

Optimasi SQL & Arsitektur Query PostgreSQL

Pelatihan Divisi Pusdatin – Kemnaker RI

Indexing • Join Strategy • JSON • Parallel • FDW



Indexing Strategy

Memahami cara kerja index, B-Tree, GIN, dan kapan efektif digunakan.



Join Optimization

Teknik join optimal, anti-pattern, dan penggunaan Lateral Join.



JSON Processing

Transformasi JSON ke tabel flat untuk analisis data.



Parallel Query

Mekanisme dan tuning untuk kinerja query pada data besar.



Foreign Data Wrapper

Query antar database dan server (Cross-Server Join).

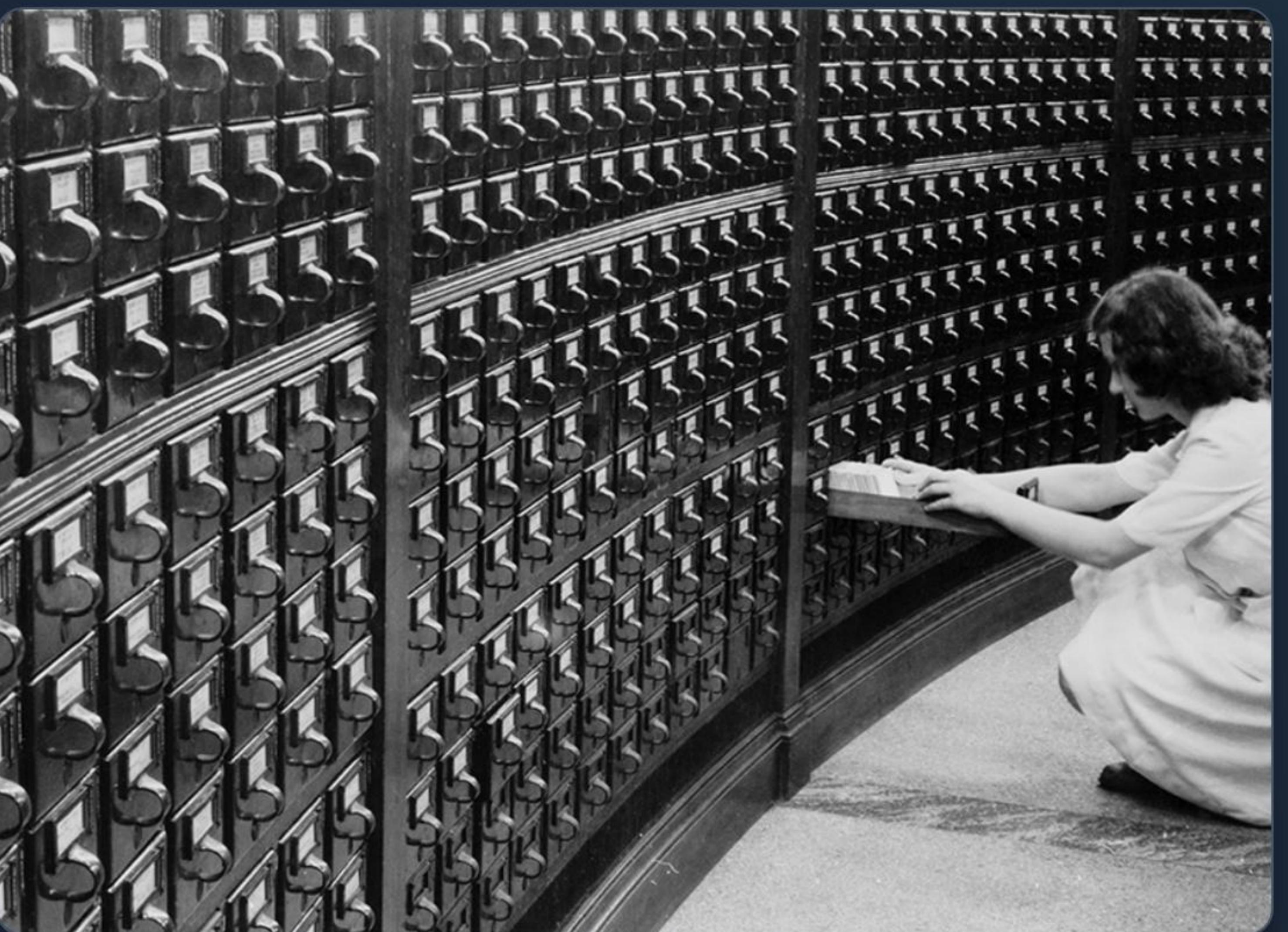
Dasar Indexing

Mengapa Index Penting?

