**8. ADATBÁZISOK TERVEZÉSE:** Hierarchikus, Relációs és NoSQL adatmodellek előnyei/hátrányai. Relációs adatmodell alapelvei, relációs tervezés folyamata, a normalizáció célkitűzései, 1.-5. normálformák. Relációs adatbázis tervezési alapmotívumok. Adattárházak tervezése: csillag-struktúra, hópehely-struktúra, multidimenzionális modell, On-line Analytical Processing.

**Adatmodellek típusai:**

Három adatmodell terjedt el: hierarchikus, relációs, és az NoSQL.

A **relációs** adatmodell szerkezetét gráffal adjuk meg. A gráfban a csúcspontok az egyedek, az élek pedig a kapcsolatok. Az egyedeket tulajdonságaikkal írjuk le, a kapcsolatokat mutatók segítségével adjuk meg.

A **hierarchikus** adatmodell szerkezetét is gráffal adjuk meg, de itt a gráf egy fa. Az adatok alá-, és fölérendeltségi viszonyban állnak.

A **NoSQL** adatmodell olyan adatstruktúrákat és adatkezelési módszereket jelent, amelyek eltérnek a hagyományos relációs adatbázisoktól. Ezek általában skálázhatóbbak és rugalmasabbak, és lehetővé teszik strukturálatlan vagy félstrukturált adatok tárolását és kezelését is. A NoSQL adatbázisok a nagy adatmennyiségek tárolására és gyors feldolgozására optimalizáltak, és különböző típusú adatok tárolására alkalmasak, például dokumentumok, grafikonok vagy kulcs-érték párok formájában.

**Relációs adatbázis struktúra**

A relációs modellben az **adatokat táblázatokban ábrázoljuk és a közöttük lévő különböző kapcsolatokat relációs műveletek** segítségével hozzuk létre. A táblázatok az adatok szerkezetének meghatározását, az adatok leírását teszik lehetővé a modell keretében, míg az adatkezelést a relációs műveletekkel valósítjuk meg.

Ezen a két alapvető feladaton kívül a relációs modell lehetőséget nyújt az adatok integritásának biztosítására, valamint a modell alkalmazásának hatékonyságát növelő szerkezetek kialakítására.

A relációs adatbázis struktúra a legelterjedtebb struktúra az adatbázis-kezelő rendszerek közül.

**Relációs adatbázis tervezési alapmotívumok:**

**Relációs adatstruktúra:** A relációs adatmodell struktúra részét meghatározó szerkezeti elemek rendszere, legfontosabb alkotó eleme a reláció.

**Előfordulás:** Adatrekord (egyed). A modellben elvárjuk, hogy minden sor minden komponense elemi legyen, és minden oszlophoz tartozzon egy értéktartomány (azaz egy elemi típus), mely meghatározza az oszlopban szereplő értékek típusát. Egy reláció nem állandó, léte során az gyakran változik, új sorok kerülnek bele, sorokat veszünk ki belőle, meglévő sorokat változtatunk meg, stb. Adott reláció sorainak előfordulását előfordulásnak nevezzük.

A relációs modellben az adatok egyszerűen reprezentálhatók: kétdimenziós táblázatban, ún. relációkban. Sorok → rekordok, oszlopok nevei → attribútumok.

**Rekord:** Adatbázis struktúra elem, mely logikailag összetartozó, és egységként kezelhető elemi adatértékek együttesét jelöli.

**Attribútum:** Az egyedeket tulajdonságokkal (attribútumokkal) írjuk le. A tulajdonság az egyed jellemzője, amely megadja, meghatározza az egyed egy részletét.

**Mező:** Az adatbázis struktúra azon egysége amelyből a rekordok felépülnek, a mező rendszerint legkisebb DB(DataBase) struktúra egység. Elemi.

**Reláció:** Az azonos szerkezetű rekord előfordulások névvel ellátott halmaza, tárolási egység a relációs adatbázisban. Adathalmaz (vagyis az információ).

**Kapcsolatok:** A kapcsolatok asszociatív jellegűek. Értékük egyszerűen változtatható, a struktúra, a kapcsolatrendszer, egyszerűen, gyorsan áttekinthető.

**PK (Primary Key):** a jelölt kulcsok közül az azonosításra kiválasztott mezőcsoport, egyedi(unique) és nem üres(not empty). A kapcsolódó relációk erre a mezőcsoportra fognak hivatkozni, azaz ennek értékét tartalmazzák a megfelelő idegen kulcsban(foreign key).

**FK( Foreign Key)**: Olyan mezőcsoport a relációs sémákban melynek célja egy megadott másik reláció valamely rekord előfordulásának a kijelölése.

**Domain:** Értékkészlet (attribútum-tartomány). Az adatbázisban tárolt adatok lehetséges értékei vagy értéktartományai az adott attribútumhoz vagy mezőhöz kapcsolódóan.

**Adatintegritás:** Az adatintegritás fontos szempontja annak, hogy az adatok megbízhatóak és konzisztensek legyenek. (kulcsok, korlátozások és triggerek segítségével)

**Effektivitás és hatékonyság:** A rendelkezésre álló erőforrások hatékony kihasználására kell törekednie. (az adatok tárolásának és lekérdezésének hatékonysága és a rendszer teljesítménye)

**Funkcionális és nem-funkcionális követelmények:**

- Funkcionális követelmények azok az elvárások, amelyek az alkalmazás funkcionalitására vonatkoznak

- Nem-funkcionális követelmények azok, amelyek az alkalmazás teljesítményére, biztonságára és használhatóságára vonatkoznak.

