*JOBSHEET*

# BASIS DATA LANJUT


**Jurusan Teknologi Informasi**

**POLITEKNIK NEGERI MALANG**

**TAHUN AJARAN 2025/2026**

## PERTEMUAN 6

***Function, View, Materialized View, Store Procedure***

Team Teaching Basis Data Lanjut:

- Candra Bella Vista, S.Kom., MT.
- Moch Zawaruddin Abdullah, S.ST., M.Kom.
- Yan Watequlis Syaifudin, ST., MMT., PhD.
- Yoppy Yunhasnawa, S.ST., M.Sc.

KEMENTERIAN PENDIDIKAN, KEBUDAYAAN, RISET, DAN TEKNOLOGI
**POLITEKNIK NEGERI MALANG**
**JURUSAN TEKNOLOGI INFORMASI**
Jl. Soekarno Hatta No. 9, Jatimulyo, Lowokwaru, Malang 65141
Telp. (0341) 404424 – 404425, Fax (0341) 404420
http://www.polinema.ac.id

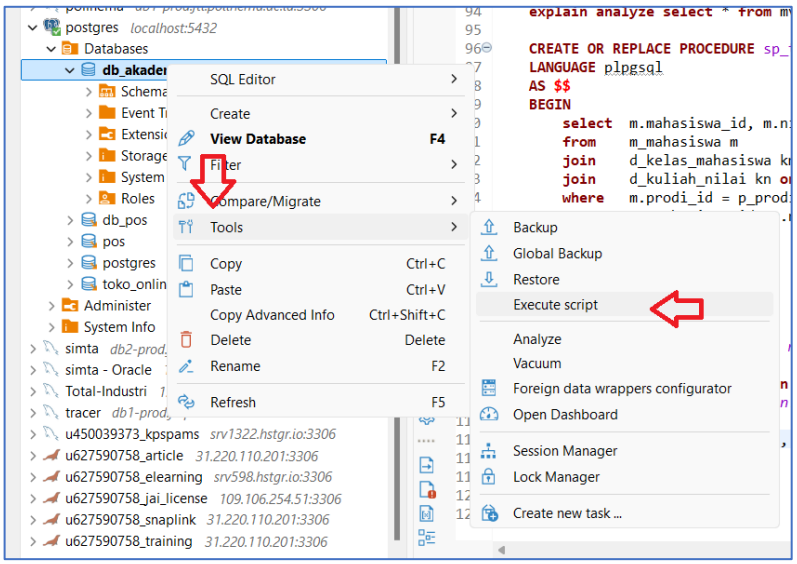
Mata Kuliah : Basis Data Lanjut
Program Studi : D4 – Teknik Informatika / D4 – Sistem Informasi Bisnis
Semester : 3 (tiga)
Pertemuan ke- : 6

# JOBSHEET 06

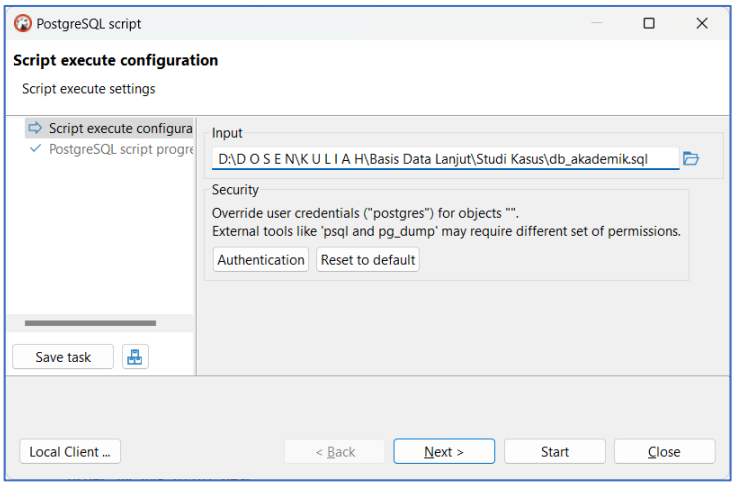
# Function, View, Materialized View, Store Procedure

## Praktikum 00 – Menyiapkan file Studi Kasus (db_akademik)

1. Download file `db_akademik.zip` yang sudah disediakan lalu ekstrak menjadi `db_akademik.sql`
2. `db_akademik.sql` berisi **data dummy** pada contoh desain basis data akademik di Polinema, jadi data yang ada **bukan data asli** (kecuali untuk data jurusan dan prodi).
3. Buka aplikasi dBeaver, lalu buat database baru bernama `db_akademik`
4. Kemudian klik kanan `Tools → Execute Script`

