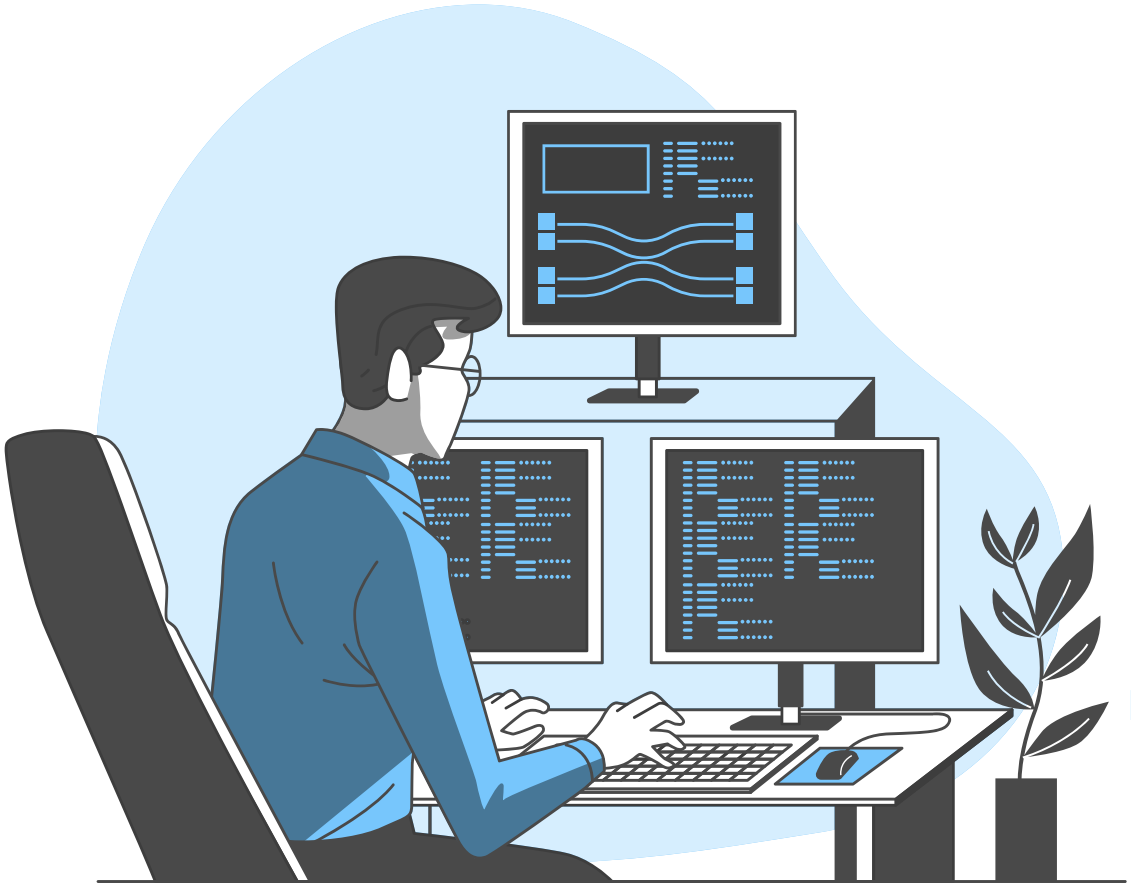
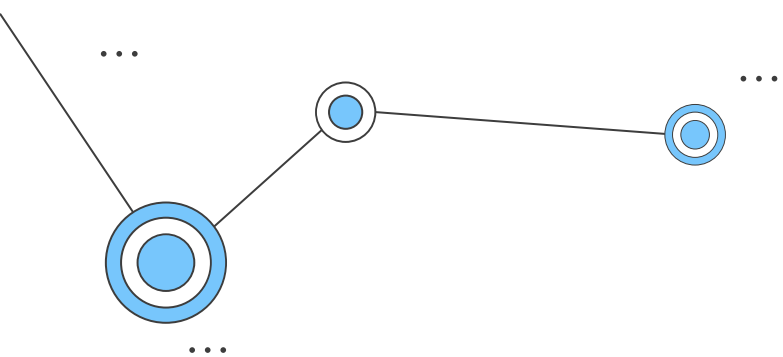


