UNTAR
Universitas Tarumanagara

Terakreditasi BAN PT A (Unggul)

UNTAR untuk INDONESIA

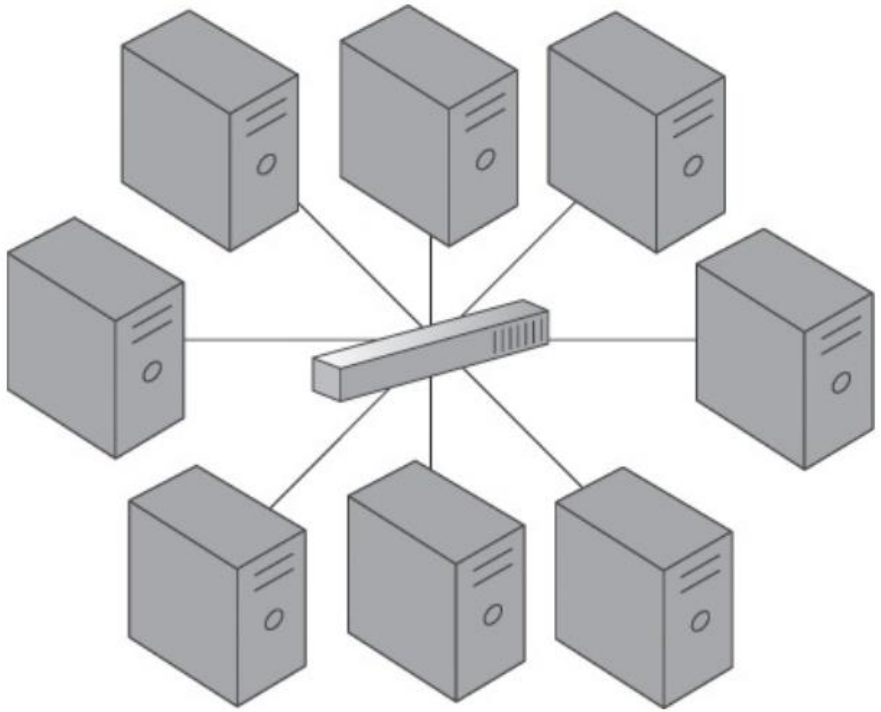
Konsep Penyimpanan

BIG DATA – TK13025

Clusters

- Cluster adalah kumpulan server, atau node yang digabungkan erat.
- Server ini biasanya memiliki spesifikasi perangkat keras yang sama dan terhubung bersama melalui jaringan untuk bekerja sebagai satu unit
- Setiap node dalam cluster memiliki sumber daya tersendiri, seperti memori, prosesor, dan hard drive.
- Sebuah cluster dapat menjalankan tugas dengan membaginya menjadi potongan-potongan kecil dan mendistribusikan eksekusi mereka ke komputer yang berbeda milik cluster.

Clusters



Simbol yang digunakan untuk mewakili sebuah cluster.

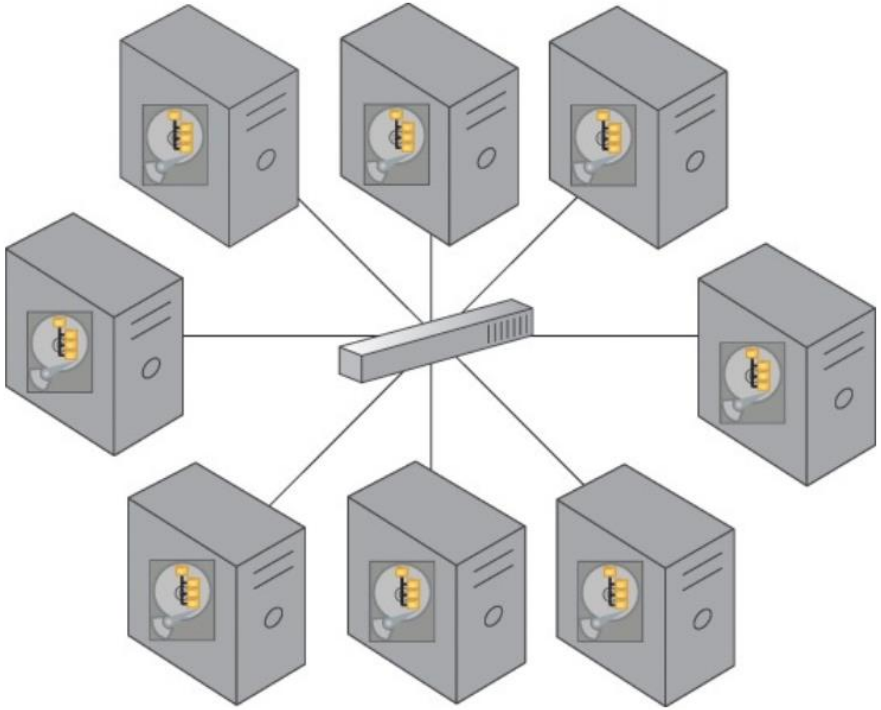
Sistem File dan Sistem File Terdistribusi

- Sistem file adalah metode menyimpan dan mengatur data pada perangkat penyimpanan, seperti flash drive, DVD, dan hard drive.
- File adalah unit penyimpanan atom yang digunakan oleh sistem file untuk menyimpan data.
- Sistem file menyediakan tampilan logis dari data yang disimpan di perangkat penyimpanan dan menyajikannya sebagai struktur pohon direktori dan file seperti yang digambarkan.
- Sistem operasi menggunakan sistem file untuk menyimpan dan mengambil data atas nama aplikasi.
- Setiap sistem operasi menyediakan dukungan untuk satu atau lebih sistem file, misalnya NTFS di Microsoft Windows dan ext di Linux.

Sistem File dan Sistem File Terdistribusi

- Sistem file terdistribusi adalah sistem file yang dapat menyimpan file besar yang tersebar di seluruh node cluster.
- Bagi klien, file tampak lokal; namun, ini hanya pandangan logis karena secara fisik file didistribusikan ke seluruh cluster.
- Tampilan lokal ini disajikan melalui sistem file terdistribusi dan memungkinkan file diakses dari berbagai lokasi.
- Contohnya termasuk Google File System (GFS) dan Hadoop Distributed File System (HDFS).

Sistem File dan Sistem File Terdistribusi



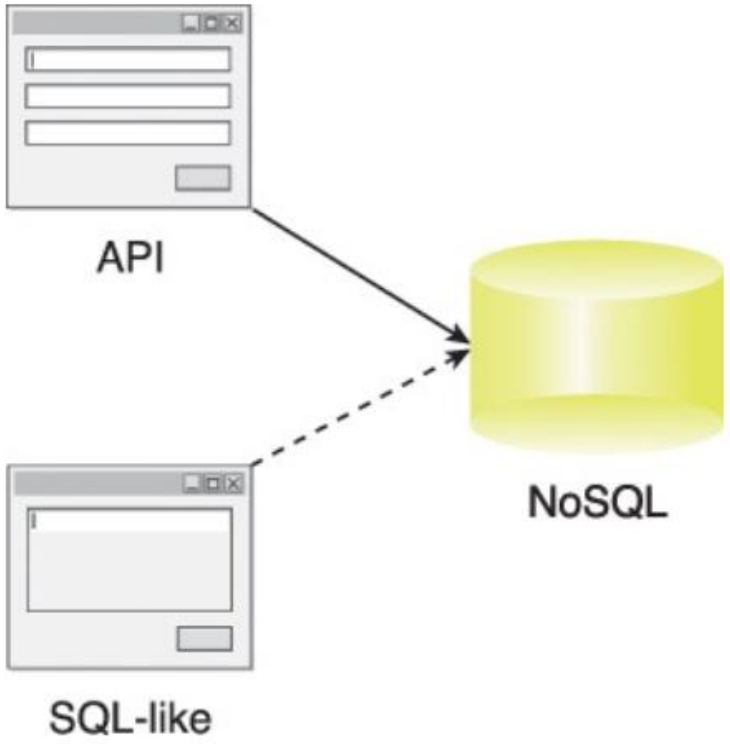
Simbol yang digunakan untuk mewakili sistem file terdistribusi.

NoSQL

- Database Not-only SQL (NoSQL) adalah database non-relasional yang sangat skalabel, toleran terhadap kesalahan, dan dirancang khusus untuk menampung data semi terstruktur dan tidak terstruktur.
- Basis data NoSQL sering kali menyediakan antarmuka kueri berbasis API yang dapat dipanggil dari dalam aplikasi.
- Basis data NoSQL juga mendukung bahasa kueri selain Bahasa Kueri Terstruktur (SQL) karena SQL dirancang untuk kueri data terstruktur yang disimpan dalam basis data relasional.
- Sebagai contoh, database NoSQL yang dioptimalkan untuk menyimpan file XML akan sering menggunakan XQuery sebagai bahasa kueri.
- Demikian juga, database NoSQL yang dirancang untuk menyimpan data RDF akan menggunakan SPARQL untuk menanyakan hubungan yang dikandungnya.
- Karena itu, ada beberapa database NoSQL yang juga menyediakan antarmuka kueri seperti SQL.