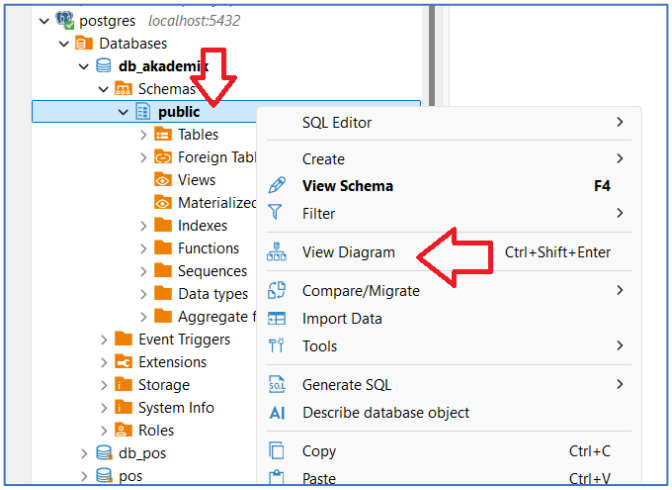



5. Kemudian arahkan ke file db_akademik.sql yang sudah ada




6. Selanjutnya klik `Next → Start`

7. Jika sudah selesai, cek digram ERD melalui klik kanan `Shemas/public → View Diagram`




8. **Pertanyaan 1**
   a. Screenshot hasil diagram yang sudah ada, dan berikan hasil analisis kalian mengenai desain basis data pada db_akademik
   b. jelaskan pula relasi yang ada

## Praktikum 01 – Penggunaan 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

- SQL Function
  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.

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

- PL/pgSQL Function
  Menggunakan bahasa procedural PostgreSQL (plpgsql) atau ***Procedural Language for PostgreSQL SQL***. Function ini mendukung variabel, kontrol alur (IF, LOOP), dan error handling.

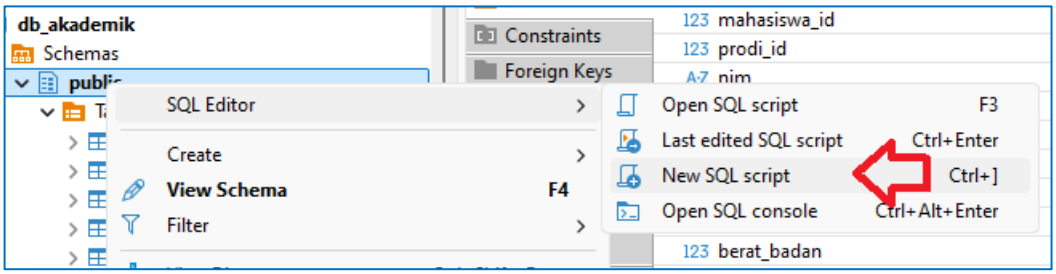
```
CREATE OR REPLACE FUNCTION nama_function(parameter tipe)
RETURNS tipe AS $$
DECLARE
    -- deklarasi variabel
BEGIN
    -- logika / query
    RETURN ...;
END;
$$ LANGUAGE plpgsql;
```

- Language Function Lain
  PostgreSQL mendukung function dalam bahasa eksternal seperti Python (plpython), C, JavaScript, dsb.

Pada praktikum ini, kita akan praktik menggunakan SQL function dan PL/pgSQL function

1. Kita coba lihat tabel `m_mahasiswa`, disana terdapat kolom untuk menyimpan data **tinggi badan** dan **berat badan**.
2. Sebagai contoh, kita bisa manfaatkan function untuk menghitung cek standar tinggi badan dan hitung BMI (*body mass index*) dari data tersebut

3. Kita buka SQL Editor dengan cara klik kanan `Schemas/public → SQL Editor → New SQL Script`




4. Kemudian kita buat **SQL function** untuk cek tinggi badan

```
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;
```