# BASIS DATA LANJUT

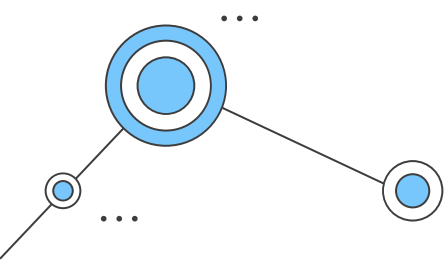
## Pertemuan 06

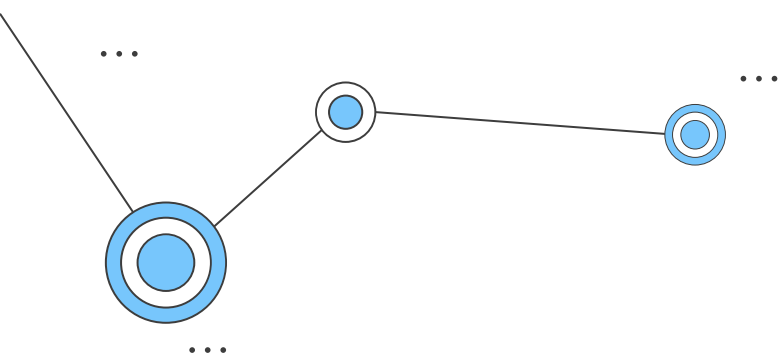
Function, View, Materialize View, & Store Procedure



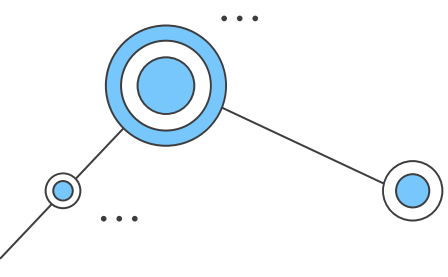
# OUTLINE

- Function
- View
- Materialized View
- Stored Procedure





# Function di PostgreSQL



# Function

- Function di PostgreSQL adalah blok kode yang dapat dipanggil untuk menjalankan sekumpulan perintah SQL maupun logika pemrograman.

- Function membantu **modularisasi kode**, mengurangi duplikasi, serta meningkatkan kinerja dengan mengeksekusi logika di sisi server

# Jenis Function di PostgreSQL

Terdapat 3 jenis function di PostgreSQL

| Jenis | Deskripsi |
|---|---|
| SQL Function | • Ditulis dengan SQL murni.<br>• Cocok untuk operasi sederhana. |
| PL/pgSQL Function | • Bahasa *procedural* bawaan PostgreSQL.<br>• Mendukung variabel, kondisi (IF), loop, error handling.<br>• Paling umum digunakan. |
| External Language Function | PostgreSQL bisa diperluas dengan bahasa lain:<br>• **plpythonu / plpython3u** → Python<br>• **plperl** → Perl<br>• **plv8** → JavaScript (V8 engine)<br>• **pljava** → Java<br>• **C Function** → untuk kinerja sangat tinggi (ditulis dalam C, di-*compile* lalu di-*load* ke PostgreSQL). |

# SQL Function – Sintaks

SQL Function ditulis langsung dengan perintah SQL tanpa logika tambahan.
Cocok untuk operasi query sederhana seperti perhitungan, transformasi data, atau pemanggilan ulang query yang sering dipakai.
Karena PostgreSQL tidak perlu mengeksekusi *procedural engine (plpgsql)*, eksekusinya lebih cepat.

