Fördelar med databassystem

- **Enkelt**
  -all data presenteras som tabeller
  -manipulation av databas görs med ett frågespråk(typ SQL)
- **Kraftfullt**
  -kan hantera komplexa relationer mellan data
  -komplicerade frågor kan ställas på enkelt sätt
- **Flexibelt**
  -lätt att ändra databasens schema
  -lätt att göra nya typer av sökningar
- Redundans kan minskas
- integriteten kan upprätthållas
- flera olika användargränssnitt
- flera applikationer/användare kan dela på databasen
- säkerheten kan förbättras
- stöd för återhämtning av data
- tiden för applikationsutveckling kan minskas

Nackdelar med databassystem

- Ofta höga kostnader för hårdvara, mjukvara och utbildning
- Komplexiteten hos ett DBHS kan ibland vara onödig eller ger för låg prestanda

Datamodeller(mindre viktigt i denna del)

Konceptuella modeller: beskrivning nära hur användare förstår data, används ofta vid databasdesign ex: E/R modellen

Implementationsmodeller: beskriver hur data organiseras i databasen. ex: relationsmodellen

Fysiska modeller: beskriver i detalj hur data är lagrad rent fysiskt.

Beskrivning av databas kallas databasens schema.

Databassystems arkitektur

Three-schema arkitektur

Externa scheman: beskriver änvändarvyerna
|
V
Konceptuella(logiska) schemat: beskriver hela databasen på en logisk nivå
|
V
Interna schemat: beskriver fysiska lagringsstrukturer och access-vägar

Dataoberoende

Logiskt dataoberoende: Går att göra ändring i det konceptuella(logiska) schemat utan att det påverkar de externa schemana eller applikationsprogrammet

Fysiskt dataoberoende: Går att göra ändringar i det interna schemat utan att det påverkar det konceptuella(logiska) schemat

Relationsmodellen

Tuple = rad
Attribut = kolumn
Domän kan liknas datatyp

varje attribut är definierat på en underliggande domän

antalet tupler kallas relationens kardinalitet
antalet attribut kallas relationens grad

Egenskaper hos en relation

- inga duplicerade tupler
- Tupler är oordnade
- Attribut är oordnade
- Alla värden är atomära(odelbara) = alltid i 1NF

Olika sorters relationer

Namngiven: relation som definierats i databasen
Basrelation: relation som inte är härledd, utan existerar “på riktigt”
Härledd relation: definierad i termer av andra relationer
Vy: namngiven härledd relation som är virtuell, existerar bara som ett uttryck
Frågeresultat: icke namngiven härledd relation som är ett resultat av att en specifik fråga exekveras

Vyer kan användas för att skapa logiskt dataoberoende(användare och applikationer inte påverkas av ändringar i databasens logiska struktur).

- tillåter att samma data kan ses ava olika användare på olika sätt samtidigt
- kan användas som “macron”
- kan användas för att gömma data (av säkerhetsskäl)

Behöver inte skriva om primärnyckel etc då vi använt det så mycket

Entitetsintegritet: Ingen del av primärnyckeln i en basrelation får vara NULL

Referensintegritet: Databasen får inte innehålla några icke matchande värden på främmandenycklar (dvs om B refererar till A, så måste A existera)

Semantiska integritetsregler(business rules): Regler som gäller en specifik databas, exempelvis “lön får inte vara under 10000 kr”.

Relationsalgebra

Select<cond>(R) = SQL ekvivalent: FROM R WHERE <cond>
Returnerar en relation som består av alla tupler från en relation som uppfyller ett visst villkor

Project<attr list>(R) = SQL ekvivalent SELECT <attr list> FROM R
Resulterande relationen har bara de attribut som specificeras i attributlistan
(Duplicerade tupler tas bort)

Union ∪ = Rel1 + Rel2 - dubbletter
Intersection ∩ = saker som finns i både Rel1 + Rel2
Difference = Rel1 - Rel2 (Rel1 - (Rel1 ∩ Rel2)

Cartesian Product = Rel1 X Rel2 alla rader gånger alla rader. Så om båda relationerna har 3 rader vardera så blir resulterande relationen 9 rader + att attributen läggs till så raderna blir längre

Theta Join = Cartesian Product med Select operation(VILLKOR)
Equijoin = Cartesian Product med Select operation(VILLKOR) där villkoret är “=”

