Database

Postgresql

Overview

- SQL
- DDL
- DML
- Data Types
- Database
- Table
- CRUD (Select, Insert, Update, Delete)
- Generate ERD

Overview

- Aggregate Function (Built in Function)
- Join
- Union
- Create View
- Create Function
- Create Procedure
- Transaction

SQL

RDBMS (Relational Database Management System) seperti postgresql menggunakan Structured Query Language (SQL)

SQL dibagi menjadi DQL, DDL,DML, dan DCL

DQL singkatan dari Data Query Language. Contoh:Select

DDL singkatan dari Data Definition Language. Contoh:Create, Alter, dll

DML singkatan dari Data Manipulation Language. Contoh:Insert, Update, Delete

DCL singkatan dari Data Control Language. Contoh: Grant, Revoke, dll

DDL

DDL digunakan untuk membuat struktur atau schema sebuah database dan object datatabse tersebut. DDL tidak berkaitan dengan data yang disimpan pada database.

Contoh:

Create database hrd => membuat database bernama hrd

Create table employee => membuat table bernama employee

Drop database hrd => menghapus database hrd

Drop table employee => menghapus table employee

Alter table employee Add column address varchar(200) => mengubah table employee dengan menambah column baru

Truncate table employee => mengosongkan table employee

DML

DML digunakan untuk mengelola data yang disimpan pada table di database, seperti menambah data, mengubah data, dan menghapus data.

Contoh:

```
INSERT INTO students (RollNo, FIrstName, LastName) VALUES ('60', 'Tom', 'Erichsen');
```

=> menambah data

```
UPDATE students SET FirstName = 'Jhon', LastName=' Wick'  WHERE StudID = 3;
```

=> mengubah data

```
DELETE FROM students

WHERE FirstName = 'Jhon';
```

=> menghapus data

Data Types

Boolean

Character Types yaitu Char, Varchar, Text

Numeric Types yaitu Integer dan floating-point number (decimal)

Serial

Temporal Types yaitu Date, TimeStamp, Time

UUID

Array yaitu integer[], char[], varchar[], dll

Json

User-Defined data types

Data Types- Boolean

Boolean dapat benilai True, False, atau Null

Ketika insert data pada column type boolean dengan nilai:

- 1, yes, y, t, true akan diconvert ke true
- 0, no, n, f, false akan diconvert ke false

Data Types- Char, Varchar, Text

Char singkatan Character, Varchar singkatan Character varying

Char biasanya digunakan untuk fixed-length, jika kurang dari length akan ditambahkan string kosong. Default length type Char = 1

Varchar biasanya digunakan untuk length bervariasi tetapi memiliki limit. Default length Varchar = unlimited

Text mirip varchar namun unlimited.

Data Types- Numeric

Numeric sama dengan decimal. Numeric biasanya digunakan untuk angka yang memiliki digit dibelakang koma. Sintaks: `NUMERIC(precision, scale)`

```
CREATE TABLE products (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    price NUMERIC(5,2)
);
```

```
INSERT INTO products (name, price)
VALUES ('Phone',500.215),
       ('Tablet',500.214);
```

id	name	price
1	Phone	500.22
2	Tablet	500.21

Data Types- Integer

Type Integer menyimpan data number yang bukan decimal

Name	Storage Size	Min	Max
SMALLINT	2 bytes	-32,768	+32,767
INTEGER	4 bytes	-2,147,483,648	+2,147,483,647
BIGINT	8 bytes	-9,223,372,036,854,775,808	+9,223,372,036,854,775,807

Data Types- Serial

Menghasilkan sequence integer (auto-increment), dan biasanya digunakan untuk column yang akan menjadi primary key

```
CREATE TABLE table_name(
    id SERIAL
);
```

=

```
CREATE SEQUENCE table_name_id_seq;

CREATE TABLE table_name (
    id integer NOT NULL DEFAULT nextval('table_name_id_seq')
);

ALTER SEQUENCE table_name_id_seq
OWNED BY table_name.id;
```

Data Types- Serial

Contoh:

```
CREATE TABLE fruits(
   id SERIAL PRIMARY KEY,
   name VARCHAR NOT NULL
);
```

```
INSERT INTO fruits(name)
VALUES('Orange');
```

```
INSERT INTO fruits(id,name)
VALUES(DEFAULT,'Apple');
```

```
 id |  name
----+--------
  1 | Apple
  2 | Orange
(2 rows)
```

Data Types- Serial

Name	Storage Size	Range
SMALLSERIAL	2 bytes	1 to 32,767
SERIAL	4 bytes	1 to 2,147,483,647
BIGSERIAL	8 bytes	1 to 9,223,372,036,854,775,807

Data Types- Date, Timestamp, Time

Type Date untuk menyimpan tanggal. Format yang digunakan postgresql `yyyy-mm-dd` contoh:2023-03-26

```
CREATE TABLE documents (
        document_id serial PRIMARY KEY,
        header_text VARCHAR (255) NOT NULL,
        posting_date DATE NOT NULL DEFAULT CURRENT_DATE
);

INSERT INTO documents (header_text)
VALUES('Billing to customer XYZ');
```

```
 document_id |       header_text       | posting_date
-------------+-------------------------+--------------
           1 | Billing to customer XYZ | 2016-06-23
(1 row)
```

Data Types- Date, Timestamp, Time

```
CREATE TABLE employees (
        employee_id serial PRIMARY KEY,
        first_name VARCHAR (255),
        last_name VARCHAR (355),
        birth_date DATE NOT NULL,
        hire_date DATE NOT NULL
);

INSERT INTO employees (first_name, last_name, birth_date, hire_date)
VALUES ('Shannon','Freeman','1980-01-01','2005-01-01'),
           ('Sheila','Wells','1978-02-05','2003-01-01'),
           ('Ethel','Webb','1975-01-01','2001-01-01');
```

