Adatbázis tervezés alapfogalmai

1.
    1. **Adatbázis (Database):** Egy szervezett gyűjtemény adatokból, amelyeket egy rendszerben tárolnak, hogy azokat könnyen elérhessük, kezelhessük és frissíthessük. Az adatbázisok célja, hogy strukturáltan és hatékonyan tárolják az adatokat.*Az adatbázis a következő implicit tulajdonságokkal rendelkezik:*
    - Az adatbázis a valós világ valamely részét reprezentálja, amelyet néha **minivilágnak** vagy **a modellezés tárgyának** nevezünk. A minivilágban bekövetkező változások megjelennek az adatbázisban.
    - Az adatbázis adatok logikailag összetartozó gyűjteménye a benne rejlő jelentéssel együtt. Az adatok egy véletlenszerű összességét nem tekinthetjük adatbázisnak.
    - Az adatbázist egy konkrét céllal tervezzük és építjük, továbbá konkrét céllal tárolt adatokkal töltjük föl. Jól meghatározott felhasználói csoport használja jól meghatározott, számukra értelmes céllal.
    2. **Adat:** alatt olyan ismert tényeket értünk, amelyek feljegyezhetők, és amelyeknek implicit jelentésük van. Ilyenek például az ismerőseink nevei, telefonszámai és címei.
    3. **Adatbázis-kezelő rendszer (DBMS):** Olyan szoftver, amely lehetővé teszi az adatok tárolását, lekérdezését, módosítását és kezelését. Példák: MySQL, PostgreSQL, Oracle Database.
    4. **Adatabsztrakció:** Az **adatabsztrakció** általánosan az adatok szervezésére és tárolására vonatkozó részletek elhagyását, illetve a leglényegesebb tulajdonságok kiemelését jelenti az adatok jobb megértése érdekében. Az adatbázis megközelítés egyik fő jellemzője, hogy támogatja az adatabsztrakciót, s így a különböző felhasználók az általuk preferált részletességi szinten láthatják az adatokat.
    5. **Adatmodell:** Ezt az absztrakciót az adatmodell biztosítja. Az **adatmodell** olyan eszközök összessége, amelyek segítségével leírható egy adatbázis szerkezete. Az *adatbázis szerkezetén* az adattípusokat, az adatok közötti kapcsolatokat és a rájuk vonatkozó megszorításokat értjük.
    6. **Az adatmodellek csoportosítása:** Számos adatmodellt terveztek, amelyeket az adatbázis szerkezetének leírására általuk használt eszközök típusai szerint csoportosíthatunk. A **magas szintű** vagy **koncepcionális adatmodellek** olyan eszközöket nyújtanak, amelyek közel állnak ahhoz, ahogyan a legtöbb felhasználó látja az adatokat, míg az **alacsony szintű** vagy **fizikai adatmodellek** által nyújtott eszközök részletesen leírják, hogyan tárolódnak az adatok a számítógépen. Az alacsony szintű adatmodellek által biztosított eszközök általában számítógépes szakemberek számára készültek, nem a tipikus végfelhasználók számára. A koncepcionális adatmodellek olyan fogalmakat használnak, mint például az egyed, a tulajdonság és a kapcsolat. Az **egyed** egy olyan valós világbeli objektumot vagy fogalmat reprezentál, amely szerepel az adatbázisban, mint például egy alkalmazott vagy egy projekt.
    7. A háromséma architektúra:
    A **belső szintnek** van egy **belső sémája**, amely az adatbázis fizikai tárolási szerkezetét írja le. A belső séma egy fizikai adatmodellt használ, és leírja az adattárolás összes részletét, valamint a hozzáférési utakat az adatbázishoz.
    A **koncepcionális szintnek** van egy **koncepcionális sémája**, amely a teljes adatbázis szerkezetét írja le a felhasználók összességének. A koncepcionális

