti, a többi tanár az ő beosztottja és a tanárok egy vagy több tantárgyat tanítanak.


*Iskola hierarchikus felépítése a diákok szemszögéből*


*Iskola hierarchikus felépítése a tanárok szemszögéből*

**Hálós adatbázis modell**

A hálós adatmodell esetén az egyes azonos vagy különböző összetételű adategységek (rekordok) között a kapcsolat egy gráffal írható le. A gráf csomópontok és ezeket összekötő élek rendszere, melyben tetszőleges két csomópont között akkor van adatkapcsolat, ha őket él köti össze egymással. Egy csomópontból tetszőleges számú él indulhat ki, de egy él csak két csomópontot köthet össze. Azaz minden adategység tetszőleges más adategységekkel lehet kapcsolatban. ebben a modellben n:m típusú adatkapcsolatok is leírhatók az 1:n típusúak mellett. A hierarchikus és a hálós modell esetén az adatbázisba fixen beépített kapcsolatok következtében csak a tárolt kapcsolatok segítségével bejárható adat-visszakeresések oldhatók meg hatékonyan (sok esetben hatékonyabban mint más modellekben). További hátrányuk, hogy szerkezetük merev, módosításuk nehézkes.




*Hálós adatmodell*

Az iskolai példánál maradva az egyes diákok illetve tanárok közötti kapcsolat hálós modellben írható le. Minden diákot több tanár tanít és minden tanár több diákot tanít.

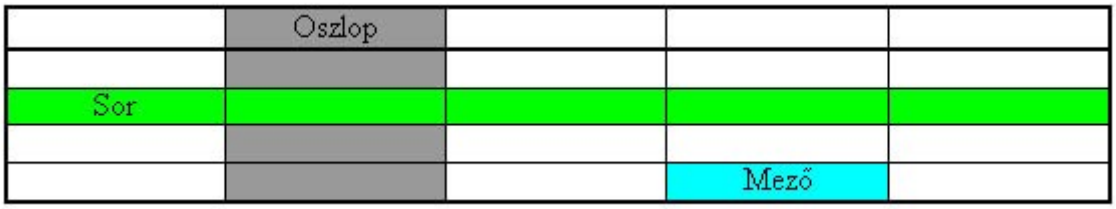
**Relációs adatbázis modell**

A relációs az egyik legáttekinthetőbb és a 80-as évektől kezdve a legelterjedtebb adatmodell. Ebben a modellben az adatokat táblázatok soraiban képezzük le. A legfontosabb eltérés az előzőekben bemutatott két modellhez képest az, hogy itt nincsenek előre definiált kapcsolatok az egyes adategységek között, hanem a kapcsolatok létrehozásához szükséges adatokat tároljuk többszörösen. Ezzel egy sokkal rugalmasabb és általánosabb szerkezetet kapunk.

A relációs adatmodell kidolgozása Codd nevéhez fűződik (1971). Azóta fontos szerepet játszik az adatbáziskezelők alkalmazásában. A relációs modell előnyei a következők:

- A relációs adatszerkezet egyszerűen értelmezhető a felhasználók és az alkalmazás készítők számára is, így ez lehet közöttük a kommunikáció eszköze.
- A logikai adatmodell relációi egy relációs adatbáziskezelő rendszerbe módosítások nélkül átvihetők.
- A relációs modellben az adatbázistervezés a normál formák bevezetésével egzakt módon elvégezhető

A reláció nem más mint egy táblázat, a táblázat soraiban tárolt adatokkal együtt. A relációs adatbázis pedig relációk és **csak** relációk összessége. Az egyes relációkat egyedi névvel látjuk el. A relációk oszlopaiban azonos mennyiségre vonatkozó adatok jelennek meg. Az oszlopok névvel rendelkeznek, melyeknek a reláción belül egyedieknek kell lenniük, de más relációk tartalmazhatnak azonos nevű oszlopokat. A reláció soraiban tároljuk a logikailag összetartozó adatokat. A reláció sorainak sorrendje közömbös, de nem tartalmazhat két azonos adatokkal kitöltött sort. Egy sor és oszlop metszésében található táblázat elemet mezőnek nevezzük, a mezők tartalmazzák az adatokat. A mezőkben oszloponként különböző típusú (numerikus, szöveges stb.) mennyiségek tárolhatók. A reláció helyett sokszor a tábla vagy táblázat, a sor helyett a rekord, az oszlop helyett pedig az attribútum elnevezés is használatos.




*Relációk elemei*

A relációs adatbázist általában egy kiszolgáló, egy *adatbázis motor* teszi elérhetővé a felhasználók számára. Az adatbázis motor képes a kérések párhuzamos kezelésére, a naplózásra, a hibák észlelésére. Kritikus hiba esetén azonnal leáll, hogy az adatok helyessége ne sérüljön.

Az adatbázis motor általában egy külön kiszolgáló számítógépen fut, de ez nem szükségszerű: futhat azon a gépen is, ahol a felhasználó dolgozik. Kisebb hálózatok esetén szokásos egy erősebb munkaállomásra telepíteni az adatbázis motort.

A fejlettebb adatbázis motorok biztosítják a tranzakciókezelést, amely óvja az adatok épségét (integritását). Ha egy felhasználó egy műveletsort nem tud befejezni (például programhiba

miatt), akkor a műveletsort (tranzakciót) vissza lehet görgetni (rollback) a kezdőponthoz. Ha a műveletsor sikeres volt, akkor pedig jóvá kell hagyni azt (commit).

## Osztott adatbázis

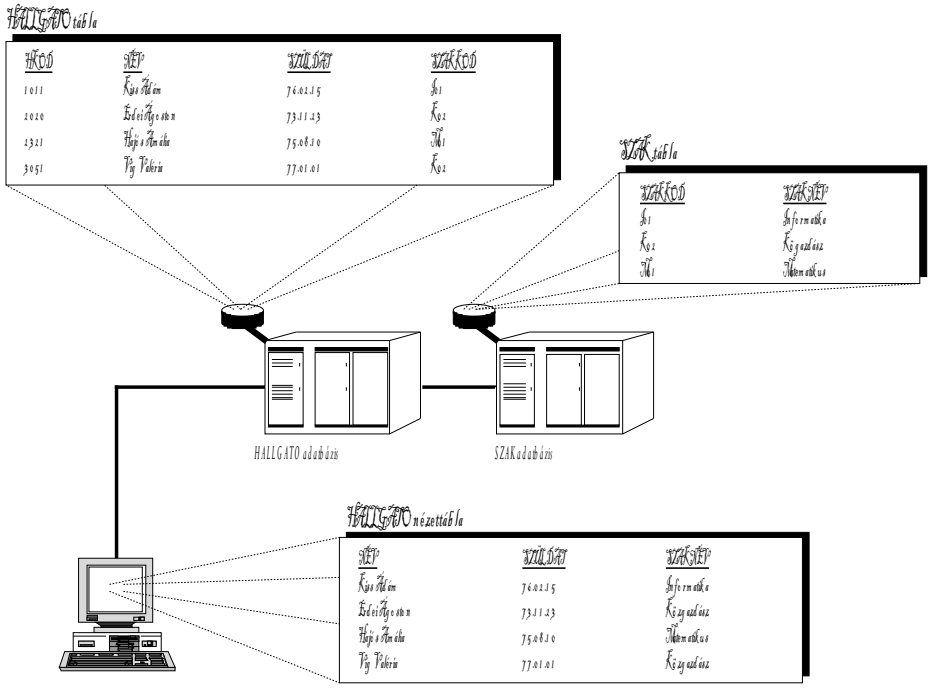
A számítógép-hálózatok kialakulásával felmerült az igény arra, hogy fizikailag különböző helyeken tárolt adatokat együtt használjanak. Így jöttek létre az osztott adatbázisok.
**Az osztott adatbázis különböző adatbázis-szerverek által vezérelt adatbázisok hálózata, amely a felhasználó számára egy logikai adatbázisként jelenik meg.** A fizikailag különböző helyeken található adatokat a felhasználók logikailag összekapcsolhatják a különböző lekérdezések során. Egy lekérdezés eredményeként kapott tábla alapulhat fizikailag különböző adatbázisokban levő adatokon is.

