DUNAÚJVÁROSI EGYETEM

Adatbáziskezelés 1. vázlat

Tartalomjegyzék

1. Alkalom

1.1. Bevezetés

Táblázatkezelő ←→ Adatbáziskezelő (sorok száma, típus és méretmegkötés, kapcsolódás)

Egy sokmezős táblából több optimalizált tábla (szétbontás, kulcsok, SELECT visszaépítés)

1.1.1. Fontosabb alapfogalmak:

- Adatmodell – logikai (szerkezeti) leírás
- Adatbázis – megvalósított adatmodell

1.1.2. Adatbázis rendszerek felépítése.

Adatbázis rendszerek további szolgáltatásai: Űrlapkészítés, Jelentéskészítés, Varázslók, mentés – visszaállítás, replikáció, grafikus felületek, stb.

1.1.3. Az adatmodellezés három szintje:

- Külső szint: az egyes felhasználók hogyan látják az adatbázist
- Középső vagy koncepcionális szint – a külső szintek összefésülésével kapjuk
- Belső vagy fizikai szint – adatok fizikai elhelyezése és elérési módja

1.1.4. Logikai adatmodellek:

- relációs
- objektum relációs
- hierarchikus
- hálós

1.1.5. A tervezés lépései (ER)

1.2. ER –modell

1.2.1. Attribútum:

- Egyszerű ←→ összetett (születési év, hely),
- egyértékű ←→ többértékű (pl. a kedvenc zenéi),
- forrás ←→ származtatott,
- kulcsattribútum

1.2.2. Egyed(típus) – egyed előfordulás (egyed sorai) – Gyenge egyed

1.2.3. Kapcsolattípusok:

- Kapcsolat foka (mennyi egyedet kapcsol össze)
- A részvétel szerint: teljes (totális) v. részleges (parciális)
- A kapcsolat kardinalitása: 1:1, 1:N, N:M, N:M:L

1.2.4. Ábrázolástechnikai kérdések:

- egyed – gyenge egyed
- attribútum, kulcs attribútum, többértékű attribútum, összetett attribútum
- kapcsolat

2. A relációs modell

2.1. Alapfogalmak

- Egyed → Tábla,
- Egyed előfordulás → rekord,
- első sor mezőnevek,
- Attribútumok → tábla oszlopai
- oszlopok száma → a táblázat fokszáma
- sorok száma → a táblázat kardinalitása
- Elsődleges kulcs
- Idegen kulcs
- Funkcionális függőség: az adott attribútumtól függ a tábla több attribútuma (pl. elsődleges kulcs).

2.2. Normalizálás

A normalizálás lényegében táblázat szétbontó relációs műveletek sorozata. Előnye:

- csökken a tárolási igény,
- megszűnnek a törlési, módosítási, beszúrási anomáliák
- Logikailag áttekinthetőbb lesz az adatbázis

0. NF

B_AZON	B_NEV	B_CIM	BETEGSEG	OSZT_AZON	OSZT_NEV	FŐORVOS	GYOGYSZER
444	Kiss Cili	Piripócs	sérv	1	sebészet	Dr. Nagy	Algopirin Semicilin
444	Kiss Cili	Piripócs	hályog	2	szemészet	Dr. Joó	Semicilin
333	Nagy Pál	Pápa	hályog	2	szemészet	Dr. Joó	Sumetrolin Demalgon

1NF:

- Az oszlopok száma és sorrendje minden sorban azonos.
- Minden oszlop csak meghatározott értékeket vehet fel. (adattípus és méret megkötés)
- A rekord oszlopai csak egy-egy értéket vehetnek fel. Többértékű tulajdonság nem lehet (pl.: szak1, szak2)
- Minden sorhoz egy egyedi kulcs tartozik, amitől az összes többi attribútum függ (nincs két egyforma sor).

B_AZON	B_NEV	B_CIM	**BETEGSEG**	OSZT_AZON	OSZT_NEV	FŐORVOS	**GYOGYSZER**
444	Kiss Cili	Piripócs	sérv	1	sebészet	Dr. Nagy	Algopirin
444	Kiss Cili	Piripócs	sérv	1	sebészet	Dr. Nagy	Semicilin
444	Kiss Cili	Piripócs	hályog	2	szemészet	Dr. Joó	Semicilin
333	Nagy Pál	Pápa	hályog	2	szemészet	Dr. Joó	Sumetrolin
333	Nagy Pál	Pápa	hályog	2	szemészet	Dr. Joó	Demalgon

2NF:

- 1NF-ban van
- A nem kulcs attribútumok funkcionálisan teljesen függenek az elsődleges kulcstól.

Beteg

B_AZON	B_NEV	B_CIM
444	Kiss Cili	Piripócs
333	Nagy Pál	Pápa

Ki Mire Mit Szed

B_AZON	**BETEGSEG**	**GYOGYSZER**
444	sérv	Algopirin
444	sérv	Semicilin
444	hályog	Semicilin
333	hályog	Sumetrolin
333	hályog	Demalgon

Osztály

BETEGSEG	OSZT_AZON	OSZT_NEV	FŐORVOS
sérv	1	sebészet	Dr. Nagy
hályog	2	szemészet	Dr. Joó

3NF:

- 2NF-ban van
- Funkcionális függés csak az elsődleges kulcsból indul ki, vagyis megszüntettük a közvetett (tranzitív) függéseket.

Probléma:

BETEGSEG	OSZT_AZON	OSZT_NEV	FŐORVOS
sérv	1	sebészet	Dr. Nagy
hályog	2	szemészet	Dr. Joó
vakbél	1	sebészet	Dr. Nagy

Megoldás:

Beteg

B_AZON	B_NEV	B_CIM
444	Kiss Cili	Piripócs
333	Nagy Pál	Pápa

Ki Mire Mit Szed

B_AZON	**BETEGSEG**	**GYOGYSZER**
444	sérv	Algopirin
444	sérv	Semicilin
444	hályog	Semicilin
333	hályog	Sumetrolin
333	hályog	Demalgon

BETEGSEG	OSZT_AZON
sérv	1
hályog	2
vakbél	1

OSZT_AZON	OSZT_NEV	FŐORVOS
1	sebészet	Dr. Nagy
2	szemészet	Dr. Joó

4NF:

- 3NF-ban van
- Legfeljebb egy többértékű függés van benne (minta és festő, minta és tárgy között) „egy sok kapcsolat”

Példa:

0. NF

Minta	Festő	Tárgy
nagy tulipán	Szabó Veronika Kalmár Kálmán	40 cm-es váza lapos tányér bombontartó
kis rózsa	Hídvégi Karola	hamutál süteményes tányér

1. 2. 3. NF

Minta	Festő	Tárgy
nagy tulipán	Szabó Veronika	40 cm-es váza
nagy tulipán	Szabó Veronika	lapos tányér
nagy tulipán	Szabó Veronika	bombontartó
nagy tulipán	Kalmár Kálmán	40 cm-es váza
nagy tulipán	Kalmár Kálmán	lapos tányér
nagy tulipán	Kalmár Kálmán	bombontartó
kis rózsa	Hídvégi Karola	hamutál
kis rózsa	Hídvégi Karola	süteményes tányér

4NF

Mit fest Ki

Minta	Festő
nagy tulipán	Szabó Veronika
nagy tulipán	Kalmár Kálmán
kis rózsa	Hídvégi Karola

Mit mire Fest

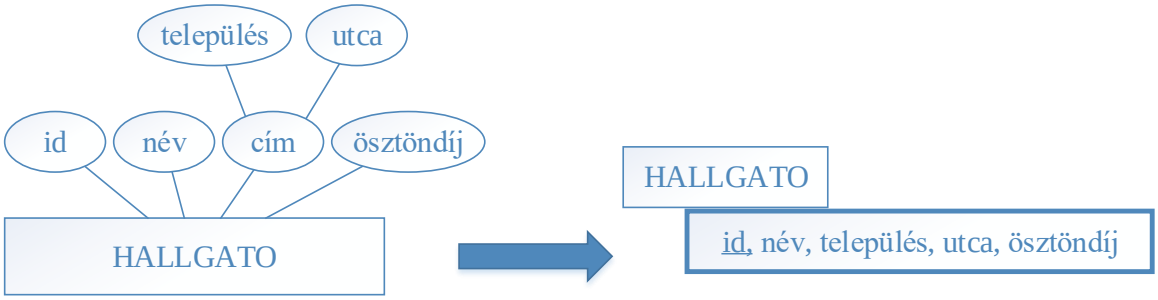
Minta	Tárgy
nagy tulipán	40 cm-es váza
nagy tulipán	lapos tányér
nagy tulipán	bombontartó
kis rózsa	hamutál
kis rózsa	süteményes tányér

2.3. Relációs műveletek

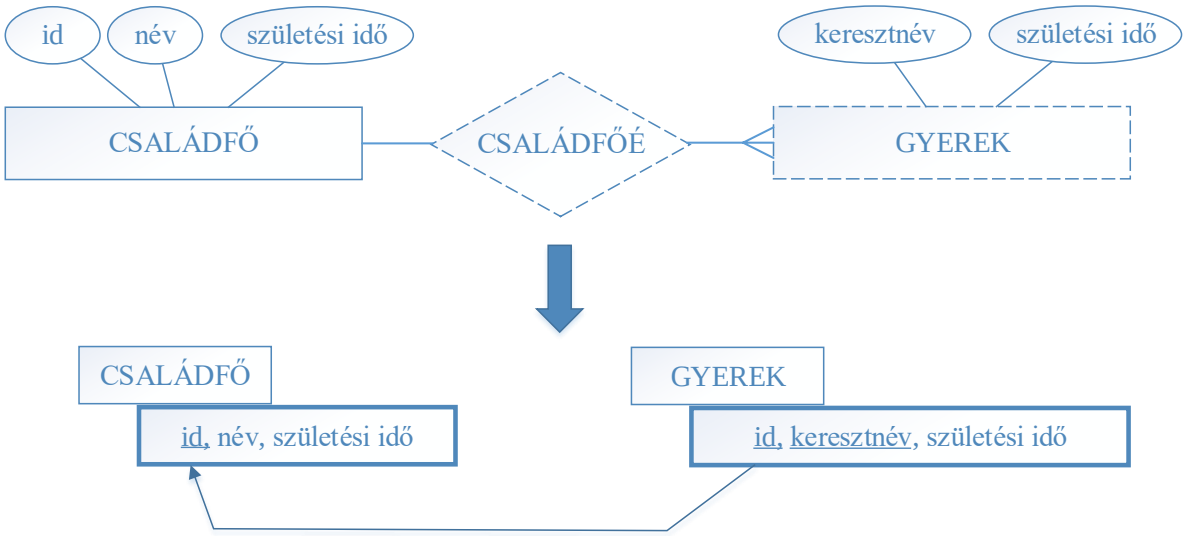
- RENAME(oszlop1,oszlop2): oszlop átnevezése
- RESTRICT: eredmény = RESTRICT tábla WHERE feltétel
- PROJECT (vetület): eredmény = tábla PROJECT(oszlop1, oszlop2) „Select”
- TIMES (keresztszorzat)
- UNION
- INTERSECTION
- JOIN