**Skálázhatóság és rugalmasság:** Könnyen alkalmazkodjon az esetleges változásokhoz és bővítésekhez és skálázhatónak kell lennie, hogy képes legyen kezelni az adatmennyiség növekedését.

**A relációs struktúra előnyei és hátrányai:**

**Előnyei:**

* **Ad-hoc:** az ad-hoc lekérdezések támogatása; (ad-hoc=erre az alkalomra. Az "ad-hoc" jelző itt azt jelenti, hogy ezek a lekérdezések nem előre definiáltak vagy rögzítettek, hanem a felhasználó igényei és kívánságai szerint alakulnak.)
* **Egyszerű:** az adatbázis tervezése, módosítása, karbantartása nagyon egyszerű;
* **Felhasználói szemlélet:** a táblázatos ábrázolásmód következtében az áltagos felhasználói szemlélethez ez áll a legközelebb;
* **Adatfüggetlenség:** ebben a modellben valósítható meg legjobban, legegyszerűbben és legnagyobb mértékben a fizikai és logikai adatfüggetlenség.

**Hátrányai:**

* **Teljesítményproblémák:** Az összetett lekérdezések és az adatok nagy mennyisége esetén a relációs adatbázisok teljesítménye csökkenhet. Ez különösen akkor jelentkezik, ha nem megfelelően optimalizált lekérdezéseket használnak vagy ha az adatbázis nem megfelelően indexelve van.
* **Korlátozott skálázhatóság:** Az adatbázis mérete és a felhasználók száma növekedésével a skálázhatóság korlátozott lehet. (Nagyobb adatmennyiségek esetén, más típusú adatbázisok (pl. NoSQL) jobban alkalmazkodnak az adatbázis méretének növekedéséhez.)
* **Összetett adatstruktúrák kezelése:** Bizonyos esetekben más adatmodell-technológiák (pl. dokumentum-orientált adatbázisok) hatékonyabbak lehetnek az ilyen típusú adatok tárolásában és lekérdezésében.
* **Kerülési kényszer (impedance mismatch):** "kerülési kényszer" (relációs adatbázisok és az objektumorientált programozás) - amikor az objektumok és az adatbázis közötti adatmodellek nem illeszkednek tökéletesen egymáshoz. Ez problémákat okozhat az adatok lekérdezésében és manipulálásában.
* **Komplex adatmodellezés:** Néha a valós világban lévő kapcsolatok és entitások összetettsége miatt a relációs adatbázisokban való megfelelő adatmodellezés bonyolulttá válha.

**Relációs adatmodell alapelvei:**

* Ahhoz, hogy ezek a táblázatok valóban relációs adatmodellt alkossanak, az egyes táblázatokban a következő feltételeknek kell teljesülnie:
* Minden relációnak neve van, mely egyértelműen azonosítja az adatmodellben.
* Adott reláció minden sorában azonos számú oszlop van. Az oszlopok számát nevezzük a reláció fokának.
* Minden oszlopnak az adott reláción belül egyértelmű neve van, amellyel azonosítható.
* Minden oszlop csak meghatározott értékeket vehet fel és azonos típusú elemi adatot tartalmazhat.
* Bármely sorban bármelyik oszlop csak egyetlen értéket vehet fel a megengedett értékek közül.
* Az oszlopok sorrendje nem befolyásolja az adatmodellt.
* Mivel a reláció, a táblázat egyed előfordulásai az egyes sorok, a táblázatnak nem lehet két teljesen azonos sora, azaz két olyan bejegyzése, melyben minden oszlopérték megegyezik.
* A sorok számát nevezzük a reláció kiterjedésének.

**Az adatbázis-tervezés lépései:** (általánosságban)

* a feldolgozandó információ elemzése,
* az információk közti kapcsolatok meghatározása,
* az eredmény ábrázolása (E/K diagram),
* adatbázisterv készítése (transzformációs lépés),
* adatbázisterv finomítása (összevonások),
* megszorítások modellezése, függőségek meghatározása,
* optimális adatbázisterv készítése (dekomponálás, normalizálás),
* az adatbázisterv megvalósítása SQL-ben (create table…, create view …, stb.).

**Relációs tervezés folyamata:**

**1. Adatgyűjtés:** Meg kell határozni, hogy milyen adatokat kell tárolni, és milyen kapcsolatok vannak ezek között.

**2. Entitások és attribútumok azonosítása:** Az adatok alapvető egységeit, az entitásokat kell azonosítani, valamint azok tulajdonságait, az attribútumokat. Az entitások lehetnek például emberek, tárgyak, események vagy fogalmak.

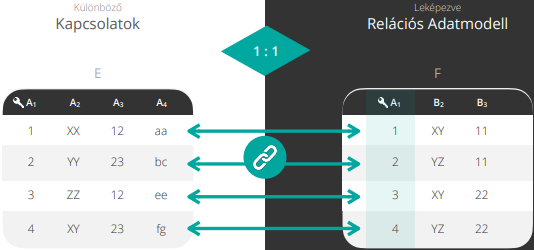
**3. Entitások közötti kapcsolatok meghatározása:** Meg kell határozni, hogy az entitások között milyen kapcsolatok állnak fenn. Ezek lehetnek egy-egy (1:1), egy-több (1:N) vagy sok-több (N:M) kapcsolatok.