NoSQL

Basis data NoSQL dapat menyediakan antarmuka kueri seperti API atau SQL.

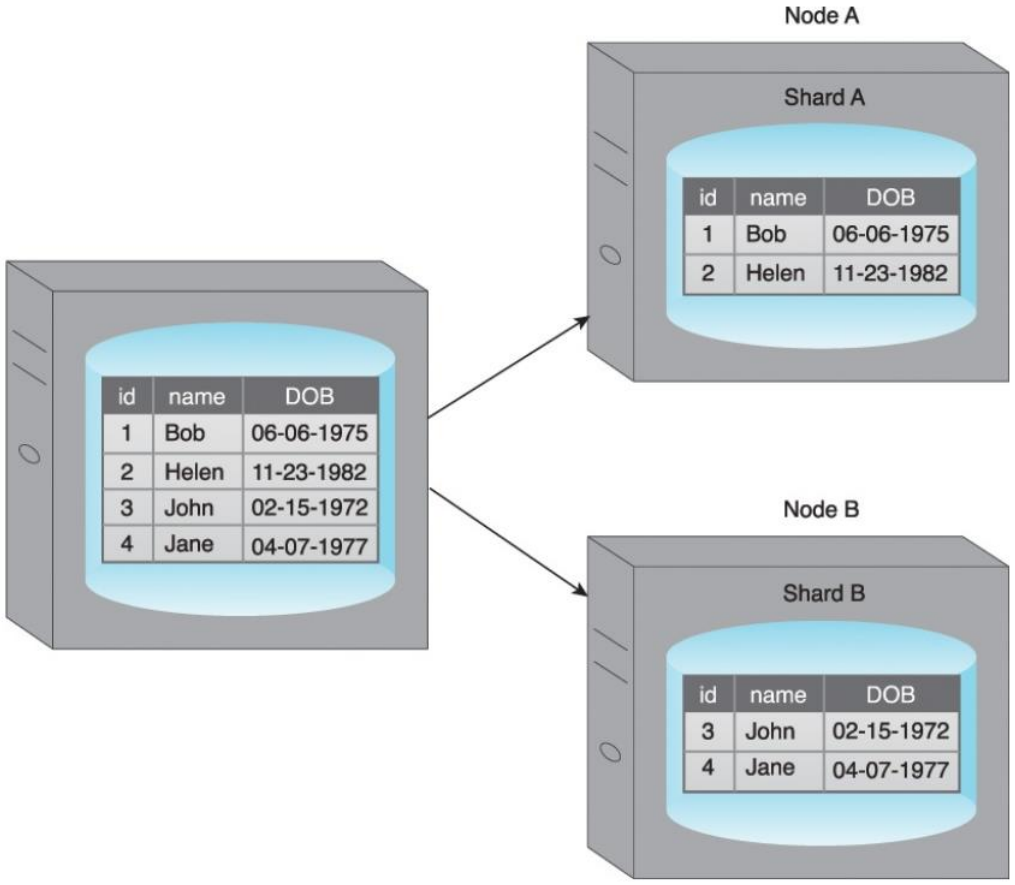


Sharding

- Sharding adalah proses mempartisi secara horizontal kumpulan data besar menjadi kumpulan kumpulan data yang lebih kecil dan lebih mudah dikelola yang disebut pecahan.
- Pecahan didistribusikan di beberapa node, di mana node adalah server atau mesin.
- Setiap pecahan disimpan pada simpul yang terpisah dan setiap simpul hanya bertanggung jawab atas data yang tersimpan di dalamnya.
- Setiap pecahan berbagi skema yang sama, dan semua pecahan secara kolektif mewakili kumpulan data lengkap.

Sharding

Contoh sharding di mana dataset tersebar di Node A dan Node B, menghasilkan Shard A dan Shard B.



Sharding

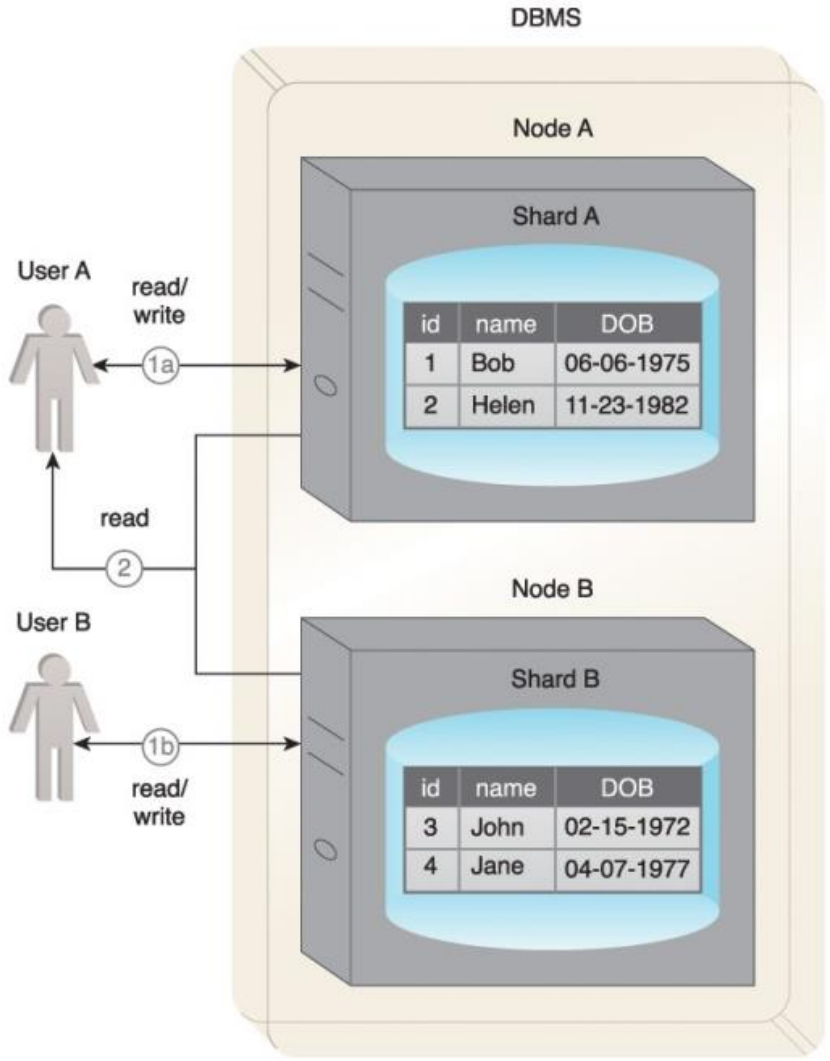
- Sharding sering kali transparan bagi klien, tetapi ini bukan keharusan.
- Penskalaan horizontal adalah metode untuk meningkatkan kapasitas sistem dengan menambahkan sumber daya berkapasitas serupa atau lebih tinggi di samping sumber daya yang ada.
- Karena setiap node hanya bertanggung jawab untuk sebagian dari keseluruhan dataset, waktu baca/tulis sangat meningkat.

Sharding

Contoh sharding di mana data diambil dari Node A dan Node B.

Setiap shard dapat secara mandiri melayani membaca dan menulis untuk subset data tertentu yang menjadi tanggung jawabnya.

Bergantung pada kueri, data mungkin perlu diambil dari kedua pecahan (shards).



