Pertemuan 7 — *Storage Engine*, NoSQL, dan *Serving*

Memilih *database* menurut pola akses: dari B-tree dan LSM, *serving store* untuk *layer* gold, sampai *immutable ledger*

Hendri Karisma

Semester Ganjil 2026/2027

Program Studi Teknik Informatika, STMIK Tazkia

Alur pertemuan hari ini

1. Mulai dari pola akses, bukan dari merek
2. B-tree: struktur klasik *database* relasional
3. LSM-tree: *memtable*, SSTable, *compaction*
4. Amplifikasi tulis, baca, dan ruang
5. NoSQL: *key-value*, *wide-column*, *document*
6. *Graph*, *time-series*, *search*, PostgreSQL berekstensi
7. OLAP *columnar*: ClickHouse, Druid, Pinot
8. *Partition*, dan CAP/PACELC sebagai lensa memilih
9. *Polyglot persistence* dan *serving store* untuk gold
10. *Immutable storage* dan *ledger*
11. Praktikum: empat pola akses dan immudb

Sasaran pertemuan ini

Anda dapat menjelaskan mengapa satu *database* cepat untuk satu pola akses dan lambat untuk pola lain, lalu memilih store yang tepat untuk tiap kebutuhan bank syariah digital, dengan alasan berupa angka pengukuran.

Mulai dari pola akses

Empat pertanyaan, empat pola akses

Pertanyaan di bank	Pola akses	Yang menentukan kecepatan
Berapa saldo rekening 1234?	*Point lookup* per *key*	Indeks, data di memori
Riwayat 50 transaksi terakhir nasabah X	*Range scan* per *key*	Data satu nasabah tersimpan berdekatan
Total transaksi seluruh cabang bulan ini	*Aggregation* atas miliaran baris	Hanya membaca kolom yang perlu, *compression*
Transaksi berketerangan “zakat” atau “wakaf”	Pencarian teks	*Inverted index*

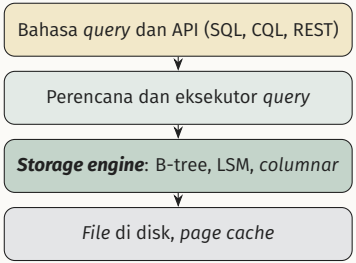
Pertanyaan yang benar

Berapa rasio baca:tulis? Membaca per *key* atau seluruh tabel? Seberapa segar datanya? Berapa besar? Apa yang terjadi bila satu node mati?

Pertanyaan yang salah

“*Database* apa yang paling cepat?” Tidak ada jawabannya. Yang ada: paling cepat **untuk pola akses apa**.

Di bawah setiap *database* ada *storage engine*



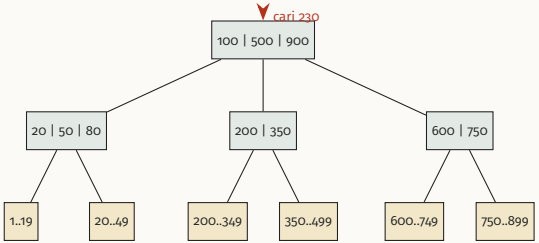
- *Storage engine* menentukan cara data disusun di disk dan memori: inilah yang membuat tulis cepat atau baca cepat.
- Satu produk bisa punya beberapa *engine*: MySQL (InnoDB, MyRocks), MongoDB (WiredTiger), ClickHouse (keluarga MergeTree).
- RocksDB adalah *library* LSM yang disematkan di banyak sistem, termasuk *state backend* Flink dan Kafka Streams (pertemuan 11).

Ringkasan

Pahami dua keluarga besar, B-tree dan LSM, dan Anda bisa menebak perilaku sebagian besar *database*.

B-tree

B-tree: pohon lebar yang dangkal



daun berisi baris (atau penunjuk ke baris), disusun terurut

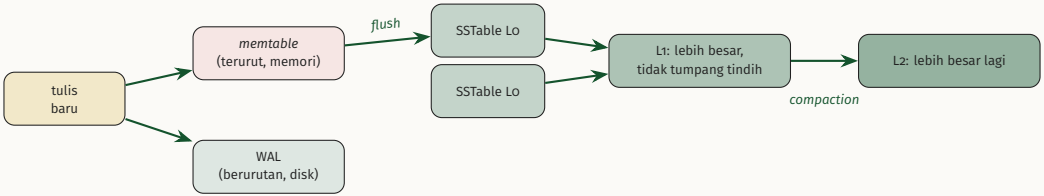
- Data disusun dalam ***page*** berukuran tetap (umumnya beberapa KB).
- Tiap *page* dalam memuat ratusan *key*, sehingga pohon tetap dangkal: jutaan baris cukup dicapai dalam beberapa lompatan.
- Pembaruan dilakukan **di tempat** (*update in place*); WAL (*write-ahead log*) menjaga data saat listrik padam.
- Dipakai PostgreSQL, MySQL InnoDB, Oracle, SQL Server.

Ringkasan

B-tree unggul untuk baca per *key* dan *range scan* yang dapat diprediksi, dengan konsekuensi tulis acak: satu baris baru bisa mengubah beberapa *page*.