**Sintaks Dasar:**

```
CREATE OR REPLACE FUNCTION nama_function(parameter tipe)
RETURNS tipe
AS $$
    SELECT ...;
$$ LANGUAGE sql;
```

**Contoh:**

```
CREATE OR REPLACE FUNCTION luas_lingkaran(radius numeric)
RETURNS numeric AS $$
    SELECT 3.14159 * radius * radius;
$$ LANGUAGE sql;

-- Contoh pemanggilan saat query
SELECT luas_lingkaran(7);
```

```
CREATE OR REPLACE FUNCTION cek_tinggi_badan(tinggi numeric)
RETURNS text AS $$
    SELECT case when tinggi < 160 then 'Pendek'
                when tinggi between 160 and 170 then 'Sedang'
                else 'Tinggi' end;
$$ LANGUAGE sql;

-- Contoh pemanggilan saat query
select  nim, nama, tinggi_badan, cek_tinggi_badan(tinggi_badan) as cek_tinggi
from    m_mahasiswa
limit   10;
```

# PL/pgSQL Function

PL/pgSQL  ***Procedural Language for PostgreSQL SQL*** merupakan bahasa prosedural bawaan PostgreSQL yang memperluas kemampuan SQL standar sehingga kita bisa membuat logika program langsung di dalam database. *PL/pgSQL function* mendukung logika kompleks.

## Sintaks Dasar:

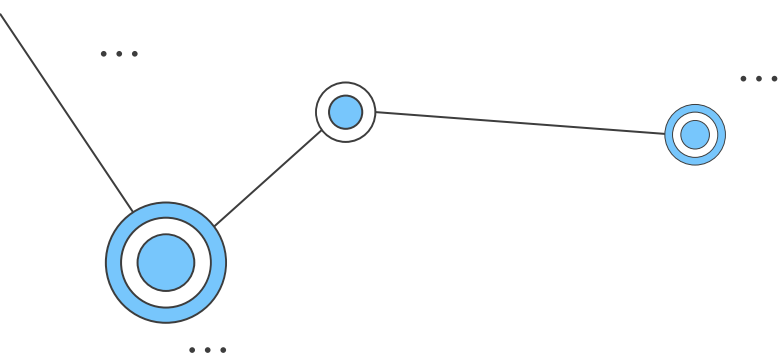
```
CREATE OR REPLACE FUNCTION nama_function(parameter tipe)
RETURNS tipe AS $$
DECLARE
    -- deklarasi variabel
BEGIN
    -- logika / query
    RETURN ...;
END;
$$ LANGUAGE plpgsql;
```

## Contoh:

```
CREATE OR REPLACE FUNCTION hitung_bmi(berat numeric, tinggi numeric)
RETURNS text AS $$
DECLARE
    hasil text;
    bmi numeric;
BEGIN
    tinggi := tinggi / 100; -- konversi tinggi dari cm ke m
    bmi := berat / (tinggi * tinggi); -- perhitungan BMI

    IF(bmi < 16) THEN
        hasil := 'Kurus Parah';
    ELSEIF(bmi >= 16 and bmi < 17) THEN
        hasil := 'Kurus Sedang';
    ELSEIF(bmi >= 17 and bmi < 18.5) THEN
        hasil := 'Kurus';
    ELSEIF(bmi >= 18.5 and bmi < 25) THEN
        hasil := 'Normal';
    ELSEIF(bmi >= 25 and bmi < 30) THEN
        hasil := 'Gemuk';
    ELSEIF(bmi >= 30 and bmi < 35) THEN
        hasil := 'Obesitas Kelas 1';
    ELSEIF(bmi >= 35 and bmi < 40) THEN
        hasil := 'Obesitas Kelas 2';
    ELSE
        hasil := 'Obesitas Kelas 3';
    END IF;

    RETURN hasil;
END;
$$ LANGUAGE plpgsql;
```

