# Bab 7

# Basis Data Terdistribusi

**POKOK BAHASAN:**

- ✓ Pendahuluan
- ✓ Tipe Basis Data Terdistribusi
- ✓ Arsitektur Basis Data Terdistribusi
- ✓ Penyimpanan Data pada Sistem Terdistribusi
- ✓ Manajemen Katalog Terdistribusi
- ✓ Query Terdistribusi
- ✓ Joins pada DBMS Terdistribusi
- ✓ Optimasi Query pada DBMS Terdistribusi
- ✓ Mengubah Data Terdistribusi
- ✓ Locking pada Sistem Terdistribusi
- ✓ Distributed Recovery

**TUJUAN BELAJAR:**

Setelah mempelajari materi dalam bab ini, mahasiswa diharapkan mampu:

- ✓ Memahami perbedaan DBMS terdistribusi dan DBMS terpusat.
- ✓ Memahami arsitektur basis data terdistribusi.
- ✓ Memahami penyimpanan data, catalog data pada system terdistribusi.
- ✓ Memahami query, join dan optimasi query pada DBMS terdistribusi
- ✓ Memahami bagaimana mengubah data, melakukan locking data pada DBMS terdistribusi.
- ✓ Memahami bagaimana menangani kegagalan pada sistem terdistribusi

## 7.1 PENDAHULUAN

Pada basis data terdistribusi (*distributed database*), data disimpan pada beberapa tempat (*site*), setiap tempat diatur dengan suatu DBMS (*Database Management System*)

yang dapat berjalan secara independent. Properti yang terutama terdapat pada basis data terdistribusi :

- Independensi data terdistribusi : pemakai tidak perlu mengetahui dimana data berada (merupakan pengembangan prinsip independensi data fisik dan logika).
- Transaksi terdistribusi yang atomic : pemakai dapat menulis transaksi yang mengakses dan mengubah data pada beberapa tempat seperti mengakses transaksi local.

Untuk trend basis data terdistribusi saat ini, pemakai harus mengetahui dimana data ditempatkan, juga harus mengetahui dimana system yang tidak mendukung independensi data terdistribusi dan transaksi terdistribusi atomic. Kedua property tersebut harus mendukung system secara efisien. Untuk system terdistribusi yang bersifat global, properti-properti tersebut kemungkinan tidak tepat karena adanya administrasi yang terlalu berlebihan dalam membuat lokasi data yang transparan.

## 7.2 TIPE BASIS DATA TERDISTRIBUSI

Terdapat dua tipe basis data terdistribusi :

- Homogen : yaitu sistem dimana setiap tempat menjalankan tipe DBMS yang sama
- Heterogen : yaitu sistem dimana setiap tempat yang berbeda menjalankan DBMS yang berbeda, baik Relational DBMS (RDBMS) atau non relational DBMS. Gambaran basis data terdistribusi yang heterogen dapat dilihat pada Gambar 7-1.

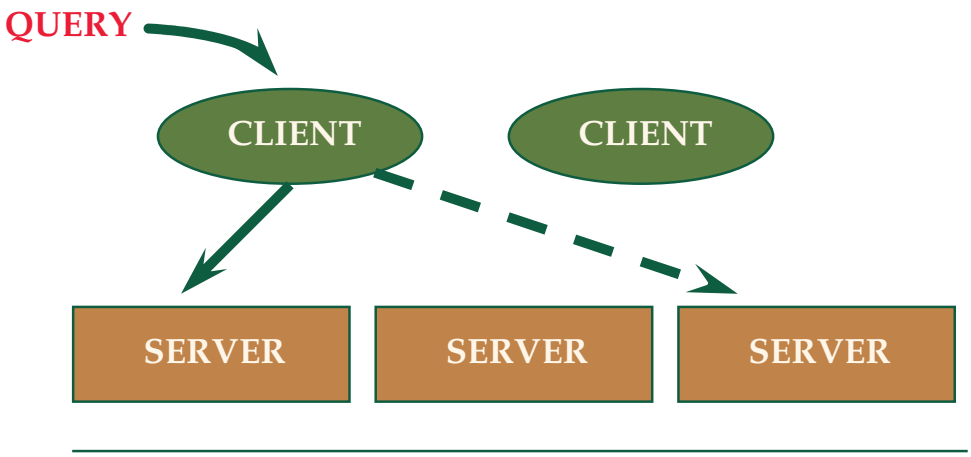

***Gambar 7-1:*** *Basis data terdistribusi heterogen*

## 7.3 ARSITEKTUR BASIS DATA TERDISTRIBUSI

Terdapat tiga pendekatan alternatif untuk membagi fungsi pada proses DBMS yang berbeda. Dua arsitektur alternatif DBMS terdistribusi adalah *Client/Server* dan *Collaboration Server.*

- *Client-Server*

  Sistem client-server mempunyai satu atau lebih proses client dan satu atau lebih proses server, dan sebuah proses client dapat mengirim query ke sembarang proses server seperti pada Gambar 7-2. Client bertanggung jawab pada antar muka untuk user, sedangkan server mengatur data dan mengeksekusi transaksi. Sehingga suatu proses client berjalan pada sebuah personal computer dan mengirim query ke sebuah server yang berjalan pada mainframe.




