**DATABASE**

# Target Pembelajan

## Indikator

1. Mengetahui database sebagai penyimpanan data
2. Mengetahui tipe database
3. Mengetahui cara query database
4. Praktek DDL,DML di SQL

## Tools

1. PostgreSql

## Proses Alur

## Peserta diharapkan mengikuti setiap langkah pada dokumen ini.

# Database

*Database* atau basis data adalah kumpulan data yang dikelola sedemikian rupa berdasarkan ketentuan tertentu yang saling berhubungan sehingga mudah dalam pengelolaannya. Melalui pengelolaan tersebut pengguna dapat memperoleh kemudahan dalam mencari informasi, menyimpan informasi dan membuang informasi.

## Relational Database

Relational Database atau basis data relasional adalah basis data yang mengorganisir berdasarkan model hubungan data.

## Jenis database

|  |  |
| --- | --- |
| Database | Desc |
| **MySQL** | WordPress, Google, Flickr, Youtube, Facebook, Joomla, phpBB, Drupal, dan MODx |
| **PostgreSQL** | adalah sistem yang dikembangkan dari MySQL |
| **MariaDB** |  |
| **MongoDB** | MongoDB adalah software database yang berorientasi pada dokumen cross platform dan open source. MongoDB menggunakan dokumen yang mirip dengan skema JSON |
| **Oracle Database** |  |
| **SAP HANA** | Sistem ini memiliki fungsi utama sebagai database server yang menyimpan dan mengambil data sesuai permintaan aplikasi. |
| **IBM Db2** | Hanya saja versi terbarunya IBM Db2 memiliki multi-fungsi yang mendukung fitur relasional dan non relasional seperti JSON dan XML. |
| **MemSQL** | sistem manajemen basis data SQL terdistribusi dan in–memory. Sistem ini juga termasuk ke dalam sistem manajemen basis data relasional (RDBMS). |
| **Interbase** | sistem DBMS(**Database Management** System) |
| **Firebird** | Firebird adalah sistem SQL yang open source dan berjalan di sistem operasi mac OS X, Windows, Linux, dan sistem operasi lainnya yang unik |

## SQL VS NOSQL

**SQL** merupakan singkatan dari Structured Query Language. Sebenarnya SQL sendiri merupakan bahasa yang digunakan untuk mengatur/mengelola data dalam database relasional.

Database relasional menggunakan ‘relasi’ (yang biasanya disebut tabel) untuk menyimpan data dan mencocokkan data tersebut dengan memakai karakteristik umum di setiap dataset. Beberapa contoh database management system yang menggunakan SQL antara lain Oracle, Sybase, Microsoft SQL Server, PostgreSQL.

**NoSQL** merupakan database yang tidak membutuhkan skema dan tidak memiliki relasi untuk setiap tabel. Semua bentuk dokumen dari NoSQL adalah JSON yang mudah dibaca dan dimengerti. NoSQL banyak diminati karena memiliki performa yang tinggi dan bersifat non-relasional sehingga dapat memakai berbagai model data. Beberapa contoh dari database NoSQL yaitu MongoDB, MarkLogic, Couchbase, CloudDB, dan Dynamo DB.

|  |  |  |
| --- | --- | --- |
| **Tipe Database** | **SQL** | **NoSQL** |
| Tipe Database | Relational database | Non-Relational database |
| Skema | Pre-defined schema | Dynamic schema |
| Kategori Database | Table-based databases | Document-based databases |
| Bahasa | Structured Query Language | Un-Structured Query Language |
| Kompleksitas Query | Sangat cocok untuk kueri yang kompleks | Tidak cocok untuk kueri yang kompleks |
| Online Processing | Digunakan untuk OLTP | Digunakan untuk OLAP |
| Basis Properties | Berdasarkan basis ACID properties | Berdasarkan basis CAP theorem |
| Skalabilitas | Dapat diskalakan secara vertikal | Dapat diskalakan secara horizontal |
| Elastisitas | Membutuhkan downtime dalam banyak kasus | Otomatis, Tidak perlu outage |
| Contoh | MySQL, Oracle | MongoDB, Cassandra |

# PostgreSQL

## **Fitur-Fitur PostgreSQL**

PostgreSQL dikenal sebagai sistem *database*dengan kecanggihan dan fungsinya yang andal. Tentunya, kinerja PostgreSQL didukung oleh fitur lengkap yang disediakan oleh sistem ini sendiri. Berikut fitur yang disediakan oleh PostgreSQL, di antaranya :

* Point-in-time Recovery : Backup server secara terus menerus;
* Procedural Languages : Mendukung bahasa prosedural seperti Python, Perl, dan lain sebagainya;
* Asynchronous Replication : Dapat menggandakan database secara asinkron;
* Data Integrity : Mendukung primary key, foreign key, dan lain sebagainya;
* Common Table : Digunakan untuk pengelolaan data hirarki. Fitur ini memungkinkan Adanya pengelolaan reaksi data pada SQL query dengan mudah dan aman;
* Savapoints : Dapat menangani error pada transaksi yang kompleks.

https://www.linkedin.com/pulse/kenali-apa-itu-postgresql-dan-perbedaanya-dengan-mysql-chandra-henny/?originalSubdomain=id

