Tutorial 3 PostgreSQL
Basis Data
CSGE602070
Semester Gasal 2022/2023

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

Note!

Baca dengan teliti agar tidak ada yang terlewat. Pastikan setiap langkah yang Anda lakukan di masukkan ke dalam laporan. **Kerjakan semua latihan Soal** dengan menampilkan SQL yang Anda buat beserta capture output dari SQL yang anda buat.

Riwayat Versi

Versi	*Timestamp*	Halaman	Perubahan
1	07-11-2022	Semua	Rilis Pertama

Note : Setiap revisi ditandai dengan highlights warna kuning Revisi

Tutorial ini menggunakan skema basis data **SIWANAP** hasil kelanjutan dari tutorial sebelumnya. Jangan lupa **SET SEARCH PATH** terlebih dahulu.

I. Stored Procedure dan Function

Apa itu stored procedure dan function?

Stored procedure dan *function* (atau *user-defined* function) merupakan **bagian dari bahasa prosedural** di dalam SQL. Di dalam PostgreSQL, bahasa ini disebut **PL/pgSQL**. Dengan keduanya, kita bisa menambahkan berbagai **elemen prosedural**, seperti *loop*, perhitungan kompleks, dan masih banyak lagi. Kita juga bisa mengembangkan fungsi yang kompleks yang **tidak bisa dicapai** dengan statement SQL biasa.

Pada awalnya PostgreSQL hanya memiliki objek ***function*** kemudian pada PostgreSQL versi 11 ke atas baru diperkenalkan yang disebut ***stored procedure***. Maka perhatikan versi PostgreSQL yang anda gunakan. Terdapat banyak kemiripan antara function dan stored procedure, perbedaannya yaitu ***function*** dapat me-*return value*, sedangkan ***stored procedure*** tidak mengembalikan *value*.

Pada tutorial ini kita akan membahas penggunaan *function*.

Bagaimana cara membuat function?

Function dalam PostgreSQL dapat dibuat dengan sintaks berikut.

```
CREATE [OR REPLACE] FUNCTION function_name (arguments)
RETURNS return_datatype AS
$$
      DECLARE
            declaration;
            [...]
      BEGIN
            < function_body >
            [...]
            RETURN { variable_name | value }
      END;
$$
LANGUAGE plpgsql;
```

Sekarang kita akan mencoba membuat fungsi yang memotong harga pada **KAMAR** menjadi 90%-nya (diskon 10%).

Contoh 1:

Buatlah sebuah *function* yang akan menghitung harga setelah diberi diskon 10%. Contoh *function*-nya adalah sebagai berikut.

```
CREATE OR REPLACE FUNCTION SIWANAP.diskon_harga(idkamar
VARCHAR(10))
RETURNS INTEGER AS
$$
    DECLARE
      harga_awal INTEGER;
      harga_diskon INTEGER;
    BEGIN
      SELECT harga INTO harga_awal
      FROM KAMAR
      WHERE id_kamar = idkamar;

      harga_diskon := (harga_awal*9/10);
```

```
        UPDATE KAMAR SET harga = harga_diskon
        WHERE id_kamar = idkamar;

        RETURN harga_diskon;
    END;
$$
LANGUAGE plpgsql;
```

Ket: perintah INTO akan menyimpan hasil select ke variabel harga_awal.

Untuk melihat fungsi yang telah dibuat dapat menggunakan perintah sebagai berikut.

```
\df
```

Untuk melihat kode fungsi yang telah dibuat, dapat menggunakan perintah sebagai berikut.

```
\df+
```

Kita dapat menjalankan *function* yang sudah dibuat dengan sintaks berikut.

```
SELECT SCHEMA_NAME.function_name([<arg1>, <arg2>, …});
```

Note: jika sudah set search_path, tidak perlu menggunakan SCHEMA_NAME.

Contoh 2:

Sebagai contoh, terapkan function yang sudah dibuat tadi pada harga kamar dengan id kamar KA01.

```
SELECT diskon_harga('KA01');
```

```
rakha.rayhan=> select harga from kamar where id_kamar = 'KA01';
 harga
--------
 170000
(1 row)

rakha.rayhan=> SELECT diskon_harga('KA01');
 diskon_harga
--------------
       153000
(1 row)

rakha.rayhan=> select harga from kamar where id_kamar = 'KA01';
 harga
--------
 153000
(1 row)
```

Terlihat bahwa **harga kamar sekarang menjadi 153000**.

