
PostgreSQL Database

Eko Kurniawan Khannedy

Eko Kurniawan Khannedy

- Technical architect at one of the biggest ecommerce company in Indonesia
- 12+ years experiences
- www.programmerzamannow.com
- youtube.com/c/ProgrammerZamanNow



Eko Kurniawan Khannedy

- Telegram : [@khannedy](https://t.me/khannedy)
- Linkedin : <https://www.linkedin.com/company/programmer-zaman-now/>
- Facebook : [fb.com/ProgrammerZamanNow](https://www.facebook.com/ProgrammerZamanNow)
- Instagram : [instagram.com/programmerzamannow](https://www.instagram.com/programmerzamannow)
- Youtube : [youtube.com/c/ProgrammerZamanNow](https://www.youtube.com/c/ProgrammerZamanNow)
- Telegram Channel : t.me/ProgrammerZamanNow
- Tiktok : [https://tiktok.com/@programmerzamannow](https://www.tiktok.com/@programmerzamannow)
- Email : echo.khannedy@gmail.com

Agenda

- Pengenalan PostgreSQL
- Menginstall PostgreSQL
- Tipe Data
- Database,Table
- Insert, Update, Delete, Select
- Transaction
- Table Relationship
- Join
- Dan lain-lain

Pengenalan Sistem Basis Data

Pengenalan Database Management System

- DBMS adalah aplikasi yang digunakan untuk me-manage data
- Tanpa menggunakan DBMS, untuk me-manage data, seperti data produk, data customer, data penjualan, kita harus simpan dalam bentuk file (misal seperti ketika menggunakan Excel)
- DBMS biasanya berjalan sebagai aplikasi server yang digunakan untuk me-manage data, kita hanya tinggal memberi perintah ke DBMS untuk melakukan proses manajemen datanya, seperti menambah, mengubah, menghapus atau mengambil data
- Contoh DBMS yang populer seperti MySQL, PostgreSQL, MongoDB, Oracle, dan lain-lain

Pengenalan Relational Database

- Ada banyak sekali jenis-jenis DBMS, seperti Relational Database, Document Database, Key-Value Database, dan lain-lain
- Namun yang masih populer dan kebanyakan orang gunakan adalah relational database
- Relational database cukup mudah dimengerti dan dipelajari karena kita sudah terbiasa menyimpan data dalam bentuk tabular (tabel) seperti di Microsoft Excel atau di Google Doc Spreadsheet
- Selain itu relational database memiliki perintah standard menggunakan SQL, sehingga kita mudah ketika ingin berganti-ganti aplikasi database (seperti MySQL, Oracle, PostgreSQL dan lain-lain)

Cara Kerja DBMS



Database Client

- Database client adalah aplikasi yang digunakan untuk berkomunikasi dengan DBMS
- Biasanya DBMS sudah menyediakan database client sederhana yang bisa kita gunakan untuk berkomunikasi dengan DBMS agar lebih mudah
- Atau kita bisa membuat aplikasi untuk berkomunikasi dengan DBMS, misal membuat aplikasi database client menggunakan Java, PHP atau bahasa pemrograman lainnya

Database File

- Mayoritas DBMS menyimpan datanya di file, walaupun ada beberapa database yang hanya menyimpan datanya di memory (RAM)
- Namun jangan berpikir file database yang disimpan berupa file seperti Excel atau CSV (Comma Separated Value), tapi jauh lebih kompleks
- Database File akan di optimasi oleh DBMS agar mempermudah DBMS dalam manajemen datanya, seperti insert, update, delete dan select
- Tiap DBMS biasanya memiliki cara masing-masing mengelola Database File nya, dan kita tidak perlu harus tau, karena yang kita perlu tahu hanya cara berkomunikasi ke DBMS

SQL

- Structured Query Language
- Merupakan bahasa yang digunakan untuk mengirim perintah ke DBMS
- SQL adalah bahasa yang mudah karena hanya berisi instruksi untuk menyimpan, mengubah, menghapus atau mengambil data melalui DBMS
- Secara garis besar, semua perintah SQL di Relational Database itu hampir sama, namun biasanya tiap DBMS ada improvement yang membedakan hal-hal kecil dalam perintah SQL, namun secara garis besar perintahnya tetap sama

Pengenalan PostgreSQL



PostgreSQL

- PostgreSQL adalah DBMS Relational OpenSource yang sangat populer, terutama di perusahaan enterprise
- Tidak hanya OpenSource, PostgreSQL juga gratis untuk digunakan
- Proyek PostgreSQL dimulai sejak tahun 1986, dibawah arahan Professor Michael Stonebreaker di Universitas California, Berkeley
- PostgreSQL sangat populer sekali dikalangan perusahaan Enterprise
- <https://www.postgresql.org/>

Kenapa Belajar PostgreSQL?

Rank				DBMS	Database Model	Score		
	May 2023	Apr 2023	May 2022			May 2023	Apr 2023	May 2022
1.	1.	1.	Oracle	Oracle	Relational, Multi-model	1232.64	+4.36	-30.18
2.	2.	2.	MySQL	MySQL	Relational, Multi-model	1172.46	+14.68	-29.64
3.	3.	3.	Microsoft SQL Server	Microsoft SQL Server	Relational, Multi-model	920.09	+1.57	-21.11
4.	4.	4.	PostgreSQL	PostgreSQL	Relational, Multi-model	617.90	+9.49	+2.61
5.	5.	5.	IBM Db2	IBM Db2	Relational, Multi-model	143.02	-2.48	-17.31
6.	6.	↑ 7.	SQLite	SQLite	Relational	133.86	-0.68	-0.87
7.	7.	↓ 6.	Microsoft Access	Microsoft Access	Relational	131.17	-0.20	-12.27
8.	8.	↑ 9.	Snowflake	Snowflake	Relational	111.73	+0.60	+18.22
9.	9.	↓ 8.	MariaDB	MariaDB	Relational, Multi-model	96.87	+0.93	-14.26
10.	10.	10.	Microsoft Azure SQL Database	Microsoft Azure SQL Database	Relational, Multi-model	79.19	+0.13	-6.14

PostgreSQL vs MySQL

- MySQL sampai sekarang menjadi salah satu DBMS OpenSource yang paling populer di dunia
- Namun, banyak perusahaan besar yang menggunakan database PostgreSQL, hal ini dikarenakan PostgreSQL memiliki fitur yang lebih kaya dibandingkan MySQL
- MySQL fokus pada performa dan kecepatan, oleh karena itu fitur nya tidak sebanyak di PostgreSQL

Menginstall PostgreSQL

Menginstall PostgreSQL

- Menginstall PostgreSQL banyak caranya, bisa download langsung dari halaman website resminya
- <https://www.postgresql.org/download/>

Menginstall PostgreSQL di Mac / Linux

- Khusus untuk pengguna Mac / Linux, selain download installer PostgreSQL di website resmi PostgreSQL, kita juga bisa menggunakan homebrew untuk menginstall PostgreSQL
- <https://brew.sh/>
- Cukup gunakan perintah : brew install postgresql@15
- Tergantung versi PostgreSQL yang tersedia

Menggunakan PostgreSQL Client

- PostgreSQL Client adalah aplikasi berbasis terminal yang disediakan oleh PostgreSQL untuk berkomunikasi dengan PostgreSQL Server
- Karena berbasis terminal, sehingga PostgreSQL Client sangat cocok untuk kita gunakan misal ketika di server production, dimana kita menginstall PostgreSQL di linux server yang berbasis terminal misal
- Kita tidak perlu menginstall PostgreSQL Client secara terpisah, karena sudah tersedia di dalam aplikasi PostgreSQL ketika kita menginstallnya
- PostgreSQL Client bisa diakses lewat terminal dengan nama program psql

DBeaver

- DBeaver adalah aplikasi GUI Client berbasis Desktop yang free dan opensource yang bisa digunakan sebagai aplikasi database client
- Aplikasi DBeaver sangat mempermudah kita melakukan manajemen data di PostgreSQL karena berbasis Desktop
- <https://dbeaver.io/>

PGAdmin

- PGAdmin adalah aplikasi berbasis web yang bisa digunakan untuk manajemen data di PostgreSQL
- Aplikasi PGAdmin dulunya berupa aplikasi desktop, namun versi terbaru diubah menjadi aplikasi web
- <https://www.pgadmin.org/>

JetBrains DataGrip

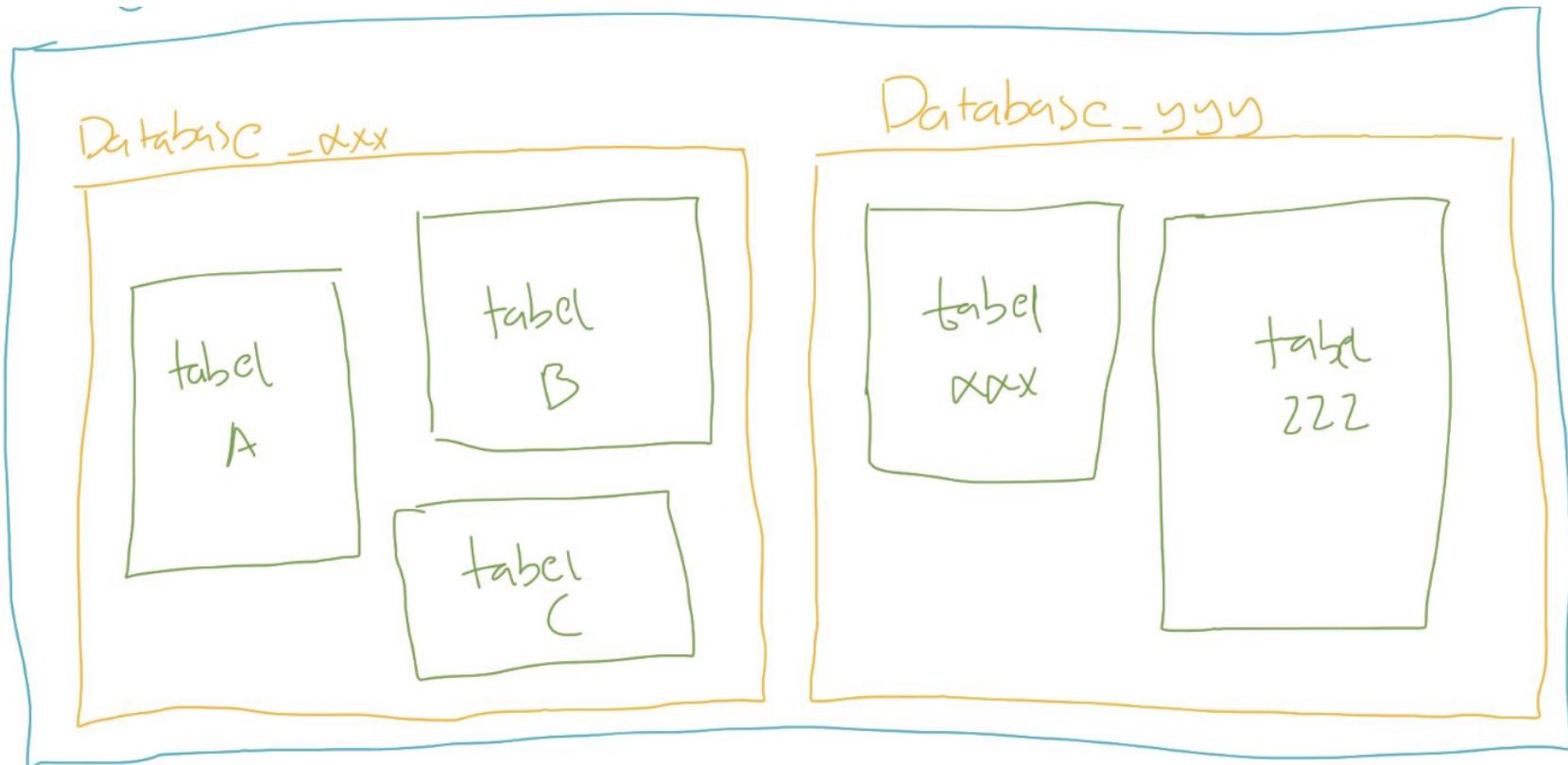
- DataGrip adalah aplikasi Database Client yang berbayar
- DataGrip mendukung banyak sekali DBMS sehingga kita cukup menggunakan DataGrip untuk manajemen semua database yang kita gunakan
- Selain mendukung Relational DBMS, DataGrip juga mendukung DBMS yang NoSQL seperti MongoDB, Cassandra, dan lain-lain
- <https://www.jetbrains.com/datagrip/>

Database

Database

- Database adalah tempat kita menyimpan table di PostgreSQL
- Jika kita misalkan table di PostgreSQL adalah sebuah file, maka database adalah folder nya, dimana kita bisa menyimpan banyak table di sebuah database
- Biasanya pembuatan kita akan membuat satu database untuk satu jenis aplikasi, walaupun satu aplikasi bisa menggunakan lebih dari satu database, namun lumrahnya, satu aplikasi akan menggunakan satu database

DBMS



Menggunakan PostgreSQL Client

- `psql --host=localhost --port=5432 --dbname=postgres --username=khannedy --password`

Melihat Semua Database di PostgreSQL

- \l
- select datname from pg_database;

Membuat Database

```
create database nama_database;
```

Menghapus Database

```
drop database nama_database;
```

Menggunakan Database

\c namadatabase

Tipe Data

Tipe Data

- Saat kita membuat tabel di Excel, kita bisa menentukan tipe data apa yang kita masukkan ke tiap kolom di Excel
- Di PostgreSQL, kita juga bisa menentukan tipe data tiap kolom yang kita buat di sebuah tabel
- Ada banyak sekali tipe data yang tersedia di PostgreSQL, dari yang sederhana, sampai yang kompleks.
- Biasanya kita akan menggunakan tipe data sesuai dengan kebutuhan kolom yang perlu kita buat

Tipe Data per Kolom

Id (number)	Nama (text)	Harga (number)	Jumlah (number)
1	Apel	5000	100
2	Jeruk	2000	200
3	Semangka	10000	50
...