# SQL

Perintah SQL

## Data Definition Language (DDL)

Tujuan dari penggunaan DDL adalah untuk membuat struktur database dengan tepat. Kemudian jenis DDL terbagi menjadi lima bagian, diantaranya adalah sebagai berikut.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Create** | 1. **Create Database**  |  | | --- | | **create** **database** dibimbing1; |  1. **Create Tabel**  |  |  | | --- | --- | | CREATE TABLE table\_name(  column1 datatype,  column2 datatype,  column3 datatype,  .....  columnN datatype,  PRIMARY KEY(columns name )  );  Contoh  **CREATE** **TABLE** barang(  id **int** **NOT** **NULL**,  nama **VARCHAR** (200) **NOT** **NULL**,  harga **DECIMAL** (18, 2),  satuan **VARCHAR** (200),  stok **int**,  **PRIMARY** **KEY** (id));  **Or with auto increment**   |  | | --- | | **Step 1**  **CREATE** **SEQUENCE** barang\_id\_seq  **INCREMENT** 1  **START** 1; |   **Atau update sequeense :**  **ALTER** **SEQUENCE** transaksi\_id\_seq **OWNED** **BY** transaksi.id;  -- perlu di alter -> ke tabel barang ?  **ALTER** **TABLE** barang **ALTER** **COLUMN** id **SET** **DEFAULT** **nextval**('barang\_id\_seq');  **Step 2**  **CREATE** **TABLE** barang(  id **int** **NOT** **NULL** **DEFAULT** **nextval**('barang\_id\_seq') ,  nama **VARCHAR** (200) **NOT** **NULL**,  harga **DECIMAL** (18, 2),  satuan **VARCHAR** (200),  stok **int**,  **PRIMARY** **KEY** (id));  Dengan mengunakan serial: tidak pelru membuat sequesnse manual  Contoh  **CREATE** **TABLE** public.users (  id serial4 **NOT** **NULL**,  username **varchar**(255) **NULL**,  email\_address **varchar**(255) **NULL**,  "password" **varchar**(255) **NULL**,  **CONSTRAINT** users\_pkey **PRIMARY** **KEY** (id)  ); | |
| **Alter** | 1. **Table name**  |  | | --- | | **ALTER TABLE table\_name**  **RENAME TO new\_table\_name;**  **Contoh**  **alter** **table** barang  **rename** **to** barang1; |  1. **Add Column**  |  | | --- | | ALTER TABLE table\_name ADD column\_name datatype;  **contoh**  **ALTER** **TABLE** barang  **ADD** newcolumn **varchar**(255);  **more column**  **ALTER** **TABLE** barang  **ADD** date1 **date**,  **ADD** quantity **int**; |  1. **Modify Type Data Column**  |  | | --- | | ALTER TABLE table\_name ALTER COLUMN column\_name TYPE datatype;  **Contoh**  **ALTER** **TABLE** barang  **ALTER** **COLUMN** date1 **TYPE** **varchar**(500); | |
| **Rename** | 1. Rename Tabel  |  | | --- | | ALTER TABLE table\_name  RENAME TO new\_table\_name;  Contoh  **ALTER** **TABLE** barang **RENAME** **TO** barang1; |  1. Rename Column  |  | | --- | | ALTER TABLE table\_name  RENAME COLUMN old\_name TO new\_name;  **Contoh**  **ALTER** **TABLE** barang  **RENAME** **COLUMN** date2 **TO** date1; | |
| **Drop** | 1. **Drop Column**  |  | | --- | | ALTER TABLE table\_name DROP COLUMN column\_name;  **Contoh**  **ALTER** **TABLE** barang  **drop** **column** newcolumn; |  1. **Drop Tabel**  |  | | --- | | DROP TABLE table\_name;  **Contoh**  **DROP** **TABLE** barang; |  1. **Drop Database**  |  | | --- | | DROP DATABASE databasename; | |
| **Truncate** | 1. **Truncate Tabel**  |  | | --- | | TRUNCATE TABLE barang; | |
| **Comment** | Syntax **Using** – symbol  Contoh  --ini comment di sql editor postgre : ctrl+/ |

Ref :

<https://www.geeksforgeeks.org/postgresql-rename-table>

<https://www.w3schools.com/sql/sql_check.asp>

https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-serial/

## Data Manipulation Language (DML)

Adalah sebuah perintah SQL yang bertujuan untuk memanipulasi data dalam database. Perintah dari DML terbagi menjadi empat bagian, yaitu:

* Mengetahui Perintah Select
* Mengetahui Perintah Insert
* Mengetahui Perintah Update
* Mengetahui Perintah Delete
* Mengetahui Perintah Join
* Mengetahui Perintah Merge
* Mengetahui Perintah Call
* Mengetahui Perintah Explain Plan
* Mengetahui Perintah Lock Table
* Mengetahui Perintah Sub Query
* Loop

Berikut Penjelasan

#### Mengetahui Perintah Insert

Pernyataan INSERT INTO digunakan untuk menyisipkan record baru ke dalam tabel.

|  |
| --- |
| INSERT INTO table\_name (column1, column2, column3, ...) VALUES (value1, value2, value3, ...);  **Contoh 1: insert one record**  **INSERT** **INTO** barang (nama,harga,satuan, stok)  **VALUES** ('pulpen', 2500, 'pcs', '200');  **Contoh 2 : Insert many record**  **INSERT** **INTO** barang (nama,harga,satuan, stok)  **VALUES** ('pulpen', 2500, 'pcs', '200'),  ('buku', 2500, 'pcs', '200'),  ('pensil', 2500, 'pcs', '200'); |