Tanpa index, PostgreSQL menggunakan **Sequential Scan**, membaca seluruh baris data. Ini sangat lambat untuk tabel besar (>10 juta baris).

Index bekerja seperti **indeks di belakang buku** atau katalog perpustakaan, membuat pencarian menjadi *logarithmic*.

```
CREATE INDEX idx_nama ON pekerja(nama);
```



B-Tree & Hash Index

B-Tree Index

- ✓ **Default** dan paling banyak dipakai.
- ✓ Mendukung: `=, <, >, BETWEEN, ORDER BY`.
- ✓ **Cocok untuk:** NIK, Tanggal Lahir, Nama.

Hash Index

- ✓ Hanya mendukung operasi `=` (Exact Match).
- ✓ Lebih cepat dari B-Tree untuk pencarian data *high cardinality*.
- ✓ **Real Case Kemnaker:** Pencarian cepat berdasarkan NIK unik.

```
CREATE INDEX idx_nik_hash ON pekerja USING hash(nik);
```

GIN & GiST Index

GIN (Generalized Inverted Index)

Sangat kuat untuk tipe data kompleks dan multidimensi.

- ✓ **JSONB:** Mencari key/value dalam dokumen JSON.
- ✓ **Array:** Mencari elemen dalam array.
- ✓ **Full Text Search:** Pencarian teks tingkat lanjut.
- ✓ Operator: @> (contains), ? (exists).

GiST Index

Digunakan untuk data geometris dan pencarian kesamaan (similarity).

- ✓ **Geospasial (PostGIS):** Koordinat lokasi.
- ✓ **Trigram (pg_trgm):** Pencarian fuzzy (mirip kata).

```
CREATE INDEX idx_json ON pekerja USING gin(data);
```

BRIN Index (Big Data)

Block Range INdex (BRIN)

Tidak mengindex per baris, tapi membaca **range blok** halaman data. Sangat ringan.

- ✓ **Cocok untuk:** Tabel > 100 Juta row.
- ✓ **Syarat:** Data terurut fisik (Sequential), misal: *Timestamp Log* atau *Auto-increment ID*.

Real Case Log Presensi (300M Rows):

```
CREATE INDEX idx_waktu ON presensi USING brin(waktu);
```



Expression Index

PostgreSQL mengindeks **hasil dari sebuah fungsi**, bukan hanya nilai kolom mentah. Sangat berguna jika query Anda sering melakukan transformasi data.

Real Case: Pencarian email *Case Insensitive*. Tanpa expression index, fungsi `lower()` akan membatalkan penggunaan index biasa.

```
CREATE INDEX idx_lower_email  
ON users (lower(email));  
  
SELECT * FROM users  
WHERE lower(email) = 'abc@gmail.com';
```

ⓘ Query ini sekarang akan menggunakan index scan.

Partial Index (Selective)

Membuat index hanya untuk sebagian data yang memenuhi kondisi tertentu (WHERE clause).

20M

TOTAL PEKERJA

400K

STATUS 'AKTIF'

```
CREATE INDEX idx_aktif ON pekerja(nik) WHERE status = 'AKTIF';
```

Index ini jauh lebih kecil, hemat storage, dan sangat cepat untuk query data aktif.

Indexing: Do's & Don'ts

✓ Efektif Jika

- ✓ Kolom **High Cardinality** (banyak nilai unik).
- ✓ Query mengambil **< 10-30%** total data.
- ✓ Digunakan untuk kolom **JOIN** dan **ORDER BY**.
- ✓ Sering digunakan di klausa **WHERE**.

✗ Hindari Jika

- ✗ Tabel sangat kecil (< 2000 baris).
- ✗ Kolom **Low Cardinality** (Gender L/P, Boolean).
- ✗ Kolom sering di-**UPDATE** (Overhead tinggi).
- ✗ Membuat index untuk semua kolom (Bloating).