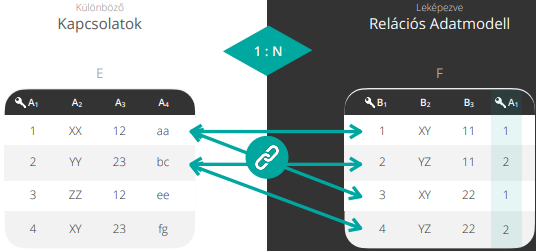
**4. Kulcsok azonosítása:** Az entitásokat azonosítani kell egyedi azonosítókkal, amelyek segítségével könnyen hivatkozhatunk rájuk. Ezek lehetnek például elsődleges kulcsok (primary keys) vagy másodlagos kulcsok (secondary keys).

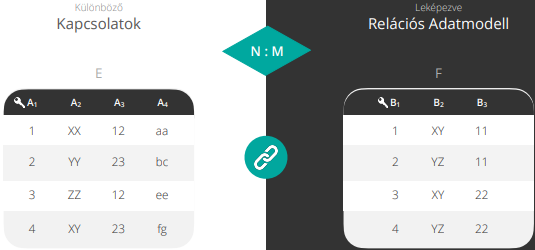
**5. Normalizáció:** Az adatok normalizálása a redundancia és a függőségek csökkentése érdekében történik.

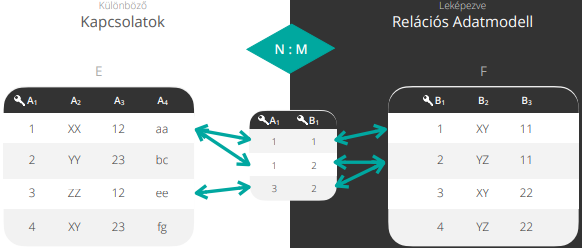
**6. Relációs séma kialakítása:** Az előző lépések alapján megtervezett entitások, attribútumok és kapcsolatok alapján létrehozunk egy relációs sémát, amely az adatok tárolásának és kezelésének strukturált formáját jelenti.

**Példák kapcsolatok szerint:**









**Hierarchikus adatmodell**

A modellben a rekordok közötti kapcsolatok szülő-gyerek elemekből felépített fa struktúrájú kapcsolatrendszert, hierarchiát alkotnak, a mezők csak elemi értékeket vehetnek fel és a rekordok elérése a rögzített kapcsolatok mentén haladva történik.

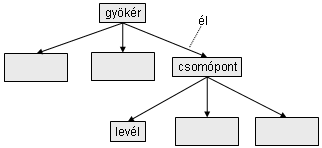
**Hierarchikus adatszerkezetek:** Szerkezetét speciális gráffal, a fával adjuk meg. A struktúraelemek hierarchikusan egymás alá vannak rendelve. A kapcsolatok jellege egy-sok: minden csomópont csak egy helyen látható, egy csomópontból viszont sok csomópont látható: olyan problémák esetén alkalmazható, melyekre jellemző a tulajdonos viszony ill. a lebontás. Pl.: fa, hierarchikus állomány.

Fa: hierarchikus adatszerkezet, mely véges számú csomópontból áll, és igazak rá a következők:

* két csomópont között a kapcsolat egyirányú, azaz egyik a kezdőpont másik a égpont.
* van a fának egy kitüntetett csomópontja, mely nem lehet végpont. Ez a fa gyökere.
* az összes többi csomópont pontosan egyszer végpont.

A fának van azonban egy rekurzív definíciója is.

* A fa üres, vagyA fának van egy kitüntetett csomópontja, ez a gyökér.
* A gyökérhez 0 vagy több diszjunkt fa kapcsolódik. Ezek a gyökérhez tartozó részfák.



<https://centroszet.hu/tananyag/adatbazis/21_hierarchikus_modell.html>

**A hierarchikus adatmodell elemei:**

* **Mező:** Az Adatbázis struktúra azon egysége, melyből a rekordok felépülnek, a mező rendszerint a legkisebb DB (Database) struktúra egység.
* **Rekord:** Adatbázis struktúra elem, mely logikailag összetartozó, és egységként kezelhető elemi adatérték együttesét jelöli.
* **PCR:** (Parent Child Relationship) csak 1:N kapcsolat ábrázolására alkalmas, nincs mögötte adatérték. A gyermek tárolja el, hogy ki a szülője.
* **Hierarchia:** Több rekordtípusnak a PCR elemeken keresztül megvalósult kapcsolatrendszerét tartalmazza.
* **Adatbázis:** Több hierarchikus sémát foglal össze.