séma elrejti a fizikai tárolási szerkezetek részleteit, csak az egyedek, adattípusok, kapcsolatok, felhasználói műveletek és megszorítások leírására korlátozódik. Általában egy reprezentációs adatmodellt szoktunk használni a koncepcionális séma leírására, amikor megvalósítunk egy adatbázist. Ez az *implementációs koncepcionális séma* gyakran egy *koncepcionális sématerven* alapul egy magas szintű adatmodellben.
A **külső (nézet) szint** számos **külső sémát (felhasználói nézetet)** foglal magában. Mindegyik külső séma az adatbázisnak egy olyan részét írja le, amely iránt egy bizonyos felhasználói csoport érdeklődik, és elrejti ez elől a felhasználói csoport elől az adatbázis többi részét. Az előző esethez hasonlóan mindegyik külső sémát jellemzően egy reprezentációs adatmodell felhasználásával implementáljuk, esetleg egy magas szintű adatmodellbeli külső sémára alapozva.

8. **Adatbázis-adminisztrátorok:** Minden olyan szervezetnél, ahol sok ember használja ugyanazokat az erőforrásokat, szükség van egy vezető adminisztrátorra, aki felügyeli és kezeli azokat. Adatbázis környezetben az elsődleges erőforrás maga az adatbázis, a másodlagos erőforrás pedig a DBMS és a kapcsolódó szoftverek. Ezen erőforrások adminisztrálása az **adatbázis-adminisztrátor** (**DBA**) feladata. A DBA felelős az adabázishoz való hozzáférés engedélyezéséért, az adatbázis használatának koordinálásáért és felügyeletéért, valamint a szükséges szoftver- és hardvererőforrások beszerzéséért. A DBA kérhető számon olyan problémák esetén, mint amilyen a biztonsági rés vagy a lassú válaszidő. Nagy szervezeteknél a DBA-t egy stáb segíti, akik ellátják ezeket a feladatokat.

9. **Adatbázis-tervezők:** Az **adatbázis-tervezők** feladata az adatbázisban tárolandó adatok azonosítása, illetve ezen adatok reprezentálásához és tárolásához a megfelelő adatszerkezetek kiválasztása. Ezeket a műveleteket jórészt még azelőtt végrehajtják, mielőtt az adatbázist ténylegesen implementálnák és feltöltenék adatokkal. Az adatbázis-tervezők felelőssége, hogy beszéljenek az adatbázis összes leendő felhasználójával abból a célból, hogy megértsék az igényeiket, és hogy egy olyan tervet hozzanak létre, amely megfelel ezeknek az igényeknek. A tervezők gyakran a DBA-t segítő csapat tagjai, akik más feladatokat kapnak, miután az adatbázis tervezése befejeződött. Az adatbázis-tervezők rendszerint elbeszélgetnek az egyes potenciális felhasználói csoportokkal, majd olyan **nézeteket** fejlesztenek az adatbázishoz, amelyek megfelelnek ezen csoportok adat- és feldolgozási követelményeinek. Ezután minden nézetet elemeznek és *integrálnak* más felhasználói csoportok nézeteivel. A végső adatbázistervnek képesnek kell lennie az összes felhasználói csoport igényeinek a támogatására.

10. **Végfelhasználók:** A **végfelhasználók** azok az emberek, akik a munkájuk során igénylik az adatbázishoz való hozzáférést lekérdezési, módosítási és jelentéskészítési célból; az adatbázis elsősorban értük létezik. A végfelhasználóknak több kategóriáját különböztetjük meg:

- Az **eseti végfelhasználók** csak időnként érik el az adatbázist, viszont minden alkalommal más-más információra lehet szükségük. Egy kifinomult adatbázis-lekérdező nyelvet használnak a kéréseik megfogalmazására; ők tipikusan közép- vagy felsővezetők, esetleg egyéb alkalomszerű böngészők.
-