**Az előnyök:**

1. A kommunikációs költségek már említett csökkenése.
2. Mindenki a számára ismerős adatokat gondozza.
3. Egy-egy csomópont kiesése esetén a többi adatai továbbra is elérhetőek.
4. Lehetséges a moduláris tervezés, a rugalmas konfigurálás.
5. Hosszabb idő alatt a rendszer gépei akár ki is cserélhetők.

**A hátrányok:**

1. A rendszer bonyolultabb és sebezhetőbb lesz,
2. Nem könnyű minden csomópontra egyformán jó személyzetet találni, másrészt, ha találunk fenyeget a szuboptimalizáció veszélye. **Szuboptimalizáció** : Az alrendszerek célkitűzéseit a rendszer általános célkitűzéseinek kárára valósítják meg.
3. Mindig valamennyi gépnek működnie kell.
4. Többféle hardvert és szoftvert kell a rendszernek kezelnie és "összehoznia".
5. Bonyolult a jogosultságok ellenőrzése (a jogosultságokat leíró táblázatokat hol tároljuk: egy csomópontban, vagy mindenütt?).




Azokat a számítógépeket, amelyek az osztott adatbázisban található adatokhoz hozzáférhetnek, ***node***-oknak nevezzük. Azt az adatbázist, amelyhez a felhasználó közvetlenül hozzákapcsolódik, ***lokális adatbázis***nak hívjuk. Az ezen felhasználó által elérhető többi adatbázist pedig *távoli (**remote**) adatbázis*nak nevezzük. Amikor egy lokális adatbázis egy távoli adatbázis adataival dolgozik, akkor a lokális adatbázis a kliens, a távoli adatbázis pedig a távoli server (remote-server). Egy node lehet server, kliens, vagy mindkettő.

Az osztott adatbázisok kezelésének egyik legnagyobb problémája a fizikailag különböző adatbázisokban végzett egyidejű, összefüggő módosítások kezelése. Ugyanis előfordulhat, hogy a módosítás sikeresen lezajlik az első adatbázisban, azonban hiba történik a második adatbázisbeli módosítás esetén. Ekkor az első adatbázisba a módosítások belekerültek, míg a másodikba nem. Így az adatok integritása már nem áll fenn. Ezért olyan megoldásra van szükség, amely garantálja, hogy az összefüggő módosítások minden adatbázisban megtörténjenek.

Az osztott adatbázisok tipikus példái a banki rendszerek. Egy banknak számos városban lehet fiókja. Minden bankfiók a saját adatbázisában tartja nyilván az ügyfeleinek adatait. Általában a bankfiókban van egy központi számítógép, amelyen ez az adatbázis található, ehhez kapcsolódhat a bankfiókban található többi számítógép. Ha a bankfiókok között nincs számítógépes kapcsolat, akkor két különböző fiókban vezetett számla közötti átutalást úgy kell kezelni, hogy az egyik bankfiókban a számláról leemelik az összeget, ezt a saját adatbázisukban átvezetik. Értesítik a másik bankfiókot (pl. postai úton), hogy az összeget írja

jóvá a másik számlán. Ezután a másik bankfiókban is megtörténik az adatbázisban az adatmódosítás. Mivel a pénzügyi tranzakció rögzítése kétszer történik meg, előfordulhat hiba (pl. a két összeg eltér egymástól), továbbá időbe telik, míg a leemelt összeg a másik számlán megjelenik.

Ha a bankfiókokban található számítógépeket illetve hálózatokat összekapcsolják egy számítógépes hálózatba, akkor lehetőség nyílik arra, hogy a fizikailag különböző városokban található adatbázisokat logikailag összekapcsolják egyetlen adatbázissá. Így a különböző helyeken vezetett számlák között történő tranzakciók könnyen kezelhetővé válnak. Egyetlen bankfiókban elegendő rögzíteni a tranzakciót, a módosítás meg fog történni mind a két adatbázisban.

Ebben a példában a bankfiókokban található adatbázisok együttesen osztott adatbázist alkotnak. Egy bankfiókban található adatbázis annak a bankfióknak a lokális adatbázisa, míg a többi bankfiókban levő adatbázis ennek a bankfióknak távoli adatbázis.

# Relációs adatbázis-kezelő rendszerek

## A relációs algebra alapfogalmai

A relációs modell felépítésében annak strukturális és integritási komponensei vesznek részt, e komponensekből áll össze a relációs adatstruktúra. Az itt megadott szabályok határozzák meg, hogy hogyan nézhet ki egy adott adatbázis. Ezek a komponensek az adatbázis bármely időpontbeli állapotaira vonatkoznak, időtől függetlenek, ezért statikus elemeknek nevezzük őket.

Az adatbázisok azonban nem holt, befagyott rendszerek, struktúrák. Az adatbázis értelme, hogy használják azt. A felhasználás során a felhasználók lekérdezhetik, vagy módosíthatják az adatbázis tartalmát. Az adatbázis akkor lesz tehát élő, ha csatlakozik hozzá egy olyan funkciócsoport is, amely lehetővé teszi az adatbázisban tárolt adatok lekérdezését és módosítását. Ezeket a komponenseket – mivel az adatbázis változásaihoz, megváltoztatásához kapcsolódnak – dinamikus komponenseknek nevezzük. A relációs adatmodell műveleti része ezeket a dinamikus adatkezelő és adatlekérdező lehetőségeket foglalja magába.

A relációkból elméletileg igen sokféleképpen, többféle mechanizmussal lehet adatokat kiolvasni. Az itt ismertetésre kerülő műveletcsoportot összefoglalóan **relációs algebrának** nevezik. A relációs algebra az **alapja** a ma már szabványként elfogadott és leginkább elterjed adatlekérdező relációs parancsnyelvnek, az **SQL-nek** is.

A **relációs adatmodell műveleti része a relációkon alapul**, vagyis minden művelet a relációkon értelmezett, bemenő operandusai csak relációk lehetnek. A relációs algebra egyszerű, de hatékony módszereket ad a kezünkbe ahhoz, hogy a meglévő relációkból új relációkat hozzunk létre.

A relációs **műveletek csoportjai**:

- hagyományos halmazműveletek – például az egyesítés, metszet, és különbség,

- műveletek, amelyek a reláció bizonyos részeit elhagyják – például a kiválasztás, vetítés,
- műveletek, amelyek két reláció sorait kombinálják – például a keresztszorzat és az összekapcsolás,
- egy művelet, ami nem befolyásolja a reláció sorait, de módosítja a reláció sémáját, vagyis az attribútumok és a reláció nevét – ez az átnevezés.

E műveletek nem elegendőek a relációkon elvégzendő összes számításhoz, ugyanis nagyon korlátozott lehetőséget biztosítanak, mégis jelentős részét tartalmazzák azoknak a műveleteknek, amelyeket az adatbázisokkal tenni akarunk.

A műveletek egyik része **adatkezelésre** (felvitel, módosítás, törlés), a másik része pedig **adatlekérdezésre** (adatok különböző szempontok szerinti megjelenítésére) használható.

