**Adatbázis, adatbázisrendszer, adatbázis-kezelő rendszer (DBMS) fogalma és jellemzői**

**Adatbázis (naív definíció)**

1. Egymással logikailag összefüggő, egymáshoz kapcsolódó, belsõ jelentéssel bíró adatok összessége (kollekciója, együttese).

2. Az adatbázis speciális célra tervezett, felépített és közzétett adatok együttese.

**Adatbázisekezelő rendszer (DBMS)**

Olyan szoftvercsomag/rendszer, amely számítógépes adatbázisok létrehozását és karbantartását támogatja.

**Adatbázisrendszer**

A DBMS szoftver magával az adatokkal együtt. Néha az alkalmazásokat is beleértjük.

Ajánlás az 1969-es Conference on Data Systems Languages-en az adatkezelésekre:

1. 1 **Összetett logikai adatszerkezetek:**

Az adatbázis (komplex) több fájlban való tárolásának

támogatása.

1. 2 **Irányított redundancia:**

Ha egy adat csak egy helyen van tárolva, akkor nem

léphet fel inkonzisztencia. Ha egy adat több helyen is

szerepel, akkor annak kikényszerítése, hogy konzisztens

legyen.

1. **3 Jogosultságkezelés:**

DBA - adatbázis adminisztrátor. Feladata a felhasználók és jogosultságaik definiálása. Minden adatbázis művelet ellenőrzés alatt hajtódik végre különböző jogosultságok

mellett.

1. **4 Konkurens hozzáférés:**

Alapvető igény, hogy konkurens (egyidejű) hozzáférés esetén is konzisztens maradjon az adatbázis. (OLTP rendszerek)

1. **5 Többféle hozzáférés:**

Többféle felhasználó: csak olvas, illetve írhat és módosíthat is. Lekérdezés lekérdező nyelvvel (SQL), menü vezérelt illetve természetes nyelvi interfésszel, GUI-k.

1. **6 Magas szintű nyelvek támogatása:**

Objektum-orientált adatbázis rendszerek kompatibilisek a C++-szal és a Java-val. Adatstruktúra kompatibilitás.

1. **7 Almodell szemlélet (nézetek):**

Nincs szükség többszöri tárolásra. Egy felhasználó több célra több nézetet hozhat létre.

1. **8 Emberi hatékonyság:**

A szabványok kikényszerítése gyorsítja a kommunikációt. Csökken az alkalmazás fejlesztés ideje (1:6-tól 1:4-ig). Flexibilitás: az igények változásával szükséges az

adatbázis szerkezetének megváltoztatása. A legfrissebb információk rendelkezésre állása hatékonyságnövelő. Gazdaságos skálázhatóság az emberi erőforrásokban.

1. **9 Program-adat függetlenség:**

A tradicionális fájl-kezelésnél a fájl szerkezetének megváltozása az összes program megváltozását eredményezi. A fájlszerkezet a DBMS katalógusban van elkülönítve az elérést biztosító programoktól. Egy adott fájlba minden további nélkül szúrhatunk be új attributumot. A felhasználói program képes működni az adatokon

függetlenül a művelet konkrét implementációjától (pl. az adó mértékének változása).

Az **adatbázis** egymással kapcsolatban álló adatok gyűjteménye. **Adat** alatt olyan ismert tényeket értünk, amelyek feljegyezhetők, és amelyeknek implicit jelentésük van. Ilyenek például az ismerőseink nevei, telefonszámai és címei. Ezeket az adatokat feljegyezhetjük egy indexelt névjegyzékbe, vagy tárolhatjuk merevlemezen is egy személyi számítógépen egy olyan szoftverrel, mint például a Microsoft Access vagy Excel. Az ilyen implicit jelentéssel bíró, egymással összefüggő adatok gyűjteménye az adatbázis.

Az adatbázis a következő implicit tulajdonságokkal rendelkezik:

* Az adatbázis a valós világ valamely részét reprezentálja, amelyet néha **minivilágnak** vagy **a modellezés tárgyának** nevezünk. A minivilágban bekövetkező változások megjelennek az adatbázisban.
* Az adatbázis adatok logikailag összetartozó gyűjteménye a benne rejlő jelentéssel együtt. Az adatok egy véletlenszerű összességét nem tekinthetjük adatbázisnak.
* Az adatbázist egy konkrét céllal tervezzük és építjük, továbbá konkrét céllal tárolt adatokkal töltjük föl. Jól meghatározott felhasználói csoport használja jól meghatározott, számukra értelmes céllal.

