

Adatbázisrendszerek

12. előadás: NoSQL adatbázisok

Alapfogalmak, NoSQL rendszerek és adatmodellek,
NoSQL adatmodellezés

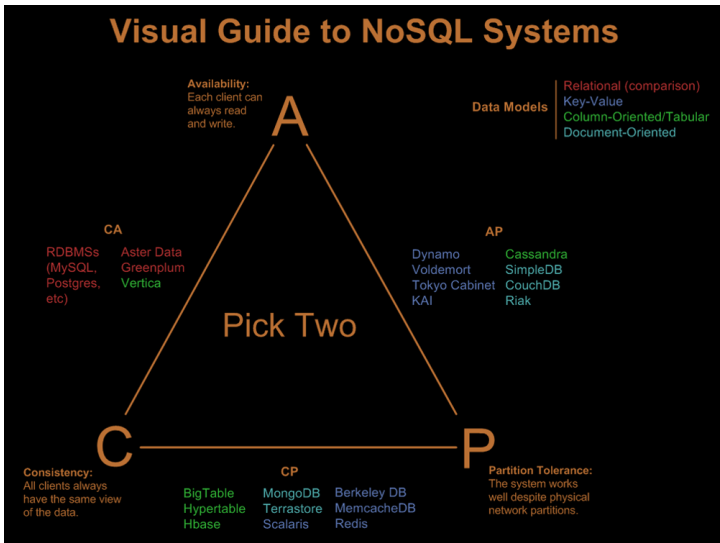
2020. május 12.



**DEBRECENI
EGYETEM**



- CAP tétel
- Konzisztenciafajták
- BASE versus ACID
- NoSQL adatmodellek
 - Kulcs-érték modell
 - Rendezett kulcs-érték modell
 - Oszlopcsalád modell
 - Dokumentum modell
 - Gráfmodell
- NoSQL adatmodellezési technikák



Állítás: Egy elosztott rendszer az alábbi három alapvető képesség közül legfeljebb kettőt tud megvalósítani:

- **konzisztencia (consistency)**, minden csomópont egy adott pillanatban ugyanazt az adatot látja
- **rendelkezésre állás (availability)**, minden kérésre érkezik válasz arról, hogy a kérés végrehajtása sikeres vagy sikertelen volt-e
- **particionálástűrés (partition tolerance)**, a rendszer egy tetszőleges üzenet elvesztése vagy a rendszer egy részének hibája esetén is tovább működik – csak a teljes rendszer hibája, pl. egy generális hálózati hiba, okozhatja a működés hibáját

A tételt Eric Brewer fogalmazta meg:

A. Fox and E.A. Brewer, "Harvest, Yield and Scalable Tolerant Systems", Proc. 7th Workshop Hot Topics in Operating Systems (HotOS 99), IEEE CS, 1999, pp. 174-178.

Bizonyítva 2002-ben:

Seth Gilbert and Nancy Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services", SIGACT News 33, 2 (June 2002), pp. 51-59.

Továbbgondolva 2012-ben:

Brewer, E., "CAP twelve years later: How the "rules" have changed", Computer , vol.45, no.2, Feb. 2012, pp.23-29 .

- **Elhagyni a particionálástűrést:** rakjunk mindent egy gépre, vagy egyetlen, atomi módon elbukó egységre (pl. egy rack-be), skálázhatósági problémát jelent
- **Elhagyni a rendelkezésre állást:** egy esemény bekövetkezésekor az érintett szolgáltatások megvárják, amíg az adatok konzisztenciája helyreáll, ezalatt elérhetetlenné válnak (particionálástűrés másik oldala)
- **Elhagyni a konzisztenciát:** ez a valóságban is sokszor így van
 - egy árucikk-adatbázis konzisztens-e, ha a raktáros épp most tör össze valamit
 - ha egyszerre két rendelés érkezik egyetlen árucikkre, akkor csak az egyiket tudjuk azonnal kiszolgálni, de a másik is beszerzés alá kerül, ha erről értesítjük az ügyfelet (néha nem probléma, de helyfoglalásnál már igen)
 - a rendelkezésre állás sokszor fontosabb a konzisztenciánál

- **Kliens oldali** konzisztencia: hogyan figyeljük meg az adatok változását?
- **Szerver oldali** konzisztencia: hogyan áramlanak a módosítások keresztül a rendszeren, és a rendszer milyen garanciákat tud adni a módosításokra vonatkozóan