***Gambar 7-2:*** *Sistem client-server*

Arsitektur ini menjadi sangat popular untuk beberapa alasan. Pertama, implementasi yang relatif sederhana karena pembagian fungis yang baik dank arena server tersentralisasi. Kedua, mesin server yang mahal utilisasinya tidak terpengaruh pada interaksi pemakai, meskipun mesin client tidak mahal. Ketiga,

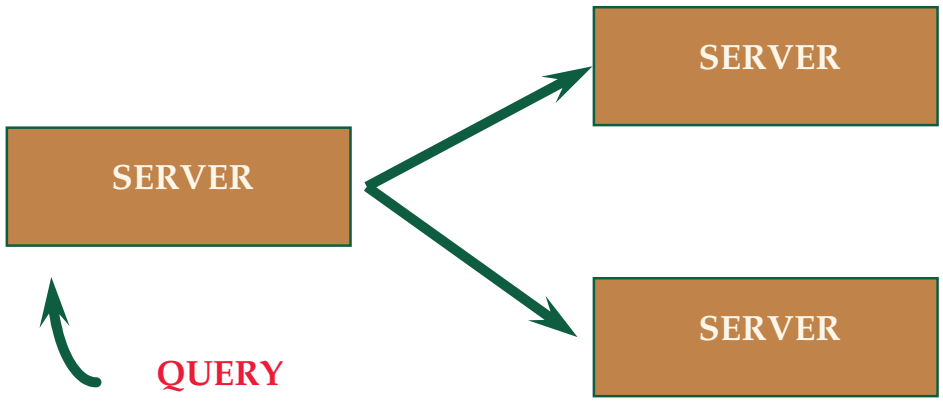
pemakai dapat menjalankan antarmuka berbasis grafis sehingga pemakai lebih mudah dibandingkan antar muka pada server yang tidak user-friendly.

Pada saat menulis aplikasi client-server, perlu diingat batasan antara client dan server dan untuk menjaga komunikasi antara keduanya yang berorientasi himpunan. Khususnya membuka kursor dan mengambil tupel pada satu waktu membangkitkan beberapa pesan dan dapat diabaikan.

- *Collaboration Server*

  Arsitektur client-server tidak mengijinkan satu query mengakses banyak server karena proses client harus dapat membagi sebuah quer ke dalam beberapa subquery untuk dieksekusi pada tempat yang berbeda dan kemudian membagi jawaban ke subquery. Proses client cukup komplek dan terjadi overlap dengan server; sehingga perbedaan antara client dan server menjadi jelas. Untuk mengurangi perbedaan diguankan alternatif arsitektur client-server yaitu sistem *Collaboration Server*. Pada sistem ini terdapat sekumpulan server basis data, yang menjalankan transaksi data lokal yang bekerjasama mengeksekusi transaksi pada beberapa server seperti pada Gambar 7-3..

  Jika server menerima query yang membutuhkan akses ke data pada server lain, sistem membangkitkan subquery yang dieksekusi server lain dan mengambil hasilnya bersama-sama untuk menggabungkan jawaban menjadi query asal.