**A hierachikus struktúra előnyei és hátrányai:**

**Előnyei:**

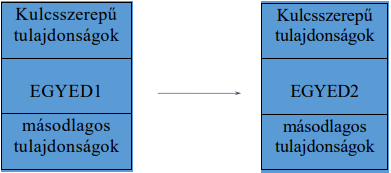
* **Linealizálható,** szalagos tárolásra alkalmas: az egyszerűség nemcsak a modellezésben jelentkezik, hanem a fizikai, belső szinten is. A sémához tartozó előfordulási fákat ugyanis egyszerű eszközökkel és egyszerű struktúrákban is könnyen tárolhatjuk. A hierarchikus előfordulási fa a szekvenciális szervezésben is hatékonyan letárolható, az előfordulási fa ugyanis linearizálható.
* **Kapcsolat nyilvántartó** – Lineális (pozíció alapú): A visszaolvasásnál az eredeti előfordulási fa egyértelmű helyreállítását úgy érjük el, hogy minden rekordtípushoz rendelünk egy típusjelzést és ezt is letároljuk az állományban. Ezáltal **követni tudjuk, hogy a hierarchia mely szintjén járunk,** és meg tudjuk határozni a "testvéreket" is.
* **Egyszerű faalgoritmusok:** A fák bejárására több különböző módszer létezik, és mindegyik linearizálja a fát, minden csomóponthoz egy bejárási sorszámot rendelve, ami megfeleltethető a csomópont szalagon elfoglalt pozíciójának. Például bejárható az elsőként baloldal utána jobboldal elven, amikor egy adott részfánál előbb a gyökér, majd a gyermekek részfái következnek balról jobbra haladva. Ennél a bejárásnál egy csomópont leszármazottai mind az elem után helyezkednek el.
* **Lekérdezés – pointeres:** A visszafelé keresés megkönnyítésére bizonyos módosításokat, bővítéseket kell végrehajtani az alapmodellen. Ennek egyik változata lehet, hogy pointereketépítünk be az egyed előfordulásokba, melyek a szülő felé mutatnak.

**Hátrányai:**

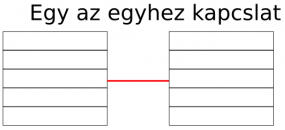
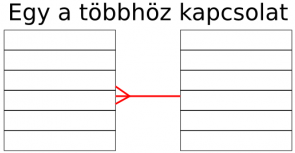
* **Tárolási mód:** Ugyan a linearizálás, a szekvenciális tárolási mód egyszerű, és hatékony a teljes hierarchia visszaállításánál, de ez a hatékonyság eltűnik, ha a műveleteknek csak egy részfát kell visszaadniuk, illetve ha a fa olyan nagy, hogy **nem fér be a memóriába.**
* **Csak egyféle keresési irányt támogat:** Csak egyféle keresési irányt támogat, a fentről-lefelé irányt. Egy adott elem őseinek, elérési útvonalának a meghatározásához az egész fát át kell olvasni, ha csak az adott elem pozíciója ismert.
* **Modellezési nehézségek:** Az előbb említett hatékonysági problémák mellett más modellezési nehézségekis fellépnek a hierarchikus modell használatakor, mint pl az: N:M és a PCR kapcsolatok tárolása; többszörös kapcsolatok.

**NoSQL adatmodell**

A nem relációs, gyakran „nem csak SQL” néven említett nagy teljesítményű adatbázisok az erősen strukturált adatok kezelésére is alkalmasak – nem korlátozódnak rögzített adatmodellekre, mint a relációs (SQL-) adatbázisok.



Az SQL adatbázis egy strukturált adatbázisrendszer, amely táblákat használ az adatok tárolására. Az adatokat a táblákban sorok és oszlopok formájában tárolja. Az SQL (Structured Query Language) nyelvet használva lehet lekérdezéseket futtatni az adatokra, illetve módosításokat végezni rajtuk, mint például adatok beszúrása, módosítása vagy törlése.



[SQL adatmodell ábrázolása](https://szit.hu/doku.php?id=oktatas:adatbazis-kezeles:tananyag)

**A NoSQL-adatbázisok négy leggyakoribb típusa a következő:**

**Dokumentum:** A dokumentum-adatbázisok a kulcs-érték típusú adatbázis elvét terjesztik ki azzal, hogy teljes dokumentumokat szerveznek gyűjteménynek nevezett csoportokba. Támogatják a beágyazott kulcs-érték párokat, és adott dokumentum bármely attribútuma alapján végzett lekérdezést lehetővé tesznek.

**Kulcs-érték:** A kulcs-érték típus kivonattáblában tárolja a kulcs-érték párokat. A kulcs-érték típus akkor a leghasznosabb, ha egy kulcs ismert, de a hozzá társított érték ismeretlen.

**Oszlopalapú:** Az oszlopalapú, széles oszlopú vagy oszlopcsalád-alapú adatbázisok hatékonyan tárolják az adatokat és kérdezik le a ritka adatok sorait. Előnyösek az adatbázis megadott oszlopain végzett lekérdezések esetén.

**Gráf:** A gráf típusú adatbázisok csomópontokból és élekből felépülő modellt használnak az egymással összefüggő adatok – például egy közösségi hálózat tagjai közötti kapcsolatok – leképezésére, és az összetett kapcsolatok egyszerűbb tárolását és követését teszik lehetővé.

**A NoSQL struktúra előnyei és hátrányai:**

**Előnyei:**

A NoSQL adatbázisok jól illeszkednek a legtöbb modern alkalmazáshoz, például mobil-, játék-, webalkalmazásokhoz, amelyek rugalmas, méretezhető adatbázisokat igényelnek nagy teljesítménnyel és funkcionalitással.