- A kliens oldalán az alábbi komponensek helyezkednek el:
 - egy tárolórendszer: tekintsük fekete dobozként
 - egy A folyamat: írja és/vagy olvassa a tárolórendszert
 - B és C folyamatok: A -tól függetlenek, és szintén írják és/vagy olvassák a tárolót, az információ megosztása érdekében kommunikálniuk kell
- A kliens oldali konzisztencia arról szól, hogy a megfigyelők (ez esetben az A , B és C folyamatok) hogyan és mikor látják a tároló egy adatobjektumának változását. Feltesszük, hogy az A folyamat végezte a módosítást. Három konzisztenciafajtáról beszélhetünk:
 - erős konzisztencia (strong consistency)
 - gyenge konzisztencia (weak consistency)
 - esetleges konzisztencia (eventual consistency)

12. előadás: NoSQL adatbázisok

Bevezetés

CAP tétel

Konzisztencia

ACID vs
BASE

NoSQL
rendszerek

NoSQL
adatmodellek

NoSQL adat-
modellezés

Miután a módosítás végrehajtott, minden (akár *A*, akár *B*, akár *C* által végzett) hozzáférés a módosított értéket adja eredményül.

- A rendszer nem garantálja, hogy a későbbi hozzáférések a módosított értéket adják eredményül.
- Több feltételnek is teljesülnie kell, mielőtt az érték visszaadásra kerül.
- A módosítás megtörténte és azon pillanat közötti időszakot, amelyre már garantált, hogy minden megfigyelő mindig a módosított értéket látja, **inkonzisztenciaablaknak** (inconsistency window) nevezzük.

- A gyenge konzisztencia egy speciális formája: a tárolórendszer garantálja, hogy amennyiben az objektumra nem érkezik új változtatás, végül majd minden hozzáférés az utoljára módosított értéket adja vissza.
- Ha nem történik hiba, akkor az inkonzisztenciaablak maximális mérete olyan tényezők alapján meghatározható, mint amilyen a kommunikáció késleltetése, a rendszer terhelése, vagy a replikációs sémába bevont replikák száma.
- A legismertebb esetleges konzisztenciájú rendszer a doménnév-feloldó szolgáltatás (domain name system, DNS).

Legyen

- N az adatok replikáit tároló csomópontok száma
- W azon replikák száma, amelyeknek a módosítás véglegesítését megelőzően nyugtáznuk kell a módosítás beérkezését
- R azon replikák száma, amelyeket érinteni kell akkor, ha egy adatobjektumra olvasási művelet érkezik

Ha $W + R > N$, akkor erős konzisztencia áll fenn: az írók és az olvasók között biztosan van átfedés.

Ha $W + R \leq N$, akkor gyenge/ esetleges konzisztencia van: elképzelhető, hogy írók és olvasók között nincs átfedés.

Példa: replikáció relációs adatbázis-kezelő rendszerben

- **szinkron** módon: a replika módosítása is a tranzakció része, ekkor pl. $N = 2$, $W = 2$ (a szinkron replikáció miatt), $R = 1$, vagyis erős konzisztencia van, hiszen mindegy, a kliens melyik replikát olvassa, konzisztens választ fog kapni (probléma akkor van, ha valamelyik csomópont nem elérhető)
- **aszinkron** módon: a másolat a tranzakciótól függetlenül, többnyire késleltetve jön létre (pl. a naplóbejegyzések átvitelével és rágörgetésével), ekkor pl. $N = 2$, $W = 1$, $R = 1$ vagyis $R + W = N$, a konzisztencia nem garantálható (vagy a módosított csomópontról olvasunk, vagy a másikról, ahová még nem biztos, eljutott a napló)

- **Horizontális particionálás:** a különböző egyedelőfordulásokat azok adatainak egyben tartásával különböző elemekre osztjuk
 - különböző sorok különböző táblába kerülnek, pl. eladási adatok esetén régiók alapján osztunk
 - egy egyedelőfordulás összes adata egyazon partícióra kerül
 - **sharding**ról akkor beszélünk, ha az egyedelőfordulásokat az adatbázis-kezelő rendszer több példánya között osztjuk szét
- **Vertikális particionálás:** az egyedelőfordulások adatait szétbontjuk
 - kevesebb oszloppal rendelkező táblákat készítünk, a maradék oszlopokat pedig további táblákba helyezzük
 - ilyen pl. a normalizálás

