Tartalom – Készítette: Szemán László

[Adatbázisok és adatbázis-alkalmazások típusai (1.1) 4](#_Toc185934940)

[Alapfogalmak (1.2) 4](#_Toc185934941)

[A hagyományos adatkezelés problémái – Részletesen (1.3) 6](#_Toc185934942)

[CODASYL ajánlás – Részletes magyarázat (1.4) 8](#_Toc185934943)

[Példa modellezésre – Repülőgép helyfoglalási rendszer (1.5) 11](#_Toc185934944)

[Egyed és egyed-előfordulás (2.1) 12](#_Toc185934945)

[Tulajdonság és tulajdonság-előfordulás (2.2) 13](#_Toc185934946)

[Kapcsolattípus (2.3) 16](#_Toc185934947)

[Bachman-féle fogalomrendszer (2.4) 18](#_Toc185934948)

[Az adatmodell és az adatbázis fogalma (2.5) 18](#_Toc185934949)

[A tulajdonságtípusok (attribútumok) osztályozása (2.6) 19](#_Toc185934950)

[A NULL érték mint tulajdonság-előfordulás (2.7) 21](#_Toc185934951)

[A kapcsolattípusok osztályozása (2.8) 22](#_Toc185934952)

[ER diagram (2.9) 24](#_Toc185934953)

[Tartomány fogalma (3.1) 25](#_Toc185934954)

[Relációséma (3.2) 25](#_Toc185934955)

[Reláció (3.3) 25](#_Toc185934956)

[A relációs modell megszorításai (3.4) 27](#_Toc185934957)

[Séma alapú megszorítások (3.5) 27](#_Toc185934958)

[Szuperkulcs (3.6) 28](#_Toc185934959)

[Kulcsjelölt (3.7) 29](#_Toc185934960)

[Egyéb típusú megszorítások (3.8) 29](#_Toc185934961)

[Relációs adatbázisséma és relációs adatbázis (3.9) 30](#_Toc185934962)

[Gyenge és erős egyedtípus (4.1) 31](#_Toc185934963)

[Absztrakt lekérdező nyelvek (4.2) 32](#_Toc185934964)

[Szelekció (WHERE) (σ) (4.3) 33](#_Toc185934965)

[A projekció (SELECT) tulajdonságai (4.4) 34](#_Toc185934966)

[Átnevezés (ρ) (4.5) 35](#_Toc185934967)

[Uniókompatibilitás (4.6) 35](#_Toc185934968)

[Descartes-szorzat, Belső szorzat (4.7) 36](#_Toc185934969)

[Általános Összekapcsolás (Join) (4.8) 37](#_Toc185934970)

[Egyenlőségen Alapuló Összekapcsolás (Equijoin) (4.9) 38](#_Toc185934971)

[Természetes Összekapcsolás (Natural Join) (4.10) 38](#_Toc185934972)

[A Relációalgebrai Műveletek Teljes Halmaza (4.11) 39](#_Toc185934973)

[Osztás és Hányados műveletek (R ÷ S) (4.12) 39](#_Toc185934974)

[JOIN Kiterjesztése (4.13) 40](#_Toc185934975)

[Funkcionális függés (functional dependency, FD) (5.1) 41](#_Toc185934976)

[A funkcionális függések tulajdonságai (5.2) 42](#_Toc185934977)

[Az Armstrong-axiómák (5.3) 43](#_Toc185934978)

[A reláció attribútumainak szemantikája (6.1) 44](#_Toc185934979)

[Redundáns információk a rekordokban és a karbantartási anomáliák (6.2) 45](#_Toc185934980)

[NULL értékek a rekordokban (6.3) 47](#_Toc185934981)

[Álrekordok és a veszteségmentes összekapcsolás (6.4) 48](#_Toc185934982)

[Relációsémák normalizációja (6.5) 49](#_Toc185934983)

[Normálformák (6.6) 49](#_Toc185934984)

[Adatbázisrendszer (Database System) felépítése (7.1) 52](#_Toc185934985)

[Adatdefiníciós nyelv (DDL) (7.2) 53](#_Toc185934986)

[Adatmanipulációs nyelv (DML) (7.3) 53](#_Toc185934987)

[Adatvezérlő nyelv (DCL) (7.4) 53](#_Toc185934988)

[Felhasználók csoportjai (7.5) 54](#_Toc185934989)

[Szerepkörök (7.6) 54](#_Toc185934990)

[Osztály és alosztály az EER modellekben (8.1) 56](#_Toc185934991)

[Specializáció és Generalizáció (8.2) 56](#_Toc185934992)

[Totális és részleges, illetve diszjunkt és átfedő specializáció (8.3) 57](#_Toc185934993)

[Predikátumdefiniált és felhasználó által definiált specializáció (8.4) 58](#_Toc185934994)

[Kategória - unió típus (8.5) 59](#_Toc185934995)

[EER séma leképezése relációs sémára (8.6) 59](#_Toc185934996)

[Specializációk és generalizációk leképezése – több relációs lehetőségek (8.7) 60](#_Toc185934997)

[Specializációk és generalizációk leképezése – egyetlen relációs lehetőségek (8.8) 60](#_Toc185934998)

[Unió típusok (kategóriák) leképezése (8.9) 61](#_Toc185934999)

[DBS csoportosítása konkurens felhasználók száma szerint (9.1) 61](#_Toc185935000)

[Tranzakció (9.2) 62](#_Toc185935001)

[Adatbázis leegyszerűsített modellje (9.3) 62](#_Toc185935002)

[Konkurens hozzáférésből adódó problémák (9.4) 63](#_Toc185935003)

[Helyreállítás (Recovery) (9.5) 64](#_Toc185935004)

[A rendszer napló (system log) (9.6) 66](#_Toc185935005)

[Tranzakciók kívánatos tulajdonságai (ACID tulajdonságok) (9.7) 66](#_Toc185935006)

[Tranzakció ütemezés (schedule vagy history) (9.8) 67](#_Toc185935007)

[Megjegyzések a sorbarendezhetőségről (9.9) 68](#_Toc185935008)

[A konkurencia vezérlés (9.10) 70](#_Toc185935009)

[Az adattárház célja (10.1) 71](#_Toc185935010)

[Az adattárház definíciója (10.2) 72](#_Toc185935011)

[Az adattárházak által támogatott alkalmazások (10.3) 73](#_Toc185935012)

[Összehasonlítás: Hagyományos adatbázis vs. Adattárház (10.4) 74](#_Toc185935013)

[Az adattárházak koncepcionális felépítése (10.5) 75](#_Toc185935014)

[Adattisztítás folyamata az adattárházakban (10.6) 75](#_Toc185935015)

[Végfelhasználói eszközök az adattárházakban (10.7) 77](#_Toc185935016)

[Az adattárházak osztályozása (10.8) 77](#_Toc185935017)

[Adattárházak építése (10.9) 78](#_Toc185935018)

[Az adattárházak implementálásának nehézségei (10.10) 80](#_Toc185935019)

[Üzleti Intelligencia (Business Intelligence – BI) (10.11) 81](#_Toc185935020)

[NoSQL Adatbázisok (11.1) 82](#_Toc185935021)

[Elosztott Tárolás Típusai (11.2) 82](#_Toc185935022)

[CAP Tétel (Brewer Tétel) (11.3) 83](#_Toc185935023)

[Lehetőségek a CAP tétel szerinti kompromisszumokra (11.4) 84](#_Toc185935024)

[NoSQL adatbázis típusok (11.5) 85](#_Toc185935025)

## Adatbázisok és adatbázis-alkalmazások típusai (1.1)

**1. Hagyományos adatbázisok / alkalmazások**

* Numerikus vagy szöveges információk tárolására és kezelésére szolgálnak.

**2. Újabb típusú adatbázis-alkalmazások**

* **Multimédia adatbázisok**: Képek, hangok, videók tárolása és kezelése.
* **Térinformatikai rendszerek (GIS)**: Geográfiai adatok és térképek kezelése.
* **Adattárházak**: Nagy mennyiségű adat tárolása és elemzése, döntéstámogató rendszerekhez.
* **Valós idejű és aktív adatbázisok**: Azonnali válaszidőt igénylő alkalmazások számára, például ipari vezérlő rendszerekben.

**3. Egyéb alkalmazási területek**

* Számos más területen használatosak, például e-kereskedelem, egészségügy, pénzügy stb.

**Adatbázisok és technológia jelentősége**

* Az adatbázis-technológia nagy hatást gyakorolt a számítógépek szélesebb körű és intenzívebb használatára.
* A legtöbb alkalmazási területen **kritikus szerepet** játszanak az adatbázisok.

## Alapfogalmak (1.2)

**Adatbázis – Naiv definíció**

1. **Adatok kapcsolódása és jelentése**:
   * Egymással **logikailag összefüggő**, egymáshoz kapcsolódó, és belső jelentéssel bíró adatok összessége.
2. **Speciális célú tervezés**:
   * Az adatbázis egy **speciális célra tervezett**, felépített és közzétett adatok együttese.

**Fontos megjegyzés:**

* **Véletlenszerű adatösszesség nem adatbázis!**
* Egy adatbázis esetén létezik egy **célcsoport** és egy előre elképzelt **alkalmazási kör**.
* Az adatbázis a **valós világ** egy részéről tárol adatokat.

**Adat**

* Az **adat** ismert tény, amely:
  + **Rögzíthető**, azaz dokumentálható.
  + **Implicit jelentéssel** bír, vagyis magától értetődően hordozza az információt.

**Adatbázis-kezelő rendszer (DBMS)**

* **Definíció**:  
  Egy olyan **szoftvercsomag** vagy **rendszer**, amely:
  + **Adatbázisok létrehozását**,
  + **Lekérdezését**,
  + És **karbantartását** támogatja.

**Adatbázisrendszer**

* **Összetevők**:
  + A **DBMS szoftver** maga,
  + Az **adatok**,
  + (Néha ide sorolhatók az alkalmazások is, amelyek az adatbázist használják.)

**Példák adatbázisokra**

**1. Egyetemi rendszer**

* **Egyedek (entitások)**:
  + Diákok,
  + Oktatók,
  + Kurzusok,
  + Tanszékek.
* **Kapcsolatok**:
  + Egy diák felvesz egy kurzust.
  + Egy oktató egy tanszék oktatója.

**2. Helyfoglalási rendszerek**

* **Egyedek (entitások)**:
  + Ügyfelek,
  + Járatok,
  + Járművek (pl. buszok, repülők),
  + Sofőrök,
  + Utazási irodák.
* **Kapcsolatok**:
  + Egy utas foglal egy járatra.
  + Egy sofőr egy buszt vezet.

## A hagyományos adatkezelés problémái – Részletesen (1.3)

**1. Állománykezelés – Egyéni megoldások**

* **Egyedi definíciók és implementációk**:  
  Minden alkalmazás fejlesztője saját adatszerkezeteket, tárolási és kezelési megoldásokat hoz létre, amelyek nem szabványosak.
  + Ezek az adatok és algoritmusok az adott alkalmazás részét képezik, így az adatok nem függetlenek az alkalmazástól.
  + **Probléma**: Az adatok újrafelhasználása más alkalmazásokban nehézkes, mivel nincs központi, egységesített adatkezelés.

**2. Gépi hatékonyság**

* Az egyedi megoldások és a fejlesztők szakmai **tudásbeli korlátai** miatt:
  + Nem biztos, hogy **hatékony algoritmusokat** vagy optimalizált megoldásokat használnak.
  + **Erőforrás-pazarlás**: Az adatok kezelése több időt és memóriát igényelhet, mint szükséges lenne.
  + **Skálázhatósági problémák**: A rendszer nehezen bír el nagyobb adatmennyiségeket vagy megnövekedett terhelést.

**3. Redundancia → Inkonzisztencia**

* Az **adatredundancia** azt jelenti, hogy ugyanaz az adat többször előfordul különböző helyeken vagy állományokban.
  + Példa: Ugyanaz a felhasználónév, cím vagy más adat több fájlban is tárolva lehet.
* **Inkonzisztencia**:
  + Ha egy adatot módosítanak az egyik helyen, de a többi előfordulásnál nem, akkor az adatok között ellentmondás keletkezik.
  + **Következmény**: Az ilyen eltérések hibás működést okozhatnak az alkalmazásokban.

**4. Rugalmas változtatás lehetőségének hiánya**

* Az adatstruktúrák és a hozzájuk kapcsolódó programok nem elég **rugalmasak**.
  + Példa: Ha egy alkalmazásban egy név vagy cím formátuma megváltozik, a módosítást az összes érintett helyen manuálisan kell végrehajtani.
  + A változások nem „**gördülnek végig**” automatikusan az adatokon és az alkalmazásokon.
* **Következmény**: Az ilyen változtatások időigényesek, és növelik a hibák valószínűségét.

**5. Adatvédelem hiánya**

* Az adatvédelem nincs beépítve a hagyományos adatkezelési módszerekbe:
  + **Automatikus mentés hiánya**:  
    Adatok elveszhetnek rendszerhibák vagy emberi hiba esetén, mivel nincs rendszeres biztonsági mentés.
  + **Jogosultságkezelés hiánya**:  
    Nem lehet szabályozni, hogy ki férhet hozzá az adatokhoz, ami növeli a **jogtalan hozzáférés** veszélyét.
  + **Bizalmas információk kezelése**:  
    Az üzleti vagy személyes adatok, például ügyféladatok, fizetések vagy személyes azonosítók könnyen sérülhetnek vagy illetéktelenek kezébe kerülhetnek.

**6. Konkurens hozzáférés hiánya**

* A hagyományos adatkezelés feltételezi, hogy **egyszerre csak egy felhasználó** dolgozik az adatokkal.
  + Ha többen próbálnak egyidejűleg hozzáférni vagy módosítani az adatokat, az adatvesztéshez vagy hibás működéshez vezethet.
  + Példa: Egy foglalási rendszerben több ügyintéző egyidejű hozzáférése hibás foglalásokhoz vezethet.
* **Modern igényekkel szembeni hátrány**: Napjaink alkalmazásai gyakran többfelhasználós környezetben működnek, amit a hagyományos megoldások nem tudnak kezelni.

**7. Egysíkú hozzáférés**

* Az adatokhoz való hozzáférés **kötött** és nem elég **rugalmas**:
  + Minden felhasználónak ugyanaz a hozzáférési mód áll rendelkezésére, függetlenül az igényeitől.
  + Példa: Egy adminisztrátor és egy végfelhasználó más típusú lekérdezéseket végezne, de ez nem támogatott.
* **Hiányosságok**:
  + Nincs lehetőség komplex lekérdezések vagy adatelemzések végrehajtására.
  + Az információk csak előre meghatározott módokon érhetők el.

## CODASYL ajánlás – Részletes magyarázat (1.4)

**1. Összetett logikai adatszerkezetek**

* **Leírás**:  
  Az adatbázisok **több fájlban való tárolásának támogatása**, ami lehetővé teszi az adatok logikailag összetett szerkezetekben való rendszerezését.
* **Előny**:
  + Az adatok jobban modellezhetik a valós világ bonyolultságát.
  + Különálló táblák és kapcsolatok használatával az adatok kezelése rugalmasabb és átláthatóbb.
* **Példa**:  
  Egy vállalati rendszerben az alkalmazottak, osztályok és projektek külön fájlokban tárolhatók, de egymással logikailag összekapcsolhatók.

**2. Irányított redundancia**

* **Leírás**:  
  A redundancia csökkentésével és kontrollált használatával biztosítható az adatok **konzisztenciája**.
  + Ha egy adat **csak egy helyen tárolódik**, az inkonzisztencia kizárt.
  + Ha redundáns adatok vannak, a DBMS feladata, hogy garantálja azok **szinkronban tartását**.
* **Előny**:
  + **Konzisztens adatok** még redundancia esetén is.
  + Csökkenthető az adatmódosítások során fellépő hibák kockázata.
* **Példa**:  
  Egy ügyfél neve egy helyen kerül tárolásra, de ha más táblákban is hivatkoznak rá, akkor a módosítás automatikusan végigvezethető.

**3. Jogosultságkezelés**

* **Leírás**:  
  A **DBA (adatbázis-adminisztrátor)** feladata a felhasználók és jogosultságaik definiálása.
  + A különböző felhasználók hozzáférési szintjeit meghatározzák, például:
    - Olvasás,
    - Írás,
    - Módosítás.
  + Minden adatbázis-művelet **ellenőrzés alatt** történik.
* **Előny**:
  + Növekvő biztonság: illetéktelen hozzáférés kizárása.
  + Üzleti alkalmazásoknál biztosítható a bizalmas adatok védelme.
* **Példa**:  
  Egy pénzügyi rendszerben az adminisztrátor teljes hozzáféréssel bírhat, míg az alkalmazottak csak olvashatják az adatokat.

**4. Konkurens hozzáférés**

* **Leírás**:  
  Az adatbázisnak képesnek kell lennie **egyidejű felhasználói hozzáférések** kezelésére, miközben biztosítja az **adatintegritást**.
  + Példa: Online tranzakció-feldolgozó rendszerek (OLTP) esetében.
* **Előny**:
  + Egyidejű lekérdezések és módosítások is lehetségesek, anélkül, hogy az adatok sérülnének vagy ellentmondásossá válnának.
* **Példa**:  
  Egy légitársasági foglalási rendszerben több ügyfél is foglalhat egyszerre, miközben a rendszer biztosítja, hogy egy ülőhelyet csak egy személy kapjon meg.

**5. Többféle hozzáférés**

* **Leírás**:  
  Az adatbázisnak támogatnia kell többféle felhasználói interakciót, például:
  + **Olvasás** és **írás** jogosultságok,
  + Különböző felhasználói interfészek, például:
    - Lekérdező nyelv (SQL),
    - Menüvezérelt rendszerek,
    - Természetes nyelvi interfészek,
    - Grafikus felhasználói felületek (GUI-k).
* **Előny**:
  + A felhasználók igényei szerint testre szabható az adatbázis elérése.
* **Példa**:  
  Egy HR-rendszerben a menedzserek grafikus felületen kérdezhetnek le adatokat, míg a rendszergazdák SQL-lekérdezéseket futtatnak.

**6. Magas szintű nyelvek támogatása**

* **Leírás**:  
  Az adatbázisok integrációja a **magas szintű programozási nyelvekkel** (pl. C++, Java).
  + **Objektumorientált adatbázisok** esetén az adatbázisok és programok közötti kapcsolat egyszerűbbé válik.
* **Előny**:
  + A programozók közvetlenül kezelhetik az adatbázist a preferált nyelvükkel.
* **Példa**:  
  Egy webshop esetében a Java alkalmazás közvetlenül hozzáfér az adatbázishoz a rendelések feldolgozására.

**7. Almodell szemlélet (nézetek)**

* **Leírás**:  
  A felhasználók különböző **nézeteket** hozhatnak létre ugyanazon adatokhoz, anélkül, hogy az adatok többször tárolódnának.
  + Egy nézet egy specifikus szempontból reprezentálja az adatokat (pl. adott osztály adatai).
* **Előny**:
  + Csökken az adatredundancia.
  + Több célra használható adatmodellezés.
* **Példa**:  
  Egy orvosi adatbázisban az orvosok egy beteg kórelőzményét látják, míg az adminisztrátor csak a beteg személyes adatait.

**8. Emberi hatékonyság**

* **Leírás**:
  + A szabványosított adatbáziskezelés növeli a hatékonyságot:
    - **Gyorsabb kommunikáció**,
    - Csökkentett fejlesztési idő (1:6-ról 1:4-re).
  + Az adatbázis szerkezete rugalmasan módosítható az igények változásával.
* **Előny**:
  + Az adatok mindig friss állapotban elérhetők, ami **idő- és költségmegtakarítást** eredményez.
* **Példa**:  
  Egy vállalatnál a riportok gyorsabb előállítása csökkenti a döntéshozatali időt.

**9. Program-adat függetlenség**

* **Leírás**:  
  Az adatbázis szerkezetének módosítása nem érinti az adatokat elérő programokat.
  + A DBMS **katalógusban** tárolja az adatstruktúrákat, elkülönítve az alkalmazásoktól.
* **Előny**:
  + Könnyebben kezelhető a változó adatszerkezet.
  + Új attribútumokat lehet hozzáadni, anélkül hogy a meglévő programok működésében fennakadást okoznának.
* **Példa**:  
  Egy banki alkalmazás képes működni az új ügyféladatokkal anélkül, hogy újrakódolást igényelne.

## ****Példa modellezésre – Repülőgép helyfoglalási rendszer (1.5)****

**Cél**

Tudni szeretnénk minden járatról az alábbi információkat:

* Az adott járathoz tartozó repülőgép.
* A járaton dolgozó személyzet.
* Az adott járaton utazó utasok.

**Adatkövetelmények**

1. **Menetrend – Járatok:**
   * Az egyes járatok indulási és érkezési ideje.
   * Az indulási és érkezési repülőterek.
2. **Erőforrások – Repülőgépek:**
   * Az egyes járatokhoz rendelt repülőgépek adatai (azonosító, típus, kapacitás).
3. **Utasok:**
   * Az utasok alapadatai (név, kapcsolattartási információk).
   * Az általuk foglalt helyek járatonként.
4. **Személyzet:**
   * A járatok személyzete (pilóták, légiutaskísérők stb.).

## ****Egyed és egyed-előfordulás (2.1)****

**Egyed fogalma**

Az **egyed** a valós világ azon eleme, amely a modellezés során a vizsgálat tárgyát képezi. Az egyed lehet:

* **Tárgy** (pl. autó, épület),
* **Jelenség** (pl. esemény, folyamat),
* **Elképzelés** (pl. terv, koncepció),
* **Személy** (pl. diák, alkalmazott),
* **Fogalom** (pl. tantárgy, földrajzi hely).

Egyedek konkrét, azonosítható példányait nevezzük **egyed-előfordulásoknak**.

**példák egyedekre és egyed-előfordulásokra**

1. **Személy (egyed):**
   * Egyed-előfordulás: *Kovács Péter*, másodéves PTI BSc hallgató.
2. **Autó (egyed):**
   * Egyed-előfordulás: Az *IHJ-818 rendszámú piros Suzuki*.
3. **Tantárgy (egyed):**
   * Egyed-előfordulás: Az *INDK501 kódú, Adatbázisrendszerek nevű, 5 kredites tantárgy*.
4. **Földrajzi hely (egyed):**
   * Egyed-előfordulás: A *Fekete-tenger*, amelynek területe *422000 km²*, legnagyobb mélysége pedig *2210 m*.