- A banktisztviselők ellenőrzik a számlaegyenlegeket, és pénzkivételeket, -betéteket könyvelnek.
- Légitársaságok, szállodák és autókölcsönzők ügyintézői ellenőrzik a kapott kérések teljesíthetőségét, és megteszik a helyfoglalást.
- A szállítási cégek fogadóállomásain az ügyintézők csomagazonosítókat visznek be vonalkódok segítségével, valamint további leíró információkat billentyűzettel, így módosítva a megérkezett és a továbbszállítandó csomagok központi adatbázisát.A **naiv** vagy **parametrikus végfelhasználók** számottevő részét teszik ki az adatbázis végfelhasználóinak. Munkájuk főleg az adatbázis állandó lekérdezése és módosítása körül forog, típuslekérdezéseket és -módosításokat használva, amelyeket **dobozolt tranzakcióknak** nevezünk, és amelyeket előzőleg gondosan leprogramoztak és teszteltek. Az ilyen felhasználók különféle feladatokat látnak el:
- A **tanult végfelhasználók** (szakemberek) magukban foglalják a mérnököket, tudósokat, üzleti elemzőket és másokat, akik alaposan megismerkednek a DBMS lehetőségeivel abból a célból, hogy saját alkalmazásokat fejlesszenek az összetett igényeik kielégítésére.
- A **független felhasználók** személyes adatbázisokat tartanak karban kész programcsomagok segítségével, amelyek egyszerűen használható menüs vagy grafikus interfészekkel rendelkeznek. Ilyen például egy olyan adócsomag felhasználója, amely személyes pénzügyi adatok sokaságát tárolja adózási célból.

11. **Rendszerelemzők és alkalmazásprogramozók (szoftvermérnökök):** A **rendszerelemzők** határozzák meg a végfelhasználók, különösen a naiv felhasználók igényeit, majd rögzítik az ezen igényeknek megfelelő dobozolt tranzakciók specifikációit. Az **alkalmazásprogramozók** a megadott specifikációk alapján implementálják ezeket a tranzakciókat mint programokat, amelyeket azután tesztelnek, nyomkövetnek, dokumentálnak és karbantartanak. Az elemzőknek és a programozóknak — akiket közös néven **szoftverfejlesztőknek** vagy **szoftvermérnököknek** hívnak — a DBMS által nyújtott szolgáltatások teljes tárházával tisztában kell lenniük ahhoz, hogy feladatukat ellássák.

12. **Relációs adatbázis (RDBMS):** Egy olyan adatbázis, amely táblák (relációk) formájában szervezi az adatokat. Az adatok sorokban (rekordokban) és oszlopokban (attribútumokban) helyezkednek el. Kapcsolatok lehetnek a táblák között. Példák: MySQL, PostgreSQL.
13. **Tábla (Table):** Az adatok tárolására használt szerkezet egy relációs adatbázisban. Minden tábla sorokból és oszlopokból áll, ahol az oszlopok a mezők (attribútumok), a sorok pedig a rekordok.
14. **Rekord (Record):** Egy sor egy táblában, amely az adott tábla oszlopainak megfelelő értékeket tartalmazza. Egy rekord egy egységnyi adatot képvisel.

15. **Attribútum (Attribute):** Egy tábla oszlopai, amelyek az adatok tulajdonságait vagy jellemzőit írják le. Például egy „Felhasználók” táblában a „név” és az „e-mail cím” attribútumok lehetnek. Az ER modellben különböző fajta attribútumok léteznek: *egyszerű* vagy *összetett*, *egyértékű* vagy *többértékű*, illetve *tárolt* vagy *származtatott* attribútumok. Az **összetett attribútumok** kisebb részekre bonthatók, amelyek több, egymástól független jelentéssel bíró elemi attribútumot reprezentálnak. Azokat az attribútumokat, amelyeket nem bontunk részekre, **egyszerű** vagy **atomi attribútumoknak** nevezzük. Az összetett attribútumok hierarchiát alkothatnak; például a Lakcím tovább bontható négy egyszerű attribútumra: Utca, Házszám, Emelet és Ajtó attribútumokra. Az összetett attribútumok nagyon hasznosak olyan szituációk modellezésénél, amelyekben a felhasználó néha egységként hivatkozik az összetett attribútumra, máskor viszont külön-külön hivatkozik annak komponenseire. Ha az összetett attribútumra csak mint teljes egészre hivatkozunk, akkor nincs szükség arra, hogy komponensekre bontsuk.