## Relációkon végezhető műveletek

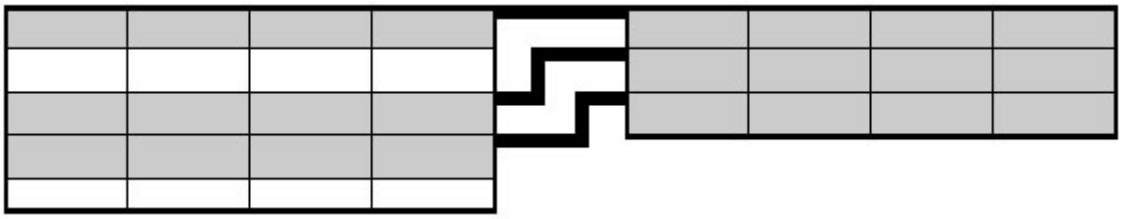
Tekintsük át a relációs algebra körébe tartozó **műveleteket**:
- átnevezés (egyoperandusú),
- szelekció (egyoperandusú),
- projekció (egyoperandusú),
- összekapcsolás (kétoperandusú),
- keresztszorzat (kétoperandusú),
- unió (kétoperandusú),
- metszet (kétoperandusú),
- különbség (kétoperandusú),
- osztás (kétoperandusú),
- kiterjesztés,
- csoportosítás.

### Relációs algebra műveletei

A relációkkal kapcsolatban a matematika egy külön ága fejlődött ki. A matematikusok műveleteket definiáltak a relációkra és a halmaz műveleteket is alkalmazták a relációkra. Röviden ismertetjük ezeket a műveleteket.

### Szelekció

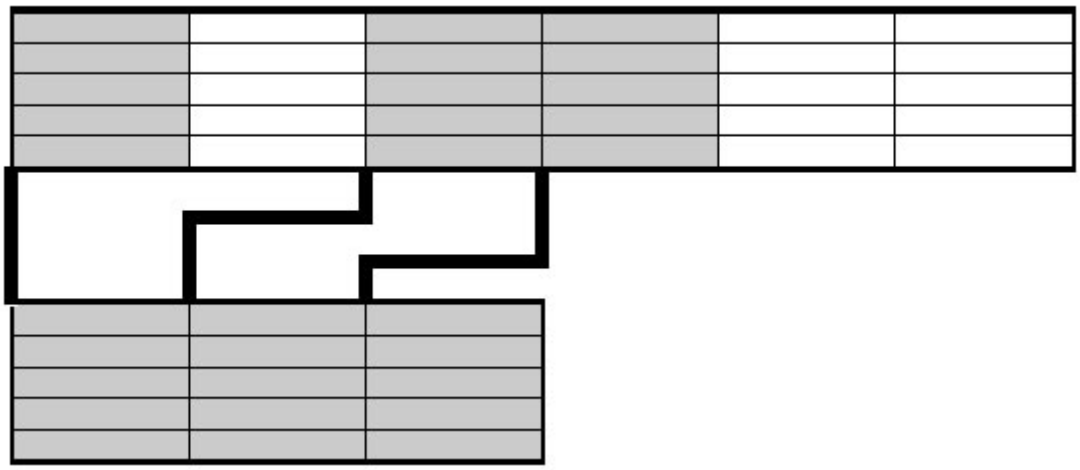
A szelekció művelete során egy relációból csak egy adott feltételt kielégítő sorokat őrizzük meg az eredmény relációban.



Szelekció, horizontális megszorítás

**Projekció**

A projekció során egy reláció oszlopai közül csak bizonyosakat őrzünk meg az eredmény relációban.



Projekció, vertikális megszorítás

**Keresztszorzat szorzat**

A Descartes szorzat két reláció sorait minden kombinációban egymás mellé teszi az eredmény relációban.

| | | |
|---|---|---|
| *1* | | |
| *2* | | |
| *3* | | |
| *4* | | |

| | | | |
|---|---|---|---|
| **1** | | | |
| **2** | | | |
| **3** | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| *1* | | | **1** | | | |
| *2* | | | **2** | | | |
| *3* | | | **3** | | | |
| *4* | | | **1** | | | |
| *1* | | | **2** | | | |
| *2* | | | **3** | | | |
| *3* | | | **1** | | | |
| *4* | | | **2** | | | |
| *1* | | | **3** | | | |
| *2* | | | **1** | | | |
| *3* | | | **2** | | | |
| *4* | | | **3** | | | |

Descartes szorzat

**Összekapcsolás**

Az összekapcsolás művelete két vagy több relációt kapcsol össze egy-egy attribútum érték összehasonlításával. Az összekapcsolás leggyakoribb esete, amikor az attribútumok egyezését vizsgáljuk, ezt egyen összekapcsolásnak nevezzük.

Ez egy speciális szorzás mely a következő műveletsorral írható le

1. Vegyük az első reláció egy sorát
2. Az összekapcsolási feltételt vizsgáljuk meg a második táblázat összes sorára, ha igaz, adjuk mindkét reláció sorát az eredményhez
3. Folytassuk az 1. ponttal amíg van még sor az első relációban

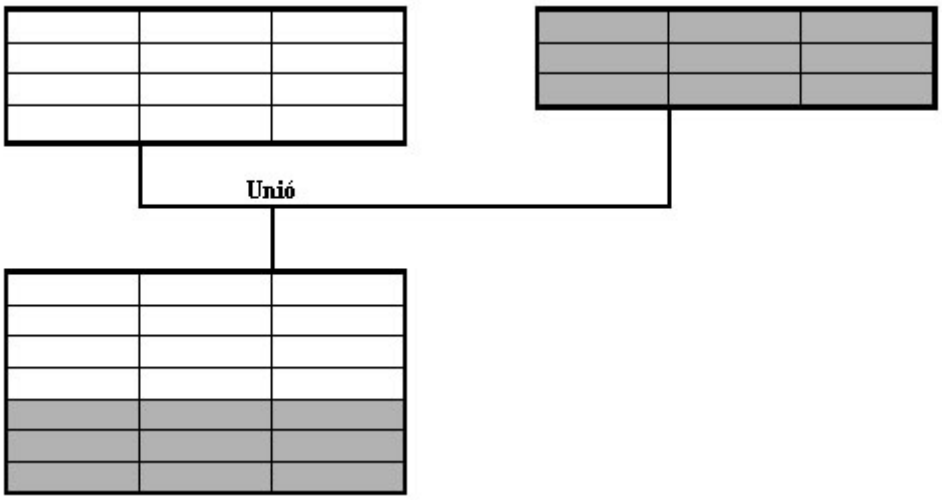
Az összekapcsolás eredmény relációjában az első reláció csak azon sorai szerepelnek, melyekre található a feltételt kielégítő sor a második relációban. Gyakran arra van szükség, hogy az első reláció valamennyi sora szerepeljen legalább egyszer az eredmény relációban. Ezt a fajta összekapcsolást külső összekapcsolásnak nevezzük.

**Halmaz műveletek**

A halmazokkal kapcsolatos alapvető műveleteket, unió metszet, különbség, a relációkra is értelmezzük. Minden értelmezett halmazművelethez legalább két operandus szükséges, a különbség esetében több sem lehet. A halmaz műveletek csak azonos szerkezetű relációk között hajthatók végre, ez alatt azt értjük, hogy a műveletbe bevont két reláció oszlopainak meg kell egyeznie az elnevezésben és a tárolt adat típusában is. A relációkra általában a komplemens képzés nem értelmezhető.

**Unió**

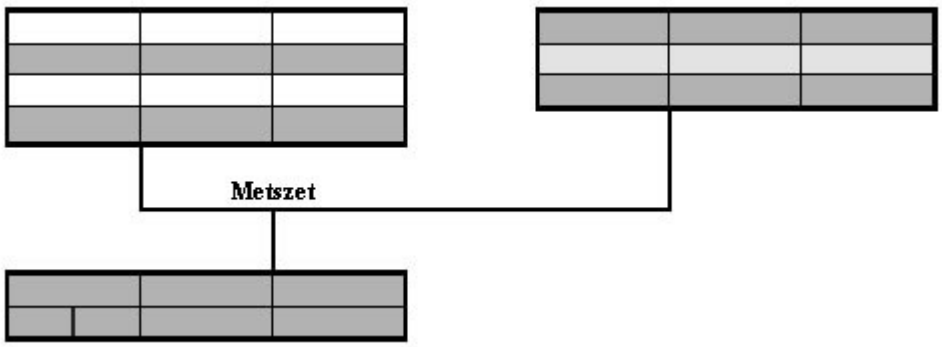
Az unió művelete azonos szerkezetű két vagy több reláció között végezhető el. Az eredmény reláció tartalmazza azokat a sorokat, melyek a műveletbe bevont relációk közül legalább egyben szerepelnek. Ha ugyanaz a sor az egyesítendő relációk közül többen is szerepelne, akkor is csak egyszer szerepel az eredmény relációban.