## ****Tulajdonság és tulajdonság-előfordulás (2.2)****

**Tulajdonság fogalma**

A **tulajdonság** az egyedeknek a modellezés szempontjából lényeges jellemzője, amely lehet bármilyen olyan információ, amely segít az egyedek leírásában és azonosításában. A tulajdonságok az egyedek konkrét jellemzői, amelyek az adatbázisban tárolt információk.

A **tulajdonság-előfordulás** pedig egy adott egyedhez rendelt konkrét érték, amely az adott jellemzőt meghatározza.

**Példák tulajdonságokra és tulajdonság-előfordulásokra**

1. **Személy (egyed)**
   * **Tulajdonságok:**
     + Név: *Kovács Péter*
     + Évfolyam: *második évfolyam*
     + Szak: *PTI BSc*
   * **Tulajdonság-előfordulások:**
     + Kovács Péter neve, második évfolyam, PTI BSc szak
2. **Autó (egyed)**
   * **Tulajdonságok:**
     + Rendszám: *IHJ-818*
     + Szín: *piros*
     + Márka: *Suzuki*
   * **Tulajdonság-előfordulások:**
     + IHJ-818 rendszám, piros szín, Suzuki márka
3. **Tantárgy (egyed)**
   * **Tulajdonságok:**
     + Kód: *INDK501*
     + Név: *Adatbázisrendszerek*
     + Kreditszám: *5 kredit*
   * **Tulajdonság-előfordulások:**
     + INDK501 kód, Adatbázisrendszerek név, 5 kredites tantárgy
4. **Földrajzi hely (egyed)**
   * **Tulajdonságok:**
     + Név: *Fekete-tenger*
     + Terület: *422000 km²*
     + Legnagyobb mélység: *2210 m*
   * **Tulajdonság-előfordulások:**
     + Fekete-tenger neve, 422000 km² terület, 2210 m legnagyobb mélység

#### **Tulajdonságtípus fogalma**

A **tulajdonságtípus** az azonos szerepű tulajdonságok **absztrakciója**, vagyis a hasonló jellemzők egy csoportosítása. A tulajdonságtípusok olyan általános kategóriák, amelyek az egyes tulajdonságokat rendszerezik, így segítik a modellezés és az adatbázisok szerkezetének logikus felépítését.

**Példák tulajdonságtípusokra**

1. **Hallgató (egyed) tulajdonságai:**
   * **Tulajdonságtípus:** Hallgató adatai
     + Név
     + Évfolyam
     + Szak
2. **Autó (egyed) tulajdonságai:**
   * **Tulajdonságtípus:** Autó adatai
     + Rendszám
     + Szín
     + Márka
3. **Tantárgy (egyed) tulajdonságai:**
   * **Tulajdonságtípus:** Tantárgy adatai
     + Kód
     + Név
     + Kreditszám
4. **Tenger (egyed) tulajdonságai:**
   * **Tulajdonságtípus:** Tenger adatai
     + Név
     + Terület
     + Legnagyobb mélység

#### Egyedtípus fogalma

Az **egyedtípus** az azonos **tulajdonságtípusokkal** rendelkező egyedek **absztrakciója**. Ez a fogalom az adatmodellben azon kategóriákat jelöli, amelyek azonos típusú és szerkezetű tulajdonságokkal rendelkező egyedeket tartalmaznak. Más szóval, az egyedtípusok egyesítik a hasonló tulajdonságokkal rendelkező egyedeket egy közös csoportba.

**Példák egyedtípusokra**

1. **Hallgató (egyedtípus):**
   * **Tulajdonságok:**
     + Név (szöveg)
     + Évfolyam (szám)
     + Szak (szöveg)
   * **Az egyedtípus tartalmazza:** Az összes olyan egyedet, aki hallgató, tehát minden egyedet, akinek a tulajdonságai megfelelnek a hallgatói adatok típusainak (név, évfolyam, szak).
2. **Autó (egyedtípus):**
   * **Tulajdonságok:**
     + Rendszám (szöveg)
     + Szín (szöveg)
     + Márka (szöveg)
   * **Az egyedtípus tartalmazza:** Minden olyan egyedet, ami autó, tehát az összes autó, melyeknek tulajdonságai a rendszám, szín és márka.
3. **Tantárgy (egyedtípus):**
   * **Tulajdonságok:**
     + Kód (szöveg)
     + Név (szöveg)
     + Kreditszám (szám)
   * **Az egyedtípus tartalmazza:** Az összes tantárgyat, amelyeknek tulajdonságai a kód, név és kreditszám.
4. **Tenger (egyedtípus):**
   * **Tulajdonságok:**
     + Név (szöveg)
     + Terület (szám)
     + Legnagyobb mélység (szám)
   * **Az egyedtípus tartalmazza:** Az összes tenger, amelyeknek a neve, területe és legnagyobb mélysége szerepel az adatbázisban.

## Kapcsolattípus (2.3)

**Kapcsolattípus fogalma**

A **kapcsolattípus** két vagy több **egyedtípus** közötti jól meghatározott viszonyt jelent. A kapcsolattípusok a különböző egyedek közötti interakciókat, összefüggéseket írják le, amelyek az adatbázisban fontos szerepet játszanak az adatok összekapcsolásában és a lekérdezésekben.

**Példák kapcsolattípusokra**

1. **Hallgató és Tantárgy (felvételi viszony):**
   * **Kapcsolat:** A hallgató felvesz egy tantárgyat.
   * **Kapcsolattípus:** Felvételi viszony
   * **Egyedtípusok:** Hallgató, Tantárgy
   * **Leírás:** Minden hallgató felvehet több tantárgyat, és egy tantárgyat több hallgató is felvehet.
2. **Hallgató és Tantárgy (teljesítési viszony):**
   * **Kapcsolat:** A hallgató teljesíti a tantárgyat.
   * **Kapcsolattípus:** Teljesítési viszony
   * **Egyedtípusok:** Hallgató, Tantárgy
   * **Leírás:** A hallgató sikeresen letudott egy tantárgyat (például vizsgázott), ami a tantárgy teljesítését jelenti.
3. **Hallgató és Autó (birtoklási viszony):**
   * **Kapcsolat:** A hallgató birtokol egy autót.
   * **Kapcsolattípus:** Birtoklási viszony
   * **Egyedtípusok:** Hallgató, Autó
   * **Leírás:** A hallgató tulajdonosa egy autóval, és a kapcsolat azt jelzi, hogy az autó a hallgató birtokában van.
4. **Két Hallgató (csoporttársi viszony):**
   * **Kapcsolat:** Két hallgató csoporttársak.
   * **Kapcsolattípus:** Csoporttársi viszony
   * **Egyedtípusok:** Hallgató, Hallgató
   * **Leírás:** Két hallgató közösen dolgozik egy projektben vagy egy kurzus keretében, és így csoporttársak lesznek.
5. **Hallgató és Tenger (nyaralási viszony):**
   * **Kapcsolat:** A hallgató nyaralni megy a tengerhez.
   * **Kapcsolattípus:** Nyaralási viszony
   * **Egyedtípusok:** Hallgató, Tenger
   * **Leírás:** A hallgató utazást tesz a tengerhez, így kapcsolat alakul ki közte és a tenger között.

#### **Kapcsolat fogalma**

A **kapcsolat (kapcsolat-előfordulás)** két vagy több **egyedtípus** egyedei között fennálló viszony, amely valós konkrét eseményt, kapcsolatot jelent. A kapcsolat például azt írja le, hogy egy egyed milyen módon és miként kapcsolódik egy másik egyedhez, és mi történik az egyedek között a valós világban.

**Példák kapcsolat-előfordulásokra**

1. **Kovács Péter felvette az Adatbázisrendszerek tárgyat**
   * **Kapcsolat:** A hallgató, Kovács Péter felvette az Adatbázisrendszerek tantárgyat.
   * **Kapcsolat típus:** Felvételi viszony
   * **Kapcsolat-előfordulás:** Az adott hallgató konkrét eseményét jelenti, hogy a Kovács Péter nevű hallgató az Adatbázisrendszerek tárgyat választotta.
2. **Kovács Péter felvette a Magas szintű programozási nyelvek 2 tárgyat**
   * **Kapcsolat:** Kovács Péter felvette a Magas szintű programozási nyelvek 2 tantárgyat.
   * **Kapcsolat típus:** Felvételi viszony
   * **Kapcsolat-előfordulás:** Ez egy másik konkrét felvételi esemény, ahol Kovács Péter ezt a konkrét tantárgyat választotta.
3. **Szabó Ferenc felvette az Adatbázisrendszerek tárgyat**
   * **Kapcsolat:** Szabó Ferenc felvette az Adatbázisrendszerek tantárgyat.
   * **Kapcsolat típus:** Felvételi viszony
   * **Kapcsolat-előfordulás:** Ez egy újabb konkrét esemény, amelyben Szabó Ferenc hallgató az Adatbázisrendszerek tantárgyat vette fel.
4. **Nagy Géza felvette az Adatbázisrendszerek tárgyat**
   * **Kapcsolat:** Nagy Géza felvette az Adatbázisrendszerek tantárgyat.
   * **Kapcsolat típus:** Felvételi viszony
   * **Kapcsolat-előfordulás:** Ez is egy adott esemény, ahol Nagy Géza választotta az Adatbázisrendszerek tantárgyat.

## Bachman-féle fogalomrendszer (2.4)

|  |  |  |
| --- | --- | --- |
|  | absztrakt | konkrét |
| Egyed | Egyedtípus | Egyed-előfordulás |
| Tulajdonság | Tulajdonságtípus | Tulajdonság-előfordulás |
| kapcsolat | kapcsolattípus | Kapcsolat-Előfordulás |

## Az adatmodell és az adatbázis fogalma (2.5)

**Koncepcionális adatmodell (séma)**

A **koncepcionális adatmodell** (más néven **séma**) az adatbázis tervezésének egy magas szintű, elméleti megközelítése, amely a valós világban előforduló elemek és azok kapcsolatai között definiálja a szerkezetet. Az adatmodell **végleges tervezési alapot** ad, amely a következőket tartalmazza:

* **Véges számú tulajdonságtípus:** Azok a jellemzők, amelyek egy-egy egyed típushoz tartoznak, és az adatokat leírják. Például a hallgató egyedtípushoz tartozhatnak olyan tulajdonságok, mint név, évfolyam, szak stb.
* **Véges számú egyedtípus:** A valós világból származó, modellezett elemek csoportjai. Például: hallgató, tantárgy, autó, stb.
* **Véges számú kapcsolattípus:** A különböző egyedtípusok közötti kapcsolatok vagy viszonyok. Például egy hallgató felveszi a tantárgyat, vagy egy autó tulajdonosa van egy személynek.

Ezek az elemek mind **abstrakt** (elméleti) szinten jelennek meg a koncepcionális adatmodellben, és az adatbázis tényleges megvalósítása során lesznek implementálva.

**Adatbázis**

Az **adatbázis** a **koncepcionális adatmodell** (séma) alapján készült, de nem csak az adatmodell struktúráját tartalmazza, hanem az adott rendszerben tárolt **adatokat** is. Az adatbázis az alábbi elemek összessége:

1. **Adatmodell:** A valós világ elemeit leíró struktúra, amely a sémát adja.
2. **Egyed-előfordulások:** A konkrét példányok, amelyek a különböző egyedtípusok egyedjeit tartalmazzák. Például: "Kovács Péter" a hallgató egyede, "INDK501" a tantárgy egyede.
3. **Tulajdonságelőfordulások:** Az egyedekhez tartozó konkrét értékek. Például Kovács Péter neve, évfolyama vagy a tantárgy kreditszáma.
4. **Kapcsolat-előfordulások:** A kapcsolattípusok tényleges előfordulásai. Például Kovács Péter és az Adatbázisrendszerek tantárgy közötti **felvételi viszony**.

A **koncepcionális adatmodell** tehát a tervezett struktúrák **elméleti** leképezése, míg az **adatbázis** a **valós** adatokat és azok kapcsolatait tartalmazó implementáció.

## A tulajdonságtípusok (attribútumok) osztályozása (2.6)

A tulajdonságok (attribútumok) osztályozása az adatmodellezésben fontos szerepet játszik, mivel meghatározza, hogyan tároljuk és kezeljük az adatokat az adatbázisban. A következő szempontok szerint osztályozhatjuk őket:

**1. A tulajdonság-előfordulás szerkezete (összetettsége) szerint:**

* **Egyszerű (atomi) tulajdonság:**
  + Az egyszerű vagy atomi tulajdonság olyan adat, amely nem bontható tovább. Az értéke egyetlen, alapvető információt tartalmaz.
  + Példák:
    - **Hallgató neve** (pl. Kovács Péter)
    - **Autó rendszáma** (pl. IHJ-818)
    - **Tantárgy kreditszáma** (pl. 5)
* **Összetett tulajdonság:**
  + Az összetett tulajdonság több egyszerű attribútumból áll, és ezek együtt alkotják a teljes adatot. Az összetett tulajdonságok további részekre bonthatók.
  + Példák:
    - **Cím**: Az egyes címek (pl. utca, város, irányítószám) kombinációja alkotja az összetett címet.
    - **Személy neve**: Az egyszerű részek (keresztnév, vezetéknév) alkotják a teljes nevet.
    - **Dátum**: Nap, hónap, év kombinációja alkotja a dátumot.

**A tulajdonság-előfordulás hány értéket vehet föl egyszerre:**

* **Egyértékű tulajdonság:**
  + Az egyértékű tulajdonságok olyan attribútumok, amelyek egyetlen értéket vehetnek fel. Az ilyen tulajdonságoknak minden egyedhez csak egy érték tartozik.
  + Példák:
    - **Hallgató neve** (Kovács Péter)
    - **Autó rendszáma** (IHJ-818)
* **Halmazértékű (többértékű) tulajdonság:**
  + A halmazértékű tulajdonságok olyan attribútumok, amelyek egyszerre több értéket is felvehetnek. Egy egyedhez több érték is társulhat.
  + Példák:
    - **Hallgató felvett tantárgyai** (Adatbázisrendszerek, Magas szintű programozási nyelvek 2)
    - **Autó típusai** (SUV, Sportkocsi)
    - **Személy telefonszámai** (06-30-123456, 06-20-987654)

**A tulajdonság-előfordulás minden esetben megjelenik-e a háttértárolón (a fizikai adatbázisban):**

* **Tárolt tulajdonság:**
  + A tárolt tulajdonságok azok, amelyeket az adatbázis fizikailag tárol. Az értékek közvetlenül mentésre kerülnek a háttértárolóra.
  + Példák:
    - **Hallgató neve**
    - **Autó rendszáma**
    - **Tantárgy kreditszáma**
* **Származtatott tulajdonság:**
  + A származtatott tulajdonságok olyan attribútumok, amelyek nem kerülnek tárolásra, hanem valamilyen számítás vagy más adat alapján keletkeznek. Az értékek nem kerülnek fizikailag tárolásra, hanem a rendszer képes őket dinamikusan előállítani.
  + Példák:
    - **Hallgató életkora** (születési dátum alapján számítva)
    - **Autó kora** (gyártási év és a jelenlegi év különbsége alapján)
    - **Tantárgy átlagos pontszáma** (egyes hallgatók eredményeinek átlagolásával)

## A NULL érték mint tulajdonság-előfordulás (2.7)

A **NULL** érték az adatbázisokban speciális értékként szerepel, amelyet különböző helyzetekben alkalmaznak. A NULL nem egy üres értéket vagy zérót jelent, hanem egy **hiányzó, ismeretlen, nem alkalmazható vagy nem értelmezett** adatot. Az adatbázisokban a NULL lehetőséget biztosít arra, hogy az adatokban szereplő információk hiányosságait, ismeretlenségét vagy nem relevanciáját megfelelően kezeljük.

**A NULL különböző jelentései:**

1. **Nem alkalmazható, nem értelmezett**:
   * A NULL érték akkor alkalmazható, amikor egy adott tulajdonság nem releváns az adott egyed számára. Ilyen esetekben a tulajdonság értéke nem alkalmazható, tehát nem létezik értelmezett adat a mezőben.
   * **Példa**: Ha egy személy nem rendelkezik diplomával, a **Diploma** attribútum értéke nem értelmezett vagy NULL.
2. **Ismeretlen**:
   * A NULL érték arra is utalhat, hogy az érték ismeretlen vagy még nem került meghatározásra.
   * **Példa**: Egy személy magassága ismeretlen lehet, ezért a **Magasság** attribútum értéke NULL.
3. **Létezik, de hiányzik**:
   * A NULL arra is vonatkozhat, hogy az attribútum értéke létezne, de valamilyen okból hiányzik. Ilyenkor az adatot nem töltötték ki, de maga az adat az adatbázisban elképzelhető, hogy jelen lenne, ha rendelkezésre állna.
   * **Példa**: A tantárgyak esetén a **Kreditszám** attribútum értéke lehet, hogy létezik, de nincs megadva minden esetben (pl. választható tantárgyaknál).
4. **Nem tudjuk, hogy létezik-e**:
   * A NULL érték egy másik gyakori alkalmazása, amikor az adat létezését sem tudjuk biztosan. Ilyenkor az érték nem ismert, és az adatbázisban nem található egyértelmű válasz arra, hogy létezik-e vagy sem.
   * **Példa**: Egy jármű **Hengerűrtartalom** attribútuma nem biztos, hogy létezik (például, ha nem minden jármű esetében van adat a motor méretéről).

## ****A kapcsolattípusok osztályozása (2.8)****

A kapcsolattípusok osztályozása segít meghatározni, hogyan viszonyulnak egymáshoz az adatbázisban szereplő egyedtípusok. Az ER (Entity-Relationship) modellben három fő szempont szerint osztályozhatjuk őket: a kapcsolat foka, a kapcsolat számossága, és a kapcsolat szorossága.

**1. A kapcsolat foka**

A kapcsolat foka meghatározza, hogy hány egyedtípus vesz részt a kapcsolatban. Különböző fokú kapcsolatok léteznek, attól függően, hogy hány entitás vesz részt a kapcsolatban.

* **Bináris kapcsolat (másodfokú)**: Két egyedtípus között létrejövő kapcsolat.
  + **Példa**: A **hallgató és tantárgy** közötti kapcsolat, ahol a hallgató felvesz egy tantárgyat. Ez egy bináris kapcsolat, mert csak két egyedtípus, a hallgató és a tantárgy szerepel benne.
* **Ternáris kapcsolat (harmadfokú)**: Három egyedtípus között létrejövő kapcsolat.
  + **Példa**: A **bank, ügyfél és számla** közötti kapcsolat. Egy ügyfél lehet több számlával is rendelkezni, és a bank kezeli az ügyfél számláit, így három egyedtípus vesz részt a kapcsolatban.
* **N-es kapcsolat**: Több egyedtípus is részt vehet a kapcsolatban (n > 3).
  + **Példa**: Bonyolultabb üzleti vagy tudományos modellek, amelyek több entitást kapcsolnak össze (például egy projekt, amelyben részt vesznek a munkavállalók, az eszközök és a különböző osztályok).

**2. A (bináris) kapcsolat számossága**

A kapcsolat számossága meghatározza, hogy legfeljebb hány kapcsolat-előfordulásban vehet részt egy egyed-előfordulás. A számosság három típusra osztható:

* **1:1 kapcsolat**: Egy egyed csak egy másik egyedhez kapcsolódhat.
  + **Példa**: A **házasság** kapcsolata Magyarországon: egy személyhez egy másik személy tartozik, tehát egy házasság 1:1 kapcsolatot jelent.
* **1:N kapcsolat**: Egy egyedhez több másik egyed kapcsolódhat.
  + **Példa**: A **tulajdonlás** kapcsolat az autó és a személy között. Egy személynek több autója is lehet, de egy autó egy személyhez tartozik, tehát 1:N kapcsolat áll fenn.
* **M:N kapcsolat**: Egy egyedhez több másik egyed is kapcsolódhat, és viszont.
  + **Példa**: A **hallgató és tantárgy** közötti **tantárgyfelvétel** kapcsolat. Egy hallgató több tantárgyat is felvehet, és egy tantárgyat több hallgató is felvehet, tehát M:N kapcsolat áll fenn.

**3. A (bináris) kapcsolat szorossága**

A kapcsolat szorossága határozza meg, hogy a kapcsolatban részt vevő egyedtípusok minden egyedének részt kell-e vennie legalább egy kapcsolat-előfordulásban.

* **Kötelező szorosság**: Az egyedeknek minden esetben részt kell venniük a kapcsolatban.
  + **Példa**: A **tulajdonlás** kapcsolat az autó és az autótulajdonos között kötelező szorosságú. Minden autóhoz tartozik egy tulajdonos, tehát minden autó részt kell, hogy vegyen a kapcsolatban.
* **Félig kötelező szorosság**: Az egyik egyed típusának minden esetben részt kell venni a kapcsolatban, de a másik egyed típusa opcionális.
  + **Példa**: A **tulajdonlás** kapcsolat az autó és a személy között félig kötelező szorosságú. Minden autó egyednek részt kell vennie egy kapcsolat-előfordulásban (mert minden autónak kell, hogy legyen tulajdonosa), de nem minden személynek kell autótulajdonosnak lennie.
* **Opcionális szorosság**: Az egyedek nem kötelesek részt venni a kapcsolatban.
  + **Példa**: A **könyv és könyvtártag** közötti **kölcsönzési viszony** opcionális szorosságú. Nem minden könyvtártag kölcsönöz könyvet, tehát nem kötelező, hogy minden könyvtártag kapcsolatba lépjen a könyvekkel.

## ER diagram (2.9) A képen szöveg, képernyőkép, diagram, Betűtípus látható Automatikusan generált leírás

A képen szöveg, vázlat, rajz, Betűtípus látható

Automatikusan generált leírás

A képen szöveg, Betűtípus, képernyőkép, diagram látható

Automatikusan generált leírás

## Tartomány fogalma (3.1)

Egy **D tartomány** atomi értékek egy halmaza.

**Jellemzői:**

* **Név**
* **Adattípus**
* **Formátum**
* **Korlátozás**
* **További információk az értelmezéshez**

**Példák:**

* **Mobiltelefonszámok**: 11 decimális számjegy, +dd-dd-dddddd formátumban.
* **Személyi számok**: 11 decimális számjegy, d dddddd dddd formátumban.
* **Nevek**: Tetszőleges hosszúságú karaktersorozat.
* **Tömegek**: Nem negatív valós szám, mértékegységgel (font vagy kg).

## Relációséma (3.2)

A **relációséma** alatt az **R(A1, A2, …, An)** jelölést értjük, ahol:

* **R** a relációséma neve,
* **A1, A2, …, An** pedig az attribútumok.