3. ER-modellből relációs modell

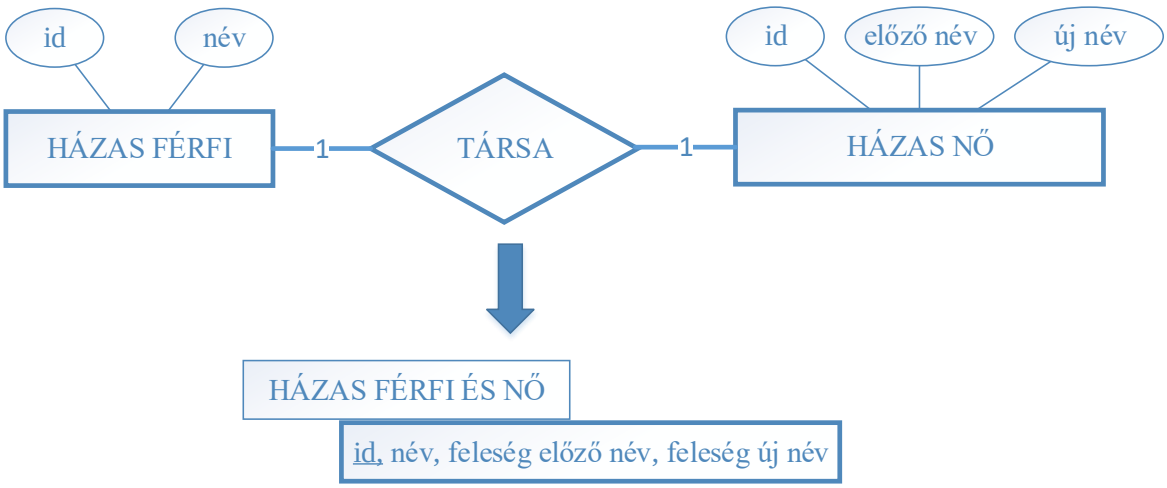
3.1. Egyed leképezése



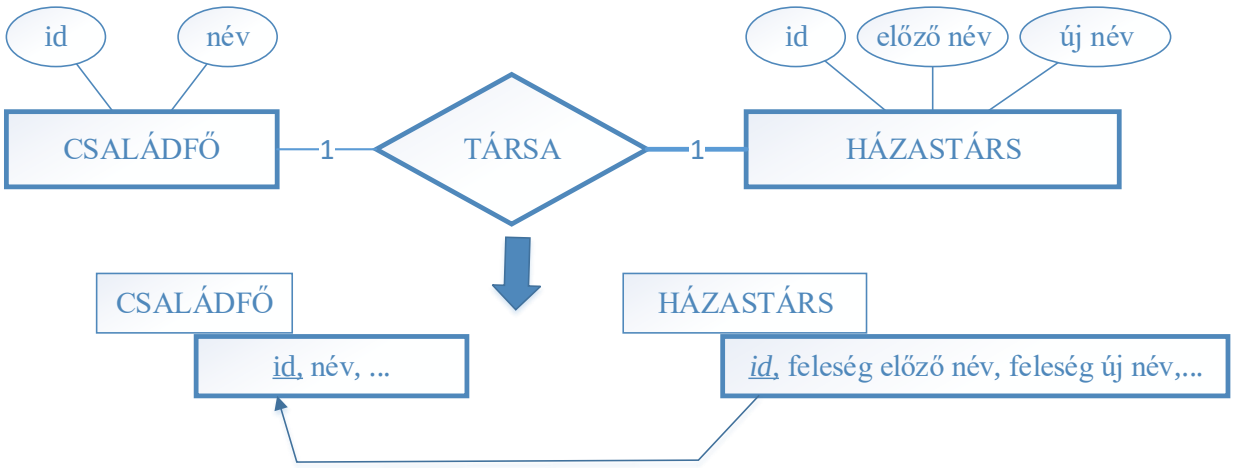
3.2. Gyenge egyed leképezése



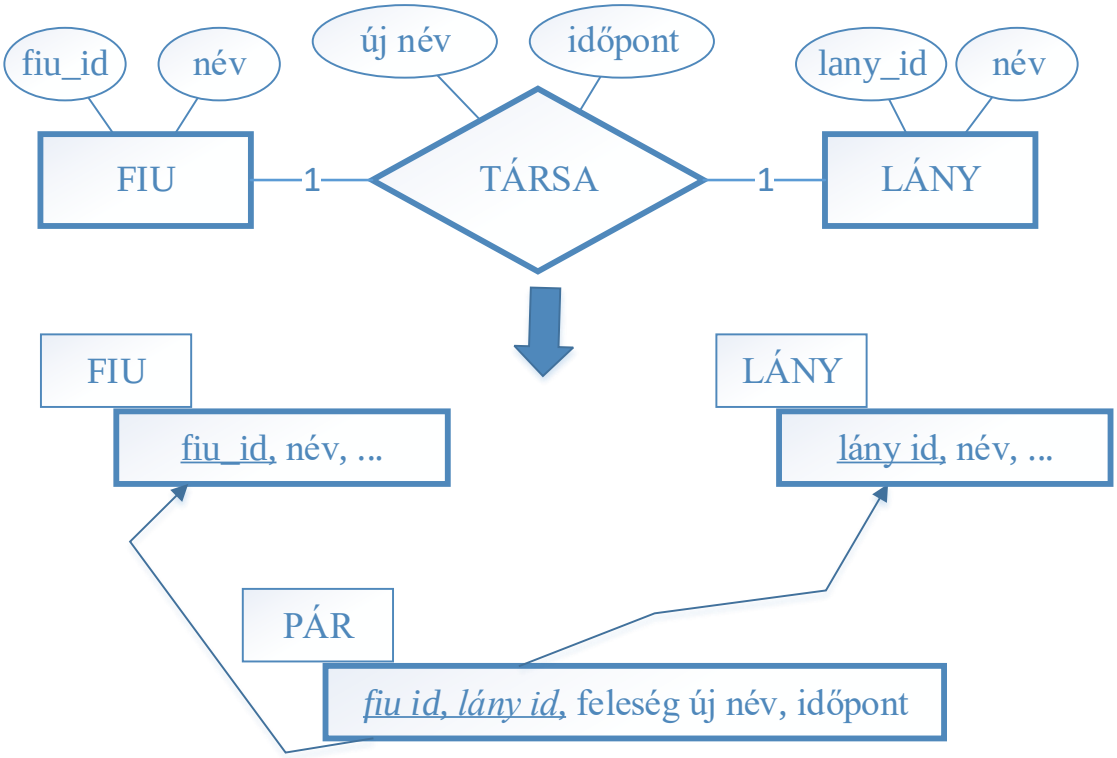
3.3. 1:1 kapcsolat, totális-totális részvétel



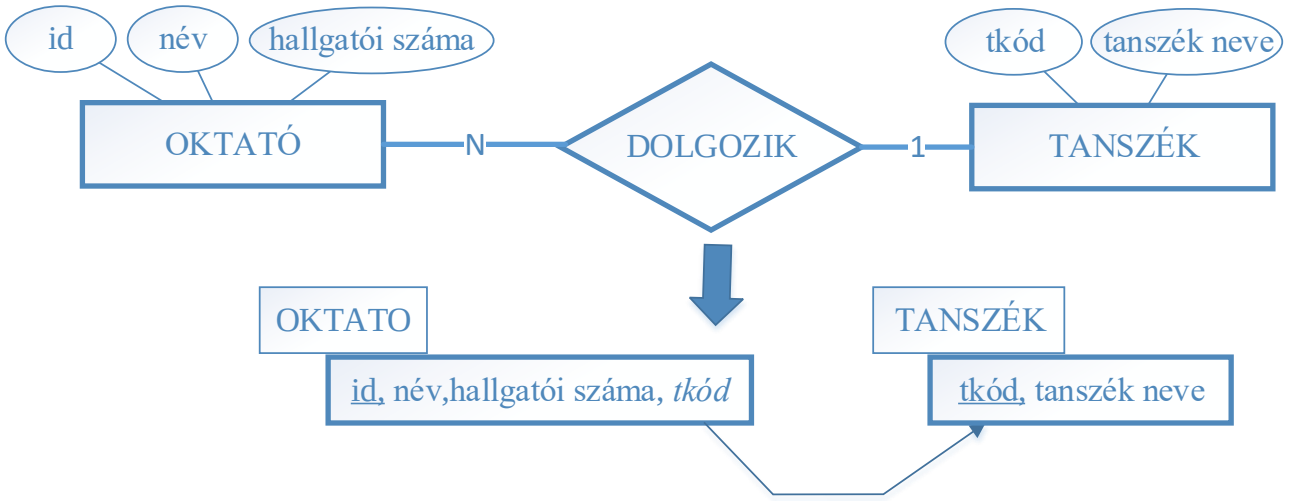
3.4. 1:1 kapcsolat, parciális-totális részvétel



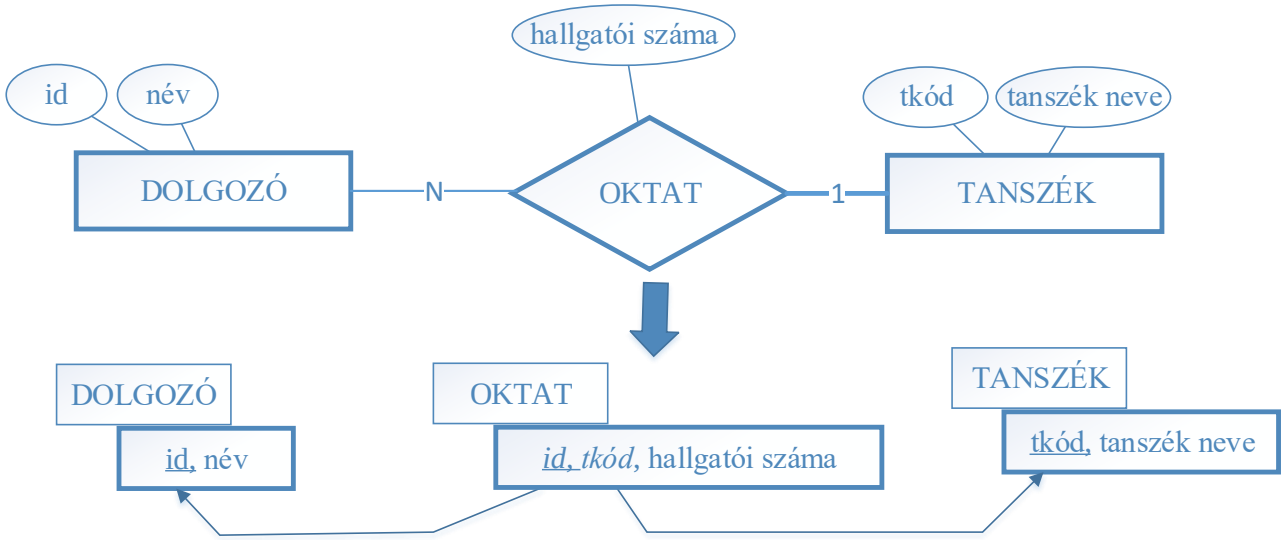
3.5. 1:1 kapcsolat, parciális-parciális részvétel