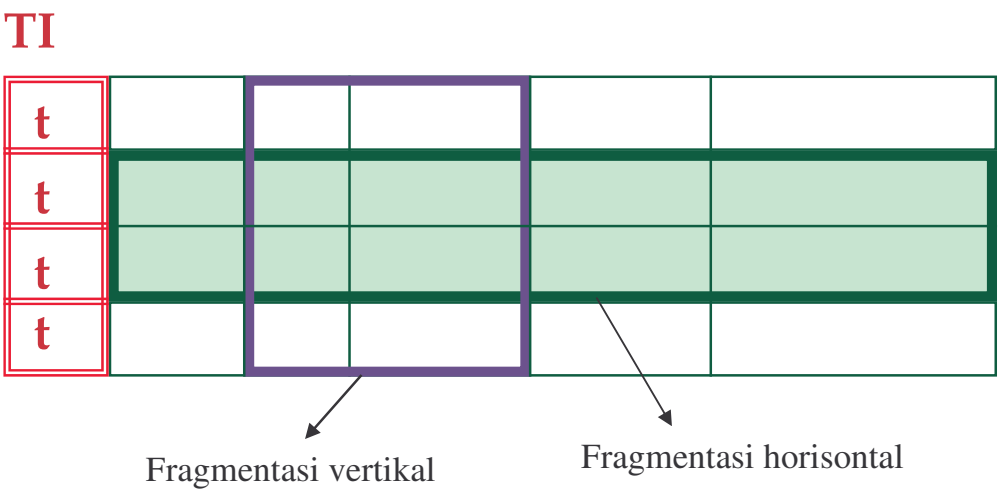

***Gambar 7-3:*** *Collaboration System*

## 7.4 PENYIMPANAN DATA PADA SISTEM TERDISTRIBUSI

Pada DBMS terdistribusi, relasi disimpan pada beberapa tempat. Pengaksesan relasi yang disimpan pada *remote side* mengakibatkan biaya melewatkan pesan dan untuk menguranginya, sebuah relasi dipartisi atau difragmentasi ke beberapa tempat, dengan fragmen dikirim pada tempat dimana fragmen tersebut sering diakses, atau replika pada pada setiap tempat dimana relasi menjadi kebutuhan yang tinggi

- Fragmentasi

  Fragmentasi terdiri dari relasi yang dibagi ke relasi atau fragmen yang lebih kecil dan mengirim fragmen, pada beberapa tempat. Terdapat dua macam fragmentasi, fragmentasi horizontal dan fragmentasi vertikal. Pada fragmentasi horisontal, setiap fragmen terdiri dari sebuah subset baris dari relasi asal. Pada fragmentasi vertikal, setiap fragment terdiri dari sebuah subset kolom dari relasi asal. Fragmentasi horisontal dan vertikal diilustrasikan pada Gambar 7-4.




***Gambar 7-4:*** *Fragmentasi horisontal dan vertikal*

Bila sebuah relasi difragmentasi, harus meliputi relasi asal dari fragmen :

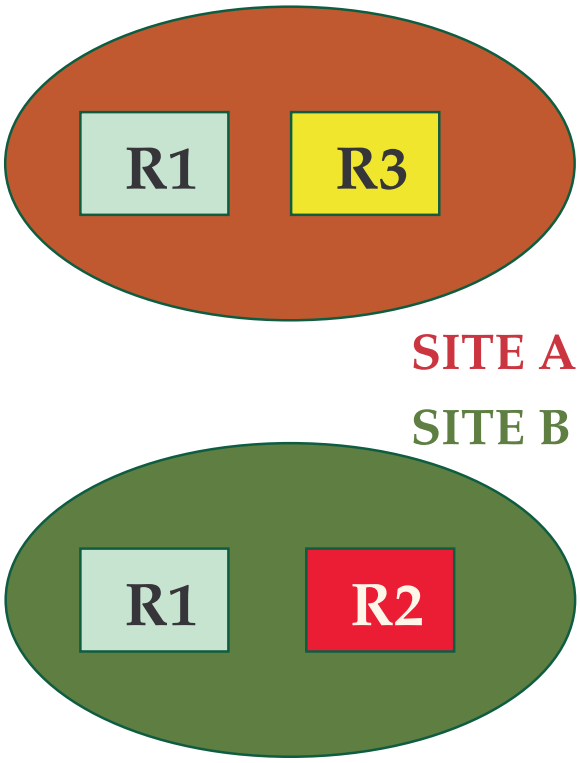
- Fragmentasi horisontal : union dari fragmen horisontal harus sama dengan relasi asal. Fragmen biasanya dibutuhkan disjoint.

- Fragmentasi vertikal : koleksi fragmen vertikal seharusnya dekomposisi *lossless-join.*

Untuk menjamin fragmentasi vertikal *lossless-join*, sistem harus menyediakan id tupel yang unik untuk setiap tupel dalam relasi asli. Jika kita berpilir bahwa relasi asal sebagai field yang berisi tambahan tupel-id sebagai kunci, field ini ditambahkan ke setiap fragmen vertikal. Sehingga dekomposisi dijamin *lossless-join*.

- Replikasi

Replikasi berarti bahwa kita menyimpan beberapa copy sebuah relasi atau fragmen relasi. Keseluruan relasi dapat direplikasi pada satu atau lebih tempat. Sebagai contoh, jika relasi R difragmentasi ke R1, R2 dan R3, kemungkinan terdapat hanya satu copy R1, dimana R2 adalah replikasi pada dua tempat lainnya dan R3 replikasi pada semua tempat. Hal ini dapat diilustrasikan pada Gambar 7-5.




***Gambar 7-5:*** *Replikasi*

Motivasi untuk replikasi adalah :

- Meningkatkan ketersediaan data : Jika sebuah tempat yang berisi replika melambat, kita dapat menemuka data yang sama pada tempat lain. Demikian pula, jika copy lokal dari relasi yang diremote tersedia, maka tidak terpengaruh saluran komunikasi yang gagal.
- Evaluasi query yang lebih cepat : query dapat mengeksekusi lebih cepat menggunakan copy local dari relasi termasuk ke remote site.