Arsitektur Join

Standard Join

INNER JOIN

LEFT / RIGHT JOIN

FULL OUTER JOIN

Advanced Join

CROSS JOIN

LATERAL JOIN

NATURAL JOIN

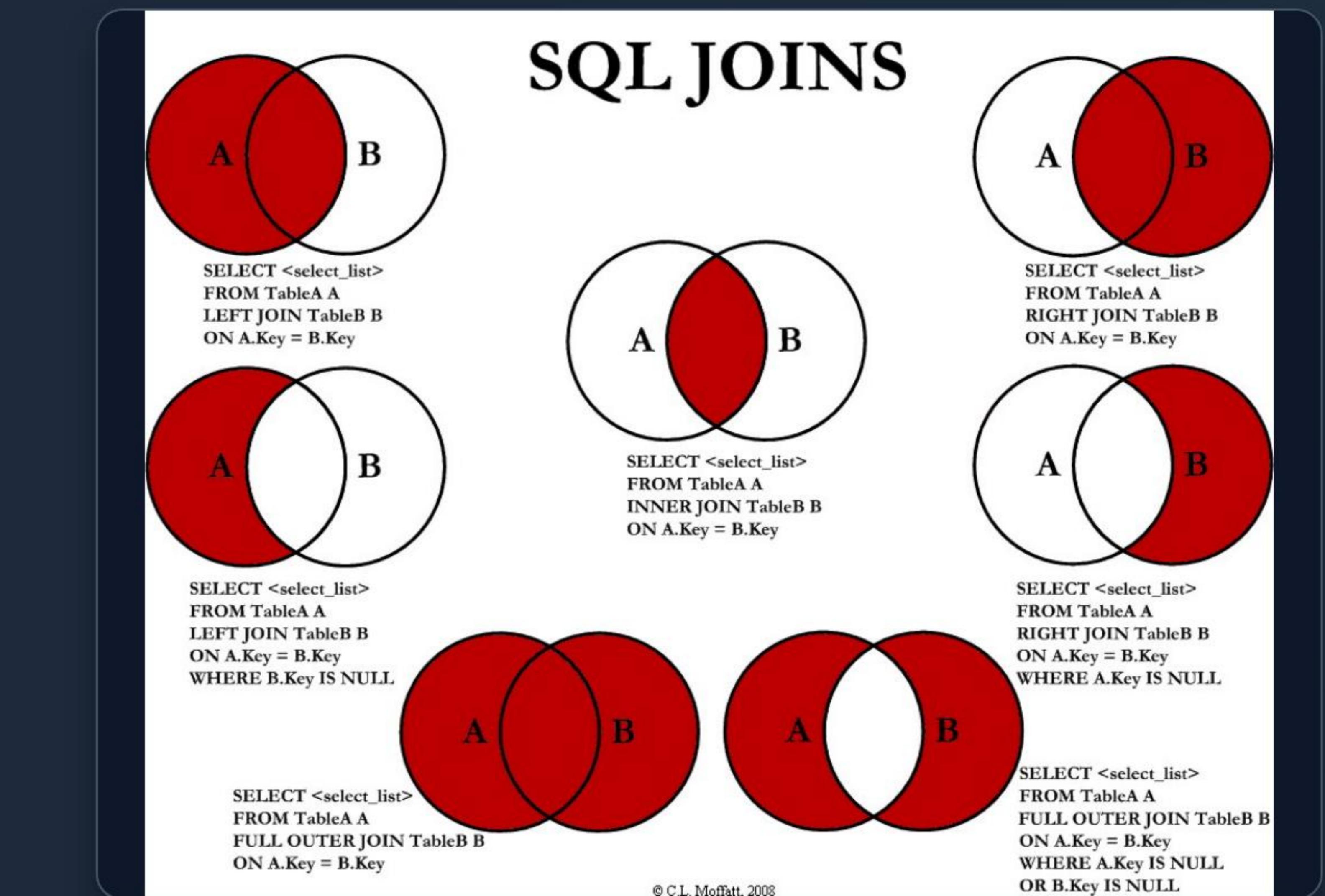
Filter Join

SEMI JOIN (EXISTS)

ANTI JOIN (NOT EXISTS)

Visualisasi Join

- ✓ **INNER JOIN:** Hanya baris yang cocok di kedua tabel.
Irisan himpunan.
- ✓ **LEFT JOIN:** Semua data kiri + data kanan yang cocok.
Jika tidak ada yang cocok, kanan = NULL.
- ✓ **RIGHT JOIN:** Kebalikan LEFT JOIN (Jarang dipakai,
sebaiknya ubah urutan tabel dan pakai LEFT JOIN).
- ✓ **FULL OUTER JOIN:** Gabungan semua data. Sangat
berat (Resource Intensive).