    Egyértékű és többértékű attribútumok. A legtöbb attribútum egy egyedben csak egy értékkel rendelkezik; az ilyen attribútumokat **egyértékűnek** nevezzük. Például az Életkor egy személy egyértékű attribútuma. Bizonyos esetekben egy attribútum értékek egy halmazával rendelkezhet ugyanazon egyedben — például egy autó Színek attribútuma vagy egy személy Diplomák attribútuma. Egyszínű autók esetén a Színek attribútum egyetlen értékkel rendelkezik, míg a két színnel fényezett autók esetében két értékkel. Hasonlóan, egy személynek lehet, hogy egyetlen diplomája sincs, egy másik személynek egy, egy harmadiknak kettő vagy több diplomája lehet; ezáltal a Diplomák attribútum különböző személyeknél különböző *számú értéket* vehet fel. Az ilyen attribútumokat nevezzük **többértékűnek**. Egy többértékű attribútumnak lehet alsó és felső korlátja, amelyek behatárolják a különböző egyedeknél a felvehető értékek számát. Például az autó Színek attribútuma legalább egy és legfeljebb három értéket vehet fel, ha feltételezzük, hogy egy autó legfeljebb három színre fényezhető.

    Tárolt és származtatott attribútumok. Bizonyos esetekben kettő (vagy több) attribútum értékei kapcsolatban állnak egymással — például egy személy Életkor és Születési idő attribtútumai. Egy konrét személy egyed esetén az Életkor értéke meghatározható az aktuális (mai) dátum és az adott személyhez tartozó Születési idő értékéből. Az Életkor attribútumot emiatt **származtatott attribútumnak** nevezzük, és azt mondjuk, hogy a Születési idő attribútumból **származtatható**, amelyet pedig **tárolt attribútumnak** nevezünk. Egyes attribútumértékek *az egyeddel kapcsolatban álló más egyedekből* származtathatók; például egy OSZTÁLY egyed Dolgozók_száma attribútuma úgy származtatható, hogy megszámoljuk az adott osztályhoz tartozó (azaz ott dolgozó) alkalmazottakat. **NULL értékek.** Előfordulhat, hogy egy adott egyed valamelyik attribútumának nincs használható értéke. Például egy cím Ajtó attribútuma csak azon címek esetén játszik szerepet, ahol a lakásokat ajtónként számozzák; másfajta épületeknél, mint például a családi házaknál, nem. Hasonlóan, a Diplomák attribútum csak diplomával rendelkező személyeknél érdekes. Az ilyen helyzetekben egy speciális értéket alkalmazunk, amelyet NULL értéknek nevezünk. A családi házak címében az Ajtó attribútum, egy diplomával nem rendelkező személy esetén pedig a Diplomák attribútum lesz NULL értékű. A

NULL értéket használjuk akkor is, ha nem ismerjük egy adott egyed valamelyik attribútumának az értékét. A NULL érték előbbi formájának a jelentése az, hogy *nem alkalmazható*, míg az utóbbié az, hogy *ismeretlen*.
Az *ismeretlen* kategória további két esetre osztható. Az egyik esetben tudjuk, hogy az attribútumérték létezik, de *hiányzik* — például NULL értékű egy személy Magasság attribútuma. A másik esetben *nem tudjuk*, hogy az adott attribútumérték létezik-e — például NULL értékű egy személy Otthoni_telefonszám attribútuma.

16. **Elsődleges kulcs (Primary Key):** Egy tábla egyedi azonosítója. Az elsődleges kulcs garantálja, hogy a tábla minden sora egyedi legyen. Például egy „Felhasználók” táblában a „FelhasználóID” lehet az elsődleges kulcs.
17. **Külső kulcs (Foreign Key):** Egy attribútum, amely egy másik tábla elsődleges kulcsára mutat. Ez a kapcsolat létrehozásához szükséges a táblák között, hogy összekössük őket.
18. **Kapcsolatok (Relationships):** A relációs adatbázisokban a táblák közötti kapcsolatokat különböző típusok szerint határozzuk meg. Ezek a kapcsolatok határozzák meg, hogyan függnek össze az adatok a különböző táblákban.

A kapcsolatok foka (Degree of Relationship)

A kapcsolat foka azt írja le, hogy hány entitástípus vesz részt egy adott kapcsolatban. Ez megadja, hogy hány entitást köt össze az adott kapcsolat.