5. Eksekusi query tersebut, dan selanjutnya coba lakukan query untuk melihat hasilnya pada tabel `m_mahasiswa`

```
21    -- Contoh pemanggilan saat query
22    select  nim, nama, tinggi_badan, cek_tinggi_badan(tinggi_badan) as kategori_tinggi
23    from    m_mahasiswa;
```

m_mahasiswa 1

select nim, nama, tinggi_badan, cek_tinggi_badan(t | Enter a SQL expression to filter results (use Ctrl+Space)

| | nim | nama | tinggi_badan | kategori_tinggi |
|---|---|---|---|---|
| 1 | 2101100001 | Dimas Dirja Samosir | 146 | Pendek |
| 2 | 2101220002 | Sabri Jamal Mahendra M.Kom. | 145 | Pendek |
| 3 | 2101780003 | Maida Usada | 153 | Pendek |
| 4 | 2101600004 | Febi Hassanah S.Ked | 145 | Pendek |
| 5 | 2101280005 | Zulfa Kuswandari | 169 | Sedang |
| 6 | 2101910006 | Dimas Setiawan S.Pd | 153 | Pendek |
| 7 | 2101930007 | Chelsea Vanesa Prastuti | 163 | Sedang |

6. Selanjutnya kita gunakan **PL/pgSQL function** untuk menghitung BMI dari data tinggi badan dan berat badan yang ada di tabel `m_mahasiswa`.

```
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;
```

7. Setelah membuat function hitung_bmi, silahkan buat query dengan menggunakan function tersebut.
8. **Pertanyaan 2**
    a. Lakukan query untuk mencoba 2 fungsi tersebut, jelaskan hasil yang didapat
    b. Berikan pendapatmu mengenai perbedaan function `cek_tinggi_badan` dan `hitung_bmi`
    c. Berikan pendapatmu mengenai SQL function dan PL/pgSQL function

## Praktikum 02 – Penggunaan View

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.

### Tujuan dan Kegunaan View

- Menyederhanakan *Query* → Query yang kompleks bisa dibungkus menjadi sebuah View. Kita cukup memanggil View tanpa harus menulis ulang query panjang.
- Meningkatkan Keamanan → View bisa digunakan untuk membatasi akses. Jadi kita bisa memberikan hak akses user tertentu untuk mengakses view (view hanya bersifat read-only, jadi kita lebih aman memberikan akses ke user tertentu). 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. Jadi pengguna tidak tahu jika terjadi perubahan pada tabel.

### Sintaks dasar View

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

Langkah praktikum View

1. Kita lihat tabel `m_mahasiswa`, dari tabel ini kita memiliki kolom *prodi_id*, *agama_id*, dan *kota_id*. Bagaimana kita bisa melihat nama prodi, nama agama, dan nama kota untuk data mahasiswa? Jawabannya adalah dengan menggunakan join.
2. Kita buat view `vw_mahasiswa` dulu untuk menampilkan data mahasiswa

```
CREATE OR REPLACE VIEW vw_mahasiswa AS
select  m.mahasiswa_id,
        m.nim,
        m.nama,
        p.prodi_nama as prodi,
        j.jurusan_nama as jurusan,
        a.agama_nama as agama,
        w.wilayah_nama as kota_lahir
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
join    r_agama a on a.agama_id = m.agama_id
join    r_wilayah w on w.wilayah_id = m.kota_id;
```

3. Selanjutnya kita bisa memanggil ***view*** tersebut layaknya ***tabel***, akan tetapi dengan menuliskan query yang menjadi sederhana

```
select * from vw_mahasiswa;
```

4. Cek dan lihat hasilnya
5. **Pertanyaan 3**
   a. Apa yang kamu ketahui mengenai view di atas?
   b. Ubahlah query dalam view `vw_mahasiswa`, agar view `vw_mahasiswa` bisa dilakukan filter bersadarkan *agama_id*, *prodi_id* dan *jurusan_id.*
   Contoh

```
select  *
from    vw_mahasiswa
where   prodi_id = 1;
```

## Praktikum 03 – Penggunaan *Materialized View*

*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* biasa (regular view) yang hanya menyimpan query dan dieksekusi ulang setiap kali dipanggil, *Materialized View* **menyimpan data snapshot dari hasil query**.

### 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>;
```

### Sintaks dasar Materialized View

```
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>;
```

### Kelebihan *Materialized View*

- Performa cepat untuk query kompleks karena hasil sudah disimpan.
- Cocok untuk laporan (*reporting*), analitik, dan *data warehouse*.
- Bisa di-*index* untuk meningkatkan kecepatan akses.

### Kekurangan *Materialized View*

- Data bisa usang (*stale*) jika tabel sumber berubah tetapi MV belum di-*refresh*.
- Membutuhkan penyimpanan disk tambahan karena hasil query disimpan fisik.
- Proses REFRESH bisa memakan waktu lama jika query besar.

Langkah praktikum Materialized View

1. Sama halnya *view* (*view* biasa) `vw_mahasiswa`, Kita bisa menggunakan *Materialized View* untuk menampilkan data mahasiswa dengan detail prodi, jurusan, agama, dan kota.
2. Kita buat *materialized view* dengan nama `mv_mahasiswa` seperti berikut.