Contoh 3:

Untuk menerapkan function ini ke semua baris pada tabel **KAMAR**, kita bisa melakukan query berikut.

```
SELECT diskon_harga(id_kamar)
FROM KAMAR;
```

Dengan demikian, **diskon_harga() akan diterapkan ke semua baris pada KAMAR**. Selain update, function diskon_harga() juga mengembalikan nilai sehingga kita akan dapat melihat hasil query berupa tabel dengan kolom diskon_harga.

Contoh 4:

Selain dengan cara tersebut, kita juga bisa melakukan looping di function untuk meng-update setiap baris di tabel KAMAR. Berikut adalah contoh function-nya.

```
CREATE OR REPLACE FUNCTION diskon_semua_harga()
RETURNS void AS
$$
    DECLARE
      temp_row RECORD;
      harga_diskon INTEGER;
    BEGIN
      FOR temp_row IN
```

```
            SELECT *
            FROM KAMAR
        LOOP
            harga_diskon := (temp_row.harga*9/10);

            UPDATE KAMAR SET harga = harga_diskon
            WHERE id_kamar = temp_row.id_kamar;
        END LOOP;
    END;
$$
LANGUAGE plpgsql;
```

Kemudian, jalankan function tersebut seperti berikut.

```
SELECT diskon_semua_harga();
```

Contoh 5:

Untuk menghapus function yang sudah dibuat, kita dapat menjalankan perintah berikut.

```
DROP FUNCTION function_name(arg_type);
```

Atau sebagai contoh untuk menghapus function diskon_harga(idkamar VARCHAR(10)), kita dapat menjalankan perintah berikut.

```
DROP FUNCTION diskon_harga(idkamar VARCHAR(10));
```

II. PostgreSQL Triggers

Apa itu trigger?

Trigger merupakan operasi pada sebuah tabel yang **otomatis dijalankan** ketika ada kejadian tertentu. Kejadian ini bisa berupa ketika melakukan *INSERT, UPDATE,* atau *DELETE*. Agar trigger bisa bekerja, kita perlu **membuat** *stored procedure* terlebih dahulu, baru kemudian menyambungkan suatu *trigger* dengan *stored procedure* tersebut ke suatu tabel.

Format sintaks PL/SQL yang digunakan untuk membuat stored procedure yang akan dipanggil *trigger* ini **sama** seperti *function* biasa. Akan tetapi, harus **bernilai trigger** dan **tidak boleh mempunyai argumen**.

Contoh 6

Mari kita coba tangani kasus untuk mencegah perawat memiliki shift lebih dari 5. Kita akan memulai dengan membuat *function trigger*-nya terlebih dahulu.

```
CREATE OR REPLACE FUNCTION cek_jumlah_shift()
RETURNS trigger AS
$$
    DECLARE
        shift_count integer;
    BEGIN
        IF(TG_OP = 'INSERT') THEN
            SELECT COUNT(*) into shift_count
            FROM SHIFT_PERAWAT
            WHERE id_perawat = NEW.id_perawat
            GROUP BY id_perawat;
            IF(shift_count >= 5) THEN
                RAISE EXCEPTION 'Maaf, perawat
                            tidak boleh memiliki shift
melebihi                    5';
            END IF;
            RETURN NEW;
        END IF;
      END;
$$
LANGUAGE plpgsql;
```

Setelah itu buat *trigger*-nya. Berikut ini sintaks untuk membuat *trigger* di dalam PostgreSQL.

```
CREATE TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF} {event [OR ...]}
 ON table_name
 [FOR [EACH] {ROW | STATEMENT}]
 EXECUTE PROCEDURE trigger_function
```

Contoh 7

Berdasarkan sintaks tersebut, kita dapat membuat *trigger* yang akan menjalankan *function* cek_jumlah_shift() **sebelum** *row* baru dimasukkan / *insert*.

```
CREATE TRIGGER trigger_cek_jumlah_shift
BEFORE INSERT ON SHIFT_PERAWAT
FOR EACH ROW
EXECUTE PROCEDURE cek_jumlah_shift();
```

Untuk memeriksa apakah *trigger* tersebut bekerja atau tidak, kita bisa menjalankan *event* yang memicu *trigger* tersebut.

