1. Átalános bevezető az adatbázis-kezelésről, a relációs modell: attribútumok, sortípusok, relációsémák, kulcsok

Táblázat	reláció
oszlop	attribútum
sor	elem (?) →azon belül mező (típusa van)
(?)	séma→reláció neve+attribútumhalmaza

Előfordulás: konkrét adatértékek.
Tábla: kapcsolat két adatbázis között.
Relációs adatmodell: táblázatban tárolás.
Magasabb rendű programnyelv, a felhasználónak nem kell pontosan ismernie a fizikai megvalósítást.
Inkább logikai adatmodell.
Kevés, de egyszerű parancs→könnyen optimalizálható.
Kulcs meghatározása: olyan mező(k), melyek a sort egyértelműen azonosítják.
Külső kulcs: olyan mező az adott táblában, ami egy másik táblában kulcs.

2. Adatbázissémák megadása SQL-ben, CREATE TABLE, PRIMARY KEY, FOREIGN KEY

SQL (Structured Query Language), utolsó szabvány: '99-es.
Adatdefiníciós nyelv (DDL) – séma megadása
Adatmanipulációs nyelv (DML) – lekérdező nyelvek, módosítások
Alaptípusok: integer, boolean, string, date, time.
Reláció létrehozása: CREATE TABLE Name (név típus, …)
folytatásként (a típus után, a vessző előtt): PRIMARY KEY→ rendszer ellenőrzi, hogy csak egy legyen. Ezen esetben S attribútum nem lehet NULL. A UNIQUE kulcsszónak hasonló a hatása, de lehet az attribútum értéke NULL is. Lehet még DEFAULT és NOT NULL.

3. A relációs algebra, alapműveletek, kifejezések

Projekció: oszlop(ok) kiválasztása: $\pi_{attr.}(R)$
Szelekciós: sor(ok) kiválasztása: $\sigma_{felt.}(R)$. Feltétel lehet: <, >, = , !=, stb.
Unió, metszet, különbség.
Descartes (direkt-) szorzat: $R_1 x R_2$.
Átnevezés: $\rho_{s(attr.)}(R_{attr.})$

4. SQL SELECT alapformája: SELECT-FROM-WHERE záradékok, UNION, INTERSECT, EXCEPT

SELECT * FROM R→R-et visszaadja
SELECT * FROM R WHERE A="a"→projekció A-ra feltétellel.
SELECT B,C (attribútum lista) FROM R (itt lehet?????) WHERE A="a" (feltételek)→projekció A-ra feltétellel, majd szelekció B,C-re.
Alapból nincsenek szűrve a többszörös elemek. Ehhez a SELECT DISTINCT használata.
SELECT = Π; FROM = ×; WHERE = σ_F
Halmaz műveletek pl: (SELECT név, cím
FROM FilmSzínész WHERE nem ='N')
INTERSECT/UNION/EXCEPT (SELECT név, cím
FROM GyártásIrányító WHERE nettóBevétel > 10000000)

5. Funkcionális függőségek, X^+ definíciója, következés, levezetés, levezetési szabályok, összekapcsolási függőség.

Funkcionális függőség az R(U) séma felett:
X→Y funkcionális függőséget jelöl, ahol: X,Y $\subseteq$ U. Ha Y=U, akkor Y kulcsfüggőség.
Az I előfordulás kielégíti X→Y-t, ha bármely t_1,t_2 eleme I sorokra $t_1[X]=t_2[X]$, akkor $t_1[Y]=t_2[Y]$. Logikai jelölés: I |= X→Y (|= → implikája)
Szövegesen: X attribútumok értéke egyértelműen meghatározza Y értékét, így amik X-re azonos értékűek, azok Y-ra is azok.
Függőségek implikációja (logikai **következés**):
Legyen F:{$X_i \rightarrow Y_i$ | i=1,…,k} funkc. függőségek véges halmaza.
Az f: X → Y funkc. függőséget implikálja F, ha minden I előfordulásra I |= F → I |= f (Vagyis, ha I kielégíti F minden függőségét, akkor I f-et is kielégíti.)
X lezárása F szerint:
X részhalmaza U, F funkcionális függőségek halmaza. F={$X_1 \rightarrow Y_1,\ldots,X_k \rightarrow Y_k$}
X lezárása F szerint: $\mathbf{X^{+F}}$={A|F |= X→A}
Lemma=F |= X→Y ←→ Y részhalmaza X^{+F}. (A legbővebb halmaz, amire F={ X→Y} teljesül.
Levezetési szabályok: (Armstrong axiómák)
1. reflexiós axióma: Y részhalmaza X esetén X→Y mindig teljesül.
2. Bővítési axióma: X→Y teljesül és V részhalmaza U tetszőleges, akkor: XV→YV is teljesül.
3. Tranzitivitási axióma: Ha X→Y és Y→Z teljesül, akkor X→Z is teljesül.