LSM-tree

LSM-tree: tulis dulu di memori, rapikan belakangan



- Tulis masuk ke ***memtable*** di memori dan ke WAL secara berurutan: cepat.
- Saat penuh, *memtable* di-*flush* menjadi ***SSTable***: *file* terurut yang tidak pernah diubah (*immutable*).
- ***Compaction*** menggabungkan SSTable, membuang versi lama dan *tombstone* (penanda hapus).
- Dipakai RocksDB, LevelDB, Cassandra, ScyllaDB, HBase, dan banyak *time-series database*.
- Strategi *compaction* (*leveled* vs *size-tiered*) menggeser keseimbangan antara *cost* baca, tulis, dan ruang.

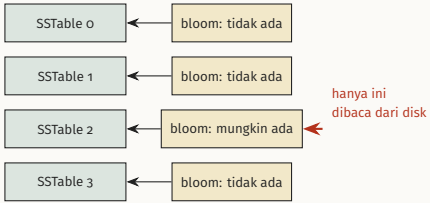
Sumber: O'Neil dkk. (1996), *The Log-Structured Merge-Tree*; Kleppmann (2017) Bab 3.

Membaca di LSM dan peran *bloom filter*

Untuk membaca *key* `rek:1234`, LSM mungkin harus memeriksa:

1. *memtable*,
2. setiap SSTable di L0,
3. satu SSTable per level berikutnya.

Tanpa bantuan, satu baca bisa menjadi banyak pembacaan disk.



Bloom filter per SSTable

Struktur ringkas di memori yang menjawab "*key* ini **pasti tidak ada**" atau "**mungkin ada**". Bisa salah positif, tidak pernah salah negatif. SSTable yang pasti tidak memuat *key* dilewati.

Ringkasan

Bloom filter menukar sedikit memori dengan pengurangan baca disk yang besar. Gagasan yang sama dipakai Parquet dan ClickHouse untuk melewati blok data.

Amplifikasi: *cost* yang tersembunyi

Tiga amplifikasi

Amplifikasi	Artinya	Kecenderungan
Tulis (*write*)	Byte yang benar-benar ditulis ke disk per byte data yang dikirim aplikasi	LSM: data ditulis ulang tiap *compaction*. B-tree: satu baris kecil bisa menulis satu *page* penuh.
Baca (*read*)	Jumlah pembacaan per satu permintaan	B-tree umumnya rendah dan stabil; LSM bergantung pada jumlah level dan *bloom filter*
Ruang (*space*)	Byte di disk per byte data hidup	LSM menyimpan versi lama sampai *compaction*; B-tree menyisakan ruang kosong di *page*

Konsekuensi praktis

SSD memiliki batas jumlah tulis seumur hidupnya; amplifikasi tulis yang tinggi mempercepat ausnya dan memakan *bandwidth* disk yang seharusnya untuk *query*.

Ringkasan

Tidak ada struktur yang menang di ketiganya sekaligus. Memilih *engine* berarti memilih amplifikasi mana yang paling bisa Anda tanggung.

Sumber: Kleppmann (2017) Bab 3; dokumentasi RocksDB tentang *compaction*.

B-tree atau LSM untuk kasus bank?

Cenderung B-tree

- Rekening, saldo, akad pembiayaan: transaksi ACID, baca per *key*, banyak *query* ad hoc
- Beban seimbang antara baca dan tulis
- Contoh: PostgreSQL untuk *database* inti

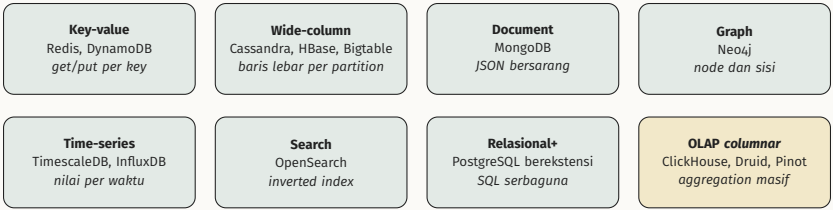
Cenderung LSM

- Log aktivitas aplikasi, *event* klik, data sensor ATM: tulis sangat deras, jarang diubah
- Data *append-only* berskala besar yang dibaca per *key* dan waktu
- Contoh: Cassandra/ScyllaDB untuk riwayat *event* per nasabah

Aturan praktis: **mulai dengan PostgreSQL**. Pindah ke store khusus ketika pengukuran menunjukkan PostgreSQL menjadi hambatan, bukan karena takut suatu hari menjadi hambatan.

Ragam NoSQL

Peta model data



“NoSQL” mencakup beragam model data yang masing-masing mengorbankan sesuatu dari model relasional (*join*, *schema* ketat, transaksi lintas baris) demi skala atau pola akses tertentu.

Key-value dan *wide-column*

Key-value: Redis

Struktur data di memori (string, hash, *sorted set*, *stream*). *Latency* sangat rendah per *key*.

Cocok: *cache* saldo, sesi, *rate limit* percobaan PIN, penghitung s*liding window* untuk anti-*fraud*.
Hati-hati: data harus muat di memori; persistensi (RDB, AOF) dan *replication* perlu diatur dengan sadar, jangan jadikan satu-satunya salinan data penting.