Sharding

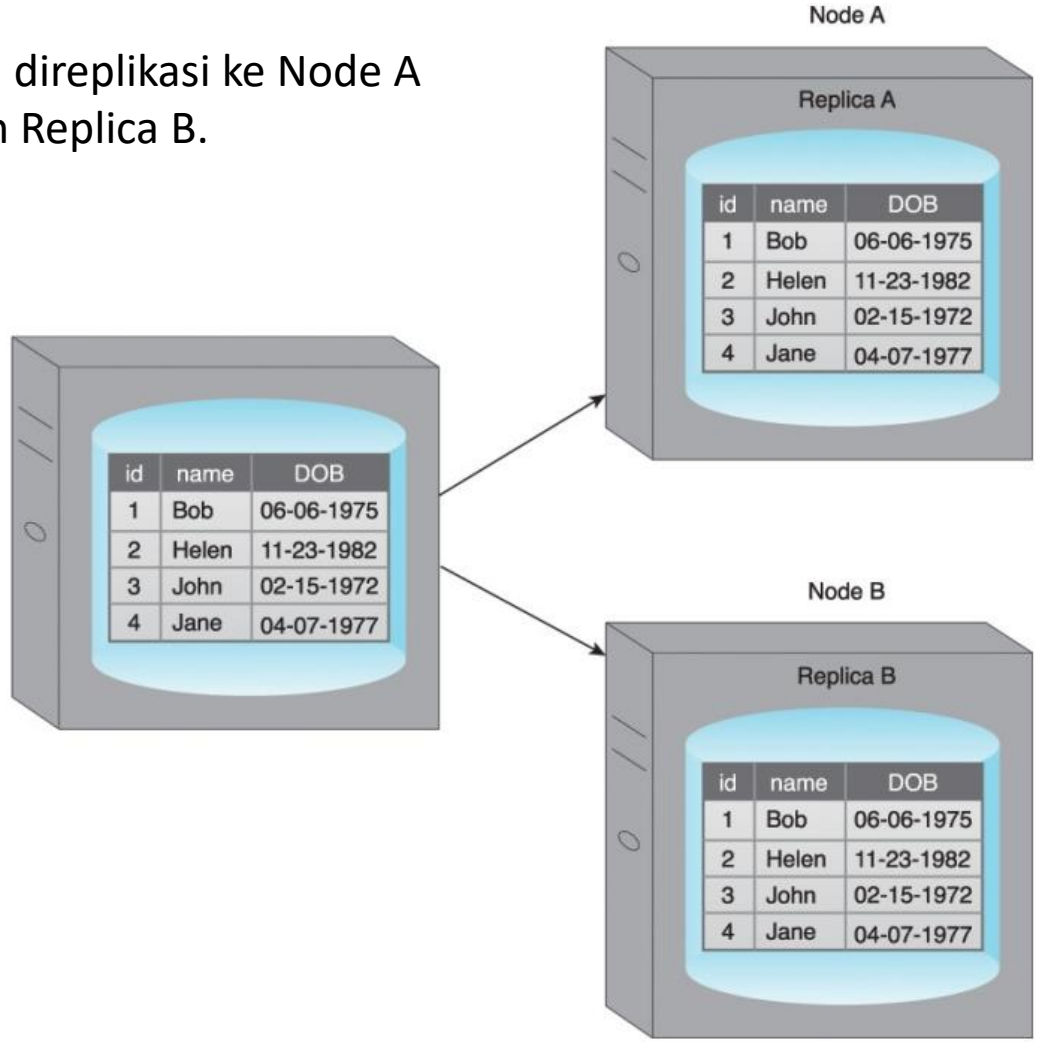
- Manfaat sharding adalah memberikan toleransi parsial terhadap kegagalan.
- Jika terjadi kegagalan node, hanya data yang disimpan di node tersebut yang terpengaruh.
- Berkenaan dengan partisi data, pola kueri perlu diperhitungkan sehingga pecahan itu sendiri tidak menjadi hambatan kinerja.
- Misalnya, kueri yang memerlukan data dari beberapa pecahan akan memberlakukan penalti kinerja.
- Lokalitas data membuat data yang biasa diakses berada di lokasi yang sama pada satu pecahan dan membantu mengatasi masalah kinerja tersebut.

Replikasi (*Replication*)

- Replikasi menyimpan banyak salinan dari kumpulan data, yang dikenal sebagai replica.
- Replikasi menyediakan skalabilitas dan ketersediaan karena fakta bahwa data yang sama direplikasi pada berbagai node.
- Toleransi kesalahan juga tercapai karena redundansi data memastikan bahwa data tidak hilang ketika node individu gagal.
- Ada dua metode berbeda yang digunakan untuk mengimplementasikan replikasi:
  - ✓ master-slave
  - ✓ peer-to-peer

Replikasi (*Replication*)

Contoh replikasi di mana kumpulan data direplikasi ke Node A dan Node B, menghasilkan Replica A dan Replica B.



Master-Slave

- Selama replikasi master-slave, node diatur dalam konfigurasi master-slave, dan semua data ditulis ke node master. Setelah disimpan, data direplikasi ke beberapa node budak.
- Semua permintaan tulis eksternal, termasuk penyisipan, pembaruan, dan penghapusan, terjadi pada node master, sedangkan permintaan baca dapat dipenuhi oleh node slave mana pun.
- Contoh replikasi master-slave di mana Master A adalah titik kontak tunggal untuk semua penulisan, dan data dapat dibaca dari Slave A dan Slave B.
- Replikasi master-slave ideal untuk beban intensif baca daripada beban intensif tulis karena permintaan baca yang meningkat dapat dikelola dengan penskalaan horizontal untuk menambahkan lebih banyak node slave.

Master-Slave

- Penulisan konsisten, karena semua penulisan dikoordinasikan oleh node master.
- Implikasinya adalah kinerja penulisan akan menurun seiring dengan peningkatan jumlah penulisan.
- Jika master node gagal, pembacaan masih dapat dilakukan melalui salah satu node slave.

Master-Slave

- Sebuah node budak dapat dikonfigurasi sebagai node cadangan untuk node master.
- Jika master node gagal, penulisan tidak didukung sampai master node dibuat kembali.
- Node master dibangkitkan dari cadangan node master, atau node master baru dipilih dari node slave.

Master-Slave

- Salah satu masalah dengan replikasi master-slave adalah inkonsistensi pembacaan, yang dapat menjadi masalah jika node slave dibaca sebelum pembaruan master disalin ke sana.
- Untuk memastikan konsistensi pembacaan, sistem pemungutan suara dapat diterapkan di mana pembacaan dinyatakan konsisten jika mayoritas slave berisi versi rekaman yang sama.
- Implementasi sistem pemungutan suara semacam itu membutuhkan mekanisme komunikasi yang andal dan cepat antara para budak.

Master-Slave

Contoh replikasi master-slave di mana terjadi inkonsistensi baca.

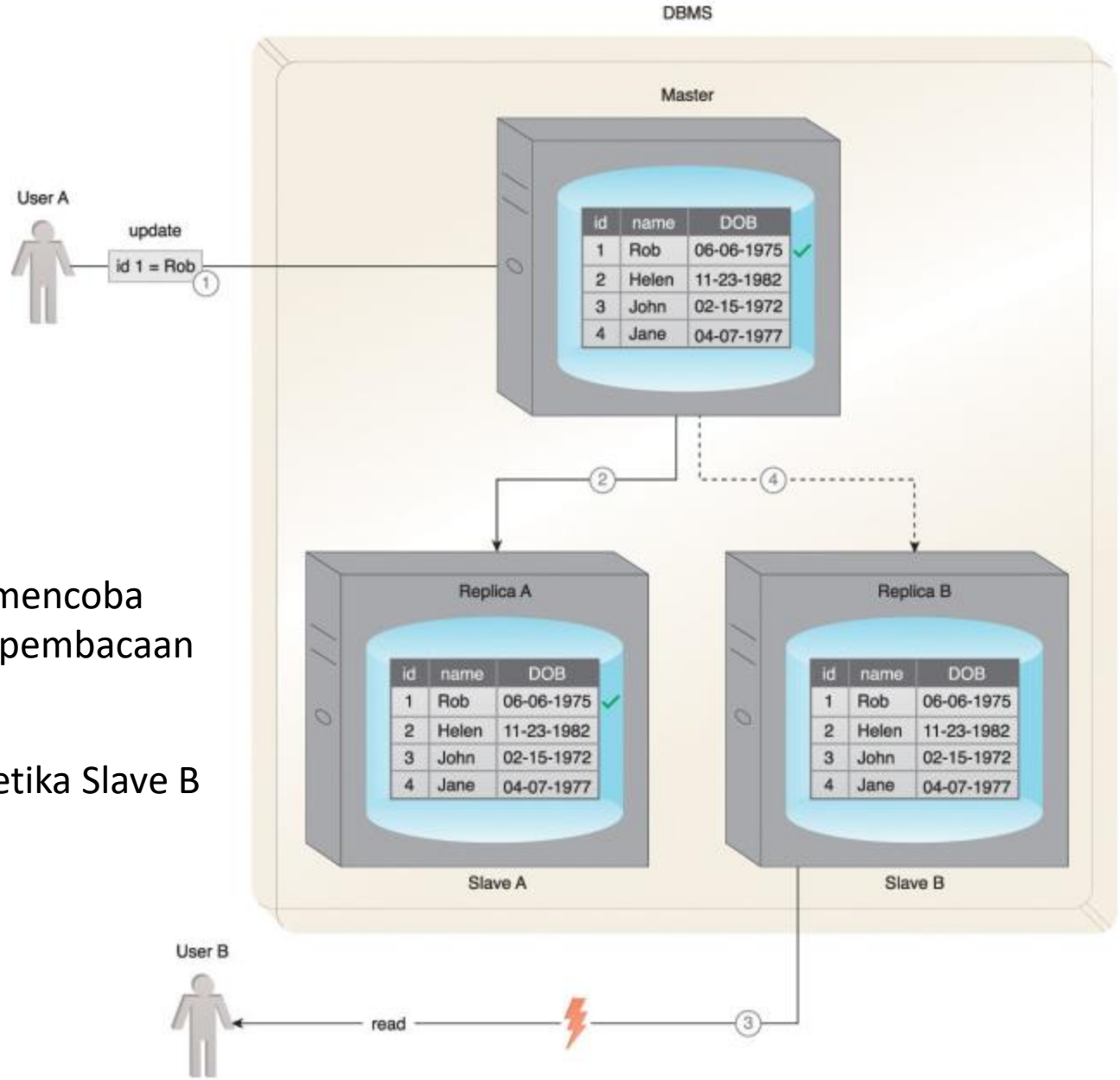
Skenario di mana inkonsistensi baca terjadi:

1. Pengguna A memperbarui data.

2. Data disalin ke Slave A oleh Master.

3. Sebelum data disalin ke Slave B, Pengguna B mencoba membaca data dari Slave B, yang menghasilkan pembacaan yang tidak konsisten.