Az unió művelet

**Metszet**

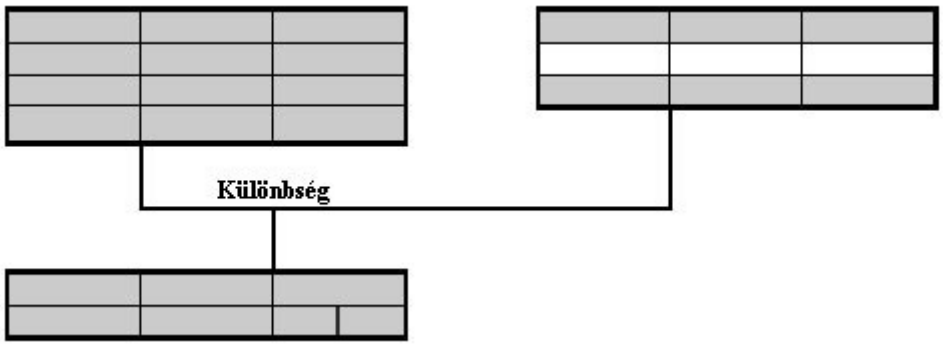
A metszet művelete azonos szerkezetű két vagy több reláció között végezhető el. Az eredmény reláció csak azokat a sorokat tartalmazza, melyek a műveletbe bevont relációk közül mindegyikben szerepelnek.




A metszet művelet

**Különbség**

A különbség művelete azonos szerkezetű két reláció között végezhető el. Az eredmény reláció csak azokat a sorokat tartalmazza, melyek a első relációban megtalálhatóak, de a másodikban nem.




A különbség művelet

# A normalizálás

**Az egyed-tulajdonság-kapcsolat modell**

**Egyed-kapcsolat adatmodell**

Az egyed-kapcsolat (Entity Relationship, ER) modellt, mint a relációs modellezés bevezetőjeként alkalmazzák. Előnyei közé tartozik az egyszerűség és a szoros kapcsolat, a könnyű konvertálhatóság a relációs modell felé. Az ER-modell, mint ahogy azt a neve mutatja, **három alapelemen** nyugszik:

- az **egyedek**,
- az egyedek közötti **kapcsolatok** és
- az egyedek **tulajdonsági** fogalmain.

Az ER-modell kizárólag a valóság strukturális leírásán alapszik, megengedve bizonyos egyszerűbb integritási feltételeket is. Ezen egyszerűség miatt, korábban vita bontakozott ki, hogy menyiben tekinthető egyáltalán adatmodellnek az ER rendszer. Egyesek szerint ez a modell csak az adattervezést támogatja, nem biztosít megfelelő precizitást, ezért nem is tekinthető adatmodellnek.

Az ER fogalomrendszerében a modellezett világ azon szereplőit, amelyek önálló léttel bírnak és melyekről több különböző adatot tartunk nyilván, **egyedeknek** nevezik. Az egyedek között vannak hasonló szerkezetűek, és vannak egymástól igen különböző felépítésűek. A hasonló felépítésű egyedek alkotnak egy **egyedtípust**. Minden egyedtípus több egyed előfordulást ölel át, ahol egy egyed előfordulás egy konkrét egyedet jelöl.

Az egyedeket az azokat jellemző **tulajdonságokkal** írjuk le. Tehát minden egyed rendelkezik egy sor tulajdonsággal, melyek más-más értékeket vehetnek fel. Az egyes egyedtípusok elsődlegesen a típushoz tartozó tulajdonságok körében térnek el egymástól. Az egyedtípus lényeges jellemzője tehát a hozzá tartozó tulajdonságok köre.

Az egyed előfordulások és az egyedtípusok nem izolált, elszigetelt szereplői a modellezett világnak. Az egyedek kapcsolatban állnak más egyedekkel, egy összetettebb struktúrát hozva létre. Tehát a modellben az egyedek mellett a **kapcsolataiknak** is szerepelni kell. Az egyedek között különböző bonyolultságú kapcsolatok állhatnak fenn és a modell akkor jó, ha alkalmas a kapcsolatok árnyalt kifejezésére.

**Anomáliák Def.:**
A nem megfelelő relációs modellből eredő **problémákat szokás anomáliáknak is nevezni**.
**Beszúrási anomáliáról** beszélünk, ha egy rekord felvitelekor, felesleges, már letárolt információkat is újra fel kell vinni.
**Módosítási anomáliáról van szó,** ha egy információegység módosításához több helyen is módosításra van szükség az adatbázisban. Ez nem csak többletmunkát jelent, de megnöveli az inkonzisztens állapot valószínűségét is, ha valahol elmaradna a módosítás.
**Törlési anomáliáról** akkor beszélünk, amikor egy információelem megszűnésekor más, hozzá nem tartozó információk is elvesznek.
A felsorolt anomáliák általában abból származnak, hogy nem az igazán összetartozó adatokat veszünk be egy relációba.

**A normalizálás szükségessége, és lépései – a normálformák**

Egy adatbázis megtervezésekor először összegyűjtjük azokat az adatokat, amelyekre szükségünk lesz. Ebből az adathalmazból, valamint az adatokon végzendő műveleti igényekből kiindulva **meghatározott lépések sorozatán** át jutunk el egyfajta **rendet tükröző relációs adatmodellhez**.

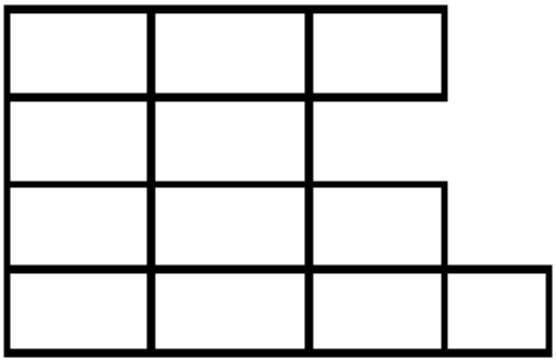
A helyes modell megtervezésére irányuló irányelveket, ennek **módszertanát** az irodalom **normalizálás** néven ismeri. A normalizálás során induláskor egyetlen táblázatot alakítunk ki, melyben elhelyezzük a szükséges tulajdonságokat. Ezután feltárjuk a táblázat belső szerkezetét, a tulajdonságok közötti összefüggéseket, függőségi viszonyokat; lényegében azt vizsgáljuk meg, hogy a feltárt függőségek eleget tesznek az ún. normál formáknak nevezett követelményeknek. A normál formákat megsértő függőségeket megszüntetjük úgy, hogy a táblázatot meghatározott szabályok szerint további táblázatokra bontjuk. A kapott új táblázatok normálformáknak való megfelelésének ellenőrzését is elvégezzük és, ha szükséges, azokat is további táblázatokra bontjuk.

A **normalizálás** tehát lényegében **táblázat szétbontó relációs műveletek sorozata**, melynek **eredményeképpen egymással kapcsolatban álló, az eredetinél kisebb tárolási igényű relációkat kapunk**; egy **tervezési metodika**, amely segítséget nyújt a **logikailag áttekinthetőbb**, (törlési, módosítási, beszúrási) **anomáliáktól mentes relációs sémák** és adatbázis sémák **kialakításában**.

