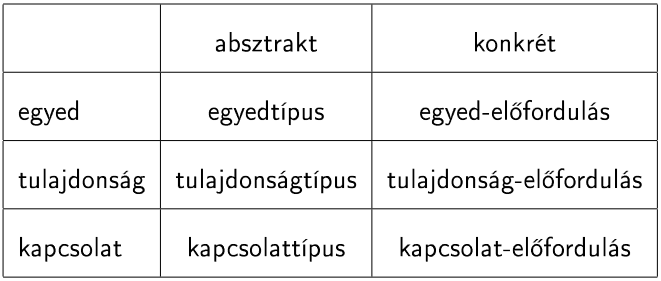
1.

Modellezés lépései:

* feladat, terület, kisvilág meghatározása
* elvárások lefektetése, mit tudjon a rendszer
* lehetséges adatok számbavétele
* elemzés és tervezés
* koncepcionális adatbázis készítése, kapcsolattípusok meghatározása
* műveletek/ tranzakciók definiálása
* logikai tervezés DBMS nyelvén
* konkrét logika séma kialakítása
* logikai séma lefordítása fizikai sémává

Bachmann fogalomrendszer:



Tulajdonságtípusok osztályozása:

* tulajdonság-előfordulás szerkezete szerint, lehet ***egyszerű/atomi*** v. ***összetett***
* tulajdonság-előfordulás hány értéket vehet fel ***egyértékű*** v. ***halmazértékű/többértékű***
* tulajdonság-előfordulás minden esetben megjelenik-e a háttértáron, ***tárolt*** v. ***származtatott*** lehet

NULL érték:

* nem alkalmazható, nem értelmezett
* ismeretlen (létezik, de hiányzik/ nem tudjuk, hogy létezik-e)

Kapcsolattípusok osztályozása:

* a kapcsolat foka: meghatározza hány egyedtípus vesz részt a kapcsolatban
* bináris kapcsolat ***számossága***: legfeljebb hány kapcsolat-előfordulásban vehet részt egy egyed-előfordulásban. *1:1, 1:N, N:M*
* kapcsolat ***szorossága***: meghatározza, hogy minden egyednek részt kell-e vennie egy kapcsolat-előfordulásban: kötelező, félig kötelező, opcionális

2.

DBMS feladatok:

* konkrét db ***definiálása***
* kezdeti db tartalmának ***betöltése***
* db kezelése, kinyerés, módosítás, elérés
* feldolgozás és megosztás konkurens felhasználók közt úgy, hogy az összes adat konzisztens és érvényes maradjon

továbbá:

* védelmi biztonsági szolgáltatások
* aktív feldolgozás az adatokon való belső műveletek végrehajtására
* adatok vizualizációja
* adatbázis és a kapcsolódó programok karbantartása

Adatbázisrendszer önleíró természete:

* DBMS katalógus tárolja egy önálló adatbázis leírását
* ez a leírás metaadatokból áll
* lehetővé teszi, hogy a DBMS különböző DB alkalmazásokkal működjön együtt

Programok és adatok elszigetelése:

* program és adat közti függetlenség
* lehetővé teszi az adatszerkezetek és a tárolás módjának változtatását anélkül, hogy a DBMS programot változtatni kellene

Adat absztrakció:

* adatmodellt használunk arra, hogy csak az adatbázis koncepcionális részét jelenítsük meg a felhasználó felé
* a programok az adatmodellre hivatkoznak az adattárolási részletekkel szemben

Adatok nézeteinek támogatása:

* minden felhasználó lehetőséget kap, hogy csak a számára releváns adatot láthassa

Adatok megosztása, többfelhasználós tranzakció feldolgozás:

* megengedi a konkurens felhasználóknak az adatkinyerést és frissítést ugyanazon db-ben
* a konkurencia ellenőrzés DBMS-en belül garantálja a tranzakciók helyes végrehajtását
* helyreállító alrendszer biztosítja, hogy minden végrehajtott tranzakció állandó bejegyzésre kerüljön
* konkurens tranzakciók sokaságának végrehajtásáról a közvetlen tranzakció feldolgozás gondoskodik (OLTP – Online Transaction Processing)

Adatmodellek:

* fogalmak összessége, ami leírja a DB szerkezetét és műveleteket melyekkel manipulálható
* az adatbázis szerkezetét konstruktorokkal definiáljuk
* jellemző konstruktorok az elemek, azok adattípusai, elemek csoportjai és a csoportok közti kapcsolatok
* megszorításokkal korlátozzuk az adatok érvényességét
* adatmodell műveletei lehetnek alapműveletekből vagy felhasználó által definiált

Adatmodellek fajtái:

* Koncepcionális (magas szintű, szemantikus): felhasználó számára könnyen érthető fogalmakkal dolgozik
* Fizikai (alacsony szintű, belsős): adatok eltárolódását leíró fogalmakkal dolgozik
* Implementációs (reprezentációs): koncepcionális és fizikai adatmodell közt helyezhető el

Sémák és előfordulások:

* adatbázis séma: az adatbázis leírása
* séma diagram: az adatbázis séma szemléltető megjelenítése
* séma konstruktor: séma vagy sémán belüli objektum komponense

Adatbázis állapota:

* adott időpillanatban tárolt aktuális adatok
* nevezik előfordulásnak is (ezt egyedi komponensekre is használjuk)
* kezdeti adatbázis állapot: evidens
* érvényes állapot: olyan állapot, ami megfelel a megszorításoknak és a szerkezetnek

különbség: a SÉMA ritkán változik az ÁLLAPOT viszont minden frissítéssel

Adatbázisrendszer:

* számítógép
* adatok
  + fizikai db
  + metaadatbázis
* szoftver
* felhasználók
  + eseti
  + naiv/ parametrikus
  + szakértő
  + DBA (admin)

Felhasználók: (annyira nem lényeg imo.)

* 2 csoport, tényleges felhasználók, és akik ezen eszközöket fejlesztik, üzemeltetik
* tényleges felhasználók:
  + DBA: adatbázis ellenőrzése/ használatának koordinálása/ monitorozás
  + Adatbázis tervezők: tartalom, szerkezet megszorítások definiálása
  + végfelhasználók: lekérdezések, riportok készítése/ frissítés
    - eseti: akkor éri el a db-t, amikor szükséges
    - naiv: jól definiált függvényeket használnak
    - szofisztikált felhasználók: tárolt adatbázishoz közel álló toolokat használnak
    - önálló: személyes adatbázis, készen csomagolt alkalmazásokkal

Háromséma-architektúra:

* belső séma: szerkezet fizikai tárolásának leírása
* koncepcionális séma: koncepcionálisan a teljes leírás (szerkezet, megszorítások..)
* külső séma: felhasználói nézetek leírása

a programok külső sémára hivatkoznak és DBMS által leképeződnek belső sémára majd végrehajtódnak, a belső DBMS szintből kinyert adatok újraformázódnak hogy a külső nézethez igazodjanak

Adatfüggetlenség:

* logikai: koncepcionális séma változása, a külső séma változása nélkül
* fizikai: a belső séma változása a koncepcionális változása nélkül