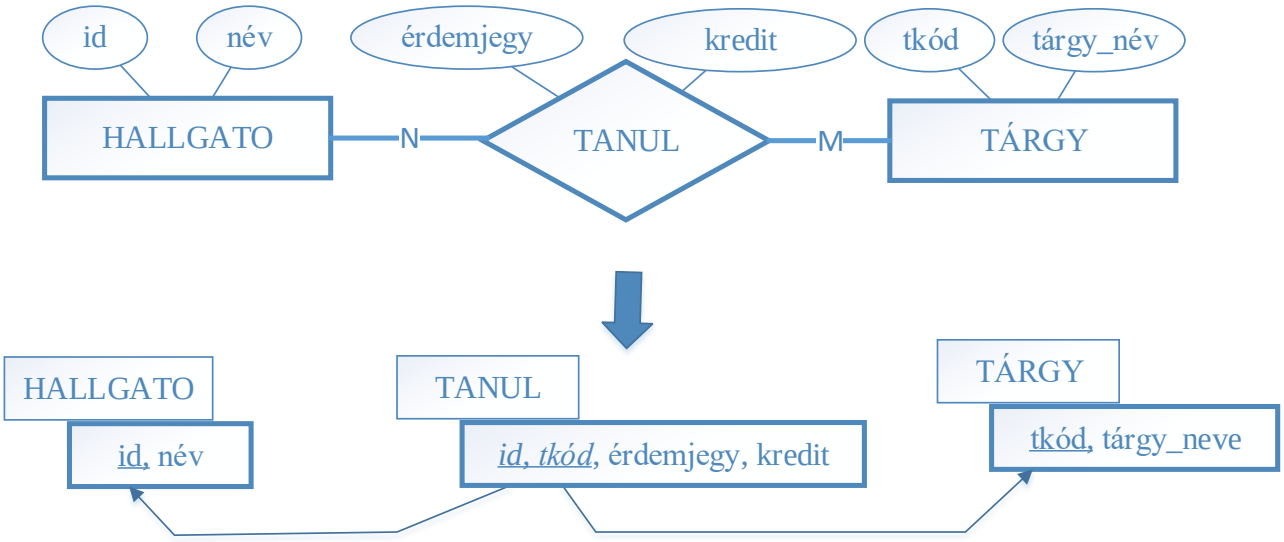
3.6. 1:N kapcsolat, totális N oldal



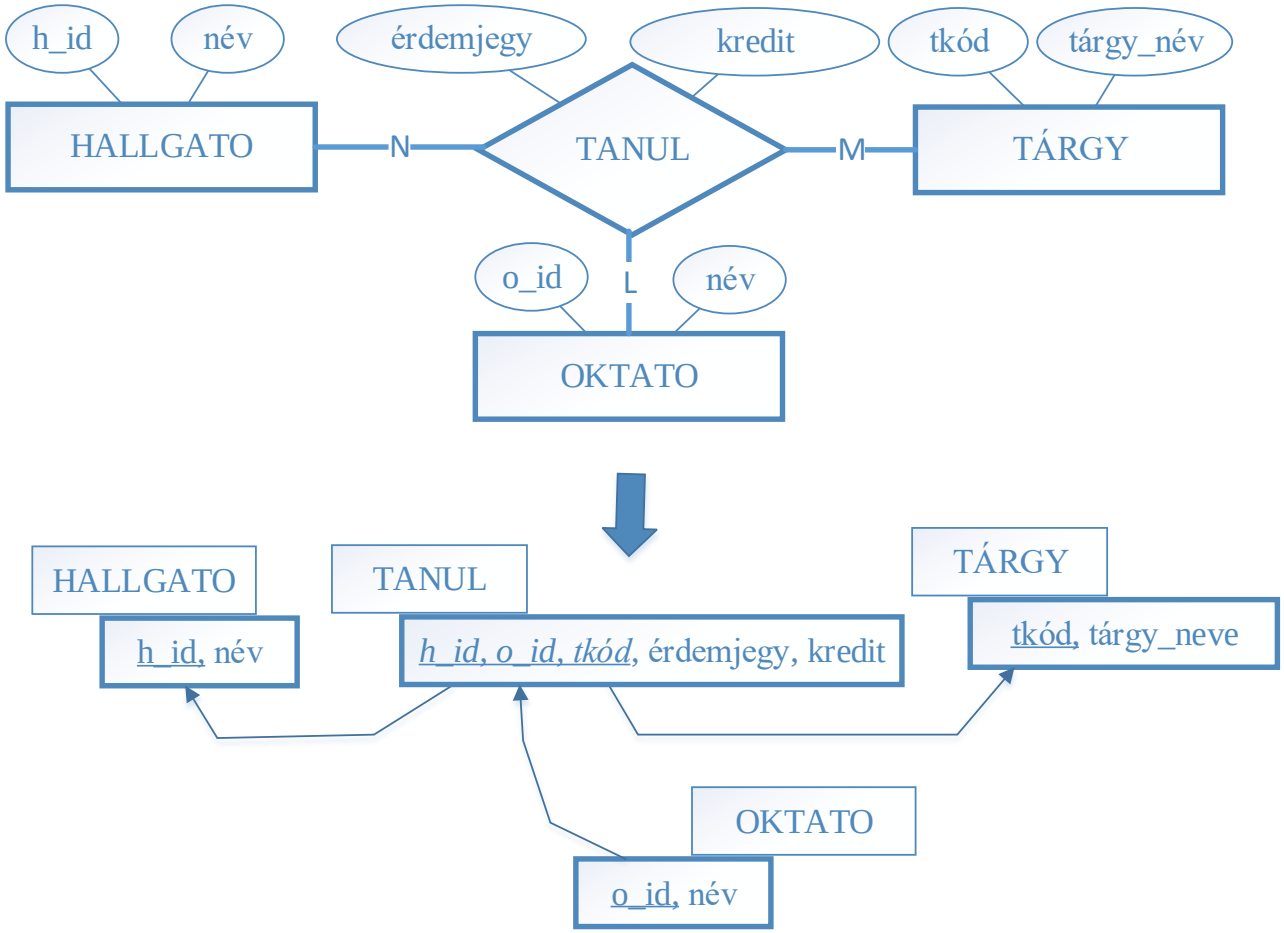
3.7. 1:N kapcsolat, parciális N oldal



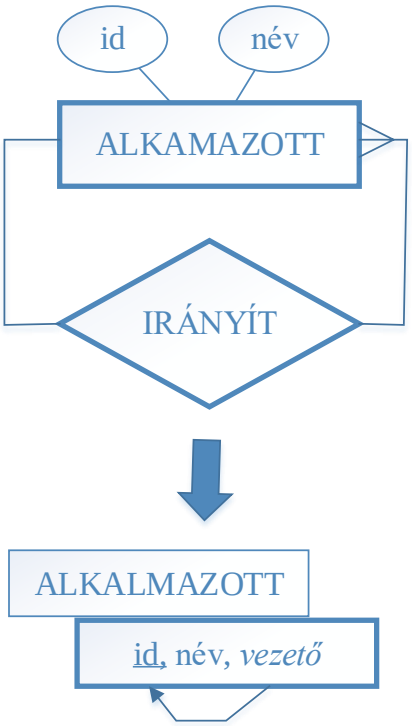
3.8. N:M kapcsolat



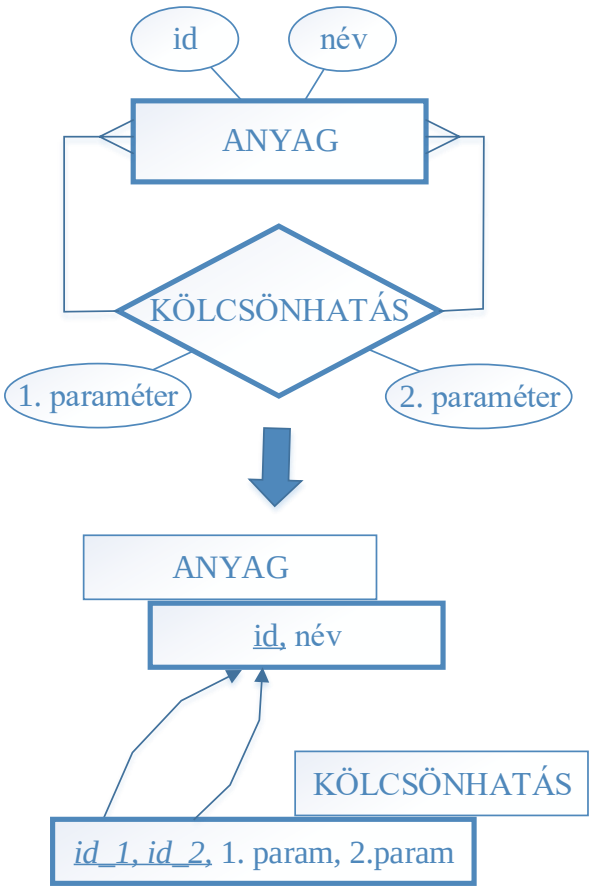
3.9. Többágú N:M kapcsolat



3.10. Rekurzív 1:N kapcsolat



3.11. Rekurzív N:M kapcsolat

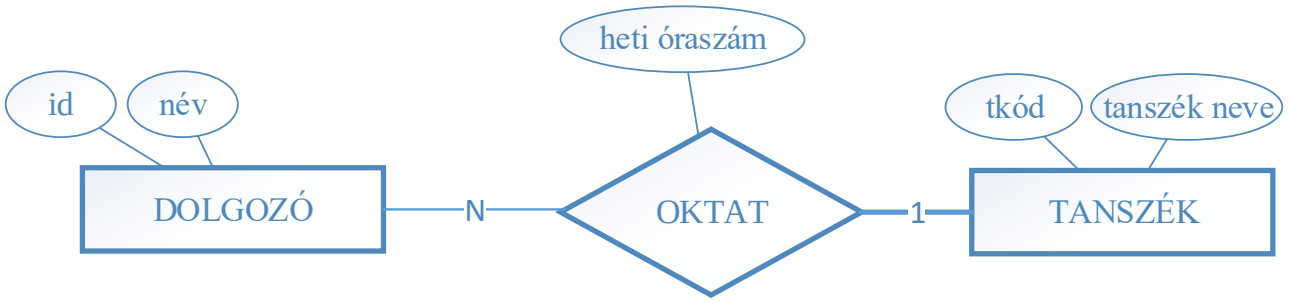


4. Kibővített ER modell

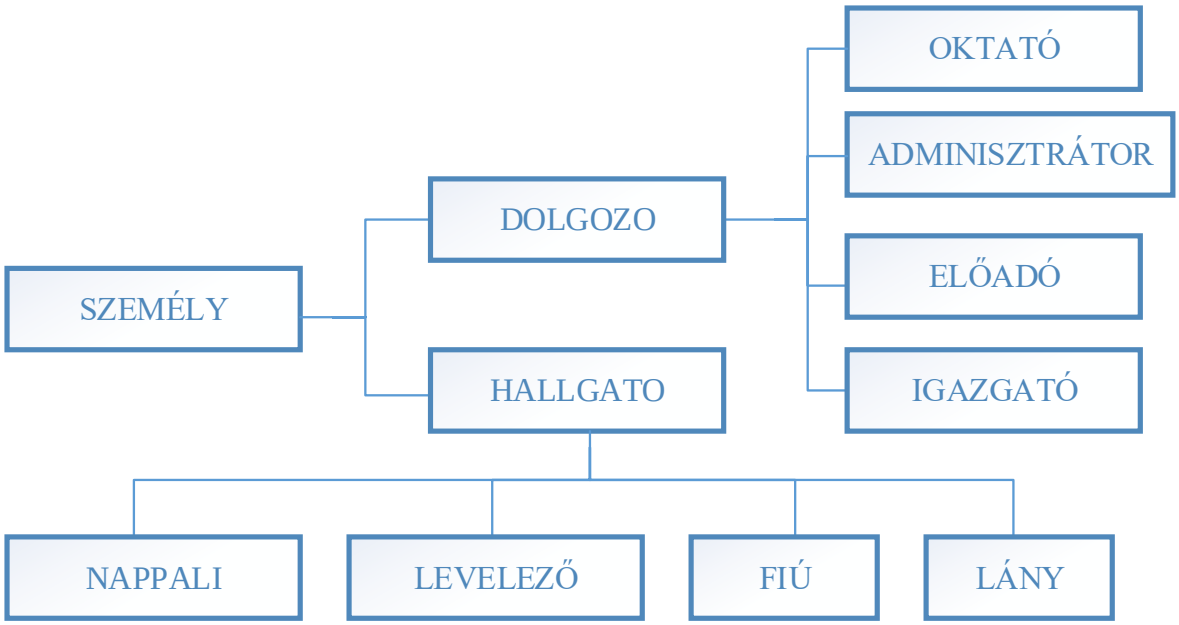
- Kibővített ER modell
  - Osztály, tulajdonságok öröklődése
  - Specializáció, általánosítás
  - Kategóriák
- EER modellből relációs modell

