Baze de date distribuite

Bază de date centralizată – atât informații legate de structura unei baze de date, cât și datele efective care sunt stocate în tabelele ei, precum și alte obiecte (proceduri stocate, view-uri, funcții, triggere etc.) sunt stocate în aceeași locație

Bază de date distribuită – o bază de date care este "spartă" în mai multe fragmente, care sunt memorate în locuri diferite (pe calculatoare diferite)

Independența Datelor Distribuite

- atunci când interacționăm cu baza de date, faptul că baza respectivă e memorată pe un singur server, sau fragmente ale ei sunt memorate pe diferite servere, nu ar trebui să ne afecteze din punct de vedere al performanței, consistenței datelor, modului în care accesăm datele etc.

Atomicitatea Tranzacțiilor Distribuite

- păstrarea celor 4 proprietăți (ACID) este mai dificilă într-un context distribuit
- pot apărea mai multe evenimente care să impacteze fie atomicitatea, fie durabilitatea, fie consistența
- un SGBD distribuite este mai complex; acele module care se ocupă de gestionarea concurenței și de recuperarea datelor trebuie să funcționeze și într-un context distribuit

BDD - Avantaje:

- este foarte util să avem datele stocate pe locații diferite; este mai rapid de procesat și prelucrat
- informațiile trebuie să fie copiate pentru a putea fi disponibile
- modularitatea
- autonomia locală

BDD – Provocări:

Proiectarea bazelor de date:

- fragmentarea și alocarea – aceste decizii pot impacta performanța

Procesarea interogărilor distribuite:

- SGBD are o misiune dificilă de a găsi calea cea mai rapidă prin care să aducă datele către noi, prin interpretarea interogării scrise; la BDD, nu se mai ia în calcul doar costul de transfer de pagini între memoria internă și hard disk, ci se ia și costul de comunicare între servere diferite
- avantajul procesării paralele – când se execută o interogare, aceasta este spartă în interogări distincte, care se execută pe servere diferite în paralel

Controlul concurenței:

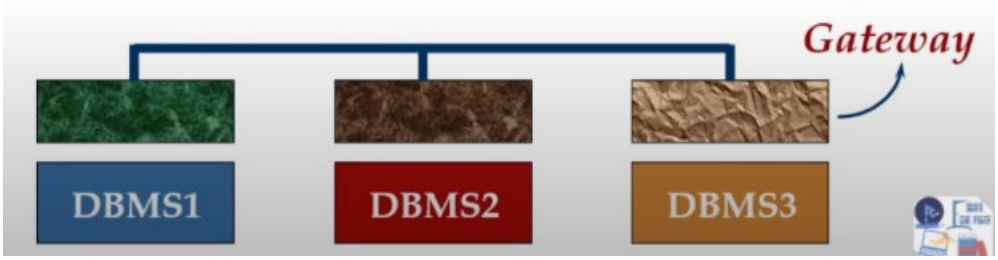
- se identifică mai greu faptul că a apărut un deadlock
- propagarea modificărilor – aceleași date pot fi stocate pe mai multe servere diferite; ne putem confrunta cu un caz de inconsistență a datelor

Păstrarea consistenței bazelor de date:

- sunt mai multe motive pentru care o tranzacție poate să eșueze: se oprește un server, se blochează o anumită conexiune între servere etc.
- sincronizarea datelor

Tipuri de baze de date distribuite:

- *SGBD singular*
  - există un singur proces corespunzător SGBD, care se comportă foarte asemănător cu modul în care se comporta cu o bază de date locală
  - e ca și cum totul ar fi stocat local, însă nu e așa
- *SGBD multiplu*
  - pe fiecare fragment avem servere diferite care acționează asupra lor
  - va trebui să existe un server principal (master) care comunică cu fiecare dintre acele servere locale
  - omogen – când vorbim de exact aceleași servere (de exemplu dacă avem SQL Server instalat în toate fragmentele)
  - eterogen – aceeași bază de date este gestionată de SGBD diferite (de exemplu, un fragment este gestionat de SQL Server, altul de MySql) → când vrem să accesăm date stocate în toate fragmentele trebuie să găsim un limbaj comun pentru toate serverele → **gateway**



Fragmentare:

- *Orizontală*
  - Primară – anumite înregistrări sunt stocate pe un site, alte înregistrări sunt stocate pe un alt site
  - Derivată – prin faptul că avem legături între tabele, și alte informații trebuie transferate (nu în totalitate)
- *Verticală*
  - anumite câmpuri dintr-o tabelă sunt puse pe un site, alte câmpuri sunt puse pe un alt site
  - pentru a putea regăsi informațiile care au fost dispersate pentru aceeași entitate, e nevoie de un punct de legătură (cheia primară), care trebuie să fie memorat în toate fragmentele
  - sunt executate niște join-uri → timp mai mare pentru a efectua operațiile

În practică, pe aceeași tabelă se poate găsi o combinație dintre cele două fragmentări.