## 7.5 MANAJEMEN KATALOG TERDISTRIBUSI

Menyimpan data terdistribusi pada beberapa tempat dapat menjadi sangat kompleks. Kita harus menyimpan data bagaimana relasi difragmentasi dan replikasi, bagaimana fragmen relasi didistribusikan ke beberapa tempat dan dimana kopi dari fragmen disimpan. Nama setiap replika dari setiap fragmen harus ada. Untuk menyediakan otonomo lokal digunakan format sebagai berikut :

**<local-name, birth-site>**

Katalog setiap tempat menggambarkan semua obyek (fragmen, replika) pada suatu tempat dan menyimpan data replika dari relasi yang dibuat pada tempat tersebut. Untuk menemukan relasi, lihat pada katalog **birth-site**. **Birth-site** tidak pernah berubah meskipun relasi dipindahkan.

## 7.6 QUERY TERDISTRIBUSI

Misalnya pada dua relasi :

Sailors(*sid:* integer, *sname:* string, *rating:* integer, *age:* real)
Reserves(*sid:* integer, *bid:* integer, *day: date*, *rname:* string)

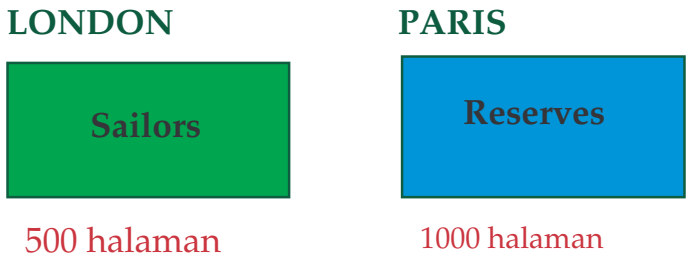
Kemudian dilakukan query berikut :

SELECT AVG(S.age) FROM Sailors S WHERE S.rating > 3 AND S.rating < 7

- Fragmentasi horisontal : tupel dengan rating < 5 pada Shanghai, >= 5 pada Tokyo. Harus menghitung SUM(age), COUNT(age) pada kedua tempat. Jika WHERE berisi hanya S.rating>6, maka hanya satu tempat.
- Fragmentasi vertikal : *sid* dan *rating* pada Shanghai, *sname* dan *age* pada Tokyo, *tid* pada kedua tempat. Harus melakukan rekonstruksi relasi dengan join pada *tid* kemudian mengevaluasi query.
- Replikasi : Sailor di-copy kan pada kedua tempat.

## 7.7 JOINS PADA DBMS TERDISTRIBUSI

Sebagai contoh, London menyimpan 500 halaman Sailor dan Paris mempunyai 1000 halaman Reserves seperti Gambar 7-6.




***Gambar 7-5:*** *Contoh Sistem Terdistribusi*

Untuk menangani joing pada DBMS terdistribusi harus diperhatikan hal-hal berikut :

- Ambil halaman sesuai kebutuhan, pada suatu nested loop join di London dengan Sailor sebagai outer dan setiap halaman Sailor, ambil semua halaman Reserver dari Paris. Jika halaman Reserves di London diambil sampai join selesai, halaman akan diambil sekali, tetapi asumsikan bahwa halaman Reserves tidak diambil semul sampai selesai, maka biaya akan menjadi mahal.
  - **Biaya :** 500 D + 500 * 1000 (D+S)
  - **D** adalah biaya membaca/menulis halaman; **S** adalah biaya pengiriman halaman.

  - Jika query tidak dikirim ke London, harus menambah biaya hasil pengiriman ke query site.
  - Dapat juga mengerjakan inner nested loop di London, ambil tupelo Reserves yang sesuai ke London sesuai kebutuhan.
- Pengiriman ke satu tempat : Kirimkan Reserves ke London.
  - Biayat: 1000 S + 4500 D (Semi Join; biaya = 3*(500+1000))
  - Jika ukuran hasil sangat besar, lebih baik mengirimkan kedua relasi ke result site dan kemudian lakukan join.

**Semijoin**

Teknik semijoin ditujukan untuk mengurangi jumlah tupelo Reserves yang dikirim. Idenya terdiri dari langkah-langkah berikut :

1. Di London, proyeksikan Sailors ke kolom join dan kirimkan ke Paris.
2. Di Paris, lakukan join pada proyeksi Sailors dengan Reserves. Hasil join disebut reduksi dari Reserves dengan Sailors
3. Kirimkan reduksi Reserves ke London
4. Pada London, join Sailors dengan Reserves yang sudah direduksi

Ide dari langkah semijoin tersebut adalah mengurangi biaya komputasi dan proyeksi pengiriman dan komputasi dan proyeksi pengiriman untuk biaya pengiriman relasi Reserves penuh. Semijoin terutama bermanfaan jika terdapat sebual seleksi pada Sailors dan jawaban tersedia di London.