Wide-column: Cassandra

Data dikelompokkan per ***partition key***, diurutkan per ***clustering key***. Rancang tabel **per *query***; satu entitas boleh muncul di beberapa tabel.

Cocok: riwayat transaksi per nasabah, *event* per perangkat, tulis deras lintas pusat data.
Hati-hati: tidak ada *join*; *query* di luar *partition key* mahal; *partition* yang terlalu besar (*hot partition*) merusak kinerja.

`PRIMARY KEY ((nasabah_id), waktu_trx)` ⇒ “50 transaksi terakhir nasabah X” dibaca dari satu *partition* yang sudah terurut.

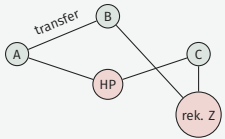
Document dan *graph*

Document: MongoDB

Satu dokumen JSON memuat data yang biasa dibaca bersama, misalnya profil nasabah beserta daftar alamat dan dokumen KYC.

Cocok: *schema* yang sering berubah, data bersarang, katalog produk.
Hati-hati: relasi banyak-ke-banyak dan laporan lintas dokumen; *schema* longgar tetap butuh validasi.

Graph: jaringan *fraud*



Tiga nasabah berbeda memakai satu perangkat dan bermuara ke satu rekening, pola ini mudah ditanyakan sebagai *traversal* graf, sulit sebagai *join* SQL berlapis.
Cocok: *fraud ring*, anti pencucian uang, rekomendasi.

Time-series, *search*, dan PostgreSQL berekstensi

Time-series

Data bernilai per waktu: *metric* ATM, beban server, kurs. *Compression* per kolom waktu, *retention* dan *downsampling* otomatis.

TimescaleDB (ekstensi PostgreSQL), InfluxDB.

Search: OpenSearch

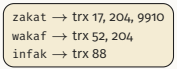
Inverted index: tiap kata menunjuk ke dokumen yang memuatnya. Pencarian teks bebas, analitik log, *vector search* untuk kemiripan *embedding*.

Bukan *database* utama: perlakukan sebagai indeks turunan.

PostgreSQL+

Satu mesin, banyak peran: OLTP, JSONB, TimescaleDB untuk *time-series*, pgvector untuk *embedding*, PostGIS untuk lokasi cabang.

Pilihan pertama yang masuk akal sebelum menambah komponen baru.



Query "zakat wakaf" cukup mengiriskan dua daftar (trx 204) tanpa memindai seluruh transaksi. Itulah alasan OpenSearch cepat untuk teks dan lambat untuk *aggregation* numerik berat.

OLAP *columnar*

ClickHouse, Druid, Pinot: *aggregation* atas miliaran baris

- *Storage* ***columnar*** dan *compression* per kolom (pertemuan 2), eksekusi *vectorized* (pertemuan 5).
- *Sparse primary index*: data disusun terurut, indeks hanya menunjuk tiap beberapa ribu baris; blok yang tidak relevan dilewati.
- Ingest dari Kafka hampir *real-time*: data siap di-*query* beberapa detik setelah tiba.
- *Materialized view* untuk *aggregation* yang terus diperbarui.

Bedanya dengan *lakehouse*

Lakehouse (pertemuan 4): murah, terbuka, cocok untuk data besar yang di-*query* sesekali. OLAP *real-time*: data di disk lokal yang cepat, *query* sub-detik berulang dari *dashboard*, *cost* per GB lebih tinggi.

Bukan pengganti OLTP

Update dan *delete* baris tunggal mahal dan bersifat asinkron di keluarga MergeTree. Jangan simpan saldo yang berubah-ubah di sini.

Tabel transaksi di ClickHouse

```
CREATE TABLE trx (
  waktu       DateTime,
  cabang_id   UInt16,
  rekening_id UInt64,
  jenis       LowCardinality(String),
  nominal     Decimal(18,2)
) ENGINE = MergeTree
PARTITION BY toYYYYMM(waktu)
ORDER BY (cabang_id, waktu);

-- dasbor: nominal per cabang per jam hari ini
SELECT cabang_id, toStartOfHour(waktu) AS jam,
       sum(nominal) AS total
FROM trx
WHERE waktu >= today()
GROUP BY cabang_id, jam;
```

- `PARTITION BY` bulan: *partition* lama mudah dipindah atau dihapus, kaitan dengan *tiering* dan retensi (pertemuan 3).
- `ORDER BY` menentukan urutan fisik dan *sparse index*. Pilih kolom yang paling sering difilter.
- `LowCardinality`: kamus nilai untuk kolom dengan sedikit nilai unik.
- *Query* hanya membaca tiga kolom, bukan lima.

Partition, dan CAP/PACELC sebagai lensa

Partition (*sharding*): membagi data ke banyak node

Range partitioning

Rentang *key* per node (A–F, G–M, ...) atau per waktu.
+ *range scan* efisien.
− rawan *hot spot*: semua transaksi hari ini jatuh di node yang sama.

Hash partitioning

`hash(kunci) mod N`, atau *consistent hashing*.
+ beban merata.
− *range scan* lintas *key* harus bertanya ke semua node.

Memilih *partition key*

Partition key adalah keputusan yang **paling mahal diubah**. Pertanyaan *key*-nya: *query* apa yang paling sering, dan apakah ada nilai *key* yang jauh lebih ramai (rekening penampung QRIS satu *merchant* besar)?