Más szóval az adatbázisnak van valamilyen forrása, ahonnan az adatok származnak, valamilyen szinten kölcsönhatásban áll a valós világ eseményeivel, és van egy közönsége, amelyet aktívan érdekel a tartalma. Egy adatbázis tetszőleges méretű és összetettségű lehet.

Az **adatbázis-kezelő rendszer (DBMS)** olyan programok gyűjteménye, amelyek lehetővé teszik a felhasználóknak, hogy létrehozzanak és karbantartsanak adatbázisokat. A DBMS egy *általános célú szoftverrendszer*, amely lehetővé teszi különböző felhasználók és alkalmazások számára az adatbázisok *definiálását*, *létrehozását*, *manipulálását* és *megosztását*. Az adatbázis **definiálása** az adatbázisban tárolandó adatokra jellemző adattípusok, adatszerkezetek és megszorítások megadását jelenti. Az adatbázis-definíció, azaz az adatbázist leíró információk szintén az adatbázisban tárolódnak adatbázis-katalógus vagy szótár formájában; ezeket **metaadatoknak** nevezzük. Az adatbázis **létrehozása** az a folyamat, amikor eltároljuk az adatokat valamilyen, a DBMS által vezérelt adathordozón. Az adatbázis **manipulálása** olyan funkciókat foglal magában, mint például az adatbázis lekérdezése konkrét adatok lekérése céljából, az adatbázis módosítása a minivilág változásának tükrözése céljából, vagy jelentések készítése az adatok alapján. Az adatbázis **megosztása** lehetővé teszi, hogy több felhasználó és program párhuzamosan hozzáférjen az adatbázishoz.

Hogy teljessé tegyük a kezdeti definícióinkat, az adatbázist, a felhasználókat, a DBMS szoftvert, illetve a DBMS-t futtató és az adatbázist tároló hardvert együtt **adatbázisrendszernek** nevezzük.

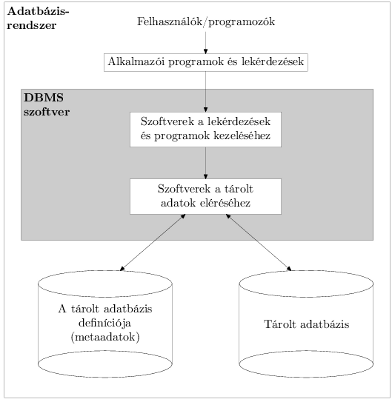


Figure 1Az adatbázisrendszer leegyszerűsített sémája

**Egyed, tulajdonság és kapcsolat fogalma és tulajdonságai**

Az **egyed** egy olyan valós világbeli objektumot vagy fogalmat reprezentál, amely szerepel az adatbázisban, mint például egy alkalmazott vagy egy projekt. A **tulajdonság** vagy **attribútum** az egyedet leíró, a modell szempontjából érdekes jellemző, mint például az alkalmazott neve vagy fizetése. Egy két vagy több egyed közötti **kapcsolat** az egyedek között fennálló valamilyen viszonyt jelöl. Ilyen például a „dolgozik rajta” kapcsolat egy alkalmazott és egy projekt között.

**Egyed**

A valós világnak az az eleme (tárgy, jelenség, elképzelés, személy, fogalom stb.), amely a modellezés tárgyát képezi. Egyednek tekinthetjük például: Kovács Péter, másodéves PTI-s hallgatót

**Tulajdonság**

Az egyednek a modellezés szempontjából lényeges jellemzője. Tulajdonságnak tekinthetjük például: a Kovács Péter nevet, a második évfolyamot, a PTI szakot

**Tulajdonságtípus**

Az azonos szerepű tulajdonságok absztrakciója. Tulajdonságtípusnak tekinthetjük például: a hallgató nevét, évfolyamát, szakját

**Egyedtípus**

Az azonos tulajdonságtípusokkal rendelkező egyedek absztrakciója. Egyedtípusnak tekinthetjük például: a hallgatót

**Kapcsolattípus**

Két vagy több egyedtípus közötti jól meghatározott viszony. Kapcsolattípusnak tekinthetjük például: a hallgató és a tantárgy között fennálló felvételi viszonyt

**Kapcsolat**

A két vagy több egyedtípus egyedei között fennálló viszony. Kapcsolatnak tekinthetjük például: Kovács Péter felvette az Adatbázisrendszerek tárgyat

**Relációs, objektum-relációs és NoSQL adatbázisok jellemzése**

Relációs

A relációs modellt Ted Codd vezette be 1970-ben az IBM Researchnél, később munkásságáért Turing díjat kapott.