Levezetés:
Az X→Y levezethető az F={$X_i \rightarrow Y_i$, i=1…k} halmazból a fenti három szabály véges sokszori alkalmazásával megkapjuk X → Y-t. (Jelölés: F |– X→Y)
Ha X→Y teljesül és Y_1,Y_2 eleme Y, akkor $X \rightarrow Y_1$ $X \rightarrow Y_2$ is teljesül, fordítva, ha $X \rightarrow Y_1$ és $X \rightarrow Y_2$ teljesül, akkor és $X \rightarrow Y_1Y_2$ is teljesül
Összekapcsolási függőség:
R(U) séma felett $X_1,\ldots,X_n$ részhalmaza U és az összes Xi uniója U.
Jelölés |><| [X1…X_n]
I |= |><| [X_1…X_n] ←→ I=π_{X1}(I)|x|…|x|π_{xn}(i)
Implikáció a funkcionális, többértékű és összekapcsolási függőségekre eldönthető.

6. Relációs adatbázisok tervezése: normál formák és normalizálás felbontással, BCNF, 3NF

Normalizálás lényege: egyszerű függőségi rendszerek teljesüljenek a relációkban. Lehetőleg minden kulcsfüggőségből következzen. Ezek a **normálformák**.
Módszer: vetítéssel dekomponálás, ezzel egyszerűbb függőségi rendszer lesz a vetületeken. DE! Ne legyen információvesztés (vetületekből elő lehessen állítani az eredetit – veszteségmentes összekapcsolási dekomponálás= v.ö.d.) és ne veszítsünk függőséget (a vetületi függőségekből megkapható legyen az eredeti).
Normalizálás dekomponálással: R(U) sémájú relációhoz az F függőségi rendszert adtuk meg. Dekomponáljuk az X_1---X_n attribútumokra való vetítéssel
Kapjuk az $\mathbf{R_i(X_i) = \pi_{xi}(F)}$ vetületet az $F_i=\pi_{xi}(F)$ vetületi függőségekkel.
A dekomponálás akkor és csak akkor veszteségmentes dekomponálású, ha
F |= |><| [X_1…X_n]
Boyce-Codd Normál Forma (BCNF):
abban van, ha: F |= X→A és A $\notin$ X, XA $\notin$ U, akkor F |= X→U(X (szuper)kulcs).
Ilyen alakra dekomponálás: (van v.ö.d. dekomp)
Lépésenként két részre bontunk egy relációt, amelyik megsérti a BCNF-et, azzal: S(V) sémájú reláció G függőségi rendszerrel megsérti a BCNF-et egy G |= X→A függőséggel, vegyük az $S_1(XA)=\pi_{XA}(S)$ és $S_2(V\text{-}A)=\pi_{V\text{-}A}(S)$ vetületeket.
Egyszerűen belátható, hogy X→A |= |><| [XA,V-A] azaz S dekomponálása G teljesülése esetén v.ö.d.
Ezt ismételjük, amíg minden BCNF-ben nem lesz. (A vetületeken a vetületi függőségi rendszereket kell használni!) Függőségőrző dekomponálás BCNF-re általában nem lehetséges.

3NF normálforma:
Ha F |= X→A és A $\notin$ X, XA $\notin$ U, akkor vagy F |= X→U, vagy A elsődleges attribútum. A elsődleges, ha A egy F-szerinti minimális kulcs eleme.
3NF-re dekomponálás:
Létezik egyszerre v.ö.d. és függőségőrző dekomponálás. Algoritmus alapja: minimális függőségi rendszerét vesszük F-nek, ami $X_i \rightarrow A_i$ alakú függőségekből áll. Az X_iA_i halmazokra vetítünk, és még egy X kulcsra.

7. Az Egyed-Kapcsolat (EK) modell, EK diagramok, kapcsolattípusok, „is a" hierarchiák, kulcsok, gyenge egyedhalmazok

EK építő elemei: fogalmak – névvel ellátott séma-elemek – előfordulások
Entitás: a többitől megkülönböztethető
Alapfogalmak:
Egyedhalmaz: entitásokból áll, entitás az alapfogalom: „idea", névvel ellátott halmaza a hasonló entitásoknak
Tulajdonságok: attribútumok, az egyedhez tartozó tulajdonság egy értéktartományból vehet fel értékeket (hossz, név, életkor stb.) Az egyedhalmazt az egyedek tulajdonságai jellemzik.
<Halmaznév>(<Tulajdonságnév1>…<Tulajdonságnévk>)
Kapcsolatok: egyedek között van, névvel azonosítjuk, a kapcsolatban álló egyedek halmazneveivel jellemezhető (vagyis megadja, hogy milyen típusú entitások között áll fent a névnek megfelelő kapcsolat).
<Kapcsolatnév>(<Tulajdonságnév1>…<Tulajdonságnévk>)
Bináris (E-K-F; N:M-hez, E-K→F;N:1-hez, E←K→F;1:1-hez), vagy többszörös (sok az egyhez, egy a sokhoz kapcsolat) kapcsolat lehet.
Graf ábrázolás: egyedhalmaz (négyzet, benne a halmaznév), tulajdonságnév bekarikázva, kapcsolatnév rombuszban.
„is a": speciális részhalmaz megadására jó. Ha A is a B, akkor e $\in$ A, akkor e $\in$ B is. Jele: háromszögben „is a".
Kulcs: az E egyedhalmaz tulajdonságainak egy K halmaza kulcs, ha bármely két egyed nem vesz fel K minden elemére azonos értéket. Legalább egy K-beli tulajdonságuk eltér.
Kulcs-feltétel: minden egyedhalmazhoz legyen kulcs („is a"-nál a gyökéren legyen, akkor a többire is kulcs). Kulcsokat aláhúzással jelöljük.
Gyenge egyedhalmaz: nincs közvetlen saját kulcsa. Ilyenkor más egyedhalmazokkal kiegészülve lesz kulcsa.