A tervezési irányelveket követelmények formájában adják meg, méghozzá több, egymásra épülő követelmény alakjában. A szakirodalom öt (néha hat) normál formát különít el. E normál formák egymásra épülése azt jelenti, hogy egyes normál formák megkövetelik más normál formák teljesülését. A normál formák egyre szigorodó követelményrendszert reprezentálnak, illetve egyre szigorodó feltételrendszert jelentenek.

**Első normál forma (1NF)**
Egy reláció első normál formában van, ha minden attribútuma egyszerű nem összetett adat (minden mezője atomi értéket hordoz), valamint ha minden mezője funkcionálisan függ a kulcsmező csoporttól. Vagyis a függőségi rendszerben léteznie kell egy kulcsnak, s minden más mezőnek ettől kell függenie.

Az alábbi táblázatok nem felelnek meg az 1NF feltételeinek, mert nem egyforma az oszlopok száma, vagy többértékű attribútumuk van, vagy azonos sorai vannak.



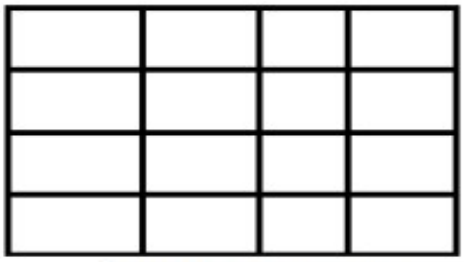
| Név | Hobbi |
|---|---|
| Kis Árpád | olvasás<br>úszás |
| Nagy Géza | tenisz<br>sakk |
| Tóth Erika | tánc<br>kung-fu<br>sütés-főzés |

| név | cím | végzettség |
|---|---|---|
| Kiss Benedek | Budapest | főiskola |
| Kovács Gábor | Szeged | egyetem |
| Kovács Gábor | Szeged | egyetem |

3-5. ábra. Relációk normál alakja – 1NF-nek nem megfelelő táblázatok.

Egy táblázat **1NF-ben van**, ha:

- **minden sora különböző**,
- **az oszlopok száma és sorrendje minden sorban azonos**,
- **minden oszlop csak egy attribútum értéket vesz fel**, továbbá
- **minden sorhoz tartozik egy egyedi kulcs, amitől az összes többi attribútum funkcionálisan függ**.



| Név | Hobbi |
|---|---|
| Kis Árpád | olvasás |
| Kis Árpád | úszás |
| Nagy Géza | tenisz |
| Nagy Géza | sakk |
| Tóth Erika | tánc |
| Tóth Erika | kung-fu |
| Tóth Erika | sütés-főzés |

| Személyi szám | név | cím | végzettség |
|---|---|---|---|
| 17204124168 | Kiss Benedek | Budapest | főiskola |
| 12711071125 | Kovács Gábor | Szeged | egyetem |
| 14908013514 | Kovács Gábor | Szeged | egyetem |

3-6. ábra. Relációk normál alakja – táblázatok módosítása 1NF előírásainak megfelelően.

**Második normál forma (2NF)**
Második normál formában van a reláció, ha az első normál formát teljesíti és ezen felül minden nem kulcs mező a teljes kulcstól függ, de nem függ a kulcs bármely valódi részhalmazától. Ezzel azt fejezzük ki, hogy a kulcs központi szerepet játszik a relációban, minden mezőnek a teljes kulcstól, s nem annak egy részétől kell függnie.

A reláció **2NF-ben van**, ha:

- **1NF-ben van** és
- **a nem kulcs attribútumok funkcionálisan teljesen függnek az elsődleges kulcstól**.

| tanazon | tannév | tancím |
|---|---|---|
| 812 | Szabó Ákos | Szeged |
| 512 | Sándor Sándor | Szeged |
| 912 | Vár Anikó | Budapest |

Ki-hova-mit

| tanazon | szak | tantárgy |
|---|---|---|
| 812 | tanár | matematika |
| 812 | tanár | biológia |
| 812 | tanár | kémia |
| 512 | filozófia | filozófia |
| 512 | filozófia | magyar |
| 512 | filozófia | történelem |
| 912 | tanár | angol |
| 912 | tanár | magyar |
| 912 | tanár | matematika |

Kar

| szak | karazon | karnév | karvez |
|---|---|---|---|
| tanár | 12 | tanárképző | Kovács |
| filozófia | 10 | bölcsész | Tóth |

3-10. ábra. Relációk normál alakja – táblázatok 2NF-ben.

Észrevehetjük, hogy ebben a lépésben megjelennek az idegen kulcsok is. A Tanuló relációban a tazon elsődleges kulcs, mert ez határozza meg a többi tulajdonságot. A Ki-hova-mit relációban elsődleges kulcs a tazon, szak és tantárgy attribútumok kombinációja, mivel ez határoz meg egy adott sort. Idegen kulcsok a tanazon és a szak mezők, mert a másik két táblára ezek segítségével hivatkozunk. A Kar relációban az elsődleges kulcs a szak, vagy a karazon.

**Harmadik normál forma (3NF)**
Harmadik normál formában van a reláció, ha teljesíti a második normál formát és ezenkívül igaz, hogy nem áll fenn tranzitív függőség, azaz nem áll fenn az egyik nem kulcs mezőből egy másik nem kulcs mezőbe irányuló függőség. Ilyen estben ugyanis a kulcs a köztes mezőn keresztül, tranzitíven határozza meg a másik mező értékét. A köztes mező a másik mezőnél egyfajta kulcs szerepet játszik.

**Példa:** A Kar nevű táblánkban vannak olyan mezők, amelyek közvetve is függnek az elsődleges kulcstól, így a szak meghatározza a kar azonosítóját, a kar azonosítója a kar nevét és a kar vezetőjét. Így még mindig maradtak rendellenességek a táblázatunkban, amit meg kellene szüntetni. Ha az egyik kar indít egy új szakot, akkor ismét be kell vinnünk a kar nevét, és vezetőjét feleslegesen. Célszerű tehát a Kar nevű táblánkat két újabb táblára bontani megszüntetve ezzel a közvetett függőségeket:

**Tanuló**

| tanazon | tannév | tancím |
|---|---|---|
| 812 | Szabó Ákos | Szeged |
| 512 | Sándor Sándor | Szeged |
| 912 | Vár Anikó | Budapest |

**Kar**

| karazon | karnév | karvez |
|---|---|---|
| 12 | tanárképző | Kovács |
| 10 | bölcsész | Tóth |

**Szak**

| szak | karazon |
|---|---|
| tanár | 12 |
| filozófia | 10 |

**Ki-hova-mit**

| tanazon | szak | tantárgy |
|---|---|---|
| 812 | tanár | matematika |
| 812 | tanár | biológia |
| 812 | tanár | kémia |
| 512 | filozófia | filozófia |
| 512 | filozófia | magyar |
| 512 | filozófia | történelem |
| 912 | tanár | angol |
| 912 | tanár | magyar |
| 912 | tanár | matematika |

3-11. ábra. Relációk normál alakja – táblázatok 3NF-ben.

**Negyedik normál forma (4NF)**
A harmadik normál formáig mindenféleképpen érdemes normalizálni a relációkat. Legtöbbször elegendő is az első három normál formának megfelelő relációk alkalmazása. Előfordulhatnak azonban olyan esetek is, amikor még ezután is maradnak anomáliák és feleslegesen tárolt adatok. A negyedik és ötödik normálforma a többértékű függőségekből adódó redundancia kiszűrését szolgálja.
Egy reláció negyedik normál formában van, ha egy XY többértékű függőséget tartalmazó relációban csak az X és Y-ban megtalálható attribútumokat tartalmazza.

Vagyis a reláció **4NF-ben van**, ha:

- **3NF-ben van,**
- **legfeljebb egy többértékű függés van benne**.