- **Bináris kapcsolat:** A kapcsolat két entitástípus között jön létre. Ez a leggyakoribb kapcsolatfajta adatbázisokban.
  - **Példa:** Egy **Diák** és egy **Tantárgy** közötti kapcsolat: egy diák beiratkozhat egy tantárgyra.
- **Ternáris kapcsolat:** A kapcsolat három entitástípus között áll fenn.
  - **Példa:** Egy **Diák**, egy **Tantárgy**, és egy **Tanár** közötti kapcsolat: egy diák egy tanár által oktatott tantárgyra iratkozhat be.
- **N-áris kapcsolat:** Ez a kapcsolat több mint három entitástípus között jön létre.
  - **Példa:** Ha egy adatbázisban tároljuk a diákokat, tantárgyakat, tanárokat és egyéb, pl. helyszíneket is, akkor az n-áris kapcsolatok révén mindezek összekapcsolhatók.

A kapcsolatok szorossága (Participation/Optionality)

A kapcsolatok szorossága azt jelzi, hogy az egyik entitás mindenképpen részt vesz-e a kapcsolatban, vagy opcionális a kapcsolata a másik entitással. Ezt gyakran teljes vagy részleges részvételnek nevezzük.

- **Teljes részvétel (Total Participation):**
  - Egy entitásnak mindenképpen részt kell vennie a kapcsolatban. Ez azt jelenti, hogy az entitás minden egyes példánya szerepel a kapcsolatban.
  - **Példa:** Egy **Rendelés** entitás mindig kötődik egy **Ügyfélhez** – azaz nem létezhet olyan rendelés, amelyhez ne tartozna ügyfél.
- **Részleges részvétel (Partial Participation):**

- Egy entitás példányainak nem kell szükségszerűen részt venniük a kapcsolatban. Ez azt jelenti, hogy az entitások közül néhány kapcsolódik, de nem feltétlenül mindegyik.
- **Példa:** Egy **Diák** nem feltétlenül vesz fel minden tantárgyat, tehát a diákoknak csak egy része kapcsolódik bizonyos tantárgyakhoz.

A kapcsolatok számossága (Cardinality)

Egy-az-egyhez (One-to-One) kapcsolat

- **Definíció:** Egy sor az egyik táblában csak egy sorhoz kapcsolódik egy másik táblában.
- **Példa:** Tegyük fel, hogy van egy **„Felhasználók”** tábla és egy **„Felhasználói_Profil”** tábla. Minden felhasználónak van egy profilja, és minden profil egy felhasználóhoz tartozik. Itt az egyes felhasználókhoz egyedi profil kapcsolódik.
  - **Felhasználók**: FelhasználóID, Név
  - **Felhasználói_Profil**: ProfilID, FelhasználóID (külső kulcs), ProfilKép

Egy-a-többhöz (One-to-Many) kapcsolat

- **Definíció:** Egy sor az egyik táblában több sorhoz kapcsolódik egy másik táblában.
- **Példa:** Van egy **„Rendelők”** tábla és egy **„Páciensek”** tábla. Egy rendelőben több páciens is megfordulhat, de egy páciens egyszerre csak egy rendelőhöz tartozik.
  - **Rendelők**: RendelőID, Név
  - **Páciensek**: PáciensID, Név, RendelőID (külső kulcs)

Több-a-többhöz (Many-to-Many) kapcsolat

- **Definíció:** Több sor az egyik táblában több sorhoz kapcsolódhat egy másik táblában.
- **Példa:** Képzeljünk el egy **„Tanfolyamok”** táblát és egy **„Diákok”** táblát. Egy diák több tanfolyamra is beiratkozhat, és egy tanfolyamot több diák is látogathat. Az ilyen kapcsolatokhoz szükség van egy összekapcsoló táblára (kapcsoló tábla), ami a kapcsolatokat kezeli.
  - **Tanfolyamok**: TanfolyamID, TanfolyamNév
  - **Diákok**: DiákID, Név
  - **Beiratkozások**: TanfolyamID (külső kulcs), DiákID (külső kulcs)

19. Redundancia az adatbázisokban

Redundancia akkor fordul elő, amikor ugyanazt az adatot több helyen, feleslegesen tároljuk egy adatbázisban. Bár bizonyos helyzetekben a redundancia hasznos lehet (pl. gyorsabb lekérdezések érdekében), általában kerülni kell, mivel több problémát is okozhat.