Tipe Data Number

Tipe Data Number

- Secara garis besar, tipe data number di PostgreSQL ada dua jenis;
- Integer, atau tipe number bilangan bulat
- Floating Point, atau tipe data number pecahan



Tipe Data Number

Name	Storage Size	Description	Range
<code>smallint</code>	2 bytes	small-range integer	-32768 to +32767
<code>integer</code>	4 bytes	typical choice for integer	-2147483648 to +2147483647
<code>bigint</code>	8 bytes	large-range integer	-9223372036854775808 to +9223372036854775807
<code>decimal</code>	variable	user-specified precision, exact	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
<code>numeric</code>	variable	user-specified precision, exact	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
<code>real</code>	4 bytes	variable-precision, inexact	6 decimal digits precision
<code>double precision</code>	8 bytes	variable-precision, inexact	15 decimal digits precision
<code>smallserial</code>	2 bytes	small autoincrementing integer	1 to 32767
<code>serial</code>	4 bytes	autoincrementing integer	1 to 2147483647
<code>bigserial</code>	8 bytes	large autoincrementing integer	1 to 9223372036854775807

DECIMAL / NUMERIC

- Selain Integer dan Floating Point, di PostgreSQL terdapat tipe data DECIMAL / NUMERIC
- Ini tipe data number khusus yang bisa ditentukan jumlah precision dan scale nya

NUMERIC	Min	Max
NUMERIC(5, 2)	-999.99	999.99
NUMERIC(5, 0)	-99999	99999
NUMERIC(3, 1)	-99.9	999
NUMERIC(3)	-999	999

Tipe Data String

Tipe Data String

- Selain number, biasanya kita sering menyimpan data di dalam tabel dalam bentuk tulisan
- Tipe data ini namanya tipe data String atau Text
- Ada banyak tipe data String di PostgreSQL

CHAR dan VARCHAR

- Pertama tipe data String di PostgreSQL adalah CHAR dan VARCHAR
- Kita bisa menentukan jumlah panjang maksimal karakter yang bisa ditampung oleh CHAR dan VARCHAR dengan menggunakan kurung buka lalu masukan jumlah maksimal karakter dan diakhiri kurung tutup
- Misal, CHAR(10) atau VARCHAR(10) artinya tipe data String dengan maksimal jumlah karakternya adalah 10 karakter
- Maksimum ukuran CHAR atau VARCHAR adalah 65535 karakter



Perbedaan CHAR dan VARCHAR

Value	CHAR (4)	Storage Required	VARCHAR (4)	Storage Required
''	' '	4 bytes	''	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

TEXT

- Selain CHAR dan VARCHAR, tipe data String yang lainnya adalah TEXT
- Berbeda dengan CHAR dan VARCHAR yang kita bisa tentukan panjang maksimum nya, TEXT tidak memiliki maksimum panjang nya

Tipe Data Date dan Time

Tipe Data Date dan Time

- Selain tipe data Number dan String, biasanya kadang kita sering menyimpan data waktu atau tanggal
- Sebenarnya bisa kita gunakan String untuk menyimpan data waktu atau tanggal, namun itu tidak direkomendasikan, karena akan menyulitkan kita ketika nanti butuh melakukan manipulasi waktu atau tanggal di PostgreSQL

Jenis-Jenis Tipe Data Date dan Time

Name	Storage Size	Description	Low Value	High Value	Resolution
<code>timestamp [(p)] [without time zone]</code>	8 bytes	both date and time (no time zone)	4713 BC	294276 AD	1 microsecond
<code>timestamp [(p)] with time zone</code>	8 bytes	both date and time, with time zone	4713 BC	294276 AD	1 microsecond
<code>date</code>	4 bytes	date (no time of day)	4713 BC	5874897 AD	1 day
<code>time [(p)] [without time zone]</code>	8 bytes	time of day (no date)	00:00:00	24:00:00	1 microsecond
<code>time [(p)] with time zone</code>	12 bytes	time of day (no date), with time zone	00:00:00+1559	24:00:00-1559	1 microsecond
<code>interval [fields] [(p)]</code>	16 bytes	time interval	-178000000 years	178000000 years	1 microsecond

Format Tipe Data Date dan Time

- Format penulisan tipe data waktu, bisa menggunakan format
- Timestamp : YYYY-MM-dd HH:mm:ss
- Date : YYYY-MM-dd
- Time : HH:mm:ss

Tipe Data Boolean

Tipe Data Boolean

- BOOLEAN adalah tipe data kebenaran, yang artinya datanya hanya ada dua jenis, benar atau salah
- Benar direpresentasikan dengan data TRUE, sedangkan salah direpresentasikan dengan data FALSE

Tipe Data Enum

Tipe Data Enum

- Saat membuat kolom, kadang ada jenis tipe data Text, namun isi datanya sudah fix, misal Jenis Kelamin, Kategori, dan sejenisnya
- Kasus seperti itu bisa menggunakan tipe data Enum
- Tipe data Enum harus dibuat terlebih dahulu, dan ditentukan Value yang diperbolehkan

Membuat Tipe Data Enum

- Untuk membuat tipe data enum, kita bisa menggunakan perintah :
- CREATE TYPE NAMA_ENUM AS ENUM ('VALUE1', 'VALUE2', 'VALUE3');

Tipe Data Lainnya

Dan Lain-Lain

- Sebenarnya masih banyak jenis tipe data yang lain yang didukung oleh PostgreSQL, namun itu bisa kita pelajari jika memang ada kebutuhan spesifik
- Seperti misal tipe data binary, json, xml dan lain-lain
- <https://www.postgresql.org/docs/current/datatype.html>

Table

Table

- Data biasanya disimpan di dalam tabel di PostgreSQL
- Tiap tabel biasanya menyimpan satu jenis data, misal ketika kita membuat aplikasi toko online, kita akan membuat tabel barang, tabel pelanggan, tabel penjual, dan lain-lain
- Sebelum kita bisa memasukkan data ke tabel, kita wajib terlebih dahulu membuat tabelnya terlebih dahulu
- Dan tiap tabel yang kita buat, wajib ditentukan kolom-kolom nya, dan tipe data tiap kolom nya
- Kita juga bisa mengubah tabel yang sudah terlanjur dibuat, seperti menambah kolom baru, mengubah kolom yang sudah ada, atau menghapus kolom

Melihat Table

- \dt
- select * from pg_tables where schemaname = 'public';



Membuat Table

```
2
3 ✓ ~ CREATE TABLE barang
4   (
5     kode    INT,
6     name   VARCHAR(100),
7     harga   INT,
8     jumlah  INT
9 );
10
11
```

Melihat Struktur Table

- \d nama_tabel

Mengubah Table

```
ALTER TABLE barang
    ADD COLUMN nama_column TEXT,
    DROP COLUMN nama,
    RENAME COLUMN nama TO nama_baru,
```

Null Value

- Null adalah nilai ketika kita tidak mengisi data ke dalam kolom
- Secara default, saat kita membuat kolom, kolom tersebut bisa bernilai NULL, jika kita tidak ingin menerima nilai NULL, kita bisa menambahkan NOT NULL ketika pembuatan kolom nya

Default Value

- Saat kita menyimpan data ke dalam tabel, lalu kita hanya menyimpan beberapa kolom (tidak semuanya), kolom yang tidak kita beri nilai secara default nilainya adalah NULL
- Jika kita ingin mengubah default value nya, kita bisa menambahkan perintah DEFAULT NILAI ketika pembuatan kolom nya
- Khusus tipe data DATETIME atau TIMESTAMP, jika kita ingin menggunakan default value dengan nilai waktu saat ini, kita bisa gunakan kata kunci CURRENT_TIMESTAMP

Membuat Ulang Table

```
TRUNCATE nama_tabel;
```

Menghapus Table

```
DROP TABLE nama_tabel;
```

Insert Data

Insert Data

- Sebelum kita meng memasukkan data kedalam tabel, tabel harus dibuat terlebih dahulu
- Kita bisa menyebutkan kolom mana yang ingin kita isi, jika kita tidak menyebutkan kolom nya, artinya kolom tersebut tidak akan kita isi, dan secara otomatis kolom yang tidak kita isi, nilainya akan NULL, kecuali memiliki DEFAULT VALUE
- Untuk memasukkan data kedalam tabel, kita bisa menggunakan perintah SQL yang bernama INSERT



Membuat Tabel Produk

```
✓ ~ CREATE TABLE products
~ (
    id          VARCHAR(10) NOT NULL,
    name        VARCHAR(100) NOT NULL,
    description TEXT,
    price       INT          NOT NULL,
    quantity    INT          NOT NULL DEFAULT 0,
    created_at  TIMESTAMP   NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```



Memasukkan Data

```
✓ INSERT INTO products(id, name, price, quantity)
  VALUES ('P0001', 'Mie Ayam Original', 15000, 100);

✓ ✓ INSERT INTO products(id, name, description, price, quantity)
  VALUES ('P0002', 'Mie Ayam Bakso Tahu', 'Mie Ayam Original + Bakso Tahu', 20000, 100);
```



Memasukkan Beberapa Data Sekaligus

```
/ √ INSERT INTO products(id, name, price, quantity)
  √ VALUES ('P0003', 'Mie Ayam Ceker', 20000, 100),
            ('P0004', 'Mie Ayam Spesial', 25000, 100),
            ('P0005', 'Mie Ayam Yamin', 15000, 100);
```

Select Data

Select Data

- Untuk mengambil data di tabel, kita bisa menggunakan SQL dengan kata kunci SELECT
- SELECT bisa digunakan untuk mengambil semua kolom yang ada di tabel, atau sebagian kolom saja
- Jika kita ingin mengambil semua kolom, kita bisa gunakan karakter * (bintang)
- Jika kita hanya ingin mengambil beberapa kolom saja, kita bisa sebutkan nama-nama kolom yang ingin kita ambil datanya



Mengambil Data

```
SELECT * FROM products;
```

```
SELECT id, name, price, quantity FROM products;
```

Primary Key

Primary Key

- Saat kita membuat tabel, idealnya tiap tabel memiliki Primary Key
- Primary key adalah sebuah kolom yang kita tunjuk sebagai id dari tabel tersebut
- Primary key adalah identitas untuk tiap baris data di dalam tabel
- Primary key harus unik, tidak boleh ada data dengan primary key yang sama
- Kita bisa menunjuk kolom yang akan kita jadikan primary key

Primary Key di Multiple Column

- Kita bisa membuat primary key dengan kombinasi beberapa kolom
- Namun disarankan untuk tetap menggunakan satu kolom ketika membuat primary key
- Kecuali ada kasus khusus, seperti membuat tabel yang berelasi MANY TO MANY (yang nanti akan kita bahas)



Menambah Primary Key Ketika Membuat Tabel

```
CREATE TABLE products
(
    id      VARCHAR(10) NOT NULL,
    name    VARCHAR(100) NOT NULL,
    description TEXT,
    price   INT          NOT NULL,
    quantity INT          NOT NULL DEFAULT 0,
    created_at TIMESTAMP   NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (id)
);
```

Menambah Primary Key di Tabel

```
✓ ▾ ALTER TABLE products  
    ADD PRIMARY KEY (id);
```

Where Clause

Where Clause

- Saat mengambil data menggunakan perintah SQL SELECT, kadang kita ingin melakukan pencarian data
- Misal, kita ingin mengambil data barang yang harganya 1jt, atau mengambil data barang yang quantity nya 0 (stok nya kosong)
- Hal ini bisa kita lakukan dengan WHERE clause setelah perintah SELECT



Mencari Data

```
SELECT id, name, price, quantity  
FROM products  
WHERE quantity = 0;
```



Update Data

Update Data

- Untuk mengubah data di tabel, kita bisa menggunakan perintah SQL UPDATE
- Saat menggunakan SQL UPDATE, kita harus memberi tahu data mana yang akan di update dengan WHERE clause
- Hati-hati ketika meng-update data di table, jika sampai WHERE clause nya salah, bisa-bisa kita malah meng-update seluruh data di tabel
- Untuk update, kita harus beritahu, kolom mana yang akan di update



Menambah Kolom Kategori