1. INSERT IGNORE… VALUES
2. INSERT… VALUES…ON DUPLICATE KEY UPDATE

https://www.duniailkom.com/tutorial-mysql-menambahkan-data-ke-dalam-tabel-insert/

#### Mengetahui Perintah Update

Pernyataan UPDATE digunakan untuk mengubah catatan yang ada dalam tabel.

|  |
| --- |
| UPDATE table\_name SET column1 = value1, column2 = value2, ... WHERE condition;  **Contoh:**  **UPDATE** barang  **SET** nama = 'Buku Gambar', harga= 3000  **WHERE** id = 1; |

#### Mengetahui Perintah Delete

Pernyataan DELETE digunakan untuk menghapus record yang ada dalam tabel.

|  |
| --- |
| DELETE FROM table\_name WHERE condition;  **Contoh:**  **DELETE** **FROM** barang **WHERE** id=1; |

#### Mengetahui Perintah Select

Pernyataan SELECT digunakan untuk memilih/menampilakn data dari database.

|  |
| --- |
| SELECT column1, column2, ... FROM table\_name;  **Contoh**:  **select** \* **from** barang;  **select** id,nama,harga,satuan **from** barang; |

Pernyataan SELECT DISTINCT digunakan untuk mengembalikan hanya nilai yang berbeda (unik).

|  |
| --- |
| SELECT DISTINCT column1, column2, ... FROM table\_name;  **Contoh:**  **SELECT** **DISTINCT** nama **FROM** barang; |

##### Order By

Berfungsi mengurutkan value column asc atau desc

|  |
| --- |
| ***Syntax:*** *SELECT column\_1, column\_2 FROM table\_name ORDER BY column\_1 [ASC | DESC], column\_2 [ASC | DESC];* |

|  |
| --- |
| **SELECT** id,nama, harga **FROM** barang b **order** **by** nama **asc**, harga **asc** ; |

##### Where clause

digunakan untuk memfilter hasil yang return oleh pernyataan SELECT.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Below table provides us with the list of comparison operators valid in PostgreSQL:   |  |  | | --- | --- | | Operator | Description | | = | Equal | | > | Greater than | | < | Less than | | >= | Greater than or equal to | | <= | Less than or equal to | | <> or =! | Not equal | | AND | Logical AND operator | | OR | Logical OR operator | |

|  |
| --- |
| **SELECT** id,nama, harga **FROM** barang b **where** harga >90000; |

##### Fetch

Klausa FETCH PostgreSQL memiliki fungsi yang mirip dengan klausa LIMIT PostgreSQL. Ini digunakan untuk mengambil sebagian dari baris yang dikembalikan oleh kueri. Karena klausa LIMIT bukan perintah SQL standar, PostgreSQL menyediakan cara standar untuk mengambil subset hasil dari kueri.

|  |
| --- |
| Syntax:  **OFFSET** **start** { **ROW** | **ROWS** }  **FETCH** { **FIRST** | **NEXT** } [ row\_count ] { **ROW** | **ROWS** } **ONLY** |

|  |
| --- |
| **SELECT** id,nama, harga **FROM** barang b **order** **by** id **FETCH** **FIRST** 5 **ROW** **ONLY**; |

|  |
| --- |
| **SELECT** id,nama, harga **FROM** barang b **order** **by** id **OFFSET** 10 **ROWS**  **FETCH** **FIRST** 5 **ROW** **ONLY**; |

##### IN

Operator PostgreSQL IN digunakan dengan klausa WHERE untuk memeriksa daftar nilai.

|  |
| --- |
| Syntax: value **IN** (value1, value2, …) |

|  |
| --- |
| **SELECT** id,nama, harga **FROM** barang b **where** id **IN** (2, 3,6,9,10) **order** **by** id ; |

##### Having

Perintah HAVING digunakan sebagai pengganti WHERE dengan fungsi agregat(COUNT, MAX, MIN, SUM, AVG)

|  |
| --- |
| Syntax:  **SELECT**  column\_1,  aggregate\_function (column\_2)  **FROM**  tbl\_name  **GROUP** **BY**  column\_1  **HAVING**  **condition**; |

|  |
| --- |
| **Contoh :**  **SELECT**  harga,  **COUNT** (id)  **FROM**  barang  **GROUP** **BY**  harga  **HAVING**  **COUNT** (harga) >= 1;    Jika dilihat dari output : harga 100.000 terdapat total 14 di tabel barang.  harga 200 terdapat total 1 di tabel barang. |

##### Group by

digunakan untuk mengelompokkan data (record) yang memiliki nilai yang sama. seperti “menemukan jumlah data barang sesuai dengan kategori”. SQL GROUP BY sering digunakan pada fungsi agregat seperti (COUNT(), MAX(), MIN(), SUM(), AVG()) yang menampilkan beberapa kolom.

|  |
| --- |
| **SELECT**  harga,  **COUNT** (id)  **FROM**  barang  **GROUP** **BY**  Harga |