Előnyei:

* egyszerűség
* matematikai megalapozottság

Alapja a matematikai reláció fogalma, ami hasonló egy értékekkel kitöltött táblázathoz (mátrixhoz).

Elméleti alapjai:

* halmazelmélet
* elsőrendű predikátumkalkulus (matematikai logika)

A relációs modellben az adatbázis relációknak egy halmaza lesz. Minden reláció durván egy flat fájlként fogható fel. A reláció értékek egy táblázata, amely sorok egy halmazából áll.

Minden egyes sor adat elemei a modellezett kisvilág egy egyed-előfordulásáról vagy egy kapcsolat-előfordulásáról tartalmaznak tényeket (információt).

A sorokat a formális modellben elem n-eseknek vagy rekordoknak fogjuk nevezni.

Minden egyes oszlop egy oszlop fejléccel (címmel) rendelkezik, amely az illető oszlopban lévő adatok jelentéséről ad információt. Az oszlop fejléceket a formális modellben attribútum neveknek (röviden attribútumoknak) fogjuk nevezni.

Minden sor rendelkezik egy olyan adatelem értékkel (vagy azok egy halmazával), amely egyértelműen azonosítja a sort a táblázatban. Ezt kulcsnak nevezzük.

Egy D tartomány atomi értékek egy halmaza.

Atominak nevezünk egy olyan értéket, amely már nem bontható tovább a relációs modell szempontjából.

Relációséma alatt az R(A1;A2; : : : ;An) jelölést értjük, ahol R a relációséma neve, A1;A2; : : : ;An pedig attribútumok. A relációs adatmodellben egy r (R) reláció nem más, mint egy

dom(A1); dom(A2); : : : ; dom(An) tartományokon értelmezett n-ed fokú matematikai reláció, amely részhalmaza azon tartományok Descartes-szorzatának, amelyek R-et definiálják:



Objektum-relációs

Sok relációs DBMS kibővült objektum adatbázis fogalmakkal egy új kategóriát az ún. objektum-relációs DBMSt (ORDBMS) létrehozva.  
A viselkedés leírására szolgáló eszközök alapvetőek az objektumorientált adatmodellekben, de már a hagyományosabb adatmodellekben is kezdenek megjelenni. Az objektum-relációs modellek például az alap relációs modellt egészítik ki többek között ilyen eszközökkel.   
Az **objektum adatmodell csoport** (object data model group, ODMG) olyan magasabb szintű implementációs adatmodellek egy új családjának tekinthető, amelyek közelebb állnak a koncepcionális adatmodellekhez.  
Fő jellemző és előny: a tervező specifikálni tudja mind a sokszorosan összetett objektumok szerkezetét, mind az ezeken az objektumokon alkalmazható műveleteket.  
Az objektum-relációs rendszerek (ORDBMS) a tradicionális RDBMS-ek OO szemlélettel való bővítései.  
Az objektumok az OO programozási nyelvekben tranziensek (átmenetiek), a futás befejeztével törlődnek Az objektumok az ODMS-ekben perzisztensek (állandóak), eltárolódnak, kés˝obb kinyerhetőek és megoszthatóak más programokkal.  
Egy ODMS általában több OO programnyelvhez kapcsolódik állandó és osztott objektumokat szolgáltatva számukra.  
Az ODMS-ekben az objektumok és a literálok tetszőleges összetettségű típus szerkezettel bírhatnak, amely az összes szükséges információt tartalmazza.

NoSQL

A NoSQL adatbázisok nem táblákban tárolják az adatokat, és általában nem használnak SQL nyelvet lekérdezésre.  
A relációs modellével szemben alkalmazásorientált megközelítést használunk.

A legtöbb NoSQL adatbázis szerver erősen optimalizált írás és olvasás műveletekre, míg ezen túl nem sok műveletet támogatnak.

A jobb sebesség és a skálázhatóság érdekében olyan adatszerkezeteket használnak tárolásra, mint a kulcs-érték párok, gráfok, dokumentum-adatbázisok vagy oszlopcsaládok.