4.1. Osztály, tulajdonságok öröklődése

ER modell probléma: a sok oldal parciális részvétel, nem elég kifejező



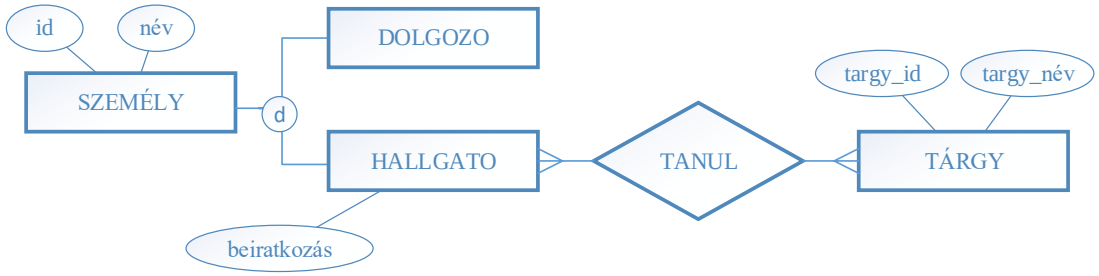
Osztályok



Előnyei:

- az adott alosztályra jellemző speciális tulajdonságok
- az alosztályok is részt vehetnek kapcsolatokban

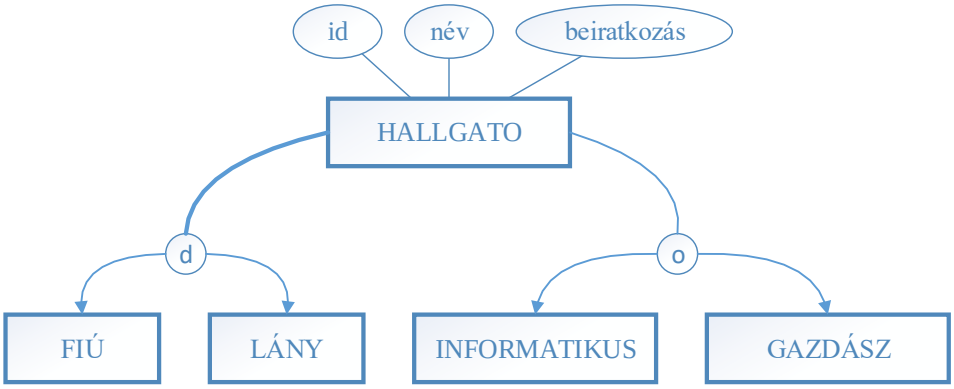
4.2. Specializáció, általánosítás



A specializáció lehetővé teszi, hogy:

- alosztályokat adjunk meg egy egyedtípushoz
- minden alosztály csak rá jellemző tulajdonságokat kapjon
- az alosztályok részt vehetnek kapcsolatokban

4.2.1. A specializáció, általánosítás esetei

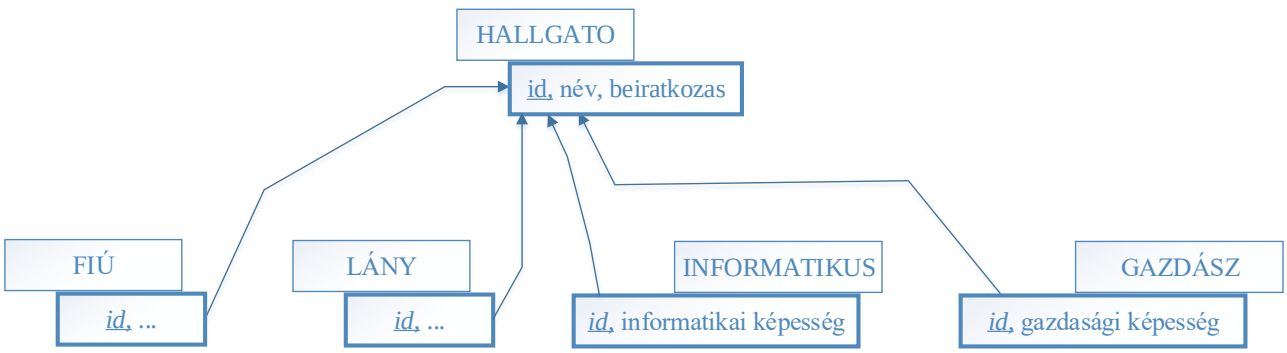


Négy esetet különböztetünk meg:

- elkülönülő (diszjunkt) és teljes (vastag vonal, nincs olyan egyedelőfordulás, amely nem része valamelyik alosztálynak is)
- elkülönülő (diszjunkt) és részleges
- átfedő és teljes
- átfedő és részleges

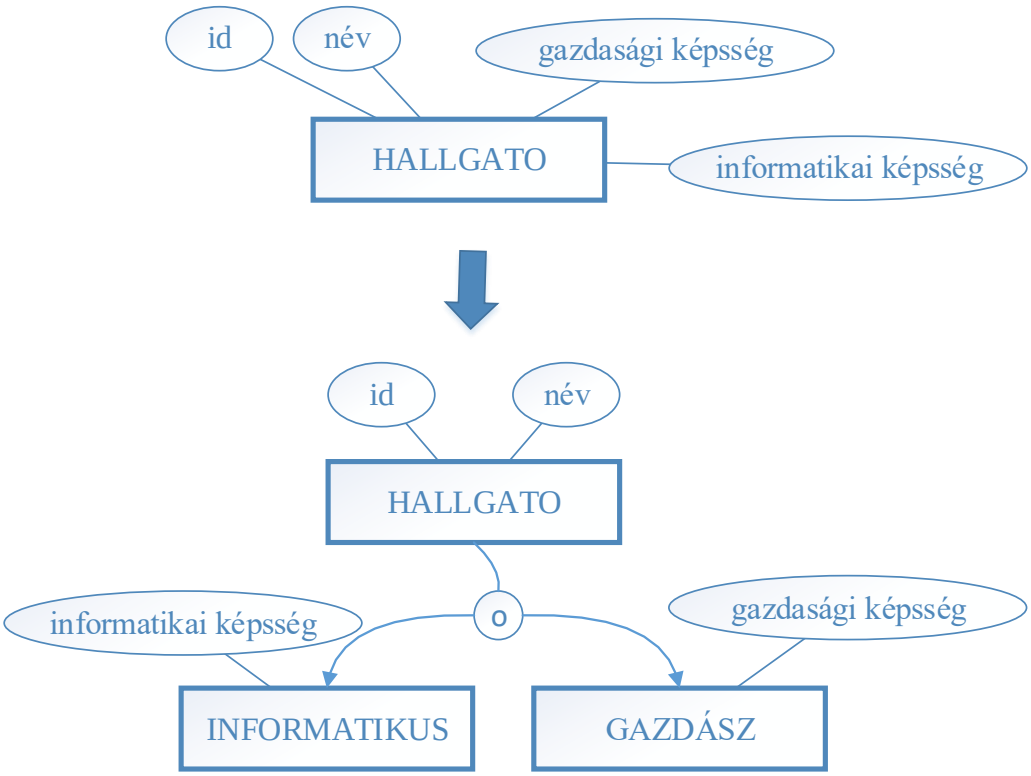
Mi történik beszúrás és törlés esetén?

4.2.2. Leképzés

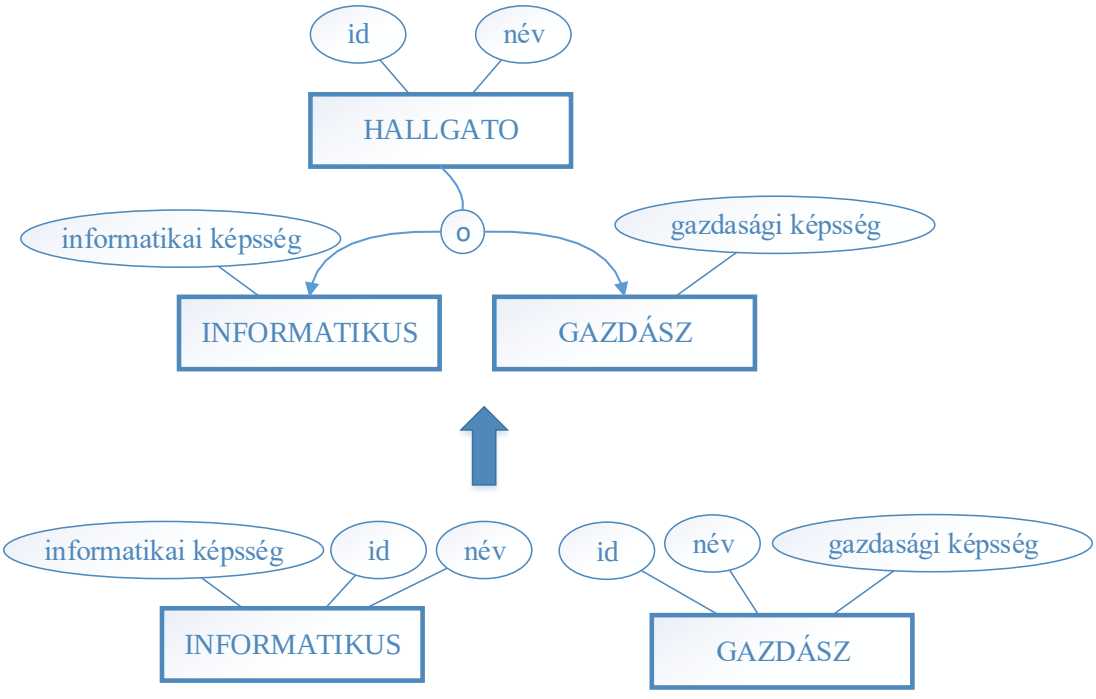


4.2.3. Általánosítás vagy specializáció?