**Bloomjoin**

Teknik bloomjoin juga ditujukan untuk mengurangi jumlah tupelo Reserves yang dikirim. Idenya terdiri dari langkah-langkah berikut :

1. Di London, hitung sebuat bit-vector dari beberapa ukuran k. Nilai kolom hash join mempunyai jangkauan 0 sampai k-1. Jika beberapa tupel melakukan teknik hashing ke I, set bit I menjadi 1 (I dari 0 sampai k-1). Kirimkan bit-vector ke Paris
2. Di Paris, lakukan hashing setiap tupel Reserves dengan cara yang sama dan abaikan tupel yang melakukan hashing ke 0 di bit-vector Sailors. Hasilnya disebut reduksi Reserves with Sailors.

3. Kirimkan bit-vector hasil reduksi Reserves ke London
4. Di London, join Sailors dengan hasil reduksi Reserves
5. Bit-vector biaya pengirimannya lebih murah, juga lebih efektif

## 7.8 OPTIMASI QUERY PADA DBMS TERDISTRIBUSI

Untuk optimasi query pada sistem terdistribusi, menggunakan pendekatan biaya, misalnya pada semua plan, mengambil yang termurah, sama dengan optimasi tersentralisasi. Perbedaan optimasi query pada sistem terdistribusi dan sistem tersentralisasi, pertama, biaya komunikasi harus dipertimbangkan. Kedua, otonomi tempat lokal harus diperhatikan. Ketiga, menggunakan metode join terdistribusi yang baru.

Query site membangun daerah global, dengan daerah local menggambarkan pemrosesan pada setiap tempat. Jika sebuah tempat dapat melakukan improvisasi pada daerah lokal, dapat dilakukan dengan bebas.

## 7.9 MENGUBAH DATA TERDISTRIBUSI

Untuk melakukan pengubahan data terdistribusi, dilakukan replikasi transaksi yang dapat dilakukan dengan cara :

- Synchronous Replication : semua copy dari relasi yang dimodifikasi (fragmen) harus diubah sebelum modifikasi transaksi commit. Distribusi data dibuat transparan ke pemakai.
- Asynchronous Replication : Copy dari sebuah relasi yang dimodifikasi hanya diubah secara periodik, copy yang berbeda akan keluar dari sinkronisasi. User harus waspada pada distribusi data. Produk saat ini mengikuti pendekatan ini.

**Synchronous Replication**

Terdapat dua teknik dasar untuk menjamin transaksi terlihat nilai yang sama dengan copy, yaitu :

- Voting : transaksi harus menulis mayoritas copy untuk memodifikasi sebuah obyek, harus membaca cukup copy untuk meyakinkan bahwa terlihat setidaknya

satu dari copy saat itu. Misalnya terdapat 10 copy, 7 penulisan untuk perubahan dan 4 copy untuk pembacaan. Setiap copy mempunyai nomor versi. Teknik ini biasanya tidak atraktif karena pembacaan adalah hal yang biasa.

- Read-any Write-all: penulisan lebih lambah dan pembacaan lebih cepat daripada teknik Voting. Teknik ini banyak digunakan pada synchronous replication

Pemilihan teknik synchronous replication akan menentukan tempat mana yang terkunci untuk seting.

Biaya pada synchronous replication adalah sebagai berikut : sebelum transaksi yang diubah commit, harus dilihat penguncian pada semua copy yang dimodifikasi. Kirimkan perintah lock ke remote site, dan sementara menunggu respon, pegang kunci yang lain. Jika tempat atau saluran gagal, transaksi tidak dapat commit sampai transaksi kembali. Meskipun tidak terjadi kegagalan, commit harus mengikuti commit protocol dengan beberapa pesan yang mahal. Karena itu alternative teknik asynchronous replication banyak digunakan.

**Asynchronous Replication**

Asynchronous replication mengijinkan memodifikasi transaksi commit sebelum semua copy diubah (dan pembaca tidak hanya melihat satu copy). Pemakai harus waspada copy yang keluar dari sinkronisasi untuk suatu periode waktu yang pendek.

Teknik asynchronous replication menggunakan dua pendekatan, yaitu Primary Site dan Peer to Peer replication. Perbedaan kedua teknik ini terletak pada berapa banyak copy yang dapat diubah atau copy master.

- Peer to Peer replication.

  Lebih dari satu copy dari suatu obyek dapat menjadi sebuah master. Perubahan ke copy master harus dipropaganda ke copy lain dengan cara yang berlainan. Jika dua copy master diubah dan terjadi suatu konflik, konflik harus dipecahkan (misalnya Tempat 1 : umur Joe mengubah 35, Tempat 2 : mengubah 36. Teknik ini bagus digunakan jika konflik tidak terjadi, misalnya setiap tempat master memiliki fragmen disjoin dan yang memiliki hak pengubahan dimiliki oleh satu master pada satu waktu.

- Primary Site replidation.