##### Like operator

Operator LIKE yang digunakan dalam klausa WHERE untuk mencari pola yang ditentukan pada kolom / field.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| * The percent sign (%) * The underscore (\_)   **Like type string**  SELECT \* FROM barang  WHERE nama LIKE ‘abc%’;  **Like type integer:**  **SELECT** \* **FROM** barang  **WHERE** **CAST**(harga **AS** **VARCHAR**(20)) **LIKE** '%10%';   |  |  | | --- | --- | | **No.** | **Statement & Description** | | 1 | **WHERE SALARY LIKE ‘200%’**  Finds any values that start with 200. | | 2 | **WHERE SALARY LIKE ‘%200%’**  Finds any values that have 200 in any position. | | 3 | **WHERE SALARY LIKE ‘\_00%’**  Finds any values that have 00 in the second and third positions. | | 4 | **WHERE SALARY LIKE ‘2\_%\_%’**  Finds any values that start with 2 and are at least 3 characters in length. | | 5 | **WHERE SALARY LIKE ‘%2’**  Finds any values that end with 2. | | 6 | **WHERE SALARY LIKE ‘\_2%3’**  Finds any values that have a 2 in the second position and end with a 3. | | 7 | **WHERE SALARY LIKE ‘2\_\_\_3’**  Finds any values in a five-digit number that start with 2 and end with 3. | |

##### Between

BETWEEN SQL digunakan untuk ketika memilih nilai dalam rentang tertentu. Nilai dapat berupa angka, teks, atau tanggal.

|  |
| --- |
| **Syntax:** value BETWEEN low AND high;  **SELECT** id,nama, harga **FROM** barang b **where** harga **BETWEEN** 10 **AND** 10000; |

|  |
| --- |
| **SELECT** id,nama, harga,created\_date **FROM** barang b **where** created\_date **BETWEEN** '2007-02-07'  **AND** '2022-10-15'; |

##### IF Statement - Prosedure

Mengembalikan "YA" jika kondisinya TRUE, atau "NO" jika kondisinya FALSE:

|  |
| --- |
| **Syntax:**  IF condition THEN  statements;  END IF; |

|  |
| --- |
| **DO** **$$**  **DECLARE**  a **integer** := 10;  b **integer** := 20;  **BEGIN**  **IF** a > b **THEN**  **RAISE** **NOTICE** 'a is greater than b';  **END** **IF**;  **IF** a < b **THEN**  **RAISE** **NOTICE** 'a is less than b';  **END** **IF**;  **IF** a = b **THEN**  **RAISE** **NOTICE** 'a is equal to b';  **END** **IF**;  **END** **$$**;  **Cara running : semua query di block** |

##### NULLIF

Fungsi NULLIF() digunakan untuk dapat mengembalikan [NULL](https://dosenit.com/sql/fungsi-null-sql-syntax-dan-contohnya) jika dua ekspresi sama, jika tidak ia mengembalikan ekspresi pertama.

|  |
| --- |
| **Syntax:**NULLIF(argument\_1,argument\_2); |

|  |
| --- |
| **SELECT** **NULLIF**(25, 95);// **output** 25  **SELECT** **NULLIF**(25, 25); //**output** **null**  **SELECT** **NULLIF**(50, 100); //**output** 50  **select** id,nama, **NULLIF** ( file\_name,**null**) **from** barang b ;  output: |

##### CASE

Perintah **CASE**sering digunakan untuk menampilkan nilai tertentu dari beberapa barisan data dengan syarat-syarat atau kondisi yang kita berikan.

|  |
| --- |
| ***CASE input\_expression***  ***WHEN when\_expression THEN result\_expression***  ***[ ...n ]***  ***[***  ***ELSE else\_result\_expression***  ***]***  ***END*** |

|  |
| --- |
| **SELECT** nama,  **CASE** nama  **WHEN** 'barang 10' **THEN** 'result barang 10'  **WHEN** 'barang 11' **THEN** 'result barang 11'  **ELSE** nama **END** **AS** nama\_alias  **FROM** barang **order** **by** nama **asc** |

##### CAST

Cast adalah sebuah fitur yang membolehkan kita untuk mengubah sebuah type data menjad tipe data lain.

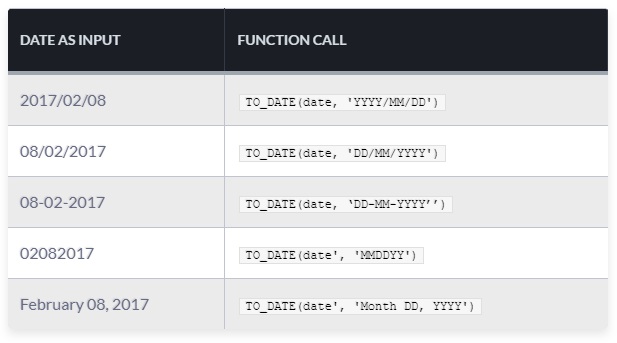
|  |
| --- |
| CAST(expression [AS](https://dosenit.com/php/keyword-as-php) datatype([*length*](https://dosenit.com/software/dbms/mysql/fungsi-string-length-pada-mysql))) |

**Nilai Parameter**

|  |  |
| --- | --- |
| **Value** | **Deskripsi** |
| expression | Required. Nilai yang akan dikonversi |
| datatype | Required. Jenis data yang akan digunakan untuk mengonversi ekspresi. Dapat berupa salah satu dari berikut: bigint, int, smallint, tinyint, bit, decimal, numeric, money, smallmoney, float, real, datetime, smalldatetime, char, varchar, text, nchar, nvarchar, ntext, binary, varbinary, atau image |
| (length) | Optional. Panjang tipe data yang dihasilkan (untuk char, varchar, nchar, nvarchar, binary dan varbinary) |

|  |
| --- |
| **SELECT** **CAST**(25.05 **AS** **varchar**); //output “25.05”  **SELECT** **CAST**('2017-03-08' **AS** **date**); // output 2017-03-08 |

##### String to Date



Contoh

|  |
| --- |
| **SELECT** **TO\_DATE**('2017-03-08' ,'YYYY-MM-DD'); |

##### Date To String

|  |
| --- |
| **SELECT** **to\_char**(created\_date , 'DD/MM/YYYY') **from** barang b |

##### COALESCE

Hanyalah untuk mengembalikan nilai non-null pertama yang ditemuinya saat membaca dari kiri ke kanan.