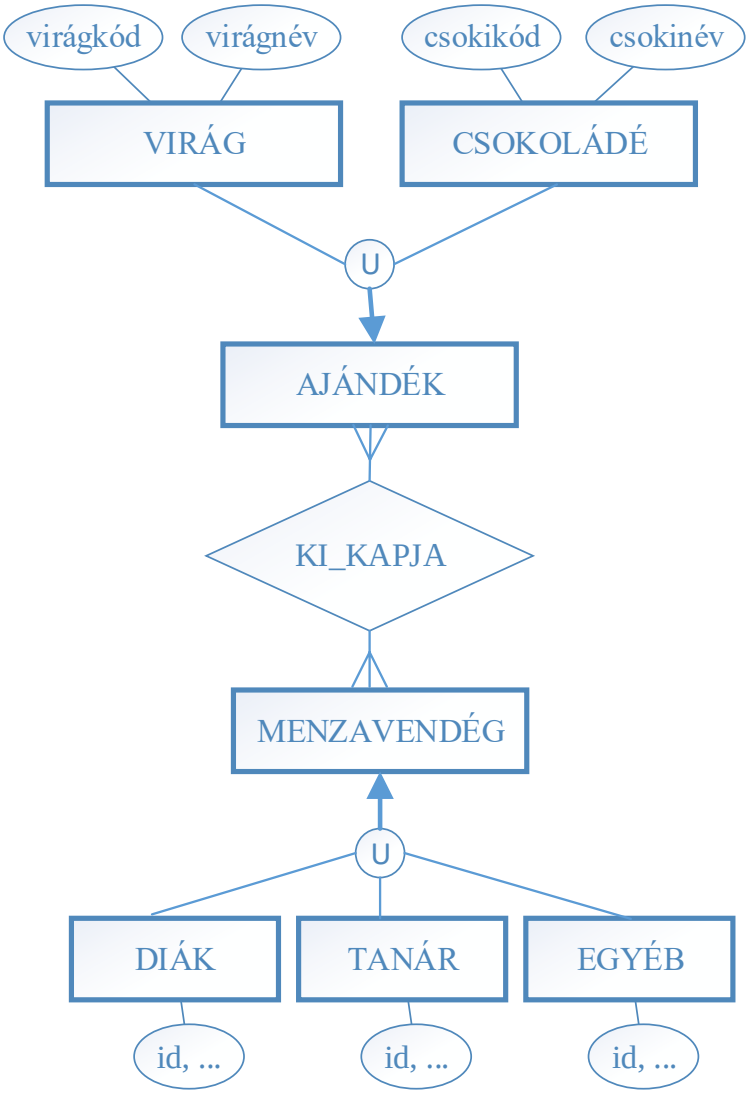
4.2.4. Felülről lefelé tervezés



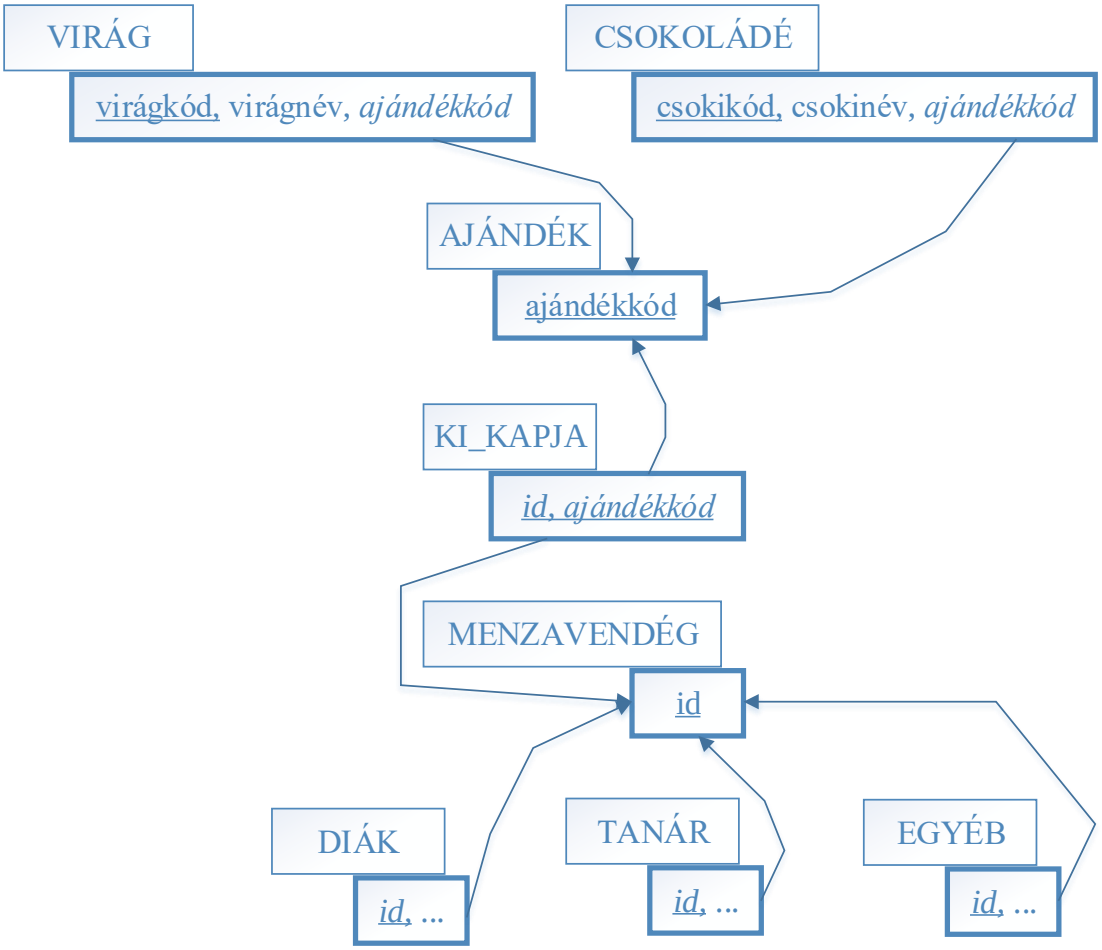
4.2.5. Alulról felfelé történő tervezés.



4.3. Kategória



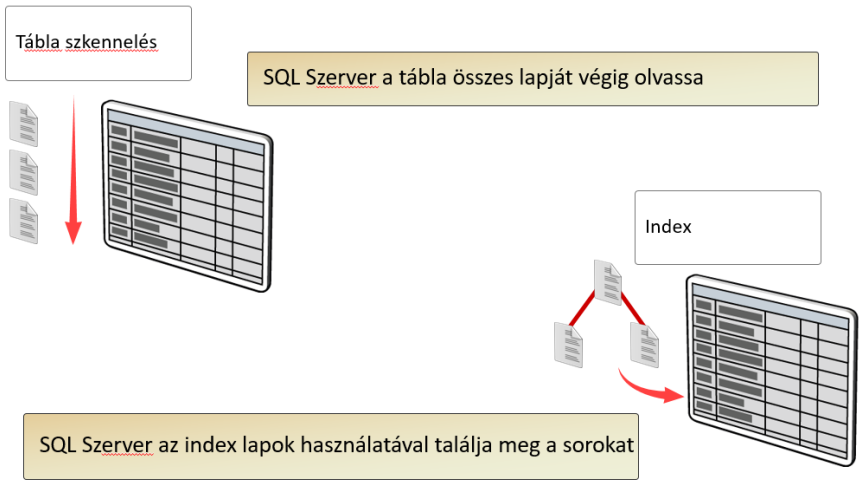
4.3.1. Leképzés



5. Nagy adatbázisok néhány kérdése

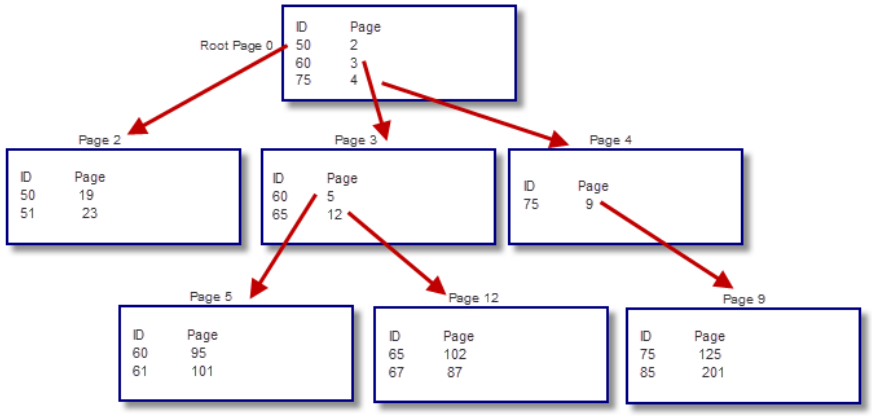
5.1. Indexelés

Hogyan éri el az SQL szerver az adatokat



Index struktúra:

- Az Indexek leggyakrabban a fa struktúrán alapulnak
- A legfelső szint (csomópont) root (gyökér) node; a legalsó csomópontokat levélnek nevezzük



Indexek jellemzői:

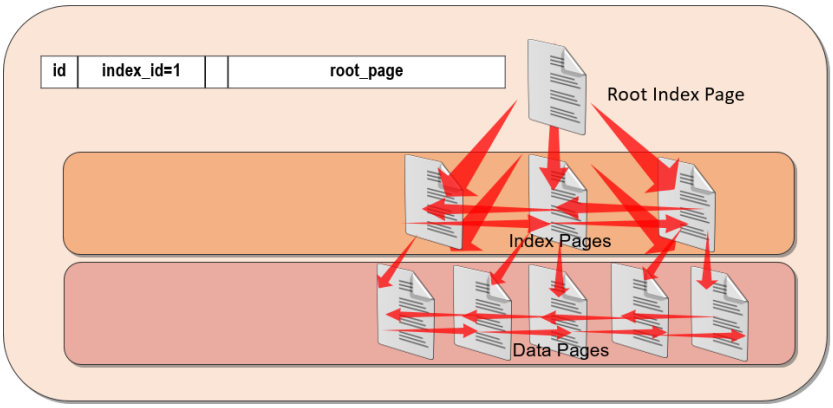
- Szelektivitás (az adott kulcsra vonatkoztatva)
  - A teljes rekord szám mekkora részét szeretnénk lekérdezni?
  - A magas szelektivitás azt jelenti, hogy az eredmény (a kiinduló halmaz soraihoz képest) kevés sor (személyi szám vs. neme).
- Sűrűség (az adott kulcsra vonatkoztatva)
  - Az adott kulcs egyedisége (duplikátum. (pl.: magasság, testsúly)
- Mélység: hány szintből áll a fastruktúra

Index típusok:

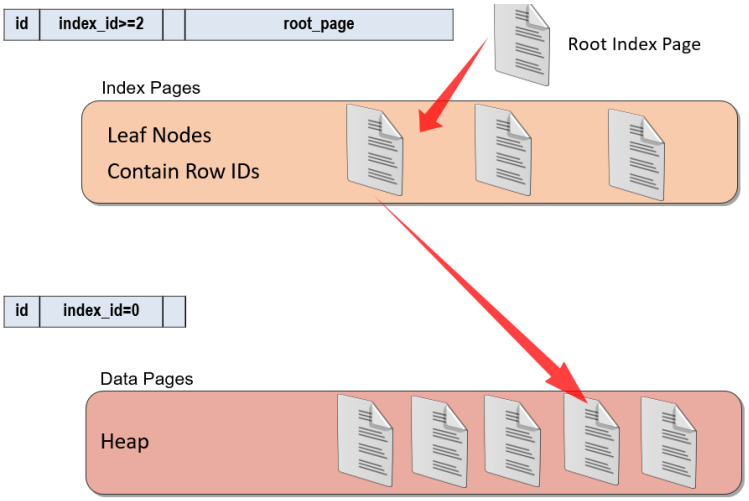
- Egy oszlopos
- Több oszlopos (a szelektivitás szerint célszerű előre rakni)
- Clustered, nonclustered
- Szűrt index
- Index kiegészítő oszlopokkal („fedőindex”)

Clustered index:

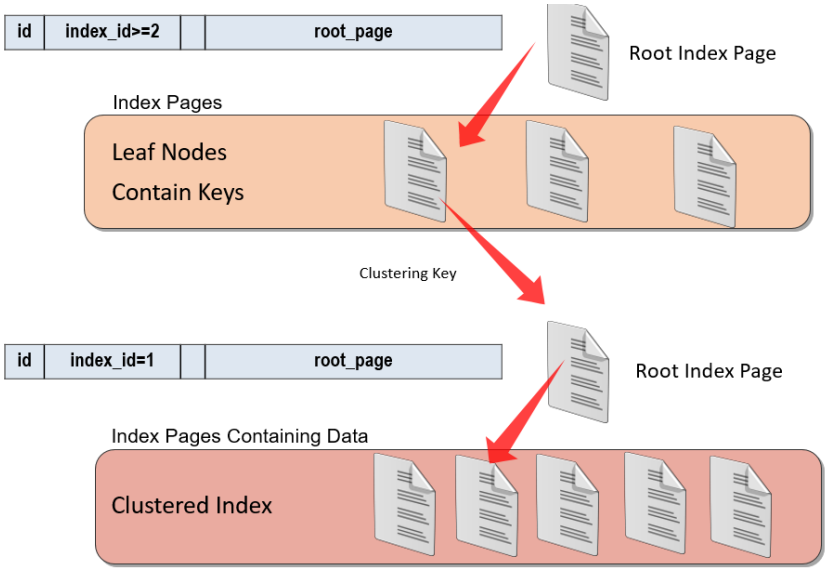
- A tábla rekordjai fizikailag sorba rendezettek.
- Egy táblának egy Clustered indexe lehet.



Nonclustered index



Nonclustered index clustered indexxel



5.2. Nézetek (View)

5.3. Szerepkörök

5.4. Tranzakciókezelés

A tranzakció az adatbázis-műveletek végrehajtási egysége. Amennyiben beágyazott SQL-t használva, a programozó készíti el a tranzakciót, akkor egy tranzakcióban több SQL lekérdezés és módosítás is szerepelhet. Tipikus beágyazott SQL rendszerben a tranzakció adatbázis-akciók végrehajtásával kezdődik, és egy COMMIT vagy ROLLBACK (abort) paranccsal fejeződik be.

Általános elvárás, hogy a tranzakciót atomosan kell végrehajtani, ami azt jelenti, hogy mindent-vagy-semmit módon, és időben egy egységként kell működnie. A tranzakciók korrekt végrehajtásának biztosítása a *tranzakciókezelő* feladata. A tranzakciókezelő részrendszer egy sor feladatot lát el, közöttük:

- Jelzéseket és a naplóban tárolandó információkat is átadja a naplókezelőnek
- Biztosítja, hogy a párhuzamosan végrehajtott tranzakciók ne zavarhassák egymás működését.