Minden **Ai** attribútum egy szerepkör neve, amelyet valamely **D tartomány** játszik. A **D**-t az **Ai**attribútum tartományának nevezzük, és **dom(Ai)**-val jelöljük.

**Példák:**

* **HALLGATÓ** (Név, Személyi\_szám, Lakcím, Szak, Évfolyam, Neptun\_kód)
* **TANSZÉK** (Tanszék\_név, Tanszékvezető\_neve, Kar)
* **AUTÓ** (Márka, Típus, Gyártási\_év, Motorszám, Rendszám)

**Megjegyzés:**

* Előfordulhat, hogy több attribútumnak is azonos a tartománya.
* Az attribútumok különböző szerepköreit, interpretációit jelölik ki a tartományoknak.

## Reláció (3.3)

A **reláció** egy adott relációséma (például **R(A1, A2, …, An)**) alapján készült adatokat tartalmazó halmaz. Ezt a halmazt **r**-nek nevezzük, és az elemei **n-esek**, azaz rendezett értéksorozatok. Az elemeket gyakran **rekordoknak** hívjuk.

* A reláció **r** például így nézhet ki:  
  **r = {t1, t2, …, tm}**, ahol **t1, t2, ..., tm** a rekordok.

Minden rekord egy **n-es**, ami egy rendezett lista, amely **n** értéket tartalmaz. Például:  
**ti = <v1, v2, …, vn>**, ahol:

* **vj** értékek (1 ≤ j ≤ n) az adott attribútum **dom(Aj)**-ának elemei lehetnek, vagy **NULL** értéket is tartalmazhatnak, ha az adat hiányzik.

**Megjegyzések:**

* A **rekordok** tulajdonképpen a definícióban említett **n-esek**.
* Ha egy rekordban az **Aj** attribútumhoz tartozó értékre van szükség, akkor arra **t[Aj]**-vel (vagy röviden **t[j]**-vel) hivatkozhatunk.

Emberibb változat: A **reláció** egy olyan adatgyűjtemény, amely egy adott struktúrában van elrendezve. A relációséma (például **R(A1, A2, …, An)**) egyfajta "tervet" ad arra, hogy milyen típusú adatokat várunk és hogyan kell kinézniük. A reláció pedig az adatok tényleges példányait tartalmazza, egy sor rekordban.

* Minden **rekord** egy sor adatot jelent, amely a relációséma meghatározott attribútumaihoz (például név, cím, életkor) tartozó értékekből áll. Ezeket az adatokat **n-eseknek** hívjuk, mivel egy sorban **n** érték található, és ezek az értékek egy meghatározott sorrendben szerepelnek.

A képen szöveg, képernyőkép, sor, Betűtípus látható

Automatikusan generált leírás

**Megjegyzések**

* A reláció **aktuális állapota** csak azokat a rekordokat tartalmazza, amelyek az adott pillanatban a valós világ állapotát tükrözik. Más szóval, a reláció adatai mindig a jelenlegi helyzetet, eseményeket vagy információkat jelenítik meg. Ahogy a valós világ változik, úgy változik a reláció is: új rekordok kerülhetnek be, régi rekordok eltűnhetnek, vagy módosulhatnak.
* A **reláció sémája**, ami meghatározza az adatstruktúra formáját (például a lehetséges attribútumokat), általában **stabil** és nem változik gyakran. Ritka kivételektől eltekintve, a séma maga viszonylag állandó, és csak akkor módosul, ha valóban szükséges, például új típusú adatokat kell hozzáadni.

## A relációs modell megszorításai (3.4)

Az adatmodell megszorításait három fő csoportba sorolhatjuk:

1. **Modell alapú, implicit megszorítások**:  
   Ezek a megszorítások az adatmodell természetéből fakadnak, és automatikusan érvényesek, anélkül, hogy kifejezetten meghatároznánk őket. Például a relációs modellben a rekordok sorrendje nem számít, tehát a rekordoknak nem kell meghatározott sorrendben szerepelniük.
2. **Séma alapú, explicit megszorítások**:  
   Ezek a megszorítások közvetlenül a **séma** részeként vannak kifejezve, tehát az adatmodell szerkezetében határozzák meg, hogy mi megengedett és mi nem. Ilyenek például az adattípusok, az attribútumok kötelező volta (nem lehet üres érték), vagy a kulcsok, amelyek meghatározzák, hogy egy rekord egyedisége hogyan biztosítható.
3. **Alkalmazás alapú, szemantikus megszorítások (üzleti szabályok)**:  
   Olyan szabályok, amelyeket nem lehet közvetlenül az adatmodell sémájában kifejezni, ezért az alkalmazói programokban kell őket definiálni és betartatni. Ezek a szabályok a valós világot modellezik, és gyakran üzleti vagy jogi előírások alapján kerülnek meghatározásra. Például egy üzleti szabály lehet, hogy egy alkalmazott csak akkor kérhet szabadságot, ha már ledolgozott legalább 6 hónapot.

## Séma alapú explicit megszorítások (3.5)

A **séma alapú megszorítások** azok a szabályok, amelyek közvetlenül a relációséma részeként vannak definiálva, és az adatmodell struktúrájában meghatározzák, hogy milyen adatokat lehet tárolni. Néhány példa ezekre a megszorításokra:

* **Tartománymegszorítások**:  
  Ezek meghatározzák, hogy egy adott attribútum milyen típusú és értékű adatokat tartalmazhat. Például egy telefonszámnak csak számjegyekből kell állnia, és lehet, hogy egy meghatározott formátumban kell szerepelnie (pl. +36-20-1234567).
* **Kulcsmegszorítás és a NULL értékekre vonatkozó megszorítás**:
  + A **kulcsmegszorítás** szerint egy relációsémának mindig rendelkeznie kell elsődleges kulccsal.
  + **NULL értékekre vonatkozó megszorítás** meghatározza, hogy egy adott attribútum értéke lehet-e **NULL** (azaz, hiányzó adat). Bizonyos attribútumoknak mindig rendelkezniük kell érvényes értékkel, míg másoknak engedélyezett lehet a NULL érték.
* **Egyedintegritási megszorítások**:  
  Az egyedintegritási megszorítás kimondja, hogy egyetlen elsődleges kulcsérték sem lehet NULL érték. Ha az elsődleges kulcs összetett, akkor annak egyik komponense sem lehet NULL érték.
* **Hivatkozásintegritási megszorítások**: A **hivatkozásintegritási megszorítás** két táblát kapcsol össze úgy, hogy biztosítja a köztük lévő adatok összhangját és konzisztenciáját. Az adattípusok egyezése: A külső kulcs mezőinek adattípusai meg kell, hogy egyezzenek a hivatkozott tábla elsődleges kulcsának adattípusaival. Érvényes értékek: A külső kulcs értékei vagy: Meg kell, hogy jelenjenek a másik tábla elsődleges kulcsának értékei között, vagy NULL értékűek lehetnek, ha nincs hivatkozás.

külső kulcs: egy táblában olyan mező vagy mezők csoportja, amely egy másik tábla elsődleges kulcsára mutat.

## Szuperkulcs (3.6)

A **szuperkulcs** fogalma az adatbázisokban azt jelenti, hogy egy adott attribútumhalmaz képes az egyes rekordok egyediségét biztosítani egy relációban. Vagyis a szuperkulcs egy olyan attribútumokból álló halmaz, amely garantálja, hogy két rekord nem rendelkezhet azonos értékekkel a kiválasztott attribútumokban.

**Triviális szuperkulcs**

Minden relációnak létezik legalább egy **szuperkulcsa**, és ez mindig az összes attribútumot tartalmazó halmaz, amit **triviális szuperkulcsnak** nevezünk. Például, ha egy **Hallgató** relációséma tartalmazza a **Neptun kódot**, **Nevet**, és **Személyi számot**, akkor a **{Neptun kód, Név, Személyi szám}** halmaz is szuperkulcs, mivel ez az összes attribútumot tartalmazza, és így minden rekord egyedisége garantált.

**Kulcs**

A **kulcs** fogalma az adatbázisokban a szuperkulcshoz képest egy szűkebb fogalom. A kulcs olyan szuperkulcs, amelyben **nincs felesleges attribútum**, azaz a lehető legkevesebb attribútumból áll, amelyek még mindig biztosítják a rekordok egyediségét. Más szóval, a kulcs egy minimális szuperkulcs, amelyből nem lehet eltávolítani egy attribútumot anélkül, hogy az egyediség ne sérülne.

A kulcsnak két fő feltételnek kell megfelelnie:

1. **Egyediség**: Bármilyen relációban a két különböző rekordnak nem lehetnek azonosak a kulcsban szereplő attribútumok értékei.
2. **Minimális szuperkulcs**: A kulcs az egyetlen szuperkulcs, amelyből nem lehet eltávolítani egy attribútumot úgy, hogy az egyediség továbbra is biztosítva maradjon.

**Kulcs típusai:**

* **Egyszerű kulcs**: Ha egy kulcs **csak egyetlen attribútumból** áll. Például a **Neptun kód** önálló kulcsként egy egyszerű kulcs. **Összetett kulcs**: Ha a kulcs **több attribútumból** áll. Például a **{Személyi szám, Név}** egy összetett kulcs, ha ezek együtt biztosítják a rekordok egyediségét, de egyik attribútum önállóan nem.

## Kulcsjelölt (3.7)

A **kulcsjelölt** olyan attribútumhalmaz, amely alkalmas arra, hogy kulcs legyen egy relációban. Más szóval, a kulcsjelöltek azok a potenciális kulcsok, amelyek képesek biztosítani a rekordok egyediségét a relációban. Egy relációsémának több kulcsjelöltje is lehet, tehát több lehetséges attribútumhalmaz is létezik, amelyek közül valamelyik végül a kulcsként választható.

#### **Elsődleges kulcs**

Az **elsődleges kulcs** a kulcsjelöltek közül az a kulcs, amelyet az adatbázis-modellező választ ki a relációsémában, és amelyet a rekordok azonosítására használ. Az elsődleges kulcsot az adatbázisban minden rekord számára egyedileg azonosítóként alkalmazzuk, ezért fontos, hogy valóban biztosítsa a rekordok egyediségét.

**Elsődleges attribútum**

* Az R relációséma egy attribútumát **elsődleges attribútumnak** nevezzük, ha az eleme R valamely kulcsjelöltjének.
* Egy attribútumot **másodlagos (leíró) attribútumnak** hívunk, ha nem elsődleges attribútum, azaz nem eleme egyetlen kulcsjelöltnek sem.

## Egyéb típusú megszorítások (3.8)

Az **egyéb megszorítások** az adatbázisokban olyan speciális szabályokat foglalnak magukban, amelyek nem tartoznak a séma- vagy hivatkozásintegritási megszorításokhoz. Ezek a megszorítások az adatok közötti logikai vagy üzleti szabályokat írják le, illetve a változásokra vonatkozó feltételeket határozzák meg.

**1. Szemantikus integritási megszorítások**

A szemantikus megszorítások az adatok tartalmára vonatkozó **üzleti logikát** fejezik ki. Ezek általában az alkalmazás specifikus követelményeiből erednek, és az adatok közötti kapcsolatokra vagy korlátokra épülnek.

**Példák:**

* **Dolgozó fizetése nem lehet nagyobb a főnökénél**: Ez azt biztosítja, hogy a hierarchikus bérstruktúra ne sérüljön az adatbázisban.
* **Egy dolgozó egy héten legfeljebb 56 órát dolgozhat egy projekten**: Ez az adatbázisba rögzíti a munkaidővel kapcsolatos szabályokat.

**2. Adatok közötti függések**

Az adatok közötti függések szabályokat definiálnak az attribútumok közötti kapcsolatokra vonatkozóan.

**Típusai:**

* **Funkcionális függés**: Ha az egyik attribútum (vagy attribútumhalmaz) értéke egyértelműen meghatározza a másik attribútum értékét. Példa: Egy dolgozó azonosítója egyértelműen meghatározza a dolgozó nevét.
* **Többértékű függés**: Egy attribútum (vagy attribútumhalmaz) értékei több másik attribútum értékhalmazát is meghatározhatják egymástól függetlenül. Példa: Egy hallgatónak több telefonszáma és e-mail címe is lehet, amelyek egymástól függetlenek.

**3. Átmenet megszorítás**

Az átmenet megszorítások az adatok **változásainak irányát vagy szabályait** írják elő. Ez nem az adatok aktuális állapotát, hanem a változások korlátozását célozza meg.

**Példa:**

* **A dolgozó fizetése csak nőhet**: Ez azt biztosítja, hogy egy fizetésmódosítás soha ne csökkentse a dolgozó bérét.

Az ilyen típusú megszorításokat szintén nehéz az adatbázis-kezelő rendszeren belül kezelni, ezért gyakran az alkalmazási logikában implementálják őket.

## Relációs adatbázisséma és relációs adatbázis (3.9)

**Relációs adatbázisséma**

Az adatbázisséma az adatbázis **terve** vagy **váza**, amely meghatározza, hogy milyen táblák (relációk) vannak benne, ezek milyen adatokat (attribútumokat) tárolnak, és milyen szabályokat (integritási megszorításokat) kell betartani az adatok érvényességéhez.

Relációs adatbázisA képen szöveg, képernyőkép, Betűtípus, diagram látható

Automatikusan generált leírás

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírásAz adatbázis a konkrét adatok gyűjteménye, amelyek az adatbázisséma által meghatározott táblákban vannak elrendezve. Ezek az adatok bármikor változhatnak (pl. új hallgató kerül felvételre, valaki szakot vált).

## Gyenge és erős egyedtípus (4.1)

**Erős egyedtípus**

Egy egyedtípus **erős**, ha rendelkezik önállóan az egyedek azonosítására alkalmas **kulcs attribútummal**. Ez azt jelenti, hogy az egyedek egyedileg azonosíthatók az adott egyedtípuson belül, más egyedtípusokra való hivatkozás nélkül.

**Gyenge egyedtípus**

Egy egyedtípus **gyenge**, ha nem rendelkezik önálló azonosításra alkalmas kulcs attribútummal, és így az egyedek azonosításához egy másik (erős) egyedtípushoz való kapcsolódás szükséges. A gyenge egyedtípus azonosításához:

1. Szükséges a **tulajdonos egyedtípus** kulcs attribútuma(i).
2. Szükséges a gyenge egyedtípus saját attribútuma(i) közül legalább egy.

**Kapcsolódás a tulajdonoshoz**

A gyenge egyedtípus mindig **totális résztvevője** az őt azonosító kapcsolatnak. Ez azt jelenti, hogy a gyenge egyed csak akkor létezhet, ha egyértelműen kapcsolódik egy tulajdonos egyedhez. A gyenge egyedtípust az erős tulajdonos egyedtípussal összekötő kapcsolatot **azonosító kapcsolatnak** nevezzük.

**Létezésfüggőség**

Egy gyenge egyed létezése attól függ, hogy van-e hozzá rendelve egy tulajdonos egyed. Ha nincs tulajdonos, a gyenge egyed nem azonosítható.

## Absztrakt lekérdező nyelvek (4.2)

**Relációalgebra**

* A matematikai halmazelméleten alapuló lekérdező nyelv.
* A lekérdezés egy kifejezés, amelyben:
  + az operátorok relációalgebrai műveletek,
  + az operandusok pedig relációk.
* A lekérdezés eredménye szintén egy reláció.

**Relációkalkulus**

* Az elsőrendű predikátumkalkulusra épülő lekérdező nyelv.
* A lekérdezés egy speciális alakú kifejezés, amely egy halmazt definiál.
* A lekérdezés eredménye az előbb említett halmaz által meghatározott reláció.

**A relációalgebra műveletei**

**Alapműveletek:**

* **Szelekció (σ)**
* **Projekció (π)**
* **Átnevezés (ρ)**

**Halmazműveletek:**

* **Unió (∪)**
* **Metszet (∩)**
* **Különbség (−)**

**Egyéb műveletek:**

* **Descartes-szorzat (belső szorzat, ×)**
* **Összekapcsolás (join)**
  + **Általános összekapcsolás (theta join, θ)**
  + **Egyenlőségalapú összekapcsolás (equijoin)**
  + \**Természetes összekapcsolás (natural join, )*
* **Külső összekapcsolások:**
  + **Bal oldali külső összekapcsolás (left outer join)**
  + **Jobb oldali külső összekapcsolás (right outer join)**
  + **Teljes külső összekapcsolás (full outer join)**
* **Hányados (÷)**

## Szelekció (WHERE) (σ) (4.3)

A szelekció egy relációalgebrai művelet, amely egy relációból kiválasztja azokat a rekordokat, amelyek megfelelnek egy adott szelekciós feltételnek.

**Általános alak:**

σ szelekciós\_feltétel (R)

* **R**: Az a reláció, amelyből a rekordokat kiválogatjuk.
* **Szelekciós feltétel**: Egy logikai kifejezés, amely részkifejezésekből épül fel.

**A részkifejezések formája:**

1. attribútum hasonlító\_operátor konstans
2. attribútum hasonlító\_operátor attribútum

* **Attribútum**: Az R egy attribútumának neve.
* **Hasonlító operátor**: Az alábbi operátorok egyike:
  + =, ≠, <, ≤, >, ≥
* **Konstans**: Az attribútum tartományából származó érték.

**Összetett szelekciós feltételek:**

* A részkifejezéseket logikai operátorokkal kapcsolhatjuk össze:
  + és (∧)
  + vagy (∨)
  + nem (¬)

**Megjegyzések**

1. A =, ≠, <, ≤, >, ≥ operátorok csak akkor alkalmazhatók, ha az attribútumok tartományai összehasonlítható elemeket tartalmaznak (pl. számok, dátumok).
   * Ha az attribútum tartományában nem összehasonlítható értékek szerepelnek (pl. színek), akkor csak a =, ≠ operátorok használhatók.
2. Bizonyos tartományok esetében további hasonlító operátorok is elérhetők, például RÉSZ-SZTRING operátor.

**A szelekció tulajdonságai**

* **Unáris művelet**: Egyetlen reláción működik.
* Az eredményül kapott reláció:
  + Foka és sémája megegyezik az R fokával és sémájával.
  + Számossága kisebb vagy egyenlő R számosságával.
* **Felcserélhetőség**:
  + Két egymásba ágyazott szelekciós művelet végrehajtási sorrendje felcserélhető.
* **Kaszkádolt szelekciók egyszerűsítése**:
  + Minden egymásba ágyazott szelekció átírható egyetlen szelekcióvá, amelynek feltétele az eredeti feltételek konjunkciója (és művelet).

## A projekció (SELECT) tulajdonságai (4.4)

**1. Unáris művelet**

A projekció csak **egy bemeneti reláción** működik, tehát unáris művelet.

**2. Az eredményül kapott reláció foka és sémája**

* Az **eredmény sémáját** az **attribútumlistában megadott attribútumok** határozzák meg.
* Az **attribútumok sorrendje** az eredményben megegyezik az attribútumlistában megadott sorrenddel.
* A reláció **foka** (az attribútumok száma) az attribútumlistában szereplő attribútumok darabszáma lesz.

**3. Számosság csökkenése vagy megőrzése**

* Ha az **attribútumlista nem tartalmaz kulcs attribútumot**, akkor:
  + Az eredményben **duplikált rekordok nem szerepelhetnek**, ezért az eredményül kapott reláció **számossága** (rekordjainak száma) kisebb lehet, mint az eredeti relációé (R).
* Ha az **attribútumlista szuperkulcsa** az R-nek, akkor:
  + Az eredmény reláció **számossága megegyezik** az eredeti relációéval (R).

**4. Egymásba ágyazott projekciók**

* Ha két egymásba ágyazott projekciót hajtunk végre, akkor:
  + Az eredmény **megegyezik a külső projekció eredményével**, ha az első projekció attribútumlistája részhalmaza a második projekció attribútumlistájának (**lista1 ⊆ lista2) = lista1**

## Átnevezés (ρ) (4.5)

Az átnevezés lehetőséget ad arra, hogy megváltoztassuk a reláció nevét, illetve az attribútumok neveit.

**Átnevezés szintaxisa:**

* **ρ S(B₁, B₂, ..., Bₙ)(R)**
  + **S**: Az új reláció neve (jelölése).
  + **B₁, B₂, ..., Bₙ**: Az új attribútumnevek, amelyeket az eredeti reláció attribútumaihoz rendelünk.
  + **R**: Az eredeti reláció, amelyet átnevezünk.

Alternatív szintaxisok:

* **ρ S(R)**: Csak a reláció nevét változtatjuk meg, de az attribútumok nevei változatlanok maradnak.
* **ρ (B₁, B₂, ..., Bₙ)(R)**: Az attribútumok nevei változnak, de a reláció neve nem.

**Az átnevezés tulajdonságai**

1. **Unáris művelet**
   * Az átnevezés egy **unáris művelet**, mert csak egyetlen bemeneti relációval dolgozik.
2. **A reláció foka és számossága**
   * Az eredményül kapott reláció **foka** (attribútumok száma) és **számossága** (rekordok száma) megegyezik az eredeti relációval (R).
   * Az átnevezés nem változtatja meg a relációban található rekordokat, csak a nevüket vagy az attribútumaik nevét módosítja.
3. **A séma változása**
   * Ha **B₁, B₂, ..., Bₙ** attribútumokat adunk meg az átnevezés során, akkor az eredmény sémája ezekkel az új nevekkel lesz meghatározva.
   * Ha nem adunk meg új attribútumneveket, akkor a séma megegyezik az eredeti reláció sémájával, és csak a reláció neve változik.

## Uniókompatibilitás (4.6)

Két reláció akkor **uniókompatibilis**, ha az alábbi két feltétel teljesül:

1. **Azonos fokszám**
   * A két relációnak ugyanannyi attribútuma kell, hogy legyen.
   * Azaz: a relációk **foka** (attribútumok száma) megegyezik.
2. **Attribútumok tartományainak megegyezése**
   * Minden egyes attribútum pár esetén az attribútumok tartományai azonosak kell legyenek.

**Unió, Metszet, Különbség**

**Definíció:** Legyen **R** és **S** két uniókompatibilis reláció.

* **R és S uniója (R ∪ S)**: egy olyan reláció, mely tartalmazza az összes olyan rekordot, mely **vagy** **R** **vagy** **S** relációhoz tartozik (az ismétlődőkből csak egyet).
* **R és S metszet (R ∩ S)**: egy olyan reláció, mely tartalmazza az összes olyan rekordot, mely **R** és **S** relációhoz is tartozik.
* **R és S különbsége (R - S)**: egy olyan reláció, mely tartalmazza az összes olyan **R**-beli rekordot, mely **S**-ben nem található meg.