Jika semua ekspresi mengevaluasi ke null, maka fungsi COALESCE akan mengembalikan null.

|  |
| --- |
| COALESCE( expression1, expression2, ... expression\_n ) |

|  |
| --- |
| query> SELECT COALESCE(null, null, null, 'A', 'B');  Hasil: 'A'    query > SELECT COALESCE('A', 'B', null, 'C', 'D');  Hasil: 'A'    query > SELECT COALESCE(null, 1, 2, 3, null, 4);  Hasil: 1    query > SELECT COALESCE(null, 'dumetschool.com', 'karyamurid.com');  Hasil: 'dumetschool.com'    query > SELECT COALESCE(null, null, null, null, null);  Hasil: NULL |

|  |
| --- |
| Misal:    **select** id,nama, **COALESCE**(file\_name , 'suppliaer diganti dengan null') **from** barang b ;  **output**: |

#### Mengetahui Perintah Join

Join adalah cara untuk menghubungkan data yang diambil dari tabel-tabel melalui sebuah kolom yang menghubungkan mereka.

**Mengapa join itu penting?**

**Join** memperbolehkan kita untuk mengambil data dari beberapa tabel melalui satu *query*.

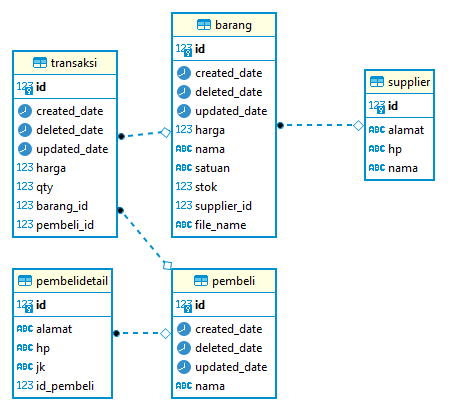
**Ada berapa jenis join?**

* **(INNER) JOIN:** Mengembalikan record yang memiliki nilai yang **cocok** di kedua tabel.
* **LEFT (OUTER) JOIN:** Mengembalikan semua record dari tabel **kiri**, dan record yang cocok dari tabel kanan
* **RIGHT (OUTER) JOIN:** Mengembalikan semua record dari tabel **kanan**, dan record yang cocok dari tabel kiri
* **FULL (OUTER) JOIN:** Mengembalikan semua record ketika ada kecocokan di tabel **kiri atau kanan**

**Berikut contoh query Join:**

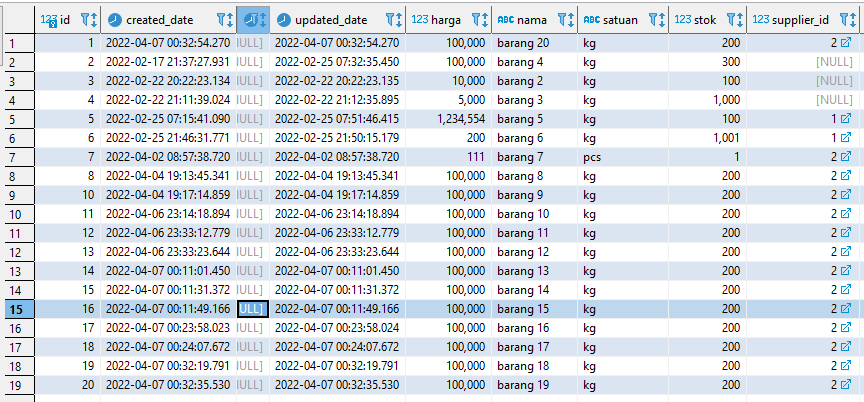
**Studi kasus ERD : penjualan barang**

****

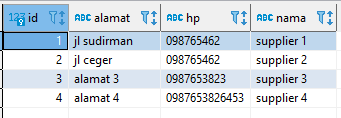
**Silahkan import data sql.**

Berikut data pada tabel-tabel.