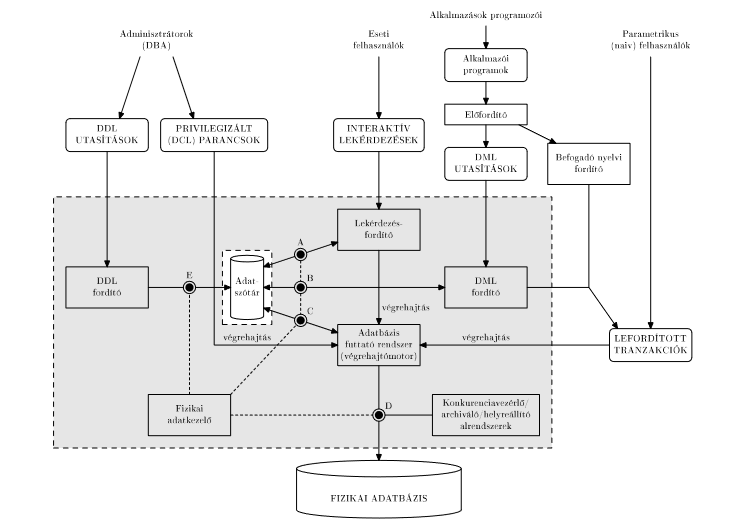
DBMS nyelvek:

* DDL – data definition language
  + DBA használja a koncepcionális séma kialakításához
  + sok esetben belső és külső séma definiálásra is használatos
  + egyes esetekben elkülönített a tárolásleíró (SDL) és a nézetleíró (VDL) nyelv
* DML – data management language
  + keresés és frissítés specifikációja
  + általános célú programozási nyelvekbe beágyazható

magas szintű nyelvek programozási nyelvbe ágyazás nélkül is használhatóak, alacsony szintűeket be kell ágyazni

DBMS interfészek:

* felhasználóbarát: menü alapú/ form alapú/ grafikus/ természetes nyelvi/ ezek kombinációi
* további: beszéd alapú/ web alapú/ parametrikus/ DBA interfészek



Utility-k

* ezen funkciók támogatása: fájlokban tárolt adat betöltése db-be/ db periodikus mentése/ fájl-szerkezet újraszervezése/ riport generálás/ hatékonyság monitorozás/ **alapvető információk tárolása basically**

DBMS architektúrák:

* centralizált DBMS: mindent egy rendszerbe egyesít, távoli elérés lehetséges, de a feldolgozás központosított
* két rétegű kliens szerver architektúra: célfeladatokra dedikált szerverekből és kliensekből áll, a kliensek szükség szerint érhetik el a specializált szervereket
* három rétegű kliens szerver architektúra: korábbi két réteg egy közbenső réteggel egészül ki amely az alkalmazás szerver (web szerver)

DBMS szerver: \*

* lekérdezési és tranzakciós szolgáltatásokat nyújt
* a szerver elérésére a kliensek API-t használnak
* kliens és szerver oldalon egyaránt telepítve kell, hogy legyen minden megfelelő modul

DBMS kliens: \*

* szerverszolgáltatások elérésére ad interfészt
* hálózatokon keresztük kapcsolódik a szerverhez

Alkalmazás szerverek: \*

* webhez való kapcsolódás és megfelelő adatokhoz való hozzáférés biztosítása
* részlegesen feldolgozott adatokat küld a db szerver és a kliensek közt
* a kliensek CSAK közbenső réteggel érhetik el a szervert

32-35. dia történelmi stuff nem lényeg imo

DBMS elhanyagolhatósága:

* fő korlát a magas költség és hardverigény
* szükségtelen lehet a DBMS ha:
  + a db és az alkalmazások egyszerűek, várhatóan nem változnak
  + nem fér bele a követelményekbe a magasabb válaszidő
  + nem szükséges a konkurens adathozzáférés
  + olyan speciális műveletek használata, amit a DBMS nem támogat
  + túl összetettek az adatok a modellezési korlátokhoz mérten

3.

Relációs modell:

* előnyei: egyszerűség, matematikai alap
* alapja a matematikai reláció fogalma, halmazelmélet, elsőrendű predikátumkalkulus

Reláció: értékek táblázata, sorok halmaza

Sor (rekord): egyed-előfordulásról vagy kapcsolat-előfordulásról tartalmaz adatokat

Oszlop: fejléce leírja, miről tartalmaz információt az adott oszlop

Reláció kulcsa: olyan adatelem, ami egyértelműen azonosítja a sort

Atomi érték: relációs adatmodell szempontjából nem tovább bontható érték

Tartomány: egy D tartomány atomi értékek halmaza

* név/ adattípus/formátum/korlátozás/ információk az értelmezéshez

r az R relációséma egy konkrét tartalma (intension)

R az r reláció kiterjesztése (extension)

Heurisztika: R egy táblázat üres váza, r az adatokkal feltöltött táblázat

Rendezés:

* a rekordokat nem tekintjük rendezettnek, a fizikai adattárolón viszont rendezve vannak
* sémán belül az értékek az attribútumok sorrendjének megfelelően helyezkednek el

Reláció:

* adott állapotban csak azokat a rekordokat tartalmazza, ami a valós világ pillanatnyi állapotát tükrözi
* a reláció sémája általában statikus

Relációs modell megszorításai:

* adatmodellben rejlő: modellalapú vagy implicit megszorítások
* adatmodell sémáiban: sémaalapú vagy explicit megszorítások
  + sémaalapú: tartománymegszorítás, kulcsmegszorítás, NULL érték megszorításai, egyedintegritás, hivatkozási integritás
    - *tartománymegszorítás*: minden rekordban minden attribútumhoz tartozó érték az adott tartományból származik vagy NULL érték, ezen tartomány minden elemének atomi értéknek kell lenni
    - *egyedintegritási megszorítás*: elsődlegeskulcs-érték nem lehet NULL, ha összetett egyik komponens se lehet NULL
    - *hivatkozás integritási megszorítások:* két reláció közti konzisztencia megteremtése
* alkalmazói programokkal kifejezett megszorítások: alkalmazásalapú vagy szemantikus megszorítások

Tartományokra jellemző adattípusok: numerikus/ karakter/ logikai/ sztring/ dátum/ egyéb speciális adattípusok

Kulcsjelölt: több kulcs esetén a kulcsok mindegyike kulcsjelölt

Elsődleges kulcs: a kulcsjelöltek közül választja ki a modellező, ennek értékeivel azonosítjuk a rekordokat. A relációsémának mindig kell elsődleges kulcs

4.