Sambungan dengan Sistem Terdistribusi

Consistent hashing, *leader-follower replication* dan *leaderless*, *quorum*, Raft, serta *crash* vs *Byzantine fault tolerance* sudah **dibedah mekanismenya** di Sister P7 Sabtu lalu (14 November). Di sini kita memakainya sebagai kriteria memilih.

CAP dan PACELC sebagai lensa memilih *database*

CAP: saat jaringan terputus (P)

Pilih tetap **konsisten** (tolak sebagian permintaan) atau tetap **tersedia** (layani, walau mungkin data basi).

PACELC: saat normal pun ada pilihan

Else, ketika tidak ada *partition*, pilih ***latency*** rendah atau ***consistency*** kuat. Pilihan inilah yang dirasakan pengguna setiap hari.

Sumber: Abadi (2012), IEEE Computer.

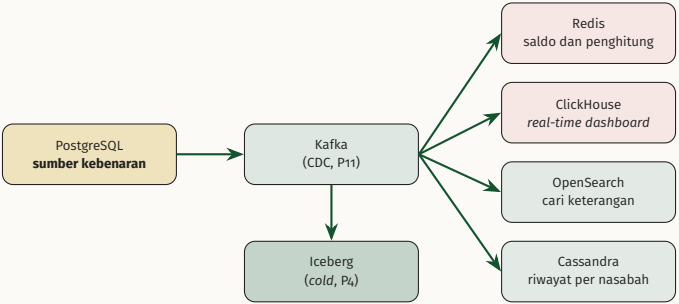
Data bank	Kecenderungan
Saldo, mutasi rekening	Konsisten (PC/EC)
Feed notifikasi	Tersedia, *latency* (PA-/EL)
Penghitung *fraud* per menit	*Latency*, toleran sedikit selisih
Dashboard manajemen	Boleh basi beberapa menit

Ringkasan

Banyak sistem dapat diatur per *query* (misalnya tingkat *consistency* di Cassandra). Tanyakan: **apa ruginya bila data ini basi beberapa detik?**

Polyglot persistence

Satu data, banyak store



Untungnya

Tiap pola akses dilayani store yang cocok; beban analitik tidak mengganggu *database* transaksi.

Cost-nya

Lebih banyak sistem untuk dioperasikan, diamankan, dan diaudit. Data turunan bisa tertinggal (*lag*). Data pribadi kini ada di lima tempat, semua harus tunduk pada UU PDP (P14–P15).

Store mana untuk *hot path*?

Store	Pakai bila	Pertimbangkan alternatif bila
Redis	*Lookup* per *key* dalam milidetik, penghitung, *cache*	Data tidak muat di memori; butuh *aggregation query*
ClickHouse	*Aggregation real-time* atas data besar untuk *dashboard* dan *alert*	Butuh *update* per baris yang sering; transaksi ACID
OpenSearch	Pencarian teks, analitik log, *vector search*	*Aggregation* numerik berat; sumber kebenaran
Cassandra / ScyllaDB	Tulis sangat deras, baca per *partition key*, multi pusat data	*Query* ad hoc, *join*, tim kecil
PostgreSQL	Transaksi, data relasional, beban sedang; dengan ekstensi untuk *time-series* dan *vector*	Tulis atau analitik melampaui satu node besar

Best practice industri

Satu sumber kebenaran, store lain sebagai turunan yang dapat dibangun ulang dari log (Kafka) atau *lakehouse*. Tambah store hanya dengan alasan terukur, dan catat siapa pemilik tiap salinan data.

Serving store untuk *layer* gold

Siapa yang membaca *layer* gold, dan dari mana

Gold disimpan di *lakehouse* (Parquet/Iceberg), tetapi *dashboard* butuh jawaban cepat untuk banyak pengguna sekaligus. Biasanya gold disalin ke satu *serving store*.

Pilihan	Pakai bila	Catatan
DuckDB (*file*)	Satu analis, data puluhan GB, demo lokal	Paling sederhana; tidak untuk banyak penulis atau banyak pengguna *dashboard* serentak
PostgreSQL	Gold relatif kecil (juta baris), butuh *row-level security*, *dashboard* internal	Pilihan aman untuk UTS; indeks pada kolom filter
ClickHouse	Gold besar, *aggregation* interaktif, data terus bertambah	Membaca Parquet langsung dari MinIO; cocok untuk UAS dan *hot path*
Trino atas Iceberg	Banyak tabel, *query ad hoc*, tanpa menyalin	*Latency* lebih tinggi; butuh *query cluster*

Best practice

Gold di *lakehouse* tetap sumber kebenaran; *serving store* hanya salinan yang bisa dibangun ulang kapan saja oleh *pipeline* yang sama.