```
CREATE TYPE PRODUCT_CATEGORY AS ENUM ('Makanan', 'Minuman', 'Lain-Lain');

▽ ALTER TABLE products
    ADD COLUMN category PRODUCT_CATEGORY;
```



Mengubah Satu Kolom

```
▽ UPDATE products
  SET category = 'Makanan'
  WHERE id = 'P0001';
```

Mengubah Beberapa Kolom

```
✓ UPDATE products
  SET category      = 'Makanan',
      description = 'Mie Ayam + Ceker'
 WHERE id = 'P0003';
```

Mengubah Dengan Value di Kolom

```
✓ UPDATE products  
  SET price = price + 5000  
  WHERE id = 'P0004';
```

Delete Data

Delete Data

- Setelah kita tahu cara menambah, mengubah dan mengambil data di tabel, terakhir yang perlu kita ketahui adalah menghapus data di table
- Untuk menghapus data di table, kita bisa menggunakan perintah SQL DELETE
- Perintah SQL DELETE sama seperti UPDATE, kita perlu memberi tahu data mana yang akan dihapus dengan WHERE clause
- Dan hati-hati, jangan sampai salah menentukan WHERE clause, karena jika salah, bisa-bisa kita akan menghapus seluruh data di table

Menghapus Data

```
› ~ DELETE  
      FROM products  
      WHERE id = 'P0009';
```

Alias

Alias

- PostgreSQL memiliki fitur untuk melakukan alias untuk kolom dan tabel
- Alias berguna jika kita ingin mengubah nama kolom atau nama tabel ketika melakukan SELECT data
- Mungkin saat ini alias untuk tabel tidak terlalu terlihat gunanya, tapi nanti ketika kita telah mempelajari tentang JOIN, maka fitur alias untuk tabel sangat berguna sekali



Alias untuk Kolom

```
▽ SELECT id          as Kode,  
        price        as Harga,  
        description as Deskripsi  
    FROM products;
```



Alias untuk Tabel

```
▼ SELECT p.id          as Kode,  
      p.price        as Harga,  
      p.description as Deskripsi  
FROM products as p;
```

Where Operator

Where Operator

- Sebelumnya di materi where clause kita sudah menggunakan operator = (sama dengan)
- Sebenarnya sangat banyak sekali operator yang bisa kita gunakan ketika menggunakan where clause
- Sekarang kita akan bahas satu per satu

Operator Perbandingan

Operator	Keterangan
=	Sama dengan
<> atau !=	Tidak sama dengan
<	Kurang dari
<=	Kurang dari atau sama dengan
>	Lebih dari
>=	Lebih dari atau sama dengan

Mencari Data dengan Operator Perbandingan

```
▽ SELECT *  
    FROM products  
   WHERE price > 15000;
```

AND dan OR Operator

- Kadang kita ingin mencari data dengan beberapa gabungan kondisi, kita bisa menggunakan operator AND dan OR
- AND dan OR digunakan untuk menggabungkan beberapa dua operator

Hasil Operator AND

Hasil Operator 1	Operator	Hasil Operator 2	Hasil Akhir
Benar	AND	Benar	Benar
Salah	AND	Benar	Salah
Benar	AND	Salah	Salah
Salah	AND	Salah	Salah



Mencari Data dengan Operator AND

```
• ▾ SELECT *  
  FROM products  
 WHERE price > 15000  
   AND category = 'Makanan' ;
```

Hasil Operator OR

Hasil Operator 1	Operator	Hasil Operator 2	Hasil Akhir
Benar	OR	Benar	Benar
Salah	OR	Benar	Benar
Benar	OR	Salah	Benar
Salah	OR	Salah	Salah

Mencari Data dengan Operator OR

```
SELECT *  
FROM products  
WHERE price > 15000  
OR category = 'Makanan';
```



Prioritas dengan Kurung ()

```
SELECT *  
FROM products  
WHERE (quantity > 100 OR category = 'Makanan')  
AND price > 10000;
```

LIKE Operator

- LIKE operator adalah operator yang bisa kita gunakan untuk mencari sebagian data dalam String
- Ini cocok sekali ketika kita hanya ingin mencari sebagian kata dalam String
- Namun perlu diingat, operasi LIKE itu sangat lambat, oleh karena itu, tidak disarankan jika datanya sudah terlalu besar di tabel
- Operasi LIKE case sensitive, jadi huruf besar dan kecil juga berpengaruh, jika kita ingin tidak case sensitive, bisa menggunakan ILIKE

Hasil Operator LIKE

LIKE Operator	Hasil
LIKE 'b%'	String dengan awalan b
LIKE '%a'	String dengan akhiran b
LIKE '%eko%'	String berisi eko
NOT LIKE	Tidak LIKE

Mencari Menggunakan LIKE Operator

```
SELECT *  
FROM products  
WHERE name ILIKE '%mie%';
```

NULL Operator

- Untuk mencari data yang berisi NULL, kita tidak bisa menggunakan operator perbandingan = NULL.
- Ada operator khusus untuk mencari data NULL, yaitu menggunakan NULL operator
- IS NULL, artinya mencari yang NULL
- IS NOT NULL, artinya mencari yang tidak NULL

Mencari Menggunakan NULL Operator

```
SELECT *  
FROM products  
WHERE description IS NULL;
```

BETWEEN Operator

- Kadang kita ingin mencari data yang \geq dan \leq secara sekaligus
- Misal kita ingin mencari products yang harganya antara 10000 sampai 20000
- Untuk melakukan ini, kita bisa menggunakan WHERE $price \geq 10000$ AND $price \leq 20000$
- Namun ada operator BETWEEN yang bisa kita gunakan agar lebih sederhana
- Untuk kebalikannya, kita bisa gunakan NOT BETWEEN

Mencari Menggunakan BETWEEN Operator

```
▽ SELECT *  
    FROM products  
   WHERE price BETWEEN 10000 AND 20000;
```

IN Operator

- Operator IN adalah operator untuk melakukan pencarian sebuah kolom dengan beberapa nilai.
- Misal kita ingin mencari products dengan category Makanan atau Minuman, maka kita bisa menggunakan operator IN
- Untuk kebalikannya, kita bisa menggunakan NOT IN

Mencari Menggunakan IN Operator

```
SELECT *  
FROM products  
WHERE category IN ('Makanan', 'Minuman');
```

Order By Clause

Order By Clause

- Untuk mengurutkan data ketika kita menggunakan perintah SQL SELECT, kita bisa menambahkan ORDER BY clause
- ORDER BY clause digunakan untuk mengurutkan data berdasarkan kolom yang dipilih, dan jenis urutan (ASC atau DESC)
- Kita juga bisa mengurutkan tidak hanya terhadap satu kolom, tapi beberapa kolom

Mengurutkan Data

```
SELECT *  
FROM products  
ORDER BY price ASC, id DESC;
```

Limit Clause

Limit Clause

- Mengambil seluruh data di tabel bukanlah pilihan bijak, apalagi jika datanya sudah banyak sekali
- Kita bisa membatasi jumlah data yang diambil dalam SQL SELECT dengan LIMIT clause
- Selain membatasi jumlah data, kita juga bisa meng-skip sejumlah data yang tidak ingin kita lihat
- LIMIT biasanya digunakan saat melakukan paging di aplikasi kita, dengan kombinasi OFFSET

Membatasi Hasil Query

```
SELECT *  
FROM products  
WHERE price > 0  
ORDER BY price ASC  
LIMIT 2;
```

Skip Hasil Query

```
SELECT *  
FROM products  
WHERE price > 0  
ORDER BY price ASC  
LIMIT 2 OFFSET 2;
```

Select Distinct Data

Select Distinct Data

- Saat melakukan query dengan SELECT, kadang kita mendapatkan data yang duplikat
- Misal kita ingin melihat semua kategori di tabel products, maka otomatis hasil query SELECT akan duplikat, karena banyak sekali produk dengan kategori yang sama
- Jika kita ingin menghilangkan data-data duplikat tersebut , kita bisa menggunakan SELECT dengan tambahan DISTINCT sebelum nama kolom nya

Menghilangkan Data Duplikat

```
SELECT DISTINCT category  
FROM products;
```

Numeric Function

Numeric Function

- PostgreSQL memiliki banyak sekali fitur untuk manipulasi data angka
- Hal ini memudahkan kita untuk memanipulasi data angka
- Secara garis besar, fitur ini dibagi menjadi dua, Arithmetic Operator dan Mathematical Function



Arithmetic Operator

Operator	Description
+	Addition - Adds values on either side of the operator
-	Subtraction - Subtracts right hand operand from left hand operand
*	Multiplication - Multiplies values on either side of the operator
/	Division - Divides left hand operand by right hand operand
%	Modulus - Divides left hand operand by right hand operand and returns remainder



Menggunakan Arithmetic Operator

```
SELECT 10 + 10 as hasil;
```

```
▼ SELECT id, price / 1000 as price_in_k  
      FROM products;
```

Mathematical Function

- Selain arithmetic operator, ada juga mathematical function
- Ini adalah kumpulan function yang terdapat di PostgreSQL yang bisa kita gunakan sebagai fungsi-fungsi matematika
- Ada banyak sekali, dan tidak bisa kita bahas semua
- <https://www.postgresql.org/docs/15/functions-math.html>



Menggunakan Mathematical Function

```
SELECT PI();
```

```
SELECT POWER(10, 2);
```

```
SELECT COS(10), SIN(10), TAN(10);
```

Auto Increment

Auto Increment

- Kadang kita butuh angka yang berurut untuk membuat primary key, misal 1, 2, 3, dan seterusnya.
- Untuk melakukan hal ini secara manual bukanlah hal bijak, apalagi jika aplikasi yang kita buat diakses oleh banyak orang secara bersamaan
- PostgreSQL memiliki tipe data Number bernama SERIAL, fitur ini bisa kita gunakan untuk membuat function yang akan otomatis mengembalikan nilai yang selalu naik ketika dipanggil
- Dengan menggunakan SERIAL, kita tidak perlu lalu memasukkan data primary key secara manual, secara otomatis nilai primary key akan naik



Membuat Tabel dengan Auto Increment

```
✓ CREATE TABLE admin
✓ (
    id          SERIAL NOT NULL,
    first_name VARCHAR(100),
    last_name  VARCHAR(100),
    PRIMARY KEY (id)
);
```



Memasukkan Data Tanpa Id

```
✓ INSERT INTO admin(first_name, last_name)
✓ VALUES ('Eko', 'Khannedy'),
          ('Budi', 'Nugraha'),
          ('Joko', 'Morro');
```



Melihat Id Terakhir

```
✓ SELECT currval(pg_get_serial_sequence('admin', 'id'));  
  
SELECT currval('admin_id_seq');
```

Sequence

Sequence

- Saat kita menggunakan tipe data SERIAL, sebenarnya dibelakangnya, PostgreSQL menggunakan Sequence
- Sequence adalah fitur dimana kita bisa membuat function auto increment
- Saat menggunakan tipe data SERIAL pada Primary Key, secara otomatis PostgreSQL akan membuat Sequence, dan memanggil function sequence nya sebagai default value untuk Primary Key nya



Kode : Membuat Sequence

```
-- Membuat Sequence
CREATE SEQUENCE contoh_sequence;

-- Memanggil sequence, otomatis increment
SELECT nextval('contoh_sequence');

-- Mengambil nilai terakhir sequence
SELECT currval('contoh_sequence');
```



Kode : Sequence dari Serial

```
CREATE SEQUENCE admin_id_seq;  
    
▼ CREATE TABLE admin  
  ▼ (  
    id              INT NOT NULL DEFAULT nextval('admin_id_seq'),  
    first_name  VARCHAR(100),  
    last_name   VARCHAR(100),  
    PRIMARY KEY (id)  
  );
```

Melihat Semua Sequence

\ds

String Function

String Function

- Sama seperti number, di PostgreSQL juga banyak menyediakan function untuk tipe data String
- Ada banyak sekali function-function yang bisa kita gunakan
- <https://www.postgresql.org/docs/15/functions-string.html>



Menggunakan String Function

```
▽ SELECT id, LOWER(name), LENGTH(name), LOWER(description)  
      FROM products;
```

Date and Time Functions

Date dan Time Function

- PostgreSQL juga menyediakan banyak sekali function yang bisa kita gunakan untuk mengolah data tipe Date dan Time
- <https://www.postgresql.org/docs/15/functions-datetime.html>



Menambah Kolom Timestamp

```
✓ SELECT id, EXTRACT(YEAR FROM created_at), EXTRACT(MONTH FROM created_at)  
FROM products;
```

Flow Control Function

Flow Control Function

- PostgreSQL memiliki fitur flow control function
- Ini mirip IF ELSE di bahasa pemrograman
- Tapi ingat, fitur ini tidak se kompleks yang dimiliki bahasa pemrograman
- <https://www.postgresql.org/docs/current/functions-conditional.html>

Menggunakan Control Flow CASE

```
SELECT id,  
       CASE category  
           WHEN 'Makanan' THEN 'Enak'  
           WHEN 'Minuman' THEN 'Seger'  
           ELSE 'Apa itu?'  
       END as category  
FROM products;
```



Menggunakan Operator