# Kesimpulan

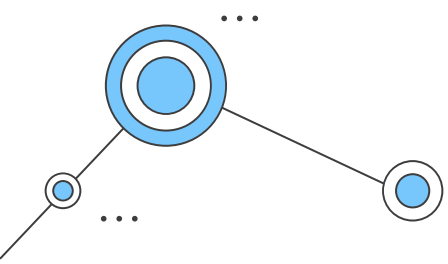
| Jenis | Fungsi |
|---|---|
| SQL Function | Gunakan **SQL Function (SQL murni)** ketika:<br>❑ Hanya butuh satu query sederhana.<br>❑ Fungsi bisa ditulis dalam bentuk **deterministic** (hasil selalu sama).<br>❑ Ingin memaksimalkan kinerja *query planner* PostgreSQL.<br>❑ Butuh fungsi kecil untuk dipakai ulang di query, misalnya perhitungan, manipulasi string, atau formatting. |
| PL/pgSQL Function | Gunakan **PL/pgSQL Function** ketika:<br>❑ Perlu logika **lebih dari satu query**.<br>❑ Butuh variabel, perulangan, percabangan, atau error handling.<br>❑ Membuat **trigger function** untuk validasi atau audit data.<br>❑ Ingin menjaga **konsistensi business logic** di dalam database, bukan hanya di aplikasi.<br>❑ Melakukan **proses batch** atau **ETL** langsung di server database. |



# View

pada

# PostgreSQL



# View – Definisi

***View*** adalah **objek basis data** di PostgreSQL yang merepresentasikan **hasil query (SELECT)** sebagai sebuah tabel virtual.

***View*** tidak menyimpan data secara fisik, melainkan hanya menyimpan definisi query.
Ketika ***View*** dipanggil, PostgreSQL akan menjalankan *query* yang mendasarinya, lalu menampilkan hasilnya seperti tabel biasa.

# View – Tujuan dan Kegunaan

- ***Menyederhanakan Query***
  - ✓ *Query yang kompleks bisa dibungkus menjadi sebuah View.*
  - ✓ *Pengguna cukup memanggil View tanpa harus menulis ulang query panjang.*
- ***Meningkatkan Keamanan***
  - ✓ *View bisa digunakan untuk membatasi akses. Misalnya, user hanya boleh melihat kolom tertentu dari tabel, bukan semua data.*
- ***Mendukung Reusability***
  - ✓ *View bisa digunakan berulang kali oleh banyak query atau aplikasi.*
- ***Membantu Abstraksi Data***
  - ✓ *Perubahan struktur tabel di backend bisa disembunyikan dari aplikasi, cukup dengan menjaga definisi View tetap sama*

# View – sintaks

**Sintak Dasar:**

```
create or replace view <nama_view> as
<-- query sql diakhiri titik koma (;) -->
```

**Contoh:**

```
create or replace view vw_mahasiswa as
select  m.mahasiswa_id, m.nim, m.nama, p.prodi_nama, j.jurusan_nama
from    m_mahasiswa m
join    m_prodi p on p.prodi_id = m.prodi_id
join    m_jurusan j on j.jurusan_id = p.jurusan_id;

-- Contoh pemanggilan saat query
select * from vw_mahasiswa;
```

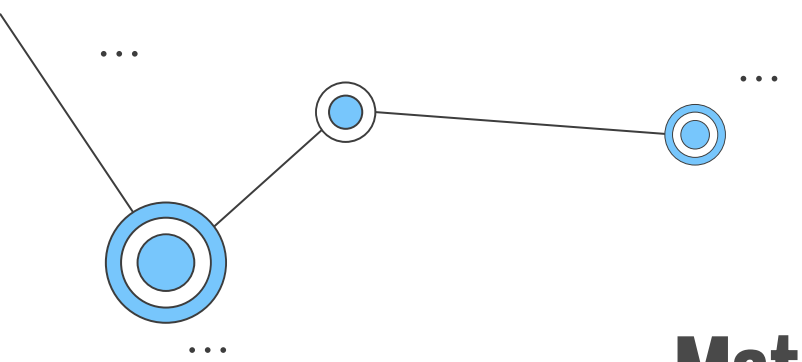
View ***vw_mahasiswa*** merupakan query representatif untuk membungkus query asli, sehingga di DB tidak ada table bernama ***vw_mahasiswa***.

```
select  m.mahasiswa_id, m.nim, m.nama, p.prodi_nama, j.jurusan_nama
from    m_mahasiswa m
join    m_prodi p on p.prodi_id = m.prodi_id
join    m_jurusan j on j.jurusan_id = p.jurusan_id;
```