Absztrakt lekérdezőnyelvek:

* relációalgebra: egy eljárást adunk meg a kinyerni kívánt információ előállítására
* relációkalkulus: deklaratív kifejezéssel adjuk meg a kinyerni kívánt információt
  + rekordalapú
  + tartományalapú

Relációalgebra műveletek: kifejezései azt írják le, hogyan kapjuk meg az eredményt

\*alpafogalmak jegyzet\* +

* szelekció
  + unáris művelet / eredményül kapott reláció foka és sémája megegyezik / eredmény számossága mindig kisebb R számosságánál / két egymásba ágyazott szelekciós művelet végrehajtási sorrendje felcserélhető / kaszkádolt szelekció átírható egybe konjunkciós formában
* projekció
  + unáris / fokát és szintjét az attribútumlistában szereplő attribútumok határozzák meg / ha az attribútumlista nem tartalmaz kulcs attribútumot az eredmény számossága kisebb lehet mint R, ha az attribútumlista R szuperkulcsa akkor az eredmény számossága megegyezik R-ével / 2 egymásba ágyazott projekciós művelet eredménye megegyezik a külső projekció eredményével
* átnevezés
  + unáris / foka, számossága megegyezik R fokával, számosságával / eredmény sémáját meghatározhatjuk
* halmazműveletek
* összekapcsolás
  + általános összekapcsolás: Theta join:AND A attribútum és B attribútum között (=, ̸=, , ≤, ≥) jelek vannak és tartományuk megegyezik.
  + természetes összekapcsolás: equijoin alapja, fölösleges attribútum(ok) kihagyásával / sémája a 2 reláció sémájának attribútumait tartalmazza, összekapcsolási alap páronként egyszer/ foka a két reláció fokszámának összegétől annyival kevesebb ahány azonos nevű attribútum van / számossága 0-tól az eredeti relációk számosságának szorzatáig tarthat
* hányados

Relációkalkulus: kifejezései azt írják le, hogy mit szeretnék eredményként kapni (dávid szerint pretty much skippelhető)

* rekordalapú, sorokkal operál / tartományalapú oszlopokkal
* random bullshit és tengernyi példa 4. előadás :))

Biztonságos kifejezések:

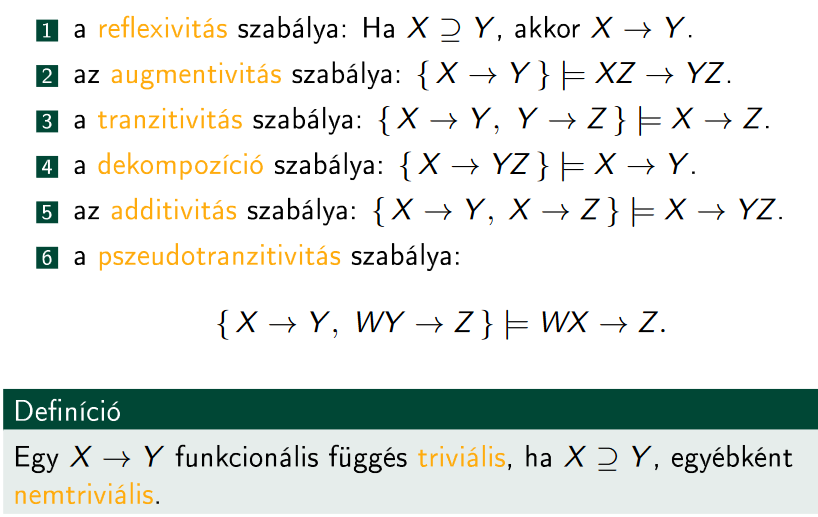
* egy kifejezés biztonságos, ha az eredményben szereplő összes érték a kifejezés tartományából való

5.

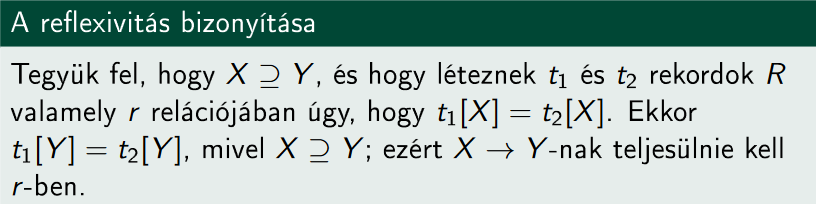
Funkcionális függés:

* magyarázat: ha a szemantika kimondja, hogy két halmaz közt funkcionális függés van, ezt megszorításként kell specifikálni
* a funkcionális függésnek eleget tevő relációsémákat legális kiterjesztésnek nevezzük
* néha automatikusan teljesül

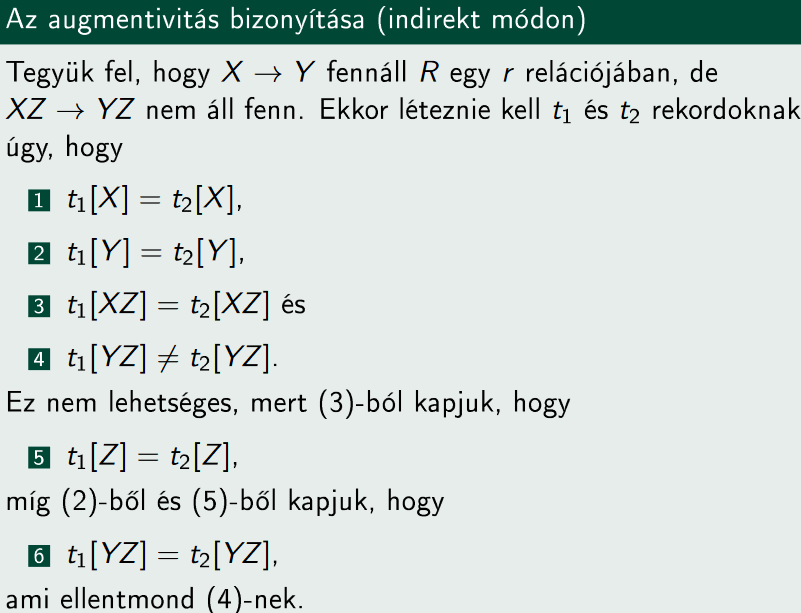
Tulajdonságai:



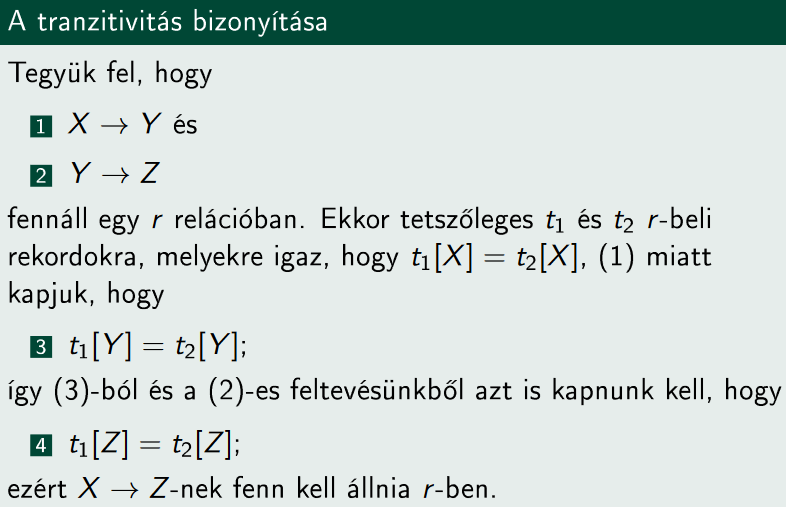
* reflexivitás: egy attribútumhalmaz mindig meghatározza önmagát vagy bármely részhalmazát