4. Data pada akhirnya akan menjadi konsisten ketika Slave B diperbarui oleh Master.



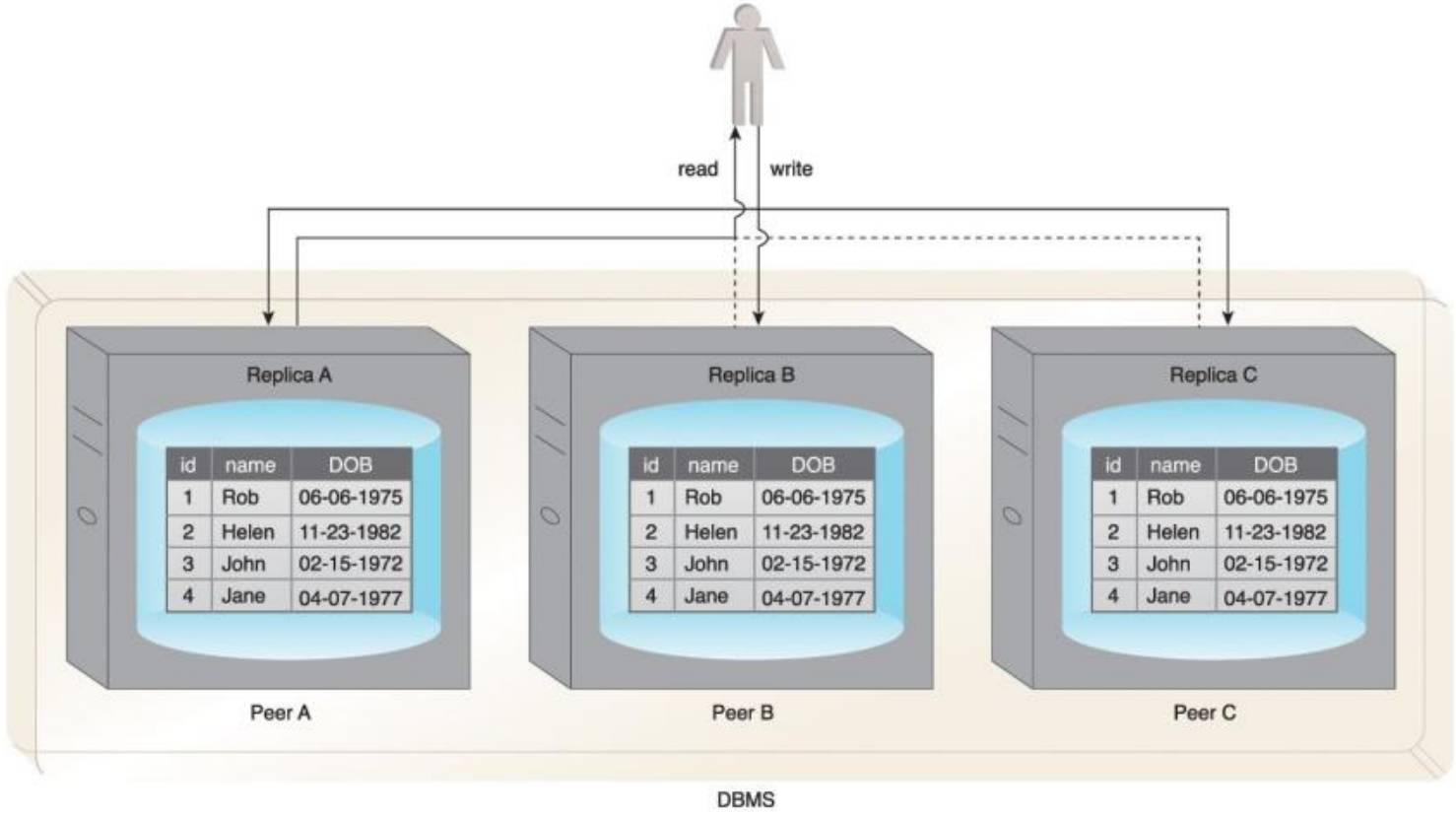
Peer-to-Peer

- Dengan replikasi peer-to-peer, semua node beroperasi pada level yang sama. Dengan kata lain, tidak ada hubungan master-slave antara node.
- Setiap node, yang dikenal sebagai peer, sama-sama mampu menangani membaca dan menulis.

Peer-to-Peer

Menulis disalin ke Rekan A, B dan C secara bersamaan.

Data dibaca dari Peer A, tetapi juga dapat dibaca dari Peer B atau C.

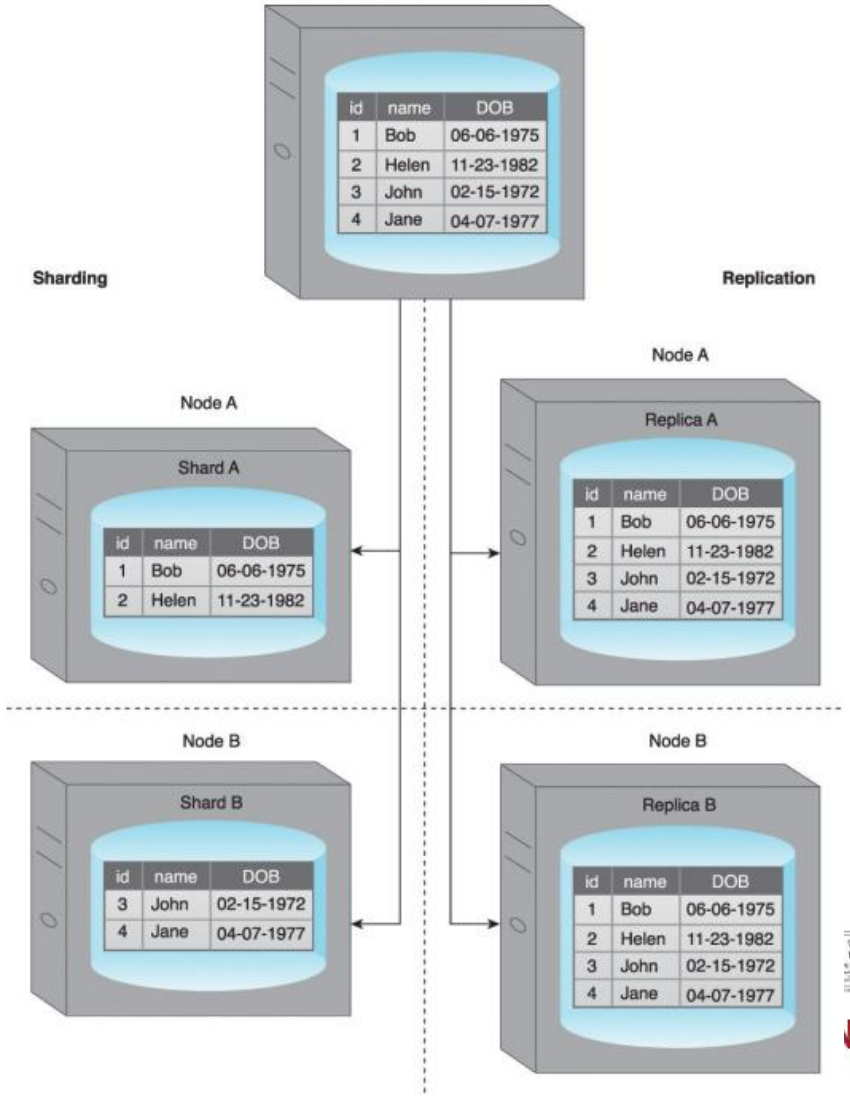


Peer-to-Peer

- Replikasi peer-to-peer cenderung menulis inkonsistensi yang terjadi sebagai akibat dari pembaruan simultan dari data yang sama di beberapa peer.
- Ini dapat diatasi dengan menerapkan strategi konkurensi pesimis atau optimis.
  - Konkurensi pesimis adalah strategi proaktif yang mencegah inkonsistensi.
  - Konkurensi optimis adalah strategi reaktif yang tidak menggunakan penguncian.