A tranzakciókezelő a tranzakció tevékenységeiről üzeneteket küld a naplókezelőnek, üzen a pufferkezelőnek tartalmát szabad-e, vagy kell-e lemezre másolni, és üzen a lekérdezésfeldolgozónak arról, hogy a tranzakcióban előírt lekérdezéseket, vagy más adatbázis-műveleteket kell végrehajtania.

A naplókezelő a naplót tartja karban. Együtt kell működnie a pufferkezelővel, hiszen a naplózandó információ elsődlegesen a memória pufferekben jelenik meg, és bizonyos időnként a pufferek tartalmát lemezre kell másolni. A napló, adat lévén, valamely háttértárolón kap helyet.

A helyreállításkezelő alrendszer akkor aktivizálódik, ha baj van. Megvizsgálja a naplót, és ha szükséges, a naplót használva, helyreállítja az adatokat. A lemez elérése a pufferkezelőn át történik.

Konzisztencia, korrektség. Bizonyos adatbázis állapotokat konzisztensnek tekintünk, míg a többi adatbázis állapotot inkonzisztensnek minősítjük. A konzisztens állapotok kielégítik az adatbázisra vonatkozó összes megszorításokat, elvárásokat.

A tranzakciókra vonatkozó alapvető feltételezés: *A korrektség alapelve:* Ha a tranzakciót minden más tranzakciótól függetlenül és rendszerhiba nélkül végrehajtjuk, és, ha induláskor az adatbázis konzisztens állapotban volt, akkor a tranzakció befejezése után ismét konzisztens állapotban lesz.

A korrektség alapelvéhez kapcsolódik a naplózás technikája, amiből két dolog következik:

- A tranzakció atomi; azaz teljes egészében vagy végrehajtandó, vagy egyáltalán nem. Ha a tranzakciónak csak egy részét sikerült végrehajtani, akkor nagy esélyünk van arra, hogy az általa előállított adatbázis állapot nem lesz konzisztens állapot.
- A párhuzamosan végrehajtott tranzakciók nagy eséllyel inkonzisztens állapothoz vezetnek, hacsak nem alkalmazunk konkurenciavezérlést.

A valós feladatok jelentős része nem érhető el egyetlen SQL utasítás végrehajtásával, hanem az elvégzendő teendők egy SQL utasításokat (is) tartalmazó utasítás-sorozat segítségével írhatók le. Ezek koordinálatlan párhuzamos végrehajtása és/vagy a végrehajtás során fellépő hibák az adatbázisban nem kívánt állapotot hozhatnak létre.

Az adatbázisrendszertől elvárjuk, hogy az ilyen problémákat helyesen kezelje; azt viszont az adatbázisrendszer „magától” nem tudhatja, hogy mely utasítássorozat az ami felhasználói szempontból egységként kezelendő; ezt az adatbázisrendszerrel tudatnunk kell. Az egységként kezelendő, együttesen végrehajtandó utasítássorozat neve: tranzakció.

Sorbarendezhetőség: A párhuzamosan végrehajtott tranzakciók végrehajtásának olyan megszervezése, hogy az adatbázisbani hatásuk olyan legyen, mintha egymástól elkülönülten, egymás után hajtanánk végre.

Műveletek atomisága: A tranzakció végrehajtása közben fellépő hibák szempontjából a tranzakció egységes egészként „atomosan” kezelendő. El kell érni, hogy a tranzakció vagy teljesen hajtódjon végre, vagy ha ez nem sikeres, akkor az adatbázis kerüljön olyan helyzetbe, mintha a tranzakciót egyáltalán nem hajtották volna végre. (Jóllehet annak egy része már végrehajtódott.)

Tranzakciók

A tranzakciók implementációja adatbázisrendszerenként eltérő lehet!
A tranzakció kezdete: START TRANSACTION vagy pl.
BEGIN TRANSACTION
A tranzakció befejezése:

- COMMIT jelzi a sikeres befejezést – az adatbázisban a tranzakció által végrehajtott változtatásokat véglegesíti. Ez előtt a változások a többi tranzakció számára vagy láthatók, vagy nem.
- • ROLLBACK jelzi a sikertelen befejezést. Ilyenkor az adatbázisrendszer a tranzakció által végrehajtott változtatásokatsemmissé teszi (visszagörgeti). A ROLLBACK utasítást az adatbázisrendszer is kiadhatja.

Csak olvasó tranzakciók
A tranzakció ilyen tulajdonságát a:
SET TRANSACTION READ ONLY; utasítással közöljük az adatbázis-kezelővel.
Piszkos adatok olvasása
Piszkos adat: a még nem véglegesített tranzakció által módosított adat. Használata párhuzamosan végrehajtott tranzakciókban nem kívánt eredményt okozhat! A piszkos adat olvasását megengedő beállítás:
SET TRANSACTION READ WRITE
ISOLATION LEVEL READ UNCOMITTED;
További elkülönítési szintek
A véglegesített adatok olvasási lehetősége:
SET TRANSACTION ISOLATION LEVEL READ COMITTED;
Az ismételhető olvasást biztosító beállítás:
SET TRANSACTION ISOLATION LEVEL REPETABLE READ;
Fenti esetekben a tranzakciók alapértelmezés szerint író-olvasók, szükség esetén a fenti utasításokat még a READ ONLY kulcsszavakkal kibővíthetjük.
A sorbarendezhető végrehajtási módot is előírhatjuk:
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
(Az SQL-ben ez az alapértelmezés, nem szükséges megadni.)

Piszkos adatoknak nevezik a még nem véglegesített tranzakciók által módosított adatokat. Piszkos adatbeolvasási műveletnek nevezik azt az adatbeolvasó műveletet, amely egy piszkos adatot olvas be. A piszkos adatok beolvasásának az a veszélye, hogy az azt kiíró tranzakció lehet, hogy sikertelenül fejeződik be (abortál), és ilyenkor az illető tranzakció által kiírt piszkos adatok törlődnek az adatbázisból. Ha egy időközben visszavont piszkos adatot beolvasó tranzakció véglegesítve lesz, akkor az adatbázis tartalma egy olyan adatra épül, amelyet időközben már visszavontak. Egyes alkalmakkor a piszkos adatbeolvasása műveletek nem jelentek problémát, de van, amikor problémákat okozhatnak. Az adatbázis-kezelő rendszernek általában sok időt és munkát okoz a piszkos adatok beolvasásának megakadályozása, ezért minden alkalmazásnál egyedi mérlegelést igényel annak eldöntése, hogy az adatbázis-kezelőnek meg kell-e akadályoznia ezt.

A **konkurrens műveletek** engedélyezésével számos probléma felmerül a műveletek végrehajtása során. Konfliktus helyzet áll elő, ha például két jelentkező is igényt tart egyidejűleg ugyanarra az erőforrásra. E konfliktus kezelésének legismertebb formája, hogy a két igénylő közül az egyiket várakoztatjuk. Ilyen várakozósorok alakulhatnak ki például a nyomtatók használatakor is, melyben a kinyomtatandó állományok várakoznak a sorra kerülésükre. A várakozósorok kezelésekor a várakoztatások mechanizmusa, algoritmusa jelent lényeges elemet az osztott erőforrások működésében.

Zárak (lock):Az előbbi probléma egy megoldási lehetősége a zárak alkalmazása. A zár adategységhez kötődő fogalom; az adategységen a zárat "valakinek" el kell helyezni. (Adategység lehet a DB egy sora, egy egész tábla vagy csupán egy mező is. Méretének megválasztása érinti a tranzakciós teljesítményt és az abortok számát is.) Amíg T1 zárat tart A-n, addig más T-k csak korlátozottan (esetleg egyáltalán nem) férhetnek A-hoz, várakozniuk kell a zár felszabadulásáig. A tranzakció lépései között:

- a zár képzése a LOCK A,
- a zár elengedése, feloldása az UNLOCK A utasítással történik.

Ütemező (scheduler): A DBMS-ben a tranzakciókezelés problémáival az ütemező foglalkozik. Ez a DBMS azon része, amely az adatelérési igényeket kezeli adott jogosultságok szerint. Fő tevékenységei:

- váratja / továbbengedi T-t;
- zárakat ad ki / old fel;

- abortálja T-t.

Protokoll:Szabályrendszer, melyet ha a résztvevő T-k betartanak, akkor bizonyos kedvező jelenségek érhetők, illetve bajok kerülhetők el.

Holtpont (deadlock):T-k egy csoportjából senki sem tud továbblépni, mert mindegyik egy olyan zárra vár, amit egy másik T tart. A holtpont tehát a zárak tartásának egy lényeges problémája. A holtpontok felismerése várakozási gráffal lehetséges, ami egy dinamikus – időben változó – objektum.

Holtponti helyzetek megoldása:

- Várakozási gráf figyelése, és szükség szerint abort. Az ütemező az esetleges holtpontból egy vagy több tranzakciót kiiktat, ezáltal a többi T futhat tovább.
- Rendezés (<) az adategységeken. Szabály: T a zárakat a < rendezés szerint növekvő sorrendben kéri. A módszer árnyoldala, hogy interaktív rendszerekben ezt nem mindig tudják teljesíteni.
- A T tranzakció egyszerre kéri az összes szükséges zárat. Az adatokon végzendő munkát T csak ezután kezdi. Az előbbi módszernél felmerült gondok lépnek itt is fel.

Éhezés (livelock):T állandóan vár, mert mindig más kapja meg a neki kellő zárat. A megoldás ötle te: a várakozókat sorba (queue, FIFO) kell tenni.

Ütemezések: Az (egyetlen) ütemezőhöz aktív T-k utasításai érkeznek.

Soros ütemezés: Először Ti1, Ti2, ..., és végül Tik utasításai hajtódnak végre.
Ebben a rendszerben nincs keveredés a T-k utasításai között, de ez a módszer alacsony tranzakciós teljesítményt ad.

Sorosítható ütemezés: Olyan ütemezés, mely hatását tekintve ekvivalens egy soros ütemezéssel. Vagyis a sorosítható ütemezés a T1,...,Tk tranzakciók utasításainak egy olyan sorozatba rendezése, amelyben minden Ti teljesen lefut, és hatása minden adaton megegyezik egy Ti1,...,Tik alakú soros ütemezés végrehajtásával. Célunk az elkövetkezendőkben, hogy biztosítsuk a sorosíthatóságot. Ez nehezebb feladat, mint a holtpontmentesítés, és függ az alkalmazott tranzakciós modelltől is.

Időbélyegzés: Minden tranzakcióhoz hozzárendelünk egy "időbélyegzőt", a tranzakciók időbélyegzőiben minden adatbázis elem utolsó olvasását és írását rögzítjük, összehasonlítjuk ezeket az értéke ket, hogy biztosítsuk, hogy a tranzakciók időbélyegzőinek megfelelő soros ütemezés ekvivalens le gyen a tranzakciók aktuális ütemezésével.

Tranzakció kezelés tipikus feladatai:

- Az ütemezőn keresztül biztosítani tranzakciók korrekt párhuzamos működését.
- Naplózással biztosítani az adatbázisban végrehajtott tevékenységek hatásainak helyreállíthatóságát.