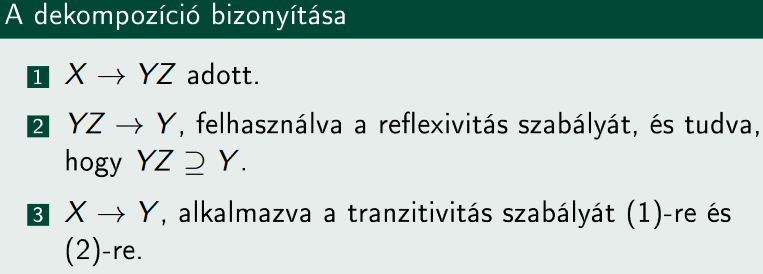
* augmentivitás: funkcionális függés mindkét oldalának bővítése újabb érvényes funkc.függést eredményez



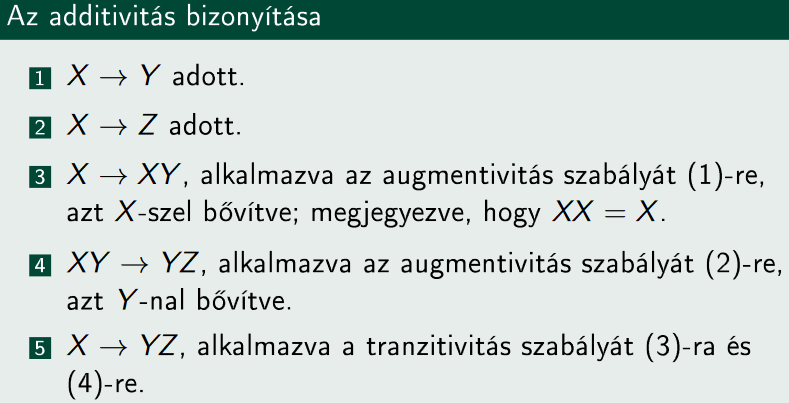
* tranzitivitás: a funkc. függések tranzitívak

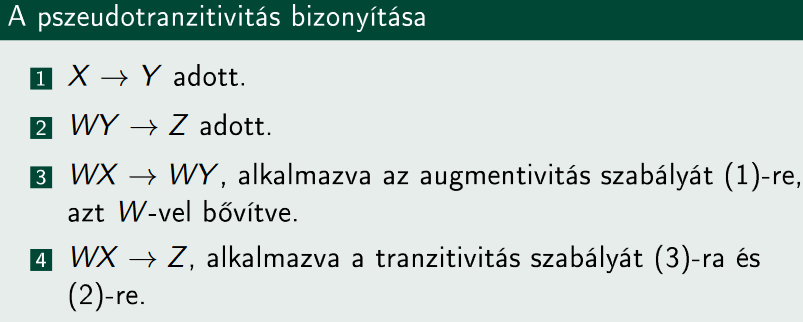


* dekompozíció: a funkc. függés jobb oldaláról eltávolíthatunk attribútumokat



* additivitás: X 🡪A1, X🡪A2 halmazok összevonhatóak X 🡪 {A1,A2..} funkc. függésség





Armstrong axiómák:

* helyesség: adott R relációsémán fennálló funkcionális függések egy F halmaza. bármely függés mely levezethtő F-ből 3 szabállyal fenn fog állni R minden r relációjában mely kielégíti F-ben a függéseket
* Teljesség: F-ből kiindulva 3 szabállyal meghatározható F+ függések halmaza, ez lesz F lezártja
* Lezártság: minden X attribútumhalmazra meghatározzuk az attribútumoknak olyan X+ halmazát, melyet X funkcionálisan meghatároz F alapján, ez az X+ lesz X F alatti lezártja
* Ekvivalencia: funkcionális függések F halmaza lefed egy másik E halmazt, minden E beli funkcionális függés benne van F+-ban ergo minden E-beli függés levezethető F-ből

E és F ekvivalens egymással, ha E+ = F+ ami annyit jelent, hogy E lefedi F-et és F lefedi E-t

(minden E-beli funkc függ levezethető F-ből és fordítva)

* minimális halmaz ha:
  + X 🡪Y funkcionális függésre F-ben Y egyszerű, azaz egy attribútumból áll
  + nem hagyható el funkcionális függés Fből, hogy F-el ekvivalens halmazt kapjunk
  + nem helyettesíthető X🡪A funkcionális függés F-ben Y🡪A függéssel ahol Y ⊂ X úgy hogy F-el ekvivalens halmazt kapjunk

funkcionális függések E halmazának minimális lefedése egy olyan halmaz mely minimális és ekvivalens E-vel

* + jellemzői: funkc függések minden halazának van vele ekvivalens minimális halmaza /

több egymással ekvivalens minimális halmaz létezhet / a minimális halmaz standard kanonikus alak, redundanciák nélkül / relációk előállításakor feltételezzük, hogy funkc függések egy minimális halmazából indulunk ki

6.

0i irányelvek:

* könnyen magyarázható, könnyen értelmezhető sémát kell tervezni

Karbantartási anomáliák:

* beszúrási anomália
* törlési anomália
* módosítási anomália

NULL érték rekordokban:

* a megtervezett relációk a lehető legkevesebb null értéket tartalmazzák
* azok az attribútumok, amik gyakran vesznek fel null értéket áthelyezhetők másik relációba
* null érték okai:
  + attribútum nem értelmezhető vagy érvénytelen
  + attribútumérték ismeretlen (de létezhet)
  + az érték létezik, de nem elérhető

Álrekordok:

* veszteségmentes összekapcsolással garantáljuk, hogy az összekapcsolási műveletek értelmes eredményt adjanak
* dekompozíciók:
  + a megfelelő összekapcsolás nemadditív vagy veszteségmentes (fontos)
  + megőrzik a funkcionális függőségeket (szükség esetén elhanyagolható)

Normalizáció:

* nem kielégítő relációsémák szétbontása kisebbekre
* normálforma: relációsmák kulcsai és bennük fennálló funkcionális függőségekkel megfogalmazott feltétel, megállapítható vele, hogy a relációséma adott NF-ban van-e
* nem szükséges a legmagasabb NF-ig normalizálni
* denormalizáció során alacsonyabb normálformában tároljuk magasabb NF-ű relációk összekapcsolását

Normálformák:

* 2NF, 3NF, BCNF: relácósémák kulcsai és bennük fennálló funkcionális függések alapján
* 4NF: kulcsok és többértékű függések
* 5NF: kulcsok és join függések

1NF elérése:

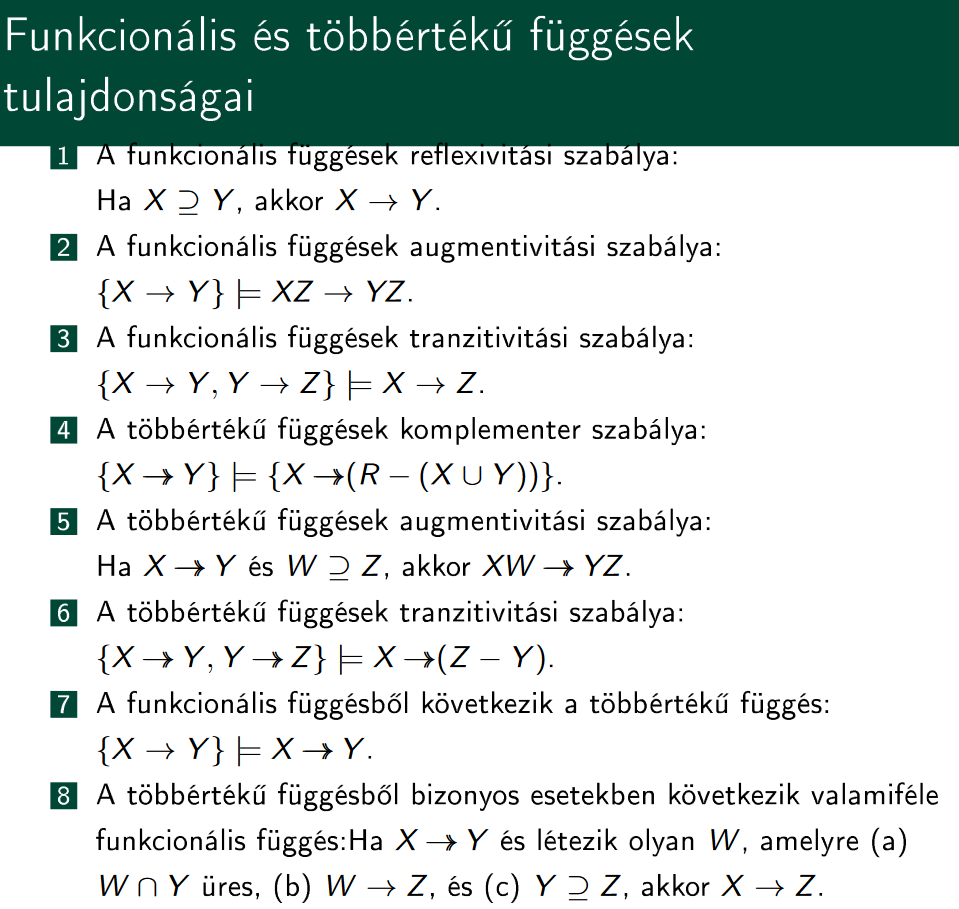
* problémás attribútumok áthelyezése másik relációba
* kulcs bővítése
* több atomi attribútum használata

2NF elérése:

* az eredeti relációból eltávolítjuk a részlegesen függő attribútumot egy másik relációba, ebben szerepelni kell az eredeti reláció elsődleges kulcsának, amelytől a másodlagos attr függ, a második reláció elsődleges kulcsa az eredeti reláció elsődleges kulcsának a része

3NF elérése:

* az eredeti relációból eltávolítjuk a tranzitívan függő másodlagos attribútumokat másik relációba, ebben elsődleges kulcsként kell szerepelnie azoknak az attribútumoknak, amelyektől a másodlagos attribútumok függnek



Boyce\_Codd NF: Egy R relációséma B-C NF-ben van ha egy X → A nemtriviális funkcionális függés fennáll R-en, akkor X egy szuperkulcsa R-nek.

4NF:

* X ->> Y többértékű függést triviális többértékű függésnek nevezzük ha:
  + Y részhalmaza X-nek
  + X ∪ Y = R
* nemtriviális többértékű függés a felső 2 feltétel egyikét se elégíti ki
* ha minden F+ beli nemtriviális X->>Y többértékű függés esetén X szuperkulcsa R-nek

5NF:

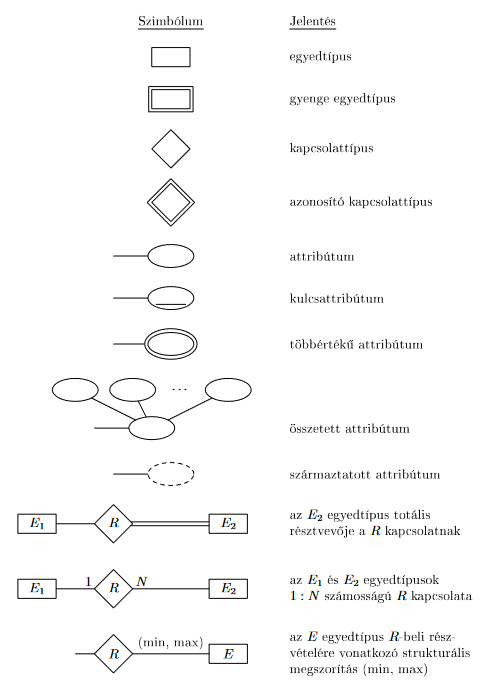
* ha F halmaz minden F+ beli nemtriviális JD(R1,R2.Rn) esetén minden Ri szuperkulcsa Rnek

Normalizáció lépései

* kezdeti univerzális relációséma
* ennek a sémának minden attribútumának a neve egyedi
* meghatározzuk az R-en fennálló funkcionális függések F halmazát
* az R sémát relációs sémák halmazára bontjuk fel
* biztosítjuk, hogy R minden attribútuma legalább egy Ri relációs sémában szerepeljen
* függőség megőrzése: minden Rben megadott X🡪Y funkcionális függés közvetlenül szerepeljen az Ri relációs sémák egyikében vagy valamely Ri-ben megjelenő függésből levezethető legyen

7.

ER modell:

* a modell kezeli egyszerű, összetett, egyértékű, halmazértékű, tárolt, származtatott tulajdonságtípusokat
* a modellben tetszőleges fokszámú kapcsolattípus ábrázolható
* 

ER séma leképzése relációs sémára:

1. **erős egyedtípusok**: az ER séma minden E erős egyedtípusához rendelünk egy R relációsémát, amely tartalmazza E összes egyszerű attribútumát, az összetett attribútumoknak csak az egyszerű komponenseit adjuk az R attribútumaihoz, kiválasztjuk E kulcsattribútumainak egyikét R kulcsául, ha a kiválasztott kulcs összetett, akkor az egyszerű attribútumai alkotják R PK-ját
2. **gyenge egyedtípusok**: az ER séma minden W gyenge egyedtípusához rendelünk egy R relációsémát melynek attribútumai W minden egyszerű komponensei és minden összetett attribútumának egyszerű komponensei, R attribútumaihoz hozzáadjuk FK attribútumként a relációsémának azon PK attribútumait amelyeket domináns egyedtípusoknak feleltettünk meg

R elsődleges kulcsa a tulajdonos egyedtípusok elsődleges kulcsainak és W gyenge egyedtípus diszkriminátorának az együttese