**Ötödik normál forma (5NF)**
Az ötödik normál forma esetén még tovább bonthatjuk a relációkat a redundancia megszüntetése érdekében. Ugyanakkor mivel ez már csak nagyobb tároló terület felhasználásával lehetséges, ezért általában az adatbázis tervezője dönt arról, hogy az ötödik normál formát és a nagyobb adatbázist vagy a redundanciát és a komplikáltabb frissítési, módosítási algoritmusokat választja.

**A normalizálás eredménye**
A normalizálás eredményeképpen olyan függőségi rendszert kaptunk, amely tiszta és egyértelmű. Minden táblánál léteznie kell egy függőségi centrumnak, a kulcsnak, és minden más mezőhöz léteznie kell függőségi kapcsolatnak a kulcsból. Ezeken a függőségeken kívül a relációséma nem tartalmazhat más függőségeket.

**Az adatbázis-tervezés lépései, üzemeltetési elemei, irányelvei**

Egy adatbázis létrehozását mindig az adatbázis tervezés előzi meg. Gondosan fel kell mérni az igényeket és meg kell fogalmazni a problémákat. Egy adatbázist manapság néhány hónap alatt fejlesztenek ki. Ennek körülbelül 70-75%-a a tervezésre, 10-15%-a a programozásra, a maradék pedig a tesztelésre fordított idő. Alapos tervezés nélkül a rendszer átláthatatlan lesz és utólagos módosítás már nagyon körülményes.

Nézzük tehát a ***tervezés lépéseit***:
1. fázis. Az igények összegyűjtése, elemzése.
2. fázis. Koncepcionális terv elkészítése.
3. fázis. Adatbázis-kezelő rendszer kiválasztása.
4. fázis. Adatbázis-kezelő rendszertől függő leképezés.
5. fázis. Fizikai tervezés.
6. fázis. Megvalósítás.

**1. fázis. Az igények összegyűjtése, elemzése, a feladat specifikációja.**
A folyamat során nagyon gondosan, átgondoltan fel kell deríteni a fő **alkalmazási területeket**, tanulmányozni kell az adott területtel rokon, már **meglévő alkalmazásokat** és azok **dokumentációit**. Meg kell vizsgálni a **jelenlegi megvalósításokat** (még a nem számítógépes megoldásokat is), valamint a körülményeket. A felhasználói igények, elvárások összegyűjtése érdekében célszerű a későbbi **felhasználókkal is elbeszélgetni**.

Az adatok, információk összegyűjtése után olyan **specifikációt** kell készíteni, mely tartalmazza a felhasználói igényeket kielégítő **tárolandó adatokat**, **valamint a feldolgozási műveleteket**, tranzakciókat, lekérdezéseket.

**2. fázis. A koncepcionális terv elkészítése.**
A terv készítésének folyamán kell a magas szintű modellt kialakítani. A modell segítségével meg kell fogalmazni az előre tervezhető lekérdezéseket és tranzakciókat. A terv **előnyei**:
- **közérthető** formában mutatja az adatbázis szerkezetét, az adatcsoportokat és azok kapcsolatait, valamint a korlátozásokat,
- olyan terv, leírás, amit az adatbázis-kezelő rendszer kiválasztásával, vagy a belső séma módosítása esetén **nem kell megváltoztatni**,
- nélkülözhetetlen abból a szempontból is, hogy mind a felhasználóknak, mind pedig a programozóknak **újabb ötleteket ad**,
- mivel könnyen megérthető, **segíti** a felhasználó és a programozó közötti **párbeszédet**.

A koncepcionális terv elkészítéséhez leggyakrabban az **egyed-kapcsolat** modellt használják, mivel **kifejezők** (az adattípusok mellett a kapcsolatok típusait is ábrázolják), **egyszerűek** (laikusok is viszonylag könnyen megérthetik), **kevés fogalmat használnak** (így rövid idő alatt megtanulható), **szemléletes ábrákat** használnak és **egyértelműek**, szinte nem lehet félreérteni azokat.

**3. fázis. Az adatbázis-kezelő rendszer kiválasztása.**
Az adatbázis-kezelő rendszer **kiválasztásában igen sok tényező játszhat szerepet**, mint például:

- a **feladat természete** (hierarchikus 1:N kapcsolat esetén választhatunk hierarchikus modellen alapuló rendszert, egyéb esetben hálós, vagy relációs modellen alapuló rendszert célszerű választani),
- **gazdaságossági megfontolások** (a hardver és szoftver költségei, betanítási és karbantartási stb. költségek),
- a **rendszer szolgáltatásai**, felhasználóbarát felület.

Egyszerűbb, pontosan tervezhető feladatok elvégzésére nem kell feltétlenül adatbázis- kezelő rendszert használni. Néha előnyösebb lehet egy házi fejlesztésű program, mint egy bonyolult és költséges rendszer. **Akkor érdemes adatbázis-kezelő rendszert használni**, ha:

- több program is használja ugyanazt az adatbázist,
- az adatok gyorsan szaporodnak, módosulnak,
- nagy az adatbázis,
- a különböző adattípusok között bonyolult kapcsolatrendszer áll fent és
- az adatellenőrzésre, az adatbiztonságra nagy az igény.

*Példa:* A kiskereskedő PC-t fog használni, melyhez elsősorban relációs adatbázis-kezelő programokat találhatunk a piacon. Ha a gépen a grafikus felületű Windows operációs rendszer fut, ezzel kompatibilis adatbázis-kezelő rendszert kell választanunk.

**4. fázis. Adatbázis-kezelő rendszertől függő leképezés.**
A **leképezés** lényegében a **logikai adatmodelltől függő szabályok alkalmazása**
(lásd: Egyed-kapcsolat modellből relációs modell).

A folyamat **automatizálható** az ún. CASE (Computer Aided Software Engineering Tools) programok segítségével. Ezek az eszközök az ER-modellből kiindulva a kiválasztott adatbázis-kezelő rendszer adatleíró nyelvén leírják az adatbázis szerkezetét.

Ebben a fázisban történik a **lekérdezések megtervezése is**, relációs műveletek sorozatával. A lekérdezések leírásához használhatunk relációs algebrát, vagy használhatunk, SQL-szerű utasításokat is.

**5. fázis. Fizikai tervezés.**
Itt kell dönteni a **tárolási szerkezetről** és a **hozzáférési módokról**, melybe az adatbázis-kezelő rendszeren kívül az operációs rendszer is beleszólhat.

Relációs adatmodell használata esetén a fizikai tervesben fontos szerepet játszik az **indexelés**. Döntő fontosságú lehet a lekérdező és aktualizáló – vagyis a beszúró, törlő és módosító – műveletek gyakorisága és egymáshoz való viszonya. Az indexelésnél azt is figyelembe kell venni, hogy az **adatok elérésének gyorsítása** mellett ez többlet helyfoglalással jár.

**6. fázis. Megvalósítás.**
Az adatleíró nyelven írt sémák alapján **létrejön** az **adatbázis szerkezete**. Az így kapott adatbázist feltölthetjük adatokkal.

A programozók **megírják** a **tranzakciók kódjait**, vagy a **felhasználói programokat**, melyek használhatják az adatbázis-kezelő nyelv parancsait (ez általában az SQL).

Valódi adatok felvitele előtt célszerű a rendszert mintaadatokkal **kipróbálni**, hogy ne túl későn derüljenek ki az esetleges hibák. Ezt már csak azért is célszerű megtenni, mert még ekkor is előfordulhat, hogy mind a felhasználó, mind a programozó olyan újabb lehetőségeket fedez fel, amire eddig nem is gondolt.

A rendszer megvalósítása után használat közben felmerülhetnek **problémák**, amiket **orvosolni** kell. Egyéb **módosítási igények** is jelentkezhetnek – részben új felhasználói kívánságok, részben a változó külső körülmények következtében –, amelyeket utólag szintén **be kell építeni**.