ACID: Atomicity, Consistency, Isolation, Durability

- Erős konzisztencia
- Konzervatív (pesszimista) ütemezés
- Bonyolult (séma)evolúció
- Közepponban a commit

BASE: Basically Available, Soft-state, Eventually consistent

- Egyszerűbb és gyorsabb
- Esetleges konzisztencia
- Agresszív (optimista) ütemezés
- Könnyebb evolúció
- Körülbelüli válaszok elegendőek

Ha a tranzakciókezelőtől befut egy kérés az ütemezőhöz, 3 lehetőség van:

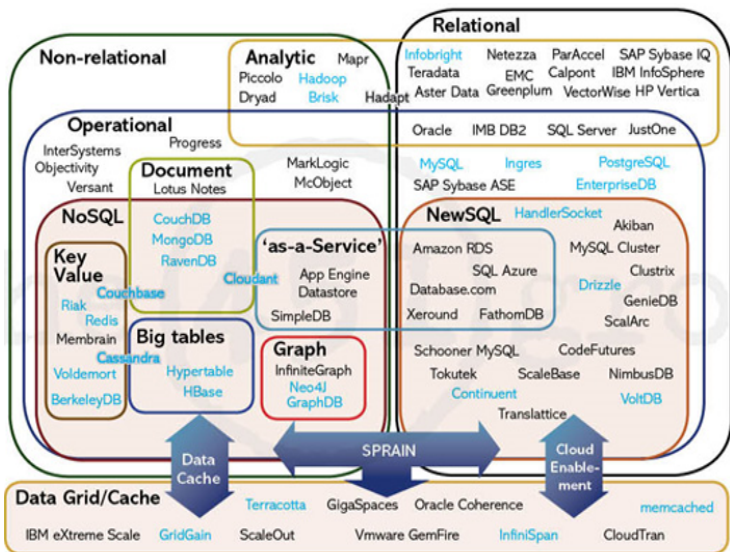
- 1 Azonnal végrehajtani
- 2 Késleltetni (sorba állítani)
- 3 Visszautasítani (abortálni)

Az **agresszív** (optimista) stratégia az 1) pontot favorizálja, azzal együtt, hogy amit adott időn belül nem lehet befejezni, vagy problémát okozna, ott a 3)-ast javasolja.

A **konzervatív** (pesszimista) megközelítés szerint a 2) a jó.

12. előadás:
NoSQL
adatbázisok

- Bevezetés
- CAP tétel
- Konzisztencia
- ACID vs BASE
- NoSQL rendszerek
- NoSQL adatmodellek
- NoSQL adat-modellezés