```
✓ ~ SELECT id,  
      price,  
      CASE  
          WHEN price <= 15000 THEN 'Murah'  
          WHEN price <= 20000 THEN 'Mahal'  
          ELSE 'Mahal Banget'  
      END as murah  
FROM products;
```



Menggunakan Control Flow Check Null

```
SELECT id,  
       name,  
       CASE  
           WHEN description IS NULL THEN 'Kosong'  
           ELSE description  
       END as description  
FROM products;
```

Aggregate Function

Aggregate Function

- PostgreSQL mendukung function-function untuk melakukan aggregate
- Misal, kita ingin melihat harga paling mahal di tabel product, atau harga termurah, atau rata-rata harga produk, atau total jumlah data di tabel, dan lain-lain
- <https://www.postgresql.org/docs/current/functions-aggregate.html>



Menggunakan Aggregate Function

```
SELECT COUNT(id) AS "Total Product" FROM products;
```

```
SELECT AVG(price) AS "Rata-Rata Harga" FROM products;
```

```
SELECT MAX(price) AS "Harga Termahal" FROM products;
```

```
SELECT MIN(price) AS "Harga Termurah" FROM products;
```

Grouping

GROUP BY

- Kadang saat melakukan aggregate, kita ingin datanya di grouping berdasarkan kriteria tertentu
- Misal kita ingin melihat rata-rata harga product, tapi ingin per category
- Atau kita ingin melihat total semua product, tapi per category
- Hal ini bisa dilakukan di PostgreSQL dengan menggunakan GROUP BY clause
- GROUP BY clause ini hanya bisa digunakan jika kita menggunakan aggregate function



Menggunakan GROUP BY

```
▽ SELECT category,  
       COUNT(id) as "Total Product"  
     FROM products  
   GROUP BY category;
```

HAVING Clause

- Kadang kita ingin melakukan filter terhadap data yang sudah kita grouping
- Misal kita ingin menampilkan rata-rata harga per kategori, tapi yang harganya diatas 10.000 misalnya
- Jika menggunakan WHERE di SELECT, hal ini tidak bisa dilakukan
- Untuk memfilter hasil aggregate function, kita harus menggunakan HAVING clause

Menggunakan HAVING Clause

```
▽ SELECT category,  
      COUNT(id) as total  
  FROM products  
 GROUP BY category  
 HAVING COUNT(id) > 1;
```

Constraint

Constraint

- Di PostgreSQL, kita bisa menambahkan constraint untuk menjaga data di tabel tetap baik
- Constraint sangat bagus ditambahkan untuk menjaga terjadi validasi yang salah di program kita, sehingga data yang masuk ke database tetap akan terjaga

Unique Constraint

- Unique constraint adalah constraint yang memastikan bahwa data kita tetap unique
- Jika kita mencoba memasukkan data yang duplikat, maka PostgreSQL akan menolak data tersebut



Membuat Table dengan Unique Constraint

```
CREATE TABLE customer
(
    id          SERIAL      NOT NULL,
    email       VARCHAR(100) NOT NULL,
    first_name  VARCHAR(100) NOT NULL,
    last_name   VARCHAR(100),
    PRIMARY KEY (id),
    CONSTRAINT unique_email UNIQUE (email)
);
```



Menambah/Menghapus Unique Constraint

```
✗ ALTER TABLE customer
    DROP CONSTRAINT unique_email;

✓ ✗ ALTER TABLE customer
    ADD CONSTRAINT unique_email UNIQUE (email);
```

Check Constraint

- Check constraint adalah constraint yang bisa kita tambahkan kondisi pengecekannya
- Ini cocok untuk mengecek data sebelum dimasukkan ke dalam database
- Misal kita ingin memastikan bahwa harga harus diatas 1000 misal
- Maka kita bisa menggunakan check constraint



Membuat Table dengan Check Constraint

```
CREATE TABLE products
(
    id          VARCHAR(100) NOT NULL,
    name        VARCHAR(100) NOT NULL,
    description TEXT,
    price       INT          NOT NULL,
    quantity    INT          NOT NULL,
    created_at  TIMESTAMP    NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (id),
    CONSTRAINT price_check CHECK ( price >= 1000 )
);
```



Menambah/Menghapus Check Constraint

```
✓ ALTER TABLE products
    ADD CONSTRAINT price_check CHECK ( price >= 1000 );

✓ ALTER TABLE products
    DROP CONSTRAINT price_check;
```

Index

Index

- Secara default, PostgreSQL akan menyimpan data di dalam disk seperti tabel biasanya
- Hal ini menyebabkan, ketika kita mencari data, maka PostgreSQL akan melakukan pencarian dari baris pertama sampai terakhir, yang artinya semakin banyak datanya, maka akan semakin lambat proses pencarian datanya
- Kita bisa ubah cara PostgreSQL menyimpan data pada kolom, agar mudah dicari, yaitu menggunakan Index
- Saat kita membuat index, PostgreSQL akan menyimpan data dalam struktur data B-Tree :
<https://en.wikipedia.org/wiki/B-tree>
- Tidak hanya akan mempermudah kita saat melakukan pencarian, index juga akan mempermudah kita ketika melakukan pengurutan menggunakan ORDER BY

Cara Kerja Index

- Kita bisa membuat lebih dari satu index di table, dan setiap kita membuat index, kita bisa membuat index untuk beberapa kolom sekaligus
- Misal kita membuat index
- (col1, col2, col3)
- Artinya kita punya kemampuan untuk mencari lebih menggunakan index untuk kombinasi query di (col1), (col1, col2) dan (col1, col2, col3)

Efek Samping Membuat Index

- Index mungkin akan mempercepat untuk proses pencarian dan query data
- Namun, saat kita membuat index, artinya PostgreSQL akan melakukan proses update data di index tiap kali kita menambah, mengubah atau menghapus data di table
- Artinya Index membuat proses pencarian dan query lebih cepat, tapi memperlambat proses manipulasi data (insert, update dan delete)
- Oleh karena itu, kita harus bijak saat membuat index

Tidak Perlu Index

- Saat kita membuat PRIMARY KEY dan UNIQUE constraint, kita tidak perlu menambahkan lagi index
- Hal ini dikarenakan PostgreSQL secara otomatis akan menambahkan index pada kolom PRIMARY KEY dan UNIQUE constraint



Membuat Table

```
✓ √ CREATE TABLE sellers
  (
    id      SERIAL      NOT NULL,
    name   VARCHAR(100) NOT NULL,
    email  VARCHAR(100) NOT NULL,
    PRIMARY KEY (id),
    CONSTRAINT email_unique UNIQUE (email)
);
```



Menambah/Menghapus Index

- CREATE INDEX sellers_name_index ON sellers (name);
- DROP INDEX sellers_name_index;

Full-Text Search

Masalah dengan LIKE operator

- Kadang kita ingin mencari sebuah kata dalam tabel, dan biasanya kita akan menggunakan LIKE operator
- Operasi yang dilakukan LIKE operator adalah dengan cara mencari seluruh data di tabel dari baris pertama sampai terakhir, hal ini membuat operasi LIKE sangat lambat
- Menambah index di tabel juga tidak akan membantu, karen LIKE operator tidak menggunakan index
- PostgreSQL menyediakan fitur Full Text Search jika ada kasus kita ingin melakukan hal ini

Full-Text Search

- Full-Text Search memungkinkan kita bisa mencari sebagian kata di kolom dengan tipe data String
- Ini sangat cocok ketika pada kasus kita memang membutuhkan pencarian yang tidak hanya sekedar operasi = (equals, sama dengan)
- <https://www.postgresql.org/docs/current/textsearch-intro.html>
- Di PostgreSQL, Full-Text Search menggunakan function to_tsvector(text) dan to_tsquery(query)
- Bahkan kita bisa menggunakan function tersebut tanpa membuat index, namun performanya akan sama saja dengan LIKE, lambat karena harus di cek satu-satu
- Operator Full-Text Search menggunakan @@, bukan =



Mencari Tanpa Index

```
SELECT *  
FROM products  
WHERE to_tsvector(name) @@ to_tsquery('mie');
```

Full-Text Search Index

- Untuk membuat index Full-Text Search kita bisa menggunakan perintah yang sama dengan index biasa, tapi harus disebutkan detail dari jenis index Full-Text Search nya



Membuat Index Full-Text Search

```
-- Get available languages
SELECT cfgname FROM pg_ts_config;

CREATE INDEX products_name_search ON products USING GIN (to_tsvector('indonesian', name));

CREATE INDEX products_description_search ON products USING GIN (to_tsvector('indonesian', description));

DROP INDEX products_name_search;

DROP INDEX products_description_search;
```



Mencari Menggunakan Full-Text Search

```
▽ SELECT *
  FROM products
 WHERE name @@ to_tsquery('mie');

▽ SELECT *
  FROM products
 WHERE description @@ to_tsquery('mie');
```

Query Operator

- `to_tsquery()` mendukung banyak operator
- `&` untuk AND
- `|` untuk OR
- `!` untuk NOT
- `""` untuk semua data



Mencari dengan Operator

```
▽ SELECT *
  FROM products
  WHERE name @@ to_tsquery('original | bakso');
```

```
▽ SELECT *
  FROM products
  WHERE name @@ to_tsquery('bakso & tahu');
```

```
▽ ▽ SELECT *
  FROM products
  WHERE name @@ to_tsquery(' !bakso');
```

Tipe Data TSVECTOR

- Kita juga bisa secara otomatis membuat kolom dengan tipe data TSVECTOR
- Secara otomatis kolom tersebut berisi text yang memiliki index Full-Text Search

Table Relationship

Table Relationship

- Dalam Relational DBMS, salah satu fitur andalan nya adalah table relationship. Yaitu relasi antar tabel
- Kita bisa melakukan relasi dari satu tabel ke tabel lain.
- Dalam kehidupan nyata pun pasti kita akan sering membuat relasi antar tabel
- Misal, saat kita membuat aplikasi penjualan, di laporan penjualan pasti ada data barang. Jika di tabel artinya tabel penjualan akan berelasi dengan tabel barang
- Misal dalam aplikasi kampus, tabel mahasiswa akan berelasi dengan tabel mata kuliah, dan tabel dosen
- Dan lain-lain

Foreign Key

- Saat membuat relasi tabel, biasanya kita akan membuat sebuah kolom sebagai referensi ke tabel lainnya
- Misal saat kita membuat tabel penjualan, di dalam tabel penjualan, kita akan menambahkan kolom id_produk sebagai referensi ke tabel produk, yang berisi primary key di tabel produk
- Kolom referensi ini di PostgreSQL dinamakan Foreign Key
- Kita bisa menambah satu atau lebih foreign key ke dalam sebuah tabel
- Membuat foreign key sama seperti membuat kolom biasanya, hanya saja kita perlu memberi tahu PostgreSQL bahwa itu adalah foreign key ke tabel lain



Membuat Table dengan Foreign Key

```
✓ CREATE TABLE wishlist
✓ (
    id          SERIAL      NOT NULL,
    id_product  VARCHAR(10) NOT NULL,
    description TEXT,
    PRIMARY KEY (id),
    CONSTRAINT fk_wishlist_product FOREIGN KEY (id_product) REFERENCES products (id)
);
```



Menambah/Menghapus Foreign Key

```
▽ ALTER TABLE wishlist
    DROP CONSTRAINT fk_wishlist_product;

▽ ALTER TABLE wishlist
    ADD CONSTRAINT fk_wishlist_product FOREIGN KEY (id_product) REFERENCES products (id);
```

Keuntungan Menggunakan Foreign Key

- Foreign key memastikan bahwa data yang kita masukkan ke kolom tersebut harus tersedia di tabel reference nya
- Selain itu saat kita menghapus data di tabel reference, PostgreSQL akan mengecek apakah id nya digunakan di foreign key di tabel lain, jika digunakan, maka secara otomatis PostgreSQL akan menolak proses delete data di tabel reference tersebut

Ketika Menghapus Data Berelasi

- Seperti yang sebelumnya dibahas, ketika kita menghapus data yang berelasi, maka secara otomatis PostgreSQL akan menolak operasi delete tersebut
- Kita bisa mengubah fitur ini jika kita mau, ada banyak hal yang bisa dilakukan ketika data berelasi dihapus, defaultnya memang akan ditolak (RESTRICT)



Behavior Foreign Key

Behavior	ON DELETE	ON UPDATE
RESTRICT	Ditolak	Ditolak
CASCADE	Data akan dihapus	Data akan ikut diubah
NO ACTION	Data dibiarkan	Data dibiarkan
SET NULL	Diubah jadi NULL	Diubah jadi NULL
SET DEFAULT	Diubah jadi Default Value	Diubah jadi Default Value



Mengubah Behavior Menghapus Relasi