Data Types- Date, Timestamp, Time

```
SELECT
        employee_id,
        first_name,
        last_name,
        EXTRACT (YEAR FROM birth_date) AS YEAR,
        EXTRACT (MONTH FROM birth_date) AS MONTH,
        EXTRACT (DAY FROM birth_date) AS DAY
FROM
        employees;
```

```
 employee_id | first_name | last_name | year | month | day
-------------+------------+-----------+------+-------+-----
           1 | Shannon    | Freeman   | 1980 |     1 |   1
           2 | Sheila     | Wells     | 1978 |     2 |   5
           3 | Ethel      | Webb      | 1975 |     1 |   1
(3 rows)
```

Data Types- Date, Timestamp, Time

Timestamp menyimpan tanggal dan waktu, sedangkan Timestamptz ada tambahan time zone. Mendapatkan current timestamp: `CURRENT_TIMESTAMP;`

```
CREATE TABLE timestamp_demo (
    ts TIMESTAMP,
    tstz TIMESTAMPTZ
);
```

```
INSERT INTO timestamp_demo (ts, tstz)
VALUES('2016-06-22 19:10:25-07','2016-06-22 19:10:25-07');
```

```
SHOW TIMEZONE;
```

```
      TimeZone
--------------------
 America/Los_Angeles
(1 row)
```

```
SET timezone = 'America/New_York';
```

```
         ts          |          tstz
---------------------+------------------------
 2016-06-22 19:10:25 | 2016-06-22 19:10:25-07
(1 row)
```

```
         ts          |          tstz
---------------------+------------------------
 2016-06-22 19:10:25 | 2016-06-22 22:10:25-04
(1 row)
```

Data Types- Date, Timestamp, Time

Type Time untuk menyimpan data waktu.

```
CREATE TABLE shifts (
    id serial PRIMARY KEY,
    shift_name VARCHAR NOT NULL,
    start_at TIME NOT NULL,
    end_at TIME NOT NULL
);
```

```
INSERT INTO shifts(shift_name, start_at, end_at)
VALUES('Morning', '08:00:00', '12:00:00'),
      ('Afternoon', '13:00:00', '17:00:00'),
      ('Night', '18:00:00', '22:00:00');
```

id	shift_name	start_at	end_at
1	Morning	08:00:00	12:00:00
2	Afternoon	13:00:00	17:00:00
3	Night	18:00:00	22:00:00

Data Types- UUID

contact_id	first_name	last_name	email	phone
8953f6b4-aa3f-4399-b7b1-81c90191aabb	John	Smith	john.smith@example.com	408-237-2345

UUID singkatan dari Universal Unique Identifier. Dihasilkan dari algoritma dan sangat unik, sehingga lebih baik dari Type Serial. `40e6215d-b5c6-4896-987c-f30f3678f608`

Memastikan sudah terinstall library UUID:

```
SELECT uuid_generate_v1();
```

Install dengan cara sebagai berikut:

```
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
```

```
CREATE TABLE contacts (
    contact_id uuid DEFAULT uuid_generate_v4 (),
    first_name VARCHAR NOT NULL,
    last_name VARCHAR NOT NULL,
    email VARCHAR NOT NULL,
    phone VARCHAR,
    PRIMARY KEY (contact_id)
);
```

```
INSERT INTO contacts (
    first_name,
    last_name,
    email,
    phone
)
VALUES
    (
        'John',
        'Smith',
        'john.smith@example.com',
        '408-237-2345'
    ),
```

Data Types- Array

Type array untuk menyimpan data dalam bentuk array. Terdapat integer[], Char[], Varchar[], text[]

```
CREATE TABLE contacts (
        id serial PRIMARY KEY,
        name VARCHAR (100),
        phones TEXT []
);
```

```
INSERT INTO contacts (name, phones)
VALUES('John Doe',ARRAY [ '(408)-589-5846','(408)-589-5555' ]);
```

```
INSERT INTO contacts (name, phones)
VALUES('Lily Bush','{"(408)-589-5841"}'),
      ('William Gate','{"(408)-589-5842","(408)-589-58423"}');
```

name	phones
John Doe	{(408)-589-5846,(408)-589-5555}
Lily Bush	{(408)-589-5841}
William Gate	{(408)-589-5842,(408)-589-58423}

Data Types- Json

Type Json menyimpan data dalam format json yaitu pasangan key dan value.

```
CREATE TABLE orders (
        id serial NOT NULL PRIMARY KEY,
        info json NOT NULL
);
```

```
SELECT info FROM orders;
```

info
{ "customer": "John Doe", "items": {"product": "Beer","qty": 6}}
{ "customer": "Lily Bush", "items": {"product": "Diaper","qty": 24}}
{ "customer": "Josh William", "items": {"product": "Toy Car","qty": 1}}
{ "customer": "Mary Clark", "items": {"product": "Toy Train","qty": 2}}

```
INSERT INTO orders (info)
VALUES('{ "customer": "Lily Bush", "items": {"product": "Diaper","qty": 24}}'),
      ('{ "customer": "Josh William", "items": {"product": "Toy Car","qty": 1}}
      ('{ "customer": "Mary Clark", "items": {"product": "Toy Train","qty": 2}}
```

Data Types- User-Defined

Type data yang dibuat untuk dapat digunakan kembali oleh data yang lain.

```
CREATE TABLE mailing_list (
    id SERIAL PRIMARY KEY,
    first_name VARCHAR NOT NULL,
    last_name VARCHAR NOT NULL,
    email VARCHAR NOT NULL,
    CHECK (
        first_name !~ '\s'
        AND last_name !~ '\s'
    )
);
```

```
CREATE DOMAIN contact_name AS
   VARCHAR NOT NULL CHECK (value !~ '\s');
```

```
CREATE TABLE mailing_list (
    id serial PRIMARY KEY,
    first_name contact_name,
    last_name contact_name,
    email VARCHAR NOT NULL
);
```

```
INSERT INTO mailing_list (first_name, last_name, email)
VALUES('Jame V','Doe','jame.doe@example.com');
```

```
INSERT INTO mailing_list (first_name, last_name, email)
VALUES('Jane','Doe','jane.doe@example.com');
```