Memuat gold ke *serving store*: dua cara ringkas

DuckDB → **PostgreSQL** (cocok untuk UTS)

```
INSTALL postgres; LOAD postgres;
ATTACH 'host=pg dbname=serving user=lab'
  AS pg (TYPE postgres);
CREATE OR REPLACE TABLE pg.gold_harian_cabang AS
  SELECT * FROM
  read_parquet('s3://gold/harian_cabang/*.parquet');
```

Parquet di MinIO → **ClickHouse** (cocok untuk UAS)

```
INSERT INTO gold_harian_cabang
SELECT * FROM s3(
  'http://minio:9000/gold/harian_cabang/*.parquet',
  'KEY', 'SECRET', 'Parquet');
```

- Jadikan langkah ini *task* terakhir di DAG (setelah *data quality test* gold lolos), dengan `CREATE OR REPLACE` atau *partition* per tanggal agar *idempotent*.
- Metabase atau Superset cukup terhubung ke *serving store*, tidak ke bronze/silver.
- Sintaks contoh; cocokkan dengan versi DuckDB dan ClickHouse di lab.

Immutable storage dan *ledger*

Mengapa ada data yang tidak boleh diubah

- ***Audit trail***: auditor perlu yakin catatan transaksi tidak disunting diam-diam, termasuk oleh admin *database* sendiri.
- ***Tamper-evidence***: kalau ada yang mengubah, perubahan itu harus **ketahuan**, walaupun pelakunya punya hak akses.
- **Retensi regulasi**: bukti transaksi wajib disimpan bertahun-tahun dalam bentuk *write once, read many* (WORM).
- ***Registry* kredit karbon dan MRV**: satu ton karbon tidak boleh dijual dua kali (*double counting*); riwayat penerbitan dan *retirement* harus bisa ditelusuri.
- **Jejak akad pembiayaan**: isi akad murabahah yang disepakati nasabah dan bank tidak boleh berubah setelah ditandatangani.

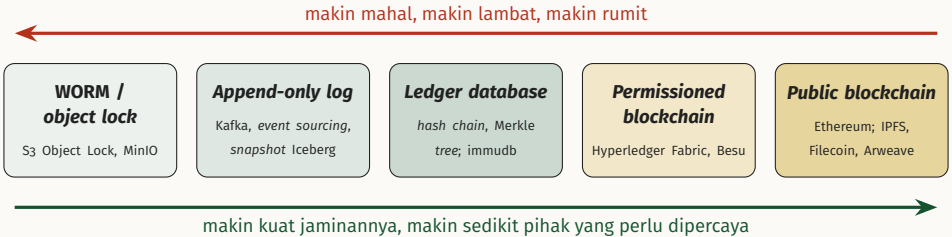
Tiga sifat yang sering tertukar

Immutable	Tidak bisa diubah setelah ditulis (dipaksa oleh sistem)
Tamper-evident	Kalau diubah, ada bukti kriptografis yang gagal diverifikasi
Decentralized	Tidak ada satu pihak yang memegang kendali penuh atas catatan

Ringkasan

Makin ke bawah makin kuat jaminannya, tetapi makin mahal, makin lambat, dan makin rumit dioperasikan.

Spektrum: dari yang ringan ke yang berat



Pertanyaan yang benar bukan “pakai blockchain atau tidak”, tetapi **siapa yang harus dicegah mengubah data, dan siapa yang harus bisa membuktikannya**.

WORM dan *object lock*

- **S3 Object Lock** (juga ada di MinIO): objek versi tertentu tidak bisa dihapus atau ditimpa sampai masa retensinya habis. Butuh *bucket versioning*.
- Mode ***governance***: pengguna dengan izin khusus masih bisa melonggarkan retensi. Cocok untuk uji coba dan kebijakan internal.
- Mode ***compliance***: **tidak ada** yang bisa menghapus sebelum retensi habis, termasuk akun *root*. Salah set durasi berarti membayar *storage* sampai habis.
- ***Legal hold***: kunci tanpa batas waktu, misalnya selama sengketa, dilepas manual.

Batasnya

WORM mencegah perubahan di satu penyedia, tetapi pemilik akun penyedia itu tetap pihak yang Anda percaya. Tidak ada bukti kriptografis untuk pihak luar.

```
# MinIO: bucket dengan object lock
mc mb --with-lock lab/arsip-akad
# retensi default 10 tahun, mode compliance
mc retention set --default COMPLIANCE 3650d \
   lab/arsip-akad
mc cp akad-0001.pdf lab/arsip-akad/
# coba hapus -> ditolak sampai retensi habis
mc rm lab/arsip-akad/akad-0001.pdf
# legal hold untuk satu objek
mc legalhold set lab/arsip-akad/akad-0001.pdf
```

Sintaks `mc` bisa berbeda antarversi; cek `mc retention --help`. Di lab pakai mode *governance* dan durasi pendek.

Append-only log dan *event sourcing*

- ***Event sourcing***: yang disimpan adalah **urutan kejadian** (`RekeningDibuka`, `Setor`, `Tarik`), bukan hanya saldo terakhir. Saldo dihitung ulang dari kejadian.
- Kafka menyimpan *log* yang hanya ditambah di ujung; tiap *message* punya *offset* tetap (P11).
- *Snapshot* Iceberg juga tidak pernah ditimpa: tiap *commit* membuat *snapshot* baru, yang lama bisa dibaca dengan *time travel* (P4).
- Koreksi dilakukan dengan **kejadian pembalik** (*reversal*), bukan menghapus baris lama, persis seperti jurnal akuntansi.