# View – Kelebihan dan Kekurangan

| Kelebihan | Kekurangan |
| --- | --- |
| ❑ Menyederhanakan query kompleks.<br>❑ Bisa digunakan sebagai lapisan keamanan.<br>❑ Meningkatkan keterbacaan kode SQL.<br>❑ Mendukung abstraksi data (aplikasi tidak langsung berinteraksi dengan tabel asli) | ❑ View biasa bisa **lambat** jika query dasarnya kompleks (seperti banyak join ke table lain, banyak melakukan operasi, kurang menerapkan proses indexing pada table asli). |

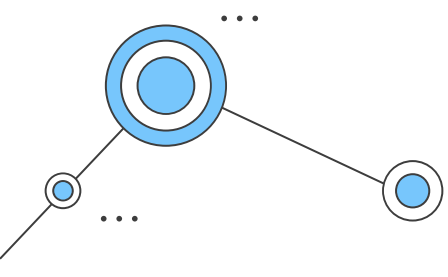
View di PostgreSQL adalah **tabel virtual** yang menyimpan query, bukan data. Cocok untuk menyederhanakan query, menjaga keamanan, dan menyediakan abstraksi data.



# Materialized View

pada

# PostgreSQL



# Materialized View – Definisi

***Materialized View (MV)*** adalah objek basis data di PostgreSQL yang mirip dengan ***View***, tetapi menyimpan hasil ***query*** secara **fisik** di dalam disk.

Berbeda dengan ***View*** (*regular view*) yang **hanya menyimpan *query*** dan dieksekusi ulang setiap kali dipanggil, ***Materialized View*** **menyimpan data snapshot dari hasil *query*** ke sebuah table baru.

Ciri Utama ***Materialized View***:

- ❑ Menyimpan hasil query di dalam tabel fisik.
- ❑ Tidak otomatis ter-update ketika data sumber berubah.
- ❑ Harus diperbarui secara manual dengan perintah:

```
REFRESH MATERIALIZED VIEW <nama_mv>;
-- atau --
REFRESH MATERIALIZED VIEW CONCURRENTLY <nama_mv>;
```

# Materialized View – sintaks

**Sintak Dasar:**

```
CREATE MATERIALIZED VIEW <nama_mv> AS
<-- query sql diakhiri titik koma (;) -->

-- Membaca materialized view
SELECT * FROM <nama_mv>;

-- Refresh isi materialized view jika data sumber berubah
REFRESH MATERIALIZED VIEW <nama_mv>;

-- Refresh sambil tetap dapat diakses (butuh unique index)
REFRESH MATERIALIZED VIEW CONCURRENTLY <nama_mv>;
```