Lateral Join (Advanced)

Konsep

LATERAL memungkinkan subquery di kanan join untuk mereferensikan kolom dari baris di kiri join.

Sangat berguna untuk query "**Top-N per Group**".

Real Case: Mengambil 1 log aktivitas *terbaru* untuk setiap user.

```
SELECT u.id, j.*  
FROM users u  
LEFT JOIN LATERAL (  
    SELECT * FROM log  
    WHERE log.user_id = u.id  
    ORDER BY created_at DESC  
    LIMIT 1  
) j ON TRUE;
```

SEMI JOIN (EXISTS)

Digunakan untuk mengecek **keberadaan** data di tabel lain tanpa mengambil datanya.

⚡ Performance Tip

EXISTS akan berhenti scanning segera setelah menemukan **1 match**. Jauh lebih cepat daripada INNER JOIN pada dataset besar jika hanya butuh filtering.

```
SELECT p.*  
FROM pekerja p  
WHERE EXISTS (  
    SELECT 1  
    FROM pelatihan h  
    WHERE h.nik = p.nik  
);
```

Hanya ambil data pekerja yang pernah ikut pelatihan.

ANTI JOIN (NOT EXISTS)

⚠ Hindari NOT IN !

Jika subquery mengandung **NULL**, NOT IN akan gagal memberikan hasil yang benar. Selalu gunakan NOT EXISTS atau LEFT JOIN ... WHERE IS NULL.

```
SELECT p.* FROM pekerja p
WHERE NOT EXISTS (
    SELECT 1 FROM bantuan b
    WHERE b.nik = p.nik AND b.tahun = 2024
);
```

UNION vs UNION ALL

FITUR	UNION	UNION ALL
Duplikat	Dihapus (Distinct)	Disimpan
Proses	Melakukan SORT berat	Langsung Append
Performa	Lebih Lambat	Sangat Cepat
Usage	Jika butuh data unik	Default choice (Recommended)

Join Best Practices

✓ Do's

- ✓ Pastikan kolom Foreign Key di-**Index**.
- ✓ Gunakan **EXPLAIN** untuk melihat algoritma join (Nested Loop, Hash Join, Merge Join).
- ✓ Gunakan **LIMIT** saat debugging join tabel besar.