8. Az E/K diagramok leképezése relációs adatbázisba

Erős egyedhalmazok: (mindegyik önmagában reláció is lehet, kulcsuk is van)
- Halmaz → Relációnév
- Tulajdonságnév → Attribútum név
- Kulcs → Primary Key

is a hierarchia: A halmazokhoz hozzávesszük a gyökér kulcsát. (így erős EH lesz)
Kapcsolat: erős osztályok egyedeit kulcsértékeik reprezentálják. A kapcsolatnak megfeleltetett reláció: az erős egyedosztályok kulcsaiból képezzük a sorait. Az 1:N kapcsolatok funkcionális függőséget jelentenek.
Gyenge egyedhalmazok: erős EH-zá tesszük (külső egyedekből szükséges tulajdonságokkal) – ez az N:1 kapcsolatokat reprezentálja.

Minden EH-hoz kulcsot rendelünk (erős lesz), és így képezzük relációba.
Ha ezzel már reprezentálunk egy kapcsolatot, azt nem kell külön feltüntetni
Az EH kulcsa idegen kulcs lesz a relációs modellben.

9. Alkérdések a WHERE záradékban és a FROM záradékban, korrelált alkérdések

A SELECT eredményét lehet tovább használni:
- Ha az eredmény skalár: **WHERE**-ben használható összehasonlításra
- Ha reláció, akkor **WHERE**-ben (több lehetőség) vagy **FROM**-ban névvel ellátva használható

1. WHERE … × Θ (SELECT …) skalár (Θ – összehasonlító rel.)
2. SELECT eredménye R
   2.1 Ha egy attr.-a van az eredmény relációnak:
   EXIST R - Nem üres
   S IN/NOT IN R - S értéke benne van/nincs benne
   S < / > ALL/ANY R - S kisebb/nagyobb minden/valamely R értékénél
   S <> ALL R - ≡ S NOT IN R
   S = ANY R - ≡ S IN R
   ↳ Ezek eredménye IGAZ/HAMIS, így tovább használhatók.
   2.2 Ha sorokat kapunk vissza: (sor használata SQL-ben:)
   A sor komponensek (k_1-k_n) lehetnek konstansok (egyszer kell kiértékelni) vagy változók (FROM listából attr., szabad sor). Utóbbit minden előfordulásra meg kell adni.

Ha *t* sor típusa = R típusa → összehasonlítható (R soraival)
t konstans sor: WHERE-ben csak 1x kell kiértékelni
t szabad sor: *t*-t minden előfordulásra ki kell értékelni

Korrelált Alkérdések:
(SELECT…) – Külső változót használ (FROM-ban is lehet alkérdés)
SELECT Cím FROM Film Régi WHERE év < ANY (SELECT év FROM Film WHERE Cím=Régi.cím) → olyan címek, aminél később is készült unolyan c. film

10. Összesítő műveletek, ismétlődések kezelése, csoportosítás, DISTINCT, GROUP BY, HAVING, ORDER BY

Statisztika: COUNT (*) – sorok száma; AVG; SUM; MIN; MAX
(SQL-ben aggregáló fv.ek)

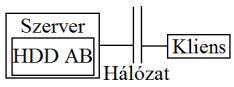
SELECT DISTINCT – duplikátumok kiszűrése
(SELECTEK között: UNION (U), INTERSECT (∩), MINUS/EXPECT (-)
→ Ha ALL-t használunk, akkor multihalmaz jön létre és többszörös előfordulásokkal számol.)
ORDER BY <attr. lista> - rendezés (attr. név: ASC / DESC)
GROUP BY (attr. név) – csoportosítás
pl: SELECT hallgató_azon, AVG(jegy) FROM Érdemjegy GROUP BY hallgató_azon
HAVING – pl.: M filmet gyártott stúdió (?)
SELECT SUM (hossz) H, Studiónév FROM filmek GROUP BY Studiónév, HAVING COUNT (*) > 4 ORDER BY H ((*) – a csoportosításra hivatkozik)
WITH – rekurzív kérdések:
WITH RECURSIVE (reláció_deklaráció) AS (<lekérdezés>)
Idézet: WITH RECURSIVE Lánc(X,Y) AS (SELECT * FROM Idéz) UNION (SELECT FROM Lánc WHERE Y=U);
SELECT X?Y FROM Lánc WHERE X=”x” AND Y=”y”;