12. előadás:
NoSQL
adatbázisok

Bevezetés

CAP tétel

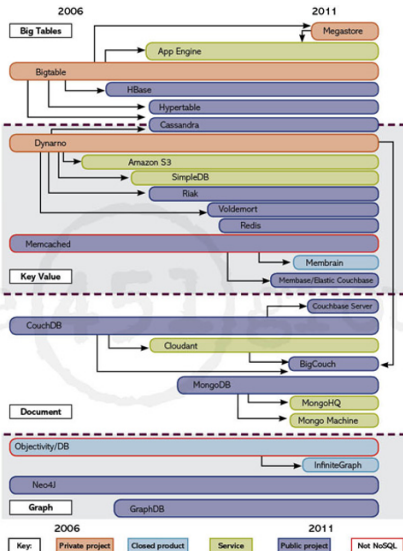
Konzisztencia

ACID vs
BASE

NoSQL
rendszerek

NoSQL
adatmodellek

NoSQL adat-
modelllezés



12. előadás:
NoSQL
adatbázisok

Bevezetés

CAP tétel

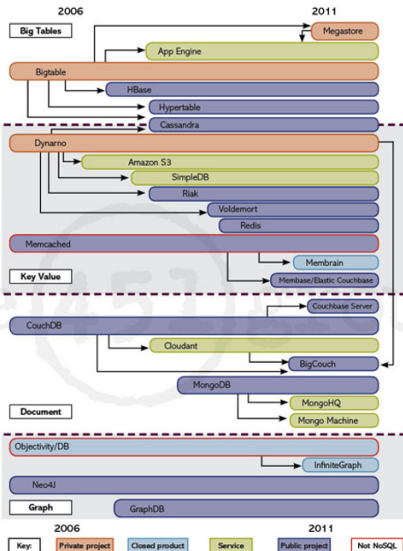
Konzisztencia

ACID vs
BASE

NoSQL
rendszerek

NoSQL
adatmodellek

NoSQL adat-
modelllezés



- Felhasználóorientált kapcsolattartásra tervezve
- Fontos az aggregált jelentések készítése → csoportképzés (group by)
- Nem várható el a felhasználoktól, hogy kezeljék a konkurenciát, adatintegritást, konzisztenciát és az adattípusok validációját → tranzakciós garanciák, sémák, hivatkozási integritás
- Az alkalmazásoknak nincs szüksége olyan gyakran adatbázisbeli aggregációra és sok esetben képesek az integritás és validitás megvalósítására: ezen funkciók megszüntetése nagy hatással van a teljesítményre és a skálázhatóságra!

- **Kulcs–érték modell:** (K, V) párok együttese, ahol K kulcs, V pedig egy érték, kiterjesztése a rendezett kulcs–érték modell: kulcstartományok feldolgozása + megnövelt aggregációs képesség
- **Oszlopcsalád/BigTable modell:** az értékeket mint map-of-maps-of-maps modellezi, oszlopcsaládok, oszlopok és időbélyeggel ellátott verziók segítségével
- **Dokumentum modell:** a sémák tetszőleges bonyolultságúak lehetnek (nem csak map-of maps), adatbázis által kezelt indexek is megjelennek
- **Gráf modell:** a rendezett kulcs–érték modellből származó oldalág, egyedek transzparens modellezése (pl. függőség)

12. előadás: NoSQL adatbázisok

Bevezetés

CAP tétel

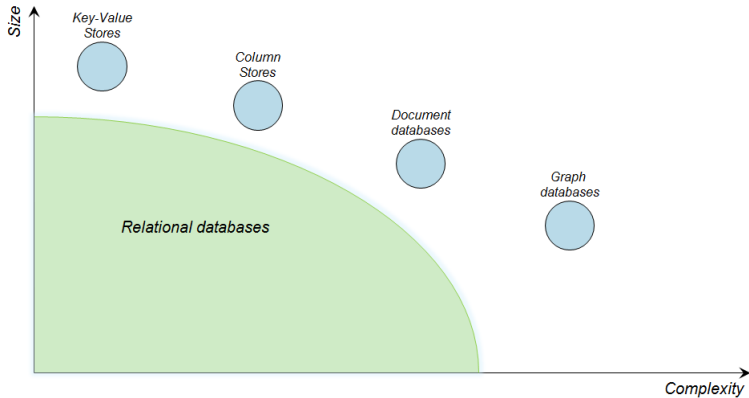
Konzisztencia

ACID vs
BASE

NoSQL
rendszerek

NoSQL
adatmodellek

NoSQL adat-
modellezés



- KVS: key-value store
- Az ezen az elven felépülő adattárak egyszerű hashtáblák
- Az érték egy blob, amit az adattár csak tárol, anélkül, hogy tudná és érdekelné, mit tartalmaz (ennek értelmezése teljesen az alkalmazás felelőssége)
- Osztott adattárakban általában esetleges konzisztenciájú adatokkal dolgozunk (eventual consistency) amelyek „előbb-utóbb” konzisztensek lesznek
- A skálázhatóságot particionálás (sharding) segítségével valósítja meg: a kulcs értéke alapján dől el, hogy a kulcs–érték párt melyik csomóponton kell tárolni (replikációra is szükség lehet!)

A lastVisit kulcshoz tartozó érték az utolsó látogatás időbélyege, a user kulcshoz tartozó érték pedig egy táblázat, amely az ügyfél-azonosítót, az ügyfél nevét és országkódját, valamint az időzónát tárolja.

```
{
  "lastVisit":1324669989288,
  "user":{
    "customerId":"91cfd5bcb7c",
    "name":"Márton Ispány",
    "countryCode":"HUN",
    "tzOffset":"CET"
  }
}
```


Mikor használjuk?

- Munkamenetadatok (lehetséges kulcs: sessionid)
- Bevásárlókosarak adatai (lehetséges kulcs: userid)
- Felhasználói profilok, beállítások

Mikor ne használjuk?