# SQL lekérdezések, és szintaktikájuk

Az SQL a strukturált lekérdező nyelv (Structured Query Language) rövidítése, melyet az IBM dolgozott ki a DB2 relációs adatbáziskezelőjéhez. Ma már a relációs adatbáziskezelők szabványosított nyelve, bár több dialektusa, bővítése alakult ki.

## Adatbázis-karbantartás (Az adatdefiníciós utasítások):

### Adatbázis létrehozása

```
CREATE DATABASE <adatbazis>;
```

A fenti paranccsal új adatbázist hozhatunk létre.

### Adatbázis törlése

```
DROP DATABASE <adatbazis>;
```

Adatbázisokat, beleértve a táblákat is, a DROP DATABASE paranccsal törölhetünk!
DROP DATABASE alkalmazása során körültekintően érdemes eljárni, ugyanis nem visszavonható!

### Adattábla létrehozása

```
CREATE TABLE <tábla> (
oszlop1 típus1 [DEFAULT <érték>] [NULL | NOT NULL] [CONSTRAINT <feltnév>
<feltétel>],
oszlop2 típus2 [DEFAULT <érték>] [NULL | NOT NULL] [CONSTRAINT <feltnév>
<feltétel>]);
```

Új adattábla létrehozása. Az utasításnak kötelezően meg kell adni az adattábla nevét, az oszlopok nevét és típusát. Ezek mellett megadhatunk az oszlopnak alapértelmezett értéket, illetve megadhatunk ellenőrzési feltételeket, amelyek segítségével ellenőrizhetjük az adott oszlopba történő adatbevitelt…

Amennyiben egy oszlopra megadjuk az alapértelmezett értéket a tábla készítésekor, és az adatbevitel során, nem határozzuk meg a mező értékét, akkor a mezőben az alapértelmezett érték fog szerepelni…

A ***NULL*** és a ***NOT NULL*** záradék segítségével meghatározhatjuk, hogy a mezőben szerepelhet-e NULL érték..

A ***CONSTRAINT*** záradék segítségével ellenőrzési feltételeket adhatunk meg az adott oszlop esetében. A feltételek közül a legfontosabb, az elsődleges kulcs definiálására vonatkozó feltétel, amelyet a következőképpen írhatunk fel: CONSTRAINT kulcs PRIMARY KEY.

A CONSTRAINT záradék segítségével további ellenőrzéseket is végezhetünk:

- A mező lehetséges értékeinek felsorolása: ***IN (felsorolás)***
- A mező lehetséges legkisebb és legnagyobb értékének megadása: ***BETWEEN érték1 AND érték2***

Az értékek felsorolása és az értékhatárok megadása esetén az adatok bevitele során ellenőrzésre kerül, hogy a megadott érték megfelel-e a megadott feltételeknek.

**Adattábla törlése**

```
DROP TABLE <tábla>;
```

Az utasítás záradékaként meg kell adni a törlendő adattábla nevét.

**Adattábla szerkezetének módosítása**

```
ALTER TABLE <tábla> <módosítások>;
```

Az ALTER TABLE utasítás segítségével a tábla szerkezetén tudunk változtatni. A lehetőségek a következők:

- *Új oszlop hozzáadása a táblához:*
  ```
  ALTER TABLE <tábla> ADD <oszlopnév> <típus>;
  ```
- *Létező oszlop törlése:*
  ```
  ALTER TABLE <tábla> DROP <oszlopnév>;
  ```
- *Létező oszlop nevének módosítása:*
  ```
  ALTER TABLE <tábla> ALTER <oszlopnév> TO <új oszlopnév>;
  ```
- *Létező oszlop típusának módosítása:*
  ```
  ALTER TABLE <tábla> ALTER <oszlopnév> TYPE <típus>;
  ```

## Adatkezelés (Adatdefiníciós utasítások):

**Adatok bevitele**

```
INSERT INTO <tábla> [(oszloplista)] VALUES (<értéklista>);
```

Az *INSERT INTO* utasítás segítségével új adatokat szúrhatunk be az adattáblába. Az utasítás záradékaként megadhatjuk az oszlopok listáját, de ez nem kötelező. Ha nem soroljuk fel az oszlopokat, akkor az utasítás alapértelmezés szerint az összes oszlopot figyelembe veszi. A *VALUES* záradékban kell megadni a beszúrandó adatok listáját, az értékek számának meg kell egyezni az oszlopok számával.

**Adatok törlése**

```
DELETE FROM <tábla> [WHERE <feltétel>];
```

A *DELETE* utasítás törli a táblából a *WHERE* záradékban meghatározott feltételnek megfelelő rekordokat. Ha a *WHERE* záradékot elhagyjuk, az az összes rekord törlését eredményezi.

**Adatok módosítása**

```
UPDATE <tábla> SET <mezo1=érték1>[,<mezo2=érték2>,...,<mezox=értékx>] [WHERE <feltétel>];
```

Az *UPDATE* utasítás segítségével a már feltöltött mezők értékét módosíthatjuk.

Az utasításnak meg kell adni a tábla nevét, illetve a *SET* záradékban a módosítandó mezők nevét, majd egyenlőségjel után az új értéket. A *WHERE* záradék használata nem kötelező, ha elhagyjuk, akkor az összes mező értéke módosítva lesz. A *WHERE* záradékban megadható feltételek azonosak a *DELETE* utasításnál leírt feltételekkel.

## Szűrés (Lekérdező utasítások):

Ebbe a csoportba mindössze egyetlen SQL utasítás tartozik, a *SELECT* utasítás. A *SELECT* utasítás talán a legsokoldalúbb az összes utasítás közül, segítségével bármilyen lekérdezés megvalósítható.

**A SELECT utasítás legegyszerűbb formája**

```
SELECT * FROM <tábla>;
```

Ebben a formában a táblában található összes adat lekérdezhető, tehát az összes oszlop összes sora megjelenítésre kerül.

**Szelekció (Korlátozás)**
A relációnak azokat a sorait kapjuk eredményül, amelyek megfelelnek egy adott feltételnek. Ebben az esetben a WHERE záradékot kell használnunk, a következő formában:

```
SELECT * FROM <tábla> WHERE <feltétel>;
```

Összetett feltételt is megadhatunk az *AND* (és) *OR* (vagy) *NOT* (nem) logikai kifejezések és a > = < relációjelek használatával.

Példa:

```
SELECT * FROM Auto WHERE szin=”fehér” AND tipus=”Opel”;
```

**Projekció (vetület)**

A projekció a táblázat leszűkítését jelenti bizonyos oszlopaira. Akkor használjuk, ha csak bizonyos oszlopok tartalmára vagyunk kíváncsiak.
A *SELECT* utasításnak megadhatjuk egyértelműen, hogy az adattábla mely oszlopait szeretnénk megjeleníteni. Ebben az esetben a * jel helyén vesszővel elválasztva fel kell sorolni az oszlopok neveit.

```
SELECT oszlop1, oszlop2,...,oszlopn FROM <tábla>;
```

**Kereszttáblás lekérdezés (Descartes szorzat):**

Két reláció keresztszorzatát úgy kapjuk meg, hogy az első reláció minden sorához hozzáírjuk a második reláció minden sorát. A műveletet abban az esetben, ha a két táblában van két azonos nevű oszlop, nem végezhetjük el. Ilyenkor az egyik oszlopot először át kell nevezni. Ezt a műveletet ritkán alkalmazzuk, mert nagyon ritkán van értelme, ezen kívül két nagyobb méretű tábla keresztszorzata egy hatalmas tárigényű, kezelhetetlen méretű táblázatot eredményezhet.

```
SELECT * FROM <tábla1>, <tábla2>;
```

## Klauzák (Záradékok):

**Feltételkezelés:**

A **WHERE** záradékban megadható feltételek nagyon sokfélék lehetnek. A legfontosabb lehetőségek a következők:

- **Egyenlőségvizsgálat:** *oszlop=érték*
  A feltétel azokat a rekordokat jelenti, ahol az oszlop értéke megegyezik az egyenlőségjel után álló értékkel.