Leginkább a big data-ban és a real-time web alkalmazásokban használatosak.

**A funkcionális függés fogalma**

A funkcionális függés egy olyan megszorítás, amely az adatbázis két attribútumhalmaza között áll fenn. Tegyük fel, hogy a relációs adatbázissémánknak *n* attribútuma van: *A*1, *A*2, ..., *An*; és gondoljunk az egész adatbázisunkra úgy, hogy azt egyetlen, **univerzális** *R* = { *A*1, *A*2, …, *An* } relációsémával írjuk le. Ez nem jelenti azt, hogy az adatbázisunkat egyetlen univerzális táblaként fogjuk tárolni; ezt a fogalmat csak az adatfüggőség formális elméletének a kialakításához használjuk.

**Definíció.** Az *R* két attribútumhalmaza, *X* és *Y* között értelmezett, *X* → *Y* alakú **funkcionális függés** egy *megszorítást* ír elő az *R* relációséma bármely *r* relációjának lehetséges rekordjaira. A megszorítás az, hogy bármely két *r*-beli *t*1 és *t*2 rekord esetén, amelyekre *t*1[*X*] = *t*2[*X*] teljesül, teljesülnie kell *t*1[*Y*] = *t*2[*Y*]-nak is.

Ez azt jelenti, hogy *r*-beli rekord *Y* komponensének az értékei *függnek* az *X* komponens értékektől; más szavakkal egy rekord *X* komponensének az értékei egyértelműen (vagy **funkcionálisan**) *meghatározzák* az *Y* komponens értékeit. Azt is mondhatjuk, hogy *X* **funkcionálisan meghatározza** *Y*-t, vagy *Y* **funkcionálisan függ** *X*-től. Az *X* attribútumhalmazt a funkcionális függés **bal oldalának**, az *Y*-t pedig a **jobb oldalának** nevezzük.

Ennélfogva *X* akkor és csak akkor határozza meg funkcionálisan *Y*-t egy *R* relációsémában, ha valahányszor *r*(*R*) két rekordja megegyezik az *X*-hez tartozó értékeikben, szükségszerűen megegyeznek az *Y*-hoz tartozó értékeikben is. Vegyük észre a következőket:

* Abból, hogy egy *R*-re előírt megszorítás szerint bármely *r*(*R*) relációpéldányban nem szerepelhet több rekord egy adott *X*-hez tartozó értékkel — azaz *X* **kulcsjelöltje** *R*-nek —, következik *X* → *Y* az *R* attribútumainak bármely *Y* részhalmazára (mivel a kulcsmegszorításból következik, hogy egyetlen legális *r*(*R*) állapotban sem lehet két olyan rekord, amelyeknek azonosak lennének az *X*-hez tartozó értékeik).
* Ha *X* → *Y* teljesül *R*-ben, még semmit sem tudunk mondani arról, hogy vajon *Y* → *X* is teljesül-e *R*-ben.

Egy funkcionális függés  **az attribútumok szemantikájának** vagy **jelentésének** egy jellemzője. Az adatbázistervezők *R* attribútumai szemantikájának — azaz hogy milyen kapcsolat van közöttük — az általuk vett értelmezését használják fel ahhoz, hogy megadják azokat a funkcionális függéseket, amelyek *R minden r* relációállapotában fennállnak. Valahányszor *R* két attribútumhalmazának a jelentése azt sugallja, hogy egy funkcionális függés fennáll, akkor felvesszük ezt a függést a megszorítások közé. Azokat a *r*(*R*) relációállapotokat, amelyek megfelelnek a funkcionális függés megszorításoknak, *R* **legális** vagy **jogszerű relációállapotainak** nevezzük. A funkcionális függések fő felhasználása tehát az, hogy tovább jellemezzük az *R* relációsémát azzal, hogy az attribútumaira olyan megszorításokat adunk, amelyek *mindig* érvényesek. Bizonyos funkcionális függéseket anélkül is felírhatunk, hogy egy konkrét relációra utalnánk, ehelyett a szóban forgó attribútumok egy tulajdonságaként adjuk meg őket a hétköznapi értelmük alapján. Például az {Állam, Jogosítványszám} → Társadalombiztosítási\_szám fennáll minden egyesült államokbeli felnőtt esetén. Az is elképzelhető, hogy egyes funkcionális függések a valós világban megszűnnek létezni, ha a kapcsolat módosul. Például az Irányítószám → Körzetszám valaha létező kapcsolat volt a postai kódok és a telefonszámkódok között az Egyesült Államokban, de a körzetszámok elburjánzása miatt ez ma már nem teljesül.