6. Hálós adatmodell

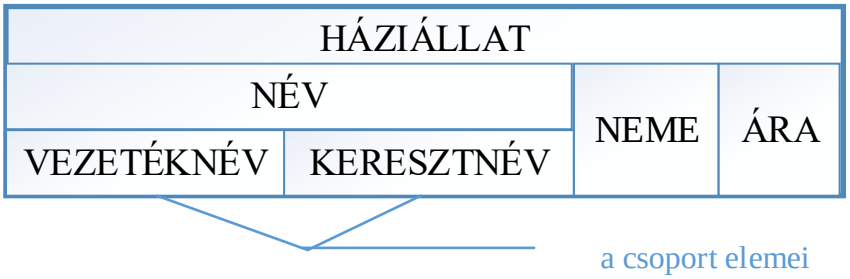
Magát a modellt, illetve a hozzá kapcsolódó adatbázis-kezelő nyelvet 1971-ben ismertette a **CODASYL DBTG** *(Conference on Data Systems and Languages Data Base Task Group).* A hálós struktúrában bármelyik csomópontot egy másik csomóponttal össze lehet kapcsolni. Lehet, de nincs értelme szinteket megkülönböztetni.

Alapfogalmak:

- rekordtípus: név és adatelemek
- összetett attribútum: csoport (group)
- többértékű attribútum: vektormező
- összetett többértékű attribútum: ismétlődő csoport (repeating group)
- halmaz (SET)

6.1. Rekordtípus

Összetett attribútum

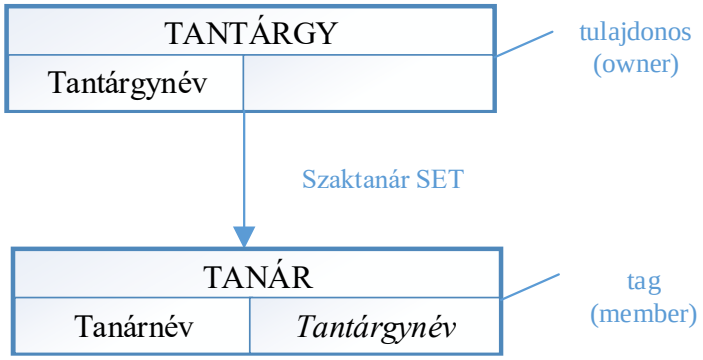


6.2. A halmaztípus (SET típus)

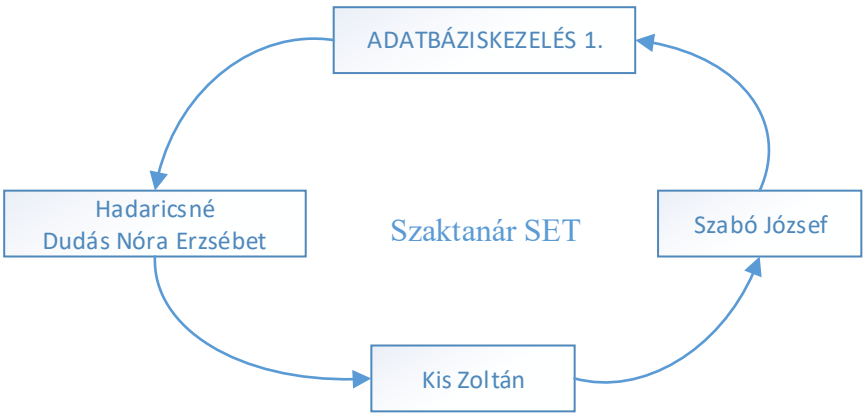
A modellben, két rekordtípus között fennálló kapcsolatot a halmaztípus írja le. A Bachman-diagrammon a SET típust nyíllal jelöljük.
A diagrammnak három alapeleme van:

- a SET típus azonosítója (neve),
- egy tulajdonos (owner) rekordtípus és
- egy tag (member) rekordtípus.

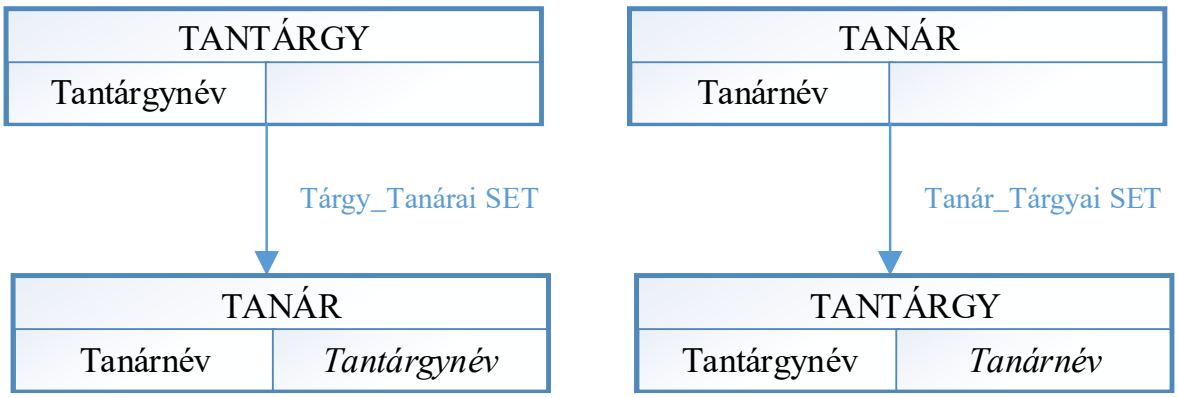
Tantárgy : Tanár, 1:N, egy rekord : több rekord



Egy adatbázisban az ilyen, tulajdonosokból és tagokból álló halmaznak annyi előfordulása van, amennyi a tulajdonosik száma. Minden előfordulásnak kell, hogy legyen egy és csakis egy tulajdonosrekordja, de a tagrekord megléte nem kötelező.



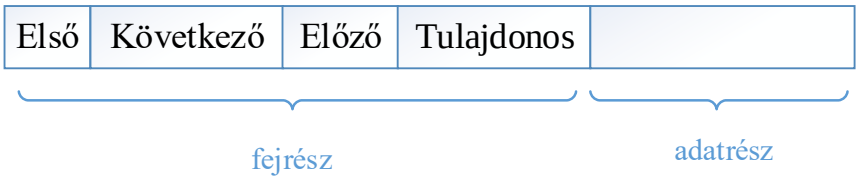
A hálós adatmodellben a sok-sok kapcsolatok is kezelhetők, úgy hogy azokat két egy-sok kapcsolatra bontják fel.



Minden egyes rekord előfordulásnál az adatok mellett egy fejrészt is képez az adatbáziskezelő rendszer. Ebben a fejrészben történik a kapcsolatok megadása. A fejrészben különböző mutatók lehetnek. A leggyakrabban használt mutatófajták az alábbiak:

- Első (first) – tulajdonostól mutat az első tagra
- Következő (next) – tagtól mutat a következő tagra
- Előző (prior) – tagtól mutat az előző tagra
- Tulajdonos (owner) – tagtól mutat a tulajdonosra

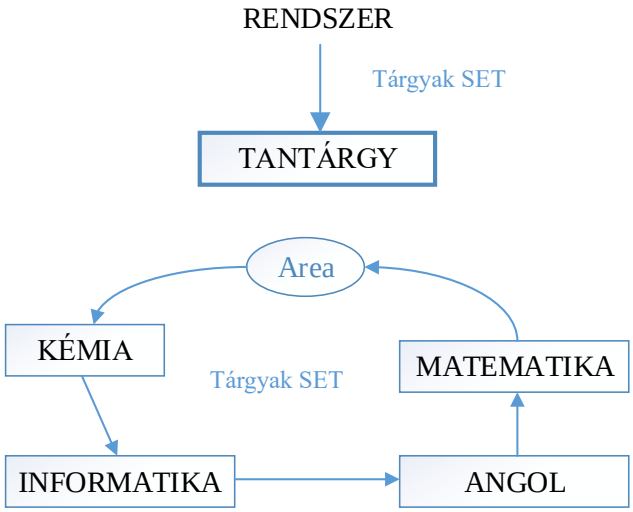
A következő ábra egy rekord szerkezetét mutatja be a hálós modellben:



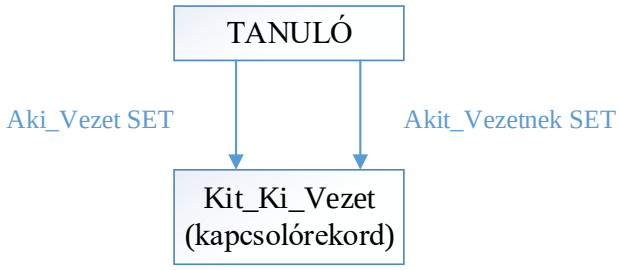
Speciális halmazok:

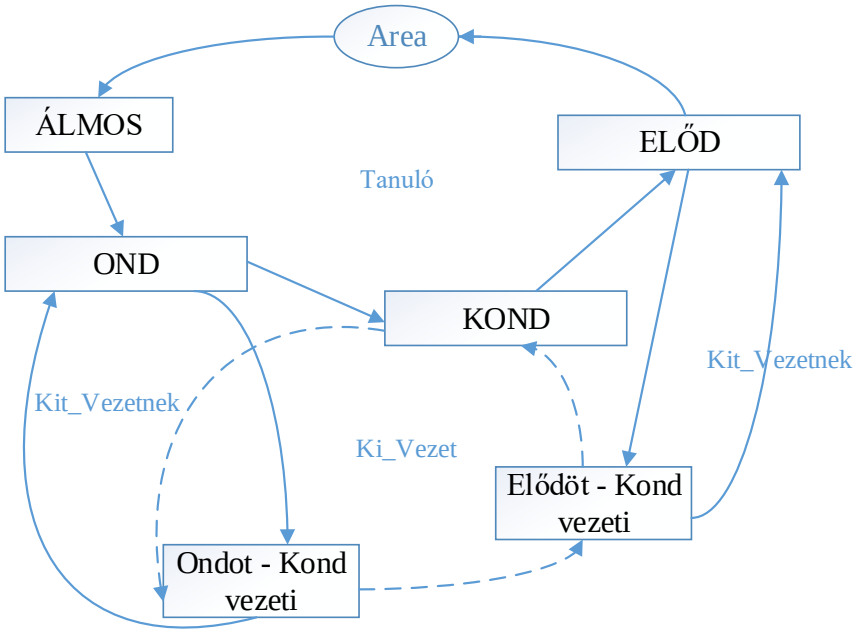
- rendszer tulajdonos halmaz: a tulajdonos ismeretlen, csak egy előfordulása van mert csak egy tulajdonos rekordja (a rendszer) van.
- rekurzív halmaz

Rendszer tulajdonos halmaz



Rekurzív halmaz



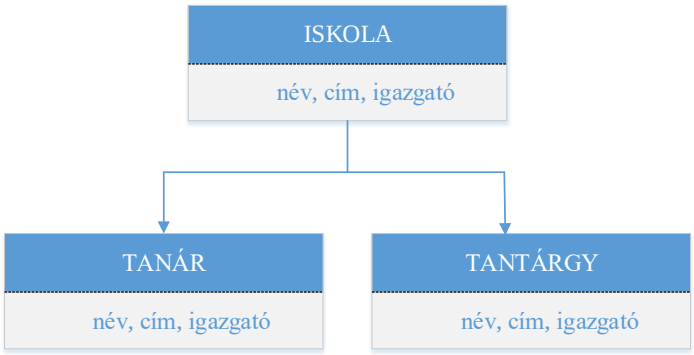


7. Hierarchikus adatmodell

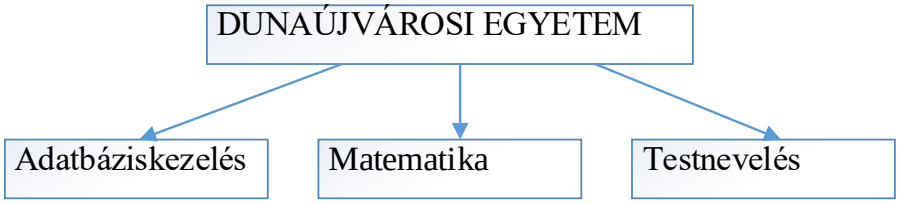
A hierarchikus modell közvetlen és természetes módon ábrázolja a hierarchikus szerkezetek egymás alá és fölérendeltségi viszonyát.