- **Reláció vizsgálata:** *oszlop<érték* | *oszlop>érték* | *oszlop<=érték* | *oszlop>=érték*
  Ez a feltétel csak numerikus és dátum típusú mezők esetén alkalmazható.

- **Értékhatárok vizsgálata:** *oszlop between érték1 and érték2*
  A feltétel akkor teljesül, ha az oszlop értéke érték1 és érték2 közé esik. Csak numerikus és dátum típusú mezok esetén alkalmazható.

- **Hasonlóság vizsgálata:** *mező LIKE ¨ %érték¨%*
  Ezt a vizsgálatot csak a karakteres típusú mezők esetén alkalmazhatjuk.
  - A % nulla vagy több bármilyen típusú karakter helyettesít.
  - Az _ pontosan egy darab bármilyen típusú karaktert helyettesít.

**Ismétlődő sorok kihagyása**

Egyes esetekben előfordulhat, hogy a tábla egy-egy sorában egy oszlopon belül azonos értékek fordulnak elő. A lekérdezés során alapértelmezés szerint minden sor megjelenítésre kerül, tehát az ismétlődő sorok annyiszor jelennek meg a képernyőn, ahányszor a táblában szerepelnek. Amennyiben szeretnénk ezeket az ismétlődéseket kiszúrni, akkor a *DISTINCT* záradékot kell használnunk a lekérdezés során. Formája a következő:

```
SELECT mezo1, DISTINCT mezo2 FROM <tabla>;
```

**Az eredmény rendezett megjelenítése**

A lekérdezés során alapértelmezés szerint az adatok a felvitel sorrendjében jelennek meg. Lehetőség van azonban az adatok sorba rendezésére az *ORDER BY* záradék használatával, az adatok növekvő és csökkenő sorrendben egyaránt megjeleníthetők.

```
SELECT * FROM <tabla> ORDER BY <mezo> [ASC|DESC].
```

A rendezés az *ORDER BY* záradékban megadott mező értéke szerint történik, a rendezés irányát az *ASC* vagy *DESC* kulcsszó használata határozza meg. Ha nem rendelkezünk egyértelműen a rendezés irányáról, akkor alapértelmezés szerint az *ASC*-t használja az SQL, azaz növekvő sorrendben rendez. Az *ORDER BY* záradék használatával történő rendezés csak a megjelenítés során kerül figyelembevételre, az adattábla rekordjainak fizikai sorrendje nem változik, tehát csak logikai rendezésről van szó.

**A talált sorok csoportosítása**

A lekérdezések során lehetőség van arra, hogy a lekérdezett sorokat valamely oszlop értéke alapján csoportosítsuk, és az egy-egy csoportot képező sorok oszlopain végezzünk el egy vagy több, oszlopra vonatkozó számítási műveletet. Ehhez a *SELECT* utasítás *GROUP BY* záradékát kell felhasználnunk, melynek formája a következő:

```
SELECT <oszlop1>, <függvény(oszlop2)> FROM <tábla> GROUP BY <oszlop1>
```

*A példa értelmezése:* A *SELECT* utasítás először lekérdezi az összes sort, majd csoportosítja azokat az *oszlop1* értéke alapján. Az egyes csoportokba tartozó sorok *oszlop2* oszlopán elvégzi a megadott függvényt, majd megjeleníti az eredményt.

### Csoportosítás feltétellel

Ha a csoportosítás során szeretnénk meghatározni, hogy a csoportok közül csak azok jelenjenek meg, amelyek bizonyos feltételeknek megfelelnek, akkor már nem használhatjuk a *WHERE* záradékot. A csoportosított adatokra a *HAVING* záradék használatával adhatunk meg feltételeket, a következő formában:

```
SELECT <oszlop1>, <függvény(oszlop2)> FROM <tábla> GROUP BY <oszlop1> HAVING <feltétel>;
```

A *HAVING* záradék esetén ugyanazokat a feltételeket használhatjuk, mint a *WHERE* záradék esetében.

## Összetett lekérdezések (adattáblák összekapcsolása):

### Belső összekapcsolások

Az adatbáziskezelés során alapvető feladat több adattábla összekapcsolása egy-egy mezon keresztül. Az ilyen lekérdezések esetében a SELECT utasítás FROM záradékában az összes érintett adattáblát fel kell sorolni, a mezők felsorolása során pedig azt is meg kell adni, hogy mely adattáblában található az adott mező. A tényleges összekapcsolást a WHERE záradék teszi lehetővé, amelyben feltételként meg kell adni, hogy csak azok a rekordok jelenjenek meg a lekérdezés során, ahol a két táblát összekapcsoló mezők értéke egyenlő.

```
SELECT tabla1.mezo1, tabla1.mezo2,...,tabla1.mezoN, tabla2.mezo1,tabla2.mezo2,...,tabla2.mezoN
FROM tabla1,tabla2 WHERE tabla1.mezo1=tabla2.mezo2;
```

*Lekérdezést értelmezése:*
A *tabla1* és a *tabla2* táblából lekérdezzük a felsorolt mezőket, amelyek esetében igaz, hogy a *tabla1* táblában és a *tabla2* táblában található *mezo1 és mez2* mezők értéke egyenlő.

A mezők felsorolásánál a tábla nevét és a mező nevét ponttal választjuk el egymástól, az egyértelmű hivatkozás érdekében. Ugyanez megfigyelhető a *WHERE* záradék feltételének meghatározása során is.

### JOIN összekapcsolás

A belső összekapcsolások másik alakjában a *JOIN* kulcsszó szerepel. Itt a két tábla közötti kapcsolatot egy külön *ON* zárodéban adjuk meg. A lekérdezések így olvashatóbak lesznek, mert az összekapcsolás feltételei elkülönülnek a *WHERE* záradék szűrőitől.

```
SELECT oszlopok FROM tabla1 JOIN tabla2 ON tabla1.mezo1=tabla2.mezo1
```

**Külső összekapcsolás**

Az adattáblák összekapcsolását legtöbb esetben a *SELECT* utasítás *WHERE* záradékban megfogalmazott feltétellel célszerű összekapcsolni. Ez a módszer azonban csak akkor használható, ha a kapcsolatot kialakító mezők értéke egyik táblában sem lehet *NULL* érték, mivel a *NULL* értéket a *WHERE* záradék nem tudja kiértékelni. Ha bármelyik táblában előfordulhat null érték a kapcsolat kialakításában résztvevő mezők esetében, akkor már nem használhatjuk ezt a megoldást, csak a táblák között ilyen esetben csak külső kapcsolat alakítható ki.

A külső összekapcsolás lényege, hogy nem kötelező a kapcsolódó mezők értékének minden sorban megegyezni, megengedhető a mezők eltérése, sőt akár a *NULL* érték is. Az összekapcsolás során rendelkezhetünk arról, hogy az adatok megjelenítése során a jobb oldali, a bal oldali esetleg mindkét táblából listázzuk ki azokat a mezőket, amelyeknek értéke a két táblában eltérő. A külső összekapcsolást a következőképpen végezhetjük el:

```
SELECT t1.mezo1,...,t1.mezoN, t2.mezo1,...,t2.mezoN FROM tabla1
INNER <LEFT | RIGHT | FULL> JOIN tabla2 ON <t1.mezo1=t2.mezo1>;
```

A három összekapcsolási mód közötti különbség a nem egyenlő mezők listázásának módjában található meg: a *LEFT* kulcsszó esetén a bal oldali, a *RIGHT* kulcsszó esetén a jobb oldali, a *FULL* kulcsszó esetén pedig mindkét oldali táblából ki lesznek listázva azok a mezők is, amelyeknek értéke nem egyenlő a másik táblában található kapcsolómező értékével.