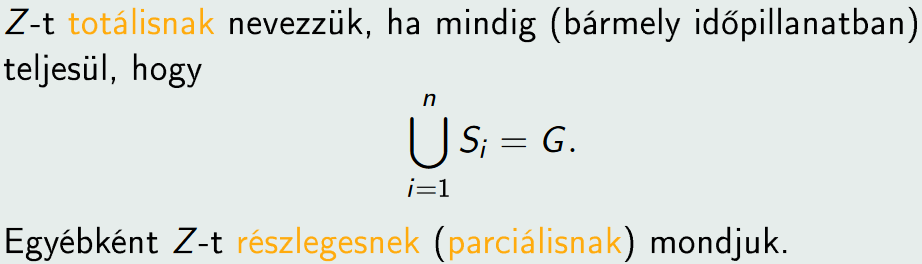
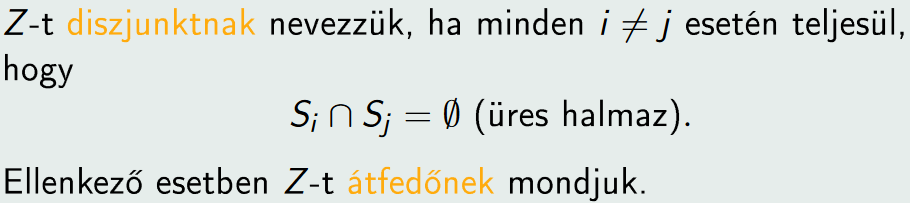
1. **bináris 1:1 kapcsolatok**
   1. *külső kulcs:* S reláció külső kulcsaként felvesszük T elsődleges kulcsát, R egyszerű attribútumait és R összetett attribútumainak egyszerű komponenseit
   2. *összevonás:* a két egyedtípust és kapcsolatot egyetlen relációba vonjuk össze (akkor lehet, ha mindkét egyedtípus totális résztvevője a kapcsolatnak)
   3. *kereszthivatkozás/ kapcsoló relációk:* felveszünk egy harmadik R relációt, hogy kereszthivatkozással lássuk el a két egyedtípusból képzett S és T relációk elsődleges kulcsait az R reláció lesz a kapcsoló reláció
2. **bináris 1:N kapcsolatok:** meghatározzuk S relációt, amit a kapcsolattípus N oldali egyedtípusából képzünk le, S külső kulcsa R-ben résztvevő másik egyedtípusból képzett T reláció elsődleges kulcsa, továbbá R egyszerű attribútumait R összetett attribútumainak egyszerű komponenseit felvesszük S attribútumaiként

**/** itt is használható kapcsoló reláció

1. **bináris M:N kapcsolatok:** minden esetben létrehozunk egy új S relációt mely R-et reprezentálja, külső kulcsa a résztvevő egyedtípusokból képzett relációk elsődleges kulcsai lesznek ezek együttese lesz S elsődleges kulcsa, továbbá felvesszük R összes attribútumának egyszerű komponenseit S attribútumaiként
2. **Többértékű attribútumok:** új R reláció ami tartalmaz egy A-nak megfelelő attribútumot valamint a annak a relációnak az elsődleges kulcsát melyet az A-t tartalmazó egyedtípusokól vagy kapcsolattípusokból képeztünk, R elsődleges kulcsát A és K együttese alkotja, ha összetett akkor az egyszerű komponenseit vegyük fel R attribútumaiként
3. **N-edfokú kapcsolattípusok:** N-edfokú R kapcsolattípus esetén (ha N>2) hozzunk létre egy S relációt mely R-et reprezentálja, S külső kulcsa lesz a kapcsolatban résztvevő egyedtípusokból képzett relációk elsődleges kulcsai, felvesszük még R egyszerű attribútumait, R összetett attribútumainak egyszerű komponenseit S attribútumaiként, S elsődleges kulcsa általában az összes külső kulcs együttese, ha valamely E adattípusból csak egy rekord vehet részt akkor S elsődleges kulcsának nem kell tartalmazni a leképzett E’ relációra hivatkozó külső kulcsot

8.

Fogalmak:

* osztály: egyedek halmaza, magában foglal minden olyan EER szerkezetet, mely egyedeket csoportsít
* alosztály: olyan osztály mely egyedeinek mindig egy másik (szuperosztály/ alosztály) kapcsolat szuperosztályához tartozó egyedek egy részhalmazát kell alkotniuk
* specializáció: olyan alosztályok halmaza melyeknek ugyanaz a szuperosztálya
* generalizált egyedtípus: (ez a szuperosztály/ alosztályok generalizációja)
* 
* 
* C-nek egy S alosztályát predikátumdefiniáltnak nevezzük, ha egy predikátumot előírva adjuk meg mely C-beli elemek elemei S-nek
* nem predikátumdefiniált osztályokat felhasználó definiáltnak nevezzük
* Z specializációt attribútumdefiniáltnak nevezünk, ha egy predikátumot használhatunk minden egyes Z-beli alosztály tagságának a megadására
* kategória: osztály mely n definiáló szuperosztály uniójának részhalmaza, szuperosztályok attribútumaira előírt predikátumot használunk egyes elemeik megadására melyek elemei a kategóriának
* kapcsolattípus definíciójának kiterjesztése: megengedve, hogy bármely egyedtípus részt vehessen egy kapcsolattípusban (gyakorlatilag kicseréljük az egyedtípust osztályra a definícióban)
* EER séma leképzése relációs sémára (lépések, mint az ER leképzésénél)
  + **Több reláció – szuperosztály és alosztályok**
    - Hozzunk létre egy L relációt a C számára Attrs(L) = {k, a1, . . . , an} attribútumokkal és PK(L) = k elsődleges kulccsal. Hozzunk létre egy Li relációt minden egyes Si alosztályhoz (1 ≤ i ≤ m) Attrs(Li) = {k} ∪ {Si attribútumai} attribútumokkal és PK(Li) = k elsődleges kulccsal. Ez a lehetőség mindenféle specializáció esetén (totális vagy részleges, diszjunkt vagy átfedő) működik.
  + **Több reláció – csak alosztály relációk** 
    - Hozzunk létre egy Li relációt minden egyes Si alosztályhoz (1 ≤ i ≤ m) Attrs(Li) = {Si attribútumai} ∪ {k, a1, . . . , an} attribútumokkal és PK(Li) = k elsődleges kulccsal. Ez a lehetőség csak olyan specializáció esetén működik, ahol az alosztályok totálisak (minden szuperosztálybeli egyednek legalább egy alosztályhoz kell tartoznia). Ha a specializáció átfedő, egy egyed több relációban is felbukkanhat.
  + **Egyetlen reláció egy típus attribútummal** 
    - Hozzunk létre egy L relációt Attrs(L) = {k, a1, . . . , an} ∪ {S1 attribútumai}∪ . . . ∪ {Sn attribútumai} ∪ {t} attribútumokkal és PK(L) = k elsődleges kulccsal. A t-t típus (vagy diszkrimináló) attribútumnak nevezzük, amely jelzi azt az alosztályt, amelyhez az egyes rekordok tartoznak. Ez a lehetőség csak olyan specializáció esetén működik, amely diszjunkt, és fennáll a veszélye annak, hogy sok NULL értéket generál, ha sok speciális attribútum szerepel az alosztályban.
  + **Egyetlen reláció több típus attribútummal**
    - Hozzunk létre egy L relációt Attrs(L) = {k, a1, . . . , an} ∪ {S1 attribútumai} ∪ . . . ∪ {Sn attribútumai} ∪ {t1,t2, . . . ,tm} attribútumokkal és PK(L) = k elsődleges kulccsal. Minden ti (1 ≤ i ≤ m) logikai típusú attribútum, amely azt jelzi, hogy egy adott rekord az Si alosztályhoz tartozik-e. Ez a lehetőség olyan specializációk esetén is működik, amely átfedő alosztályokat tartalmaz.
* unió típusok leképzése: különböző kulcsokkal rendelkező szuperosztályok által definiált kategória leképzéséhez célszerű új kulcsattribútumot bevezetni, amelyet helyettesítő kulcsnak nevezünk, ezt minden relációban felveszünk, amelyet a szuperosztályból képzünk

9.