✗ Don'ts

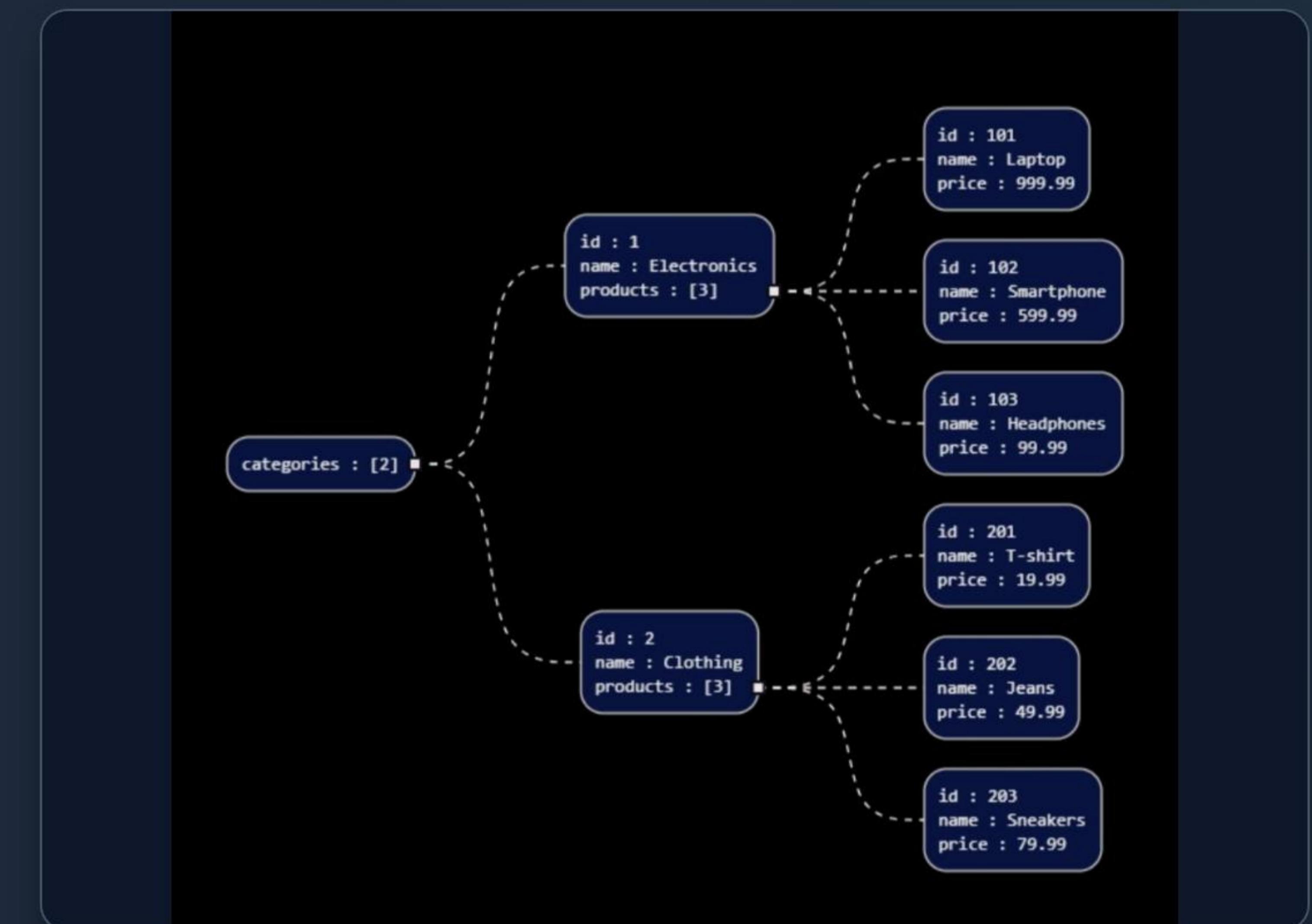
- ✗ Join pada hasil fungsi: ON `lower(a.nama) = lower(b.nama)` (Kecuali ada index).
- ✗ Menggunakan **SELECT *** pada tabel dengan banyak kolom.
- ✗ Filter di **HAVING** padahal bisa di **WHERE**.

JSON to Flat: jsonb_each

PostgreSQL sangat powerful untuk data semi-terstruktur.

jsonb_each(): Mengurai object JSON menjadi baris key-value.

```
SELECT key, value
FROM jsonb_each(
  '{"nama":"budi","usia":30}' :: jsonb
);
```



JSON to Flat: jsonb_array_elements

Jika data berbentuk Array [], gunakan fungsi ini untuk menjadikannya baris tabel (unnest).

Input JSON

```
'[100, 200, 300]'
```

Output Query

ELEM

```
100
```

```
200
```

```
300
```

```
SELECT elem FROM jsonb_array_elements('[100,200,300]'::jsonb) elem;
```

Flatten Nested JSON

Menggabungkan ekstraksi field dan unnesting array dalam satu query kompleks.

```
SELECT  
    x->>'id' AS id,  
    y->>'nama' AS nama  
FROM mytable,  
    jsonb_array_elements(data->'pegawai') y,  
    jsonb_each(data) x;
```

Optimasi JSON Index

Gunakan **GIN** index dengan operator class `jsonb_path_ops` untuk performa maksimal pada operasi `contains (@>)`.