A redundancia káros következményei:

1. **Tárolási pazarlás (Storage Waste):**

- Ha ugyanazokat az adatokat többször tároljuk, több tárhelyet foglalunk le, ami különösen nagyobb adatmennyiségnél jelentős erőforráspazarlást jelenthet. Például egy **„Ügyfelek”** táblában ha minden rendeléshez újra és újra eltároljuk az ügyfél összes adatait (név, cím, stb.), az sok fölösleges adatot eredményez.

2. **Inkonzisztens adatok (Inconsistent Data):**
   - A redundancia növeli annak kockázatát, hogy az adatok különböző helyeken eltérő módon frissülnek. Ha egy adatot többször tárolunk, de csak egy helyen frissítjük, akkor ellentmondás keletkezhet, ami adatvesztéshez vagy hibás információkhoz vezethet.
   - **Példa:** Ha egy alkalmazott címét több különböző táblában tároljuk, és csak az egyik helyen frissítjük, akkor a többi helyen régi adat marad, és az adatbázis inkonzisztens lesz.
3. **Karbantartási nehézségek (Maintenance Overhead):**
   - Ha az adatok több helyen szerepelnek, és egy változtatás szükséges, akkor minden érintett táblát vagy rekordot frissíteni kell. Ez növeli a karbantartási időt és bonyolultságot.
   - **Példa:** Ha egy ügyfél nevét változtatjuk meg, akkor minden olyan helyen, ahol az ügyfél neve tárolva van, manuálisan kell frissíteni, ami megnöveli a hibázás esélyét.
4. **Anomáliák (Anomalies):** A redundancia különböző **anomáliákat** okozhat, amikor adatokat beszúrunk, frissítünk vagy törlünk az adatbázisban.
   - **Beszúrási anomália (Insertion Anomaly):** Előfordulhat, hogy nem tudunk egy adatot beszúrni, mert szükség van egy másik, már létező adat jelenlétére.
     - **Példa:** Egy rendelés csak akkor létezhet, ha van ügyfél is hozzá. Ha egy új ügyfelet akarunk felvenni, de még nem adott le rendelést, az adatbázis lehet, hogy nem engedi bevinni az adatokat.
   - **Törlési anomália (Deletion Anomaly):** Adatok törlésekor fontos információkat is elveszíthetünk.
     - **Példa:** Ha egy ügyfél összes rendelését töröljük egy táblából, akkor az ügyfél adatai is eltűnhetnek, még akkor is, ha az ügyfélnek más fontos adatai lennének, amiket meg szeretnénk tartani.
   - **Módosítási anomália (Update Anomaly):** Ha egy adatot több helyen tárolunk, akkor minden helyen frissíteni kell, ami megnöveli az inkonzisztens adatok előfordulásának esélyét.
     - **Példa:** Ha egy ügyfél címét több táblában tároljuk, de csak az egyik táblában módosítjuk, akkor a cím különböző helyeken eltérő lesz.
5. **Teljesítménycsökkenés (Performance Issues):**
   - A redundancia gyakran lassítja az adatbázis teljesítményét, különösen a frissítési és karbantartási feladatok során. Ha több helyen kell adatokat módosítani, az több erőforrást igényel, ami lassítja az adatbázis válaszidejét.

20. Redundancia kezelése

A redundancia minimalizálása érdekében alkalmazzuk a **normalizálást**, amely segít megszüntetni az ismétlődő adatokat és az anomáliákat. A normalizálás különböző lépései (1NF, 2NF, 3NF, BCNF) segítenek az adatok megfelelő szervezésében, így hatékonyabb és megbízhatóbb adatbázisstruktúrát hozunk létre.