```
ALTER TABLE wishlist
ADD CONSTRAINT fk_wishlist_product FOREIGN KEY (id_product) REFERENCES products (id)
ON DELETE CASCADE ON UPDATE CASCADE;
```

Join

Join

- PostgreSQL mendukung query SELECT langsung ke beberapa tabel secara sekaligus
- Namun untuk melakukan itu, kita perlu melakukan JOIN di SQL SELECT yang kita buat
- Untuk melakukan JOIN, kita perlu menentukan tabel mana yang merupakan referensi ke tabel lain
- Join cocok sekali dengan foreign key, walaupun di PostgreSQL tidak ada aturan kalau JOIN harus ada foreign key
- Join di PostgreSQL bisa dilakukan untuk lebih dari beberapa tabel
- Tapi ingat, semakin banyak JOIN, maka proses query akan semakin berat dan lambat, jadi harap bijak ketika melakukan JOIN
- Idealnya kita melakukan JOIN jangan lebih dari 5 tabel, karena itu bisa berdampak ke performa query yang lambat



Melakukan JOIN Table

```
▽ SELECT *  
  FROM wishlist  
        JOIN products ON products.id = wishlist.id_product;
```

```
↙ ▽ SELECT products.id, products.name, wishlist.description  
  FROM wishlist  
        JOIN products ON products.id = wishlist.id_product;
```



Membuat Relasi ke Table Customers

```
▽ ALTER TABLE wishlist
    ADD COLUMN id_customer INT;

▽ ALTER TABLE wishlist
    ADD CONSTRAINT fk_wishlist_customer FOREIGN KEY (id_customer) REFERENCES customer (id);
```



Melakukan JOIN Multiple Table

```
• ▾ SELECT customer.email, products.id, products.name, wishlist.description  
  FROM wishlist  
        JOIN products ON wishlist.id_product = products.id  
        JOIN customer ON wishlist.id_customer = customer.id;
```

One to One Relationship

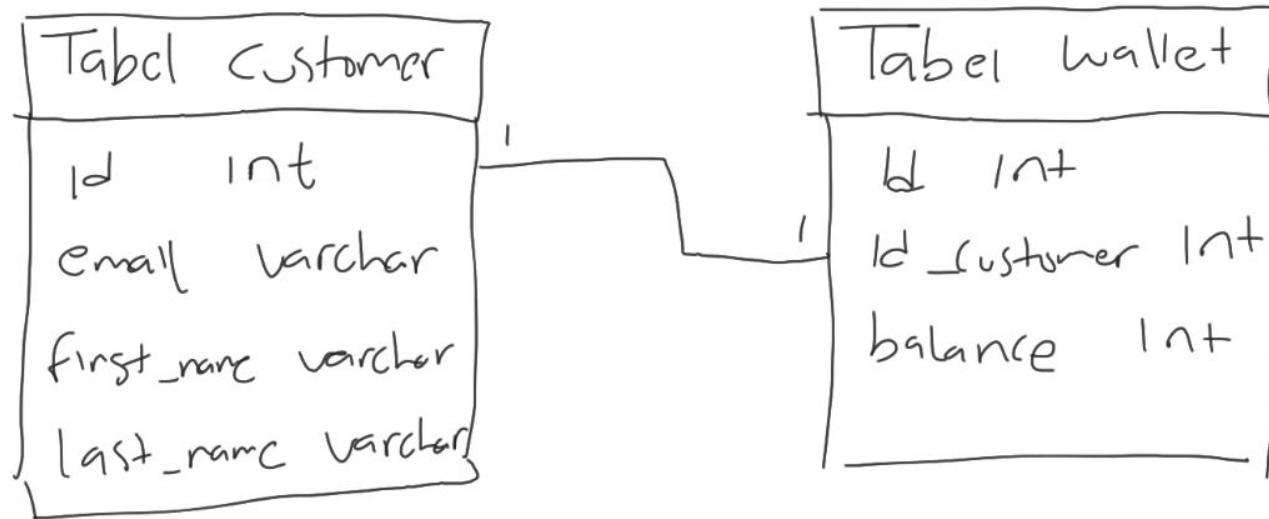
Jenis-Jenis Relasi Tabel

- Sekarang kita sudah tau untuk melakukan relasi antar tabel, kita bisa menggunakan FOREIGN KEY
- Dan untuk melakukan SELECT beberapa tabel, kita bisa menggunakan JOIN
- Dalam konsep relasi, ada banyak jenis-jenis relasi antar tabel
- Sekarang kita akan bahas dari yang pertama yaitu One to One relationship

One to One Relationship

- One to One relationship adalah relasi antar tabel yang paling sederhana
- Artinya tiap data di sebuah tabel hanya boleh berelasi ke maksimal 1 data di tabel lainnya
- Tidak boleh ada relasi lebih dari 1 data
- Contoh misal, kita membuat aplikasi toko online yang terdapat fitur wallet, dan 1 customer, cuma boleh punya 1 wallet

Diagram One to One Relationship



Membuat One to One Relationship

- Cara membuat One to One relationship cukup mudah
- Kita bisa membuat kolom foreign key, lalu set kolom tersebut menggunakan UNIQUE KEY, hal ini dapat mencegah terjadi data di kolom tersebut agar tidak duplikat
- Atau cara lainnya, kita bisa membuat tabel dengan primary key yang sama, sehingga tidak butuh lagi kolom untuk FOREIGN KEY



Membuat Table Wallet

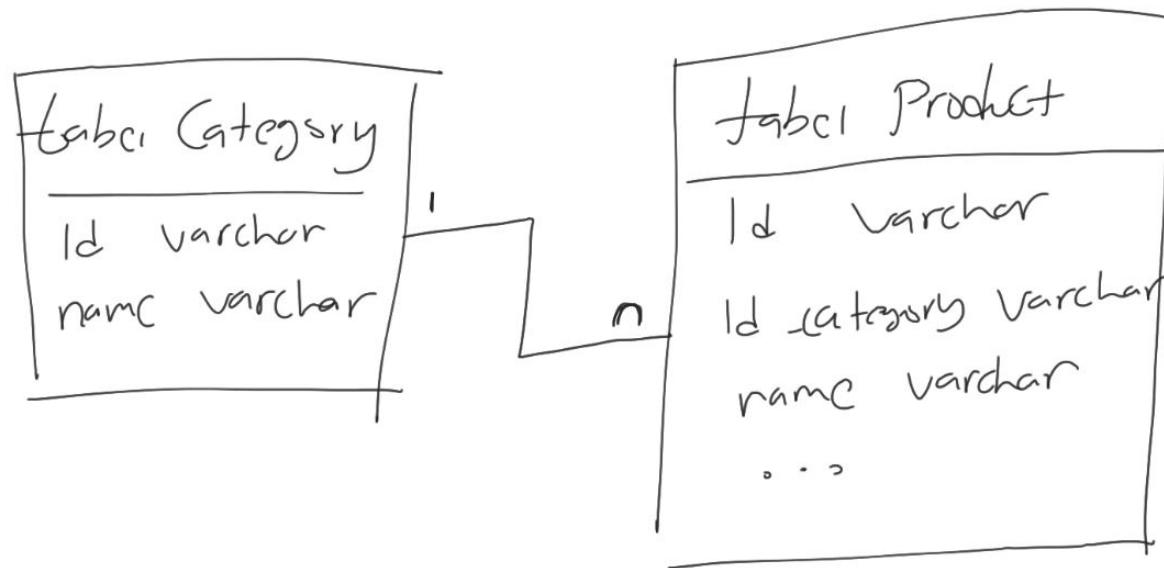
```
CREATE TABLE wallet
(
    id          SERIAL NOT NULL,
    id_customer INT      NOT NULL,
    balance     INT      NOT NULL DEFAULT 0,
    PRIMARY KEY (id),
    CONSTRAINT wallet_customer_unique UNIQUE (id_customer),
    CONSTRAINT fk_wallet_customer FOREIGN KEY (id_customer) REFERENCES customer (id)
);
```

One to Many Relationship

One to Many Relationship

- One to many relationship adalah relasi antar tabel dimana satu data bisa digunakan lebih dari satu kali di tabel relasinya
- Berbeda dengan one to one yang cuma bisa digunakan maksimal 1 kali di tabel relasinya, one to many tidak ada batasan berapa banyak data digunakan
- Contoh relasi antar tabel categories dan products, dimana satu category bisa digunakan oleh lebih dari satu product, yang artinya relasinya nya one category to many products
- Pembuatan relasi one to many sebenarnya sama dengan one to one, yang membedakan adalah, kita tidak perlu menggunakan UNIQUE KEY, karena datanya memang bisa berkali-kali ditambahkan di tabel relasi nya

Diagram One to Many Relationship





Membuat Table Category

```
▽ CREATE TABLE categories
▽ (
    id    VARCHAR(10) NOT NULL,
    name  VARCHAR(100) NOT NULL,
    PRIMARY KEY (id)
);
```



Mengubah Tabel Product

```
▽ ALTER TABLE products
    DROP COLUMN category;

▽ ALTER TABLE products
    ADD COLUMN id_category VARCHAR(10);

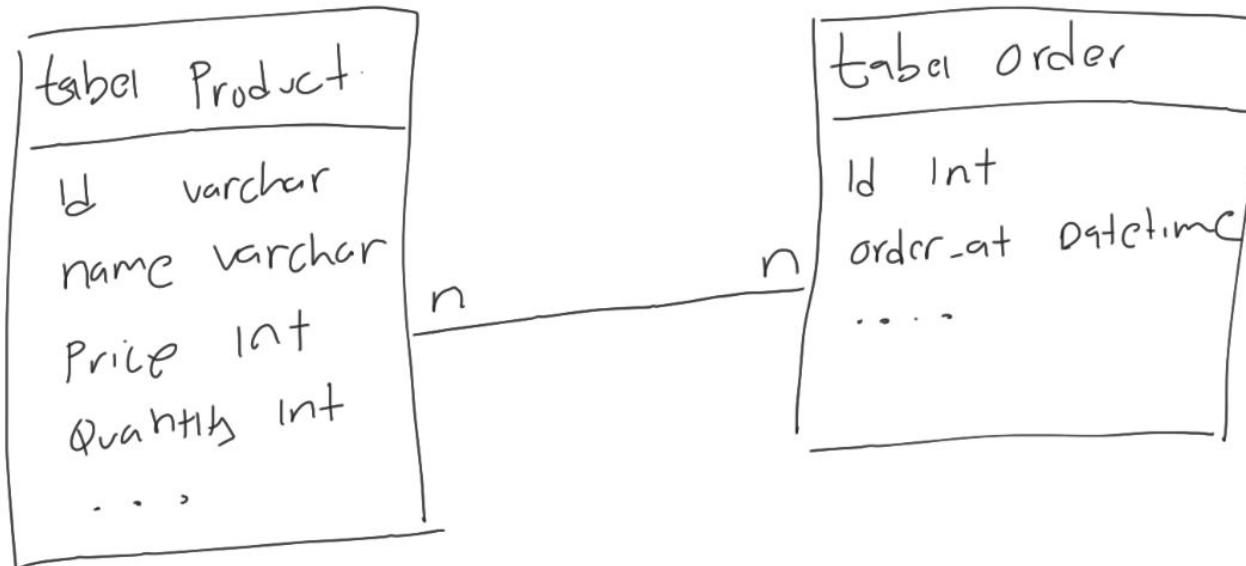
▽ ALTER TABLE products
    ADD CONSTRAINT fk_product_category FOREIGN KEY (id_category) REFERENCES categories (id);
```

Many to Many Relationship

Many to Many Relationship

- Many to Many adalah table relationship yang paling kompleks, dan kadang membingungkan untuk pemula
- Many to Many adalah relasi dimana ada relasi antara 2 tabel dimana table pertama bisa punya banyak relasi di table kedua, dan table kedua pun punya banyak relasi di table pertama
- Ini memang sedikit membingungkan, bagaimana caranya bisa relasi kebanyakan secara bolak balik, sedangkan di table kita cuma punya 1 kolom?
- Contoh relasi many to many adalah relasi antara produk dan penjualan, dimana setiap produk bisa dijual berkali kali, dan setiap penjualan bisa untuk lebih dari satu produk

Diagram Many to Many Relationship



Bagaimana Implementasi Many to Many?

- Sekarang pertanyaannya, bagaimana implementasi many to many?
- Apakah kita harus menambahkan id_order di table products? atau id_product di table orders?

Id Product di Table Order

- Jika kita menambahkan id_product di table orders, artinya sekarang sudah benar, bahwa 1 product bisa dijual berkali-kali
- Namun masalahnya adalah, berarti 1 order hanya bisa membeli 1 product, karena cuma ada 1 kolom untuk id_product
- Oke kalo gitu kita tambahkan id_product1, id_product2, dan seterusnya. Solusi ini bisa dilakukan, tapi tidak baik, artinya akan selalu ada maksimal barang yang bisa kita beli dalam satu order

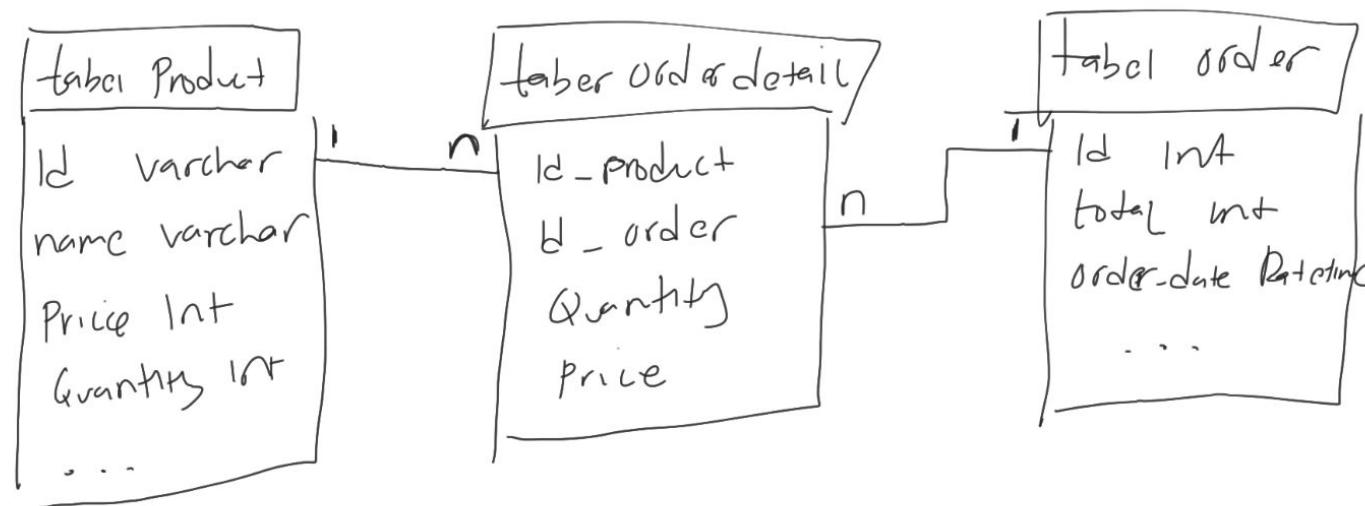
Id Order di Table Product

- Jika kita tambahkan id_order di table products, artinya sekarang 1 order bisa membeli lebih dari 1 product, oke sudah benar
- Tapi sayangnya masalahnya terbalik sekarang, 1 product cuma bisa dijual satu kali, tidak bisa dijual berkali-kali, karena kolom id_order nya cuma 1
- Kalupun kita tambah id_order1, id_order2 dan seterusnya di table product, tetap ada batasan maksimal nya
- Lantai bagaimana solusinya untuk relasi many to many?

Membuat Table Relasi

- Solusi yang biasa dilakukan jika terjadi relasi many to many adalah, biasanya kita akan menambah 1 tabel ditengahnya
- Tabel ini bertugas sebagai jembatan untuk menggabungkan relasi many to many
- Isi table ini akan ada id dari table pertama dan table kedua, dalam kasus ini adalah id_product dan id_order
- Dengan demikian, kita bisa menambahkan beberapa data ke dalam tabel relasi ini, sehingga berarti satu product bisa dijual beberapa kali di dalam table order, dan satu order bisa membeli lebih dari satu product

Diagram Many to Many Relationship





Membuat Table Order