Sharding dan Replication

- Sharding dan replikasi dapat digabungkan untuk meningkatkan toleransi kesalahan terbatas yang ditawarkan oleh sharding, mendapatkan manfaat tambahan dari peningkatan ketersediaan dan skalabilitas replikasi.
- Perbandingan sharding dan replikasi yang menunjukkan bagaimana dataset didistribusikan antara dua node dengan aplikasi yang berbeda

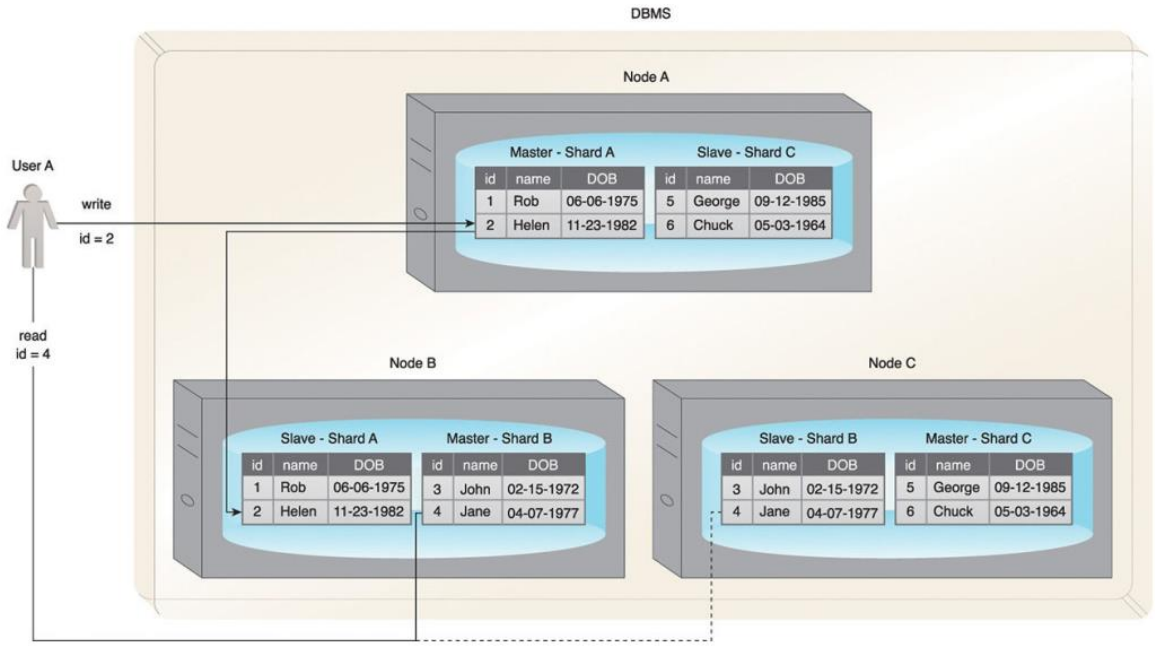


Menggabungkan Sharding dan Replikasi Master-Slave

- Saat sharding digabungkan dengan replikasi master-slave, beberapa shard menjadi budak dari satu master, dan master itu sendiri adalah shard.
- Meskipun menghasilkan banyak master, satu slave-shard hanya dapat dikelola oleh satu master-shard.
- Konsistensi penulisan dipertahankan oleh master shard.
- Jika master-shard menjadi non-operasional atau terjadi pemadaman jaringan, toleransi kesalahan terkait dengan operasi penulisan akan terpengaruh.
- Replika pecahan disimpan di beberapa node budak untuk memberikan skalabilitas dan toleransi kesalahan untuk operasi baca.

Menggabungkan Sharding dan Replikasi Master-Slave

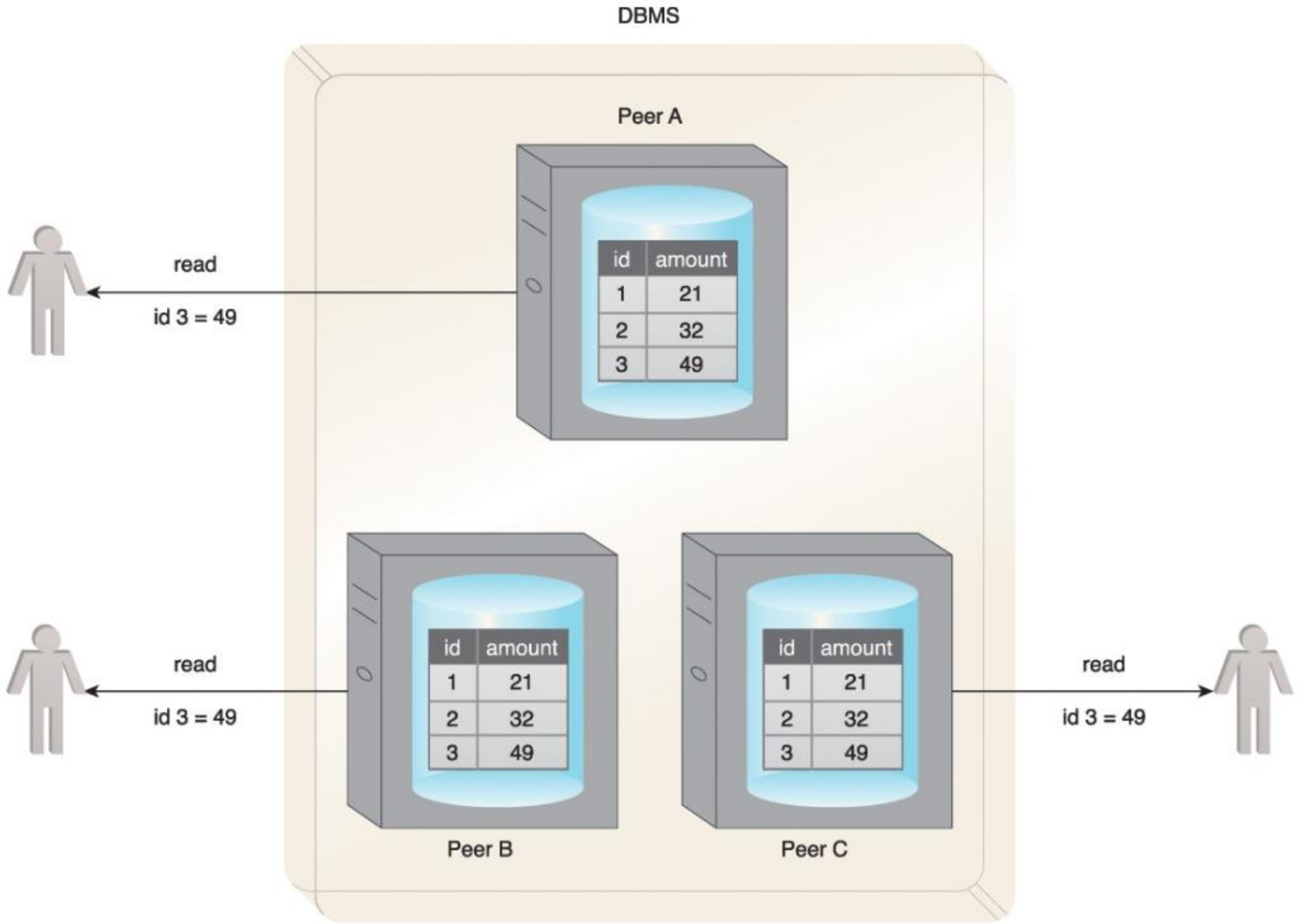
- Setiap node bertindak sebagai master dan slave untuk shard yang berbeda.
- Menulis (id = 2) ke Shard A diatur oleh Node A, karena ini adalah master untuk Shard A.
- Node A mereplikasi data (id = 2) ke Node B, yang merupakan slave untuk Shard A.
- Bacaan (id = 4) dapat dilayani langsung oleh Node B atau Node C karena masing-masing berisi Shard B.
- Gambar di samping menunjukkan contoh yang menunjukkan kombinasi sharding dan replikasi master-slave.



Teorema CAP

- Teorema Consistency, Availability, and Partition tolerance (CAP), juga dikenal sebagai teorema Brewer, mengungkapkan tiga kendala yang terkait dengan sistem basis data terdistribusi.
- Ini menyatakan bahwa sistem database terdistribusi, berjalan di sebuah cluster, hanya dapat menyediakan dua dari tiga properti berikut:
  - Konsistensi – Pembacaan dari node mana pun menghasilkan data yang sama di beberapa node.
  - Ketersediaan – Permintaan baca/tulis akan selalu diakui dalam bentuk keberhasilan atau kegagalan.