**Rugalmasság:** Rugalmas sémákat kínálnak, amelyek gyorsabb fejlesztést tesznek lehetővé. Ezért alkalmasak félig strukturált és strukturálatlan adatokra.

**Sémamentesség:** A relációs adatbázisokkal ellenétben nincs előre definiált fix séma. Kevesebb erőforrás megy el adatbázis modellezésre, tervezésre.

**Skálázhatóság:** Támogatják a gyors, hatékony skálázhatóságot. A megfelelő terhelésmegosztás érdekében újabb egységeket lehet könnyedén a rendszerre csatlakoztatni.

**Nagy teljesítmény:** Meghatározott adatmodellekhez és hozzáférési mintákhoz vannak optimalizálva, hogy nagyobb teljesítményt érjenek el, mint a relációs adatbázisok.

**Széles funkcionalitás:** Olyan API-kat és adattípusokat kínálnak, amelyeket kifejezetten a megfelelő adatmodellekhez terveztek.

**Hátrányai:**

**Struktúra:** A relációs adatbázisokban az SQL nyelv segítségével jól strukturált, matematikai bizonyosságú lekérdezések használhatóak (relációs kalkulus, relációs algebra). Ez a lehetőség a NoSQL adatbázisoknál nem adott.

**Kisebb a konzisztencia:** a legtöbb nem támogatja teljes mértékben az ACID (Atomicity, Consistency, Isolation, Durability) típusú tranzakciókat. Az ACID tulajdonságok fontosak a tranzakciók megbízhatósága és adatintegritása szempontjából.

(Adatbázisok esetén az ACID az Atomicity (atomiság), Consistency (konzisztencia), Isolation (izoláció), és Durability (tartósság) rövidítése. Ezek az adatbázis-kezelő rendszer tranzakciófeldolgozó képességeinek alapelemei. Ezek nélkül az adatbázis integritása nem garantálható. <https://hu.wikipedia.org/wiki/ACID>)

**Hiányzó uniformitás:** Ezek a rendszerek sokféle architektúrát és megközelítést ölelnek fel, nincs általánosan elfogadott standard az adatmodellre, az adatlekérdezésre vagy a tranzakciókezelésre vonatkozóan. Akalmazkodni kell az adott NoSQL adatbázis sajátosságaihoz és nyelvéhez.

**Relációk elutasítása:** a NoSQL adatbázisok tranzakciói képesek lehetnek arra, hogy elutasítsák a relációkat bizonyos esetekben. Ennek hátrányai: a tranzakció visszavonásának vagy elutasításának következményeivel járó adatinkonzisztencia és az adatbázis integritásának veszélyeztetése.

(Amikor egy tranzakció elutasít egy relációt, az azt jelenti, hogy az adatbázis rendszer visszavonja vagy nem fogadja el a tranzakció által előírt változtatásokat az adatbázisban. Ez általában akkor fordulhat elő, ha a tranzakció megsérti az adatbázis által definiált integritási megszorításokat vagy üzleti logikát. Például, ha egy tranzakció egy olyan adatot próbál beilleszteni az adatbázisba, amely egy másik táblához hivatkozik, de az adott referencia nem létezik, akkor a tranzakció elutasítható lehet.)

**Normalizálás**

Az adatbázis tervezése során a legfontosabb feladat a logikai adatmodell kialakítása, mely azt jelenti, hogy: az adatok laza halmazából egy jól felépített, átlátható logikai adatstruktúrát hozunk létre. Ezt a folyamatot nevezzük NORMALIZÁLÁS-nak.

Normalizálás (normálforma dekompozíció): Az eljárás, amelyben a kedvezőtlen normálformájú egyedet lebontjuk több kívánt normálformájú egyedre.

**Célja:** hogy az adathalmaz numerikus oszlopainak értékeit úgy módosítsa, hogy közös skálát használjon, anélkül, hogy torzítaná az értéktartományok közötti különbségeket vagy elveszítené az információkat, vagyis a redundancia csökkentése. Eszerint a redundancia csökkentése az anomáliák (rendellenességek az adatbázison belül) elkerülése érdekében normalizálunk, de úgy, hogy a funkcionális függőségeket ne veszítsük el.

**Redundancia fogalma:**

A logikai adatbázis tervezés egyik fő célja a redundanciák megszüntetése. Redundanciáról akkor beszélünk, ha valamely tényt vagy a többi adatból levezethető mennyiséget ismételten (többszörösen) tároljuk az adatbázisban.

A redundancia, a szükségtelen tároló terület lefoglalása mellett, komplikált adatbázis frissítési és karbantartási műveletekhez vezet, melyek könnyen az adatbázis inkonzisztenciáját okozhatják. Egy adatbázis akkor inkonzisztens, ha egymásnak ellentmondó tényeket tartalmaz.

A normalizáció segítségével az adatokat kisebb egységekre bontjuk, és ezáltal minimalizáljuk a redundanciát. Ez nemcsak hatékonyabb adatbázis-struktúrát eredményez, hanem egyszerűbbé is teszi az adatok kezelését és karbantartását.

**Normálformák:**

Az adatbázisok belső szerkezetét jellemzik a normálformák. Ha az adatbázis eleget tesz egy biozonyos feltételnek, akkor egy adott normálformában van.