  Tepat satu copy dari suatu relasi digunakan sebagai primary copy atau master copy. Replika pada tempat lain tidak langsung diubah. Primary copy dipublikasikan. Tempat lain menjalankan (fragmen) ke relasi ini, terdapat beberapa copy sekunder. Isu utama adalah bagaimana pengubahan primary copy dapat dipropaganda ke copy sekunder ? Hal in idapat dilakukan dalam dua langkah. Langkah pertama ambil pengubahan yang dibuat dengan transaksi commit, kemudian aplikasikan perubahan tersebut. Exactly one copy of a relation is designated the primary or master copy. Replicas at other sites cannot be directly updated.

**Implementasi Primary Site pada Asynchronous Replication**

Isu utama dalam mengimplementasikan primary site replication adalah menentukan barapa banyak perubahan ke primary copy dipropaganda ke copy sekunder. Perubahan biasanya dipropaganda dalam dua langkah yaitu Capture dan Apply. Perubahan dibuat dengan transaksi commit ke primary copy yang diidentifikasi selama langkah Capture dan dipropaganda ke copy sekunder selama langkah Apply.

*Mengimplementasikan Langkah Capture*

Langkah Capture diimplementasikan dengan satu dari dua pendekatan, yaitu Log-Based Capture dan Procedureal Capture.

- Log-Based Capture : log (menyimpan recovery) digunakan untuk membangkitkan Change Data Table (CDT). Jika hal ini dikerjakan ketika log terakhir ditulis ke disk, harus menghapus perubaan ke subsequent yang dihentikan transaksi.
- Procedural Capture: suatu prosedur yang secara otomatis dibangkitkan (trigger) mengerjakan capture.

Implementasi capture dengan Log-Based Capture lebih baik karena lebih murah dan lebih cepat tetapi harus memahami detail dari property log.

*Mengimplementasikan Langkah Apply*

Proses Apply pada tempat sekunder secara periodic mengakibatkan perubahan ke table CDT dari primary site, dan mengubah copy. Periode didefinisikan oleh timer

atau pemakai/aplikasi. Replika dapat dipandang lebih dari relasi yang dimodifikasi. Jika hal ini terjadi, replica terdiri dari pengubahan pandangan material yang naik sebagai perubahan relasi.

Log-Based Capter ditambah Apply yang terus-menerus akan meminimalkan delay pada propaganda perubahan. Procedureal Capture ditambah application-driven Apply merupakan cara yang fleksibel untuk perubahan proses.

**Data Warehousing : Sebuah contoh Replication**

Trend yang berkembang saat ini adalah membangun "warehouses" data yang sangat besar dari beberapa tempat. Hal ini memungkinkan untuk query pendukung keputusan yang kompleks dari data pada keseluruhan organisasi. Warehouse dapat dipandang sebagai instance dari asynchronous replication. Data sumber biasanya dikontrol dengan DBMS yang berbadi, penekanannya pada cleaning data dan menghapus kesalahan pada pembuatan replikasi. Prosedur Capture dan aplikasi Apply baik untuk lingkungan ini.

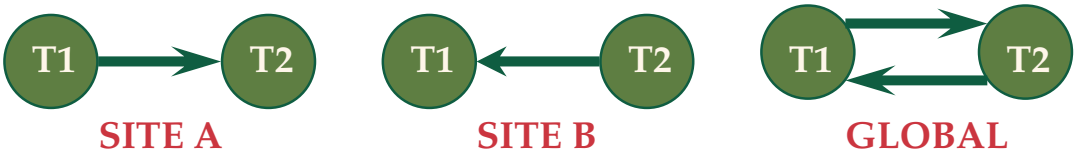
## 7.10 LOCKING PADA SISTEM TERDISTRIBUSI

Untuk menangani penguncian obyek pada beberapa tempat digunakan cara :

- Sentralisasi : satu tempat melakukan semua penguncian dan membuka kunci untuk semua obyek
- Primary Copy : semua penguncian untuk suatu obyek dikerjakan pada tempat primary copy dari obyek tersebut. Untuk pembacaan membutuhkan akses ke tempat terkunci sebaik tempat dimana obyek disimpan.
- Terdistribusi penuh : penguncian untuk suatu copy dilakukan pada tempat dimana copy disimpan. Hal in akan mengunci semua tempat pada saat menulis obyek.

**Distributed Deadlock Detection**

Setiap tempat menangani local wait-for graph. Deadlock global akan terjadi jika local graph tidak membentuk siklus seperti dapat dilihat pada Gambar 7-6 menunjukkan terjadi deadlock global.




***Gambar 7-6:*** *Deadlock Global*

Untuk mendeteksi deadlock digunakan tiga solusi yaitu : Sentralisasi, yaitu mengirim semua local graph ke satu tempat. Hierarki yaitu mengorganisasi tempat ke dalam suatu hirarki dan mengirim local graph ke parent dari hirarki. Timeout yaitu menghentikan transaksi jika menunggu terlalu lama.