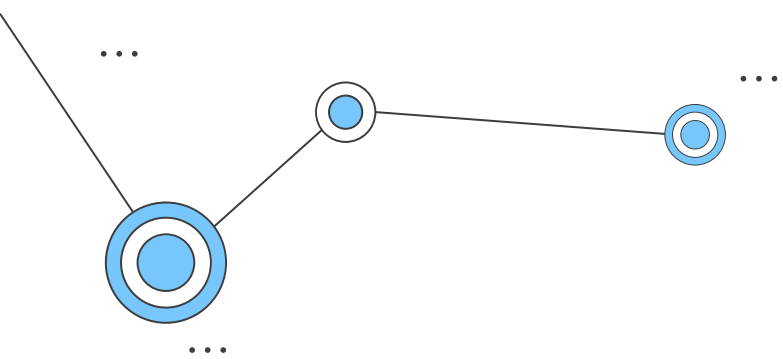
**Contoh:**

```
CREATE MATERIALIZED VIEW mv_mahasiswa AS
select  m.mahasiswa_id, m.nim, m.nama, p.prodi_nama, j.jurusan_nama
from    m_mahasiswa m
join    m_prodi p on p.prodi_id = m.prodi_id
join    m_jurusan j on j.jurusan_id = p.jurusan_id;

-- Membaca materialized view
SELECT * FROM mv_mahasiswa;

-- Refresh isi materialized view jika data sumber berubah
REFRESH MATERIALIZED VIEW mv_mahasiswa;

-- Refresh sambil tetap dapat diakses (butuh unique index)
REFRESH MATERIALIZED VIEW CONCURRENTLY mv_mahasiswa;
```

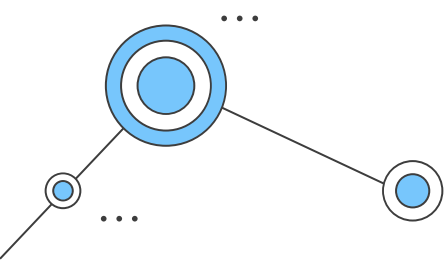
# Materialized View – Kelebihan dan Kekurangan

| Kelebihan | Kekurangan |
| --- | --- |
| ❑ **Performa cepat** untuk query kompleks karena hasil sudah disimpan.<br>❑ Cocok untuk **laporan (reporting), analitik, dan data warehouse**.<br>❑ Bisa di-*index* untuk meningkatkan kecepatan akses. | ❑ Data bisa **usang (stale)** jika tabel sumber berubah tetapi MV belum di-*refresh*.<br>❑ Membutuhkan **penyimpanan disk tambahan** karena hasil query disimpan fisik.<br>❑ Proses REFRESH bisa memakan waktu lama jika query besar. |

***Materialized View*** di PostgreSQL adalah snapshot data dari query yang disimpan di disk. Cocok untuk analitik dan query berat, tetapi perlu ***di-refresh*** agar tetap konsisten dengan data sumber.



# Perbandingan antara
# View dan Materialized View



# Perbandingan

Coba kita bandingkan untuk cek performa melalui **EXPLAIN ANALYZE**

## View

```
CREATE OR REPLACE VIEW vw_mahasiswa AS
select  m.mahasiswa_id, m.nim, m.nama, p.prodi_nama, j.jurusan_nama
from    m_mahasiswa m
join    m_prodi p on p.prodi_id = m.prodi_id
join    m_jurusan j on j.jurusan_id = p.jurusan_id;
```

```
94  explain analyze select * from vw_mahasiswa;
```

Results 1

explain analyze select * from vw_mahasiswa | Enter a SQL expression to filter results (use Ctrl+Space)

| | QUERY PLAN |
|---|---|
| 1 | Hash Join (cost=21.10..134.15 rows=3470 width=330) (actual time=0.047..1.239 rows=3470 loops=1) |
| 2 | Hash Cond: (p.jurusan_id = j.jurusan_id) |
| 3 | -> Hash Join (cost=1.43..105.28 rows=3470 width=220) (actual time=0.037..0.779 rows=3470 loops=1) |
| 4 | Hash Cond: (m.prodi_id = p.prodi_id) |
| 5 | -> Seq Scan on m_mahasiswa m (cost=0.00..93.70 rows=3470 width=42) (actual time=0.014..0.179 rows=3470 loops=1) |
| 6 | -> Hash (cost=1.19..1.19 rows=19 width=190) (actual time=0.017..0.017 rows=37 loops=1) |
| 7 | Buckets: 1024 Batches: 1 Memory Usage: 11kB |
| 8 | -> Seq Scan on m_prodi p (cost=0.00..1.19 rows=19 width=190) (actual time=0.008..0.011 rows=37 loops=1) |
| 9 | -> Hash (cost=14.30..14.30 rows=430 width=122) (actual time=0.007..0.007 rows=7 loops=1) |
| 10 | Buckets: 1024 Batches: 1 Memory Usage: 9kB |
| 11 | -> Seq Scan on m_jurusan j (cost=0.00..14.30 rows=430 width=122) (actual time=0.005..0.005 rows=7 loops=1) |
| 12 | Planning Time: 0.182 ms |
| 13 | Execution Time: 1.393 ms |