11. Módosítási utasítások: INSERT, DELETE, UPDATE; A CREATE VIEW utasítás; (Munkarelációk: WITH záradék)

INSERT – sor beszúrása
1 sor: INSERT INTO Reláció (attr. lista) VALUES (v_1-v_n);
több sor: INSERT INTO Reláció (attr. lista) SELECT DISTINCT Stúdió FROM Film WHERE Stúdiónév NOT IN (SELECT név FROM Stúdió)
DELETE – sor törlése: DELETE FROM R WHERE <feltétel>;
1 sor: Primary Key lesz a feltétel
UPDATE – sor módosítása: UPDATE R SET <új értékek> WHERE <feltételek>;
Primary Key nem módosítható!
CREATE VIEW – névvel ellátott, előre definiált lekérdezés (WITH záradék)
VIEW: Séma szinten definiált alkérdések:
CREATE VIEW (relációnév(attr.lista)) AS (<lekérdezés>) SELECT…
- Alapértelmezésben használatkor értékelődik ki.

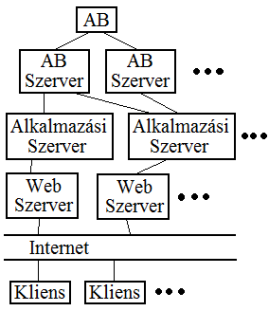
„materializált” VIEW: tárolódik a kiértékelt VIEW, változáskor frissül

12. SQL környezetek, szerver-kliens architektúra, háromrétegű architektúra, procedurális nyelvi környezetek



Kliens-szerver architektúra (90’s – kétrétegű)
A kliensen futnak a felhasználói feladatok.

Háromrétegű architektúra (ma)
Az alkalmazási szervereken futnak a felhasználói feladatok.
(az AB Szerverek és az Alkalmazási Szerverek összemosódnak.)



Magasabb rendű nyelvekbe (pl C) beágyazás jó, mert interaktívan használható és a rekurzió megoldható.
Befogadó nyelv (C) + Beágyazott SQL → Elő feldolgozás → Befogadó nyelvi prg. → Fv. hívások → Befogadó nyelv fordító (SQL könyvtár) → Tárgykód
Beágyazott SQL-nél nem relációs algebrát használunk, hanem soronként manipulálunk.
Előtag C-hez: EXEC SQL …
Változók megadása → :x
Skalár lekérése: EXEC SQL SELECT (…) INTO :x
1 sor: EXEC SQL SELECT (…) INTO (:a, :b, :c)
Több sort is soronként lehet, sormutatóval (CURSOR)
EXEC SQL DECLARE <sormutató név> CURSOR FOR <SQL lekérdezés>
Használata: EXEC SQL OPEN <sormutató név>;
EXEC SQL FETCH FROM <sorm. név> INTO <vált.k listája>;
EXEC SQL CLOSE <sormutató név>;
Még pl.: DELETE CURRENT OF <sormutató név> R.név;

13. Fizikai adatszervezés: fájlok, indexek, B-fák, tördelő címzés, összetett indexek, bittérképek

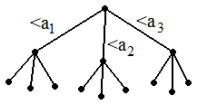
Alapfeladat: rekordok elhelyezése/visszakeresése (háttértároló blokkjain)
Rekord szerkezet: mezők (hierarchikus struktúra); fejléc; kulcsmezők
Fájl: háttértároló blokkok halmaza; címzése: fizikai cím/logikai cím (prg.)
Ha a visszakereséshez nincs támogatás, akkor az egészet végig kell olvasni. Ha van, akkor kulcs/mező szerinti visszanyerés (index alapú vagy tördeléses).

Indexelés: (kulcs – cím) párok: egyet ad vissza
(mező – cím) párok: többet is visszaadhat
- Sűrű index: minden rekordról egy bejegyzés
- Ritka index: blokkonként egy kulcsérték (ezek diszjunkt intervallumokat alkotnak)

Általában az index tábla nem fér a memóriába → rendezett index-fájl készül

Relációs rsz-eknél: 1 sor 1 rekord, és egy fájl több reláció sorait is tartalmazhatja

Változó AB → változó indexállomány
Index fájl – **Kereső fa:**
él=blokk cím; csomópont=index blokk
(a_1,cím$_1$; a_2,cím$_2$; a_3,cím$_3$)



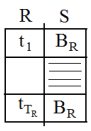
Nem jó, ha túl mély a fa → kiegyensúlyozott kereső fa kell:
B, B$^+$ fa: garantálja a kiegyensúlyozottságot → módszer: belső csomópontban (m+1, 2m) között van minden esetben az indexbejegyzések száma.
A rekord mélység: ($\log_{m+1}$n, $\log_{2m}$n) között. (CREATE INDEX ON (attr.név))
Tördelő táblák: kulcsból címet számolunk h függvénnyel: h(k)=cím
Ez a cím egy táblázatra mutat, és ott egy kosár címe található, ami egy háttértároló blokkra mutat.
Visszakeresés: k →blokk→rekord keresése
Elhelyezés/beillesztés feladata: h(k)=c →blokkcím →leolvasás, ha van benne hely, beillesztem a rekordot; ha telített a blokk, akkor folytató blokkba kerül (megoldás: felezés)
B fában lehet intervallumra keresni. (Több dimenziós indexelés kevésbé hatékony)