## 7.11 DISTRIBUTED RECOVERY

Proses pemulihan pada DBMS terdistribusi lebih kompleks daripada pada DBMS tersentralisasi karena sebab berikut :

- Terjadi kegagalan yang baru, misalnya saluran komunikasi dan remote site.
- Jika sub transaksi dari suatu transaksi mengeksekusi tempat yang berbeda, semua atau tidak ada yang harus commit. Hal ini memerlukan commit protocol untuk menangani hal tersebut.

Suatu log ditangani pada setiap tempat, sebagaimana pada DBMS tersentralisasi dan aksi commit protocol ditambahkan pada log.

**Two-Phase Commit (2PC)**

Tempat dimana transaksi asal disebut koordinator, sedangkan tempat lain dimana transaksi mengeksekusi disebut sub ordinat. Jika sebuah transaksi ingin commit maka coordinator mengirim pesan **prepare** ke semua sub ordinat. Sub ordinat memaksa menulis rekaman log **abort** atau **prepare** dan mengirim suatu pesan **no** or **yes** ke coordinator. Jika coordinator menerima pesan yes, maka coordinator memaksa menulis suatu rekaman log **commit** dan mengirim pesan **commit** ke semua sub ordinat. Sebaliknya, memaksa menulis rekaman log **abort** dan mengirim pesan **abort**. Sub Ordinat memaksa menulis rekaman log **abort/commit** berdasarkan pesan yang didapat,

kemudian mengirim pesan **ack** ke koordinator. Koordinator menulis rekaman log **end** setelah mendapatkan semua pesan ack.

Pada 2PC terdapat dua bentuk komunikasi, pertama voting kemudian terminasi. Kedua bentuk komunikasi diinisialisasi oleh coordinator. Setiap tempat dapat memutuskan untuk abort suatu transaksi. Setiap pesan merupakan keputusan oleh pengirim, untuk menjamin bahwa keputusan bertahan dari kegatalah. Untuk itu pertama kali disimpan pada local log. Semua commit protocol menyimpan log untuk suatu transaksi yang terdiri dari id transaksi dan id coordinator. Rekaman abort/commit oleh coordinator juga termasi id semua sub ordinat.

**Restart Setelah Suatu Kegagalan pada Suatu Tempat**

- Jika kita mempunyai rekaman log commit atau abort untuk transaksi T, tetapi tidak mempunyai rekaman end, maka harus redo/undo T. Jika tempat ini adalah coordinator untuk T, tetap kirimkan pesan **commit/abort** ke sub ordinat sampai diterima pesan **ack**.
- Jika kita menyiapkan rekaman log transaksi T, tetapi tidak commit/abort, tempat ini adalah sub ordinat untuk T. Sub ordinat terus menghubungi coordinator untuk menemukan status T, kemunian menulis rekaman log **commit/abort** redo/undo T dan menulis rekaman log **end.**
- Jika kita tidak meyiapkan rekaman log untuk T, secara sepihak akan abort dan undo T. Tempat ini kemungkinan coordinator. Jika ya, sub ordinat mungkin mengirim pesan

Jika coordinator untuk transaksi gagal, sub ordinat yang memilih yes tidak dapat memutuskan apakah commit atau abort T sampai coordinator pulih. Pada saat itu T akan diblok. Meskipun jika semua sub ordinat saling mengetahui (melebihi yang tersedia pesan) maka transaksi adakn diblok sampai satu dari sub ordinat memilih no.

Jika suatu remote site tidak merespon selama commit protocol untuk transaksi T, aik karena tempat gagal atau saluran gagal :

- Jika tempat tersebut adalah coordinator untuk T, maka abort T
- Jika tempat tersebut adalah sub ordinat, dan tidak memilih yes, maka harus abort T

- Jika tempat tersebut sub ordinat dan memilih yes, maka akan diblok sampai coordinator me-respon.

Pesan ack digunakan untuk menyampaikan coordinator bahwa koordinat dapat melupakan suatu transaksi, sampai coordinator menerimas semua ack, harus menyimpan T pada table transaksi.

- Jika coordinator gagal setelah mengirim pesan siap tetapi sebelum menulis rekaman log commit/abort, jika coordinator kembali maka akan abort transaksi
- Jika sub transaksi tidak diubah, status commit atau abort tidak relevan.

Apabila coordinator menghentikan (abort) T, koordiator tidak mengerjakan T dan menghapusnya dari table transaksi segera. Jadi coordinator tidak menunggu **acks,** "presumes abort" jika transaksi tidak di dalam table transaksi. Nama sub ordinat tidak disimpan dalam rekaman log **abort**. Sub ordinat tidak mengirim ack pada abort. Jika sb transaksi tidak diubah, sub ordinat merespon pesan prepare dengan menulis yes/no. Koordinat mengabaikan pembaca. Jika semua sub transaksi adalah pembaca, fase kedua tidak diperlukan.