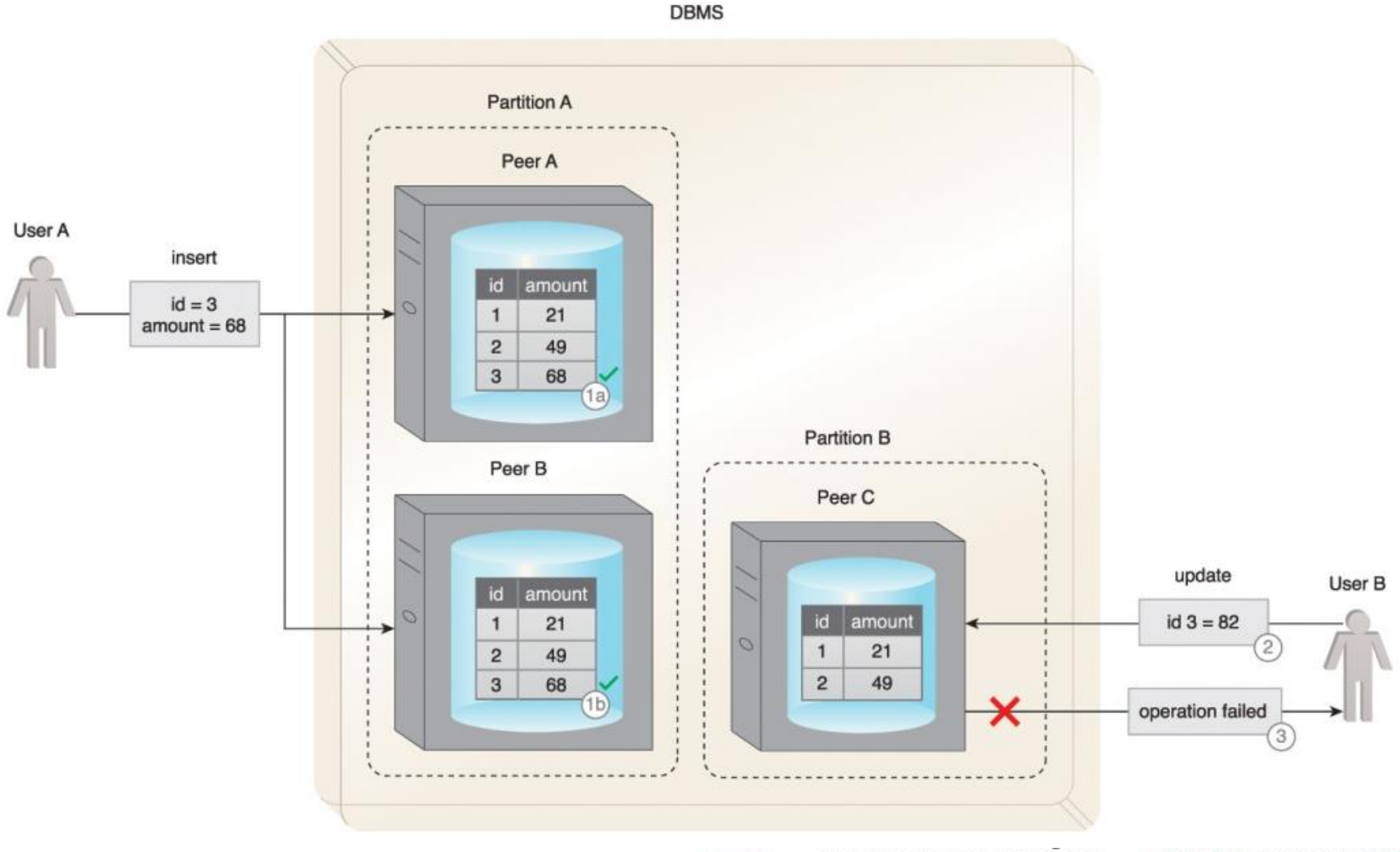
Konsistensi: ketiga pengguna mendapatkan nilai yang sama untuk kolom jumlah meskipun tiga node berbeda melayani catatan.



Ketersediaan dan toleransi partisi: jika terjadi kegagalan komunikasi, permintaan dari kedua pengguna masih dilayani (1, 2).

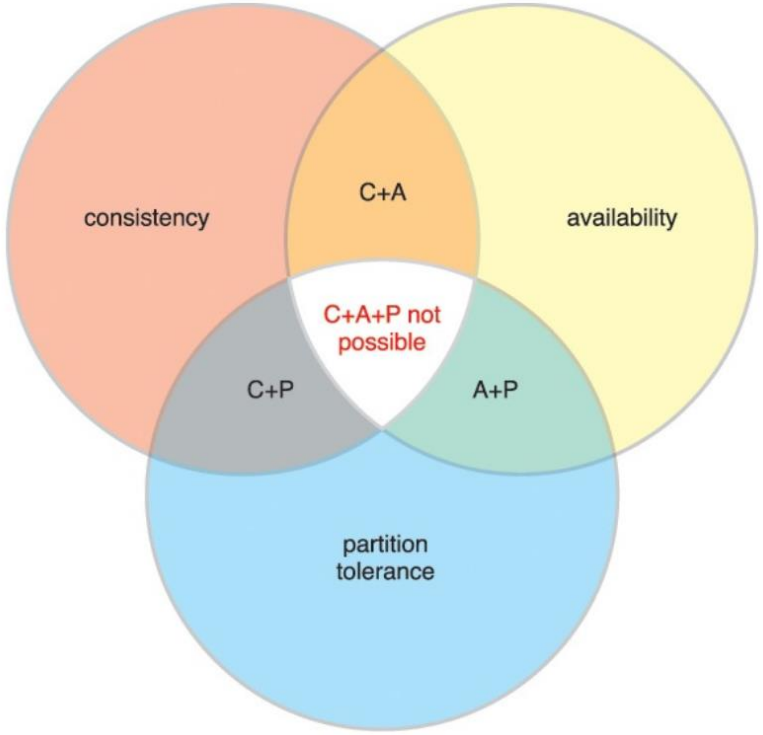
Namun, dengan Pengguna B, pembaruan gagal karena catatan dengan id = 3 belum disalin ke Peer C.

Pengguna diberi tahu (3) bahwa pembaruan telah gagal.



Teorema CAP

- Skenario berikut menunjukkan mengapa hanya dua dari tiga sifat teorema CAP yang dapat didukung secara bersamaan.
- Diagram Venn yang merangkum teorema CAP.



Teorema CAP

- Jika konsistensi (*consistency* - C) dan ketersediaan (*availability* - A) diperlukan, node yang tersedia perlu berkomunikasi untuk memastikan konsistensi (*consistency* - C).
- Oleh karena itu, toleransi partisi (*partition tolerance* - P) tidak dimungkinkan.
- Jika konsistensi (*consistency* - C) dan toleransi partisi (*partition tolerance* - P) diperlukan, node tidak dapat tetap tersedia (*availability* - A) karena node akan menjadi tidak tersedia saat mencapai keadaan konsistensi (*consistency* - C).
- Jika ketersediaan (*availability* - A) dan toleransi partisi (partition tolerance - P) diperlukan, maka konsistensi (*consistency* - C) tidak dimungkinkan karena kebutuhan komunikasi data antar node.
- Jadi, database dapat tetap tersedia (*availability* - A) tetapi dengan hasil yang tidak konsisten.

Teorema CAP

- Dalam database terdistribusi, skalabilitas dan toleransi kesalahan dapat ditingkatkan melalui node tambahan, meskipun ini menantang *consistency* (C).
- Penambahan node juga dapat menyebabkan *availability* (A) terganggu karena *latency* yang disebabkan oleh peningkatan komunikasi antar node.

Teorema CAP

- Sistem database terdistribusi tidak dapat 100% toleran terhadap partisi (P).
- Meskipun gangguan komunikasi jarang terjadi dan bersifat sementara, toleransi partisi (P) harus selalu didukung oleh database terdistribusi; oleh karena itu, CAP umumnya merupakan pilihan antara memilih C+P atau A+P.
- Persyaratan sistem akan menentukan mana yang dipilih.

ACID

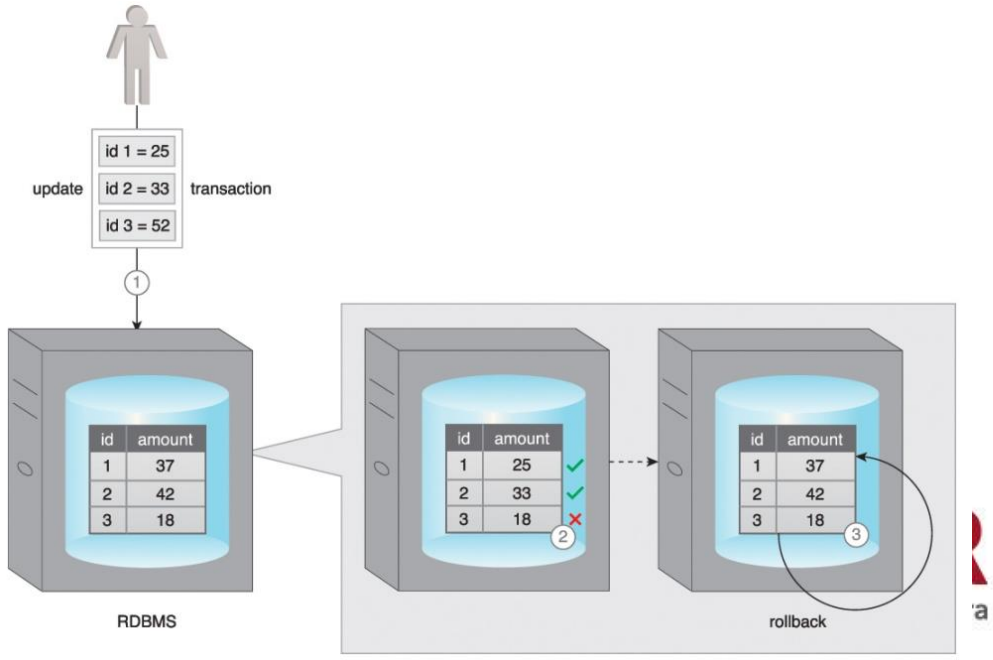
- ACID adalah prinsip desain database yang terkait dengan manajemen transaksi.
- ACID adalah akronim yang berarti:
  - Atomisitas (atomicity)
  - Konsistensi (consistency)
  - Isolasi (isolation)
  - Daya tahan (durability)

ACID

- ACID adalah gaya manajemen transaksi yang memanfaatkan kontrol konkurensi pesimis untuk memastikan konsistensi dipertahankan melalui penerapan kunci catatan.
- ACID adalah pendekatan tradisional untuk manajemen transaksi database karena dimanfaatkan oleh sistem manajemen database relasional.
- Atomicity memastikan bahwa semua operasi akan selalu berhasil atau gagal sepenuhnya.
- Tidak ada transaksi parsial.