## Materialized View

```
CREATE MATERIALIZED VIEW mv_mahasiswa AS
select  m.mahasiswa_id, m.nim, m.nama, p.prodi_nama, j.jurusan_nama
from    m_mahasiswa m
join    m_prodi p on p.prodi_id = m.prodi_id
join    m_jurusan j on j.jurusan_id = p.jurusan_id;
```

```
94  explain analyze select * from mv_mahasiswa;
```

Results 1

explain analyze select * from mv_mahasiswa | Enter a SQL expression to filter results (use Ctrl+Space)

| | QUERY PLAN |
|---|---|
| 1 | Seq Scan on mv_mahasiswa (cost=0.00..85.70 rows=3470 width=85) (actual time=0.015..0.188 rows=3470 loops=1) |
| 2 | Planning Time: 0.046 ms |
| 3 | Execution Time: 0.276 ms |

Dari hasil EXPLAIN ANALYZE didapatkan bahwa

- ***view vw_mahasiswa*** akan menjalankan query asli (ada operasi join) dan membutuhkan waktu lebih.
- ***Materialized View mv_mahasiswa*** langsung mengambil data dari fisik table, sehingga Langkah querynya sedikit dan cepat.

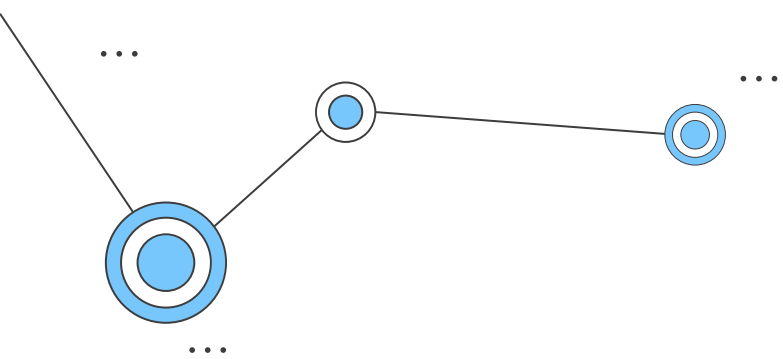
# Perbedaan : View dan Materialized View

| Aspek | *View* | *Materialized View* |
|---|---|---|
| Penyimpanan | Tidak menyimpan data, hanya definisi query. | Data hasil query disimpan secara fisik (seperti tabel). |
| Performa | Lambat untuk query kompleks karena selalu di-query/dihitung ulang. | Cepat karena hasil sudah tersedia. Seperti melakukan query pada 1 tabel |
| Update data | Selalu terbaru | Perlu `REFRESH` secara manual |
| Penggunaan Utama | Menyederhanakan query kompleks, keamanan, abstraksi data. | Analitik, laporan, agregasi data besar, data warehouse |
| Indexing | Tidak bisa langsung di-*index*. Mengikuti struktur index table asli | Bisa di-*index* seperti tabel biasa. |
| Konsumsi Ruang | Tidak butuh penyimpanan tambahan. | Membutuhkan ruang penyimpanan tambahan, karena menyimpan salinan data di disk. |

# Kesimpulan

Gunakan ***View*** biasa bila ingin data selalu *up-to-date, query* yang tidak terlalu kompleks, dan hanya butuh abstraksi query.