Natural Join = Cartesian Product med Select operation(VILLKOR) där villkoret är “=” och tar bort dubbletten av join attributet(det attribut som är med i villkoret VarX = VarY)

Division:R1/R2. R1 måste innehålla 2 attribut, R2 måste innehålla 1 attribut.
Vilken/vilka tupler från det unika attribut i R1 innehåller alla tupler från attributet i R2
Alltså kolla vilka tupler i R1 som innehåller värdet från tuplen i R2

E/R Modellen

Entiteter: Existerar fysiskt, ex: bil,student. Existerar konceptuellt, ex: univ.kurs,jobb(Rita Rektangel)

Starka(står för sig själva) Svaga (beroende av andra entiteter)
Tips: leta efter substantiv i kravspecen

Entiteter har attribut, enkla, sammansatta, nyckelattribut etc.
Attribut(rita streck från Entiteten och en ellips/cirkel)

Samband
Tips: Leta verb i kravspecen
1-1
1-n
n-m

Översättning av E/R Diagram

Stark entitet blir basrelation, med sina tillhörande attribut
Mångvärdesattribut blir egen relation

1-n: Främmande nyckel på “många” sidan som refererar till sambandet på “en” sidan.

n-m: Blir en egen relation, Innehåller en främmandenyckel till vardera sida, primärnyckeln kan bli främmandenycklarna tillsammans eller så kan ett nytt attribut introduceras

SQL

Triggers och Procedurer
Procedurer: Typ som funktioner som kan kallas för att göra en viss uppgift.
Fördelar:

- Kan anropas av många applikationer
- Trafiken över nätverket kan minskas vid vissa tillfällen
- Kan användas för att kontrollera komplexa integritetsvillkor

Triggers: Regel som aktiveras vid en speciell händelse
Event: Insert, Update etc, Condition: självförklarande, Action: Kod/Kod+Procedur

Normalisering

VIKTIGT
Syftet med normalisering: Minska redundansen i den lagrade informationen

1NF: Relationens underliggande domäner endast innehåller atomära värden
Exempel: Attributet Color kan inte innehålla 2 värden på “samma ruta” som “Blue, Green” utan de måste då delas upp på 2 rader för att uppnå denna normalform.
2NF: Om relationen uppfyller 1NF och varje icke-nyckelattribut är till fullo funktionellt beroende av varje kandidatnyckel
3NF: Om relationen uppfyller 2NF och varje icke-nyckelattribut inte är till fullo funktionellt beroende av något annat icke-nyckelattribut.

BCNF: Om relationen är i 1NF och varje determinant är en kandidatnyckel.
Till fullo = icke reducerbar determinant.
Funktionell beroende: Från X kan vi veta vad A, B och C är. Exempelvis Reg.nr för en bil kan få reda på modell etc.

Alla attribut är funktionellt beroende av primärnyckeln(per definition)

Indexering

förmodligen inte det viktigaste

Primary index
- knutet till primärnyckel
- består av ett primärnyckelvärde och en pekare till ett block
- får lika många rader som aantalet diskblock datafilen består av
- är nondense(ej index till varje post)

Clustering indexes
- knutet till ett icke-nyckelfält som har dubbletter
- pekare från indexxxfilen för varje distinkt värde på clustering-fältet
- vanligt att man reserverar ett helt block(eller flera) för varje värde på clustering-fältet
- nondense

Secondary indexes
- indexfil med två värden, indexfällt och pekare till post/block
- en datafil kan ha månggaa sekundärindex, ett för varje attribut
- kan vara dense eller nondense
- tar mer plats och ger längre söktider än ett primärindex, men vinsten är större

Multilevel indexes
- Index med flera nivåer
- behöver en andra nivå bara om den första nivån kräver mer än ett block för disklagring

- behöver en tredje nivå bara om den andra nivån kräver mer än ett block för disklagring

B+träd
- är en form av sökträd som möjliggör en effektiv insättning och borttaagning aav poster
- vaarje nod i trädet motsvarar ett diskblock
- träden är alltid balaanserade, dvs allaa löv ligger på samma nivå(gör att accesstiderna blir förutsägbara
- i Bträd kan datapekare finnas i alla inre noder och i löven medan B+träd endast har datapekare i löven
- finns pekare mellan löven i b+träd vilket möjliggör sekventiell sökning

Transaktioner

Transaktion = en mängd operationer utförda i sekvens, måste utföras i sin helhet eller inte alls
commit = utförts korrekt och lagras nu permanent
rollback = fel uppstod och ändringarna “spolas tillbaka”

Egenskaper(ACID)
Atomicity = transaktion ska utföras helt eller inte alls
Consistency = transaktionen gör att databaser växlar mellan konsistenta tillstånd
Isolation = transaktionen skall inte påverkas av andra transaktioner
Durability = ändringar är permanenta

Problem

Lost update: Transaktion B läser av objekt X innan transaktion A kunnat skriva sin/sina uppdateringar på objekt X. Alltså går transaktion A’s uppdateringar på objekt X förlorade.