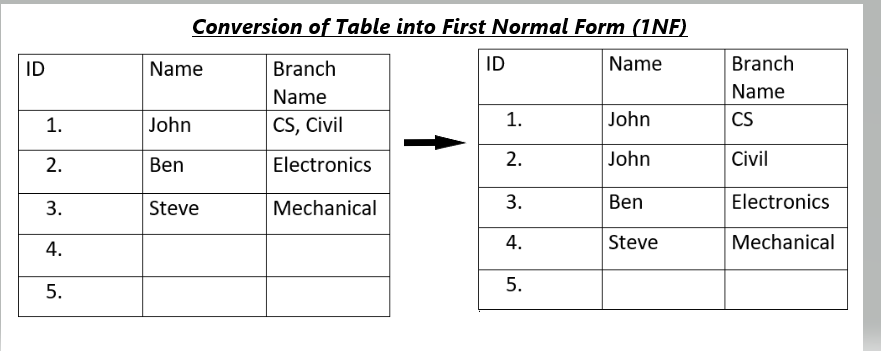
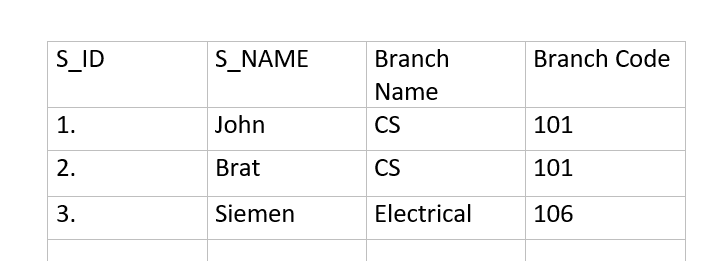
* 1NF: Egy R relációról azt mondjuk, hogy első normálformában van, ha minden sorában pontosan egy attribútum érték áll.
* 2NF: ha első normálformában van és ha minden másodlagos attribútum teljesen függ a kulcstól.
* 3NF: ha második normálformában van és egyetlen másodlagos attribútum sem függ tranzitíven a kulcstól.

**Normálformára hozás:**

**1NF-re:** minden “többszörös” attribútumértékő sort annyi sorban írunk fel, ahányszoros az attribútumérték a sorban, vagy a kulcshoz egy külső kulcsú relációt készítünk és ebből a többszörös attribútumértékű oszlop soraiból annyi sort hozunk létre, ahányszoros az attribútumérték van ebben a sorban.

Lépései:

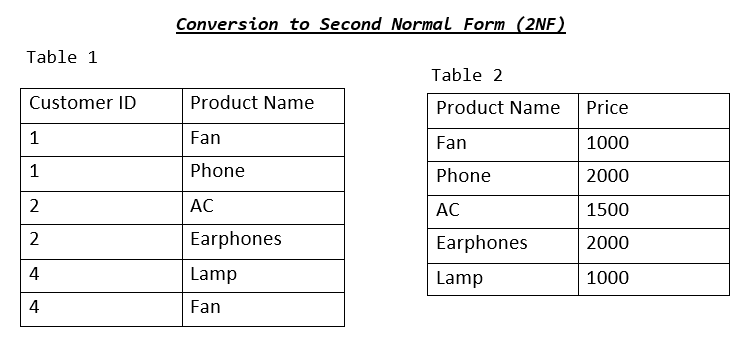
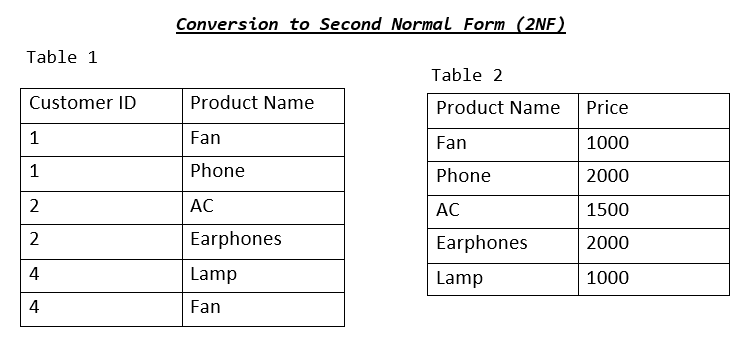
* Az egyes táblákban az ismétlődő csoportok megszünttetése
* Az egymáshoz kapcsolódó adatok minden halmazához külön tábla létrehozása
* Az egymáshoz kapcsolódó adatok minden halmazát elsődleges kulccsal azonosítani



**2NF-re:** a kulcsnak azon attribútumhalmazaiból, melyek maguk is meghatározzák (funkcionálisan) a másodlagos attribútumokat, valamint ezen másodlagos attribútumokból önálló relációt hozunk létre. Így több relációra bontjuk az eredeti relációt.