A normalizálás szerepe a redundancia csökkentésében:

- A **normalizált táblák** minimalizálják az ismétlődést, mivel az adatokat külön táblákba rendezzük, ahol az információk csak egyszer szerepelnek.
- **Kapcsolatok** létrehozása a táblák között (külső kulcsok segítségével) biztosítja, hogy az adatok csak egyszer kerüljenek tárolásra, és ne legyen szükség azok több helyen történő tárolására.

21. **Dekompozíció (Decomposition)** az adatbázisokban

A **dekompozíció** az az eljárás, amelynek során egy összetett táblát vagy relációt kisebb, egyszerűbb táblákra bontunk szét, hogy csökkentsük az adatbázisban lévő redundanciát és a belőle adódó anomáliákat. A cél az, hogy a különböző adatokat külön táblákba rendezzük, és a táblák között kapcsolatot hozzunk létre (például külső kulcsok segítségével), ezzel biztosítva az adatok integritását és pontosságát.

A dekompozíció szerepe a redundancia csökkentésében

A dekompozíció kulcsfontosságú eszköz az adatbázis normalizálása során, különösen a **második (2NF), harmadik (3NF)** és **Boyce-Codd normál forma (BCNF)** elérésében.

22. A **relációs adatmodell megszorításai** olyan szabályok, amelyek biztosítják az adatok helyességét, konzisztenciáját és integritását a relációs adatbázisokban. A legfontosabb megszorítások a következők:
    1. **Kulcs megszorítás (Key Constraints):**
        - Minden relációban (táblában) kell lennie egy **primer kulcsnak** (primary key), amely egyedi módon azonosítja a sorokat. Egy sor minden attribútuma (oszlopa) azonos kell legyen, ha két sor primer kulcsa megegyezik.
    2. **Idegen kulcs megszorítás (Foreign Key Constraints):**
        - Az idegen kulcs biztosítja, hogy egy táblában lévő érték egy másik táblában érvényes (létező) értékre hivatkozzon. Ezzel fenntartható a relációk közötti kapcsolatok integritása.
    3. **Domain megszorítások (Domain Constraints):**
        - Minden attribútumnak (oszlopnak) előre meghatározott adattípusa és értéktartománya van (pl. szám, dátum, szöveg). Csak az adott típushoz illő értékek tárolhatók az adott attribútumban.
    4. **Null érték megszorítás (Null Value Constraints):**
        - Meghatározza, hogy egy attribútum elfogadhat-e **null** (nem ismert vagy nem létező) értéket. Egyes oszlopok kötelezően kitöltendők lehetnek, ilyenkor nem fogadhatnak el null értéket.
    5. **Egyediség megszorítás (Unique Constraint):**
        - Biztosítja, hogy egy attribútum vagy attribútumcsoport értékei egyediek legyenek a reláció minden sorában, azaz ne legyenek ismétlődő értékek.
    6. **Referenciális integritás megszorítás (Referential Integrity Constraints):**
        - Az idegen kulcsokkal kapcsolatos szabályokat érvényesíti: biztosítja, hogy egy tábla egy adott sorára való hivatkozás csak akkor történhet meg, ha az a sor létezik.

Ezek a megszorítások alapvető szerepet játszanak az adatbázis integritásának és pontosságának fenntartásában.

23. **Normalizálás (Normalization):** Az adatok szervezése oly módon, hogy minimalizáljuk a redundanciát (ismétlődést) és javítsuk az adatok integritását. A normalizáció több lépésből (NF – normál formák) áll, amelyek mindegyike segít a hatékony adatstruktúra kialakításában.

Első normál forma (1NF)

- **Feltétel:** Minden oszlopnak atomikus (oszthatatlan) értékeket kell tartalmaznia, azaz nem lehetnek többértékű oszlopok (pl. listák).
- **Példa:** Tegyük fel, hogy van egy „Rendelések” tábla, ahol egy oszlopban tároljuk a megrendelt termékeket vesszővel elválasztva:
  - **Rendelések**: RendelésID, VevőNév, Termékek („Toll, Könyv”)
  - Az 1NF szerint a „Termékek” oszlopot külön sorokba kell bontani, így minden termék külön sorban szerepel.
  - **Rendelések (1NF)**:
    - 1, Péter, Toll
    - 1, Péter, Könyv