ACID

- Seorang pengguna mencoba untuk memperbarui tiga catatan sebagai bagian dari transaksi.
- Dua catatan berhasil diperbarui sebelum terjadinya kesalahan.
- Dua catatan berhasil sebelum terjadinya kesalahan.

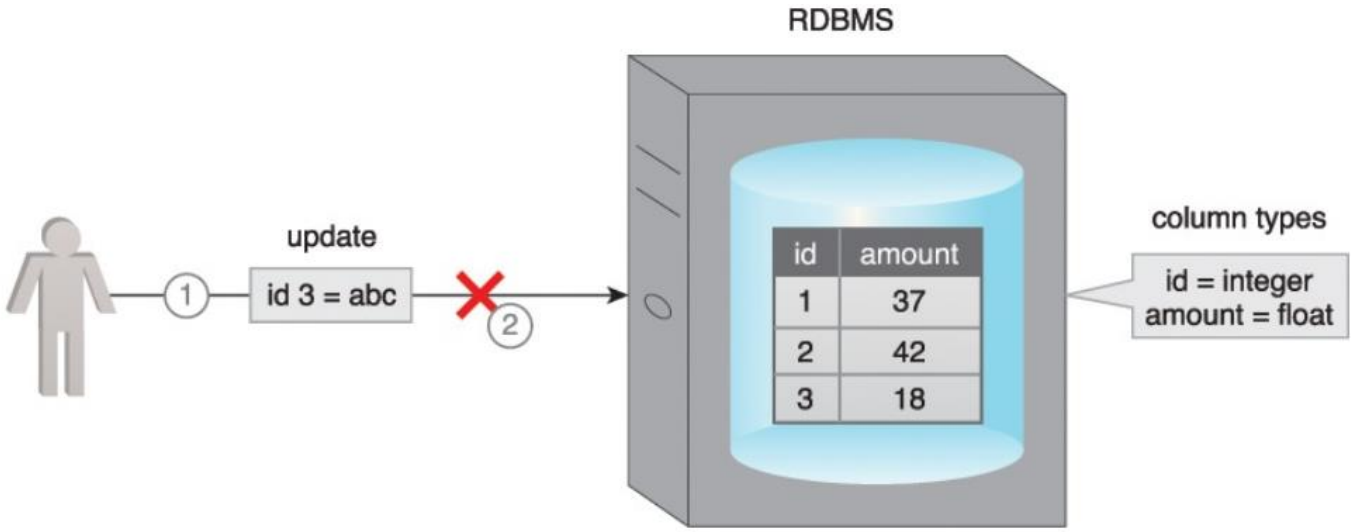


ACID

- Konsistensi memastikan bahwa database akan selalu tetap dalam keadaan konsisten dengan memastikan bahwa hanya data yang sesuai dengan batasan skema database yang dapat ditulis ke database.
- Dengan demikian database yang dalam keadaan konsisten akan tetap dalam keadaan konsisten setelah transaksi berhasil.

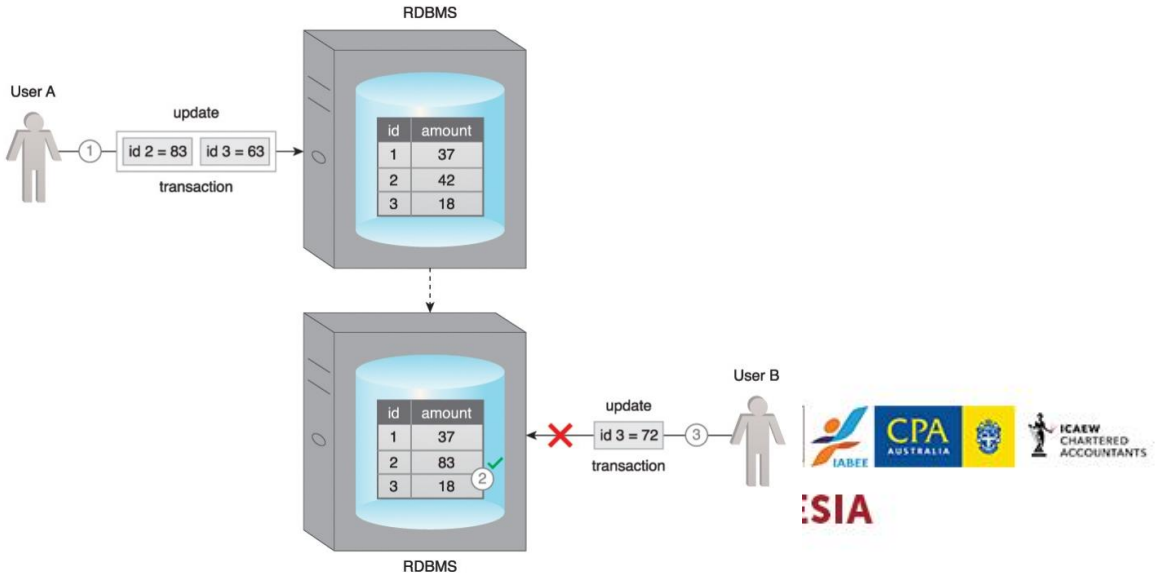
ACID

- Pengguna mencoba memperbarui kolom jumlah tabel yang bertipe float dengan nilai varchar.
- Basis data menerapkan pemeriksaan validasinya dan menolak pembaruan ini karena nilainya melanggar pemeriksaan batasan untuk kolom jumlah.



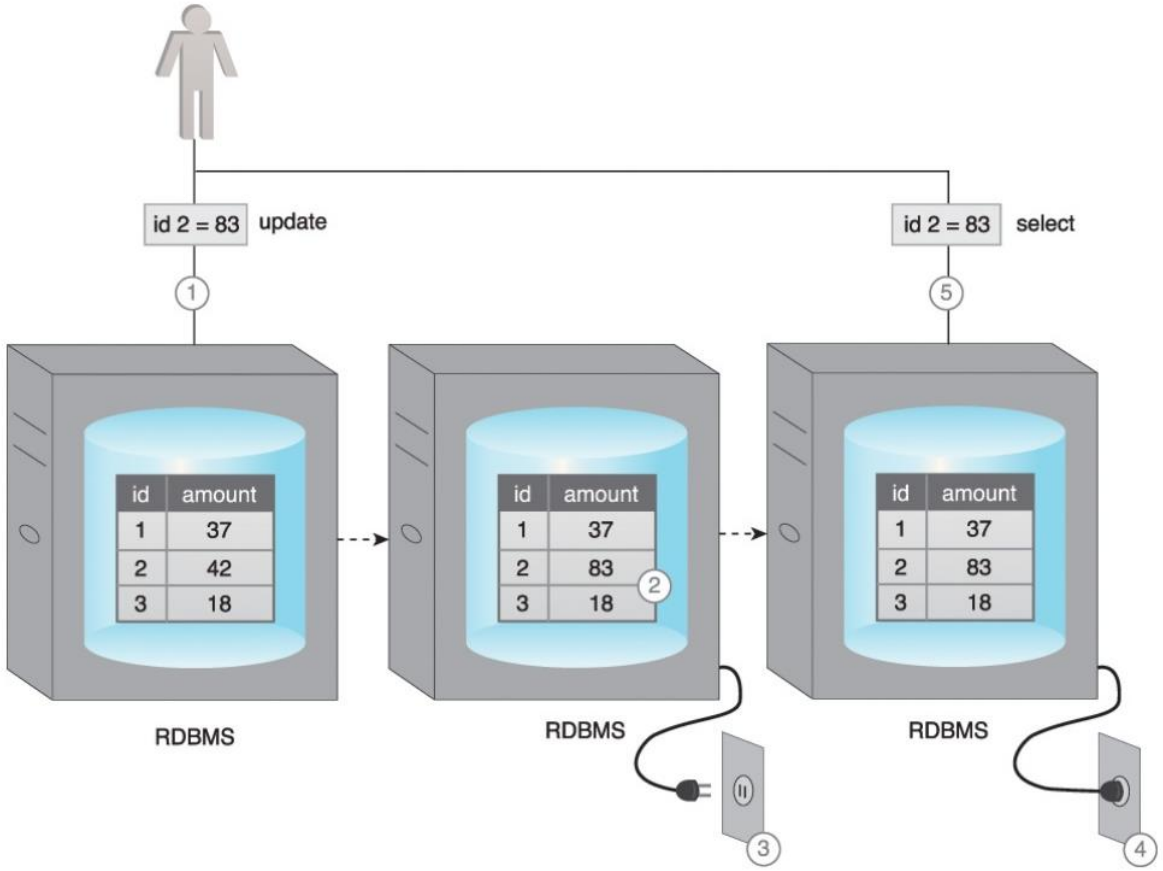
ACID

- Pengguna A mencoba memperbarui dua catatan sebagai bagian dari transaksi.
- Basis data berhasil memperbarui catatan pertama.
- Namun, sebelum dapat memperbarui catatan kedua, Pengguna B mencoba memperbarui catatan yang sama.
- Basis data tidak mengizinkan pembaruan Pengguna B hingga pembaruan Pengguna A berhasil atau gagal sepenuhnya.
- Ini terjadikarena record dengan id3 di lock oleh database sampai transaksi selesai.