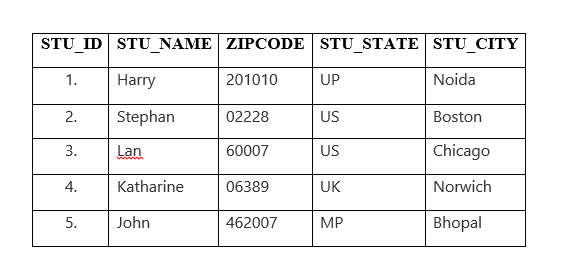
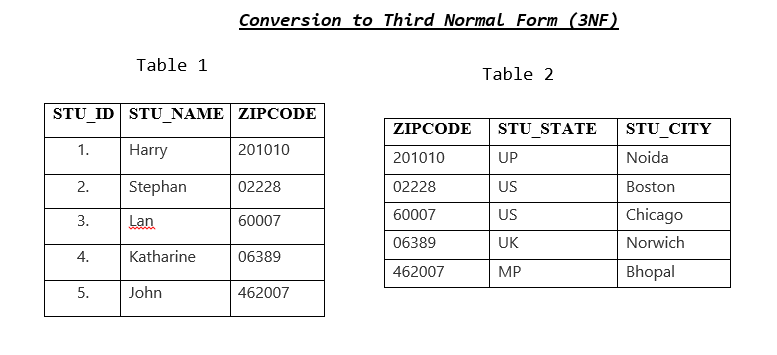
Lépései:

* Külön táblák létrehozása a több rekordra vonatkozó értékcsoportokhoz
* A táblákat külső kulccsal való összekapcsolása



**3NF-re:** megszüntetjük a tranzitív függıségeket, mégpedig úgy, hogy a tranzitív függőségben részt vevő attribútumhalmazok felhasználásával új relációkat készítünk.

Ehhez el kell távolítani azokat a mezőket, amelyek nem függenek a kulcstól.

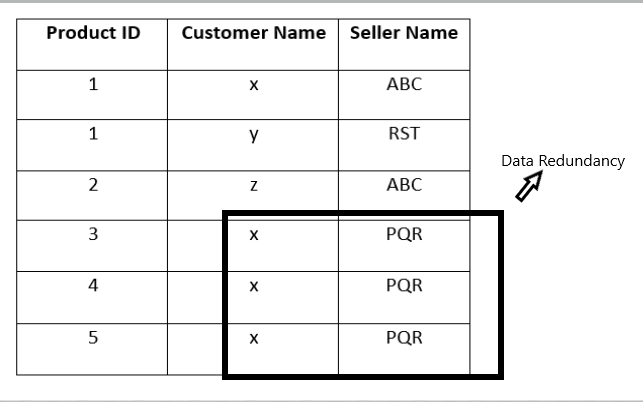
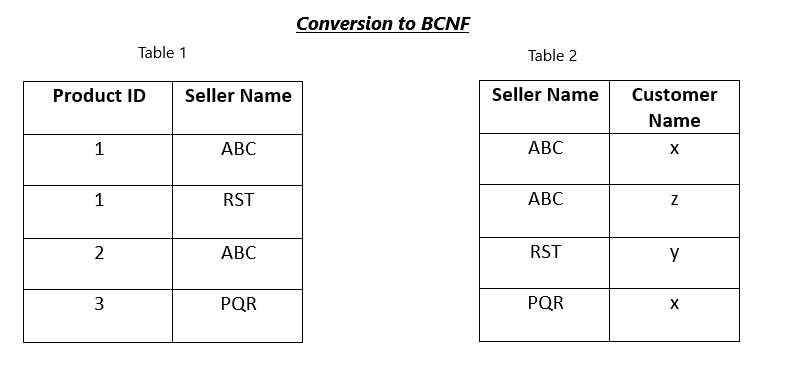


**Boyce-Codd normálforma (BCNF):**

A Boyce-Codd normálforma (BCNF) a relációs adatbázisok tervezésének egy speciális formája. Ezt az adatbázis-tervezési szabályt arra tervezték, hogy megszüntesse azokat a lehetséges anómalikus helyzeteket, amelyek a második normálforma (2NF) vagy a harmadik normálforma (3NF) esetén előfordulhatnak.

**Definíció**: Egy relációs tábla akkor és csak akkor van Boyce-Codd normálformában, ha minden **nem-triviális funkcionális függőség** a kulcs attribútumok valamelyikéből ered.

Ez azt jelenti, hogy minden nem-kulcs attribútum **közvetlenül és teljesen funkcionálisan függ** a relációban található valamelyik kulcs attribútum kombinációjától. Amennyiben egy relációs tábla nem felel meg ennek a feltételnek, akkor nem tartozik a Boyce-Codd normálformába, és általában szét kell választani annak érdekében, hogy kielégítse ezt a követelményt.



**Adattárházak**

**Az adatfeldolgozás forrásai:**

* Tranzakciós adatbázis (OLTP – On-Line Transaction Processing)
* Operatív adattár (ODS – Operational Data Store)
* Adattárház (**OLAP** – On-Line Analytical Processing)

**Adattárház definiálása -** Data warehouse:

Nagy méretű adatbázis, amely kifejezetten a vállalat vagy szervezet üzleti adatjainak tárolására és elemzésére szolgál. Általában összegyűjtik és integrálják az eltérő forrásokból származó adatokat, például tranzakciós rendszerekből, CRM (ügyfélkapcsolat-kezelő) rendszerekből, pénzügyi rendszerekből stb. Ezek az adatok rendszerint tisztítottak és átalakítottak, hogy összhangban legyenek az adattárházban tárolt adatmodellekkel és elemzési célokkal.