**Kommutativitás:**

* Az **unió** és a **metszet** műveletek **kommutatívak**, ami azt jelenti, hogy az operandusok sorrendje nem befolyásolja az eredményt.
  + **Unió:** R ∪ S = S ∪ R
  + **Metszet:** R ∩ S = S ∩ R

**Asszociativitás:**

* Az **unió** és a **metszet** műveletek **asszociatívak**, vagyis az operandusok csoportosítása nem változtatja meg az eredményt.
* **Unió:** ( R ∪ S ) ∪ T = R ∪ ( S ∪ T )
* **Metszet:** ( R ∩ S ) ∩ T = R ∩ ( S ∩ T )

A **különbség** művelet **általában nem kommutatív**, vagyis az operandusok sorrendje befolyásolja az eredményt.

* **Különbség:** R−S ≠ S−R

## ****Descartes-szorzat, Belső szorzat (4.7)****

A **Descartes-szorzat** (más néven **belső szorzat**) két reláció összekapcsolására szolgáló művelet. Az alábbiakban ismertetem a műveletet és annak tulajdonságait.

**Általános alak:**

Legyenek **R(A₁, A₂, ..., Aₙ)** és **S(B₁, B₂, ..., Bₘ)** két reláció, akkor a Descartes-szorzat az alábbiak szerint van definiálva: R × S

**Művelet ismertetése:**

* **Két tetszőleges sémájú reláció** között elvégezhető bináris művelet.
* **Eredmény:** Az eredményül kapott reláció **Q** egy **n + m** fokszámú reláció, amelynek a sémája az alábbiak szerint alakul: Q(A1,A2,...,An,B1,B2,...,Bm)
* **Számosság:** Ha **|R|** és **|S|** az eredeti két reláció számosságát jelöli, akkor az eredményül kapott reláció számossága: ∣R×S∣ = ∣R∣×∣S∣
* **Művelet eredménye:** Az eredményül kapott reláció minden rekordja tartalmazza az eredeti két reláció minden rekordjának összes lehetséges kombinációját. Azaz, minden **R** rekordot kombinálunk minden **S** rekorddal.

## ****Általános Összekapcsolás (Join) (4.8)****

Az **általános összekapcsolás** (vagy **join**) egy olyan bináris művelet, amely két reláció összekapcsolását végzi egy meghatározott feltétel alapján.

**Művelet általános alakja:**

Legyenek **R(A₁, A₂, ..., Aₙ)** és **S(B₁, B₂, ..., Bₘ)** két reláció, az összekapcsolás művelet általános alakja: R⋈ összekapcsolási feltétel ​S

**Művelet ismertetése:**

* **Bináris művelet:**  
  Az összekapcsolás egy **bináris művelet**, amely két relációra (R és S) van alkalmazva.
* **Eredmény:**  
  Az eredményül kapott reláció **Q** egy **n + m** fokszámú reláció, amelynek a sémája: Q(A1,A2,...,An,B1,B2,...,Bm)
* **Szűrés:** Az eredményül kapott relációban benne lesz az **R** és **S** relációk rekordjainak minden olyan kombinációja, amely kielégíti az összekapcsolási feltételt.

**Az összekapcsolási feltétel:**

Az összekapcsolási feltétel az **R** és **S** attribútumainak bizonyos kapcsolatát írja le, és lehet egy vagy több feltétel is, amelyek meghatározzák, hogy mikor kell egy rekordot összekapcsolni a két reláció között.

A csatlakozási feltétel általános alakja a következő: Ai Θ Bj

Ahol:

* **Aᵢ** az **R** reláció **i**-edik attribútuma.
* **Bⱼ** az **S** reláció **j**-edik attribútuma.
* **Ai** és **Bj** attribútumok tartományai megegyeznek (pl. mindkettő szám, dátum stb.).
* **Θ** egyike annak a {=, ≠, <, ≤, >, ≥} halmazba tartozó összehasonlító műveletnek, amelyet az összekapcsolási feltétel meghatározására használunk.

## ****Egyenlőségen Alapuló Összekapcsolás (Equijoin) (4.9)****

Az **egyenlőségen alapuló összekapcsolás** olyan általános összekapcsolási művelet, amelynek összekapcsolási feltételében kizárólag az **egyenlőségjel (=)** szerepel összehasonlító műveleti jelként.

**Definíció:**

Az **egyenlőségen alapuló összekapcsolás** (vagy **equijoin**) művelet az összekapcsolás egy speciális esete, amikor az összekapcsolás alapja az egyenlőség operátor. Az ilyen összekapcsolás eredményeként az összekapcsolt rekordok minden párja tartalmazza az azonos értéket az összekapcsolt attribútumokban.

**Tulajdonságok:**

* **Az összekapcsolás feltétele:** Az összekapcsolásban szereplő attribútumok tartalmának egyenlőségét kell figyelembe venni, azaz **Aᵢ = Bⱼ**, ahol **Aᵢ** az egyik reláció attribútuma, **Bⱼ** pedig a másik reláció attribútuma.
* **Eredmény:**  
  Az egyenlőségen alapuló összekapcsolás eredményeként kapott reláció minden rekordjában legalább egy pár azonos érték található az összekapcsolás alapját képező attribútumokban.

## Természetes Összekapcsolás (Natural Join) (4.10)

Az **egyenlőségen alapuló összekapcsolás** eredményeként kapott relációkban előfordulhatnak felesleges attribútumok, mivel az összekapcsolás alapjául szolgáló attribútumok tartalmazzák azonos értékeket. A **természetes összekapcsolás** ezen a problémán segít úgy, hogy az egyenlőségen alapuló összekapcsolásból származó relációkban eltávolítja azokat az **azonos nevű attribútumokat**, amelyek tartalmuk miatt ismétlődnének.

* **Az összekapcsolás alapjául szolgáló attribútumok nevei:** Az összekapcsolandó két relációban az összekapcsolás alapjául szolgáló attribútumok neveinek meg kell egyezniük, hogy a természetes összekapcsolás végrehajtható legyen.

**Természetes Összekapcsolás Jellemzői:**

1. **Sémazás:** Az eredményül kapott reláció sémája az eredeti két reláció sémájának attribútumait tartalmazza, de az összekapcsolás alapjául szolgáló attribútumok közül csak **páronként egyet** tartalmaz.
2. **Fokszám:** Az eredményül kapott reláció foka az eredeti két reláció fokszámának összegénél annyival kevesebb, ahány azonos nevű attribútum található mindkét relációban.
3. **Számosság:** Az eredményül kapott reláció számossága a két eredeti reláció számosságának szorzataként alakulhat, de az nem haladhatja meg az eredeti két reláció számosságának szorzatát. Az eredmény számossága tehát **0-tól az eredeti relációk számosságainak szorzatáig terjedhet**.

## A Relációalgebrai Műveletek Teljes Halmaza (4.11)

A relációalgebrai műveletek teljes halmaza azt jelenti, hogy az ezen műveletek segítségével bármely más relációalgebrai művelet kifejezhető. Ez a tulajdonság alapvető ahhoz, hogy a relációalgebrai rendszer rugalmas és kifejező legyen, mivel a különböző lekérdezési műveletek egyetlen halmazból levezethetők.

Bebizonyítható, hogy a relációalgebrai operátorok halmaza teljes halmaz, azaz bármelyik másik relációalgebrai művelet kifejezhető ezen operátorokkal végzett műveletek sorozataként. Ez azt jelenti, hogy egy adott műveletet, például szelekciót, projekciót, uniót, metszetet vagy bármely más műveletet ki lehet fejezni a relációalgebrai műveletek egy adott kombinációjával.

Ez a bizonyítás gyakran a következő műveletek segítségével történik, amelyek **teljes halmazt** alkotnak, tehát bármely más művelet kifejezhető velük:

1. **Szelekció (σ):** Rekordok szűrése a megadott feltétel alapján.
2. **Projekció (π):** A relációban szereplő attribútumok kiválasztása.
3. **Unió (∪):** Két reláció egyesítése, duplikált rekordok eltávolításával.
4. **Metszet (∩):** Két reláció közös rekordjainak kiválasztása.
5. **Különbség (−):** Az egyik relációból a másikban nem szereplő rekordok eltávolítása.
6. **Descartes-szorzat (×):** Két reláció minden lehetséges kombinációjának létrehozása.
7. **Összekapcsolás (Join):** Két reláció összekapcsolása egy közös attribútum alapján.

A relációalgebrai műveletek teljes halmaza lehetővé teszi bármely más művelet levezetését, például összekapcsolást, egyenlőségen alapuló összekapcsolást (equijoin), természetes összekapcsolást (natural join), stb.

## Osztás és Hányados műveletek (R ÷ S) (4.12)

**Osztás (÷) művelet**

Az **osztás** művelet a relációalgebrában egy olyan művelet, amely két relációt vesz figyelembe, és azok alapján hoz létre egy harmadik relációt. Az osztás művelet alapvetően akkor alkalmazható, ha a két reláció között van valamilyen típusú részleges összefüggés, amit a következőképpen definiálhatunk.

* **Jelölések:**
  + Jelöljük **R**-t az egyik relációval, amelynek sémája **Z**-t alkotó attribútumokból áll.
  + Jelöljük **S**-t a másik relációval, amelynek sémája **X**-et alkotó attribútumokból áll.

**Feltétel:**

Az osztás művelete akkor hajtható végre, ha az **S** sémáját alkotó attribútumok halmaza **X** **alhalmaza** az **R** sémáját alkotó attribútumok halmazának, azaz **X ⊆ Z**.

**Eredmény:**

Az osztás művelet eredménye egy új reláció, amely azokat a rekordokat tartalmazza, amelyek **R**-ben szerepelnek és azokhoz a rekordokhoz tartozó értékek minden lehetséges **S**-beli értékre is jelen vannak. Az **R** és **S** reláció közötti kapcsolatot a következőképpen értelmezzük:

* Az **R ÷ S** műveletet akkor hajtjuk végre, ha az **R** és **S** relációk között az **X** attribútumok (S sémája) a **Z** attribútumoknak (R sémája) részhalmazát képezik.
* Az eredményrelációban szereplő rekordok azok, amelyek az **R** relációban szereplő rekordok közül az összes **S** relációban szereplő értékkel összefüggésben vannak.

**Séma:**

Jelöljük **T**-vel az eredményrelációt. Az eredményreláció sémája **Y** lesz, ahol:

Y=Z−XY = Z - XY=Z−X

**Z** az **R** sémáját alkotó attribútumok halmaza, és **X** az **S** sémáját alkotó attribútumok halmaza. Így az **Y** halmaz azokat az attribútumokat tartalmazza, amelyek az **R** és **S** közötti osztási művelet eredményeként megmaradnak.

**Hányados művelet (I):**

A hányados művelet alapvetően az osztás művelet kiterjesztése. A hányados művelet az alábbi műveletek sorozataként is értelmezhető, és a részletes műveletek hasonlóak az osztás végrehajtásához.

## JOIN Kiterjesztése (4.13)

**Inner join**  
Azok a rekordok, melyekhez nincs kapcsolódó rekord a másik relációban, nem kerülnek bele az összekapcsolás utáni relációba.

**Outer join**  
Nem számít, hogy van-e kapcsolódó rekord.

* **Left outer join**
* **Right outer join**
* **Full outer join**

## ****Funkcionális függés (functional dependency, FD) (5.1)****

A **funkcionális függés** egy olyan szabály, amely azt mondja meg, hogy ha az egyik attribútum (vagy attribútumok halmaza) értéke adott, akkor egy másik attribútum (vagy attribútumok halmaza) értéke is meghatározott.

Például egy adatbázisban legyen egy tábla, amiben szerepelnek **név**, **születési dátum** és **lakcím** mezők. A **név** alapján egyértelműen meghatározható a **születési dátum** és a **lakcím**, tehát ha két rekordban ugyanaz a név szerepel, akkor a két rekordnak ugyanaz lesz a születési dátuma és a lakcíme is. Ezt funkcionális függésnek nevezzük: a **név → születési dátum, lakcím**.

Tehát egy funkcionális függés azt jelenti, hogy ha egy adott attribútum (vagy azok halmaza) értéke megegyezik két rekordban, akkor egy másik attribútum (vagy azok halmaza) értékének is meg kell egyeznie ugyanebben a két rekordban. A **funkcionális függés** egyszerűbben kifejezve azt jelenti, hogy egy attribútum (vagy attribútumok csoportja) értéke alapján egy másik attribútum (vagy attribútumok) értéke meghatározható.

Más szavakkal: egy **R** relációsémában **X** akkor és csak akkor határozza meg funkcionálisan **Y**-t, ha valahányszor két rekord megegyezik **X** értékeiben, akkor szükségszerűen megegyezik **Y** értékeiben is.

**Megjegyzés:**

* Ha egy relációban **X** egy szuperkulcs (azaz **X** értéke alapján mindig egyedien beazonosítható egy rekord), akkor **X** funkcionálisan meghatározza **Y**-t. Ez azért van, mert ha **X** egy szuperkulcs, akkor nem létezhet két olyan rekord, amelynek ugyanaz az **X** értéke, de különböző **Y** értékei.
* Ha egy relációban **X → Y** teljesül, akkor nem tudunk arról biztosat mondani, hogy **Y → X** is igaz-e. Ha mindkét irányú függés teljesül (azaz **X → Y** és **Y → X** is igaz), akkor **X** és **Y** kölcsönösen meghatározzák egymást, ezt nevezzük kölcsönös funkcionális függésnek.
* Ha egyik irányú függés sem teljesül, azaz **X → Y** és **Y → X** sem igaz, akkor azt mondjuk, hogy **X** és **Y** funkcionálisan függetlenek.

**Funkcionális függés jelölés**

A funkcionális függések felírásakor a halmazokat jelölő zárójeleket és a halmaz elemeit elválasztó vesszőket megállapodás szerint elhagyhatjuk, ha az attribútumokat egybetűs nevekkel azonosítjuk. Ezt az egyszerűsített jelölést gyakran használjuk, különösen akkor, ha az attribútumok nevei egy betűből állnak.

**Példák:**

* **{A, B} → {C}** helyett **AB → C**  
  (A **{A, B}** halmazt egyszerűsítve **AB**-ként írjuk.)
* **{A, B, C} → {D, E}** helyett **ABC → DE**  
  (A **{A, B, C}** halmazt **ABC**-ként, és a **{D, E}** halmazt **DE**-ként írjuk.)

**Halmazok uniójának jelölése:**

* Ha **X** és **Y** attribútumhalmazok, akkor a funkcionális függés mindkét oldalán használható az **XY** egyszerűsítés, amely a két halmaz unióját jelöli.
  + **X ∪ Y → Z** helyett **XY → Z** (Az **X ∪ Y** halmaz uniója helyett **XY**-ként jelöljük.)

Ezáltal a jelölés egyszerűsödik, könnyebben olvashatóvá válik, és elkerülhetjük a zárójelek és vesszők használatát, ha azok nem szükségesek a megértéshez.

## A funkcionális függések tulajdonságai (5.2)

A képen szöveg, Betűtípus, képernyőkép, sor látható

Automatikusan generált leírás

**Triviális és nemtriviális funkcionális függés**

Egy funkcionális függés **X → Y** akkor **triviális**, ha az **X** attribútumhalmaz tartalmazza az **Y** attribútumhalmaz összes elemét, azaz **X** halmaz a **Y** halmaz szuperhalmaza. Más szóval, ha **X** tartalmazza **Y**-t, akkor a függés triviális.

Például:

* Ha **X = {A, B}** és **Y = {A}**, akkor a **X → Y** függés triviális, mert **X** tartalmazza **Y**-t.

**Nemtriviális funkcionális függés**

Akkor beszélünk **nemtriviális** funkcionális függésről, ha **X** nem tartalmazza **Y**-t. Tehát, ha **X** és **Y** nem kapcsolódnak szorosabban egymáshoz (nem tartalmazza egyik halmaz a másikat).

**Reflexivitás**:

* Ez a szabály azt mondja, hogy egy attribútumhalmaz mindig meghatározza önmagát, vagyis bármely **X** attribútumhalmazra igaz, hogy **X → X**. Azaz, ha valamilyen attribútumhalmaz szerepel az egyik oldalon, akkor a másik oldalon is ugyanaz az attribútumhalmaz szerepelhet.

**Példa**:

* Ha **X = {A, B}**, akkor **X → X** igaz, vagyis a **{A, B}** halmaz meghatározza önmagát.

**Augmentivitás**:

* Az augmentivitás szabálya szerint, ha egy funkcionális függés **X → Y** igaz, akkor ha mindkét oldalát ugyanazzal az attribútumhalmazzal bővítjük, az új függés is igaz lesz. Azaz, ha **X → Y**, akkor **XZ → YZ** is igaz, ahol **Z** bármely attribútumhalmaz.
* **Példa**:
  + Ha **{A} → {B}**, akkor **{A, C} → {B, C}** is igaz.

**Tranzitivitás**:

* A tranzitivitás szabálya szerint, ha **X → Y** és **Y → Z**, akkor **X → Z** is igaz. Ez azt jelenti, hogy ha **X** meghatározza **Y**-t, és **Y** meghatározza **Z**-t, akkor **X** is meghatározza **Z**-t.
* **Példa**:
  + Ha **{A} → {B}** és **{B} → {C}**, akkor **{A} → {C}** is igaz.

**Dekompozíció**:

* A dekompozíció szabálya azt mondja, hogy ha **X → Y** igaz, akkor **X → Y1** és **X → Y2** is igaz, ahol **Y1** és **Y2** a **Y** részhalmazai. Tehát a jobb oldalon szereplő attribútumokat leválaszthatjuk, és az új függések is érvényesek maradnak.
* **Példa**:
  + Ha **{A} → {B, C}**, akkor **{A} → {B}** és **{A} → {C}** is igaz.

**Additivitás**:

* Az additivitás szabálya szerint, ha több funkcionális függés van ugyanazzal az attribútumhalmazzal a bal oldalon, akkor azokat egyetlen függéssé összevonhatjuk. Ha **X → A1**, **X → A2**, ..., **X → An**, akkor egyetlen **X → {A1, A2, ..., An}** függésre összevonhatók.
* **Példa**:
  + Ha **{A} → {B}**, **{A} → {C}**, akkor **{A} → {B, C}** is igaz.

## Az Armstrong-axiómák (5.3)

William Ward Armstrong 1974-ben bizonyította be, hogy három alapvető szabály – a **reflexivitás**, az **augmentivitás** és a **tranzitivitás** – együtt helyes és teljes rendszert alkot a funkcionális függések levezetésére. Ezt a rendszert **Armstrong-axiómáknak** nevezzük.

**1. Helyesség:**

* A helyesség azt jelenti, hogy ha egy adott **R** relációsémán van egy funkcionális függésekből álló **F** halmaz, akkor bármelyik olyan függés, amely levezethető a három szabály segítségével, biztosan fennáll a reláció minden olyan **r** rekordjában, amely teljesíti az **F** halmazban lévő függéseket.
* Tehát a három szabály (reflexivitás, augmentivitás, tranzitivitás) helyesek, mivel a segítségükkel levezetett újabb függések mindig megfelelnek az adatbázis tényleges működésének.

**2. Teljesség:**

* A teljesség azt jelenti, hogy a reflexivitás, augmentivitás és tranzitivitás szabályainak ismételt alkalmazásával előállítható az **F**-ből levezethető összes funkcionális függés teljes halmaza.
* Más szavakkal: ha **F**-ből indulunk ki, és csak a három szabályt alkalmazzuk, akkor az összes olyan funkcionális függést levezethetünk, amelyet **F**-ből kiindulva le lehet vezetni. Ezt az eredményt **F+**-nak hívjuk, és **F+** az **F** lezártja.

**Az Armstrong-axiómák**

A három alapvető axióma, amelyet Armstrong axiómákként ismertünk el:

1. **Reflexivitás**:
   * Ha **X** egy attribútumhalmaz, akkor **X → X** mindig igaz. Ez az axióma azt mondja, hogy egy attribútumhalmaz mindig meghatározza önmagát.
2. **Augmentivitás**:
   * Ha **X → Y**, akkor **XZ → YZ** is igaz, ahol **Z** bármely attribútumhalmaz. Ez azt jelenti, hogy egy funkcionális függés mindkét oldalát ugyanazzal az attribútummal bővítve az új függés is igaz lesz.
3. **Tranzitivitás**:
   * Ha **X → Y** és **Y → Z**, akkor **X → Z** is igaz. Ez az axióma azt mondja, hogy a funkcionális függések tranzitívak, vagyis ha egy attribútum meghatározza egy másik attribútumot, és az utóbbit meghatározza egy harmadik, akkor az első attribútum meghatározza a harmadikat is.

Ezek az axiómák együtt alkotják a funkcionális függések levezetésére és ellenőrzésére szolgáló rendszert, amely biztosítja, hogy az adatbázis-sémák szabályosak és konzisztens módon meghatározzák a függőségeket.

## A reláció attribútumainak szemantikája (6.1)

A reláció attribútumainak szemantikája az adatbázisok tervezése során alkalmazott irányelvek, amelyek célja a jól érthető, logikus, és megfelelően strukturált adatbázisok létrehozása. Ez a gyakorlat arra irányul, hogy az attribútumok és relációk jelentése könnyen érthető legyen, és az adatbázis sémája ne tartalmazzon semmilyen félrevezető vagy felesleges információt.

**1. Nem hivatalos irányelv:**

* Egy reláció **minden rekordja egy egyedet vagy egy kapcsolat-előfordulást reprezentáljon**.  
  Ez az irányelv biztosítja, hogy az adatbázis könnyen értelmezhető és logikusan tagolt legyen.

**Fontos következmények:**

1. **Egyedi egyedek és kapcsolatok elkülönítése**:
   * Például a **DOLGOZÓ**, az **OSZTÁLY** és a **PROJEKT** attribútumai ne legyenek összekeverve egy relációban. Minden entitástípusnak külön relációja legyen.
2. **Külső kulcsok használata**:
   * Az egyik relációban lévő egyedre való hivatkozás a másik relációban **külső kulcsokkal (foreign key)** történjen. Ez biztosítja az adatok összekapcsolhatóságát anélkül, hogy redundanciát okoznánk.
3. **Attribútumok szétválasztása**:
   * Az egyedekhez tartozó attribútumokat elkülönítve kell kezelni a kapcsolatok attribútumaitól. Például az alkalmazott neve és az osztály, ahol dolgozik, külön relációkban jelenjenek meg (egyik az alkalmazottakhoz, másik az osztályokhoz tartozik).

**2. Tömören: egyszerűség és érthetőség:**

* Olyan sémát kell tervezni, amely **logikusan felépített, könnyen magyarázható és értelmezhető relációról relációra**.
* Az attribútumok szemantikáját – vagyis a jelentésüket – egyértelművé kell tenni, hogy a relációk és attribútumok közötti kapcsolatok pontosan érthetők legyenek.