- Ha kapcsolatok vannak az adatok között
- Többműveletes tranzakciók esetén: pl. ha több kulcs mentésekor valamelyik sikertelensége esetén a többi hatását is vissza szeretnénk vonni
- Ha a lekérdezést az adatok (és nem pedig a kulcsok) alapján kell elvégezni
- Ha kulcsok halmazán kell műveletet végezni

- A kulcsok növekvően rendezettek (mint egy szótárban)
- Sok rekord feldolgozásánál könnyen skálázható (scaling-out)
- Kétféle keresés:
 - pontos
 - tartomány

- BigTable-szerű adatbázisok
- Oszlopcsalád: kapcsolódó – gyakran együttesen elért – adatok csoportja, sorokban tárolva
- Egy sor kulcsához számos oszlop tartozik
- Egy oszlop egy kulcs–érték pár (oszlopnév-oszlopérték)
- Minden oszlophoz tartozik egy időbélyeg: pl. adatok lejátságának eldöntésére, írási konfliktusok feloldására

Példa egy oszlopra:

```
{  
  name: "fullName",  
  value: "Márton Ispány",  
  timestamp: 12345667890  
}
```

Mikor használjuk?

- Eseménynaplózás
- Tartalomkezelő rendszerek, blog platformok
- Számlálók: pl. egy webalkalmazásban meg kell számolni és kategorizálni kell az oldalak látogatóit
- Lejáró oszlopok: pl. demóhozzáférés vagy megadott ideig mutatott hirdetések, a nem kellő oszlopok egy megadott idő (time to live, TTL) után automatikusan törlődnek.

Mikor ne használjuk?

- Ha ACID tranzakciókra van szükségünk
- Ha a lekérdezés mintái változnak: az oszlop családok áttervezésére lehet szükség

12. előadás: NoSQL adatbázisok

Bevezetés

CAP tétel

Konzisztencia

ACID vs
BASE

NoSQL
rendszerek

NoSQL
adatmodellek

NoSQL adat-
modellezés

- Dokumentumok tárolására és lekérésére szolgálnak: XML, JSON, BSON, stb.
- A dokumentumok önleíróak, hierarchikus szerkezetűek, kollektiókat és skalár értékeket tartalmazhatnak
- Úgy képzelhetjük el őket, mint olyan kulcs–érték tárolók, ahol az érték megvizsgálható (az maga a dokumentum)
- Egyetlen dokumentum szintjén atomi tranzakciókról beszélhetünk
- Skálázás: sharding segítségével, fontos a kulcs megválasztása

Mikor használjuk?

- Eseménynaplózás: különféle alkalmazásoknak különféle naplózási szükségleteik vannak, különféle események jönnek létre
- Tartalomkezelő rendszerek, blog platformok
- Webes ill. valós idejű analitika: a dokumentum egyes részeinek módosításával könnyen tárolhatók, pl. az oldalletöltések vagy az egyedi látogatók (új metrikákat is könnyen hozzáadhatunk)
- E-kereskedelmi alkalmazások: rugalmas szerkezetre van szükségük a rendelések és termékek tárolására

Mikor ne használjuk?

- Különböző műveleteken átívelő összetett tranzakciók esetén (bár léteznek atomi, dokumentumközi műveleteket támogató rendszerek, pl. RavenDB)
- Változó aggregátumokra vonatkozó lekérdezések esetén

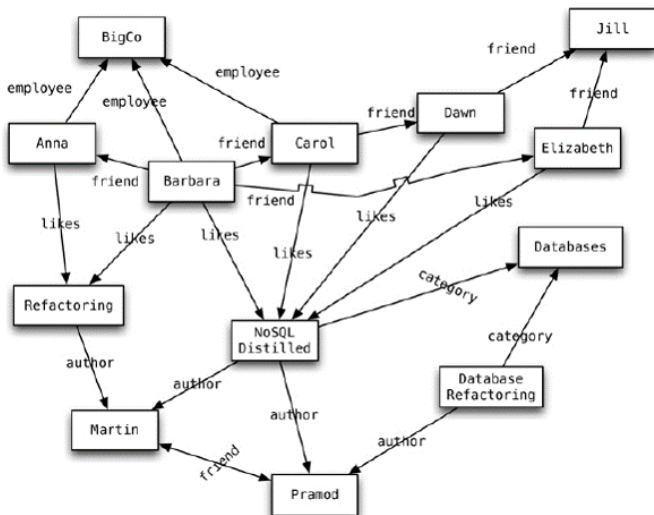
- Csomópontok: az egyedek tulajdonságokkal együtt történő leírásai
- Élek: irányítottak (az egyedek kapcsolatait írják le), van típusuk, lehetnek attribútumaik
- Egy lekérdezés tulajdonképpen egy bejárás
- Mivel összekapcsolódó csomópontokon operálnak, általában nem támogatják azok különböző szerverekre osztását (egy szerveren belül az adatok mindig konzisztensek)
- Skálázhatóság: nehéz ügy
 - RAM / csak olvasható hozzáférést biztosító csomópontok hozzáadása
 - szakterület-specifikus tudás alapján sharding

12. előadás:
NoSQL
adatbázisok

Bevezetés

CAP tétel

Konzisztencia

ACID vs
BASENoSQL
rendszerekNoSQL
adatmodellekNoSQL adat-
modellkezés

Mikor használjuk?