Bittérképek: rekord sorszámozás 1 - n-ig, attribútumnak m értéke lehet.
a egy érték: n hosszú bitvektor → amelyik rekordban A értéke a, ott 1 bit, egyébként 0
A teljes térkép A-ra: m db n hosszúságú bitvektor: n · m bit (ha m nagy, akkor a vektorok ritkák)
((Pl.: σ A=a ^ B=b (R) →))

14. relációs algebrai műveletek kiértékelési költsége: I/O lépések száma, méretek, memória méret szerepe

Szelekció minél mélyebben – nincs egzakt optimalizálás
Direkt szorzat szűrés.
Költségbecslés: háttértároló blokkok írási/olvasási (I/O) száma
(a leglassabb a háttértár felé?)
Relációs környezetben: sorok száma=T_R; blokkok száma=B_R, képméret=I_A(értékek száma)
Az output méret adott, az alapműveletek költsége → memória mérete blokkokban (M) a legfontosabb

Direkt szorzat: R × S ; Eredmény: $T_RB_S + T_SB_R$ db (R és S rendezve)
Az eljárás költsége: M > min(B_R,B_S) → *egymenetes algoritmus:*
Beolvassuk a kisebb relációt a legfeljebb M-1 méretű blokkba, és blokkonként olvassuk a másikat.
Több menetes algoritmus: M < min(B_R,B_S)
Beolvasunk R-ből M-1 blokkot, ehhez a teljes S-t blokkonként; és ezt ismételjük M-1 blokkonként



$$B_R + \frac{B_R}{M-1} B_S \,, \quad \frac{B_R}{M-1}$$ - nél kevesebb blokkolás nem elég.

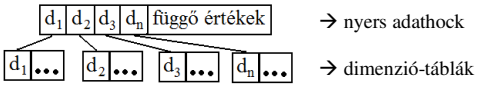
Összekapcsolás: R(A,B) |><| S(B,C) – Mérete: $\frac{T_R B_S + T_S B_R}{I_B}$ (t_1,B_S, B=b)
- ha M elég nagy: M > min(B_R,B_S), akkor a × egymenetes algoritmusa jó
- ha M kicsi: érdemes mindkét relációt B szerint rendezni
Rendezés költsége: $2B_R\log_M B_R + 2B_S\log_M B_R + B_R + B_S$
(rendezéssel összemérhető feladatok: N·$\log_B$N)

Tehát a direktszorzatnak és az összekapcsolásnak ~ négyzetes költsége van.

15. OLAP: on-line analitikus feldolgozások, a ROLAP csillagsémája, dimenziók, dimenziótáblák, szeletelés-kockázás, mélyreásás-felgörgetés, a MOLAP adatkocka és vetületi összesítései

OLAP – megoldások aggregáló műveletekre (COUNT, SUM …)
↳ Váljon el az operatív működéstől.
↳ Tényadatok halmaza
paraméterek=dimenziók – értékek=függőértékek
(pl.:hallgató, kurzus , érdemjegy)
Alap-ténytábla: nyers adatkocka
↳ +kapcs: Dimenzió-táblák: a dimenzió értékek külső kulcsok a dimenzió-táblán

ROLAP (Reláció OLAP) csillagsémája:



Alaptáblához kapcsolunk dimenzió-táblákat, és GROUP BY művelettel csoportosítható lekérdezést hajtok végre.
(Pl.: Eladások(sorszám, dátum, Forgalmazó, ár); Autók(sorszám, modell, szín); Forgalmazó(név, város, állam, tel) → az Eladások, Autók és a Forgalmazó a dimenziók)

Kockázás: Adatkocka felépítése (pl: tényleges kocka és az élei: sorszám,dátum,forgalmazó)
Szeletelés: valamelyik dimenzió értékét (intervallumát) rögzítjük (és így kivágunk egy szeletet a kockából)

Lekérdezés csillag-sémából:
SELECT (csoportosító attr.ok összesítések)
FROM <ténytáblák 0 vagy néhány dim.táblával összekapcsolva>
WHERE (…)
GROUP BY (attr., …)

Pl.: SELECT állam, AVG(ár)
FROM Eladások, Forgalmazók
WHERE Eladások.forgalmazó=Forgalmazók.név AND dátum>='2006-01-04'
GROUP BY állam;

Lefúrás/mélyre ásás = finomabb csoportosítás
Felgörgetés = durvább csoportosítás (pl.: hónapról évre)