Tekintsük a 2-es (b) ábrán szereplő DOLG\_PROJ relációsémát; az attribútumok jelentése alapján tudjuk, hogy a következő funkcionális függéseknek kell fennállniuk:

1. Szsz → Dnév
2. Pszám → {Pnév, Phelyszín}
3. {Szsz, Pszám} → Órák

Ezek a funkcionális függések azt mutatják, hogy (a) egy dolgozó személyi száma (Szsz) egyértelműen meghatározza a dolgozó nevét (Dnév), (b) a projektszámnak egy értéke (Pszám) egyértelműen meghatározza a projekt nevét (Pnév) és helyszínét (Phelyszín), és (c) az Szsz és a PSzám értékeinek együttese egyértelműen meghatározza azt az időtartamot, ahány órát jelenleg a dolgozó a projekten hetente dolgozik (Órák). Másképpen úgy is mondhatjuk, hogy a Dnév funkcionálisan függ az Szsz-től (az Szsz funkcionálisan meghatározza a Dnév attribútumot), vagy *az Szsz egy adott értéke esetén ismerjük a Dnév értékét*, stb.

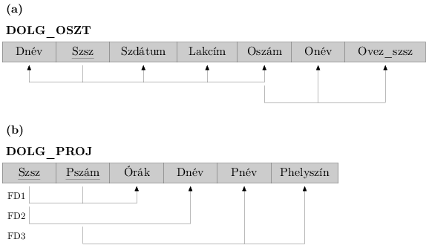
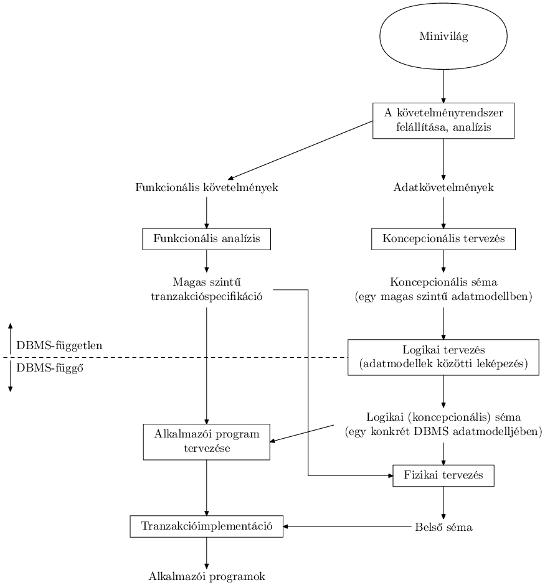


Figure 2

**Koncepcionális adatbázis-tervezés, az ER modell és leképezése relációs modellre**

A koncepcionális adatmodellek olyan fogalmakat használnak, mint például az egyed, a tulajdonság és a kapcsolat. A tervezés első lépése **a követelményrendszer felállítása és analízise**. Ennek a lépésnek az eredménye a felhasználók követelményeinek írásban rögzített, lényegretörő listája. Az adatkövetelmények megadásával párhuzamosan hasznos, ha az alkalmazás ismert **funkcionális követelményeit is megadjuk** Amint az összes követelményt összegyűjtöttük és elemeztük, a következő lépés az adatbázis **koncepcionális sémájának** létrehozása egy magas szintű koncepcionális adatmodell felhasználásával. Ezt a lépést **koncepcionális tervezésnek** nevezzük. A koncepcionális séma a felhasználók adatkövetelményeinek egy tömör leírása, és az egyedtípusok, a kapcsolatok és a megszorítások részletes leírását tartalmazza; ezeket a magas szintű adatmodell eszközeivel fejezzük ki. Mivel ezek az eszközök nem tartalmaznak megvalósítási részleteket, a laikus felhasználók számára rendszerint könnyebben érthetők, így a velük folytatott kommunikáció során jól használhatók.  
Az adatbázis-tervezés következő lépése az adatbázis tényleges megvalósítása egy kereskedelmi DBMS felhasználásával. A legújabb kereskedelmi DBMS-ek egy implementációs adatmodellt — mint amilyen a relációs vagy az objektum-relációs adatmodell — használnak, a koncepcionális sémát tehát a magas szintű adatmodellről az implementációs adatmodellre alakítjuk át. Ezt a lépést **logikai tervezésnek** vagy **az adatmodell leképezésének** nevezzük; ennek eredménye egy, a DBMS implementációs adatmodelljében leírt adatbázisséma.