```
CREATE TABLE orders
(
    id      SERIAL      NOT NULL,
    total    INT         NOT NULL,
    order_date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (id)
);
```



Membuat Table Order Detail

```
✓ CREATE TABLE orders_detail
✓ (
    id_product VARCHAR(10) NOT NULL,
    id_order    INT        NOT NULL,
    price       INT        NOT NULL,
    quantity    INT        NOT NULL,
    PRIMARY KEY (id_product, id_order)
);
```



Membuat Foreign Key

```
▽ ALTER TABLE orders_detail
    ADD CONSTRAINT fk_orders_detail_product FOREIGN KEY (id_product) REFERENCES products (id);

▽ ALTER TABLE orders_detail
    ADD CONSTRAINT fk_orders_detail_order FOREIGN KEY (id_order) REFERENCES orders (id);
```

Melihat Data Order, Detail dan Product-nya

```
SELECT *  
FROM orders  
    JOIN orders_detail ON orders_detail.id_order = orders.id  
    JOIN products ON orders_detail.id_product = products.id;
```

Jenis-Jenis Join

Jenis-Jenis Join

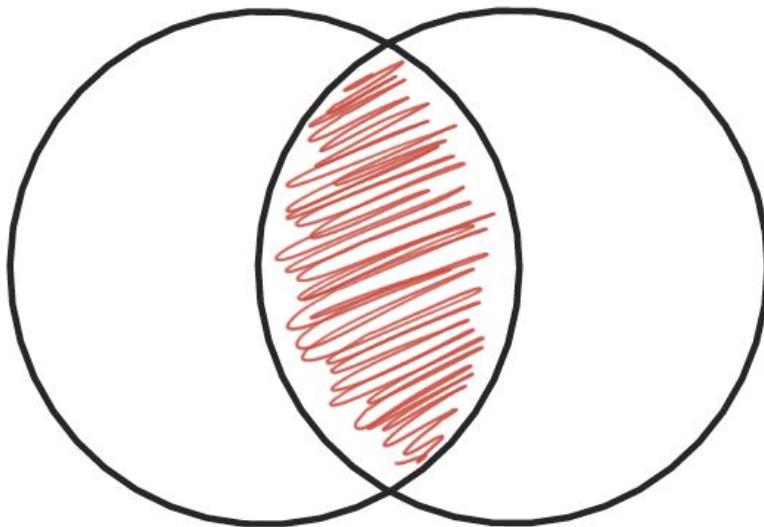
Sebelumnya kita sudah bahas tentang JOIN table, tapi sebenarnya ada banyak sekali jenis-jenis JOIN table di PostgreSQL, diantaranya :

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN

Inner Join

- INNER JOIN adalah mekanisme JOIN, dimana terdapat relasi antara tabel pertama dan tabel kedua
- Jika ada data di tabel pertama yang tidak memiliki relasi di table kedua ataupun sebaliknya, maka hasil INNER JOIN tidak akan ditampilkan
- Ini adalah default JOIN di PostgreSQL
- Jika kita menggunakan JOIN seperti yang sudah kita praktekan sebelumnya, sebenarnya itu akan melakukan INNER JOIN

Inner Join Diagram





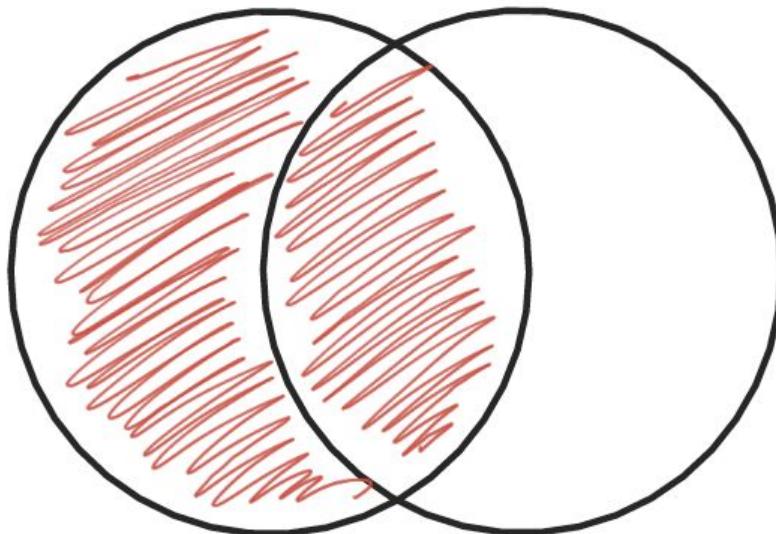
Melakukan Inner Join

```
▽ SELECT *  
  FROM categories  
        INNER JOIN products ON products.id_category = categories.id;
```

Left Join

- LEFT JOIN adalah mekanisme JOIN seperti INNER JOIN, namun semua data di table pertama akan diambil juga
- Jika ada yang tidak memiliki relasi di table kedua, maka hasilnya akan NULL

Left Join Diagram





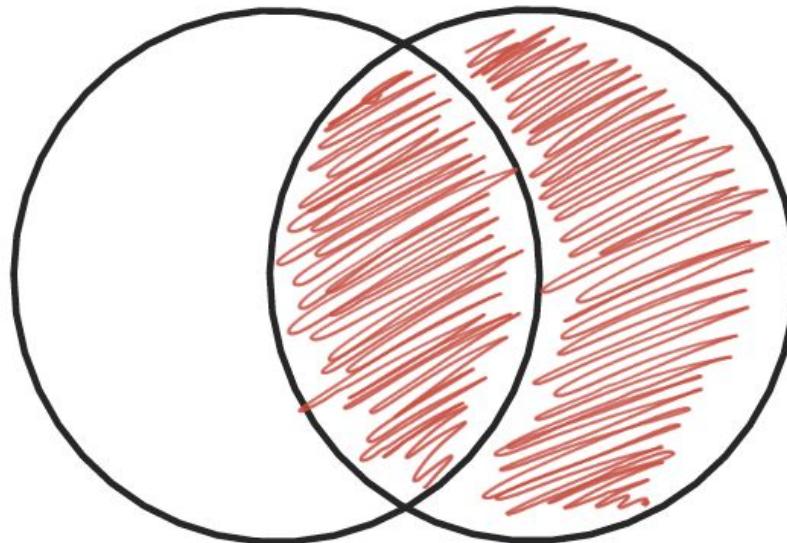
Melakukan Left Join

```
▽ SELECT *  
      FROM categories  
            LEFT JOIN products ON products.id_category = categories.id;
```

Right Join

- RIGHT JOIN adalah mekanisme JOIN seperti INNER JOIN, namun semua data di table kedua akan diambil juga
- Jika ada yang tidak memiliki relasi di table pertama, maka hasilnya akan NULL

Right Join Diagram





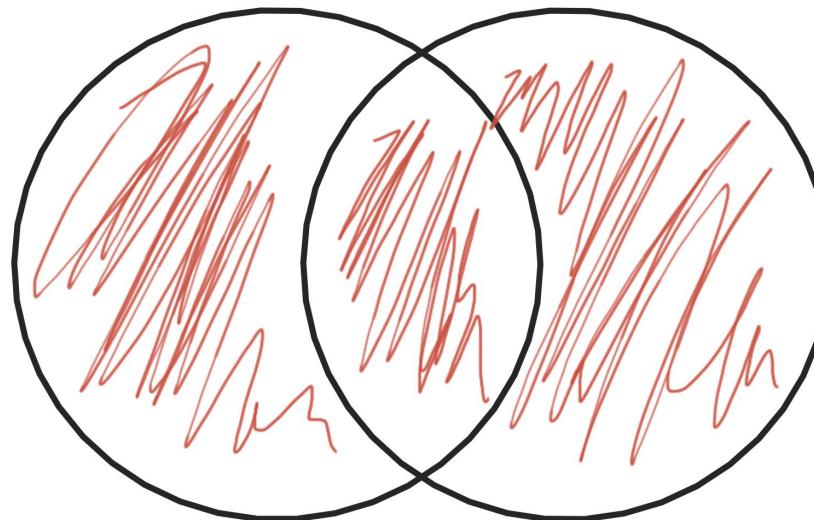
Melakukan Right Join

```
SELECT *  
FROM categories  
RIGHT JOIN products ON products.id_category = categories.id;
```

Full Join

- Full Join adalah join dimana semua data di tabel pertama dan kedua akan ditampilkan
- Jika tidak ada data join, maka hasilnya akan berisi data Null

Full Join Diagram





Menggunakan Full Outer Join

```
SELECT *  
FROM categories  
    FULL JOIN products ON products.id_category = categories.id;
```

Subqueries

Subquery di WHERE

- PostgreSQL mendukung pencarian data menggunakan WHERE dari hasil SELECT query
- Fitur ini dinamakan Subquery
- Contoh, kita ingin mencari products yang harganya diatas harga rata-rata, artinya kita akan melakukan SELECT dengan WHERE price > harga rata, dimana harga rata-rata perlu kita hitung menggunakan query SELECT lainnya menggunakan aggregate function AVG

Melakukan Subquery di WHERE Clause

```
SELECT *  
FROM products  
WHERE price > (SELECT AVG(price) FROM products);
```



Subquery di FROM

- Selain di WHERE clause, Subquery juga bisa dilakukan di FROM clause
- Misal kita ingin mencari data dari hasil query SELECT, itu bisa kita lakukan di PostgreSQL



Melakukan Subquery di FROM Clause