Database

Kumpulan dari table dan object lainnya seperti fungsi, procedure,view, trigger, type, dll

Command Prompt - psql -U postgres

```
C:\Users\User>psql
Password for user User:
```

Command Prompt - psql -U postgres

```
C:\Users\User>psql
Password for user User:
psql: error: connection to server at "localhost" (::1), port 5432 failed: fe_sendauth: no password supplied

C:\Users\User>psql -U postgres
Password for user postgres:
```

```
postgres=# CREATE Database Test;
CREATE DATABASE
```

```
postgres=# DROP Database test;
DROP DATABASE
postgres=#
```

Table

Kumpulan dari record atau baris. Record adalah kumpulan dari kolom

```sql
CREATE TABLE links (
   link_id serial PRIMARY KEY,
   title VARCHAR (512) NOT NULL,
   url VARCHAR (1024) NOT NULL
);
```

```sql
ALTER TABLE links
ADD COLUMN active boolean;

ALTER TABLE links
DROP COLUMN active;

ALTER TABLE links
RENAME COLUMN title TO link_title;

ALTER TABLE links
ADD COLUMN target VARCHAR(10);

ALTER TABLE links
ALTER COLUMN target
SET DEFAULT '_blank';

ALTER TABLE links
ALTER COLUMN target Type varchar(50)
```

```sql
ALTER TABLE links
RENAME TO urls;
```

Table

```sql
CREATE TABLE customers (
    id SERIAL PRIMARY KEY,
    customer_name VARCHAR NOT NULL
);


INSERT INTO
   customers (customer_name)
VALUES
   ('Apple'),
   ('Samsung'),
   ('Sony');


 ALTER TABLE customers
 ADD COLUMN contact_name VARCHAR NOT NULL;
```

```
ERROR:  column "contact_name" contains null values
```

```sql
ALTER TABLE customers
ADD COLUMN contact_name VARCHAR;

UPDATE customers
SET contact_name = 'John Doe'
WHERE id = 1;

UPDATE customers
SET contact_name = 'Mary Doe'
WHERE id = 2;

UPDATE customers
SET contact_name = 'Lily Bush'
WHERE id = 3;

ALTER TABLE customers
ALTER COLUMN contact_name SET NOT NULL;
```

Table

```sql
CREATE TABLE publishers (
    publisher_id serial PRIMARY KEY,
    name VARCHAR NOT NULL
);
```

```sql
CREATE TABLE books (
    book_id serial PRIMARY KEY,
    title VARCHAR NOT NULL,
    isbn VARCHAR NOT NULL,
    published_date DATE NOT NULL,
    description VARCHAR,
    category_id INT NOT NULL,
    publisher_id INT NOT NULL,
    FOREIGN KEY (publisher_id)
       REFERENCES publishers (publisher_id),
    FOREIGN KEY (category_id)
       REFERENCES categories (category_id)
);
```

```sql
CREATE TABLE categories (
    category_id serial PRIMARY KEY,
    name VARCHAR NOT NULL
);
```

```sql
ALTER TABLE books
DROP CONSTRAINT books_publisher_id_fkey
```

```sql
ALTER TABLE books
ADD CONSTRAINT books_publisher_id_fkey
    FOREIGN KEY (publisher_id)
       REFERENCES publishers (publisher_id)
    ON DELETE CASCADE ON UPDATE NO ACTION;
```

Table

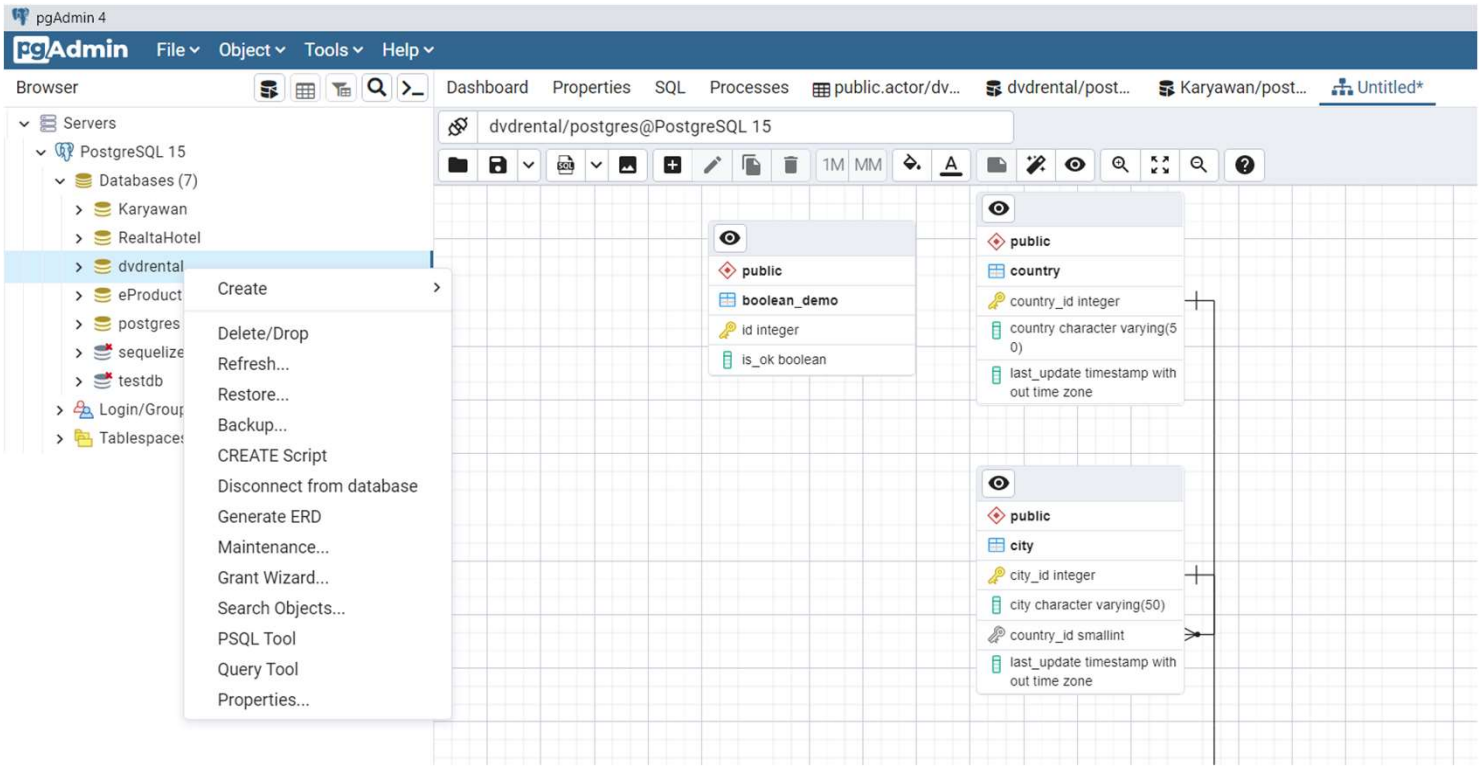
```
ALTER TABLE publishers
DROP COLUMN publisher_id cascade;


drop table books;
drop table publishers;
drop table categories;

drop table publishers cascade;
drop table categories cascade;
drop table books;
```