```
CREATE MATERIALIZED VIEW mv_mahasiswa AS
select  m.mahasiswa_id,
        m.nim,
        m.nama,
        p.prodi_nama as prodi,
        j.jurusan_nama as jurusan,
        a.agama_nama as agama,
        w.wilayah_nama as kota_lahir
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
join    r_agama a on a.agama_id = m.agama_id
join    r_wilayah w on w.wilayah_id = m.kota_id;
```

3. Kita eksekusi query tersebut, kemudian kita bisa memanggil materialized view tersebut seperti berikut

```
-- Membaca materialized view
SELECT * FROM mv_mahasiswa;
```

4. Cek dan lihat hasilnya
5. Silahkan tambahkan beberapa data mahasiswa
6. Kemudian query `mv_mahasiswa` seperti tahap 3 di atas. Cek apa yang terjadi
7. Jika data tidak muncul, maka kita perlu melakukan ***refresh data*** untuk `mv_mahasiswa`

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

8. Lakukan kembali query `mv_mahasiswa`, dan lihat hasilnya
9. Pertanyaan 3
   a. Apa yang kamu ketahui mengenai view di atas?
   b. Ubahlah query dalam view vw_mahasiswa, agar view vw_mahasiswa bisa dilakukan filter bersadarkan agama_id, prodi_id dan jurusan_id.
   Contoh

```
SELECT  *
FROM    mv_mahasiswa
WHERE   jurusan_id = 1;
```

   c. Gunakan EXPLAIN ANALYZE untuk membandingkan `vw_mahasiswa` dengan `mv_mahasiswa`. Analisa dan jelaskan pendapatmu

## Praktikum 04 – Penggunaan *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 *Stored Procedure* dengan function:

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

### Kegunaan Stored Procedure

- 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.

### Sintaks dasar *Stored Procedure*

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

Langkah praktikum *Stored Procedure*

1. Stored Procedure bisa digunakan untuk melakukan banyak hal, bisa digunakan untuk mengambil data, melakukan proses CRUD, menggunakan transaksi, dsb.
2. Dipraktikum ini, kita akan membuat SP untuk melakukan validasi input data agama.
3. Kita buat *stored procedure* dengan nama `tambah_agama` seperti berikut.

```
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;
$$;
```

4. Untuk mengakses SP tersebut kita jalankan perintah query berikut

```
call tambah_agama('islam');
```

5. Cek dan lihat hasilnya
6. **Pertanyaan 4**
    a. Apa yang kamu ketahui mengenai SP di atas?
    b. Apa yang dilakukan oleh SP diatas pada data baru?

## Soal Latihan Jobsheet 06

Silahkan kerjakan soal-soal berikut untuk mengasah kemampuan anda

1. Buatlah **satu *function*** baru (bisa *SQl function* atau *PL/pgSQL function*) untuk menghitung Proporsi Ideal Berat Badan (PIBD).
    a. Rumus PIBD untuk Pria
    ```
    Berat Ideal = (Tinggi Badan - 100) - ((Tinggi Badan - 100) * 0.1)
    ```
    b. Rumus PIBD untuk Wanita
    ```
    Berat Ideal = (Tinggi Badan - 100) - ((Tinggi Badan - 100) * 0.15)
    ```

    Mahasiswa dikatakan memiliki berat ideal jika
    ```
    absolute(Berat Ideal - Berat Badan) <= 2
    ```
2. Buatlah 2 *view* biasa untuk menampilkan
    *a.* List mahasiswa, kelas, dan dosen wali (DPA)
    *b.* List kurikulum prodi berserta mata kuliahnya
3. Buatlah 2 *materialized view* untuk menampilkan
    *a.* Data dosen berserta mata kuliah yang diajar
    *b.* Data mahasiswa berserta nilai mata kuliah yang didapat

4. Buatlah 1 stored procedure untuk
    *a.* Melakukan validasi insert data mahasiswa, **hanya bisa input** mahasiswa dengan tahun angkatan 2021, panjang nim adalah 10 digit, berat badan antara 48-95, dan tinggi badan antara 150 – 180cm.

**** Sekian, dan selamat belajar ****