Tranzakciók:

* egyfelhasználós rendszer: egy időben legfeljebb egy felhasználó használhatja a rendszert
* többfelhasználós rendszer: konkurens módon több felhasználó érheti el a rendszert
  + konkurencia típusai: **összefésült**: egy cpu-n hajtódnak végre a konkurens processek, **párhuzamos**: konkurens módon több cpu-n hajtódnak végre a processek

Tranzakció fogalma:

* egy végrehajtás alatt álló program mely a DB-feldolgozás egy logikai egysége, egy tranzakció egy vagy több műveletből állhat
* egy alkalmazói program egynél több tranzakciót is tartalmazhat
* ha a tranzakció nem módosít, csak lekérdez read-only tranzakciónak nevezzük

Tranzakció állapotai:

* aktív
* részlegesen véglegesített
* véglegesített (commit)
* hibás
* megszakított

műveletek:

* begin\_transaction
* read vagy write
* end\_transaction
* commit\_transaction
* rollback vagy abort
* visszaállítás során az undo vagy a redo használhatóak

Tranzakció (heurisztikus):

* ez lehet önálló, vagy beágyazott (programon belül)
* (határai: Begin transaction és End transaction)
* egy alkalmazói program több elkülönült tranzakciót tartalmaz begin.. és end.. közé fogva
* tranzakciók szempontjából a DB egyszerű modelljét használjuk
* szemcsézettség (granularitás): adatok különböző méretű egységei
* alapműveletek írás és olvasás

Írás: read\_item(X) :megkeresi X elemet tartalmazó lemezblokk címét 🡪 átmásolja ezt a címet a fő memória pufferébe 🡪átmásolja a pufferből az X nevű program változóba

Olvasás: write\_item(X): megkeresi X elemhez tartozó lemezblokk címét 🡪átmásolja a címet a fő memória pufferébe 🡪átmásolja X elemet az X nevű program változóból a puffer megfelelő területére 🡪visszamásolja a frissített blokkot a pufferből a lemezre

Konkurencia kontroll:

* elvesztett frissítés probléma: mikor két tranzakció ugyanazokat az elemeket éri el és az összefésülés közben meghibásodnak
* időleges frissítés probléma (dirty read): mikor egy tranzakció frissít egy DB elemet, majd valami miatt a tranzakció hibásan fejeződik be és ezt a hibásan frissített elemet más tranzakció eléri, mielőtt megjavulna
* helytelen összegzés probléma: amikor tranzakció rekordok összegzése közben egy másik tranzakció frissít néhányat így az összegző függvény olyan értékekkel számolhat, ami frissítés előtt vagy után állhat
* célja:
  + elkülönítés kikényszerítése a konfliktusos tranzakciók közt
  + adatbázis konzisztenciájának megőrzése tranzakciók konzisztencia megőrző végrehajtása révén
  + olvasás-írás és írás-írás konfliktusok feloldása

Helyreállítás lehetséges okai:

* Számítógép hiba (rendszerösszeomlás): hardveres vagy szoftveres hiba tranzakció végrehajtásakor, hardveres sérülés esetén a memória tartalma elveszhet
* Tranzakció vagy rendszerhiba: bizonyos műveletek a tranzakcióban hibát eredményezhetnek (0-val való osztás, overflow…), akárcsak hibás paraméterek
* Lokális hiba vagy kivételt észlel a tranzakció: bizonyos esetekben szükséges lehet a tranzakció törlése
* konkurencia kontroll kikényszerítés: dönthet a tranzakció megszakításáról vagy újraindításáról, ennek oka lehet a szerializálhatóság követelménye vagy, mert több tranzakció deadlock állapotban van
* lemezhiba: a lemez egyes blokkjai elvesztették az adataikat
* fizikai probléma: bármilyen egyéb katasztrófa bekövetkezése

syslog:

* a tranzakciókat nyomon követi és naplózza
* visszaállításhoz szükséges lehet
* lemezen van, lemezhibát és katasztrófát kivéve minden hibára immunis
* periodikusan archiválni kell

Log rekordok típusai:

* [start\_transaction,T],
* [write\_item, T,old\_value,new\_value],
* [read\_item,T,X], [commit,T], [abort,T],

*ahol, T egy egyértelmű tranzakció azonosító.*

* a kaszkádolt rollbacket kerülő helyreállító protokollok nem igénylik a read műveletet, erősebb protokollok egyszerűbb write bejegyzést igényelnek melyek a new\_value-t nem tartalmazzák

Helyreállítás loggal:

* undo segítségével visszavonható egyetlen írási művelet az old\_value-val pedig az összes érintett adatbázis elem eredeti értékének a visszaállítása lehetséges
* a redo-t használhatjuk arra, hogy kikényszerítsük az írási művelet hatását minden tranzakcióban érintett elem értékének new\_value-ra való állításával
* Commit: akkor éri el a tranzakció ezt a pontot, ha minden művelet végrehajtódott
* Rollback: azoknál a tranzakcióknál kell, ahol van start\_transaction,T bejegyzés, de nincs commit a végén

Ütemezés:

* fontos a tranzakciós műveletek sorrendje, amikor összefésülve egy szálon hajtjuk végre
  + visszaállítható: egyetlen olyan T tranzakció sem véglegesítődik, amíg nem végleges minden olyan T’ tranzakció, amely olyan elemet ír ki, amelyet T beolvas
  + kaszkádmentes: minden tranzakció csak olyan elemet olvas be, amelyet egy már elfogadott tranzakció ír ki
* Szeriális ütemezés: …ha minden T tranzakcióra fennáll, hogy az összes T-beli művelet közvetlenül egymás után hajtódik végre
* Szerializálható ütemezés: szerializálható egy ütemezés, ha ekvivalens ugyanazon tranzakciók szeriális ütemezésével
  + eredmény ekvivalens: ugyanazt a végső állapotot eredményezi az ütemezés
  + konfliktus ekvivalens: amikor bármely 2 konfliktusos művelet sorrendje ugyanaz
  + konfliktus szerializálható: amikor az ütemezés konfliktus ekvivalens egy szeriális ütemezéssel

10.