Generate ERD

Klik kanan nama database lalu pilih menu Generate ERD



Crud (Insert,Update,Delete)

Insert digunakan untuk melakukan tambah data pada table di database.

```
INSERT INTO table_name(column1, column2, …)
VALUES (value1, value2, …);
```

```
INSERT INTO table_name(column1, column2, …)
VALUES (value1, value2, …)
RETURNING *;
```

```
CREATE TABLE links (
        id SERIAL PRIMARY KEY,
        url VARCHAR(255) NOT NULL,
        name VARCHAR(255) NOT NULL,
        description VARCHAR (255),
        last_update DATE
);
```

```
INSERT INTO links (url, name, last_update)
VALUES('https://www.google.com','Google','2013-06-01')
returning *;
```

```
INSERT INTO links (url, name)
VALUES('http://www.postgresql.org','PostgreSQL')
RETURNING id;
```

```
INSERT INTO links (url, name)
VALUES('http://www.oreilly.com','O''Reilly Media');
```

Crud (Insert,Update,Delete)

Sintaks insert multiple rows:

```
INSERT INTO table_name (column_list)
VALUES
    (value_list_1),
    (value_list_2),
    ...
    (value_list_n);
```

```
INSERT INTO
    links(url,name, description)
VALUES
    ('https://duckduckgo.com/','DuckDuckGo','Privacy & Simplified Search Engine
    ('https://swisscows.com/','Swisscows','Privacy safe WEB-search')
RETURNING *;
```

```
CREATE TABLE person (
        id SERIAL PRIMARY KEY,
        first_name VARCHAR (50),
        last_name VARCHAR (50),
        email VARCHAR (50) UNIQUE
);
```

=

```
CREATE TABLE person (
        id SERIAL  PRIMARY KEY,
        first_name VARCHAR (50),
        last_name  VARCHAR (50),
        email      VARCHAR (50),
        UNIQUE(email)
);
```

Crud (Insert,Update,Delete)

Contoh penggunaan constraint Unique:

```
INSERT INTO person(first_name,last_name,email)
VALUES('john','doe','j.doe@postgresqltutorial.com');
```

```
INSERT INTO person(first_name,last_name,email)
VALUES('jack','doe','j.doe@postgresqltutorial.com');
```

```
[Err] ERROR:  duplicate key value violates unique constraint "person_email_key"
DETAIL:  Key (email)=(j.doe@postgresqltutorial.com) already exists.
```

Crud (Insert,Update,Delete)

Update digunakan untuk mengubah data pada table di database.

```
UPDATE table_name
SET column1 = value1,
    column2 = value2,
    ...
WHERE condition;
```

Update Join adalah update data table yang nilai barunya berkaitan dengan data di table lain.

```
UPDATE t1
SET t1.c1 = new_value
FROM t2
WHERE t1.c2 = t2.c2;
```

Crud (Insert,Update,Delete)

Contoh Update Join:

```sql
CREATE TABLE product_segment (
    id SERIAL PRIMARY KEY,
    segment VARCHAR NOT NULL,
    discount NUMERIC (4, 2)
);
```

```sql
INSERT INTO
    product_segment (segment, discount)
VALUES
    ('Grand Luxury', 0.05),
    ('Luxury', 0.06),
    ('Mass', 0.1);
```

```sql
CREATE TABLE product(
    id SERIAL PRIMARY KEY,
    name VARCHAR NOT NULL,
    price NUMERIC(10,2),
    net_price NUMERIC(10,2),
    segment_id INT NOT NULL,
    FOREIGN KEY(segment_id) REFERENCES product_segment(id)
);
```

```sql
INSERT INTO
    product (name, price, segment_id)
VALUES
    ('diam', 804.89, 1),
    ('vestibulum aliquet', 228.55, 3),
    ('lacinia erat', 366.45, 2),
    ('scelerisque quam turpis', 145.33, 3),
    ('justo lacinia', 551.77, 2),
```

Crud (Insert,Update,Delete)

Lanjutan contoh Update Join:

```
UPDATE product
SET net_price = price - price * discount
FROM product_segment
WHERE product.segment_id = product_segment.id;
```

```
UPDATE
    product p
SET
    net_price = price - price * discount
FROM
    product_segment s
WHERE
    p.segment_id = s.id;
```