**Cél**: Az adatbázisok logikája tükrözze az adatok valódi szerkezetét és a közöttük lévő kapcsolatokat.

## Redundáns információk a rekordokban és a karbantartási anomáliák (6.2)

**1. Redundancia és problémák:**

Amikor az információkat redundánsan (többszörösen) tároljuk egy adatbázisban, az alábbi problémák léphetnek fel:

* **Tárhely pazarlás**: Az ismétlődő adatok felesleges helyet foglalnak, ami nagy adatbázisok esetén jelentős költségnövekedést eredményezhet.
* **Karbantartási anomáliák**: A redundancia miatt a következő problémák léphetnek fel az adatok frissítése vagy kezelése során:
  + **Beszúrási anomáliák**: Az új rekordok hozzáadása során előfordulhat, hogy egyes adatok redundanciája miatt nem tudjuk következetesen kezelni az adatokat. Példa: Ha egy alkalmazotthoz új projektet szeretnénk felvenni, de a projektadatok már máshol redundánsan szerepelnek, nehéz lehet biztosítani az adatbázis konzisztenciáját.
  + **Törlési anomáliák**: Egy rekord törlése nem szándékosan más kapcsolódó információk elvesztéséhez is vezethet. Példa: Egy projekt törlése egy alkalmazott nyilvántartásából az alkalmazott minden adatának elvesztését eredményezheti.
  + **Módosítási anomáliák**: Ha redundáns adatokat kell frissíteni, minden előfordulásukat módosítani kell. Ha ezt nem tesszük meg következetesen, inkonzisztens adatok keletkezhetnek.

**2. Nem hivatalos irányelv:**

Az adatbázis tervezése során **kerüljük a beszúrási, törlési és módosítási anomáliák** előfordulását.

* **Megoldás**: Az adatbázis sémáját úgy alakítsuk ki, hogy minimalizáljuk a redundanciát. Ez tipikusan **normalizálási technikákkal** érhető el, ahol az adatokat különböző relációkra osztjuk, és a redundáns információkat külső kulcsok segítségével kötjük össze.

**3. Kivételes esetek:**

Ha a redundancia és az anomáliák valamilyen okból elkerülhetetlenek (például teljesítményoptimalizálási szempontok miatt), akkor ezeket **dokumentálni kell**, hogy az alkalmazások és az adatbázis-kezelők ezeket figyelembe vehessék, és elkerüljék a hibákat.

**Példák a karbantartási anomáliákra a DOLG\_PROJ relációban**

**1. Módosítási anomália:**

* Reláció: DOLG\_PROJ(Dszsz, Pszám, Dnév, Pnév, Órák)
* **Probléma**: Ha a **P1** azonosítójú projekt nevét **"Számlázás"**-ról **"Ügyfél-számlakezelés"**-re változtatjuk, akkor a módosítást **minden egyes olyan rekordnál végre kell hajtanunk**, ahol a projekt azonosítója **P1**. Például, ha 100 dolgozó is részt vesz ezen a projekten, akkor 100 különböző rekordot kell módosítani. Ha valamelyik kimarad, inkonzisztens adatállapot jön létre (ugyanaz a projekt többféle néven szerepel).

**2. Beszúrási anomália:**

* Reláció: DOLG\_PROJ(Dszsz, Pszám, Dnév, Pnév, Órák)
* **Példa 1**: Nem tudunk új projektet hozzáadni, ha nincs legalább egy dolgozó, aki ehhez a projekthez tartozik. Például, ha szeretnénk létrehozni egy új projektet (pl. **P2: Adatbázis-tervezés**), de még nincs dolgozó hozzárendelve, akkor nem tudunk új rekordot beszúrni.
* **Példa 2** (fordított probléma): Nem tudunk új dolgozót hozzáadni, ha nincs egyetlen projekthez sem hozzárendelve. Például, ha egy új dolgozót (pl. **D123**) szeretnénk regisztrálni, de nem tudjuk megmondani, melyik projekten fog dolgozni, akkor nem tudjuk felvinni a relációba.

**3. Törlési anomália:**

* Reláció: DOLG\_PROJ(Dszsz, Pszám, Dnév, Pnév, Órák)
* **Példa 1**: Ha törlünk egy projektet (pl. **P1: Számlázás**), akkor az összes dolgozót is elveszítjük, akik ezen a projekten dolgoztak, mert ezek a dolgozók a projekt nélkül nem szerepelhetnek a relációban.
* **Példa 2** (fordított probléma): Ha egy dolgozó (pl. **D101**) az egyetlen, aki egy adott projekten dolgozik (pl. **P1**), és őt töröljük, akkor a projekt információja is elveszik, mert a projekt önállóan nem tárolható.

## NULL értékek a rekordokban (6.3)

**Mi az a NULL érték?**

A **NULL** érték olyan adatbázisokban használatos speciális érték, amely az alábbi esetek valamelyikét jelzi:

1. Az attribútum nem értelmezhető vagy érvénytelen az adott rekordnál.
2. Az attribútum értéke **ismeretlen** (de elvileg létezhet).
3. Az érték biztosan létezik, de **nem elérhető** az adott pillanatban.

**Irányelv:**

A relációkat úgy kell megtervezni, hogy a rekordokban található **NULL értékek számát minimalizáljuk**. Ez azért fontos, mert a NULL értékek:

* **Inkonzisztenciát okozhatnak** az adatbázisban (például az összegző műveletek vagy a szűrések során).
* **Kétségeket vethetnek fel** az adatok megbízhatóságával kapcsolatban.
* **Megnehezíthetik az alkalmazások működését** (például olyan lekérdezésekben, ahol a NULL értékeket külön kell kezelni).

**Megoldás a NULL értékek kezelésére:**

1. **Attribútumok külön relációkba helyezése:**  
   Azokat az attribútumokat, amelyek **gyakran vesznek fel NULL értéket**, érdemes **külön relációkba szervezni**, ahol az elsődleges kulcs kapcsolódik az eredeti táblához.
2. **NULL értékek elkerülése alternatív alapértékekkel:**  
   Ha lehetséges, definiáljunk **alapértelmezett értékeket** olyan attribútumokhoz, amelyek NULL értékeket tartalmazhatnak (például üres sztring, nulla, "ismeretlen").

## Álrekordok és a veszteségmentes összekapcsolás (6.4)

**Mi az álrekord?**

Egy **álrekord** (spurious tuple) olyan rekord, amely egy **összekapcsolási művelet (join)** eredményeként jön létre, de nem létezett az eredeti adatbázisban, és nem hordoz értelmes információt. Ez általában a rossz adatbázis-tervezés vagy nem megfelelő összekapcsolási feltételek miatt fordul elő.

**Mi a veszteségmentes összekapcsolás?**

A veszteségmentes összekapcsolás tulajdonsága garantálja, hogy ha egy relációt **dekomponálunk (szétbontunk)** két vagy több részrelációra, akkor azokat újra összekapcsolva (JOIN művelettel) visszaállíthatjuk az eredeti relációt **minden további rekord vagy adatvesztés nélkül**.

* **Cél:** Olyan dekompozíciók létrehozása, amelyek megfelelnek a veszteségmentes összekapcsolás követelményeinek.

**Irányelv:**

* A relációkat úgy kell megtervezni, hogy kielégítsék a **veszteségmentes összekapcsolás** feltételét.
* Egy **természetes összekapcsolás** (natural join) eredményeként **nem szabad álrekordokat előállítani**.

**Dekompozíció két kulcsfontosságú tulajdonsága:**

1. **Veszteségmentes összekapcsolás (nemadditivitás):**  
   Az összekapcsolás visszaállítja az eredeti relációt **minden további vagy felesleges rekord nélkül**.
   * Ez az alapfeltétel, amit minden esetben teljesíteni kell.
2. **Funkcionális függések megőrzése:**  
   A dekompozíció során az eredeti relációban fennálló funkcionális függések megmaradnak a részrelációkban is.
   * Ez a tulajdonság kevésbé szigorú, és ha szükséges, **feláldozható** a veszteségmentesség érdekében.

**Összekapcsolási művelet:**  
A két részreláció természetes összekapcsolása álrekordokat eredményezhet, ha a közös attribútum (**Pszám**) alapján nincs egyértelmű kapcsolat.

## Relációsémák normalizációja (6.5)

**Definíció:** A normalizáció az a folyamat, amelynek során szétbontjuk a normálformák definíciójának nem eleget tevő relációsémákat úgy, hogy az attribútumaikat dekompozícióval több kisebb relációsémába helyezzük át.

**Definíció:** A normálforma a relációsémák kulcsai és a bennük fennálló funkcionális függések segítségével megfogalmazott feltétel, amellyel megállapítható, hogy a relációséma egy adott normálformában van-e.

**A normálformák használata a gyakorlatban**

* A normalizációt úgy hajtják végre a gyakorlatban, hogy a kapott tervek magas színvonalúak legyenek, és rendelkezzenek az elvárt tulajdonságokkal.
* A normálformák gyakorlati alkalmazhatósága megkérdőjelezhetővé válik, ha az alapjukat képező megszorítások nehezen értelmezhetők vagy nehezen ismerhetők fel.
* Az adatbázis-tervezőknek nem szükséges a lehető legmagasabb normálformáig normalizálniuk (általában csak 3NF-ig, BCNF-ig vagy 4NF-ig).
* A denormalizáció az a folyamat, amelynek során magasabb normálformájú relációk összekapcsolását letároljuk alap relációként – alacsonyabb normálformában.

## Normálformák (6.6)

**Első normálforma (1NF)**

A **Első normálforma (1NF)** biztosítja, hogy egy relációban az attribútumok értékei atomikusak, azaz nem tartalmazhatnak összetett, többértékű vagy beágyazott adatokat. A 1NF-teljesítés követelményei a következőek:

**Tiltott elemek:**

* **Összetett attribútumok:** Az attribútumok nem tartalmazhatnak több értéket (például egy cím attribútum tartalmazza az utcát, várost és irányítószámot egybe).
* **Többértékű attribútumok:** Az attribútumok nem tárolhatnak több értéket egyszerre (például egy dolgozó több telefonos számot).
* **Beágyazott relációk:** Az attribútumok nem tartalmazhatnak olyan adatokat, amelyek más rekordokban nem atomiak. Azaz nem tartalmazhatnak összetett vagy többszintű adatokat.

**A reláció definíciójának részét képezi**: Minden attribútum értéke atomikus, azaz egy értéket jelent, nem pedig egy adathalmazt.

**Második normálforma (2NF)**

A **második normálforma (2NF)** célja, hogy eltávolítsa azokat a funkcionális függéseket, amelyek részlegesek, azaz az attribútumok nem függenek teljes mértékben az elsődleges kulcstól. A második normálformában a relációséma akkor van, ha:

1. **Teljes funkcionális függés**: Egy X → Y funkcionális függés teljes funkcionális függés, ha X-ből bármely A attribútum eltávolítva a függés már nem áll fenn. Ez azt jelenti, hogy ha egy attribútumot eltávolítunk X-ből, akkor a funkcionális függés megszűnik.
   * **Példa teljes függésre**: Ha egy dolgozó azonosítója (Dszsz) meghatározza a dolgozó nevét (Dnév), akkor Dszsz → Dnév egy teljes funkcionális függés, mert a dolgozó azonosítójának eltávolítása miatt nem tudjuk meghatározni a nevét.
2. **Részleges függés**: Egy X → Y funkcionális függés részleges függés, ha X-ből egy attribútum eltávolítható úgy, hogy a függés továbbra is fennáll. Ez a függés nem tekinthető teljesnek, ha egy attribútum eltávolítása után is meghatározható a függés.
   * **Példa részleges függésre**: Ha egy dolgozó ID-ja (Dszsz) és a projekt száma (Pszám) együttesen meghatározzák a projekt nevét (Pnév), de a projekt száma önállóan is meghatározza a projekt nevét, akkor a projekt neve részben függ a dolgozó ID-jától.
3. **Második normálforma (2NF)**: A reláció akkor van 2NF-ben, ha az elsődleges kulcs minden másodlagos (leíró) attribútumától teljesen függ. Más szavakkal, egy reláció akkor van 2NF-ben, ha minden nem-kulcs attribútum teljes mértékben függ a kulcstól, és nem csak annak egy részétől (részleges függés).

A második normálforma (2NF) felismerését megkönnyítheti két egyszerű kritérium, különösen abban az esetben, ha a relációnak csak egy kulcsjelöltje van. Ezek a következőek:

1. **Ha a kulcs egyetlen attribútumból áll, akkor a reláció 2NF típusú.**
   * **Indoklás:** Ha a kulcs egyetlen attribútumból áll, akkor nem létezik részleges függés, mivel nincs több attribútum a kulcsban, amitől függhetne a többi attribútum. Ezért a reláció biztosan 2NF típusú.
2. **Ha a relációban nincsenek másodlagos (leíró) attribútumok, akkor a reláció 2NF típusú.**
   * **Indoklás:** Ha minden attribútum része a kulcsnak, akkor nincs olyan nem-kulcs attribútum, amely részben függene a kulcs egyik részétől. Mivel minden attribútum kulcselem, így nincs részleges függés, ezért a reláció már eleve 2NF-ben van.

**Harmadik normálforma (3NF)**

A harmadik normálforma (3NF) alapvetően a tranzitív függés fogalmán alapul. A tranzitív függés azt jelenti, hogy ha egy attribútum X-ből függ egy másik attribútumtól Y, és Y-ből függ egy harmadik attribútumtól Z, akkor X-ből is közvetve függ Z.

**Definíciók**

1. **Tranzitív függés:**
   * Egy **X → Y** funkcionális függés tranzitív függés, ha létezik egy **Z** attribútumhalmaz, amely nem kulcsjelölt és nem része R egyetlen kulcsának sem, és fennállnak az alábbi függések:
     + **X → Z**
     + **Z → Y**
2. **Harmadik normálforma (3NF):**
   * Egy **R** relációséma harmadik normálformában (3NF-ben) van, ha:
     + A reláció **2NF-ben van** (vagyis nem tartalmaz részleges függéseket),
     + És nincs olyan másodlagos (leíró) attribútum, amely tranzitívan függene az elsődleges kulcstól.

**Összefoglalva:**

* **2NF**: A reláció **2NF**-ben van, ha nincs részleges függés. Vagyis, ha egy attribútum részben függ egy összetett kulcs egy részétől, azt el kell távolítani.
* **3NF**: A reláció **3NF**-ben van, ha nincsenek tranzitív függések. Ha egy nem kulcs attribútum függ egy másik nem kulcs attribútumtól, azt el kell távolítani, hogy az attribútum közvetlenül csak a kulcstól függjön.

A **Boyce-Codd-féle normálforma** (BCNF) egy olyan normálforma, amely szigorúbb, mint a harmadik normálforma (3NF), és azt mondja ki, hogy minden funkcionális függés bal oldali része (a bal oldali attribútumhalmaz) **szuperkulcs** kell, hogy legyen. Nézzük meg lépésről lépésre, mit jelent mindez.

**Mi az a szuperkulcs?**

* **Szuperkulcs** az a halmaz, amely elég ahhoz, hogy **egyedileg meghatározzon minden rekordot** a relációban. Ez azt jelenti, hogy a szuperkulcs minden rekordra egyedülálló értéket ad. Például, ha egy alkalmazott EmployeeID-ja egyedi, akkor EmployeeID szuperkulcs lehet, mert egyedülállóan meghatározza az alkalmazottat.

**Mi a Boyce-Codd-féle normálforma (BCNF)?**

Egy reláció **BCNF-ben van**, ha minden olyan nemtriviális funkcionális függés, amely a relációban fennáll, **olyan attribútumra** mutat, amely **szuperkulcs**.

**Mi az a nemtriviális funkcionális függés?**

A nemtriviális függés azt jelenti, hogy egy attribútum (vagy attribútumhalmaz) meghatároz egy másik attribútumot, de nem triviálisan, azaz nem egy olyan nyilvánvaló kapcsolat, mint például egy attribútum meghatározása saját magától.

**Hogyan működik ez a gyakorlatban?**

Ha egy reláció **3NF**-ben van, az azt jelenti, hogy **vagy a funkcionális függés bal oldali része szuperkulcs**, vagy **az attól függő attribútum elsődleges attribútum**. A BCNF azonban azt mondja, hogy **minden funkcionális függés bal oldali része szuperkulcs kell, hogy legyen** – függetlenül attól, hogy az attribútum elsődleges vagy sem.

**Hogyan ismerhetjük fel, hogy egy reláció BCNF-ben van-e?**

* Ha egy relációban van egy olyan **funkcionális függés**, hogy **X → A** (tehát X meghatározza A-t), és X **nem szuperkulcs**, akkor a reláció **nem BCNF-ben** van.
* Ha minden ilyen függés bal oldala szuperkulcs, akkor a reláció **BCNF-ben van**.

## Adatbázisrendszer (Database System) felépítése (7.1)

* **Adatok**
  + **Adatbázis**
  + **Adatszótár (metaadatok)**
* **Szoftver(ek)**
* **Felhasználók**

**Adatok (DB)**

* **Tárolt adatbázis**: adatok halmaza
* **Adatszótár (Data Dictionary)**: A tárolt adatokra vonatkozó információk együttese
  + az adatok szerkezetét leíró információk
  + a felhasználókra vonatkozó adatok
  + az alkalmazásokról szóló információk.

**Adatbázis-kezelő rendszerek (DBMS)**

Olyan programrendszer, melynek feladata az adatbázishoz történő hozzáférések biztosítása és az adatbázis belső karbantartási feladatainak ellátása, azaz:

* Adatbázisok létrehozása
* Adatbázisok tartalmának definiálása
* Adatok tárolása
* Adatok lekérdezése
* Adatok védelme
* Adatok titkosítása
* Hozzáférési jogok kezelése
* Fizikai adatszerkezet szervezése

## Adatdefiníciós nyelv (DDL) (7.2)

* **Leírás**: Az adatbázisokban tárolt adatok illetve a közöttük fennálló logikai kapcsolatok leírására szolgáló nyelv.
* **Fizikai adatfüggetlenség**: Az adataival dolgozó felhasználói programok és a felhasználók ad hoc információkérései függetlenek az adatok tárolási és elérési módjától.
* **Logikai adatfüggetlenség**: Az adatok olyan szervezése, rögzítése, mely biztosítja, hogy azok függetlenek legyenek a felhasználói programoktól. Ehhez külön kell választani az adatok fizikai leírását és rögzítését a programok által látott logikai adatszerkezetektől.
* **Példák**: CREATE, ALTER, DROP, RENAME

## Adatmanipulációs nyelv (DML) (7.3)

* **Leírás**: Az adatleíró nyelvvel definiált adatbázisok karbantartására szolgáló műveletek leírására használt nyelv.
* **Feladatok**:
  + Már létező táblák adatokkal való feltöltése.
  + Már létező adatok módosítása, törlése.
  + Az adatok lekérdezésének lehetővé tétele.
  + Az adatbázisban tárolt adatelemekhez való szelektív hozzáférés támogatása.
* **Példák**: INSERT, UPDATE, DELETE, SELECT

## Adatvezérlő nyelv (DCL) (7.4)

* **Alapvető funkciói**:
  + **Tranzakció kezelés**: A feladat szempontjából összetartozó lépések sorozata, például gázszámla kiegyenlítése.
  + **Adatvédelmi feladatok, konkurens hozzáférés szabályozása**:
    - **Konkurens hozzáférés**: Az adott pillanatban az egyik felhasználó olvashatja, míg egy másik írhatja az adatbázist. Míg az írási művelet be nem fejeződik, addig az olvasó az eredeti adatokat látja. Ha egyszerre többen szeretnének írni, akkor sorba kell állniuk.
  + **Hozzáférési jogok kezelése**.
  + **Konzisztencia biztosítása**: Logikai ellentmondásmentesség, például több táblát érintő tranzakciók esetén, a többszörösen előforduló adatok változtatásának végrehajtása minden előfordulási helyükön, valamint integritás (teljesség megőrzése).
  + **Egyéb, a rendszer működését befolyásoló eszközök, például áramkimaradások, lemezhibák kezelése**.
* **Példák**: COMMIT, ROLLBACK, GRANT, REVOKE.

## Felhasználók csoportjai (7.5)

**Előtérben tevékenykedők**

* **Adatbázis adminisztrátorok (DBA)**: Felelősek az adatbázisok napi karbantartásáért és a rendszerek megfelelő működéséért.
* **Adatbázis tervezők**: Ők tervezik az adatbázisok struktúráját, sémáit és biztosítják a normálformák alkalmazását.
* **Végfelhasználók**: Azok a felhasználók, akik az adatbázis által biztosított adatokat használják az üzleti vagy személyes céljaikra.
* **Rendszerelemzők, alkalmazás programozók**: A rendszer specifikációját készítik el, valamint alkalmazásokat fejlesztenek, amelyek interakcióba lépnek az adatbázissal.

**Háttérben dolgozók**

* **DBMS tervezők és fejlesztők**: Az adatbázis-kezelő rendszer fejlesztéséért és karbantartásáért felelősek.
* **Kapcsolódó csomagok (pl. modellező, tervező szoftver) fejlesztői**: Az adatbázishoz kapcsolódó segédprogramok fejlesztése.
* **Üzemeltetők és karbantartók**: Felelősek az adatbázis-kezelő rendszerek működésének folyamatos fenntartásáért és hibák elhárításáért.

## Szerepkörök (7.6)

**Adatbázis adminisztrátor (DBA)**

Az **adatbázis adminisztrátor (DBA)** felelős az adatbázissal kapcsolatos központi tevékenységekért. Fő feladatai közé tartozik:

* Az adatbázis megszervezése, beleértve az adatmodellek leírását, tárolási módok és keresési stratégiák meghatározását.
* Az adatbázis újraszervezése, ha hatásfok csökkenést tapasztal.
* Az adatbázisban tárolt információk védelme.
* Az adatok helyességének ellenőrzése.
* Az új felhasználói igények kielégítése.
* Az adatbázis hangolása a teljesítmény növelése érdekében.

**Adatbázis tervezők**

Az **adatbázis tervezők** felelősek a tartalom, a szerkezet és a megszorítások definiálásáért, valamint az adatbázis felé intézett funkciók és tranzakciók kialakításáért. Együttműködnek a végfelhasználókkal, hogy megértsék igényeiket, és a legmegfelelőbb adatbázis-tervet készítsék el.

**Végfelhasználók**

A **végfelhasználók** az adatokat lekérdezésekre, riportok készítésére használják, illetve egyes esetekben frissítik az adatbázis tartalmát. Ők azok, akik ténylegesen használják az adatbázisokat, és miattuk létezik az adatbázis.