Adattárházak: az adattárház adatok téma-orientált, integrált, nemváltozó időbélyeggel rendelkező összessége a menedzsment döntéseinek támogatására

Alapfogalmak:

* OLAP – online analytical processing: adattárházakból származó adatok elemzése
* DSS – decison support system: a vezető döntéshozókat támogatják
* adatbányászat – data mining: a tudásfeltárás fontos eszköze
* koncepcionális szerkezete: adattisztítás/ újraformázás 🡪OLAP 🡪adatbányászat

Jellemzőik:

* gyorsaságra optimalizálják
* hangsúlyos a régebbi adatok elérése és használata
* változatlan, ellenben a tranzakciós adatbázisokkal
  + Többdimenziós koncepcionális nézet
  + Általános dimenziókezelés
  + Korlátlan dimenzió és aggregációs szint
  + Dimenziók közötti műveletek korlátlansága
  + Dinamikus ritka mátrixok kezelése
  + Kliens-szerver architektúra
  + Többfelhasználós támogatás
  + Hozzáférhetőség
  + Átláthatóság
  + Intuitive adatmanipuláció
  + Konzisztens riportoló képesség
  + Flexibilis riportolás

Fajtái:

* vállalati: projekt, nagy idő és erőforrás ráfordítást igényel
* virtuális: operatív adatbázisok különböző nézetei, ezeket a nézeteket fizikailag is létrehozzák
* adatpiac: a szervezet egy jól meghatározott részét célozza meg

Adatmodelljei:

* hagyományosan kétdimenziós (táblázat) adatokkal foglalkozik, a többdimenziós (adatkocka) modellekben a lekérdezések hatékonysága jobb
* az adattárházak ténylegesen ki tudják használni ezt
* a többdimenziós egyes dimenzióit képes előtérbe helyezni

Többdimenziós sémák:

* Dimenzió-tábla: dimenziók attribútumainak rekordja
* Tény-tábla: minden rekordja egy rögzített tény adat
* Csillag séma: egy tény-táblát és minden dimenzióhoz egy egyszerű táblát tartalmaz
* Hópehely séma: továbbfejlesztett csillag séma, tartalmazza még a dimenzió-táblák egy hierarchiáját
* Tény konstelláció: ugyanazon dimenziók között osztozó táblák halmaza, behatárolják a lehetséges lekérdezéseket
* Indexelés: indexelést használnak a nagy hatékonyságú elérés támogatására – bitmap indexelés

Adattárházak építése:

* tisztában kell lenni a későbbi felhasználással
* ad-hoc lekérdezést támogatni kell
* alkalmas sémát kell alkalmazni a használathoz
* lépési
  + adatok gyűjtése a tárház számára
  + biztosítása annak, hogy a tárolás hatékonyan találkozik a lekérdezési követelménnyel
  + teljes áttekintése a környezetről, ahol működni fog

Adatok összegyűjtése:

* több heterogén forrásból kell kinyerni
* az adatokat formázni kell
* adatok tisztítása az érvényesség miatt
* modellhez illesztés
* adatok betöltése

Adattárházak építése:

* Tároljuk le az adatokat az adattárház adatmodelljének megfelelően.
* Hozzuk létre és tartsuk karban a szükséges adatszerkezeteket.
* Hozzuk létre és tartsuk karban a megfelelő elérési utakat.
* Gondoskodjunk az időben változó adatokról amint új adatokat adunk az adattárházhoz.
* Támogassuk az adattárház adatok naprakészre hozását.
* Frissítsük az adatokat.
* Tisztítsuk az adatokat.
* A használat megtervezése.
* Az adatmodell illeszkedése.
* A használható adatforrások jellemzői.
* A metaadat komponens tervezése.
* Moduláris komponens tervezése.
* A menedzselhetőség és a változás megtervezése.
* Elosztott és paralel architektúrák figyelembevételén alapul.

Adattárházak működése:

* gönygyölítés (roll-up)
* lefúrás (drill-down)
* pivot slice - dice
* rendezés
* szelekció
* származtatott attribútumok

11.

Objektum-orientált adatbázis rendszerek (OO): a tervező specifikálni tudja mind a sokszorosan összetett objektumok szerkezetét, mind az ezeken az objektumokon alkalmazható műveleteket.

* Object Databases (O(O)DB): Objektum-orientált adatbázisok
* Object Database Management Systems (OD(B)MS): Objektum-orientált adatbázis-kezel® rendszerek

Osztály: egy objektum belső adatstruktúrájának leírása egy osztály deklarációban

Absztrakt adattípus (absztrakt adatobjektum): objektum szerkezete és típus konstruktorok

Műveletek egységbezárása (encapsulation): a belső adatstruktúra elrejtése, az alkalmazható külső műveletek, metódusok meghatározása

Perzisztencia (állandóság) és tranziencia (ideiglenesség)

Típus vagy osztály hierarchia és öröklődés

Hibrid OO programozási nyelvek: C++

Objektum:

* komponensei:
  + állapot és viselkedés
* tranzisztens(átmeneti):
  + a futás befejeztével törlődik. OO nyelvben
* prezisztens(állandó):
  + eltárolódnak, később kinyerhetőek és megoszthatóak más programokkal. ODMS nyelvben.
* Példányváltozók (instance variables): azon értékeket hordozzák, melyek az objektum belső állapotát írják le.
* Teljes egységbezárás: minden a felhasználók által használható műveletnek előre definiáltnak kell lenni.
* részei: interfész (szignatúra) és metódus (törzs). Az első a művelet nevét és paramétereit írja le, a második a művelet implementációját specifikálja.
* Reláció-kezelés: az egységbezárás miatt ez elrejtésre került a külső felhasználók számára (korai rendszerek problémája), az ODMG szabvány ezt megoldotta a bináris kapcsolatok explicit megjelenítésével, inverz hivatkozási párral
* Művelet túlterhelés (operator overloading): ugyanaz a művelet többféle típusú objektumon is végrehajtható, a művelet nevéhez több különböző implementáció tartozik attól függően, hogy milyen objektumra kívánjuk alkalmazni.
* Késői kötés: a futás során kapcsolódik össze a művelet neve és a megfelelő implementáció.
* OO databázisokban egy összetett objektum állapota más objektumok alapján határozódik meg ún. típus konstruktorok segítségével.
* Alapvető típus konstruktorok: atom, rekord (struct, tuple), kollekció (több-értékű).

Objektum-relációs adatbázisok jellemzői:

* Típus konstruktorok melyekkel összetett objektumokat hozhatunk létre
* Objektumok azonosítását biztosító mechanizmus a reference type segítségével.
* Műveletek egységbezárása a felhasználó által definiált típusokon (UDT). A felhasználó által definiált eljárás (UDR) szintén megjelenik.
* Öröklődési mechanizmus az UNDER kulcsszó segítségével.

UDT:

* összetett szerkezetű objektumok létrehozása
* egy típus deklarációjának elválasztása a tábla (reláció) létrehozásától
* rekord típusú konstruktor a ROW kulcsszóval rekord típusú attribútumok létrehozására
* 4-féle kollekció típus: ARRAY, MULTISET, LIST és SET
* Objektum azonosító egyértelmű
* A példányosítható UDT-khez táblákat (relációkat) is létrehozhatunk.
* Az UDT-khez műveleteket (metódusokat) is definiálhatunk: CREATE TYPE típusnév ( );
* Attribútumok és műveletek három fajtája: PUBLIC - látható az UDT interfészen, PRIVATE nem látható az UDT interfészen, PROTECTED - csak az altípusok számára látható

12.

CAP tétel:

* Egy elosztott rendszer az alábbi 3 alapvető képesség közül legfeljebb 2-t tud megvalósítani
  + konzisztencia: minden csomópont egy adott pillanatban ugyanazt az adatot tárolja
  + rendelkezésre állás: minden kérésre érkezik válasz arról, hogy a kérés végrehajtása sikeres vagy sikertelen volt-e
  + particionálástűrés: a rendszer egy tetszőleges üzenet elvesztése vagy a rendszer egy részének hibája esetén is tovább működik, csak a teljes rendszer hibája okozhat működési hibát

NoSQL:

* 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)