Crud (Insert,Update,Delete)

Delete digunakan untuk menghapus data dari table di database.

```
DELETE FROM table_name
WHERE condition;
```

```
DELETE FROM links
WHERE id = 8;
```

```
DELETE FROM links
WHERE id = 7
RETURNING *;
```

```
DELETE FROM links
WHERE id IN (6,5)
RETURNING *;
```

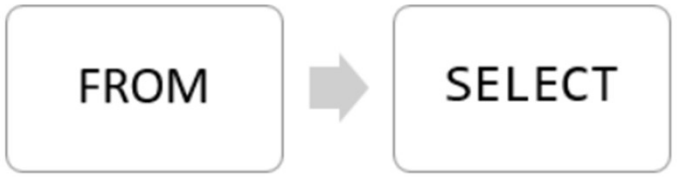
```
DELETE FROM links;
```

Select - distinct

Select digunakan untuk mengambil data dari table di database.

```
SELECT * FROM customer;
```

```
SELECT
   first_name,
   last_name,
   email
FROM
   customer;
```




```
SELECT
   DISTINCT column1
FROM
   table_name;
```

Select - where

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<> or !=	Not equal
AND	Logical operator AND
OR	Logical operator OR
IN	Return true if a value matches any value in a list
BETWEEN	Return true if a value is between a range of values
LIKE	Return true if a value matches a pattern
IS NULL	Return true if a value is NULL
NOT	Negate the result of other operators

Select - where

Operator LIKE bersifat case-sensitif, yang berarti huruf besar dan huruf kecil dibedakan.

Sedangkan operator ILIKE bersifat case-insensitif, yang berarti mengabaikan apakah data yang dicari huruf besar atau huruf kecil

Operator	Equivalent
~~	LIKE
~~*	ILIKE
!~~	NOT LIKE
!~~*	NOT ILIKE

Select - limit

```
SELECT
        film_id,
        title,
        release_year
FROM
        film
ORDER BY
        film_id
LIMIT 5;
```

	film_id integer	title character varying (255)	release_year integer
1	1	Academy Dinosaur	2006
2	2	Ace Goldfinger	2006
3	3	Adaptation Holes	2006
4	4	Affair Prejudice	2006
5	5	African Egg	2006

```
SELECT
        film_id,
        title,
        release_year
FROM
        film
ORDER BY
        film_id
LIMIT 4 OFFSET 3;
```

	film_id integer	title character varying (255)	release_year integer
1	4	Affair Prejudice	2006
2	5	African Egg	2006
3	6	Agent Truman	2006
4	7	Airplane Sierra	2006

Select - fetch

```
SELECT
    film_id,
    title
FROM
    film
ORDER BY
    title
FETCH FIRST ROW ONLY;
```

	film_id integer	title character varying (255)
1	1	Academy Dinosaur

```
SELECT
    film_id,
    title
FROM
    film
ORDER BY
    title
FETCH FIRST 5 ROW ONLY;
```

	film_id integer	title character varying (255)
1	1	Academy Dinosaur
2	2	Ace Goldfinger
3	3	Adaptation Holes
4	4	Affair Prejudice
5	5	African Egg

```
SELECT
    film_id,
    title
FROM
    film
ORDER BY
    title
OFFSET 5 ROWS
FETCH FIRST 5 ROW ONLY;
```

	film_id integer	title character varying (255)
1	6	Agent Truman
2	7	Airplane Sierra
3	8	Airport Pollock
4	9	Alabama Devil
5	10	Aladdin Calendar

Aggregate Function

Aggregate Function yang terdapat pada postgresql antara lain:

AVG() => untuk menghitung nilai rata-rata suatu kumpulan data

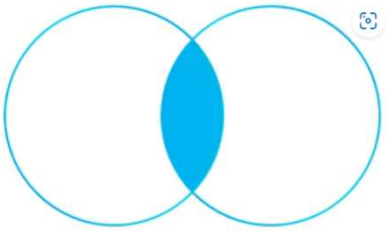
COUNT() => untuk menghitung jumlah baris data

MAX() => untuk memperoleh nilai terbesar suatu kumpulan data

MIN() => untuk memperoleh nilai terkecil suatu kumpulan data

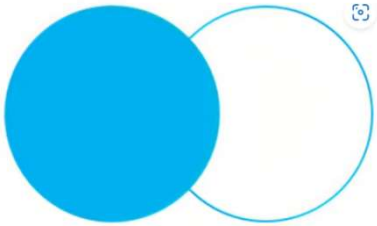
SUM() => untuk menghitung jumlah total suatu kumpulan data

Select Multiple Table - Join



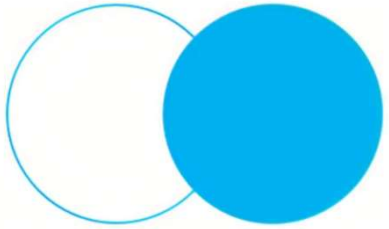
Inner Join

```
SELECT * FROM a
INNER JOIN b ON a.key = b.key
```



Left Inner Join

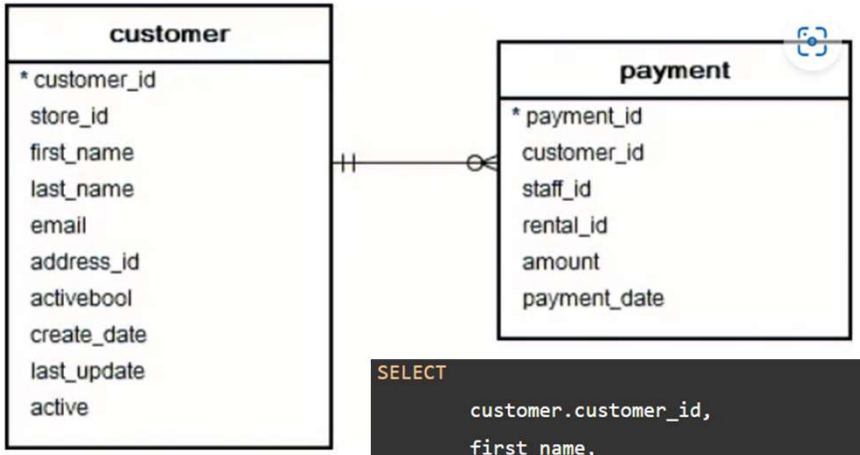
```
SELECT * FROM a
LEFT JOIN b ON a.key = b.key
```