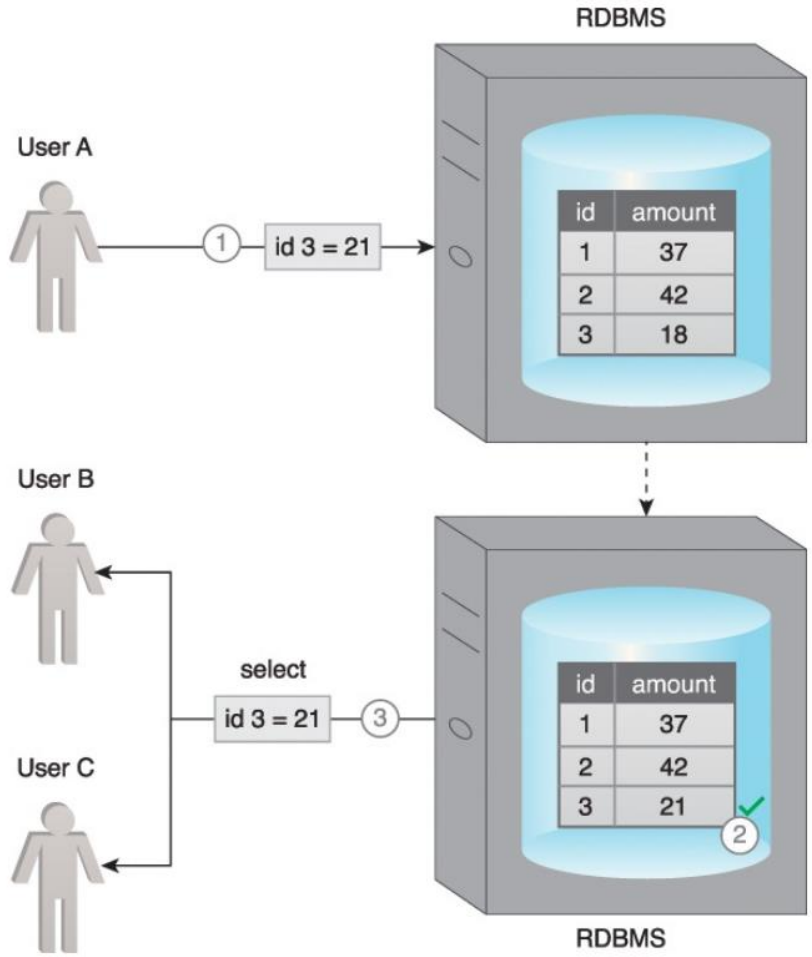
ACID

- Seorang pengguna memperbarui catatan sebagai bagian dari transaksi.
- Basis data berhasil memperbarui catatan.
- Tepat setelah pembaruan ini, terjadi kegagalan daya.
- Basis data mempertahankan statusnya saat tidak ada daya.
- Listrik dihidupkan kembali.
- Basis data menyajikan catatan sesuai pembaruan terakhir saat diminta oleh pengguna.



ACID

- Pengguna A mencoba memperbarui catatan sebagai bagian dari transaksi.
- Basis data memvalidasi nilai dan pembaruan berhasil diterapkan.
- Setelah transaksi berhasil diselesaikan, ketika Pengguna B dan C meminta catatan yang sama, database memberikan nilai yang diperbarui kepada kedua pengguna.



BASE

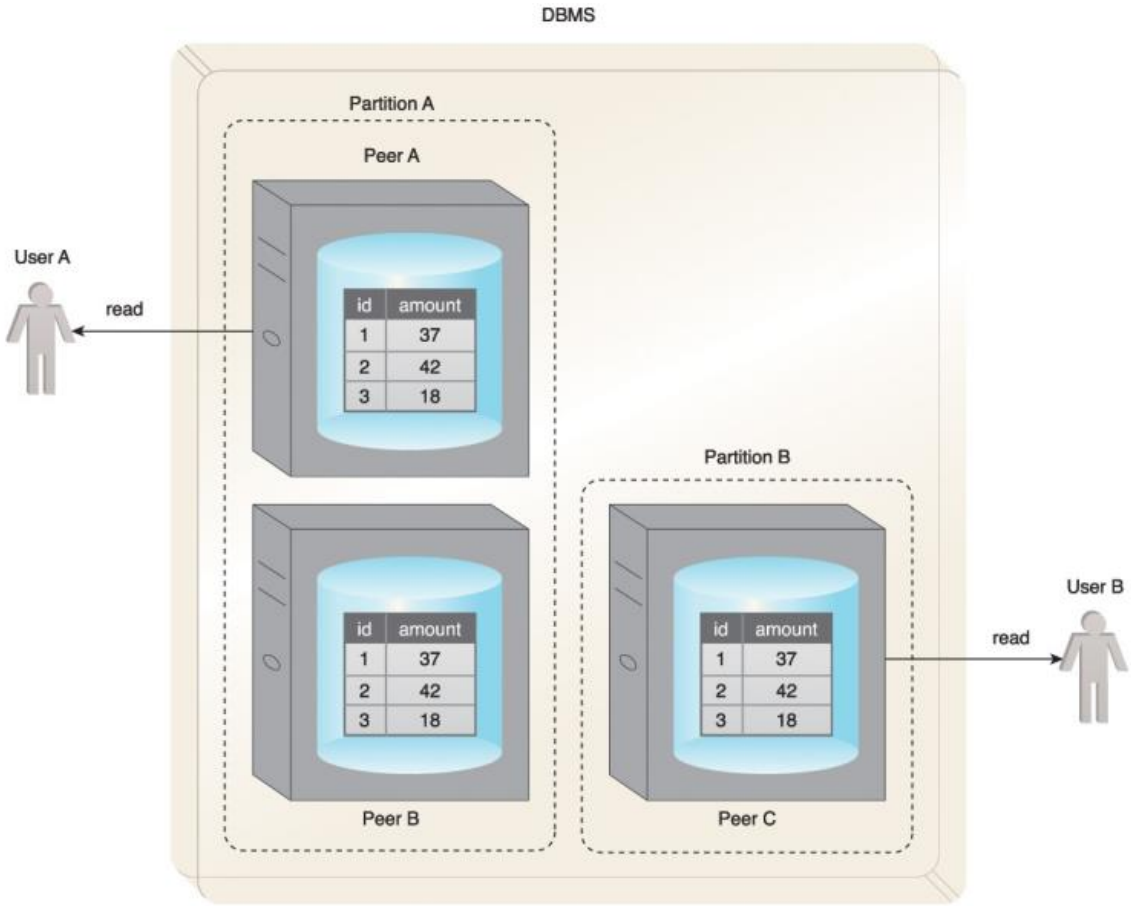
- BASE adalah prinsip desain database berdasarkan teorema CAP dan dimanfaatkan oleh sistem database yang menggunakan teknologi terdistribusi.
- BASE adalah singkatan dari :
  - *basically available* (pada dasarnya tersedia)
  - *soft state* (keadaan lunak)
  - *eventual consistency* (konsistensi akhirnya)

BASE

- Ketika database mendukung BASE, database lebih menyukai ketersediaan daripada konsistensi.
- Dengan kata lain, database adalah A+P dari perspektif CAP.
- BASE memanfaatkan konkurensi optimis dengan melonggarkan batasan konsistensi kuat yang diamanatkan oleh properti ACID.
- Jika basis data “basically available”, basis data itu akan selalu mengakui permintaan klien, baik dalam bentuk data yang diminta atau pemberitahuan berhasil/gagal.
- Database pada dasarnya tersedia, meskipun telah dipartisi sebagai akibat dari kegagalan jaringan.

BASE

- Basis data pada dasarnya tersedia, meskipun telah dipartisi sebagai akibat dari kegagalan jaringan.
- Pengguna A dan Pengguna B menerima data meskipun database dipartisi oleh kegagalan jaringan.
- Gambar di samping menunjukkan:
  - Pengguna A memperbarui catatan pada Rekan A.
  - Sebelum rekan-rekan lain diperbarui, Pengguna B meminta catatan yang sama dari Rekan C.
  - Basis data sekarang dalam keadaan lunak, dan data basi dikembalikan ke Pengguna B.



BASE

- *Soft state* (keadaan lunak) berarti bahwa database mungkin berada dalam keadaan tidak konsisten saat data dibaca; dengan demikian, hasilnya dapat berubah jika data yang sama diminta lagi.
- Ini karena data dapat diperbarui untuk konsistensi, meskipun tidak ada pengguna yang menulis ke database di antara dua pembacaan.
- Properti ini terkait erat dengan konsistensi akhirnya.