**Végfelhasználók csoportjai**

* **Eseti felhasználók**: Alkalomszerűen érik el az adatbázist, különböző információk lekérdezésére. Ők gyakran vezetők, vagy azok, akik csak ritkán dolgoznak adatbázisokkal.
* **Naiv vagy parametrikus felhasználók**: Ők a legnagyobb csoport, akik folyamatosan lekérdezik és módosítják az adatbázist, jól megtervezett, tesztelt tranzakciók segítségével. Példák: banki ügyfelek, helyfoglalással foglalkozó ügynökök, közösségi média felhasználók.
* **Szofisztikált felhasználók**: Üzleti elemzők, tudósok, mérnökök, akik ismerik a rendszer képességeit, és saját alkalmazásokat készítenek az összetett követelményeikhez.
* **Önálló felhasználók**: Főként személyes adatbázist tartanak fenn kész csomagolt alkalmazásokkal, például egy adóbevallást készítő program használata.

**Rendszerelemzők, alkalmazás programozók**

A **rendszerelemzők** határozzák meg a végfelhasználók követelményeit és a szükséges tranzakciók specifikációit. Az **alkalmazás programozók** implementálják ezeket a specifikációkat, tesztelik, dokumentálják és karbantartják. A DBMS képességeit jól ismerik a munkájukhoz.

**DBMS tervezők és fejlesztők**

A **DBMS tervezők és fejlesztők** felelősek a DBMS (adatbázis-kezelő rendszer) moduljainak, interfészeinek elkészítéséért. Tipikus modulok közé tartozik:

* Lekérdezés-feldolgozás.
* Interfész-feldolgozás, együttműködés az operációs rendszerrel és programozási nyelvek fordítóival.
* Adatok elérése és konkurencia vezérlés.
* Adat helyreállítás és biztonság kezelés.

**Kapcsolódó csomagok fejlesztői**

A **kapcsolódó csomagok fejlesztői** szoftvercsomagokat készítenek, amelyek adatbázis modellezéssel, tervezéssel, teljesítmény növeléssel, szimulációval vagy teszt adatgenerálással foglalkoznak. Gyakran független cégek fejlesztik ezeket a csomagokat.

**Üzemeltetők és karbantartók**

Az **üzemeltetők és karbantartók** felelősek az adatbázisrendszer hardver- és szoftverkörnyezetének működtetéséért és karbantartásáért.

## Osztály és alosztály az EER modellekben (8.1)

**Osztály**

* Az **osztály** az egyedek egy halmaza vagy kollekciója.
* Tartalmaz minden olyan szerkezetet az EER sémában, amely egyedeket csoportosít, például:
  + Egyedtípusokat,
  + Alosztályokat,
  + Szuperosztályokat,
  + Kategóriákat.

**Alosztály és szuperosztály kapcsolata**

* Egy **alosztály** (S) az egyedek egy részhalmaza, amelyre további tulajdonságok vagy attribútumok jellemzőek a teljes halmazhoz képest.
* A teljes halmazra (azaz az összes egyedre) **szuperosztályként** (C) hivatkozunk.

**Jelölés**

* Az alosztály/szuperosztály kapcsolatot C / S formában jelöljük.
* Mindig teljesül, hogy: **S ⊆ C** (az alosztály minden eleme a szuperosztály része).

## Specializáció és Generalizáció (8.2)

**Specializáció**

A **specializáció** azt jelenti, hogy egy általános csoportot (szuperosztályt) kisebb, specifikusabb csoportokra (alosztályokra) bontunk.

* A szuperosztály tartalmazza az alapvető tulajdonságokat, amelyek minden elemre jellemzőek.
* Az alosztályok pedig azokra az egyedekre koncentrálnak, amelyek az alapvető tulajdonságokon kívül további, különleges tulajdonságokkal is rendelkeznek.

**Egyszerű példa:**

* **Szuperosztály**: **Jármű**
* **Alosztályok**:
  + **Autó** (pl. ülések száma, csomagtartó mérete),
  + **Teherautó** (pl. rakománykapacitás),
  + **Motor** (pl. hengerűrtartalom).

**Generalizáció**

A **generalizáció** az ellentétes folyamat: itt több specifikusabb csoportot (alosztályokat) összevonunk egy általános csoportba (szuperosztályba).

**Egyszerű példa:**

* **Alosztályok**:
  + **Mérnök** (pl. szakterület, projekt neve),
  + **Adminisztratív dolgozó** (pl. ügyintézési osztály).
* **Szuperosztály**: **Alkalmazott** (pl. név, belépési dátum, fizetés).

**Specializáció példányai – Fontos szabály**

* **Nem kötelező**, hogy a szuperosztály **minden egyede** benne legyen valamely alosztályban.
  + Ez azt jelenti, hogy lehetnek olyan egyedek, amelyek csak a szuperosztályhoz tartoznak, de egyik alosztályhoz sem.
* **Fordítva viszont kötelező**:
  + Minden alosztály egyede **mindig** része a szuperosztálynak is!
  + Ez logikus, hiszen az alosztály mindig egy részhalmaza a szuperosztálynak.

## Totális és részleges, illetve diszjunkt és átfedő specializáció (8.3)

**Totális specializáció (Total)**

* Egy specializáció **totális**, ha a szuperosztály minden egyede tartozik **valamelyik alosztályhoz**.
* Azaz a szuperosztály nem tartalmaz olyan egyedeket, amelyek ne lennének benne legalább egy alosztályban.

**Részleges specializáció (Partial)**

* Egy specializáció **részleges**, ha a szuperosztály tartalmazhat olyan egyedeket, amelyek nem tartoznak egyetlen alosztályhoz sem.

**Diszjunkt specializáció (Disjoint)**

* Egy specializáció **diszjunkt**, ha egy szuperosztály egyede egyszerre **csak egyetlen alosztályhoz** tartozhat.

**Átfedő specializáció (Overlapping)**

* Egy specializáció **átfedő**, ha egy szuperosztály egyede egyszerre **több alosztályhoz is tartozhat**.

## Predikátumdefiniált és felhasználó által definiált specializáció (8.4)

**Predikátumdefiniált specializáció (összetett szabályok)**

Ez egy olyan alosztály, ahol egy **feltétel (predikátum)** dönt arról, hogy mely egyedek tartoznak bele.  
Gondolj rá úgy, mint egy automatikus szűrőre: ha az egyed megfelel a feltételnek, akkor az alosztály része lesz.

**Hogyan működik?**

* Van egy szuperosztályunk, például **Diákok**.
* Írunk egy szabályt (feltételt), például: "Legyen az életkor kevesebb, mint 18".
* Az alosztályba, amit mondjuk **Kiskorú diákoknak** hívunk, bekerül minden diák, aki megfelel ennek a feltételnek.

**Felhasználó által definiált specializáció**

Ez egy olyan alosztály, ahol **nem egy feltétel dönti el**, hogy ki tartozik bele, hanem **te vagy más felhasználók manuálisan választják ki** az egyedeket.

**Hogyan működik?**

* Itt nincs automatikus szabály, te döntöd el, hogy mely egyedek kerüljenek az alosztályba.
* Például a szuperosztály a **Diákok**, és van egy alosztályod, amit **Kiemelt diákoknak** hívsz. Ide csak azokat a diákokat veszed fel, akik valami különlegeset értek el, például jó jegyet kaptak vagy díjat nyertek.

**Attribútumdefiniált specializáció (egyszerű szabályok)**

Ez egy olyan típusú specializáció, ahol egy konkrét **attribútum** értéke dönti el, hogy melyik alosztályba tartozik egy egyed. Ez tehát egyfajta szabály, amely az attribútum értékét veszi alapul.

**Hogyan működik?**

* Van egy szuperosztály, például **Alkalmazott**.
* Az alosztályokat úgy határozzuk meg, hogy az egyik attribútum (mondjuk a **munkakör**) egy konkrét értéket vesz fel.

## Kategória - unió típus (8.5)

Egy **kategória** (más néven unió típus) egy olyan **osztály**, amely több **szuperosztály** (D1, D2, …, Dn) elemeiből áll, de ezeknek az elemeknek csak egy részhalmazát tartalmazza.

**Egyszerűbben:**

* Képzeld el, hogy több csoportot (szuperosztályt) összegyűjtesz egy nagyobb halmazba, de nem veszel bele minden elemet, csak azokat, amelyek megfelelnek bizonyos feltételeknek. Ez a részhalmaz lesz a **kategória (T)**.

**Kapcsolattípus kiterjesztése**

Az EER modellben nemcsak **egyedtípusok** (pl. Hallgató) vehetnek részt kapcsolattípusokban, hanem **bármilyen osztály**, például:

* Egy szuperosztály,
* Egy alosztály,
* Vagy akár egy kategória (mint a VIP példában).

**EER Diagram grafikus jelölése**

Az EER diagramon a kategóriákat ugyanúgy **téglalapokkal** ábrázoljuk, mint az egyedtípusokat, így a jelölések **konzisztensek** az ER modellel.

## EER séma leképezése relációs sémára (8.6)

1. Erős egyedtípusok leképezése
2. Gyenge egyedtípusok leképezése
3. Bináris 1:1 számosságú kapcsolattípusok leképezése
   * (a) Külső kulcs használata
   * (b) Összevonás
   * (c) Kereszthivatkozás v. kapcsoló reláció használata
4. Bináris 1:N számosságú kapcsolattípusok leképezése
5. Bináris M:N számosságú kapcsolattípusok leképezése
6. Többértékű attribútumok leképezése
7. N-edfokú kapcsolattípusok leképezése
8. Specializációk és generalizációk leképezése
9. Unió típusok (kategóriák) leképezése

## Specializációk és generalizációk leképezése – több relációs lehetőségek (8.7)

**(a) Több reláció – szuperosztály és alosztályok:**

* Először létrehozunk egy relációt a szuperosztály számára, amely tartalmazza a szuperosztály összes attribútumát. A kulcs ebben a relációban egy egyedi azonosító lesz, amely az összes egyedet azonosítja.
* Ezután minden alosztály számára létrehozunk egy külön relációt. Minden ilyen reláció tartalmazza az alosztály saját attribútumait, valamint a szuperosztály kulcsát is, hogy összekapcsolhassuk az alosztályokat a szuperosztállyal.
* Ez a megoldás mindenféle specializáció esetén működik, legyen az totális (amikor minden egyed a szuperosztály valamelyik alosztályának része), részleges (amikor nem minden egyed tartozik alosztályhoz), diszjunkt (amikor egy egyed csak egy alosztályhoz tartozik) vagy átfedő (amikor egy egyed több alosztályhoz is tartozhat).

**(b) Több reláció – csak alosztály relációk:**

* Ebben az esetben minden alosztályhoz létrehozunk egy külön relációt, amely tartalmazza az alosztály attribútumait és a szuperosztály kulcsát. Az alosztályok mindegyike saját magától egyedi attribútumokkal rendelkezik, de a szuperosztály kulcsa biztosítja, hogy az alosztályokat összekapcsoljuk a szuperosztállyal.
* Ez a megoldás akkor alkalmazható, ha a specializáció totális, tehát minden egyed biztosan tartozik valamelyik alosztályhoz.
* Ha a specializáció átfedő, előfordulhat, hogy egy egyed több alosztályhoz is tartozik, így több relációban is szerepelhet.

## Specializációk és generalizációk leképezése – egyetlen relációs lehetőségek (8.8)

**(c) Egyetlen reláció egy típus attribútummal:**

* Létrehozunk egy **L** nevű relációt, amely tartalmazza az összes szükséges attribútumot:
  + Egy kulcs attribútumot (**k**), ami az egyedek azonosítására szolgál.
  + A szuperosztály és az alosztályok attribútumait is.
  + Egy **típus attribútumot** (**t**), amit diszkrimináló attribútumnak nevezünk. Ez mutatja meg, hogy az egyes rekordok melyik alosztályhoz tartoznak.
* Az **elsődleges kulcs (PK)** ebben az esetben a **k** lesz.
* Ez a megoldás **csak diszjunkt specializációk** esetén alkalmazható, amikor egy egyed csak egy alosztályhoz tartozik.
* Hátrányként felmerülhet, hogy **sok NULL érték** jelenhet meg, ha sok különböző attribútum található az alosztályokban.

**(d) Egyetlen reláció több típus attribútummal:**

* Ebben az esetben létrehozunk egy **L** nevű relációt, amely szintén tartalmazza az összes szükséges attribútumot:
  + A kulcs attribútumot (**k**).
  + A szuperosztály és az alosztályok attribútumait.
  + **Típus attribútumokat** (**t1, t2, ..., tm**), amelyek logikai típusúak, és megmutatják, hogy egy adott rekord melyik alosztályhoz tartozik.
* Az **elsődleges kulcs** itt is a **k** lesz.
* Ez a megoldás akkor is alkalmazható, ha **átfedő specializációk** vannak, amikor egy egyed több alosztályhoz is tartozhat.

## Unió típusok (kategóriák) leképezése (8.9)

Amikor különböző kulcsokkal rendelkező szuperosztályok által definiált kategóriát akarunk leképezni, célszerű egy új kulcsot bevezetni, amit **helyettesítő kulcsnak** nevezünk. Ez a helyettesítő kulcs segít abban, hogy a kategória számára egy új, közös kulcsot hozzunk létre.

* **Helyettesítő kulcs:** Olyan kulcsattribútum, amit a kategóriát reprezentáló relációban hozunk létre. Ezzel biztosítjuk, hogy a különböző szuperosztályokból származó adatokat össze tudjuk kapcsolni.
* A helyettesítő kulcsot nemcsak a kategória relációjába, hanem azokba a relációkba is bele kell venni, amiket a kategória szuperosztályaiból képzünk.

Ez segít abban, hogy a különböző szuperosztályok adatainak kezelése és a kapcsolódó relációk megfelelően működjenek.

## DBS csoportosítása konkurens felhasználók száma szerint (9.1)

1. **Egyfelhasználós rendszer:**  
   Egyidejűleg legfeljebb egy felhasználó használhatja a rendszert.
2. **Többfelhasználós rendszer:**  
   Egyidejűleg (konkurens módon) több felhasználó is elérheti a rendszert. A legtöbb rendszer ilyen típusú.

**Konkurencia típusa**

* **Összefésült egyszálas feldolgozás:**  
  A folyamatok (processzek) konkurens végrehajtása egy CPU-n fésülődik össze.
* **Párhuzamos feldolgozás:**  
  A folyamatok (processzek) egyidejűleg (konkurens módon) több CPU-n hajtódnak végre.

## Tranzakció (9.2)

A tranzakció egy adatbázis-folyamat, amely egy logikai egységet alkot, és egy vagy több adatbázis-hozzáférési műveletből épül fel. Az adatbázis-hozzáférési műveletek lehetnek:

* **Olvasás** – kinyerés,
* **Írás** – beszúrás, törlés, módosítás.

A tranzakciót alkotó adatbázis-műveletek:

* Beágyazhatóak egy alkalmazói programba, vagy
* Megadhatóak explicit módon egy magas szintű lekérdező nyelv segítségével (pl. SQL).

**Tranzakció határainak meghatározása**

A tranzakció határait megadhatjuk egy alkalmazói programban explicit módon a **begin transaction** és **end transaction** utasításokkal, ahol a két határ között elhelyezkedő összes adatbázis-hozzáférési művelet alkot egy tranzakciót.

Egy alkalmazói program egynél több tranzakciót is tartalmazhat, ha több tranzakció elhatároló szerepel benne.

**Típusok**

* **Read-only tranzakció:**  
  A tranzakciót alkotó adatbázis-műveletek **nem módosítják** az adatbázist, csak lekérdezik azt.
* **Read-write tranzakció:**  
  A tranzakciót alkotó adatbázis-műveletek **módosítják** is az adatbázist.

## Adatbázis leegyszerűsített modellje (9.3)

* **Adatbázis:** Nevesített adatelemek (named data item) összessége.
* Az adatok finomsága (granularity) az adatelemek (rekord, blokk (lemez), mezőérték) méretét jellemzi.  
  Az ismertetett fogalmak függetlenek a konkrét finomságtól.
* Az adatelem egyedi nevét a programozó tipikusan nem használja, csak egy módot biztosít az egyértelmű azonosításhoz.

**Tranzakció alapműveletei a leegyszerűsített adatbázis modellben**

* **Read\_item(X):**  
  Beolvassa az adatbázis X elnevezésű elemét egy programváltozóba. Az egyszerűség kedvéért jelöljük a programváltozót is X-szel.
* **Write\_item(X):**  
  Az X programváltozó értékét az adatbázis X elnevezésű elemébe rögzíti.

**Az írás és olvasás művelete részletesen**

1. **Read\_item(X):**
   * Megkeresi az X elemet tartalmazó lemezblokk címét.
   * Átmásolja ezt a lemezblokkot a fő memória pufferébe (amennyiben ez a blokk nincs már benne valamelyik fő memória pufferben).
   * Átmásolja az X elemet a pufferből az X nevű programváltozóba.
2. **Write\_item(X):**
   * Megkeresi az X elemet tartalmazó lemezblokk címét.
   * Átmásolja ezt a lemezblokkot a fő memória pufferébe (amennyiben ez a blokk nincs már benne valamelyik fő memória pufferben).
   * Átmásolja az X elemet az X nevű programváltozóból a puffer megfelelő területére.
   * Visszamásolja a frissített blokkot a pufferből a lemezre (rögtön vagy egy későbbi időpontban).

## Konkurens hozzáférésből adódó problémák (9.4)

1. **Az elveszett frissítés problémája**  
   Ugyanazt az adatbázis elemet két tranzakció művelete is használja, mely helytelen értéket eredményez.
2. **Ideiglenes frissítés (piszkos olvasás)**  
   Egy adatbázis elem frissítése után a tranzakció leáll, de még a visszaállítás előtt egy másik tranzakció kiolvassa az értékét.
   * **Piszkos olvasás:** Be nem fejezett tranzakcióból származó átmeneti érték olvasása.
   * Példa: T1 tranzakció leáll és emiatt az X értékét vissza kell állítani az új értékre.
3. **Helytelen összegzés**  
   Aggregáló függvény használata közben egy másik tranzakció frissítést végez, ami hibás összegzést eredményez.
4. **Megismételhetetlen olvasás problémája**  
   Egy tranzakció ugyanannak az elemnek két különböző olvasása során különböző értékeket kap, mert egy másik tranzakció megváltoztatta az elem értékét.

## Helyreállítás (Recovery) (9.5)

Valahányszor egy tranzakciót elküldünk egy DBMS-hez végrehajtásra, a rendszernek biztosítania kell, hogy:

* A tranzakció összes műveletének sikeres végrehajtása és az eredmények tartós rögzítése megtörténik, vagy
* Ha ez nem lehetséges, a tranzakciónak semmilyen hatása ne legyen.

**Tranzakciók típusai:**

* **Véglegesített (committed)**: A tranzakció sikeresen befejeződött, és az adatok tartósan rögzítésre kerültek.
* **Megszakított (aborted)**: A tranzakció nem fejeződött be, és az adatok visszaállításra kerültek.

**Mindent vagy semmit elv:**  
Ez az elv azt jelenti, hogy ha egy tranzakció bármely része nem hajtható végre, akkor az összes művelet visszavonásra kerül.

**Sikertelen tranzakció okai**

1. **Számítógép hiba (rendszerösszeomlás)**  
   A tranzakció futása alatt hardver (pl. memória), szoftver vagy hálózati hiba miatt meghiúsul a tranzakció sikeres befejezése.
2. **Tranzakció vagy rendszerhiba**  
   Valamelyik tranzakciós művelet fut hibára (pl. túlcsordulás, nullával osztás). Okozhatja hibás paraméterérték, vagy hibás programlogika is. A felhasználó leállíthatja a tranzakció futását.
3. **A tranzakció által felismert lokális hiba vagy kivétel**  
   Példa: a művelet adata nem található, fedezet hiány. Olyan kivétel, amelyet a program képes kezelni, és valójában nem is számít hibának.
4. **Konkurencia szabályozás kényszerítése**  
   A konkurencia szabályozó metódus megszakíthatja a tranzakciót a sorbarendezhetőség elvének megsértése miatt, vagy mert szükséges egy holtpont feloldásához. Általában később ezek automatikusan újra elindulnak.
5. **Lemezhiba**  
   Hibás olvasás vagy írás (esetleg az író/olvasó fej hibája) miatt egy lemez blokk elveszítheti adatát.
6. **Fizikai problémák, katasztrófák**  
   Egyéb okokból származó hibák, mint áramkimaradás, tűz, lopás, szabotázs, tévedésből származó adatfelülírás, stb.

**Megjegyzés:** Az első négy ok gyakoribb. Ezek esetén a rendszernek rendelkeznie kell elegendő információval a gyors helyreállításhoz.

**Tranzakciók állapota**

A DBMS helyreállító (recovery manager) a következő műveleteket követi nyomon:

* **BEGIN\_TRANSACTION**  
  A tranzakció futásának indítását jelzi.
* **READ** vagy **WRITE**  
  A tranzakció részeként elvégzett adatelem olvasását vagy írását jelzi.
* **END\_TRANSACTION**  
  Jelzi a **READ** és **WRITE** tranzakciós műveletek ill. a tranzakció futásának befejeződését.
* **COMMIT\_TRANSACTION**  
  A tranzakció sikeres befejeződését jelzi, azaz biztonságosan rögzítésre kerültek a változások és nem lesznek visszavonva.
* **ROLLBACK** (vagy **ABORT**)  
  A tranzakció sikertelen befejeződését jelzi, ami a már végrehajtott változtatások visszavonását eredményezi.

A képen szöveg, képernyőkép, Betűtípus, diagram látható

Automatikusan generált leírás

## A rendszer napló (system log) (9.6)

A rendszer naplója egy szekvenciális, csak továbbfűzhető állomány a lemezen, amely nincs kitéve más hibáknak, csak lemezhibáknak vagy katasztrófa okozta hibáknak. Tipikusan egy (vagy több) memória puffer tárolja a log fájl utolsó részét, azaz a bejegyzések először mindig ide kerülnek. Amikor megtelik, vagy bizonyos körülmények fennállnak, tartalma a lemezen lévő fájlhoz hozzáfűződik.

A lemezen lévő log fájl rendszeresen archiválásra kerül a katasztrofikus hibák okozta károk kivédése érdekében.