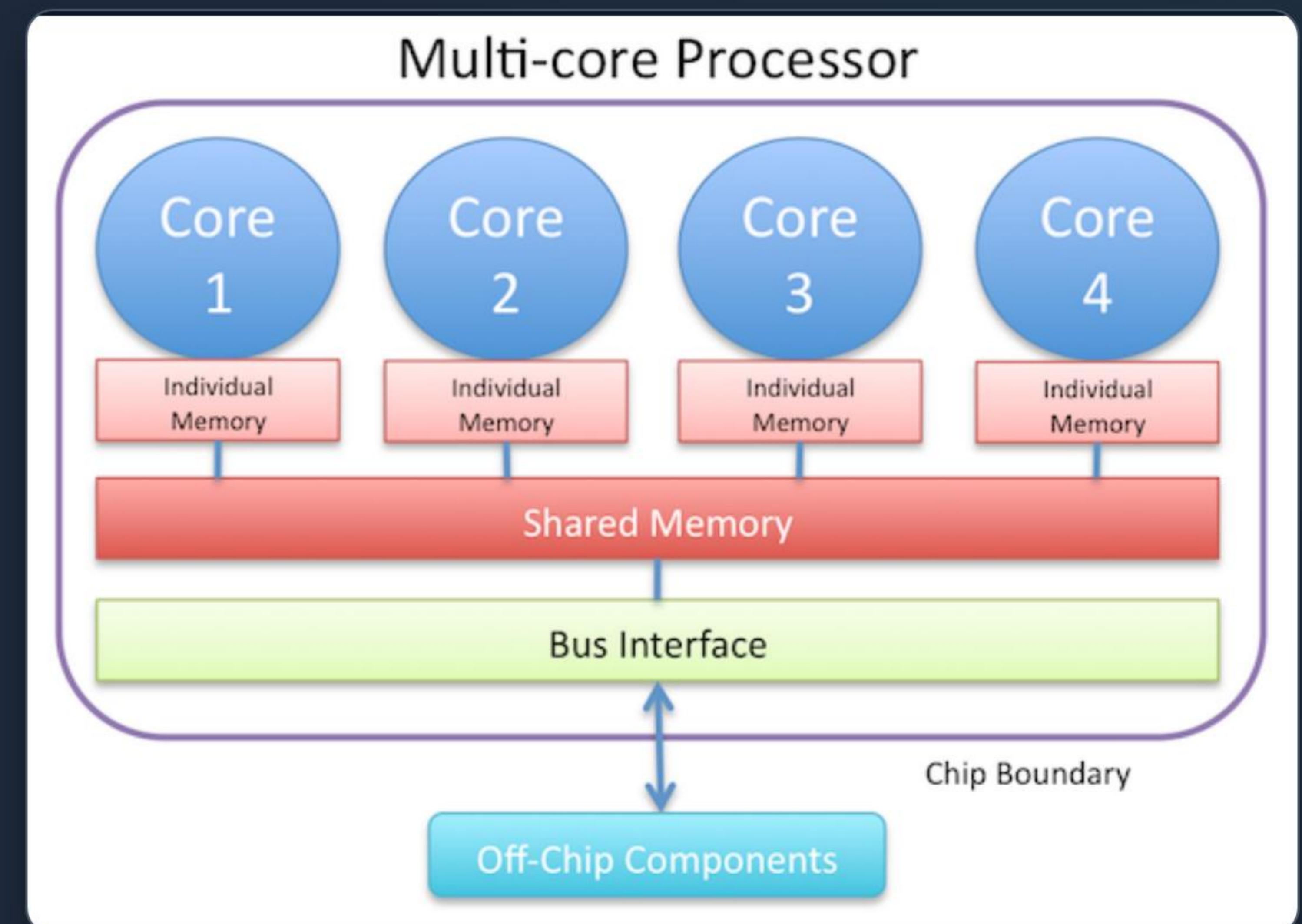
```
CREATE INDEX idx_json_gin  
ON laporan  
USING gin(data jsonb_path_ops);
```

Parallel Query Architecture

PostgreSQL dapat memecah satu query menjadi beberapa proses worker untuk dieksekusi secara bersamaan pada CPU multi-core.

Kapan Triggered?

- ✓ Seq Scan tabel besar.
- ✓ Join tabel besar (Hash Join).
- ✓ Agregasi (COUNT, SUM).



Jenis Parallelism



Parallel Seq Scan

Worker membaca blok data yang berbeda dari tabel yang sama secara simultan.



Parallel Join

Biasanya terjadi pada Hash Join. Tabel di-hash secara paralel oleh worker.



Parallel Aggregate

Perhitungan parsial dilakukan per worker, lalu digabung (Finalize Aggregate).

EXPLAIN ANALYZE: Workers Planned: 4, Workers Launched: 4

Tuning Parallel Query

Parameter Utama

- ✓ `max_worker_processes`: Limit total worker DB.
- ✓ `max_parallel_workers_per_gather`: Limit worker per satu query.
- ✓ `min_parallel_table_scan_size`: Batas ukuran tabel untuk trigger parallel scan.