Dirty read(non repeatable read): Transaktion A läser objekt X vid 2 olika tillfällen och då Transaktion B har förändrat objekt X emellan så får dessa reads i transaktion A olika resultat

Incorrect summary: Transaktion A och Transaktion B hanterar samma objekt, då transaktion A kollar på ett förändrat objekt(förändrat av Transaktion B) konto2 men ett oförändrat konto3 pga timing. Så blir summorna fel. se F12_F13 PDFen för bilder

Konflikter i scheman(VIKTIGT)
Kommer förmodligen på tenta
Identifiera konflikter, rita precdensgrafer för att se om schema är serialiserbart.
uppnås om precedensgrafen saknar cykler

operationer är i konflikt om de uppfyller följande 3 villkor:

- Tillhör olika transaktioner
- Accessar samma objekt X
- Minst en av operationerna är en write(X)

Seriellt schema: en transaktion i taget. nästa börjar när den före är helt klar
Automatiskt seriellt: transaktionerna är oberoende av varandra

Metod:
Gör transaktionslista. exempelvis:
T1: r1(X), w1(X)
T2: r2(X)
T3:r3(X), w3(X)

Gör sedan en konflikt lista för de som uppfyller kraven i 3 punkts listan högre upp.
i detta fall: r1(X) och w3(X), w1(X) och r2(X), w1(X) och r3(X), w1(X) och w3(X), r2(X) och w3(X).

För att rita precedensgrafen, gör en nod för varje transaktion. i detta fall T1, T2 och T3
Vilken operation i konflikt listan ovan utförs före den andra, rita pil för varje konflikt
exempelvis om r1(X) görs innan w3(X) så ritar vi in pil från T1 till T3, men om w3(X) skulle göras innan r1(X) så ritar vi en pil från T3 till T1 istället.
Gå igenom varje konflikt och se om det uppstår en cykel i grafen. om det gör så är det schemat inte serialiserbart.

Concurrency control(låsning etc)

Shared/Exclusive (Read/Write) Locks. tänk mutex (mutual exclusion)

- En transaktion som vill göra en operation på ett objekt måste skaffa ett lås på det objektet
- om det inte beviljas så ställs transaktionen i väntekö
- 3 låsoperationer, read-lock(X), write-locked(X) och unlock(X)

Two phase locking protocol(2PL)
om alla transaktioner följer protokollet för 2PL så garanteras serialiserbarhet
2PL: alla låsningar föregår första unlock i transaktionen.

Återhämtnings strategier
Write ahead logging: innan en commit så loggas det så det finns “bevis” på att en commit kunde/har gjorts sedan får uppdateringen göras. detta kallas “Write ahead log rule” Om fel inträffar innan en transaktion gjort commit så görs alla uppdateringar “ogjorda”

Immediate update: uppdatering kan skrivas till databasen innan transaktionen gjort commit (steal)

Deferred update: skjuter upp eventuella uppdateringar till databasen tills dess att transaktionen avslutat sin exekvering och gjort commit (no steal)

Distribuerade databaser/Big Data/NoSQL

Meh bara läste lite på PDFen. Troligen inte stor del av tenta ändå

Säkerhet

Discretionary access control:

- Användare beviljas accessrättigheter till objekt i databasen på individuell basis.
- Gramt/Revoke privilegier på kontonivå
- privilegier kan specificeras på relationsnivå typ tillgång till SELECT, INSERT, UPDATE etc.
-

Mandatory access control:

- användare och objekt delas in i olika säkerhetsklasser
- används i daatabaser som har höga krav på säkerhet
- får bara läsrättigheter om användarens säkerhetsklass är större än eller lika med objektets
- får bara skrivrättigheter om användarens säkerhetsklass är mindre än eller lika med objektets

Audit trail:

- Speciell logg där detaljer om alla operationer som har utförts registreras