Proprietăţi ale fragmentării:

$$R \Rightarrow \mathbf{F} = \{F_1, F_2, \ldots, F_n\}$$

Completitudine

$\forall x \in R, \exists F_i \in \mathbf{F}$ astfel încât $x \in F_i$

Disjunctivitate

$\forall x \in F_i, \neg\exists F_j$ astfel încât $x \in F_j, i \neq j$

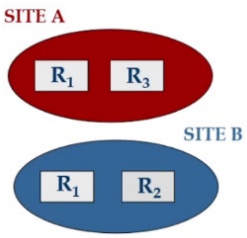
Reconstrucție

Există o funcție g astfel încât

$$R = g(F_1, F_2, \ldots, F_n)$$

Replicare:

- avantaje:
  - crește disponibilitatea datelor
  - putem avea o evaluare mai rapidă a interogărilor



- probleme:
    - propagarea modificărilor: sincron vs. asincron

Replicare sincronă:

- o modificare, odată realizată pe un fragment, se distribuie aproape imediat pe celelalte replici
- face actualizarea respectivă transparentă pentru utilizatori – se interacționează cu baza de date ca și cum ar fi o bază de date locală (puțin mai încet)

Replicare asincronă:

- se realizează o modificare, însă celelalte replici nu se actualizează imediat
- utilizatorii știu că la un moment dat, când accesează informațiile, nu sunt ultimele informații posibile
- se poate construi un sistem distribuit care să aibă o replicare sincronă, însă răspunsul vine mai târziu
- multe dintre produsele curente urmează această abordare

Tehnici de replicare sincronă:

A. Citește-orice / Modifică-tot (ROWA)
- dacă un fragment este replicat pe 10 servere, dacă vrem să citim informația din acel fragment, de obicei este adusă de pe server-ul cel mai apropiat (din motive de viteză)
- avantajează cititorul
- cel care scrie, va trebui să scrie pe toate cele 10 replici, ceea ce îl pune în dezavantaj
- se aplică în situațiile când citirile sunt multe, iar modificările sunt puține

B. Votare (consens al cvorumului)
- când modificăm un fragment, nu trebuie să actualizăm modificările pe toate cele 9 copii, ci doar pe o majoritate a lor
- nu se poate citi doar de pe o singură replică, deoarece nu avem garanția că vom citi ultimele date salvate → trebuie citite suficiente copii pentru a se asigura accesul la una dintre copiile recente
- nu e o abordare des utilizată

Replicare *Peer-to-Peer*:

- avem mai multe copii ale unui fragment (10), dintre care o parte sunt copii *master* – toate aceste copii sunt actualizate imediat când apare o modificare (3 din 10)
- modificările unei copii *master* trebuie să fie propagate către celelalte copii
- conflict: dacă pe 2 copii *master* se operează modificări asupra aceleași înregistrări în același timp? ! COLIZIUNE

- ce modificare se transmite copiilor secundare?
- transferul de modificări se face de la copia principală la copiile secundare (nu foarte frecvent, nu imediat)

Replicare cu *site* principal:

- doar o copie a unei tabele este considerată copie primară / *master*
- celelalte copii sunt *subscriberi*
- copiile secundare nu primesc notificare cu actualizarea, ele se actualizează independent la un moment dat, după o interogare la copia *master*
- se face în două feluri:
    - pe baza unui log:
        - la nivelul replicii principale există o tabelă specială, numită *Change Data Table*, unde se țin minte toate modificările care s-au operat pe fragment
        - modificările tranzacțiilor care se anulează trebuie înlăturate din CDT
        - se ia textul din tabelă și se re-execută tranzacțiile salvate pe copii
    - procedural:
        - din când în când se face câte un snapshot al bazei de date și se stochează într-o tabelă
        - când vine *subscriber*-ul, verifică dacă există vreun snapshot mai recent, ia ultimul snapshot și îl copiază peste baza de date