Az utolsó lépés a **fizikai tervezés** fázisa, amelynek során megadjuk az adatbázis állományainak belső tárolási szerkezetét, indexeit, elérési útjait és állományszervezési módjait. Ezekkel a tevékenységekkel párhuzamosan tervezzük meg és implementáljuk az alkalmazói programokat mint adatbázis tranzakciókat a magas szintű tranzakcióspecifikációnak megfelelően.



Az ER modell az adatokat mint *egyedeket*, *kapcsolatokat* és *attribútumokat* írja le. Az ER modell által kezelt alapvető objektum az **egyed**, amely a valós világnak egy olyan *darabja*, amely önálló léttel bír. Az egyed lehet fizikai szinten létező objektum (például személy, autó, ház, dolgozó) vagy lehet fogalmi szinten létező objektum (például vállalat, foglalkozás vagy egy egyetemi tantárgy). Minden egyednek vannak **attribútumai** — az őt leíró tulajdonságok. Például egy dolgozó egyedet a dolgozó nevével, életkorával, címével, fizetésével és foglalkozásával lehet leírni. Egy konkrét egyed minden egyes attribútumához tartozik egy érték. Az egyedeket leíró attribútumértékek fogják alkotni az adatbázisban tárolt adatok nagy részét.

Az ER séma leképezése relációs sémára

Azokat az egyedtípusokat, amelyek nem rendelkeznek saját

kulcsattribútumokkal, gyenge egyedtípusoknak nevezzük.

Ezzel ellentétben azokat a (hagyományos) egyedtípusokat,

amelyekeknek van kulcsattribútumuk, er ˝os egyedtípusoknak nevezzük.

1. **Erős egyedtípusok leképezése.** Az ER séma minden *E* erős egyedtípusához rendeljünk hozzá egy *R* relációsémát, amely tartalmazza *E* összes egyszerű attribútumát. Az összetett attribútumoknak csak az egyszerű komponenseit adjuk hozzá *R* attribútumaihoz. Válasszuk *E* kulcs attribútumainak egyikét az *R* relációséma elsődleges kulcsául. Ha az *E*-ből választott kulcs összetett, akkor annak egyszerű attribútumai együttesen fogják alkotni *R* elsődleges kulcsát.
2. **Gyenge egyedtípusok leképezése.** Az ER séma minden *W* gyenge egyedtípusához rendeljünk hozzá egy *R* relációsémát, melynek attribútumai legyenek *W* összes egyszerű attribútuma és *W* összetett attribútumainak egyszerű komponensei. Továbbá adjuk hozzá *R* attribútumaihoz külső kulcs attribútumként azoknak a relációsémáknak az elsődleges kulcs attribútumait, amelyeket a domináns egyedtípusoknak feleltettünk meg; ezzel képezzük le a *W*-hez tartozó azonosító kapcsolattípust. *R* elsődleges kulcsa a tulajdonos egyedtípusok elsődleges kulcsainak és a *W* gyenge egyedtípus diszkriminátorának az együttese.

Ha egy *E*2 gyenge egyedtípus tulajdonosa a szintén gyenge *E*1 egyedtípus, akkor *E*1-et *E*2 előtt kell leképezni, mert az *E*2 leképezéséhez szükség van az *E*1-ből képzett relációséma elsődleges kulcsára.