```
▽ SELECT MAX(price)
▽ FROM (SELECT products.price as price
        FROM categories
        JOIN products ON products.id_category = categories.id) as contoh;
```

Set Operator

Set Operator

PostgreSQL mendukung operator Set, dimana ini adalah operasi antara hasil dari dua SELECT query. Ada banyak jenis operator Set, yaitu :

- UNION
- UNION ALL
- INTERSECT, dan
- EXCEPT



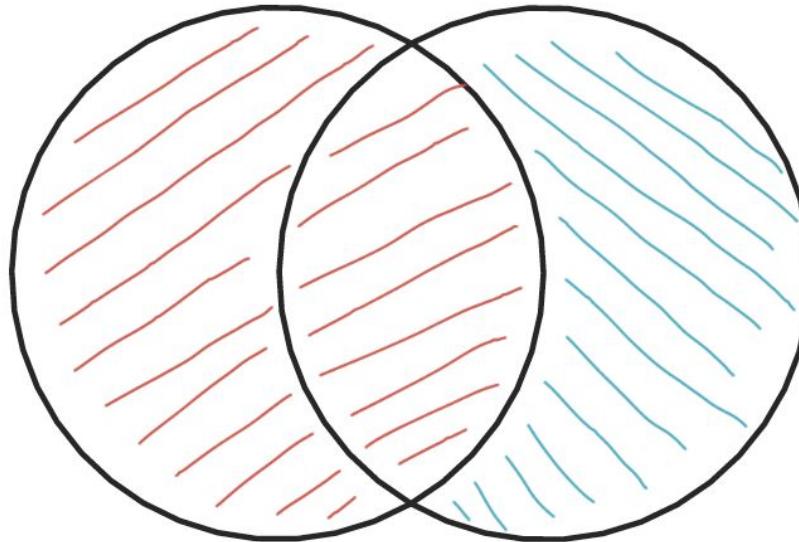
Membuat Table Guest Book

```
✓ CREATE TABLE guestbooks
✓ (
    id      SERIAL      NOT NULL,
    email   VARCHAR(100) NOT NULL,
    title   VARCHAR(100) NOT NULL,
    content TEXT,
    PRIMARY KEY (id)
);
```

UNION

- UNION adalah operasi menggabungkan dua buah SELECT query, dimana jika terdapat data yang duplikat, data duplikatnya akan dihapus dari hasil query

Diagram UNION





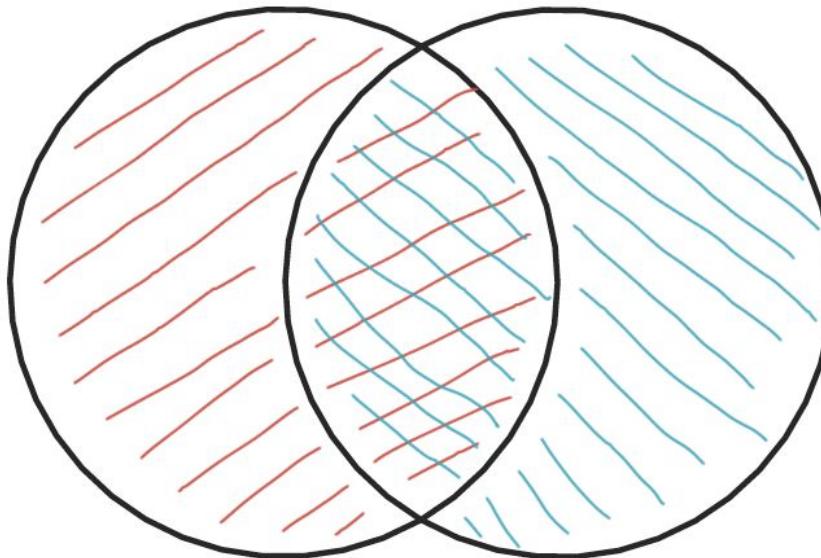
Melakukan Query UNION

```
› ▾ SELECT DISTINCT email  
    FROM customer  
    UNION  
› ▾ SELECT DISTINCT email  
    FROM guestbooks;
```

UNION ALL

- UNION ALL adalah operasi yang sama dengan UNION, namun data duplikat tetap akan ditampilkan di hasil query nya

Diagram UNION ALL





Melakukan Query UNION ALL

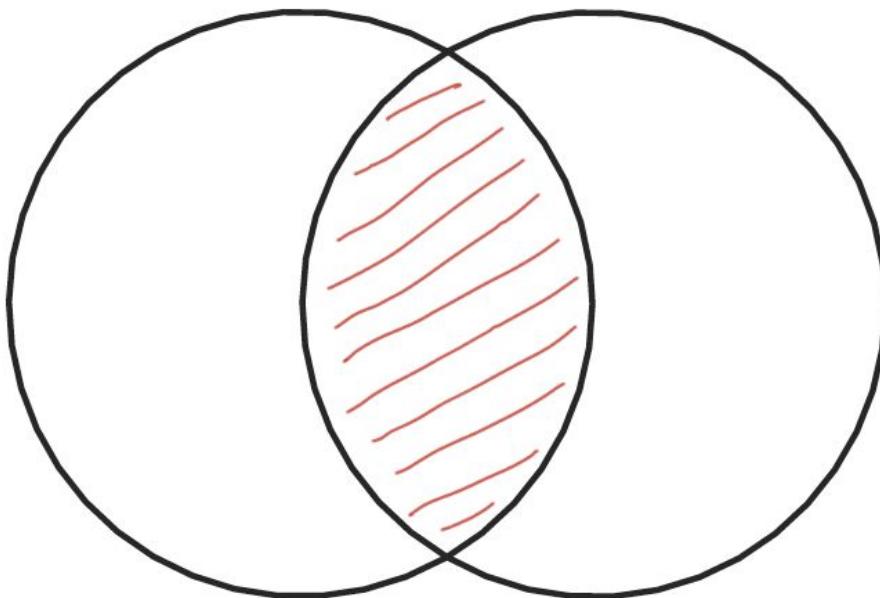
```
▽ SELECT DISTINCT email
  FROM customer
  UNION
▽ SELECT DISTINCT email
  FROM guestbooks;

▽ ▽ SELECT email, COUNT(email)
▽ FROM (SELECT DISTINCT email
        FROM customer
        UNION ALL
        SELECT DISTINCT email
        FROM guestbooks) as contoh
        GROUP BY email;
```

INTERSECT

- INTERSECT adalah operasi menggabungkan dua query, namun yang diambil hanya data yang terdapat pada hasil query pertama dan query kedua
- Data yang tidak hanya ada di salah satu query, akan dihapus di hasil operasi INTERSECT
- Data nya muncul tidak dalam keadaan duplikat

Diagram INTERSECT





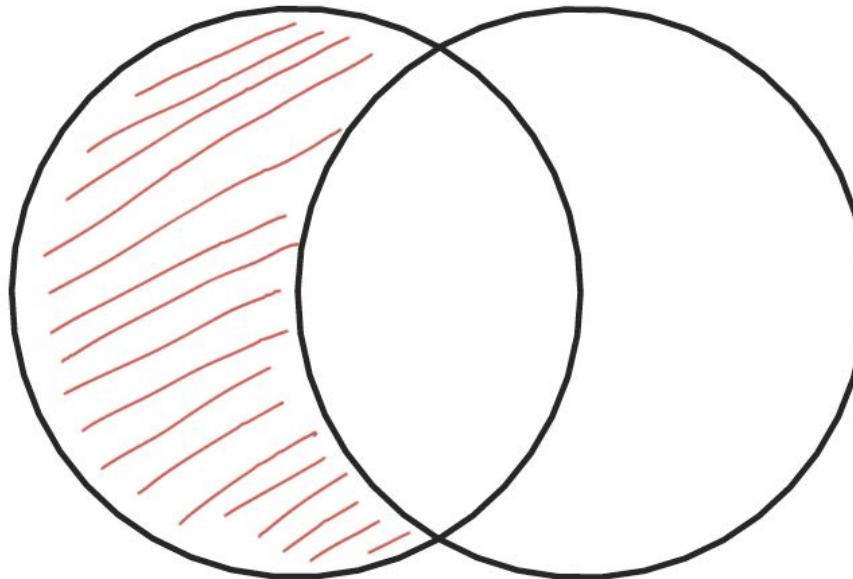
Melakukan Query INTERSECT

```
SELECT DISTINCT email  
FROM customer  
INTERSECT  
SELECT DISTINCT email  
FROM guestbooks;
```

EXCEPT

- EXCEPT adalah operasi dimana query pertama akan dihilangkan oleh query kedua
- Artinya jika ada data di query pertama yang sama dengan data yang ada di query kedua, maka data tersebut akan dihapus dari hasil query EXCEPT

Diagram EXCEPT





Melakukan Query EXCEPT