Adatkockák: MOLAP (Multidimensional OLAP)
Egy nyers ténytáblából kockát épít: dimenziók mentén adunk egy felbontást, a legfinomabb cellákba az aggregált értékek kerülnek.
A kocka határain lehet summázott sorokat (felületeket) csinálni (*).
A technika: (d_1, … , d_k, összesítés)
(a_1, … , a_n, összeg) → konkrét elemi cella a kockában
(*, a_2, … , a_k, *összeg)
Pl.: CREATE MATERIALIZED VIEW EladásokKocka AS
SELECT modell, szín, dátum, forgalmazó, SUM(összesít), SUM(db)
FROM Eladások
GROUP BY modell, szín, dátum, forgalmazó WITH CUBE;

16. Az objektum-relációs modell, a felhasználó által definiált adattípusok (UDT) az SQL-99-ben

Az objektum-orientált adatkezelők nem teljesen jók abból a szempontból, hogy visszafelé nem kompatibilisek. Ezért inkább beépítettek objektum-orientált elemeket a relációs rendszerekbe. Ezek visszafelé kompatibilisek.
Új elemek a modellben:
1) Az attribútumokhoz összetett szerkezetekkel rendelkező típusú adatok is rendelhetők (pl. egy reláció, azaz sorokból álló halmaz), nem csak atomi értékek. (sor tölti be az objektum szerepét, csak az attribútumokhoz rendelhető összetett érték).
2) Metódusok (mint az objektum-orientált modelleknél)
3) Azonosító rendelhető a sorokhoz (nem ugyanaz, mint a kulcs!). A sor tölti be az azonosítóval rendelkező objektum szerepét. Ez azt jelenti (az ODL-hez viszonyítva), hogy az objektumokhoz tartozó adattípus csak sor lehet.
4) Hivatkozások (references). A sor-azonosítókra lehet hivatkozni, pointerként használható, többféle módon.

Beágyazott relációk: spec. eset, a '80-as években már ismerték. Az ilyen szerkezetben csak felváltva használható a sor-halmaz konstrukció. Ez azt jelenti, hogy egy attribútumhoz értéktípusként egy relációséma rendelhető, értékként pedig egy relációelőfordulás.
A séma építése:
Alap: egy atomi típus, ami egy attribútum típusa lehet.
Indukció: Egy reláció típusa lehet bármilyen séma, ami egy vagy több attribútumnévből áll, és minden attribútum típusa legális. Az attribútum típusa is lehet ilyen séma.
Pl.: R(<A:egész, B:<C:sztring, D:sztring, E:<F:egész>>>)
Hivatkozások: Beágyazott relációknál előfordul, hogy egy sor több relációban is szerepel. Ezt elég egyszer tárolni és többször hivatkozni rá (pointerrel).
R reláció egy sorára hivatkozás: *R, több sorára: {*R}
Pl.: A:*R, A:{*R}
Hivatkozás, mint objektumazonosító (OID): tipikus OODB-kben nem hozzáférhető, ORM-ben igen.
SQL-99 szabvány:
Felhasználó által definiált típusok (UDT-k): lehet egy tábla típusa, egy attribútum típusa, ami valamilyen táblához tartozik.
Egyszerű eset: CREATE TYPE Hőfok AS INTEGER
CREATE TYPE Hossz AS INTEGER
Összetett eset: Táblatípus, mint UDT: alakja: CREATE TYPE T AS (<attribútum deklarációk>)
Pl.: CREATE TYPE Lakcímtípus AS (
utca CHAR(50),
város CHAR(20)
);
Metódus deklaráció: Pl.: CREATE TYPE Lakcímtípus AS (
utca CHAR(50),
város CHAR(20))
METHOD házszám() RETURNS CHAR(10);
Külön deklarálva: CREATE METHOD házszám() RETURNS CHAR(10)
FOR lakcímtípus
BEGIN
tetszőleges sztring, ami egy program kódja
END;
UDT reláció deklarálásnál:
CREATE TABLE <relációnév> OF <UDT név>
(<spec. elemek listája – PRIMARY KEY, FOREIGN KEY, sorra vonatkozó feltételek stb.>)
Hivatkozás T sorra: REF(T), pl.: A REF(T) SCOPE R: A attr. az R reláció T sorára hivatkozik. Ha nincs SCOPE, bármely T típusú sorra vonatkozhat! Ha x egy REF(T), ami a t sorra hivatkozik, akkor t sor A attribútumára x→A hivatkozik. A teljes sorra a DEREF operátorral hivatkozhatunk (DEREF(T))
- Sorazonosítók létrehozása: REF IS <attribútumnév> <generálás módja>
SYSTEM GENERATED, vagy DERIVED (kulcs alapján)>
UDT-vel adott tábla sortípust jelent, hivatkozás attribútumokra megfigyelő metódussal: t.x()
UDT-komponensek létrehozása és módosítása: generálás: T() →T típusú sort generál
változtatás: t.x(v)→a t sor x attribútumának v értéket adunk.
UDT-k rendezése:
1. CREATE ORDERING FOR T EQUALS ONLY BY STATE; (teljes tartalom egyezés)
2. CREATE ORDERING FOR T ORDERING FULL BY RELATIVE WITH F; (Az F(x1,x2) fv-nyel kell megadni!)