Immutable belum tentu *tamper-evident*

Admin Kafka bisa menghapus *topic*, mengubah retensi, atau menulis ulang isi *partition*. *Snapshot* Iceberg bisa dikedaluwarsakan (*expire snapshots*) dan *file*-nya dihapus. Keduanya tidak menyediakan bukti kriptografis bahwa riwayat tidak disunting.

Ringkasan

Pola ini sudah cukup untuk sebagian besar kebutuhan audit internal, asalkan akses admin dibatasi dan dicatat (P14).

Hash chain dan Merkle *tree*: membuat perubahan ketahuan

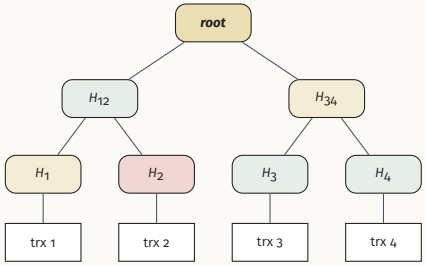
Hash chain: tiap catatan menyimpan *hash* catatan sebelumnya.

$$h_i = H(h_{i-1} \parallel r_i)$$

r_i: catatan ke-i; h_i: *hash*-nya; H: fungsi *hash* (misalnya SHA-256); $\parallel$ (dibaca "sambung"): penyambungan dua deret byte. Mengubah satu catatan lama mengubah semua *hash* sesudahnya; cukup simpan h_{terakhir} di tempat lain (dikirim ke auditor, dicetak di laporan) untuk mendeteksinya.

Merkle *tree*: *hash* disusun berpasangan sampai satu *root hash*.

- ***Inclusion proof***: membuktikan satu catatan ada di dalam pohon cukup dengan $\approx \log_2 n$ *hash* (n: jumlah catatan; $\log_2$: logaritma basis 2), bukan seluruh data.
- ***Consistency proof***: membuktikan pohon baru adalah perpanjangan pohon lama, tanpa ada yang disunting di tengah.



Membuktikan `trx 2` ada: kirim H_1 dan H_{34} (kuning). Penerima menghitung $H_2 \rightarrow H_{12} \rightarrow$ *root* dan mencocokkan dengan *root* yang ia simpan.

Sumber: Pola yang sama dipakai Certificate Transparency (RFC 6962) dan blockchain.

Ledger database: tamper-evident tanpa blockchain

- **immudb** (*open source*, Codenotary): *database key-value* dan SQL yang menyimpan setiap penulisan di Merkle *tree*. Klien memverifikasi bukti kriptografis saat membaca dan menulis (*verified read*/*write*).
- Nilai lama tidak ditimpa: menulis kunci yang sama membuat versi baru, riwayatnya tetap bisa dibaca.
- Klien menyimpan *root* terakhir yang sudah diverifikasi. Kalau server "menulis ulang sejarah", *consistency proof* berikutnya gagal.
- Satu operator, satu *database*: tidak perlu jaringan antarorganisasi.

Amazon QLDB pernah menjadi contoh populer layanan *ledger database* terkelola, tetapi AWS sudah menghentikannya; cek status terkini sebelum merujuk.

Alternatif sederhana: PostgreSQL + *hash chain*

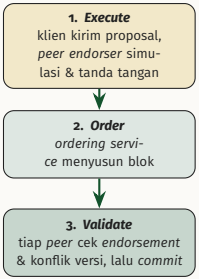
Tabel `jurnal` dengan kolom `hash_prev` dan `hash`, hanya boleh `INSERT` (cabut hak `UPDATE`/`DELETE`), dan *root* harian diterbitkan ke luar (surel ke auditor, *object lock*).

Ringkasan

Kalau hanya **satu** organisasi yang menulis dan pihak lain cukup diberi bukti, *ledger database* hampir selalu lebih tepat daripada blockchain.

Hyperledger Fabric: *permissioned blockchain* untuk konsorsium

- Peserta dikenal: identitas dari *certificate authority* tiap organisasi (MSP).
- ***Peer*** menyimpan *ledger* dan menjalankan ***chaincode*** (*smart contract*) dalam Go, Java, atau JavaScript.
- ***Channel***: *ledger* terpisah untuk sekelompok organisasi. ***Private data collection***: data hanya dibagi ke sebagian anggota, *ledger* cukup menyimpan *hash*-nya.
- ***Endorsement policy***: transaksi sah bila disetujui organisasi tertentu, misalnya “bank DAN auditor”.
- ***Ordering service***: mengurutkan transaksi menjadi blok; Raft (*crash fault tolerant*), dan sejak versi 3.x juga BFT.
- Kini di bawah LF Decentralized Trust (dulu Hyperledger Foundation).



Urutan ***execute–order–validate*** berbeda dari blockchain klasik (*order–execute*): transaksi disimulasikan paralel lebih dulu, lalu yang konflik ditolak saat validasi.

Sumber: Androulaki dkk., *Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains*, EuroSys 2018.

Besu, *public blockchain*, dan *web3 storage*

Hyperledger Besu

Klien Ethereum (Java, *open source*) yang bisa ikut jaringan publik atau membentuk jaringan *permissioned* dengan konsensus QBFT. Cocok bila ingin *smart contract* Solidity dan perangkat Ethereum, tetapi anggotanya terbatas.