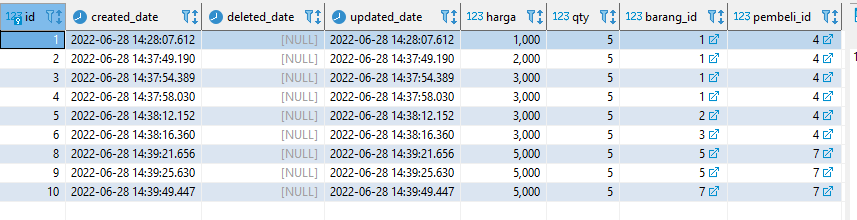
**Tabel barang**



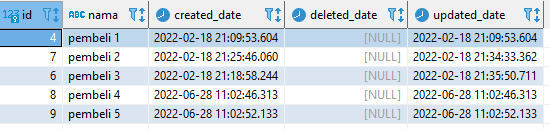
**Tabel supplier**



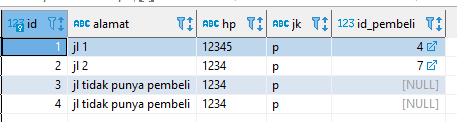
**Tabel Transaksi**



**Tabel Pembeli**



**Tabel Pembeli Detail**



|  |
| --- |
| **Inner JOIN:**  SQL INNER JOIN  **2 tabel**  **select** \* **from** pembeli p  **inner** **join** pembelidetail p2  **on** p.id=p2.id\_pembeli;  **select** \* **from** pembeli p  **join** pembelidetail p2  **on** p.id=p2.id\_pembeli;    **3 tabel**  **select** p.id **as** idtabelpembeli, p2.id **as** idtabelpembelidetail, t.id **as** idtransaksi, p.nama **as** namapembeli **from** pembeli p  **join** pembelidetail p2 **on** p.id=p2.id\_pembeli  **join** transaksi t **on** t.pembeli\_id = p.id ; |

|  |
| --- |
| SQL RIGHT JOIN  **2 tabel**  **select** \* **from** pembeli p  **right** **join** pembelidetail p2  **on** p.id=p2.id\_pembeli; |

|  |
| --- |
| SQL LEFT JOIN  **2 tabel**  **select** \* **from** pembeli p  **left** **join** pembelidetail p2  **on** p.id=p2.id\_pembeli; |

|  |
| --- |
| SQL FULL OUTER JOIN  **select** \* **from** pembeli p  **full** **join** pembelidetail p2  **on** p.id=p2.id\_pembeli; |

#### Mengetahui Perintah Merge

**Apa itu merge Update ?**

Merge Update/Upsert adalah perintah standard SQL untuk melakukan INSERT record baru kalau kondisi tidak memenuhi, atau UPDATE/DELETE existing record kalau kondisinya memenuhi.

Dengan kata lain Merge Update/Upsert ini adalah satu perintah SQL yang **menggabungkan** INSERT, UPDATE, DELETE dalam satu perintah.

Perintah ini menyelesaikan permasalahan umum di dunia Software Engineering yang biasanya membutuhkan kasus yang serupa, yaitu :

 INSERT data jika belum ada, UPDATE data jika sudah ada.

[**Apakah semua versi database mengakomodasi ini ?**](https://www.huzefril.com/posts/database/sql-merge-update/#apakah-semua-versi-database-mengakomodasi-ini-)

Tentu saja tidak, walaupun sudah dimasukkan dalam standar [ANSI SQL:2003](https://en.wikipedia.org/wiki/SQL:2003), namun tidak semua relational database mengakomodasinya.

Walaupun beberapa database tidak mengakomodasinya, tapi mereka mempunyai perintah khusus yang mirip dengan standard MERGE INTO …

Berikut datar database yang mengakomodasinya baik yang mengikuti standar atau mempunyai perintah yang mirip :

| **Database** | **Disupport sejak Versi** | **Perintah** |
| --- | --- | --- |
| Oracle | Oracle 9i | MERGE INTO … |
| SQL Server | SQL Server 2008 | MERGE INTO … |
| PostgreSQL | PostgreSQL 9.5 | INSERT … ON CONFLICT … DO UPDATE SET … |
| MySQL | MySQL 5.6 | INSERT … ON DUPLICATE KEY REPLACE … |

**PostgreSQL**

|  |
| --- |
| Syarat: column harus bersipat unique/ unik: gunakan perintah ini untuk menambahkan unique di column nama:  **ALTER** **TABLE** barang  **ADD** **UNIQUE** (nama);  **Untuk drop unique:**  **ALTER** **TABLE** barang  **DROP** **CONSTRAINT** barang\_nama\_key;  **Contoh 1: insert**  **INSERT** **INTO** barang (nama, satuan)  **VALUES**('barang3','pcs')  **ON** **CONFLICT** (nama)  **DO** **NOTHING**;    Gambar diatas diperlihatkan result = 1, artinya barang3 berhasil di insert.  Jika barang dengan nama yang sama = barang3 diinsert ulang , maka tidak akan diinsert. Karena ada perintah **DO** **NOTHING**; contoh    **Contoh 2 : insert dan update**  Jika “barang3” ada di column nama, maka lakukan update else insert  **INSERT** **INTO** barang (nama)  **VALUES**('barang3')  **ON** **CONFLICT** (nama)  **DO**  **UPDATE** **SET** nama = 'barang update', harga = 123;        **jika ‘barang4’ : do insert** |

<https://www.huzefril.com/posts/database/sql-merge-update/>

https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-upsert/#:~:text=In%20relational%20databases%2C%20the%20term,combination%20of%20update%20or%20insert).

#### Mengetahui Perintah Call

Digunakan untuk memanggil prosedure: next time akan dibahas terpisah

#### Mengetahui Perintah Explain Plan

Pada awalnya, query dapat dieksekusi dengan baikÃ‚Â  lancar, namun seiring dengan berjalannya waktu, eksekusi query terasa lambat entah karena query yang tidak optimal atau data yang semakin besar. Jika sudah demikian, maka mau tidak mau kita harus melakukan optimasi query . Kenapa? karena hal ini akan berdampak signifikan bagi kelangsungan aplikasi kita.