**Log bejegyzések (log records) típusai**

1. **[start\_transaction, T]**  
   A T tranzakció indítása.
2. **[write\_item, T, X, old\_value, new\_value]**  
   A T tranzakció megváltoztatta az X adatbázis elem értékét old\_value-ról new\_value-re.
3. **[read\_item, T, X]**  
   A T tranzakció elvégezte az X adatbázis elem olvasását.
4. **[commit, T]**  
   A T tranzakció sikeresen befejeződött, és a végrehajtott változtatások rögzíthetőek az adatbázisba.
5. **[abort, T]**  
   A T tranzakció sikertelen befejeződését jelzi, amely után a végrehajtott műveletek vissza lesznek vonva.

* T : egyedi tranzakciós azonosító

**Tranzakció commit point-ja**

Egy T tranzakció commit ponthoz ér, ha minden adatbázis művelet sikeresen végrehajtásra került, és minden műveletének az eredménye rögzítésre került a log fájlban.

A commit pont után a tranzakció végrehajtott (committed), azaz az eredményeinek véglegesen rögzítésre kellett kerülni az adatbázisban, és kiíródik egy **[commit, T]** rekord a log fájlba.

## Tranzakciók kívánatos tulajdonságai (ACID tulajdonságok) (9.7)

1. **Atomosság (atomicity)**  
   A tranzakció a feldolgozás atomi egysége; vagy teljes egészében végrehajtódik, vagy egyáltalán nem. Ha a tranzakció közben hiba történik, az összes módosítás visszavonásra kerül, mintha a tranzakció soha nem hajtódott volna végre.
2. **Konzisztencia megőrzés (consistency preservation)**  
   Egy tranzakció konzisztenciát megőrző, ha a teljes és önálló végrehajtása az adatbázist egy konzisztens állapotból egy másik, szintén konzisztens állapotba viszi át. A tranzakciók nem sérthetik az adatbázis előre meghatározott szabályait, vagy integritási feltételeit.
3. **Elkülönítés (isolation)**  
   Egy tranzakciónak látszólag más tranzakcióktól elkülönítve kell végrehajtódnia. Ez azt jelenti, hogy a tranzakció végrehajtása nem állhat kölcsönhatásban semelyik másik konkurensen végrehajtott tranzakcióval sem. Míg a tranzakció be nem fejeződik, más tranzakciók nem láthatják a köztes eredményeket (elkerülve például az "elmosódott" vagy "félkész" adatokat).
4. **Tartósság vagy állandóság (durability or permanency)**  
   Egy véglegesített tranzakció által az adatbázison véghezvitt módosításoknak megőrzésre kell kerülniük az adatbázisban. Miután a tranzakció sikeresen befejeződött (committed), annak eredményei tartósak lesznek, és semmilyen rendszerhiba nem veszélyeztetheti azok elvesztését.

## Tranzakció ütemezés (schedule vagy history) (9.8)

A tranzakció ütemezés a tranzakciókhoz tartozó műveletek végrehajtásának meghatározott sorrendje egy szálon, azaz az összes művelet úgy van sorrendbe állítva, hogy az összes tranzakció saját műveleteit végrehajtja, azokat nem keverve más tranzakciók műveleteivel.

**Ütemezés és konfliktusos műveletek:**

Egy ütemezés két művelete konfliktusban van, ha a következő feltételek mindegyike teljesül:

1. Különböző tranzakciókhoz tartoznak.
2. Ugyanazzal az X adatelemmel dolgoznak.
3. Legalább az egyik művelet write\_item(X), azaz módosítja az X adatelemet.

**Ütemezés típusok visszaállíthatóság alapján**

1. **Visszaállítható ütemezés**  
   Egy ütemezés visszaállítható, ha nem kerül véglegesítésre (commit) egyetlen T tranzakció sem, amíg nem véglegesítődnek azok a tranzakciók (T') is, amelyek olyan adatokat írnak, amelyeket T előzőleg beolvasott. Így biztosítható, hogy ha egy tranzakciót vissza kell állítani, akkor nem szükséges más tranzakciókat is visszaállítani.
2. **Kaszkádmentes ütemezés**  
   Minden tranzakció csak olyan adatot olvashat be, amelyet már egy véglegesített tranzakció írt ki. Ez elkerüli a piszkos olvasást, amikor egy tranzakció egy másik tranzakció nem véglegesített változtatásait olvassa.
3. **Szigorú ütemezés**  
   A tranzakciók sem olvasni, sem írni nem változtathatnak meg egy adatelem értékén, amíg az adott elem utolsó módosítójának tranzakciója nem lett véglegesítve (commit) vagy megszakítva (abort). Mivel minden tranzakciós művelet csak egy másik tranzakció teljes befejeződése után történhet, ez biztosítja, hogy a tranzakciók teljesen elkülönítve működjenek.

**Fontos megjegyzés:** Minden szigorú ütemezés kaszkádmentes, és minden kaszkádmentes ütemezés visszaállítható.

**Ütemezés típusok sorbarendezhetőség alapján**

1. **Soros ütemezés**  
   Egy ütemezés soros, ha minden tranzakció műveletei egy sorozatban, közvetlenül egymás után kerülnek végrehajtásra. Ez azt jelenti, hogy egy tranzakció összes művelete a másik tranzakciók műveletei előtt vagy után következik, nem keverednek.
2. **Sorba rendezhető ütemezés**  
   Egy ütemezés sorba rendezhető, ha a tranzakciók műveletei olyan sorrendben vannak, hogy azok ekvivalens egy soros ütemezéssel. Ez azt jelenti, hogy a tranzakciók közötti műveleti sorrendek átrendezése után a végső adatbázis állapot ugyanaz marad.

**Ekvivalencia típusok**

* **Eredmény ekvivalencia**  
  A két ütemezés eredmény ekvivalens, ha mindkettő ugyanazt a végső adatbázis állapotot eredményezi, függetlenül a tranzakciók műveleteinek sorrendjétől.
* **Konfliktus ekvivalencia**  
  A két ütemezés konfliktus ekvivalens, ha bármely két konfliktusos művelet sorrendisége megegyezik mindkét ütemezésben. Konfliktusos műveletek azok, amelyek ugyanazt az adatbázis elemet módosítják vagy olvassák.
* **Nézet ekvivalencia**  
  A két ütemezés nézet ekvivalens, ha a tranzakciók közötti adatmegosztás (pl. olvasás) hasonló módon történik, így az ütemezések ugyanazt az adatbázis állapotot eredményezik.

**Konfliktus sorba rendezhető ütemezés**: Egy ütemezés konfliktus ekvivalens egy soros ütemezéssel, ha a műveletek sorrendje úgy van elrendezve, hogy a tranzakciók műveletei közötti konfliktusok nem változtatják meg a végső állapotot.

## Megjegyzések a sorbarendezhetőségről (9.9)

1. **A sorbarendezhetőség nem jelenti azt, hogy az ütemezés soros.**
   * Az, hogy egy ütemezés sorbarendezhető, nem garantálja, hogy a tranzakciók műveletei egyetlen tranzakcióként, teljesen egymás után következően kerülnek végrehajtásra. Csak azt jelenti, hogy a műveletek sorrendje úgy van kialakítva, hogy azok ekvivalens egy soros ütemezéssel. Tehát lehetnek olyan ütemezések, amelyek nem sorosak, de konfliktus ekvivalens egy soros ütemezéssel.
2. **A sorbarendezhetőségből következik, hogy az ütemezés helyes.**
   * Ha egy ütemezés sorbarendezhető, akkor biztosított, hogy az adatbázis konzisztens marad a tranzakciók végrehajtása után. Ez a helyes ütemezésre vonatkozó alapelvet jelenti, mivel a tranzakciók egy soros ütemezés sorrendjére rendeződtek, így nem sérül a konzisztencia.
3. **A sorbarendezhetőséget nehéz ellenőrizni, mivel nem könnyű előre meghatározni, hogy egy ütemező hogyan fésüli össze a műveleteket.**
   * Az ütemezés ellenőrzése nem egyszerű, mivel az ütemező döntéseitől függ, hogy miként kapcsolja össze a különböző tranzakciók műveleteit. Az ütemezés eredménye és annak helyessége gyakran az ütemező implementációjától függ, így nem mindig könnyű előre látni vagy biztosítani a sorbarendezhetőséget.
4. **A gyakorlatban protokollokat használnak a sorbarendezhetőség biztosítására.**
   * Az adatbázis-kezelő rendszerek gyakran alkalmaznak különféle konkurencia-szabályozási protokollokat, hogy biztosítsák a tranzakciók megfelelő sorrendjét és elkerüljék a problémákat, mint a versenyhelyzetek és inkonzisztens adatállapotok. Ilyen például a zárolási protokollok alkalmazása.
5. **Egy ütemezés kezdete és vége nem meghatározható, ezért a teljes ütemezés ellenőrzését a véglegesített tranzakciókbeli műveletek ellenőrzésére redukálják.**
   * Mivel nem mindig könnyen meghatározható egy ütemezés kezdete és vége, a gyakorlatban inkább a véglegesített (committed) tranzakciók műveleteit vizsgálják, mivel azok már biztosan nem változnak, és azok biztosítják a helyes végső állapotot.
6. **Egy gyengébb ekvivalencia ütemezés az ún. nézet (view) ekvivalencia, melyet itt nem tárgyalunk.**
   * A nézet ekvivalencia egy olyan típusú ekvivalencia, ahol az ütemezések sorrendje nem szükséges, hogy teljesen konfliktus ekvivalens legyen, de a végső állapot, amelyet az ütemezések eredményeznek, ugyanaz. Ezt a típusú ekvivalenciát nem mindig tárgyalják részletesen, mivel nem minden esetben szükséges alkalmazni.
7. **Van algoritmus a konfliktus sorbarendezhetőség ellenőrzésére, amely a precedencia gráfon alapszik.**
   * A konfliktus sorbarendezhetőség ellenőrzésére léteznek algoritmusok, amelyek a precedencia gráf elvét alkalmazzák. Ebben a gráfban a tranzakciók műveletei közötti függőségeket ábrázolják, és ha a gráfban nincs ciklus, akkor az ütemezés konfliktus ekvivalens egy soros ütemezéssel.

## A konkurencia vezérlés (9.10)

A **konkurencia vezérlés** célja, hogy biztosítsa az adatbázisban végrehajtott tranzakciók közötti elkülönítést, hogy azok ne sértsék meg egymás eredményeit. Különösen fontos szerepe van a tranzakciók közötti konfliktusok (olvasás-írás, írás-írás) feloldásában, hogy az adatbázis konzisztens maradjon. Az alábbiakban kifejtjük a konkurencia vezérlés részleteit:

**Célja:**

1. **Elkülönítés kikényszerítése:**
   * A cél, hogy a tranzakciók között olyan mértékű elkülönítést biztosítsunk, hogy az adatbázisba történő műveletek ne zavarják meg egymás hatását. Például, ha két tranzakció (T1 és T2) ugyanazzal az adatelemmel dolgozik, akkor a konkurencia vezérlés biztosítja, hogy egyetlen tranzakció sem változtathatja meg az adatot a másik tranzakció végrehajtása előtt.
2. **Konzisztencia megőrzése:**
   * Az adatbázis konzisztens állapota biztosított, ha minden tranzakció a megfelelő módon kerül végrehajtásra, tehát azok nem okoznak ellentmondásos állapotokat.
3. **Konfliktusok feloldása:**
   * Az olvasás-írás és írás-írás konfliktusok feloldása révén biztosítjuk, hogy a tranzakciók ne sértsék meg az adatbázis integritását. Például ha T1 és T2 tranzakciók egyszerre próbálnak hozzáférni ugyanahhoz az adatelemhez (A), akkor a konkurencia vezérlés eldönti, hogy melyik tranzakció férhet hozzá, és ha szükséges, visszaállítja a tranzakciókat vagy várakoztatja őket.

**A kétfázisú zárolás módszere**

A kétfázisú zárolás egy olyan módszer, amely a tranzakciók biztonságos végrehajtását segíti elő. A következő kulcsfontosságú műveleteket alkalmazza:

1. **Lock(X) - Zárolás:**
   * Ez a művelet biztosítja, hogy egy tranzakció kizárólagosan hozzáférhessen egy adatbázis elemhez (X), hogy elvégezhesse rajta a kívánt műveletet, legyen szó olvasásról vagy írásról.
2. **Unlock(X) - Feloldás:**
   * Ez a művelet törli a zárolást egy adatbázis elemen (X), lehetővé téve más tranzakciók számára, hogy hozzáférjenek ahhoz.

Ez a módszer nagyon szigorú, és gyakorlatban nem mindig alkalmazható, mivel túlzottan megkötötte a tranzakciók végrehajtását, ami a teljesítményt is csökkentheti.

**Megosztott és kizárásos zárolás**

A **megossztott zárolás (shared lock)** és **kizárásos zárolás (exclusive lock)** alapvető különbsége a következő:

* **Megosztott zárolás (shared lock / read\_lock):**
  + Több tranzakció is olvashatja ugyanazt az adatelem (X) egyszerre, ha csak olvasási jogot kérnek rá. Azonban, ha valaki írni akar ugyanarra az adatelemre, akkor nem adható meg megosztott zárolás.
* **Kizárásos zárolás (exclusive lock / write\_lock):**
  + Ha egy tranzakció kizárásos zárolást kér egy adatbázis elemre (X), akkor más tranzakciók nem férhetnek hozzá ahhoz az elemhez, sem olvasási, sem írási jogon. Csak a tranzakció, amelyik a kizárásos zárolást kérte, végezhet el rajta műveletet.

**Zárolás kezelés**

A **Lock Manager** feladata, hogy kezelje a zárolásokat az adatelemeknél. A zárolásokat egy **zárolási táblán** (lock table) tárolja, amely nyomon követi, hogy mely tranzakciók kértek zárolásokat, és milyen típusú zárolások vannak érvényben egy-egy adatbázis elemre.

## Az adattárház célja (10.1)

1. **Eltérés a tradicionális adatbázisoktól**
   * A hagyományos adatbázisok elsődleges célja az **adatok integritásának biztosítása** adatmódosítások során.
   * Nem a lekérdezések optimalizálására készültek.
2. **Az adattárház sajátosságai**
   * Az adattárház felhasználói általában **adatokat olvasnak**, nem írnak vagy módosítanak.
   * **Nagymennyiségű adat gyors elérése** a legfontosabb elvárás.
3. **Adatok forrásai és elemzési igények**
   * Az adattárházban tárolt adatok **több forrásból származnak** (pl. különböző rendszerek, adatbázisok).
   * Az elemzések jellemzői:
     + **Ismétlődő jellegűek**
     + **Típusuk előre megjósolható**
4. **Döntéshozás támogatása**
   * Nagy szükség van olyan eszközökre, amelyek képesek:
     + **Historikus adatokon alapuló információkat szolgáltatni**
     + Gyors és **megbízható döntéshozást** támogatni.
5. **Az adattárház technológiák**
   * Az adattárházak és az **OLAP (Online Analytical Processing)** rendszerek:
     + Az adatokat feldolgozzák és elemzik.
     + **A döntéshozást támogatják elsődlegesen.**

## Az adattárház definíciója (10.2)

***W. H. Inmon definíciója:***

**"Az adattárház egy tematikus, integrált, az adatok történetiségét tároló, tartós adatrendszer, amelynek fő célja az adatokból történő hatékony információkinyerés biztosítása, elsősorban a döntéshozatali folyamatok támogatása céljából."**

**Az adattárház jellemzői**

**1. Tárgyorientált, tematikus (Subject-Oriented):**

* Az adattárház az **elemzési követelményekre és a döntéshozatal különböző szintjeire** fókuszál.
* Példák tematikákra: eladás, ügyfélviselkedés, termékek, rendelés.
* **Eltérés a hagyományos adatbázisoktól:**
  + A hagyományos adatbázisok funkcióorientáltak (pl. eladás regisztrálása).
  + Az adattárház tematikus, azaz a döntéshozók szempontjaira fókuszál.
* Példák különböző cégek tematikáira:
  + **Biztosítási cég:** ügyfél, kötvény, követelés, prémium.
  + **Gyártó cég:** termék, rendelés, szállító, nyers áruk.
  + **Kiskereskedő:** termék, értékesítés, szállító.

**2. Integrált:**

* Az adattárház **több heterogén adatforrásból** veszi át az adatokat.
* Az adatok **szabványos formára alakítva** kerülnek az adattárházba.
  + Egységes kulcsstruktúra és adatmegjelenítési mód szükséges.
  + Például az adattárház képes felismerni, ha különböző rendszerekben eltérő jelölésekkel ugyanarról az adatról van szó.
* Az integráció biztosítja az **adatok konzisztenciáját** az adattárházban.

**3. Nem illékony (Non-Volatile):**

* Az adattárházba bevitt adatok **tartósan tárolódnak**, csak explicit törlés esetén tűnnek el.
  + Az adatok általában **5-10 évig vagy tovább** megőrződnek.
* Az adattárházban **minden adat pillanatfelvétele időbélyeggel ellátva kerül tárolásra.**
* Nem szokás gyorsan érvénytelenné váló adatokat bevinni az adattárházba.

**4. Időfüggő (Time-Variant):**

* Az adattárház adatai **történeti adatokat** tartalmaznak, amelyek több éves tevékenységeket ölelnek fel.
* Az adatokat időpontok és időintervallumok szerint kezelik, követve a forrásrendszerek változásait.
* **Eltérés a hagyományos adatbázisoktól:**
  + Hagyományos adatbázis: a rekordok frissülnek vagy törlődnek.
  + Adattárház: az új adat betöltésekor a művelet **időbélyege** is rögzül.
  + Az adatok nem módosulnak, helyette új rekordok jönnek létre egy másik időbélyeggel.

**Röviden:** Több forrásból gyűjtött, egységesített sémával tárolt információk összessége.

## Az adattárházak által támogatott alkalmazások (10.3)

1. **OLAP (Online Analytical Processing):**
   * Az adattárházban lévő **komplex adatok elemzésére szolgáló eszköz**.
   * Lehetővé teszi az adatok különböző dimenziók szerinti elemzését, például idő, földrajzi hely, terméktípus szerint.
   * Támogatja az üzleti folyamatok részletes vizsgálatát és összehasonlítását.
2. **DSS (Decision Support Systems) / EIS (Executive Information Systems):**
   * A **vezető döntéshozók munkáját támogatja** az összetett, stratégiai döntések meghozatalában.
   * Könnyen érthető jelentésekkel, diagramokkal és kulcsfontosságú mutatókkal segíti a gyors döntéshozatalt.
3. **Data Mining (Adatbányászat):**
   * **Tudásfeltárásra** szolgáló eszköz, amely a meglévő adatokban rejtett minták, kapcsolatok és trendek felfedezésére irányul.
   * Az **ad hoc új tudás keresését** jelenti, például vásárlói szokások elemzése vagy anomáliák felfedezése érdekében.
   * Példák:
     + Potenciális ügyfelek szegmentációja.
     + Csalásmegelőzési rendszerek fejlesztése.

**Az adattárház jellemzői:**

* **Fő cél:**  
  Az adattárházakat az adatok **kinyerésére (extract), feldolgozására és megjelenítésére** tervezték, az elemzés és a döntéshozás támogatása érdekében.
* **Idősorok és trendek elemzése:**  
  Nagy hangsúlyt fektet a **történeti adatokra**, amelyek lehetővé teszik az időbeli változások és trendek vizsgálatát. Az adatfrissítés általában **inkrementális**.
* **Adatforrások:**
  + **Több adatforrásból** származó, **nagymennyiségű adatot** kezel.
  + Az adatforrások lehetnek egymástól független rendszerek, különböző platformok, eltérő adatmodellek, vagy akár egyszerű állományok.

## ****Összehasonlítás: Hagyományos adatbázis vs. Adattárház (10.4)****

|  |  |  |
| --- | --- | --- |
| Szempont | Hagyományos adatbázis | Adattárház |
| Adatmodell | Relációs, normalizált | Csillagséma, hópehelyséma |
| Orientáció | Tranzakció | Elemzés |
| Funkció | Napi műveletek | Döntés támogatás |
| Felhasználók száma | Sok (ezer) | Kevés (száz) |
| Adatbázis mérete | Néhány GB | 100 GB-tól több TB-ig |
| Párhuzamossági szint | Magas | Alacsony |
| Módosítás gyakorisága | Magas | Nincs módosítás |
| Adat redundancia | Alacsony | Magas |
| Felhasználó típusa | Irodai dolgozó | Vezető, döntéshozó |
| Használat | Megjósolható, ismétlődő | Ad hoc, nem strukturált |
| Adattartalom | Aktuális, részletes | Történeti, összegzett |
| Adatok szervezése | A működésnek megfelelően | Az elemzési problémáknak megfelelően |
| Adatstruktúra | Tranzakciókra optimalizált | Összetett lekérdezésekre optimalizált |
| Használat gyakorisága | Magas | Közepestől az alacsonyig |
| Elérés módja | Olvasás, insert, update, delete | Olvasás és hozzáfűzés (append) |
| Egy elérés hány rekordot érint | Keveset (10 sor) | Sokat (millió sort) |
| Válaszidő | Rövid | Lehet hosszú |
| Tervezés és megvalósítás | A teljes rendszert egy időben | Inkrementális |

## Az adattárházak koncepcionális felépítése (10.5)

**Az adattárház kezelésének folyamata:**

1. **Adatok tisztítása és újraformázása (ETL):**
   * Az **ETL (Extract, Transform, Load)** folyamat három lépésből áll:
     + **Extract (Kinyerés):** Az adatok kinyerése különböző forrásokból.
     + **Transform:** Az adatok tisztítása, szabványosítása, egységesítése az adattárház formátumához.
     + **Load:** Az adatok betöltése az adattárházba.
   * Az adattisztítás során a hibás vagy hiányos adatokat kiszűrik, és a szükséges formázási műveleteket végzik el.
2. **Elemzési eszközök és alkalmazások:**  
   Az adattárházra épülő rendszerek képesek új és releváns információkat generálni:
   * **OLAP (Online Analytical Processing):** Az adatok többdimenziós elemzésére szolgál.
   * **Data Mining (Adatbányászat):** Rejtett minták és szabályok felfedezése az adatokban.
   * **DSS (Decision Support System) és EIS (Executive Information System):** A döntéshozók munkáját segítik az elemzési eredményekkel.

**Tipikus adattárház keretrendszer:**

* **Adatpiac (Data Mart):**
  + Az adattárház részét képezi, de egy adott témára vagy részterületre fókuszál.
  + Példák: Értékesítési adatpiac, ügyféladatokat kezelő adatpiac.
  + **Hatókör:**
    - Az adatpiac: **osztályszintű**, azaz kisebb, specifikus elemzési igényeket szolgál ki.
    - Az adattárház: **szervezeti szintű**, az egész vállalatot lefedi és hosszú távú stratégiai elemzéseket támogat.