🚫 Parallel Gagal Jika:

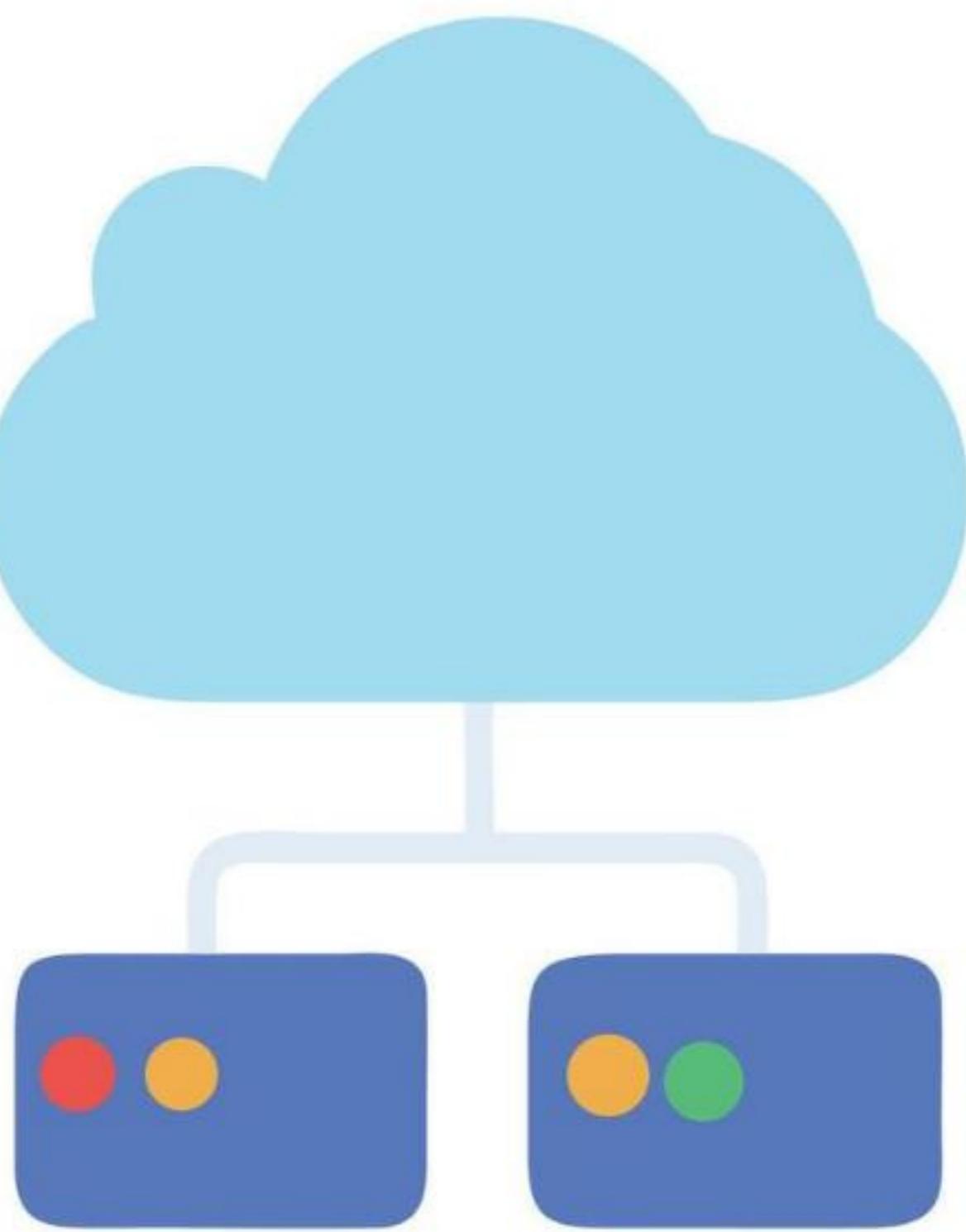
- ✓ Query menulis data (INSERT/UPDATE) - *sebagian besar*.
- ✓ Menggunakan fungsi yang tidak **IMMUTABLE**.
- ✓ Tabel terlalu kecil (overhead worker > benefit).
- ✓ Cursor atau LIMIT kecil.