Digunakan untuk :  optimalisasi sebuah query. Sangat penting ketika jumlah data menjadi besar. lalu bagaimana caranya kita mengetahui kalau query yang kita gunakan sudah baik.

Ini akan kelihatan untuk query join many tabel.

Contoh

|  |
| --- |
|  |

<https://www.postgresql.org/docs/current/sql-explain.html>

<https://petainteraktif.wordpress.com/2007/07/02/optimalisasi-query-dengan-fungsi-explain/>

<https://www.postgresql.org/docs/current/sql-explain.html>

https://jagowebdev.com/optimasi-query-mysql-buku-ebook/

#### Mengetahui Lock Table

Lock Table PostgreSQL didefinisikan sebagai Lock Table untuk akses dari pengguna, kita dapat mengunci tabel dari akses baca atau akses tulis. Kunci sangat berguna dan penting di PostgreSQL untuk mencegah pengguna memodifikasi satu baris atau semua tabel.

**How to Lock Table in PostgreSQL?**

|  |
| --- |
| **lock** **table** barang **IN** **ACCESS** **SHARE** **MODE**;  **jika ingin select tabel barang ,akan eror, karena sudah ke kunci**    **Selama belum di commit atau di rollback transaction tidak dapat diakses di tabel tersebut..**  **Melihat activity :**  **select** \* **from** pg\_stat\_activity ; |

**How to UNLock Table in PostgreSQL?**

|  |
| --- |
| **commit**;  atau rollback; |

<https://www.educba.com/postgresql-lock-table/>

https://www.postgresql.org/docs/current/sql-lock.html

#### Mengetahui Perintah Sub Query

**Apa itu sub query ?**

adalah query dalam query SQL lain dan tertanam dalam klausa WHERE. Sebuah subquery digunakan untuk mengembalikan data yang akan digunakan dalam query utama sebagai syarat untuk lebih membatasi data yang akan diambil. Subqueries dapat digunakan dengan SELECT, INSERT, UPDATE, dan DELETE statements bersama dengan operator seperti =, <,>,> =, <=, IN, BETWEEN dll.

Ada beberapa aturan yang subqueries harus mengikuti:

* Subqueries harus tertutup dalam tanda kurung.
* Sebuah subquery hanya dapat memiliki satu kolom pada klausa SELECT, kecuali beberapa kolom yang di query utama untuk subquery untuk membandingkan kolom yang dipilih.
* ORDER BY tidak dapat digunakan dalam subquery, meskipun permintaan utama dapat menggunakan ORDER BY.GROUP BY dapat digunakan untuk melakukan fungsi yang sama seperti ORDER BY dalam subquery.
* Subqueries yang kembali lebih dari satu baris hanya dapat digunakan dengan beberapa value operator, seperti operator IN.
* Daftar SELECT tidak bisa menyertakan referensi ke nilai-nilai yang mengevaluasi ke BLOB, ARRAY, CLOB, atau NCLOB.
* Sebuah subquery tidak dapat segera tertutup dalam fungsi set.
* Operator BETWEEN tidak dapat digunakan dengan subquery;Namun, BETWEEN dapat digunakan dalam subquery.

Sintaks dasar dari sub query adalah sebagai berikut:

|  |
| --- |
| SELECT column\_name [, column\_name ]  FROM   table1 [, table2 ]  WHERE  column\_name OPERATOR        (SELECT column\_name [, column\_name ]        FROM table1 [, table2 ]        [WHERE]) |

**Contoh:**

|  |
| --- |
| **SELECT** \* **FROM** barang b **WHERE** ID **in**  (**SELECT** ID **FROM** barang **WHERE** id > 4) ; |

https://sis.binus.ac.id/2018/09/10/sql-subquery/

#### Loop dan IF - Prosedure

Loop mendefinisikan loop tanpa syarat yang mengeksekusi blok kode berulang kali sampai diakhiri oleh pernyataan exit  atau return .

|  |
| --- |
| **loop**  statements;  **end** **loop**;  **contoh 1**:  **DO**  **$do$**  **BEGIN**  **FOR** i **IN** 1..25 **loop**  **INSERT** **INTO** public.barang  (nama, satuan) -- declare target columns!  **values**(i, 'PCS');  **END** **LOOP**;  **END**  **$do$**; |

**IF**

|  |
| --- |
| **loop**  statements;  **if** **condition** **then**  **exit**;  **end** **if**;  **end** **loop**;  **CONTOH 2:**  **DO**  **$do$**  **BEGIN**  **FOR** i **IN** 1..25 **loop**  **if** i !=5 **then**  **INSERT** **INTO** public.barang  (nama, satuan) -- declare target columns!  **values**(i, 'PCS');  **end** **if**;  **END** **LOOP**;  **END**  **$do$**;  ANGKA 5 TIDAK TERINSERT |

**Loop if else**

|  |
| --- |
| **CONTOH 3:**  **DO**  **$do$**  **BEGIN**  **FOR** i **IN** 1..25 **loop**  **if** i <=5 **then**  **INSERT** **INTO** public.barang  (nama, satuan) -- declare target columns!  **values**(i, 'PCS');  **else**  **exit**;  **end** **if**;  **END** **LOOP**;  **END**  **$do$**; |

**IF ELSE IF ELSE END**

|  |
| --- |
| **Contoh 4:**  **DO**  **$do$**  **BEGIN**  **FOR** i **IN** 1..25 **loop**  **if** i !=5 **then**  **INSERT** **INTO** public.barang  (nama, satuan) -- declare target columns!  **values**(i, 'PCS');  **elseif** (i !=3) **then**  **INSERT** **INTO** public.barang  (nama, satuan) -- declare target columns!  **values**(i, 'PCS');  **else**  **exit**;  **end** **if**;  **END** **LOOP**;  **END**  **$do$**; |

https://www.w3schools.com/sql/sql\_case.asp

#### Index

https://i-3.co.id/index-pada-database/#:~:text=Dengan%20index%2C%20kita%20dapat%20dengan,proses%20pencarian%20(query)%20data.