Right Inner Join

```
SELECT * FROM a
RIGHT JOIN b ON a.key = b.key
```

Join-Inner Join




```sql
SELECT
        customer.customer_id,
        first_name,
        last_name,
        amount,
        payment_date
FROM
        customer
INNER JOIN payment
    ON payment.customer_id = customer.customer_id
ORDER BY payment_date;
```

```sql
SELECT
        c.customer_id,
        first_name,
        last_name,
        email,
        amount,
        payment_date
FROM
        customer c
INNER JOIN payment p
    ON p.customer_id = c.customer_id
WHERE
    c.customer_id = 2;
```

Join-Inner Join

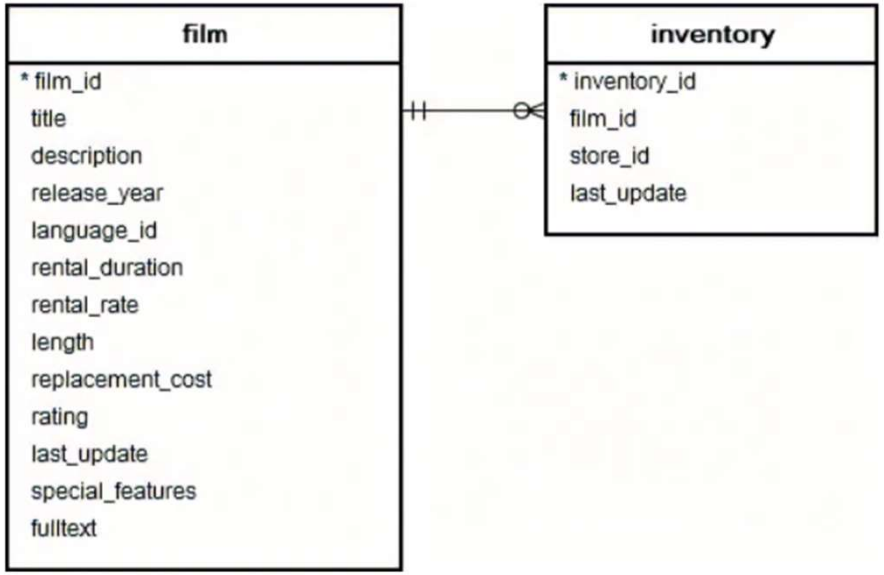
	customer_id integer	first_name character varying (45)	last_name character varying (45)	amount numeric (5,2)	payment_date timestamp without time zone
1	416	Jeffery	Pinson	2.99	2007-02-14 21:21:59.996577
2	516	Elmer	Noe	4.99	2007-02-14 21:23:39.996577
3	239	Minnie	Romero	4.99	2007-02-14 21:29:00.996577
4	592	Terrance	Roush	6.99	2007-02-14 21:41:12.996577
5	49	Joyce	Edwards	0.99	2007-02-14 21:44:52.996577
6	264	Gwendolyn	May	3.99	2007-02-14 21:44:53.996577
7	46	Catherine	Campbell	4.99	2007-02-14 21:45:29.996577
8	481	Herman	Devore	2.99	2007-02-14 22:03:35.996577
9	139	Amber	Dixon	2.99	2007-02-14 22:11:22.996577

Join-Inner Join

```
SELECT
        customer_id,
        first_name,
        last_name,
        amount,
        payment_date
FROM
        customer
INNER JOIN payment USING(customer_id)
ORDER BY payment_date;
```

	customer_id integer	first_name character varying (45)	last_name character varying (45)	amount numeric (5,2)	payment_date timestamp without time zone
1	416	Jeffery	Pinson	2.99	2007-02-14 21:21:59.996577
2	516	Elmer	Noe	4.99	2007-02-14 21:23:39.996577
3	239	Minnie	Romero	4.99	2007-02-14 21:29:00.996577
4	592	Terrance	Roush	6.99	2007-02-14 21:41:12.996577
5	49	Joyce	Edwards	0.99	2007-02-14 21:44:52.996577
6	264	Gwendolyn	May	3.99	2007-02-14 21:44:53.996577
7	46	Catherine	Campbell	4.99	2007-02-14 21:45:29.996577
8	481	Herman	Devore	2.99	2007-02-14 22:03:35.996577
9	139	Amber	Dixon	2.99	2007-02-14 22:11:22.996577

Join-left Inner Join




```
SELECT
        film.film_id,
        title,
        inventory_id
FROM
        film
LEFT JOIN inventory
    ON inventory.film_id = film.film_id
ORDER BY title;
```

Join-right Inner Join

Kebalikan dari left inner join

Subquery

Nested query, yaitu query dalam query.

Contoh soal: Menampilkan film yang rental rate nya diatas rata-rata rental rate.