Foreign Data Wrapper (FDW)

Konsep Dasar

Memungkinkan PostgreSQL mengakses data di server eksternal (Postgres lain, MySQL, Oracle, CSV) seolah-olah itu adalah tabel lokal.

Use Case Pusdatin: Integrasi antar server database Kemnaker tanpa perlu ETL/API kompleks untuk query ad-hoc.



Langkah 1: Instalasi FDW

Aktifkan extension dan definisikan server target.

```
-- 1. Enable Extension  
CREATE EXTENSION postgres_fdw;  
  
-- 2. Define Remote Server  
CREATE SERVER srv_kemnaker  
FOREIGN DATA WRAPPER postgres_fdw  
OPTIONS (host '10.10.10.2', dbname 'db_utama');
```

Langkah 2: Mapping & Import

Petakan user lokal ke user remote, lalu import skema.

```
-- 3. User Mapping  
CREATE USER MAPPING FOR current_user  
SERVER srv_kemnaker  
OPTIONS (user 'remote_usr', password 'xxx');
```

```
-- 4. Import Tables  
IMPORT FOREIGN SCHEMA public  
FROM SERVER srv_kemnaker  
INTO fdw_schema;
```

Cross-Server Query

```
SELECT a.nik, b.nama  
FROM lokal.peserta a  
JOIN fdw_schema.peserta b  
ON a.nik = b.nik;
```

Query ini menggabungkan data lokal dengan data dari server 10.10.10.2 secara transparan.

Limitasi FDW

- ✓ **Latensi Jaringan:** Sangat mempengaruhi kinerja join besar.
- ✓ **Pushdown:** Tidak semua filter/aggregate dikirim ke server remote (data mentah ditarik dulu).
- ✓ **Parallelism:** Seringkali tidak bisa parallel scan di sisi remote.

Ringkasan Materi



Indexing

B-tree, Hash, GIN, Partial



Joins

Exists, Lateral, Union All



JSON

Flattening & Indexing



Parallel

Workers & Tuning



FDW

Cross-Server Query

Sesi Tanya Jawab

Image Sources



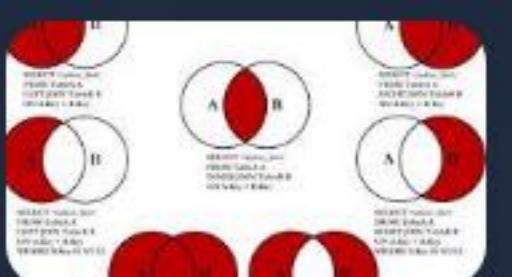
<https://americanlibrariesmagazine.org/wp-content/uploads/2015/11/0116editions1.jpg>

Source: americanlibrariesmagazine.org



https://png.pngtree.com/png-vector/20241217/ourmid/pngtree-futuristic-server-rack-glowing-with-blue-leds-png-image_14807708.png

Source: pngtree.com



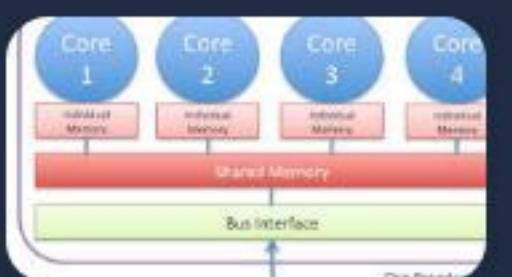
<https://i.sstatic.net/UI25E.jpg>

Source: stackoverflow.com



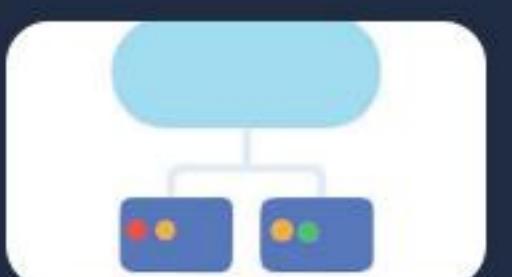
https://ik.imagekit.io/qsj9rwkvv/IMG_20240728_232454.jpg?updatedAt=1729778304662

Source: jsonviewer.tools



<https://www.cs.wustl.edu/~jain/cse567-11/ftp/multcore/fig1.png>

Source: www.cs.wustl.edu



https://static.vecteezy.com/system/resources/previews/062/440/658/non_2x/cloud-computing-server-network-connection-icon-on-black-vector.jpg

Source: www.vecteezy.com