```
▽ SELECT DISTINCT email  
  FROM customer  
  EXCEPT  
▽ SELECT DISTINCT email  
  FROM guestbooks;
```

Transaction

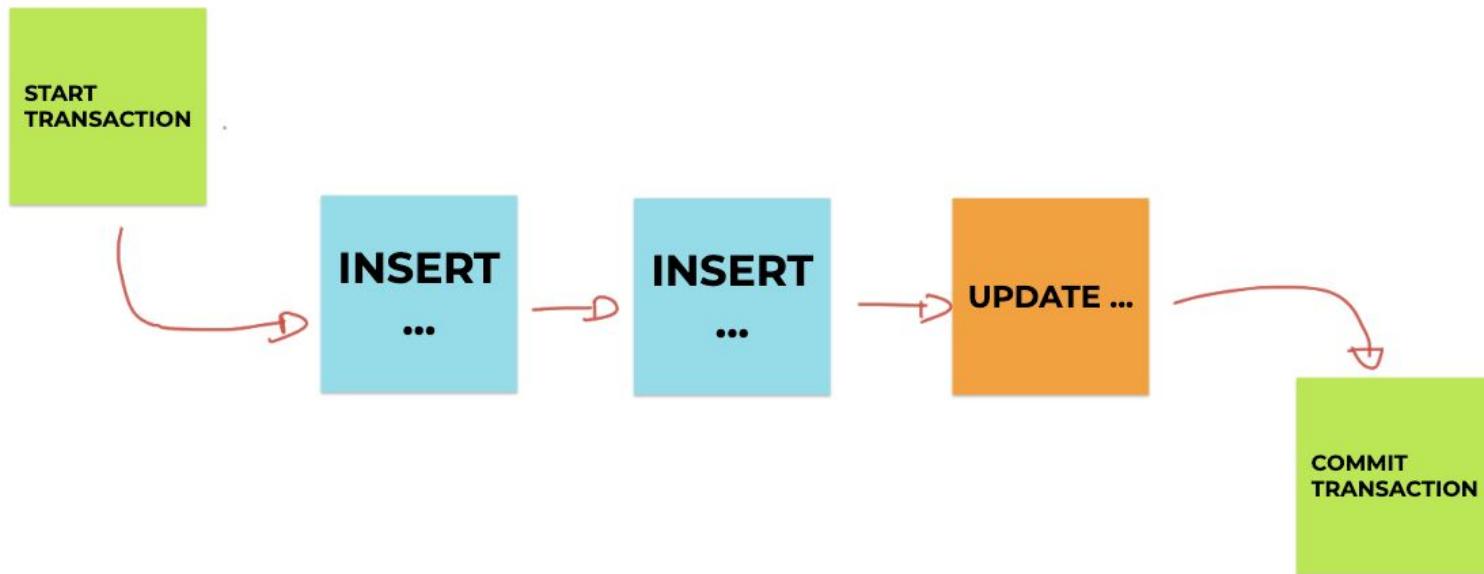
Kenapa Butuh Transaction?

- Saat membuat aplikasi berbasis database, jarang sekali kita akan melakukan satu jenis perintah SQL per aksi yang dibuat aplikasi
- Contoh, ketika membuat toko online, ketika customer menekan tombol Pesan, banyak yang harus kita lakukan, misal
 - Membuat data pesanan di tabel order
 - Membuat data detail pesanan di tabel order detail
 - Menurunkan quantity di tabel produk
 - Dan yang lainnya
- Artinya, bisa saja dalam satu aksi, kita akan melakukan beberapa perintah sekaligus
- Jika terjadi kesalahan di salah satu perintah, harapannya adalah perintah-perintah sebelumnya dibatalkan, agar data tetap konsisten

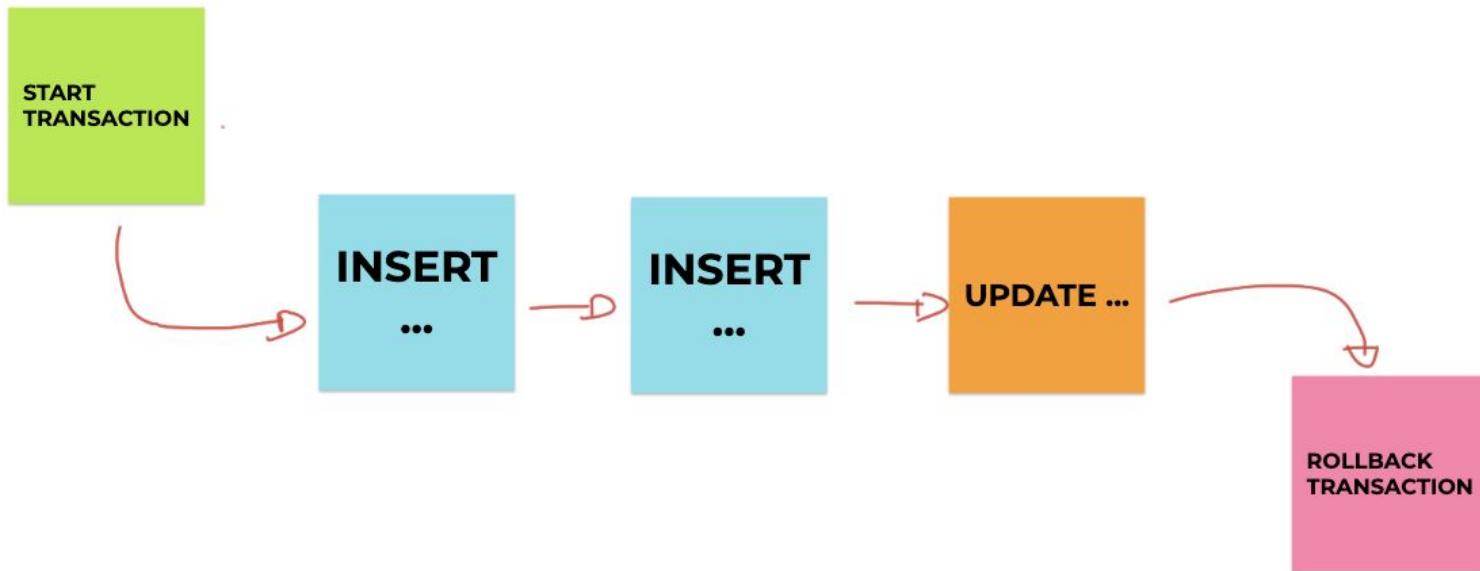
Database Transaction

- Database transaction adalah fitur di DBMS dimana kita bisa memungkinkan beberapa perintah dianggap menjadi sebuah kesatuan perintah yang kita sebut transaction
- Jika terdapat satu saja proses gagal di transaction, maka secara otomatis perintah-perintah sebelumnya akan dibatalkan
- Jika sebuah transaction sukses, maka semua perintah akan dipastikan sukses

Membuat Transaction



Membatalkan Transaction



Transaction di PostgreSQL

Perintah	Keterangan
START TRANSACTION	Memulai proses transaksi, proses selanjutnya akan dianggap transaksi sampai perintah COMMIT atau ROLLBACK
COMMIT	Menyimpan secara permanen seluruh proses transaksi
ROLLBACK	Membatalkan secara permanen seluruh proses transaksi

Yang Tidak Bisa Menggunakan Transaction

- Perintah DDL (Data Definition Language) tidak bisa menggunakan fitur transaction
- DDL adalah perintah-perintah yang digunakan untuk merubah struktur, seperti membuat tabel, menambah kolom, menghapus tabel, menghapus database, dan sejenisnya
- Transaction hanya bisa dilakukan pada perintah DML (Data Manipulation Language), seperti operasi INSERT, UPDATE dan DELETE

Locking

Locking

- Locking adalah proses mengunci data di DBMS
- Proses mengunci data sangat penting dilakukan, salah satunya agar data benar-benar terjamin konsistensinya
- Karena pada kenyataannya, aplikasi yang akan kita buat pasti digunakan oleh banyak pengguna, dan banyak pengguna tersebut bisa saja akan mengakses data yang sama, jika tidak ada proses locking, bisa dipastikan akan terjadi RACE CONDITION, yaitu proses balapan ketika mengubah data yang sama
- Contoh saja, ketika kita belanja di toko online, kita akan balapan membeli barang yang sama, jika data tidak terjaga, bisa jadi kita salah mengupdate stock karena pada saat yang bersamaan banyak yang melakukan perubahan stock barang

Locking Record

- Saat kita melakukan proses TRANSACTION, lalu kita melakukan proses perubahan data, data yang kita ubah tersebut akan secara otomatis di LOCK
- Hal ini membuat proses TRANSACTION sangat aman
- Oleh karena itu, sangat disarankan untuk selalu menggunakan fitur TRANSACTION ketika memanipulasi data di database, terutama ketika perintah manipulasinya lebih dari satu kali
- Locking ini akan membuat sebuah proses perubahan yang dilakukan oleh pihak lain akan diminta untuk menunggu
- Data akan di lock sampai kita melakukan COMMIT atau ROLLBACK transaksi tersebut

Locking Record Manual

- Selain secara otomatis, kadang saat kita membuat aplikasi, kita juga sering melakukan SELECT query terlebih dahulu sebelum melakukan proses UPDATE misalnya.
- Jika kita ingin melakukan locking sebuah data secara manual, kita bisa tambahkan perintah FOR UPDATE di belakang query SELECT
- Saat kita lock record yang kita select, maka jika ada proses lain akan melakukan UPDATE, DELETE atau SELECT FOR UPDATE lagi, maka proses lain diminta menunggu sampai kita selesai melakukan COMMIT atau ROLLBACK transaction

Deadlock

- Saat kita terlalu banyak melakukan proses Locking, hati-hati akan masalah yang bisa terjadi, yaitu DEADLOCK
- Deadlock adalah situasi ada 2 proses yang saling menunggu satu sama lain, namun data yang ditunggu dua-duanya di lock oleh proses lainnya, sehingga proses menunggunya ini tidak akan pernah selesai.

Contoh Deadlock

- Proses 1 melakukan SELECT FOR UPDATE untuk data 001
- Proses 2 melakukan SELECT FOR UPDATE untuk data 002
- Proses 1 melakukan SELECT FOR UPDATE untuk data 002, diminta menunggu karena di lock oleh Proses 2
- Proses 2 melakukan SELECT FOR UPDATE untuk data 001, diminta menunggu karena di lock oleh Proses 1
- Akhirnya Proses 1 dan Proses 2 saling menunggu
- Deadlock terjadi

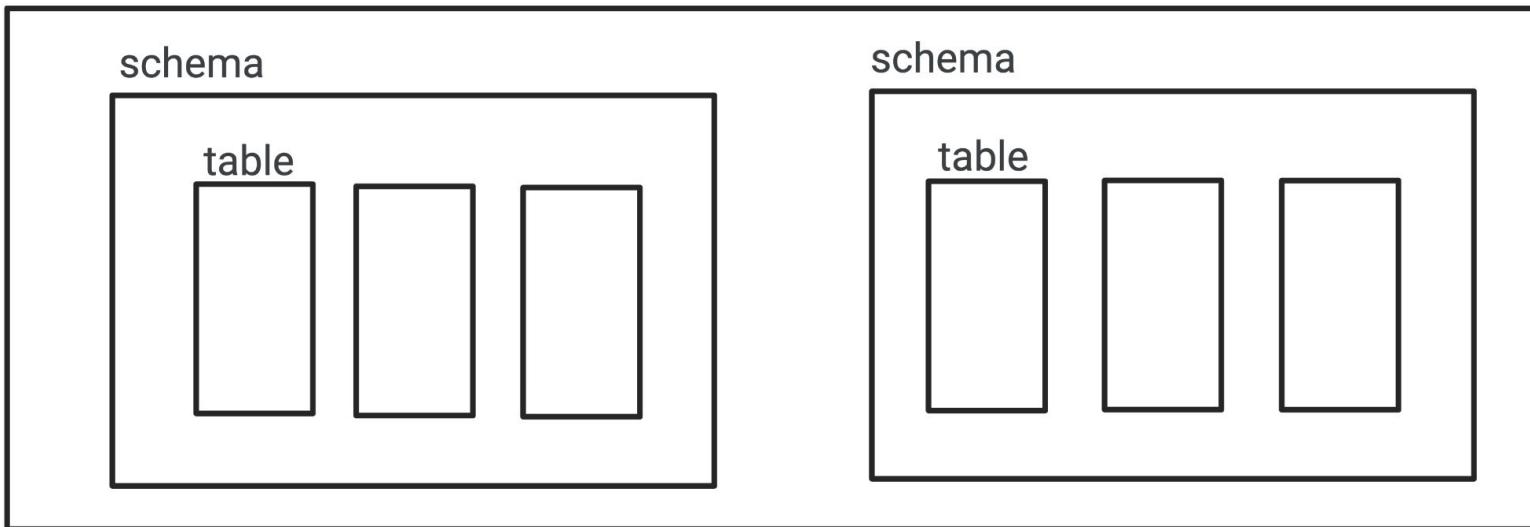
Schema

Schema

- Diawal, kita mengibaratkan bahwa database adalah sebuah folder, dan table adalah file-file nya
- Di PostgreSQL, terdapat fitur bernama Schema, anggap saja ini adalah folder didalam database
- Saat kita membuat database, secara tidak sadar sebenarnya kita menyimpan semua table kita di schema public
- Kita bisa membuat schema lain, dan pada schema yang berbeda, kita bisa membuat table dengan nama yang sama

Diagram Schema

database



Public Schema

- Saat kita membuat database di PostgreSQL, secara otomatis terdapat schema bernama public
- Dan saat kita membuat table, secara otomatis kita akan membuat table tersebut di schema public



Melihat Schema Saat Ini

```
SELECT current_schema();
```

```
SHOW search_path;
```

Membuat dan Menghapus Schema

```
CREATE SCHEMA contoh;
```

```
✓ DROP schema contoh;
```



Pindah Schema

```
SET search_path TO contoh;
```

```
SHOW search_path ;
```

```
SELECT current_schema();
```

Membuat Table di Schema

- Saat kita membuat table, secara otomatis PostgreSQL akan membuatkan table di schema yang sedang kita pilih
- Jika kita ingin menentukan schema secara manual tanpa menggunakan schema yang sedang dipilih, kita bisa menambahkan prefix nama schema di awal nama table nya
- Misal namaschema.namatable
- Termasuk jika ingin melakukan operasi DML ke table, bisa menggunakan prefix namaschema



Membuat Table

```
✓ CREATE TABLE contoh.products
✓ (
    id SERIAL      NOT NULL,
    name VARCHAR(100) NOT NULL
);

✓ INSERT INTO contoh.products(name)
✓ VALUES ('iPhone'),
        ('Play Stasion');

✓ SELECT * FROM contoh.products;
```

User Management

Root User

- Secara default, PostgreSQL membuat user utama sebagai super administrator
- Namun best practice nya, saat kita menjalankan PostgreSQL dengan aplikasi yang kita buat, sangat disarankan tidak menggunakan user utama
- Lebih baik kita buat user khusus untuk tiap aplikasi, bahkan kita bisa batasi hak akses user tersebut, seperti hanya bisa melakukan SELECT, dan tidak boleh melakukan INSERT, UPDATE atau DELETE

Hak Akses dan User

- Dalam user management PostgreSQL, kita akan mengenal istilah Hak Akses dari User

Membuat/Menghapus User

```
CREATE ROLE eko;  
CREATE ROLE budi;  
  
DROP ROLE eko;  
DROP ROLE budi;
```

Role Option

- Saat membuat user / role, terdapat banyak opsi yang bisa kita gunakan
- Kita bisa lihat semuanya disini :
- <https://www.postgresql.org/docs/current/sql-createrole.html>
- <https://www.postgresql.org/docs/current/sql-alterrole.html>

Menambah Option ke User

```
ALTER ROLE eko LOGIN PASSWORD 'rahasia';  
ALTER ROLE budi LOGIN PASSWORD 'rahasia';
```



Daftar Hak Akses

- Ada banyak sekali hak akses di PostgreSQL
- Kita bisa melihatnya di daftar tabel yang terdapat di halaman berikut :
- <https://www.postgresql.org/docs/current/sql-grant.html>
- Setelah membuat user / role, kita bisa menambahkan hak akses ke user tersebut
- Dan kita juga bisa menghapus hak akses yang sudah kita berikan ke user
- <https://www.postgresql.org/docs/current/sql-revoke.html>



Menambah/Menghapus Hak Akses ke User

```
GRANT INSERT, UPDATE, SELECT ON ALL TABLES IN SCHEMA public TO eko;
```

```
GRANT INSERT, UPDATE, SELECT ON customer TO budi;
```

```
REVOKE INSERT, UPDATE, SELECT ON customer FROM budi;
```



Backup Database

Backup Database

- Saat membuat aplikasi menggunakan database, ada baiknya kita selalu melakukan backup data secara reguler
- Untungnya PostgreSQL mendukung proses backup database
- Untuk melakukan backup database, kita tidak menggunakan perintah SQL, melainkan PostgreSQL menyediakan sebuah aplikasi khusus untuk melakukan backup database, namanya adalah pg_dump
- <https://www.postgresql.org/docs/current/app-pgdump.html>

Melakukan Backup Database

```
pg_dump --host=localhost --port=5432 --dbname=belajar --username=khannedy --format=plain  
--file=/Users/khannedy/backup.sql
```

Restore Database

Restore Database

- Selain melakukan backup database, di PostgreSQL juga kita bisa melakukan proses restore data dari file hasil backup
- Untuk melakukan restore database, kita bisa menggunakan aplikasi psql dari PostgreSQL

Membuat Database untuk Restore

```
CREATE DATABASE belajar_restore;
```

Restore Database

```
psql --host=localhost --port=5432 --dbname=belajar_restore --username=khannedy  
--file=/Users/khannedy/backup.sql
```

Materi Selanjutnya

Materi Selanjutnya

- PostgreSQL ACID
- PostgreSQL Table Partitioning
- Studi Kasus Database Design
- Belajar Bahasa Pemrograman