## RINGKASAN:

- Pada basis data terdistribusi, data disimpat pada beberapa lokasi dengan tujuan untuk membuat distribusi yang transparan. Pada basis data terdistribusi, *distributed data independence* (pemakai tidak perlu mengetahui lokasi data ) dan *distributed transaction atomicity* (dimana tidak ada perbedaan antara transaksi terdistribusi dan transaksi local). Jika semua lokasi menjalankan perangkat lunak DBMS yang sama, system disebut homogen, selain itu disebut heterogen.
- Arsitektur sistem basis data terdistribusi terdapat tiga tipe. Pada system Client-Server, server menyediakan fungsi DBMS dan client menyediakan antar muka pemakai. Pada Collaboration system system, tidak terdapat perbedaan antara proses client dan server.
- Pada DBMS terdistribusi, suatu relasi difragmentasi dan direplikasi pada beberapa tempat. Dalam fragmentasi horizontal, setiap partisi terdiri dari himpunan baris dari relasi asal. Dalam fragmentasi vertika, setiap partisi terdiri dari himpunan kolom pada relasi asal. Pada replikasi, disimpan beberapa copy dari relasi atau suatu partisi pada beberapa tempat.

- Jika suatu relasi difragmen dan direplika, setiap partisi memerlukan nama global yang unik yang disebut relation name. Manajemen catalog terdistribusi diperlukan untuk menyimpan rekaman dimana data disimpan.
- Pada pemrosesan query dalam DBMS terdistribusi, lokasi partisi dari relasi perlu dihitung. Join dua relasi dapat dilakukan dengan mengirim satu relasi ke tempat lain dan membentuk local join. Jika join melibatkan kondisi seleksi, jumlah tupel yang diperlukan kemungkinan kecil. Semijoin dan Bloomjoin mengurangi jumlah tupel yang dikirim ke jaringan dengan mengirim informasi terlebih dahulu yang mengijinkan mem-filter tupel yang tidak relevan. Optimasi query pada system terdistribusi harus mempertimbangkan komunikasi dengan model biaya.
- Pada synchronous replication, semua copy dari relasi replica diubah sebelum transaksi commit. Pada asynchronous replication, copy hanya diubah secara periodic. Terdapat dua teknik untuk menjamin synchronous replication. Secara voting, perubahan harus menulis mayoritas copy dan membaca harus mengakses cukup copy untuk menjamin bahwa satu copy sudah tersedia. Pada replikasi peer-to-peer, lebih dari satu copy dapat diubah dan strategi conflict resolution dapat mengubah konflik yang terjadi. Pada replikasi primary site, terdapat satu primary copy yang dapat diubah, copy sekunder lain tidak dapat diubah. Pengubahan pada primary copy dipropaganda menggunakan capter dan kemudian apply ke tempat lain.
- Jika suatu transaksi melibatkan aktivitas pada tempat yang berbeda, maka memanggil aktivitas sub transaksi.
- Pada DBMS terdistribusi, manajemen lock berupa lokasi sentral, primary copy atau terdistribusi penuh. Deteksi deadlock pada system terdistribusi dibutuhkan.
- Pemulihan pada DBMS terdistribusi dilakukan menggunakan commit protocol yang mengkoordinasi aktivitas pada tempat yang berbeda yang dilibatkan pada transaksi. Pada Two-Phase Commit, setiap transaksi didesain oleh tempat coordinator. Sub transaksi dieksekusi pada tempat sub ordinat. Protokol menjamin bahwa perubahan dibuat oleh beberapa transaksi dapat dipulihkan. Jika tempat coordinator bertabrakan, sub ordinat di blok, dan sub ordinat harus menunggu coordinator pulih.

## LATIHAN SOAL :

1. Apakah keuntungan DBMS terdistribusi dibandingkan dengan DBMS tersentralisasi?
2. Gambarkan arsitektur Client-Server dan Collaboration-Server.
3. Pada arsitektur collaboration server, jika suatu transaksi dikirim ke DBMS, akan digambarkan bagaimana aktivitas tempat yang berbeda dikoordinasi. Secara khusus, gambarkan aturan manager transaksi pada tempat berbeda, konsep atomic transaksi terdistribusi.
4. Definisikan fragmentasi dan repkasi dalam hal dimana data disimpan.
5. Apakah perbedaan antara replikasi synchronous dan asynchronous ?
6. Definisikan distributed data independence.
7. Bagaimana teknik voting dan read-one write-all diimplementasikan pada replikasi synchronous ?
8. Berikan penjelasan bagaimana asynchronous replication diimplementasikan. Khususnya, jelaskan maksud capture dan apply.
9. Apakah perbedaan antara log-based dan procedureal untuk implementasi capture?
10. Mengapa pemberian nama unik pada obyek basis data lebih kompleks pada DBMS terdistribusi ?