## ****Adattisztítás folyamata az adattárházakban (10.6)****

Az **adattisztítás** célja, hogy a forrásadatokat a lehető legjobb minőségűvé tegye az adattárházba történő betöltés előtt. Ez biztosítja a konzisztens, egységes és elemzésre alkalmas adatokat.

**A tisztítás lépései:**

1. **Elemekre bontás (elementizing):**
   * Az adatokat kisebb, **atomi részekre** bontják.
   * Példa: Egy cím esetében az egyes elemek elkülönítése:
     + **Város**, **irányítószám**, **utca**, **házszám**, **emeletszám**, **ajtószám**.
2. **Szabványosítás (standardizing):**
   * **Egységes jelölések** bevezetése az adatok konzisztenciájának biztosítása érdekében.
   * Példa:
     + A különböző formátumban megadott címek egységesítése, pl.:
       - „krt.” → „körút”
       - „utca” → „u.”
   * Ez segíti a későbbi feldolgozást és elemzést.
3. **Verifikálás (verifying):**
   * Az **adatok konzisztenciájának ellenőrzése** a szabványosítás után.
   * Példa:
     + Egy lakcímnél ellenőrizzük, hogy az **irányítószám** megfelel-e a megadott **város** nevének.
   * Ez segít kiszűrni a hibás adatokat.
4. **Illesztés (matching):**
   * Az aktuálisan vizsgált adat vagy rekord (vagy annak egy része) **összevetése a céladatbázissal**.
   * Példa:
     + Egy ügyfél neve és címe alapján ellenőrizhető, hogy az adattárházban már szerepel-e.
     + Ha az ügyfél már létezik, a rendszer ellenőrzi, hogy a megadott telefonszám, lakcím vagy más adat helyesen párosítható-e az ügyfélhez.
   * Az illesztés során belső **korrelációkat** tárnak fel, hogy az adatok mennyire felelnek meg az adattárházban már tárolt **szabályszerűségeknek**.
5. **Dokumentálás (documenting):**
   * A tisztítás során végzett műveleteket és az adatok módosítását **rögzíteni kell**.
   * Ez általában a **metaadatok** kiegészítésével vagy módosításával történik.
   * Példa:
     + Az adat tisztításának ideje, a szabványosított formátum rögzítése, és az esetleges illesztési szabályok dokumentálása.

**Az adattisztítás előnyei:**

* **Konzisztens adatok:** Az adattárház adatai pontosak, egységesek, és alkalmasak az elemzésekre.
* **Hibák csökkentése:** Kevesebb duplikált vagy hibás adat.
* **Elemzési pontosság:** Az adattisztítás alapja az, hogy az elemzési folyamat eredményei relevánsak és pontosak legyenek.

## Végfelhasználói eszközök az adattárházakban (10.7)

**OLAP eszközök**

* **Ad hoc lekérdezések** támogatása, amelyek rugalmas, nem előre meghatározott módon készíthetők el.
* A felhasználók saját elemzési igényeik szerint állíthatnak össze lekérdezéseket.

**Riportoló eszközök**

* **Előre definiált lekérdezések** alapján készített jelentések.
* Példák: Értékesítési jelentések, havi pénzügyi statisztikák.

**Statisztikai eszközök**

* Az adatok **statisztikai módszerekkel** történő elemzése.
* Példa: Korrelációelemzés, trendvizsgálat.

**Adatbányászati eszközök**

* **Rejtett minták és szabályok felfedezése** az adatokban.
* Cél: Értékes tudás feltárása, például ügyfélviselkedési minták azonosítása.

## Az adattárházak osztályozása (10.8)

Az adattárházak osztályozása az adatmennyiség és a funkciók alapján történik.

**1. Vállalatszintű adattárház**

* Nagy volumenű, átfogó adattárház, amely az egész vállalatot lefedi.
* **Jellemzői:**
  + **Nagy projektek**, amelyek jelentős idő- és erőforrás-befektetést igényelnek.
  + Több osztály adatait integrálja, és szervezeti szintű döntéshozást támogat.

**2. Virtuális adattárház**

* Az adatok a forrásadatbázisokban maradnak, az adattárház materializált nézeteken keresztül érhető el.
* **Jellemzői:**
  + Gyors elérés.
  + Nincs fizikai adatmásolás, az adatok dinamikusan generálódnak a lekérdezések során.

**3. Logikai adattárház**

* Az adatok **egyesítésére, terjesztésére és virtualizációjára** épít.
* **Jellemzői:**
  + Nincs teljes adattárolás egy helyen, az adatok különböző forrásokból érhetők el.
  + Rugalmas és költséghatékony megoldás.

**4. Adatpiac (Data Mart)**

* Az adattárház **egy részterületére fókuszál**, például egy osztály vagy részleg számára készült.
* **Jellemzői:**
  + Kisebb méretű, kevesebb témát fed le.
  + Gyorsabban kialakítható, mint egy vállalatszintű adattárház.
  + Példa: Marketing részleg adatpiaca az ügyféladatok és kampányinformációk tárolására.

## ****Adattárházak építése (10.9)****

Az adattárház építése egy összetett folyamat, amely magában foglalja a célok, a források és a felhasználói igények alapos megértését. Az építési folyamat során az alábbi szempontokat és lépéseket kell figyelembe venni:

**Az adattárház tervezésének kulcsfontosságú szempontjai**

1. **Felhasználási cél meghatározása**
   * Az adattárház tervezőjének meg kell értenie, hogy milyen típusú lekérdezések és elemzések lesznek előtérben.
   * Példák: Marketingorientált elemzések, termékfogyasztói adatok elemzése, vagy non-profit szervezeteknél az adományokra fókuszáló elemzések.
2. **Ad-hoc lekérdezések támogatása**
   * A tervezésnek biztosítania kell a gyors és hatékony, nem előre meghatározott lekérdezések lehetőségét.
3. **Megfelelő sémaválasztás**
   * Példa: Csillagséma vagy hópehelyséma, attól függően, hogy milyen típusú elemzéseket terveznek.

**Az adatok begyűjtése az adattárház számára**

1. **Adatok kinyerése több forrásból**
   * Az adatok különböző, heterogén rendszerekből származhatnak, például tranzakciós adatbázisokból, külső adatforrásokból vagy egyszerű fájlokból.
2. **Adatok formázása és egyeztetése**
   * Az adatnevek, jelentések és tartományok harmonizálása a források között.
   * Az adatok egységes struktúrába rendezése.
3. **Adattisztítás**
   * Az adatok ellenőrzése és javítása:
     + **Elemekre bontás** (elementizing).
     + **Szabványosítás** (standardizing).
     + **Verifikálás** (verifying).
     + **Illesztés** (matching).
   * A tisztítás gyakran munkaigényes folyamat, és nehéz teljesen automatizálni.
4. **Adatok illesztése az adattárház adatmodelljéhez**
   * Az adatoknak kompatibilisnek kell lenniük az adattárházban használt sémaformátummal.
5. **Adatok betöltése és frissítése**
   * A frissítési elvek figyelembevétele:
     + Az adatok frissessége.
     + Az adattárház elérhetőségének követelményei.
     + Az adatbetöltési folyamat hatékonysága (beleértve az indexek újraépítését).

**Adatok tárolása az adattárházban**

1. **Az adattárház adatmodelljének megfelelő struktúra létrehozása**
   * Az adatok tárolása időbélyegekkel, a történeti elemzések támogatására.
2. **Elérési utak és indexek karbantartása**
   * Az adatok gyors elérhetőségének biztosítása.
3. **Frissítés és adatkezelés**
   * Az új adatok hozzáfűzése az adattárházhoz.
   * Az időfüggő adatok megfelelő kezelése az elemzési követelmények szerint.

**Tervezési szempontok az adattárház környezetében**

1. **Felhasználók és használati célok azonosítása**
   * Milyen típusú felhasználók (vezetők, elemzők) fogják használni az adattárházat, és milyen kérdésekre keresnek választ.
2. **Adatmodell kialakítása**
   * Az adatmodellnek illeszkednie kell a tervezett elemzési problémákhoz.
3. **Források elemzése**
   * Az elérhető források és az infrastruktúra kapacitásainak meghatározása.
4. **Metaadat-kezelés**
   * Az adatokhoz tartozó információk (metaadatok) pontos tervezése és nyomon követése.
5. **Moduláris komponensek kialakítása**
   * A könnyű bővíthetőség és karbantarthatóság érdekében.
6. **Elosztott és párhuzamos architektúrák figyelembevétele**
   * Az elosztott adattárházak lehetővé teszik az autonóm adattárolási egységek együttműködését.

## ****Az adattárházak implementálásának nehézségei (10.10)****

Az adattárházak bevezetése és működtetése jelentős kihívásokat tartogat, amelyek az időigényes fejlesztéstől az összetett adminisztratív feladatokig terjednek. Az alábbiakban összefoglaljuk a legfontosabb nehézségeket:

**Időigényes fejlesztési folyamat**

* Az adattárház építése évekig is eltarthat, különösen nagyméretű, vállalati szintű megoldások esetén.
* Az időigényesség oka:
  + Több forrásrendszer integrációja.
  + Adattisztítás, egységesítés és adatmodell-tervezés.
  + Az üzleti követelmények és az informatikai megoldások összehangolása.

**Minőség és konzisztencia fenntartása**

* Az adatok tisztasága és következetessége kulcsfontosságú.
* Problémák:
  + Heterogén forrásokból származó adatok következetlensége.
  + Az adatduplikációk és hibák azonosítása és javítása.

**Változó üzleti igények**

* Az adattárházaknak rugalmasnak kell lenniük az üzleti igények változásaira:
  + Új adatforrások integrálása vagy meglévők eltávolítása.
  + Az adattárházat úgy kell megtervezni, hogy ezek a változások minimális átszervezést igényeljenek.

**Széleskörű adminisztrációs ismeretek**

* Az adattárház adminisztrációja összetettebb, mint a hagyományos adatbázisoké:
  + Nagyobb adatmennyiségek kezelése.
  + Időalapú adatok tárolása és frissítése.
  + Az adatbetöltési folyamatok optimalizálása és az ETL-folyamatok karbantartása.

## Üzleti Intelligencia (Business Intelligence – BI) (10.11)

Az üzleti intelligencia az adattárházak által támogatott elemzési eszközök és technikák gyűjteménye. Fő célja, hogy a nyers adatokat értelmezhető és hasznos információvá alakítsa át.

**BI alkalmazások területei**

1. **Közvetlen lekérdezés és riportkészítés**
   * A felhasználók közvetlen hozzáféréssel kérhetnek le adatokat vagy készíthetnek jelentéseket.
2. **Adatbányászat**
   * Rejtett mintázatok és összefüggések feltárása az adatokból.
3. **Standard riportok**
   * Előre definiált, rendszeresen generált riportok.
4. **Elemző alkalmazások**
   * Összetett, ad-hoc elemzési műveletek támogatása.
5. **Dashboardok és scoreboardok**
   * Vizualizációs eszközök (diagramok, statisztikák, mutatók) az adatok gyors áttekintéséhez.
6. **Működési (Operational) BI alkalmazások**
   * Olyan felhasználók számára készült, akik napi feladataik során használják az adatokat. Példa:
     + Ügyfélszolgálatosok régi ügyféladatokat keresnek vissza.

**Az adattárházak és a BI kapcsolata**

* Az adattárház az üzleti intelligencia alapját képezi, mivel a BI-eszközök az adattárházban tárolt információkra támaszkodnak.
* A BI célja az adatok felhasználásának optimalizálása, hogy a szervezet pontosabb és gyorsabb döntéseket hozzon.

## NoSQL Adatbázisok (11.1)

* **NoSQL**: "Not Only SQL" vagy "Not SQL", gyűjtőnév, amely több, nem relációs alapú adatbázistípusra utal.
* **Jellemzők**:
  + **Nem relációs szerkezet**: Az adatok meghatározzák a felépítést, nem a hagyományos táblák és kapcsolatok.
  + **Skálázhatóság**: Képesek horizontálisan skálázódni, azaz több szerver hozzáadásával a teljesítmény növelhető.
  + **Elosztott működés**: Az adatbázisok több szerveren is működhetnek párhuzamosan.
  + **Változó adatstruktúrák**: Rugalmasan kezelik az adatokat, amelyek nem kötődnek fix struktúrához.
  + **Óriási adatmennyiség kezelése**: Kifejezetten nagy adatok kezelésére lettek tervezve.
* **Közös tulajdonságok**:
  + Felhasználás-központú tervezés.
  + Jobb horizontális skálázhatóság, mint a relációs adatbázisoknál.
  + Elosztott rendszer, több szerverre osztva az adatokat.

## Elosztott Tárolás Típusai (11.2)

Az elosztott tárolás két fő megközelítést alkalmaz, mindkettő különböző célokat szolgál:

**1. Replikáció**

* **Leírás**: Ugyanazok az adatok több példányban tárolódnak különböző szervereken, így az adattartalom minden szerveren azonos.
* **Cél**:
  + **Biztonság**: Ha egy szerver kiesik, az adatok nem vesznek el, mert több példány is létezik.
  + **Másodlagos cél**: Terhelés megosztása, különösen olvasási műveleteknél, hogy a rendszer ne terhelődjön túl.

**2. Sharding**

* **Leírás**: Az adatok egy példányban tárolódnak, de több szerveren, és a szerverek adattartalma különbözik (kivéve, ha replikációval kombinálják).
* **Cél**:
  + **Terhelés megosztása**: Minden szerver saját adatot tárol, és az olvasási-írási műveletek is rájuk terhelődnek.

**3. Kombinált megoldás**

* Egyes adatbázisok mind a **replikációt**, mind a **shardingot** alkalmazzák a biztonság és a terhelés megosztásának érdekében.

## CAP Tétel (Brewer Tétel) (11.3)

Eric Brewer 1999-es tételét (CAP tétel) az elosztott rendszerek három alapvető képessége között az alábbiakban foglalja össze:

Egy elosztott rendszer legfeljebb két alapvető képességet tud egyszerre megvalósítani a három közül:

* **Konzisztencia** (Consistency)
* **Rendelkezésre állás** (Availability)
* **Particionálás tűrés** (Partition tolerance)

Ez azt jelenti, hogy egy rendszer választhat ezek közül két tulajdonságot, de nem képes mindháromra egyszerre biztosítani teljeskörű támogatást.

**CAP Tétel – Konzisztencia**

* **Jelentés**: Az adatok több példányban vannak tárolva (replikáció), különböző csomópontokon.
* **Konzisztencia**:
  + Minden csomóponton ugyanazt az (legfrissebb) adatot kell visszaadni egy adott időpontban.
  + Ha olvasás történik bármely csomópontból, az azonos értéket ad vissza, vagy hibát jelez.
  + **Probléma**: A szinkronizálás időbe telik, így teljes mértékű konzisztenciát nem mindig lehet biztosítani.

**CAP Tétel – Rendelkezésre állás**

* **Jelentés**: Az adatok folyamatosan elérhetőek, minimális késleltetéssel, és minden kérésre választ ad a rendszer (sikeres vagy sikertelen válasz).
* **Jellemzők**:
  + **Konzisztenciát** nem garantál!
  + Ha egy rendszerben egy kritikus komponens kiesik, nem eredményez működésképtelenséget (például redundáns csomópontok).
  + **Master-Slave modell**: A master csomópont szolgálja ki az írási és olvasási kéréseket, míg a slave csomópontok csak olvasást végeznek. Ha a master kiesik, a slave csomópontok választanak egy új master-t.

**CAP Tétel – Particionálás Tűrés**

* **Jelentés**: A rendszer akkor is működőképes marad, ha a csomópontok közötti kommunikáció megszakad, vagy üzenetek vesznek el.
* **Jellemzők**:
  + Ha a hálózat két csomópontja között megszakad, vagy valamelyik csomópont hibát szenved el, a rendszer képes válaszolni a kliensek kérésére.
  + A hibák után a rendszernek képesnek kell lennie a szinkronizálásra.
  + **Fontosság**: Mivel minden elosztott rendszerben előfordulhat hálózati hiba, a partíció tűrés alapvető tulajdonság.

## ****Lehetőségek a CAP tétel szerinti kompromisszumokra (11.4)****

**1. Elhagyni a particionálás tűrést**

* **Leírás**: Ebben az esetben az adatokat nem osztjuk el több csomópont között, hanem mindent egyetlen szerveren, vagy egyetlen, atomi módon elbukó egységen tárolunk. Ez megoldhatja a partíciók elosztásával járó problémákat, de **skálázhatósági problémát** okozhat.
* **Példa**: Az összes adat egy gépre történő koncentrálása (például egy rack-en) nem biztosít horizontális skálázódást, tehát a rendszer nem tud hatékonyan kezelni hatalmas adatforgalmat vagy terhelést.

**2. Elhagyni a rendelkezésre állást**

* **Leírás**: Ebben az esetben a rendszer hajlandó leállni egy esemény, hiba, vagy karbantartás során, hogy biztosítsa az adatok konzisztenciáját. A rendszer nem válaszol a kérésekre, amíg a konzisztencia nem áll helyre, így bizonyos időre **elérhetetlenné válik**.
* **Példa**: Ha egy adatbázisban egy hiba miatt az adatok nem egyeznek, akkor a rendszer "várakozik", amíg az adatokat sikerül szinkronizálni, és csak ekkor biztosítja a kéréseket. Ezzel biztosítható a magas konzisztencia, de az alkalmazás időnként nem lesz elérhető.

**3. Elhagyni a konzisztenciát**

* **Leírás**: A konzisztencia elhagyása a gyakorlatban azt jelenti, hogy az adatok lehetnek ideiglenesen ellentmondásosak, amíg a rendszer azokat nem frissíti. Bár ez bizonyos esetekben elkerülhetetlen, nem minden esetben szükséges.
* **Példa**: Egy árucikk-adatbázis nem szükséges, hogy mindig konzisztens legyen. Ha például egy raktáros most tör össze valamit, akkor a készletadatok azonnal nem tükrözik a tényleges helyzetet. Ha két rendelés érkezik ugyanarra az árucikkre, az első rendelést ki tudják szolgálni, de a másodikat lehet, hogy nem azonnal, hanem csak egy későbbi beszerzés után.

**4. A rendelkezésre állás gyakran fontosabb, mint a konzisztencia**

* **Leírás**: Az üzleti és alkalmazási környezetek gyakran fontosabbnak tartják, hogy a rendszer folyamatosan elérhető legyen (rendelkezésre álljon), mint azt, hogy az adatok minden pillanatban tökéletesen konzisztens állapotot tükrözzenek. Ilyenkor a rendszer a válaszidőt optimalizálja, és biztosítja, hogy ne legyen adatvesztés, még akkor sem, ha a legfrissebb adatokat nem mindig azonnal tükrözi a rendszer.
* **Példa**: Online bolt esetében, ha egy árucikket eladnak, a rendelkezésre állás biztosítása érdekében a rendszer gyorsan lezárja az eladást, még akkor is, ha a készletadatok azonnali frissítése késlekedhet.

## ****NoSQL adatbázis típusok (11.5)****

**1. Kulcs-érték tárolók**

* **Leírás**: A kulcs-érték tárolók (Key-Value stores) a legegyszerűbb típusú NoSQL adatbázisok. Az adatokat (K, V) párokban tárolják, ahol K a kulcs és V az érték. Ez a típus lényegében egy asszociatív tömb, amely gyors keresést biztosít kulcsok alapján.
* **Jellemzők**:
  + Rugalmas adattípusok kezelése.
  + Jellemzően kulcs szerinti lekérdezések.
  + Adatok például: sessionid (munkamenet), userid (bevásárlókosár).
* **Mikor használjuk**:
  + Munkamenet adatok, bevásárlókosarak, felhasználói profilok, beállítások.
* **Mikor ne használjuk**:
  + Ha kapcsolatok vannak az adatok között, többműveletes tranzakciók szükségesek, vagy ha adatokon alapuló lekérdezések szükségesek.

**2. Oszlopcsaládok**

* **Leírás**: Az oszlopcsaládok adatbázisok tárolása a hagyományos táblákhoz hasonló, de sokkal rugalmasabb. Az adatokat oszlopok, oszlopcsaládok és időbélyeggel ellátott verziók formájában tárolják.
  + **Sorok felépítése**: Kulcs + oszlop (oszlopnév, érték, időbélyeg).
* **Jellemzők**:
  + Rugalmas adatmodellezés.
  + Az oszlopok időbélyeggel történő verziózása.
* **Mikor használjuk**:
  + Eseménynaplózás, tartalomkezelő rendszerek, blog platformok.
  + Számlálók, például weboldalak látogatottságának mérése.
  + Lejáró oszlopok (TTL - Time To Live).
* **Mikor ne használjuk**:
  + Ha ACID tranzakciókra van szükség, vagy ha gyakoriak a lekérdezések áttervezésére.

**3. Gráf adatbázisok**

* **Leírás**: A gráf adatbázisok a csomópontokat és irányított éleket használják az adatok tárolására, és kifejezetten az adatelemek közötti összefüggések modellezésére szolgálnak. A legfontosabb jellemzője, hogy az adatokat hálózatos (gráf) struktúrában tárolja, így könnyen reprezentálhatók komplex, hierarchikus kapcsolatok.
  + Csomópontok, kapcsolatok, tulajdonságok, címkék.
  + Kiválóan alkalmasak kapcsolatok gyors és hatékony keresésére.
* **Mikor használjuk**:
  + Összekapcsolódó adatok (pl. szociális hálózatok, céges hálózatok).
  + Útvonalválasztás, helyalapú szolgáltatások.
  + Ajánlói rendszerek.
* **Mikor ne használjuk**:
  + Ha globális gráfműveletek szükségesek, vagy ha sok csomópont módosítása történik.

**4. Dokumentumtárolók**

* **Leírás**: A dokumentumtárolók az adatokat komplex, hierarchikus dokumentumok formájában tárolják, amelyek kulcs-érték párokat, tömböket és beágyazott dokumentumokat tartalmazhatnak. A dokumentumok lehetnek JSON, BSON vagy XML formátumban.
  + **Jellemzők**:
    - Önállóan leíró dokumentumok.
    - Tetszőleges bonyolultságú sémák.
    - Az adatok tárolása általában kollekciókban történik.
* **Mikor használjuk**:
  + Eseménynaplózás.
  + Tartalomkezelő rendszerek, blog platformok.
  + Webes és valósidejű analitika.
  + E-kereskedelmi alkalmazások.
* **Mikor ne használjuk**:
  + Ha összetett tranzakciók és változó aggregátumokra vonatkozó lekérdezések szükségesek.