Public blockchain: Ethereum

Siapa pun bisa membaca dan menulis; keamanan dari *proof of stake*. *Smart contract* dijalankan semua *node*; setiap operasi dibayar dengan *gas*. Semua data **terbuka selamanya**, sehingga tidak cocok untuk data pribadi.

Web3 storage

- **IPFS**: *file* dialamatkan menurut isinya (*content addressing*), alamatnya disebut CID, yaitu turunan *hash* isi *file*. *File* yang sama selalu ber-CID sama; isi berubah berarti CID berubah. Tidak otomatis permanen: *file* hilang bila tidak ada yang *pin*.
- **Filecoin**: pasar *storage* di atas IPFS; penyedia dibayar dan harus membuktikan masih menyimpan data.
- **Arweave**: bayar sekali untuk *storage* yang dirancang permanen.

Sumber: Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008; Benet, *IPFS, Content Addressed, Versioned, P2P File System*, 2014.

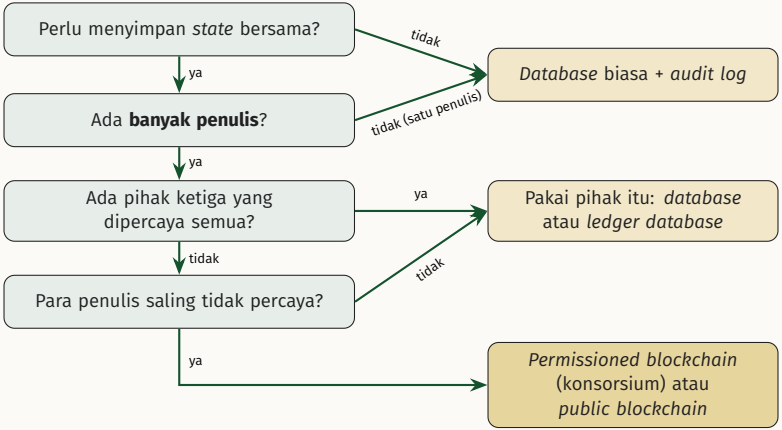
Trade-off: siapa yang dipercaya, dan berapa *cost*-nya

	Throughput	***Latency***	***Finality***	***Cost***	**Yang dipercaya**
WORM / *object lock*	tinggi	milidetik	langsung	murah	penyedia *storage*
Append-only log	sangat tinggi	milidetik	langsung	murah	operator dan admin
Ledger database	tinggi	milidetik	langsung	sedang	satu operator + bukti
Fabric / Besu	sedang	detik	setelah blok	tinggi	mayoritas anggota
Public blockchain	rendah	detik–menit	bertahap	*gas* per transaksi	protokol, bukan satu organisasi

Privasi: dari yang paling mudah dijaga (log internal) sampai yang paling sulit (*public blockchain*, terbuka untuk semua).

Kolom di atas bersifat kualitatif. Angka TPS yang beredar sangat bergantung pada *configuration*; ukur sendiri bila perlu.

Apakah benar butuh blockchain?



Sumber: Disederhanakan dari Wüst dan Gervais, *Do you need a Blockchain?*, CVCBT 2018.

Jebakan UU PDP: blockchain tidak bisa menghapus

Masalahnya

UU PDP memberi subjek data hak meminta penghapusan. Blockchain, dan juga *object lock* mode *compliance*, dirancang agar data **tidak bisa** dihapus. Data pribadi yang sudah masuk ke sana tidak bisa ditarik kembali.

Aturan praktis

- **Jangan pernah** menyimpan data pribadi *on-chain*: nama, NIK, alamat, nomor rekening.
- Simpan hanya *hash* atau *commitment* dari dokumen.
- Untuk data berentropi rendah (NIK, nomor telepon), pakai *salt* atau HMAC dengan kunci rahasia; *hash* polos bisa ditebak dengan mencoba semua kemungkinan.

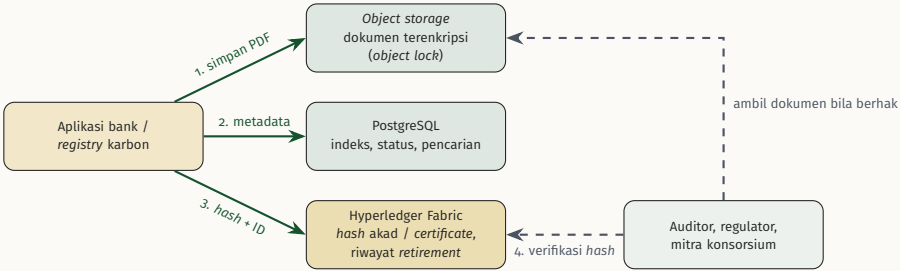
Data asli tetap *off-chain*

Dokumen disimpan di *object storage*, dienkripsi dengan kunci per nasabah. Untuk memenuhi hak penghapusan, hapus kuncinya (*crypto-shredding*, P14): *hash* di *ledger* tetap ada, tetapi tidak lagi menunjuk ke data yang bisa dibaca.

Ringkasan

Retensi WORM dan hak penghapusan harus diputuskan bersama tim kepatuhan **sebelum** sistem dibangun, agar siap ketika nasabah meminta (P15).