1. **Bináris 1:1 számosságú kapcsolattípusok leképezése.** Minden bináris 1:1 számosságú *R* kapcsolattípus esetén meg kell határozni az *R*-ben részt vevő egyedtípusokból képzett *S* és *T* relációsémákat. Három lehetséges megközelítés létezik: (a) [külső kulcs használata](file:///C:\Home\Documents\Egyetem\Adatb\0046_fejezetek_az_adatbazisrendszerek_elmeletebol.docx#bin_1_1_kulsokulcs), (b) [összevonás](file:///C:\Home\Documents\Egyetem\Adatb\0046_fejezetek_az_adatbazisrendszerek_elmeletebol.docx#bin_1_1_osszevonas) és (c) [keresztreferencia vagy kapcsoló relációséma használata](file:///C:\Home\Documents\Egyetem\Adatb\0046_fejezetek_az_adatbazisrendszerek_elmeletebol.docx#bin_1_1_kereszthivatkozas). Az első megközelítés a leghasznosabb, és azt célszerű alkalmazni, hacsak bizonyos feltételek nem állnak fenn, ahogy azt mindjárt látni fogjuk:
2. **Külső kulcs használata.** Válasszuk ki az egyik relációsémát (mondjuk *S*-et), és vegyük fel *S* külső kulcsaként *T* elsődleges kulcsát. Célszerű *S*-nek azt a relációsémát választani, amelyiket abból az egyedtípusból képeztünk le, amelyik *totális résztvevője* az *R* kapcsolatnak. Vegyük fel továbbá *R* egyszerű attribútumait, illetve *R* összetett attribútumainak egyszerű komponenseit *S* attribútumaiként.
3. **Összevonás.** Egy másik lehetőség az 1:1 kapcsolatok leképezésére, ha a két egyedtípust és a kapcsolatot egyetlen relációba vonjuk össze. Ezt akkor tehetjük meg, ha *mindkét egyedtípus totális résztvevője* a kapcsolatnak.
4. **Kereszthivatkozás vagy kapcsoló relációséma használata .** A harmadik lehetőség, hogy felveszünk egy harmadik *R* relációsémát abból a célból, hogy kereszthivatkozással lássuk el a két egyedtípusból képzett *S* és *T* relációsémák elsődleges kulcsait. Ahogy látni fogjuk, ezt a megközelítést alkalmazzuk a bináris M:N kapcsolatoknál is. Az *R* relációsémát **kapcsoló relációsémának** nevezzük, mert *R* relációpéldányainak minden rekordja egy kapcsolat-előfordulást reprezentál, amely *S* egy relációjának egy rekordját *T* egy relációjának egy rekordjával kapcsolja össze.
5. **Bináris 1:N számosságú kapcsolattípusok leképezése.** Minden bináris 1:N számosságú *R* kapcsolattípus esetén meg kell határozni azt az *S* relációsémát, amelyiket a kapcsolattípus N-oldali egyedtípusából képeztünk. Vegyük fel *S* külső kulcsaként az *R*-ben részt vevő másik egyedtípusból képzett *T* relációséma elsődleges kulcsát; mindezt azért tesszük, mert az N-oldali egyed-előfordulások a másik oldalról legfeljebb egy egyed-előforduláshoz tartoznak. Vegyük fel továbbá *R* egyszerű attribútumait, illetve *R* összetett attribútumainak egyszerű komponenseit *S* attribútumaiként.

Egy másik megközelítés szerint most is használhatunk kapcsoló relációsémát (kereszthivatkozást), ahogy az 1:1 kapcsolatoknál tettük. Ekkor egy külön *R* relációsémát hozunk létre, amelynek attribútumai *S* és *T* elsődleges kulcsai, és amelynek elsődleges kulcsa megegyezik *S* elsődleges kulcsával. Ezt a módszert célszerű alkalmazni akkor, ha *S*-ben kevés rekord vesz részt a kapcsolatban, ugyanis ekkor a külső kulcsos megközelítés használatakor rengeteg NULL érték szerepelne a külső kulcsban.

1. **Bináris M:N számosságú kapcsolattípusok leképezése.** Minden bináris M:N számosságú *R* kapcsolattípus esetén hozzunk létre egy új *S* relációsémát, amely *R*-et reprezentálja. Vegyük fel *S* külső kulcsaként a kapcsolatban részt vevő egyedtípusokból képzett relációsémák elsődleges kulcsait; ezek együttese alkotja *S* elsődleges kulcsát. Vegyük fel továbbá *R* egyszerű attribútumait, illetve *R* összetett attribútumainak egyszerű komponenseit *S* attribútumaiként. Vegyük észre, hogy egy M:N kapcsolatot nem tudunk egyetlen külső kulccsal reprezentálni az egyik résztvevő relációsémában (ahogy az 1:1 és 1:N kapcsolattípusok esetén tettük) az M:N számosság miatt; mindenképpen létre kell hoznunk egy külön *S kapcsoló relációsémát*.