- Összekapcsolódó adatok: szociális, céges stb. hálóok
- Útvonalválasztás, hely alapú szolgáltatások
- Ajánlói rendszerek

Mikor ne használjuk?

- Ha az összes (vagy elegendően sok) csomópontot módosítani kell
- Globális (a teljes gráfot érintő) gráfműveletek

- A relációs modellével szemben alkalmazásorientált megközelítést használunk
 - a relációs modellezést az elérhető adatok szerkezete vezérli, „milyen válaszaim vannak?”
 - a NoSQL modellezést az alkalmazásfüggő elérési minták (vagyis a támogatandó lekérdezések fajtái) vezérik, „milyen kérdéseim vannak?”
- Az adatszerkezetek és algoritmusok mélyebb megértését igénylik
- Az adattöbbszörözés és a denormalizáció alapértelmezés

- Ugyanazon adat több dokumentumba/táblába másolása a lekérdezésfeldolgozás egyszerűsítése ill. optimalizálása, vagy pedig a felhasználói adatok egy bizonyos adatmodellnek történő megfeleltetése érdekében.
- **Kompromisszumok:** Lekérdezési adatmennyiség vagy I/O per lekérdezés vs. összadatmennyiség, Feldolgozási bonyolultság vs. összadatmennyiség
- **Alkalmazhatóság:** kulcs-érték tárolók, dokumentum-adatbázisok, BigTable-szerű adatbázisok

- A kulcs–érték tárolók és a gráfadatbázisok általában nem rendelkeznek megszorítással az értékekre vonatkozóan. Pl. egy felhasználói account olyan összetett kulccsal rendelkező bejegyzések segítségével modellezhető, mint UserID_név, UserID_email vagy UserID_üzenetek stb. Ha egy felhasználónak nincsenek emailjei vagy üzenetei, akkor a megfelelő bejegyzés nem létezik.
- A BigTable alapú modellek egy **oszlopcsaládon** belüli oszlopok változó halmazával és egy **cella** változó számú **verziójával** biztosítja a soft schema képességeket.
- A dokumentum-adatbázisok természetüknél fogva nem rendelkeznek sémával, bár egyikük-másikuk lehetővé teszi a bejövő adatok felhasználó által definiált séma segítségével történő validációját.
- **Alkalmazhatóság:** kulcs–érték tárolók, dokumentum-adatbázisok, BigTable-szerű adatbázisok

Lehetővé teszi, hogy a komplex belső szerkezettel (pl. beágyazott egyedekkel) rendelkező egyedeket osztályokba soroljuk és variálhassuk az egyes egyedek szerkezetét. Ennek két fő előnye van:

- az $1 : N$ kapcsolatok beágyazott egyedek segítségével történő minimalizálása (és ezáltal az összekapcsolások számának csökkentése)
- az üzleti egyedek és a heterogén üzleti egyedek egyetlen dokumentumkollekció ill. tábla segítségével történő modellezésének „technikai” különbségeinek elrejtése

- Minden terméknek vannak attribútumai, amelyek minden termék számára közősek, pl. azonosító, ár, leírás.
- Különbőféle termékek azonban különböző attribútumokkal rendelkezhetnek (pl. könyv esetén szerző, farmer esetén hossz).
 - Ezen attribútumok némelyike $1 : N$ vagy $M : N$ természetű, mint a track a zenei albumok esetén.
 - Néhány egyed nem modellezhető fix típus segítségével, pl. a farmerek attribútumai gyártóspecifikusak vagy akár márkafüggők is lehetnek (félíg strukturált szerkezet).
- Ez ugyan megoldható relációsan, de a megoldás messze nem elegáns.
- A soft schema lehetővé teszi, hogy egyetlen aggregátum (a termék) segítségével modellezhessük az összes terméktípust és attribútumaikat.