Langkah penyelesaian:

1. Mendapatkan nilai rata-rata rental rate film

```
select round(avg(rental_rate),2) from film;
```

	avg_rate numeric
1	2.98

1. Menampilkan film dengan kondisi rental rate diatas nilai rata-rata yang diperoleh pada langkah 1.

```
select title,rental_rate from film where rental_rate > 2.98
```

Subquery

```
SELECT title,rental_rate
FROM film
WHERE
    rental_rate > (
        SELECT AVG (rental_rate)
        FROM film
    );
```

Subquery sebagai alternatif inner join.

```
SELECT distinct film.film_id,title
FROM film
INNER JOIN inventory ON inventory.film_id = film.film_id
order by film.film_id
```

```
SELECT film_id,title
FROM film
WHERE
    film_id IN (
        SELECT film_id
        FROM inventory
        )
order by film_id
```

Subquery

```sql
SELECT distinct first_name,last_name
FROM customer join payment
on payment.customer_id = customer.customer_id
```

```sql
SELECT first_name,last_name
FROM customer
WHERE
EXISTS (
    SELECT 1
    FROM payment
    WHERE payment.customer_id = customer.customer_id
);
```

```sql
SELECT first_name,last_name
FROM customer
WHERE customer_id
in (
    SELECT distinct customer_id
    FROM payment
);
```

Union

Union digunakan untuk menggabungkan data dari dua atau lebih statement select. Union berbeda dengan Union All. Union menggabungkan data namun mengabaikan data yang sama (duplikat). Union All menggabungkan semua data termasuk yang duplikat dari dua atau lebih statement select.

Syarat: Kolom yang ditampilkan harus sama

```
SELECT * FROM top_rated_films;
```

	title character varying	release_year smallint
1	The Shawshank Redemption	1994
2	The Godfather	1972
3	12 Angry Men	1957

```
SELECT * FROM most_popular_films;
```

	title character varying	release_year smallint
1	An American Pickle	2020
2	The Godfather	1972
3	Greyhound	2020

Union

```
SELECT * FROM top_rated_films
UNION
SELECT * FROM most_popular_films;
```

	title character varying	release_year smallint
1	An American Pickle	2020
2	Greyhound	2020
3	The Shawshank Redemption	1994
4	The Godfather	1972
5	12 Angry Men	1957

```
SELECT * FROM top_rated_films
UNION ALL
SELECT * FROM most_popular_films;
```

	title character varying	release_year smallint
1	The Shawshank Redemption	1994
2	The Godfather	1972
3	12 Angry Men	1957
4	An American Pickle	2020
5	The Godfather	1972
6	Greyhound	2020

View

View sering disebut virtual table, karena datanya berasal dari hasil query.

Contoh:

```
CREATE VIEW item_discount AS
select item.id,price, discount
from item join sales_item on item.id=sales_item.item_id
order by item.id
```

Menggunakan view dengan cara yang sama menggunakan table:

```
select * from item_discount
```

```
drop view item_discount
```

	id integer	price numeric (6,2)	discount numeric (3,2)
1	1	199.60	0.19
2	1	199.60	0.10
3	1	199.60	0.17

View - Latihan View

	purchase_order_number bigint	company character varying (60)	quantity integer	supplier character varying (30)	name character varying (30)	price numeric (6,2)	total numeric	salesperson text
1	20166617	Verizon	2	Johnston & Murphy	Malek	154.62	309.24	Samantha Moore
2	20173238	Kroger	2	Allen Edmonds	Grandview	153.97	307.94	Jennifer Smith
3	20173238	Kroger	2	Johnston & Murphy	Ramsey	166.87	333.74	Jennifer Smith
4	201611637	IBM	2	Johnston & Murphy	Ramsey	159.82	319.64	Michael Robins...
5	201647847	TJX	1	Allen Edmonds	Grandview	128.77	128.77	Jennifer Smith

View - Latihan View

```sql
CREATE VIEW purchase_order_overview AS
SELECT sales_order.purchase_order_number, customer.company,
sales_item.quantity, product.supplier, product.name, item.price,
(sales_item.quantity * item.price) AS Total,
CONCAT(sales_person.first_name, ' ', sales_person.last_name) AS Salesperson
FROM sales_order JOIN sales_item
ON sales_item.sales_order_id = sales_order.id
JOIN item ON item.id = sales_item.item_id
JOIN customer ON sales_order.cust_id = customer.id
JOIN product ON product.id = item.product_id
JOIN sales_person ON sales_person.id = sales_order.sales_person_id
ORDER BY purchase_order_number;
```

Function

Sintaks Fungsi dalam database terdiri dari 2 bagian utama, yaitu:

Bagian header dan bagian body.

Header berisi keyword create function diikuti nama fungsi, parameter(optional), type output, dan langguage.

```
create [or replace] function function_name(param_list)
   returns return_type
   language plpgsql
  as
```

Function

Body berisi block yang terdiri dari deklarasi variable dan logic . Body biasanya diawali dengan $$ dan diakhiri dengan $$

```
$$
declare
-- variable declaration
begin
 -- logic
end;
$$
```

```
create or replace function get_film_count(len_from int, len_to int)
returns int
language plpgsql
as
$$
declare
   film_count integer;
begin
   select count(*)
   into film_count
   from film
   where length between len_from and len_to;

   return film_count;
end;
$$;
```

```
select get_film_count(40,90);
```

Function- return type copy dari column

```
create or replace function find_film_by_id(p_film_id int)
returns film.title%type
language plpgsql
as $$
declare
   film_title film.title%type;
begin
  -- find film title by id
  select title
  into film_title
  from film
  where film_id = p_film_id;

  if not found then
     raise 'Film with id % not found', p_film_id;
  end if;

  return film_title;

end;
$$;
```

Function-return type row copy dari row table

```
create or replace function find_film_by_id_rowtype(p_film_id int)
returns film
language plpgsql
as $$
declare
   film_title film;
begin
  -- find film title by id
  select *
  into film_title
  from film
  where film_id = p_film_id;

  if not found then
     raise 'Film with id % not found', p_film_id;
  end if;

  return film_title;

end;
$$;
```