Azt is vegyük észre, hogy az 1:1 és 1:N kapcsolattípusokat mindig leképezhetjük az M:N kapcsolattípusok leképezéséhez hasonló módon, a kereszthivatkozás (kapcsoló relációséma) használatával úgy, ahogy azt korábban láttuk. Ez a megközelítés különösen hasznos akkor, ha kevés kapcsolat-előfordulás létezik, mert így elkerülhetjük a sok NULL érték felbukkanását a külső kulcsokban. Ebben az esetben a kapcsoló relációséma elsődleges kulcsát a kapcsolatban részt vevő egyedtípusokból képzett relációkra hivatkozó külső kulcsok közül *csak az egyik* alkotja. 1:N számosságú kapcsolattípus esetén a kapcsoló relációséma elsődleges kulcsa az N-oldali egyedtípusból képzett relációsémára hivatkozó külső kulcs lesz. 1:1 számosságú kapcsolattípus esetén bármelyik külső kulcsot használhatjuk a kapcsoló relációséma elsődleges kulcsaként, ha a hozzá tartozó relációban nincs NULL érték.

1. **Többértékű attribútumok leképezése.** Minden egyes *A* többértékű attribútum esetén hozzunk létre egy új *R* relációsémát. Ez az *R* relációséma tartalmazzon egy, az *A*-nak megfelelő attribútumot, valamint annak a relációsémának a *K* elsődleges kulcsát — *R* külső kulcsaként —, amelyet az *A*-t tartalmazó egyedtípusból vagy kapcsolattípusból képeztünk. *R* elsődleges kulcsát *A* és *K* együttese alkotja. Ha a többértékű attribútum összetett, akkor az egyszerű komponenseit vegyük fel *R* attribútumaiként.
2. **Az *n*-edfokú kapcsolattípusok leképezése.** Minden *n*-edfokú *R* kapcsolattípus esetén, ahol *n* > 2, hozzunk létre egy új *S* relációsémát, amely *R*-et reprezentálja. Vegyük fel *S* külső kulcsaként a kapcsolatban részt vevő egyedtípusokból képzett relációsémák elsődleges kulcsait. Vegyük fel továbbá *R* egyszerű attribútumait, illetve *R* összetett attribútumainak egyszerű komponenseit *S* attribútumaiként. *S* elsődleges kulcsa általában az összes külső kulcs együttese. Ha azonban az *R*-ben részt vevő valamely *E* egyedtípusból csak egy rekord vesz részt a kapcsolatban (számossági megszorítás), akkor *S* elsődleges kulcsának nem kell tartalmaznia az *E*-ből képzett *E*' relációsémára hivatkozó külső kulcsot.

**Az SQL elemei: DDL, DML, DCL,** **egyszerű lekérdezések és táblák összekapcsolása**

DDL- Data Definition Language

A DBA (adatbázis adminisztrátor) és adatbázis tervezők használják azért, hogy az adatbázis koncepcionális sémáját meghatározzák.

Sok DBMS-ben a DDL-t arra is használják, hogy a belső és a külső sémákat (nézeteket) definiálják.  
Az SQL DDL nyelve ad eszközöket a tartománymegszorítás, kulcsmegszorítás és egyéb (not null, unique, stb.) kényszerítésre.

DML - Data Manipulation Language

Arra használjuk, hogy az adatbázisból való keresést illetve az adatbázis frissítését specifikáljuk.  
A DML utasítások beágyazhatóak olyan általános célú programozási nyelvekbe mint a COBOL, C, C++, JAVA. A programozási nyelvek szintén nyújthatnak olyan függvény könyvtárakat, melyekkel DBMS-t érhetünk el.  
Fajtái:

* magas szintű vagy nem-procedurális nyelvek. Ilyen pl. az SQL. „Halmaz” orientáltak, azt mondják meg, hogy mit keresünk és nem azt, hogy hogyan. Deklaratív nyelvnek is nevezik.
* alacsony szintű vagy procedurális nyelvek. Az adatokat egy rekord egy időben elv alapján keresik. Ciklusok szükségesek több rekord kinyeréséhez mutatók pozicionálása útján.

DCL - Data Control Language

A DCL nyelv egy, az adatbázisok hozzáférésének szabályozásához használatos nyelv. DCL parancsok például a GRANT és a REVOKE.

Egyszerű lekérdezések és táblák összekapcsolása

SELECT \* FROM surveys

WHERE year >= 2000;

SELECT genus, species

FROM species

WHERE taxa = 'Bird'

ORDER BY species\_id ASC;

SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate  
FROM Orders  
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;