Arsitektur contoh: akad dan *certificate* kredit karbon



Mengapa terpisah: pencarian dan laporan cepat di PostgreSQL, dokumen besar murah di *object storage*, dan hanya **bukti kecil** yang masuk *ledger* bersama.

Mencegah *double counting*: *chaincode* menolak *retirement* kedua untuk ID *certificate* yang sama, dengan *endorsement* dari *registry* dan verifikator.

Praktikum mini: membuktikan perubahan terdeteksi

```
# 1) hash chain buatan sendiri
import hashlib, json
def h(prev, rec):
    s = prev + json.dumps(rec, sort_keys=True)
    return hashlib.sha256(s.encode()).hexdigest()
log, prev = [], "0" * 64
for r in [{"akad": 1, "nilai": 120},
          {"akad": 2, "nilai": 75}]:
    prev = h(prev, r); log.append((r, prev))
log[0][0]["nilai"] = 1  # coba sunting diam-diam
p = "0" * 64
for r, hh in log:
    p = h(p, r); print(p == hh)  # False: ketahuan
```

```
# 2) immudb di Docker:
# docker run -d -p 3322:3322 codenotary/immudb
from immudb import ImmudbClient
c = ImmudbClient("localhost:3322")
c.login("immudb", "immudb")     # kredensial lab
c.verifiedSet(b"akad:1", b"120")
c.verifiedSet(b"akad:1", b"130") # versi baru
print(c.verifiedGet(b"akad:1"))  # terverifikasi
print(c.history(b"akad:1", 0, 10, False))
```

Opsional: jalankan `fabric-samples/test-network` (`./network.sh up createChannel`, lalu `deployCC` contoh `asset-transfer-basic`).

Nama fungsi SDK dan perintah bisa berubah antarversi; cocokkan dengan dokumentasi immudb dan Fabric yang dipakai di kelas.

Praktikum

Praktikum: lima store dalam satu `docker-compose`

```
services:
  postgres:
    image: postgres:16
    environment: {POSTGRES_PASSWORD: lab}
    ports: ["5432:5432"]
  redis:
    image: redis:7
    ports: ["6379:6379"]
  scylla:
    image: scylladb/scylla
    command: --smp 1 --memory 1G
  clickhouse:
    image: clickhouse/clickhouse-server
    ports: ["8123:8123"]
  opensearch:
    image: opensearchproject/opensearch:2
    environment:
      discovery.type: single-node
      DISABLE_SECURITY_PLUGIN: "true"
```

- Batasi memori tiap kontainer agar laptop 8 GB tetap sanggup; jalankan bergiliran bila perlu.
- Data: transaksi sintetis bank syariah (skrip pembangkit dibagikan), misalnya 10 juta baris, 100 ribu rekening.
- Versi *image* di atas contoh; pakai yang dibagikan di kelas.
- *Security plugin* OpenSearch dimatikan **hanya untuk lab lokal**; jangan di produksi (P14).

Praktikum: mengukur dengan jujur

```python
import time, statistics, random
import psycopg

def ukur(fn, n=2000):
    for _ in range(100): fn()        # pemanasan
    t = []
    for _ in range(n):
        a = time.perf_counter(); fn()
        t.append((time.perf_counter() - a) * 1e3)
    t.sort()
    return {"p50": t[n // 2], "p99": t[int(n * .99)],
            "rata2": statistics.mean(t)}

con = psycopg.connect("postgresql://postgres:lab@localhost")
def saldo_pg():
    rid = random.randint(1, 100_000)
    con.execute("SELECT saldo FROM rekening WHERE id=%s",
                (rid,)).fetchone()

print(ukur(saldo_pg))
```

Empat pola akses

1. Saldo per rekening (*point lookup*)
2. 50 transaksi terakhir nasabah (*range scan*)
3. Total transaksi bulan ini (*aggregation*)
4. Keterangan memuat “zakat” (teks)

Laporkan P50, P99, dan *throughput* tulis per store dalam satu tabel. Tandai pasangan pola–store yang tidak masuk akal (misalnya *aggregation* di Redis) dan jelaskan mengapa.

Tugas untuk pertemuan berikutnya

1. Kumpulkan laporan praktikum: tabel P50/P99 untuk empat pola akses di lima store, *throughput* tulis, dan satu paragraf per pola tentang store pilihan Anda beserta alasannya.
2. Rancang *polyglot persistence* untuk bank syariah digital: store sumber kebenaran, store turunan, dan cara turunan dibangun ulang bila rusak.
3. Baca Kleppmann *Designing Data-Intensive Applications* Bab 3 dan Bab 6.
4. Buka lagi catatan Sister P7 (Sabtu lalu): *replication*, *quorum*, Raft, dan CFT vs BFT adalah mekanisme di balik pilihan *consistency* yang kita bahas hari ini.
5. Tugas besar UTS: *pipeline* bronze-silver-gold harus sudah berjalan *end-to-end*; pilih *serving store* untuk gold, siapkan *dashboard* atau *query*, dan latih demo dengan *checklist* di dek P08.

Pertemuan depan

UTS (P8, Senin 23 November): presentasi dan demo tugas besar "*Pipeline* ETL bronze-silver-gold". Satu perintah harus menyalakan seluruh sistem di laptop presentasi.

Terima kasih

Pertanyaan?