**Az alkalmazás célja (OLAP):**

* Bonyolult elemzések elvégzése (idősorozat-elemzés, extrapoláció)
* Összefüggések felismerése (korreláció-elemzés)
* Lekérdezés, listázás

**Jellegzetességek:**

* Nincs tranzakció
* Tartalmazza az archivált adatokat (a változások időbélyeggel ellátva)
* Nagy adatmennyiség
* A tervezés alapja: az elemzés szükséges adatai hatékonyan elérhetők legyenek

**A tervezés lépései:**

A tervezés célja: Hatékony nyilvántartás a tulajdonságok közötti új összefüggések felismerése érdekében

1. Az elemzési **célok megfogalmazása** (a Stratégia)
2. A nyilvántartandó **tulajdonságok** és azok **forrásának kijelölése** (Attribútumok és Adatforrások)
3. Az **attribútumok szétválasztása** tény- és dimenzióadatokra
4. **Tárolási szerkezet létrehozása** a tényadatok és a dimenzióadatok szerint

* **Csillag-struktúra**
* A nevét arról kapta, hogy a dimenziótáblák mintegy "csillagként" veszik körül a központi ténytáblát.
* Az elemzési adatok (és azonosítók) egy központi, ún. *ténytáblába* kerülnek (ezen történnek az elemzések)
* Az értelmezési adatok a *dimenziótáblákba* kerülnek, melyek azonosítók segítségével kapcsolódnak a ténytáblához

Ténytábla (fact table): Ez a központi tábla tartalmazza azokat az adatokat, amelyeket elemzünk vagy riportokat készítünk róluk. Általában tartalmazza a mérőszámokat (pl. értékesítési összegek, mennyiségek stb.) és a tranzakciókat vagy eseményeket (pl. vásárlások, tranzakciók időpontja stb.). Minden rekord egy egyedi tranzakciót vagy eseményt képvisel.

Dimenziótáblák (dimension tables): Ezek a táblák tartalmazzák azokat az értelmező adatokat, amelyekkel összekapcsoljuk a ténytábla rekordjait. Például ha az értékesítési adatokat elemzzük, akkor a dimenziótáblák tartalmazhatják a termékek, vásárlók, helyszínek stb. adatait. Ezek azonosítók (kulcsok) segítenek kapcsolódni a ténytáblához.

Az összekapcsolás általában egy-egy dimenziótábla és a ténytábla közötti relációk segítségével történik. Ezek a kapcsolatok a dimenziótábla kulcsait (azonosítóit) és a ténytábla sorokat egyesítik.

Galaxis-struktúra: Csillag-struktúra együttes, melynél ugyanazon dimenziótáblákat több ténytábla is használja

* **Hópehely-struktúra (kevéssé használt)**
* Olyan csillag-struktúra, melynél a dimenziótáblák normalizáltak (többszintűek)

Hasonlít a csillag-struktúrához, de eltér abban, hogy a dimenziótáblák normalizáltak vagy többszintűek. Ez azt jelenti, hogy a dimenziótáblák nem közvetlenül kapcsolódnak a ténytáblához, hanem egymáshoz kapcsolódnak, létrehozva egy hierarchikus szerkezetet.

Előnyei: jobb adatintegritás és az adatok redundanciájának csökkentése. Mivel a dimenziótáblák normalizáltak, az adatokat általában hatékonyabban lehet tárolni és kezelni, mivel azok nem ismétlődnek a táblák között.

Azonban használata bonyolultabb lehet a lekérdezések és az adatelemzés szempontjából, mivel a dimenziótáblák többszintű hierarchiát alkotnak, és az adatok elérése több táblán keresztül történik. Emellett a normalizálás növelheti a lekérdezések teljesítményét, mivel több táblát kell összekapcsolni az adatok lekérdezésekor.

**Adattárház tervezés eredményei:** A struktúra-létrehozás eredménye - több tábla keletkezik, a ténytábla és a kapcsolódó dimenzió-táblák között idegen kulcs kapcsolat van

**Előnyei:**

* Célorientált, de hatékony
* A kitűzött elemzési feladatok hatékony elvégzése nagy adatmennyiségen
* Az elemzési adatok hatékony megjelenítése, listázása

**Hátrányai:**

* A hatalmas adatmennyiség miatt nagy erőforrásigény
* Csak a megtervezett elemzési feladatokra hatékony
* A tervezettől nagyon eltérő elemzések egyáltalán nem végezhetők el

**Továbbfejlesztés elemzési célokra:**

* Relációs Adattárház (ROLAP)
* Ez az eddigiekben bemutatott OLAP, melynek szerkezete relációs.
* Előnye, hogy az SQL eszközökkel használható
* Hátránya, hogy az elemzéshez szükséges adatkigyűjtések táblafeldolgozásból történnek.

* Multidimenziós Adattárház (**MOLAP**)
* Előfeldolgozott (kigyűjtött, csoportosított, ú.n. aggregált) adatokat tárol
* A tárolás n-dimenziós kockákban törénik, ahol a dimenziók a ROLAP-nál mondotthoz hasonlóak. (például EladásSzám<Termék, Vásárló, Idő>))
* Előnye: Igen egyszerűen elemezhető
* Hátránya: Még nagyobb és redundáns tárhely (nyilván nem vásárol minden terméket, minden vásárló, minden nap)