Function-return type table

```
create or replace function get_film ()
returns table (
        film_title varchar,
        film_release_year int
    )
    language plpgsql
as
$$
begin
    return query
            select
                title,
                release_year::integer
            from
               film;

end;
$$;
```

```
create or replace function get_film_uppercae ()
returns table (
    film_title varchar,
    film_release_year int
)
language plpgsql
as
$$
declare
    var_r record;
begin
    for var_r in(
             select title, release_year
             from film
        ) loop
        film_title := upper(var_r.title) ;
        film_release_year := var_r.release_year;
           return next;
    end loop;
end;
$$
```

```
create or replace function get_data()
returns setof film
language plpgsql
as
$$
begin
      return query
      select * from film;
end;
$$;
```

Function-tanpa deklarasi returns

```
create or replace function find_film_by_id_param(
    in p_film_id int,
    out film_title varchar)
language plpgsql
as
$$

begin
  -- find film title by id
  select title
  into film_title
  from film
  where film_id = p_film_id;

  if not found then
     raise 'Film with id % not found', p_film_id;
  end if;


end;
$$;
```

```
create or replace function get_film_stat(
    out min_len int,
    out max_len int,
    out avg_len numeric)
language plpgsql
as $$
begin

  select min(length),
         max(length),
         avg(length)::numeric(5,1)
  into min_len, max_len, avg_len
  from film;

end;
$$;
```

Function-dengan cursor

```
create or replace function get_film_titles(p_year integer)
   returns text
   language plpgsql
   as
   $$
declare
     titles text default '';
     rec_film   record;
     cur_films cursor(p_year integer)
         for select title, release_year
         from film
         where release_year = p_year;
begin
   -- open the cursor
   open cur_films(p_year);

   loop
    -- fetch row into the film
      fetch cur_films into rec_film;
    -- exit when no more row to fetch
      exit when not found;

    -- build the output
      if rec_film.title like '%ful%' then
         titles := titles || ',' || rec_film.title || ':' || rec_film.release_year;
      end if;
   end loop;

   -- close the cursor
   close cur_films;

   return titles;
end;
$$
```

Latihan Function - tanpa/pakai cursor

state character (2)	total_customer bigint
CA	4
GA	1
IL	1
NJ	2
TX	11

```
create or replace function cust_dashboard()
returns table(
 state_cust customer.state%type,
 total_customer int
)
-- returns setof record
language plpgsql
as
$$
-- declare
--   rec_cust record;
begin
    return query
       select state,cast(count(*) as integer)
       from customer
       group by state;

--      return next rec_cust;
end;
$$
```

```
create or replace function cust_dashboard_cursor()
returns table(
 state_cust customer.state%type,
 total_customer int
)
language plpgsql
as
$$
declare
  cust_cursor cursor for
       select state,cast(count(*) as int) from customer
       group by state;
begin
    open cust_cursor;
     return query
       fetch all cust_cursor;
    close cust_cursor;

end;
$$;
```

Procedure

Fungsi tidak dapat melakukan transaction dan commit atau rollback, hal ini dapat dilakukan dalam store procedure (SP). Selain itu, perbedaan SP dengan Fungsi yaitu SP tidak ada return, sedangkan fungsi ada return

```
create [or replace] procedure procedure_name(parameter_list)
language plpgsql
as $$
declare
-- variable declaration
begin
-- stored procedure body
end; $$
```

Procedure

```
select * from accounts;
```

	id integer	name character varying (100)	balance numeric (15,2)
1	1	Bob	10000.00
2	2	Alice	10000.00

```
call transfer(1,2,1000);
```

```
SELECT * FROM accounts;
```

	id integer	name character varying (100)	balance numeric (15,2)
1	1	Bob	9000.00
2	2	Alice	11000.00

```
create or replace procedure transfer(
   sender int,
   receiver int,
   amount dec
)
language plpgsql
as $$
begin
    -- subtracting the amount from the sender's account
    update accounts
    set balance = balance - amount
    where id = sender;

    -- adding the amount to the receiver's account
    update accounts
    set balance = balance + amount
    where id = receiver;

    commit;
end;$$
```

Transaction

Sintaks diawali dengan:

Begin Transaction atau Begin Work

atau Begin.

Diakhiri dengan Commit atau Rollback

```
BEGIN;

-- deduct 1000 from account 1
UPDATE accounts
SET balance = balance - 1000
WHERE id = 1;

-- add 1000 to account 2
UPDATE accounts
SET balance = balance + 1000
WHERE id = 2;

-- select the data from accounts
SELECT id, name, balance
FROM accounts;

-- commit the transaction
COMMIT;
```

id	name	balance
1	Bob	10000
2	Alice	10000

id	name	balance
1	Bob	9000
2	Alice	11000

Transaction

```
INSERT INTO accounts(name, balance)
VALUES('Jack',0);
```

id	name	balance
1	Bob	9000
2	Alice	11000
3	Jack	0

```
BEGIN;

-- deduct the amount from the account 1
UPDATE accounts
SET balance = balance - 1500
WHERE id = 1;

-- add the amount from the account 3 (instead of 2)
UPDATE accounts
SET balance = balance + 1500
WHERE id = 3;

-- roll back the transaction
ROLLBACK;
```

id	name	balance
1	Bob	9000
2	Alice	11000
3	Jack	0

Transaction

```
CREATE OR REPLACE PROCEDURE updateorder(
                        IN data json,
                        IN data2 json)
LANGUAGE 'plpgsql'
AS $$
declare
   rows record;
                        r record;
                        dt record;
begin
  begin
    select * from json_to_recordset(data2) as y(id int, totalproduct int,totalprice int) into dt;
    update orders set totalproduct=dt.totalproduct, totalprice=dt.totalprice where id=dt.id returning id
                         into rows;
   FOR r IN select * from json_to_recordset(data) as x(id int, quantity int)
   LOOP
    update order_detail set quantity=r.quantity where id=r.id;
                         if not found or rows is null then
                           rollback;
                             raise 'Id tidak ditemukan';
                         else
       commit;
                         end if;
                        END LOOP;
                        end;
end;
$$;
```