17. A féligstruktúrált adatmodell és az XML, az XML DTD és séma

Web eredményei: globális infrastruktúra, std.ek a dok. cserében, HTML megjelenítési formátum, dok. visszakereséshez jól tervezett felh. felület, XML a dok.ok szerkezettek együtti cseréhez

AB tech. eredményei: tárolási technikák és lekérdező nyelvek nagy ABok hatékony eléréséhez (erősen strukturált adatokhoz), adatmodellek, új modell: féligstruktúrált adatmodell, ami feloldja az erős AB rsz.en belüli struktúrát.

Paradigmaváltás: Kiens-Szerver AB → Mediátorok → Kliens-Szerver

A féligstruktúrált adatmodell: "séma nélküli", "önleíró"
Címke-érték párok megadása, az érték lehet összetett is!
Minden személynél más lehet az adatleíró szerkezet → a címkék jelentése alapján értünk mindent.
(pl.: egyszerű:{név:"Nagy István",tel:123…,email:"…"}
összetett:{név:{unév:"István", csnév:"Nagy"},tel…….})
Fa struktúrában jól ábrázolhatóak → élcímkézett, rendezetlen fa

Alaptípusok: szokásos (számok, string…), újabb (data, gif, zip, kódolt adat,… ezekhez kellenek "tag"-ek)

Relációs AB reprezentálása:
{r1:{row:{a:a1, b:b1, c:c1},
row:{a:a2, b:b2, c:c2}};

a	b	c
a1	b1	c1
a2	b2	c2

XML (Extensible Markup Table) – adatcseréhez formátum
Gyökerei: dokumentum markup-nyelv (innen a hiányosságok is)
HTML-hez hasonló, de nem a megjelenítési forma, hanem a tartalom leírása a cél.
HTML: rögzített elhatárolók, jelölők: ; …
Újdonság: - Új jelölők (tag) tetszőlegesen definiálható (pl.: <személy> </személy>)
- A struktúra tetszőleges mélységbe beágyazható
- XML dok.hoz tartozhat egy opcionális nyelvleírás
Speciális kiterjesztések: jelölők és hivatkozások típusai

Az alapszintaxis: Fő komponense az elem, amit <tag>-ek (nyitó és záró) határolnak. A jelölő pár között lehet szöveg, egyéb elem (ekkor ez részelem ill subelement), vagy ezek keveréke.

Míg HTML-ben megjelenítéskor a jelölők eltűnnek, XML-ben nem.
Üres tartalom: <házas> </házas> → rövidítés: <házas/>

Lehet attribútumokat rendelni a jelölőkhöz, ezek tulajdonságok lesznek, és (név, érték) párokból fognak állni. (Pl.: <hangszer><név nyelv=”magyar”> tambura </név> <ár pénznem=”fabatka”> 1 </ár> </hangszer>)
Más forma: <személy név=”István” kor=”47”…> </személy>

Jól formált XML dok.: kevés feltétel – a jelölők egymásba ágyazása helyes legyen és az attribútumok egyértelműek legyenek a jelölőkön belül. (így ki lehet fejezni címkézett fába)

DTD (Documentum Type Definitions)
„kis” nyelvtan az XML dok. szerkezetéhez, de része is a dok.nak (opcionális)
Pl.: <!DOCTYPE db[
<!ELEMENT db(person*)>
<!E person(name, age, email())>
<!E name(#PCDATA)>
<!E age(#PCDATA)>
<!E email(#PCDATA)>
]>
Magyarázat: reguláris kifejezések → e*,e+,e?,elé…
Lehet rekurzív, pl.: bináris fa leírása:
<!ELEMENT node (leafl(node,node))>
<!ELEMENT leaf(#PCDATA)>

Előfordulás
<node>
<node> <leaf> 1 </leaf> </node>
<node> <leaf> 2 </leaf> </node>
</node>
<node>
<node> <leaf> 3 </leaf> </node>
</node>

Az XML **séma** leírása maga is egy XML dok. (névtartományt használ: www.....)
A séma dok. formája:
<? xml version =”1.0” encoding = „utf-8”?>
<xs:schema xmls:xs=”http://...”)
</xs: schema>
Az xs: prefix ezen a dokumentumon belül az elhatárolókat az XML schema szabályai kötik.
Összetevők, elhatárolók:
<xs: element name=*elemnév* type=*elemtípus*>
megszorítások és/vagy szerkezeti info
</xs:element>
type lehet egyszerű vagy összetett → egszerű pl: xs:integer, xs:boolean…
→összetettre egy pl: <xs:complexType name=*típusnév*>
<xs:sequence>
elemdefiníciók listája
</xs:sequence> </xs:complexType>