Második normál forma (2NF)

- **Feltétel:** A tábla legyen 1NF-ben, és minden nem kulcs attribútum teljes függésben legyen az elsődleges kulcstól (azaz ne függjön a kulcs részétől).
- **Példa:** A fenti példában tételezzük fel, hogy egy **„VevőID”** az elsődleges kulcs, de a vevő neve és az egyes rendelésekhez tartozó termékek között nincs közvetlen kapcsolat, így létre kell hozni egy külön táblát a termékekhez:
  - **Vevők**: VevőID, VevőNév
  - **Rendelések**: RendelésID, VevőID, Termék

Harmadik normál forma (3NF)

- **Feltétel:** A tábla legyen 2NF-ben, és az attribútumok között ne legyen tranzitív függőség, vagyis egy nem kulcs attribútum ne függjön egy másik nem kulcs attribútumtól.
- **Példa:** Tegyük fel, hogy a rendeléseknél a vevők címét is tároljuk a „Vevők” táblában:
  - **Vevők (nem normalizált):** VevőID, VevőNév, VevőCím
  - A vevő címe nem függ közvetlenül a rendeléstől, ezért külön táblába kell helyezni:
  - **Vevők (3NF):** VevőID, VevőNév
  - **Címek:** CímID, VevőID, Cím

Boyce-Codd normál forma (BCNF)

- **Feltétel:** Egy tábla BCNF-ben van, ha 3NF-ben van **és** minden determináns (az oszlop vagy oszlopok csoportja, amely más oszlopok értékeit egyértelműen meghatározza) szuperkulcs.

- Ez azt jelenti, hogy ha bármely oszlop vagy oszlopcsoport egy másik oszlopot meghatároz, akkor annak szuperkulcsnak kell lennie (egy tábla egyedi azonosítójának).
- A BCNF akkor szükséges, ha egy tábla többkomponensű kulcsokat tartalmaz, és ezek között nem teljes függőségek állnak fenn.

Példa BCNF-re:

Képzeljük el, hogy van egy **„Kurzusok”** tábla, amely tartalmazza, hogy egy tanár milyen tantárgyakat tanít egy bizonyos teremben. Tegyük fel, hogy minden tanár csak egy tárgyat taníthat egy szobában, de egy tárgyat több tanár is taníthat.

- **Kurzusok (nem BCNF):**
  - TanárNév, Szoba, Tantárgy
  - Az elsődleges kulcs: (TanárNév, Szoba)

Ebben az esetben a **Szoba** és a **Tantárgy** között egy kapcsolat van: egy tárgy csak egy adott teremben tanítható. Ezért a **Szoba** meghatározza a **Tantárgy** attribútumot. Ez azonban nem teljesíti a BCNF feltételt, mert **Szoba** nem szuperkulcs.

A BCNF normalizálás lépése:

9. Szedjük szét a táblát két részre:
   - **Szoba_Tantárgy** (Szoba, Tantárgy): Ebben a táblában minden tantárgyhoz hozzárendelünk egy szobát, és a **Szoba** lesz az elsődleges kulcs.
   - **Tanár_Szoba** (TanárNév, Szoba): Ebben a táblában a tanár nevét és a szobát tároljuk, ahol tanít, és a **TanárNév** az elsődleges kulcs.

Ezáltal a táblák BCNF-ben vannak, mert minden determináns (Szoba a **Szoba_Tantárgy** táblában, TanárNév a **Tanár_Szoba** táblában) szuperkulcs.

Összefoglalva a normál formákat:

10. **1NF:** Minden mező atomi értékű.
11. **2NF:** 1NF-ben van, és nincs részleges függőség az elsődleges kulcsra.
12. **3NF:** 2NF-ben van, és nincs tranzitív függőség a nem kulcs attribútumok között.
13. **BCNF:** 3NF-ben van, és minden determináns szuperkulcs.

A BCNF alkalmazásával biztosítjuk, hogy ne legyenek olyan indirekt kapcsolatok, amelyek problémát okozhatnak a táblák közötti összefüggések kezelésében.