Contoh 8

Sebagai contoh, coba masukkan suatu shift_perawat dimana perawat dengan id tersebut sudah memiliki shift berjumlah 5.

```
INSERT INTO SHIFT_PERAWAT (id_shift_perawat, id_perawat,
id_rawat_inap, waktu_mulai, waktu_akhir)
VALUES ('SP101', 'PE13', 'RI20', '2020-11-30 00:00',
'2020-11-30 12:00');
```

Amati apakah *trigger* tersebut berhasil. Bandingkan bila tabel SHIFT_PERAWAT dimasukkan data yang tidak menimbulkan *error* seperti berikut:

```
INSERT INTO SHIFT_PERAWAT (id_shift_perawat, id_perawat,
id_rawat_inap, waktu_mulai, waktu_akhir)
VALUES ('SP101', 'PE11', 'RI20', '2020-11-30 00:00',
'2020-11-30 12:00');
```

III. Latihan

Pada latihan ini, soal dengan tanda **[SQL]** harus dijalankan pada database masing-masing dan sertakan *screenshoot*-nya sesuai ketentuan pada format laporan tutorial lab.

1. **[SQL]** Jalankan seluruh contoh 1 hingga contoh 8 di atas!
2. **[SQL]** Buatlah function/stored procedure dengan nama **check_validity** dan trigger dengan nama **trigger_check_validity** untuk setiap INSERT pada tabel **RAWAT_INAP**

untuk memastikan bahwa **tgl_masuk** terjadi sebelum **tgl_keluar** (tgl_masuk dan tgl_keluar pada hari yang sama juga tidak boleh). Berikan exception message seperti berikut 'Input tidak valid pastikan bahwa tanggal masuk sebelum tanggal keluar' atau disesuaikan dengan kreativitas kalian tetapi masih dalam pengertian yang sesuai.

Kemudian jalankan ketiga perintah berikut.

```
INSERT INTO RAWAT_INAP VALUES ('RI51', 'KA01', 'PA03',
'2022-11-06', '2022-11-08');
```

```
INSERT INTO RAWAT_INAP VALUES ('RI52', 'KA05', 'PA18',
'2022-11-10', '2022-11-08');
```

```
INSERT INTO RAWAT_INAP VALUES ('RI53', 'KA01', 'PA38',
'2022-11-11', '2022-11-11');
```

Untuk tiap operasi (create function, create trigger, & insert), *screenshot query* yang ditulis serta hasil dari query tersebut. *Screenshot* pula \df dan \d RAWAT_INAP setelah semua operasi dijalankan.

3. **[SQL]** Terlebih dahulu lakukan penambahan kolom pada tabel **RAWAT_INAP** dengan nama **jml_biaya** dengan tipe integer dan nilai default = 0.

   Buatlah function/stored procedure dengan nama **calculate_cost** dan trigger dengan nama **trigger_calculate_cost** untuk setiap INSERT dan UPDATE pada tabel **RAWAT_INAP**. Function bertujuan untuk menghitung **jml_biaya** yang perlu dibayarkan oleh pasien untuk rawat inap. Perhitungan matematisnya: Jumlah malam dirawat * harga kamar (kolom **harga** pada tabel **KAMAR**). Contoh apabila seorang pasien menginap pada kamar KA02 dengan harga 137700 dan dia dirawat dari 2022-11-06 dan keluar pada 2022-11-08 (2 malam) maka jumlah biayanya adalah 2 * 137700 = 275400. Perlu diperhatikan bahwa kolom **tgl_keluar** pada tabel **RAWAT_INAP** bisa kosong (null), untuk kasus ini maka **calculate_cost** tidak akan dijalankan (Hint: gunakan if pada function untuk menghandle ini).

   Kemudian jalankan kedua perintah berikut.

```
INSERT INTO RAWAT_INAP VALUES ('RI52', 'KA05', 'PA18',
'2022-11-10', '2022-11-12');
```

```
SELECT * FROM RAWAT_INAP WHERE id_rawat_inap='RI52';
```

Untuk tiap operasi (alter table, create function, create trigger, insert, dan select), *screenshot query* yang ditulis serta hasil dari query tersebut. *Screenshot* pula \df dan \d RAWAT_INAP setelah semua operasi dijalankan.