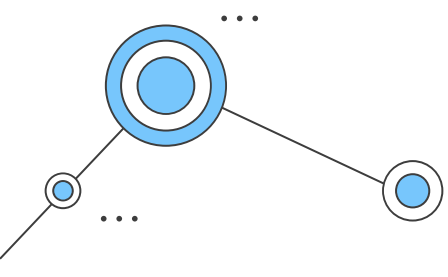
Gunakan ***Materialized View*** bila ingin akses cepat ke *query* kompleks, harus perlu *refresh data* secara berkala.



# Stored Procedure

pada

# PostgreSQL



# Stored Procedure

- ***Stored Procedure*** adalah blok program yang disimpan di dalam *server* database PostgreSQL, berisi satu atau lebih perintah SQL maupun logika pemrograman (PL/pgSQL atau bahasa lain).
- ***Stored Procedure*** diperkenalkan mulai PostgreSQL versi 11.

Bedanya dengan ***function***:

- ***Function*** harus mengembalikan nilai (*RETURN*).
- ***Procedure*** tidak wajib mengembalikan nilai, tetapi bisa menjalankan operasi kontrol transaksi (COMMIT, ROLLBACK).

# Stored Procedure – Tujuan dan Kegunaan

- Membungkus logika bisnis → kode SQL kompleks disimpan di server, tidak perlu ditulis ulang di aplikasi.
- Mengurangi komunikasi client-server → cukup panggil procedure sekali, bukan kirim query berulang.
- Mendukung transaksi → bisa mengatur BEGIN, COMMIT, dan ROLLBACK langsung dalam procedure.
- Keamanan → akses ke tabel bisa dibatasi lewat procedure.
- Reusability → dapat digunakan oleh banyak aplikasi berbeda.

# Stored Procedure – sintaks

**Sintak Dasar:**

```
CREATE OR REPLACE PROCEDURE nama_procedure (nama_parameter tipe, ...)
LANGUAGE plpgsql
AS $$
BEGIN
    -- blok perintah SQL
END;
$$;
```

**Contoh:**

```
CREATE OR REPLACE PROCEDURE tambah_agama(in_agama varchar)
LANGUAGE plpgsql
AS $$
BEGIN
    -- operasi kompleks (jika ada)
    IF not exists(select * from r_agama where lower(agama_nama) = trim(lower(in_agama))) then

        -- Simpan ke tabel agama
        INSERT INTO r_agama (agama_nama)
        VALUES (in_agama);

        RAISE NOTICE 'Data agama berhasil ditambahkan';
    else
        RAISE NOTICE 'Data agama sudah ada. Data tidak disimpan';
    end if;

END;
$$;

-- cara memanggil di query
call tambah_agama('islam');
```

# Perbedaan : Function dan Stored Procedure

| Aspek | Function | Procedure |
|---|---|---|
| Return Value | Wajib mengembalikan nilai (`RETURN`). | Tidak wajib, bisa tanpa `RETURN`. |
| Transaksi | Tidak bisa mengendalikan `COMMIT` / `ROLLBACK`. | Bisa mengatur `COMMIT`, `ROLLBACK`, `SAVEPOINT`. |
| Pemanggilan | `SELECT function_name(...)` | `CALL procedure_name(...)` |
| Tujuan | Perhitungan, manipulasi data, query. | Query kompleks lebih untuk mengeksekusi aksi (INSERT, UPDATE, DELETE, transaksi), eksekusi logika bisnis, batch, kontrol transaksi. |

# Kesimpulan

- ***Function***
  - ✓ selalu ada return value (skalar, record, tabel, atau void).
  - ✓ cocok untuk perhitungan kecil atau query yang butuh nilai balik.
- ***Stored Procedure***
  - ✓ tidak wajib mengembalikan nilai. Jika perlu hasil, gunakan OUT parameter atau simpan hasil ke tabel.
  - ✓ cocok untuk aksi besar, *batch processing*, atau penyimpanan transaksi.

# Thanks!

Do you have any questions?


Team Teaching Matakuliah Basis Data Lanjut
JTI POLINEMA