3. Tétel

a) Adatbázisrendszerek I.

Hierarchikus, hálós és relációs modellek

- adatok szerkezetének leírására szolgálnak

Hierarchikus:

- 1:n kapcsolatot jelent (szülő-gyermek) fa szerkezetű pl. Iskola->osztályok->diákok

Hálós:

- Hierarchikus továbbfejlesztése, tetszőleges gráf szerkezet, egy egyedtípusnak több őse is lehet

- előnye több-több kapcsolat, hátránya nagy tárigény

Relációs:

- normál formák bevezetésével egzakt módon elvégezhető

- strukturált lekérdező nyelvet használunk hozzá (SQL)

Kulcsok a relációs modellben

- reláció = táblázat, relációs adatbázis = relációk összessége

- A kulcs: egyértelműen meghatározza az egyedtípus elemét

- Elsődleges kulcs: Egyedi és nem lehet NULL

- Másodlagos kulcs: Keresési célokra használható, Nem azonosítja egyedileg a rekordokat

- Idegen kulcs: Másik táblázat elsődleges kulcsára hivatkozik, Kapcsolatot hoz létre táblák között, biztosítja az adatintegritást

- Kulcsmegszorítások:

- Elsődleges kulcs (volt)

- Egyedi kulcs: ua. mint az elsődleges, de lehet null

- Egyedintegritási megszorítás

- pl. NOT NULL, idegen kulcsnál ugyanaz a típus

Kapcsolatok: egy-egy típusú, egy-sok típusú, sok-sok típusú

- 1:1 (1 férj 1 feleség)

- 1:N (1 nő több gyerek, 1 diák több könyv)

- N:M (1:N kapcsolatokra bontjuk: 1 diák több tantárgy, 1 tantárgyat több diák)

Anomáliák

- Beszúrási anomália: egy rekord felvétele egy másik, hozzá logikailag nem kapcsolódó adat beszúrását kívánja meg.

- Törlési anomália: az elem törlésekor a nem hozzá tartozó adatcsoportot is elveszítjük

- Módosítási anomália: egy adat megváltoztatása miatt, az adat összes előfordulási helyén el kell végezni a módosításokat az adatbázisban.

Funkcionális függőségek, tranzitivitás

- Funkcionális függőség:

- Egy a többhöz kapcsolat (AUTÓ\_RENDSZÁM -> TÍPUS, TULAJDONOS) (A → B)

- Funkcionálisan független: a viszony a két tulajdonságtípus között nem áll fenn pl. tanuló szemének színe és az iskola helye

- Tranzitív funkcionális függőség:

- AUTÓ\_RENDSZÁM → TULAJDONOS\_ID | TULAJDONOS\_ID → TULAJDONOS\_NÉV | AUTÓ\_RENDSZÁM → TULAJDONOS\_NÉV (A → B és B → C, akkor A → C is igaz)

Normálformák

- normalizálás célja: A normalizálás célja, hogy elkerüljük az anomáliákat, csökkentsük a redundanciát, és biztosítsuk az adatbázis integritását

- 1NF: Minden attribútumnak atomi értéke van, azaz minden mező egyetlen értéket tartalma

- 2NF: 1NF-ben van és minden nem elsődleges attribútum teljes funkcionális függőségben van (több táblába)

- 3NF: 2NF-ben van és nincs benne tranzitív függőség

- BCNF: minden determináns kulcs

- 4NF: BCNF-ben van és nincs benne többértékű függőség

- 5NF: 4NF-ben van és nincs benne illesztési függőség

b) Adatszerkezetek és algoritmusok

Algoritmus hatékonyságát befolyásoló algoritmizálási és adatkonstrukciós szempontok

- Hatékonysági szempontok:

- FONTOS A MEGFELELŐ ADATSZERKEZET KIVÁLASZTÁSA

- Végrehajtási idő (különböző méretű bemenetek esetén) (8 16 és 64 bites egészek összeadásánál vagy lebegőpontos számokkal való műveletek)

- Tárigény csökkentése (1024 elemű 8-bites / 64 bites tömbök)

- Bonyolultság csökkentése (vezérlési szerkezetek számától és helyzetétől)

- Algoritmizálási szempontok:

- rekurzió (egyszerűbb, memóriaigényes) vs iteráció (bonyolultabb, kevesebb memória)

- heurisztikus megközelítés (keresési algoritmus nagy gráfokban)

- Adatkonstrukciós szempontok:

- pl. tömbök, láncolt listák, hash táblák, fák, gráfok

- adatszerkezetek hozzáférése

- memóriahasználat optimalizálása: bináris adatok tárolása

Dinamikus adatszerkezetek (verem, sor, lista, hash-tábla) kezelésének modellje, a kapcsolódó adatszerkezetek implementációi, műveletei és alkalmazásai

- Verem:

- homogén, szekvenciális adatszerkezet.

- Alapját egy tömb képezi.

- Elemeihez csak az „egyik végén” férünk hozzá, vagyis azt az elemet tudjuk először kivenni, amit utoljára beletettünk - LIFO.

- Használható például posztfixes matematikai számítások kiértékelésére (Pl.+(2,5))

- Sor:

- a verem fordítottja: egyik végét írni, míg másik végét olvasni lehet.

- A legelőször belekerülő elemet olvashatjuk ki először - FIFO.

- Homogén adatszerkezet, alapját szintén egy tömb képezi

- Lista:

- mutató: a következő elemre mutat

- az elemek szétszórtan helyezkednek el a mem.-ben

- szekvenciálisan érhetők el

- dinamikus tárigény

- változatai: rendezett lista, ciklikus lista, többszörösen láncolt lista, kétirányú lista

- Hash-tábla:

- kulcs-érték párokat tárol

- gyors keresés a kulcsok alapján (indexelés)

- adatok hash-függvény segítségével tárolják (hash kódok)

Kereső algoritmusok és hatékonyságuk

- Eldöntés

- Lineáris keresés (összesen végigmegy, még ha nincs is benne a keresett elem)

- Kiválasztás

- Strázsás keresés (a ciklus feltételben csak egy vizsgálat 2/3-a a futási ideje a lin.-hez képest)

- Lineáris keresés rendezett sorozatban (pl. megállhatunk ha az aktuális elem nagyobb a keresettnél)

- Bináris keresés (csak egy részhalmazon kell vizsgálódnunk)

Programozási tételek értelmezése különböző homogén adatszerkezetek esetében

- alap-algoritmusok tömbökkel, listákkal, halmazokkal, sorokkal, vermekkel

- lineáris keresés, összegzés, maximális keresés, rendezésnél mondjuk buborékrendezés megfordítása – reverse elemek sorrendje

Rekurzió: rekurzió és iteráció

- Rekurzió: a függvény saját magát hívja meg

- báziskritérium: feltétel/állapot, azt mondja meg hogy mikor álljanak meg a hívások

- rekurzív lépés: aktuális probléma kisebb problémákra bontása

- pl. faktoriális képzés, Fibonacci-számok, hanoi tornya

- Iteráció (sokszor bonyolultabb, kevesebb memória) - ciklusváltozó

A fa adatszerkezet és műveletei

- hierarchikus adatszerkezet

- csomópontokból és összekötő élekből áll

- műveletei: beszúrás, törlés, keresés

- bejárásai:

- preorder (aktuális csomópont feldolg. majd bal, majd jobb részfák rekurzív bejárása)

- inorder (bal részfa majd aktuális csomópont végül jobb részfa)

- postorder: (bal és jobb részfák rekurzívan, majd csomópontok)