18. XML-adatok lekérdezése: XPath, XQuery, XSLT

XPath: XML-dokumentumokat, mint rendezett fát jár be. A bejárás egy szintjén valamilyen adattételekkel tér vissza. Adatmodellje: tételek szekvenciája. A bejárás köv. szintjén a tételek mindegyikének bejárását folytatja. Egyértelmű sorrendben újabb szekvencia keletkezik.
Tételek lehetnek:
1) Egyszerű típusú érték.
2) Csomópont (ezekből folytatható a bejárás). Legfontosabbak:
a) Dokumentumok (XML-dok-ot tartalmazó fájlok-lokális vagy URL elérés)(formája: doc(fájlnév)
b) Elemek (XML-elemek)
c) Attribútumok (nyitó jelölőkben)

XPath kifejezés a dokumentum gyökerével kezdődik és élek leírását tartalmazza:
$/T_1/T_2…/T_n$
(Példa a családfa, oda-vissza tengelyekkel (szülő/gyerek tengely)
Útkifejezés egy lépése lehet: /jelölőnév, /attribútum név, /tengely: jelölőnév, /tengely: attribútum név
Rövidítések jelölése: //az összes leszármazott és a megtalált leszármazottja is
/.. szülő
/. saját maga
/@ attribútum
A kifejezések kiértékelése mindig egy dokumentum szövegkörnyezetében történik.
Példák: /Filmszínész/Színész/Név kif.
attribútum elérés: $/T_1…/T_n/@A$: A attribútum értéke.
Tengely: él iránya, típusa jelenti a tengelyeket.
: bármi: pl.: /FilmszínészAdat//@*: minden attribútum értékét visszaadja a FilmszínészAdatból.

Abszolút xPath kif.: gyökértől/…
Relatív xPath kif.: nem /-rel kezdődik, egy kurrens csomópontra alkalmazzuk

Feltételek útkifejezésekben: /jelölő[feltétel, ami lehet =,<,>,!= stb.]
spec feltételek: [i]: sor i-edik elemére igaz, [T]: igaz, ha van T jelölőjű gyereke, [A]: igaz, ha van A nevű attribútuma.
Pl.:/Filmek/Film/Verzió[1]/@év
/ Filmek/Film/Verzió/[Színész]

XQuery: szekvenciákat kezel, hasonló a lekérdezése az SQL-hez, csak SELECT-FROM-WHERE helyett FOR, LET, WHERE, RETURN – FLWR-FLOWER-struktúra
for, let kiválaszt egy tételekből álló szekvenciát, where megszűri a feltételekkel, return minden tételből készít egy eredmény tételt. Pl.: RETURN <üdvözlet>Helló Világ!</üdvözlet> (egy XML-doksit ad vissza, konstans szöveg).
XQuery: kisbetű-nagybetű érzékeny!!!
XPath kifejezés: szekvenciát adó kifejezés XQueryben

Változók $ jellel kezdődnek.
Pl.: Let $filmek:=doc(„filmek.xml”)
for $m in $filmek/Filmek/Film
return $m/Verzió/Színész

let: értéket ad egy változónak → let $változó:=kif.
for:egy változóhoz ciklusban egy szekvencia tételei rendeli → for $változó in kif.

{} használata: $m értékét illeszti be (egyébként csak szövegként jelenne meg $m)
pl.:
let $filmek:=doc(„filmek.xml”)
for $m in $filmen/Filmek/Film
return <Film filmcím={$m/@filmcím}>{$m/Verzió/Színész}</Film>
Szekvenciák:
let $SzínészSeq:=(
let $filmek:=doc(„filmek.xml”)
for $m in $filmek/Filmek/Film
return $m/Verzió/Színész
)
return <Színészek>{$SzínészSeq}</Színészek>

XQuery beépített kifejezései:
- data(E):XML elem tartalmának eléréhez, ahol E egy elemet előállító kifejezés, az elem értékét adja vissza.
pl.: összekapcsolás: let $filmek:=doc(„filmek.xml”)
$színészek:=doc(„színészek.xml”)
for $1 in $filmek/Filmek/Film/Verzió/Színész,
$2 in $színészek/Színészek/Színész6
where data($s1)=data($s2)
return $2/Cím/Város
- XQuery összehasonlító operátorai: =, <, >, !=, eq, gt, lt, ne, le, ge stb.
- Duplikátum kiszűrése: disctinct_values(kif)
- Kvantifikálás: [every, some] változó in kif_1 satisfies kif_2 (logikai értékű)
- Összesítések: count, sum, max → mindegyik után: (kif)
- Elágazás: if (kif_1) then kif_2 else kif_3 → lehet a return-ben is használni
- rendezés, záradék a legvégén: order kif_1, kif_2,…,kif_n→ilyen sorrendben kapja a return a tételeket

XSLT (Extensible Stylesheet Language for Transformation):
(XML-ből HTML-be transzformálni)
XSLT specifikációkat is egy xml doksiban adjuk meg. alakja:
<? xml … ?>
<xsl:stylesheet xmlns:xsl=”http://www.w3.org/1999/xsl/Transform”>
…
</xsl:stylesheet>