Alapfogalmak:

- Rekordtípus:
- Szülő-gyerek kapcsolat (PCR): 1:N kapcsolat, NINCS NEVE



Előfordulási ábra:



Szabályok:

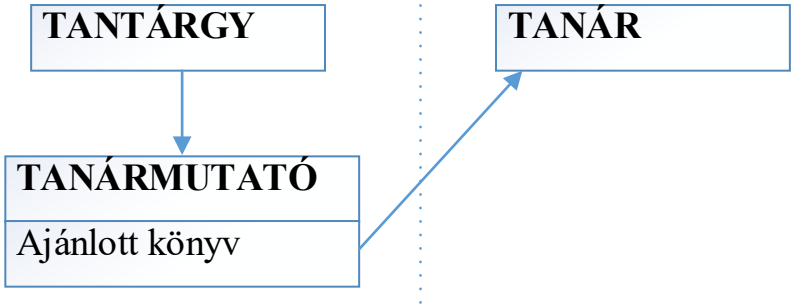
- Egyedül a gyökér rekordtípus nem vehet részt gyerekként PCR típusban.
- Minden rekordtípus (kivéve a gyökér) gyerekként pontosan egy PCR típusban vehet részt.
- Bármelyik rekordtípus szülőként tetszőleges számú (nulla vagy több) PCR típusban részt vehet.
- Ha egy rekordtípus szülőként több PCR típusban is részt vesz, akkor a gyerekei (a gyerek rekordtípusok) balról jobbra rendezettek.

Virtuális szülő gyerek kapcsolat (VPRC)

A probléma:

- N:M kapcsolat.
- Egy rekordtípus több mint egy PCR típusban szerepel gyerekként.

A virtuális (vagy mutató) rekordtípus egy olyan rekordtípus, amelynek valamennyi rekordja tartalmaz egy mutatót (pointert), ami egy másik szinten lévő rekordtípusra mutat. A mutatót tartalmazó rekordtípust virtuális gyereknek hívjuk, amelyre mutat azt pedig virtuális szülőnek.

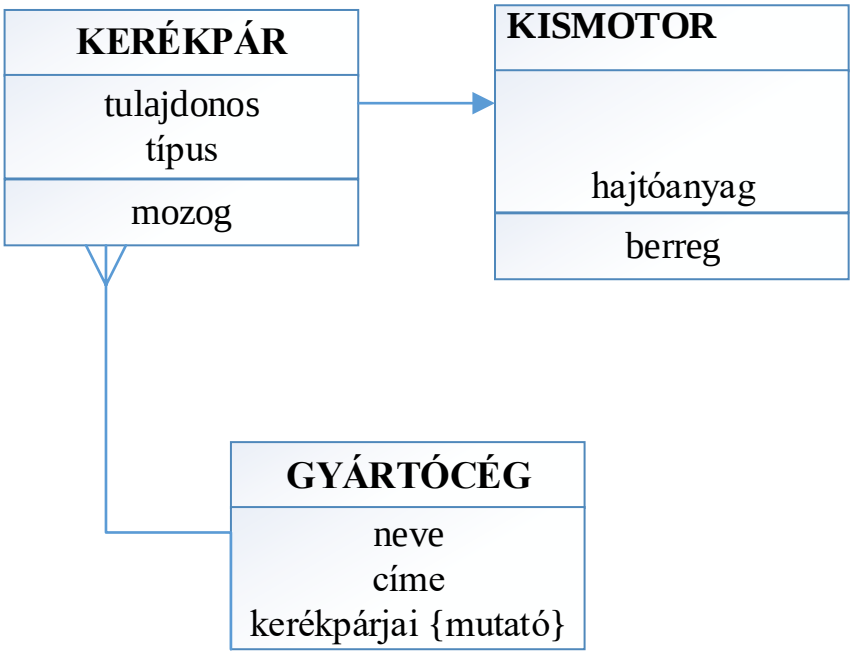


8. Objektum orientált adatmodell

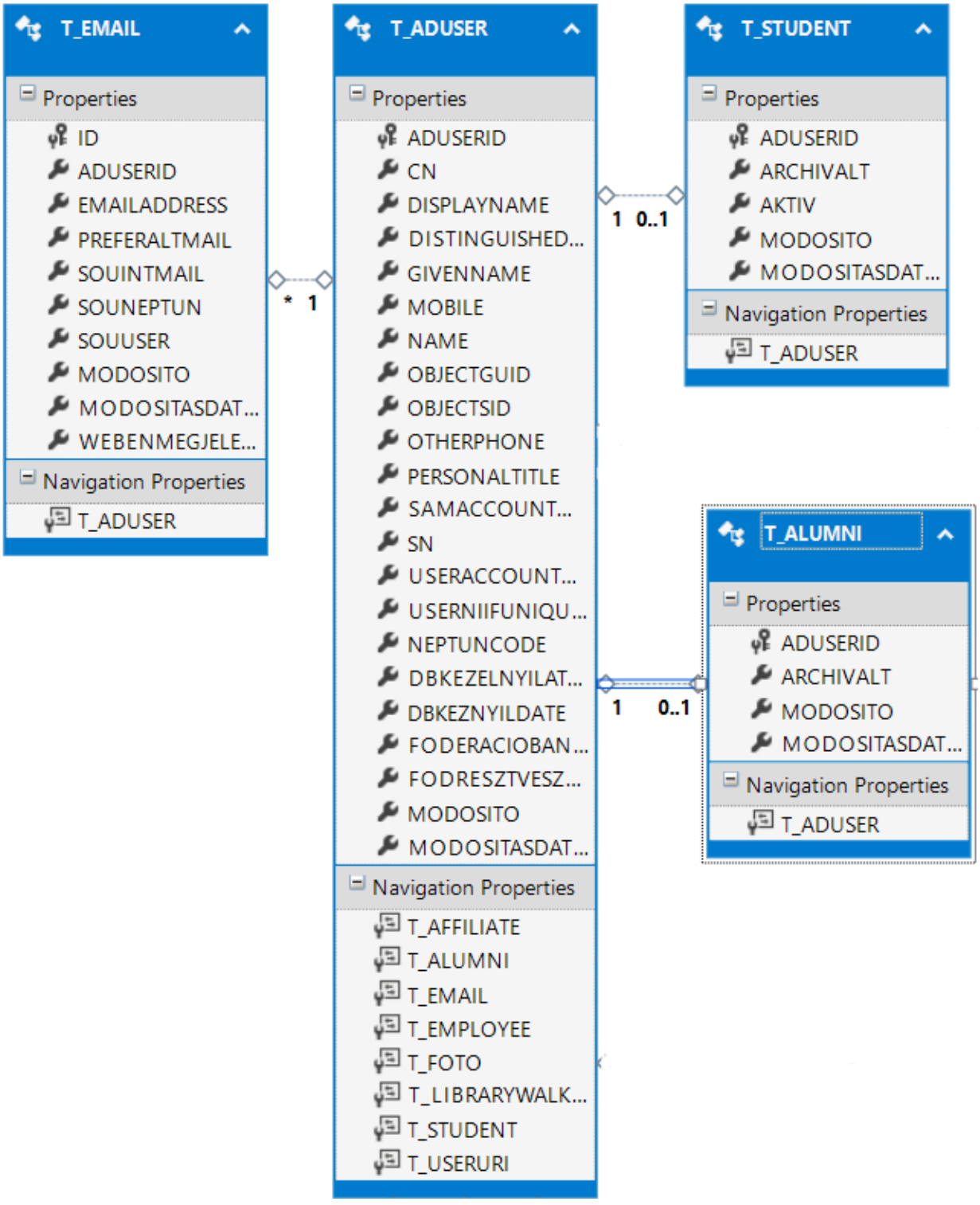
Az objektum olyan egyed, amelynek nem csak tulajdonsága, hanem viselkedése is van.

8.1. Fogalmak:

- Osztály, objektum, egységbe zárás
- Öröklés, hierarchia
- Többalakúság (polimorfizmus): ugyanazt a műveletet a hierarchia különböző szintjén más-más konkrét tartalommal végezhető el. pl.: kör vagy négyzet rajzolása



8.2. OOP - Példa


T_EMAIL
Properties
ID
ADUSERID
EMAILADDRESS
PREFERALTMAIL
SOUINTMAIL
SOUNEPTUN
SOUUSER
MODOSITO
MODOSITASDAT...
WEBENMEGJELE...
Navigation Properties
T_ADUSER
* 1
T_ADUSER
Properties
ADUSERID
CN
DISPLAYNAME
DISTINGUISHED...
GIVENNAME
MOBILE
NAME
OBJECTGUID
OBJECTSID
OTHERPHONE
PERSONALTITLE
SAMACCOUNT...
SN
USERACCOUNT...
USERNIIFUNIQU...
NEPTUNCODE
DBKEZELNYILAT...
DBKEZNYILDATE
FODERACIOBAN...
FODRESZTVESZ...
MODOSITO
MODOSITASDAT...
Navigation Properties
T_AFFILIATE
T_ALUMNI
T_EMAIL
T_EMPLOYEE
T_FOTO
T_LIBRARYWALK...
T_STUDENT
T_USERURI
1 0..1
T_STUDENT
Properties
ADUSERID
ARCHIVALT
AKTIV
MODOSITO
MODOSITASDAT...
Navigation Properties
T_ADUSER
1 0..1
T_ALUMNI
Properties
ADUSERID
ARCHIVALT
MODOSITO
MODOSITASDAT...
Navigation Properties
T_ADUSER

```
public partial class T_ADUSER
    {
        public long ADUSERID { get; set; }
        public string CN { get; set; }
        public string DISPLAYNAME { get; set; }
        public string DISTINGUISHEDNAME { get; set; }
        public string GIVENNAME { get; set; }
        public string MOBILE { get; set; }
        public string NAME { get; set; }
        public string OBJECTGUID { get; set; }
        public string OBJECTSID { get; set; }
        public string OTHERPHONE { get; set; }
        public string PERSONALTITLE { get; set; }
        public string SAMACCOUNTNAME { get; set; }
        public string SN { get; set; }
        public string USERACCOUNTCONTROL { get; set; }
        public string USERNIIFUNIQUEID { get; set; }
        public string NEPTUNCODE { get; set; }
        public string DBKEZELNYILATKELFOGAD { get; set; }
        public Nullable<System.DateTime> DBKEZNYILDATE { get; set; }
        public string FODERACIOBANRESZTVESZ { get; set; }
        public Nullable<System.DateTime> FODRESZTVESZDATE { get; set; }
        public long MODOSITO { get; set; }
        public System.DateTime MODOSITASDATUMA { get; set; }

        public virtual T_AFFILIATE T_AFFILIATE { get; set; }
        public virtual T_ALUMNI T_ALUMNI { get; set; }
        public virtual ICollection<T_EMAIL> T_EMAIL { get; set; }
        public virtual T_EMPLOYEE T_EMPLOYEE { get; set; }
        public virtual ICollection<T_FOTO> T_FOTO { get; set; }
        public virtual T_LIBRARYWALKIN T_LIBRARYWALKIN { get; set; }
        public virtual T_STUDENT T_STUDENT { get; set; }
        public virtual ICollection<T_USERURI> T_USERURI { get; set; }
   }

public partial class T_STUDENT
    {
        public long ADUSERID { get; set; }
        public string ARCHIVALT { get; set; }
        public string AKTIV { get; set; }
        public long MODOSITO { get; set; }
        public System.DateTime MODOSITASDATUMA { get; set; }

        public virtual T_ADUSER T_ADUSER { get; set; }

    }
public partial class T_EMAIL
    {
        public long ID { get; set; }
        public long ADUSERID { get; set; }
        public string EMAILADDRESS { get; set; }
        public string PREFERALTMAIL { get; set; }
        public string SOUINTMAIL { get; set; }
        public string SOUNEPTUN { get; set; }
        public string SOUUSER { get; set; }
        public long MODOSITO { get; set; }
        public System.DateTime MODOSITASDATUMA { get; set; }
        public string WEBENMEGJELENIK { get; set; }

        public virtual T_ADUSER T_ADUSER { get; set; }

    }
```