#### Referensi

<https://www.w3schools.com/sql/sql_delete.asp>

https://www.codepolitan.com/tujuh-teknik-join-di-sql-596c537f0deb3/

## Data Control Language (DCL)

Berfungsi untuk mengatur hak akses apa saja yang dimiliki oleh pengguna. DCL terbagi menjadi dua jenis, yaitu:

|  |  |
| --- | --- |
|  |  |
| Grant | perintah yang dapat dilakukan oleh admin database untuk memberikan akses kepada user yang lainnya. |
| Revoke | untuk mencabut atau menghapus hak akses bagi user tertentu sesuai dengan keinginan admin database |
|  |  |

#### Schema

**Apa Itu Schema?**

Schema adalah salah satu object di database yang berisi sekumpulan dari beberapa tabel, view, index, data type, operator, dan function.  
  
Schema dapat dianalogikan sebagai direktori dalam sebuah sistem operasi, tetapi di dalam schema tidak boleh ada schema lagi (nested schema).

**Sintak Dasar**

|  |
| --- |
| CREATE SCHEMA schema\_name;  Contoh :  **CREATE** **SCHEMA** schemadibimbing;  **Setelah di create, akan tampil di mana ?**    **Bagaimana menggunakan schema di DBEVER ?**    **Membuat Table Di Schema**  Sebenarnya sama dengan fungsi create sebelumnya: yang berbeda hanya ditambahkan namaschem.namatabel. contoh :  CREATE TABLE **schema\_name**.table\_name (  ...  );  **Menghapus Schema**  DROP SCHEMA myschema [CASCADE];  **DROP** **SCHEMA** schemadibimbing; |

http://nano-tutorial.blogspot.com/2015/05/membuat-schema-di-postgresql.html

#### Role/USER

Role/user merupakan user yang diberikan akses pada database.

|  |
| --- |
| Create ROLE  **CREATE** **ROLE** roledibimbing **WITH** LOGIN **ENCRYPTED** **PASSWORD** 'password'; |

|  |
| --- |
| Drop Role  DROP ROLE demorole1; |

|  |
| --- |
| Create a superuser  **CREATE** **ROLE** mysuperuser **WITH** SUPERUSER CREATEDB CREATEROLE LOGIN **ENCRYPTED** **PASSWORD** 'password'; |

**Cara melihat role?**

|  |
| --- |
| **Klik nama database – view database** |

Second, [create a new role](https://www.postgresqltutorial.com/postgresql-roles/) called jim with the LOGIN and PASSWORD attributes:

|  |
| --- |
| CREATE ROLE jim  LOGIN  PASSWORD 'securePass1'; |

Third, grant all privileges on the film table to the role jim:

|  |
| --- |
| GRANT ALL  ON film  TO jim; |

Finally, grant the SELECT privilege on the actor table to the role jim:

|  |
| --- |
| **GRANT** **SELECT**  **ON** barang  **TO** roledibimbing;  Melihat role user yang ditambahkan pada suatu tabel |

Step 2. Revoke privileges from a role

To revoke the SELECT privilege on the actor table from the role jim, you use the following statement:

|  |
| --- |
| REVOKE SELECT  ON actor  FROM jim; |

To revoke all privileges on the film table from the role jim, you use REVOKE statement with the ALL option like this:

|  |
| --- |
| REVOKE ALL  ON film  FROM jim; |

https://docs.rackspace.com/support/how-to/postgresql-creating-and-dropping-roles/

#### Grant

Bagaimana cara memberikan akses/grant ke pengguna di PostgreSQL?

|  |
| --- |
| 1. **Grant CONNECT to the database:**   GRANT CONNECT ON DATABASE database\_name TO username;  **Contoh :**  **GRANT** **CONNECT** **ON** **DATABASE** dibimbing1 **TO** postgres; |

|  |
| --- |
| 1. **Grant USAGE on schema:**   GRANT USAGE ON SCHEMA schema\_name TO username;  **Contoh:**  **GRANT** **USAGE** **ON** **SCHEMA** public **TO** postgres;  **Yang mana skcema itu ?** |

|  |
| --- |
| 1. **Grant on all tables for DML statements: SELECT, INSERT, UPDATE, DELETE:**   GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA schema\_name TO username;  **Contoh :**  **GRANT** **SELECT**, **INSERT**, **UPDATE**, **DELETE** **ON** **ALL** **TABLES** **IN** **SCHEMA** public **TO** postgres; |

|  |
| --- |
| 1. **Grant all privileges on all tables in the schema:**   GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA schema\_name TO username;  **Contoh** :  **GRANT** **ALL** **PRIVILEGES** **ON** **ALL** **TABLES** **IN** **SCHEMA** public **TO** postgres;    Contoh **one tabel:**  Syntax:GRANT privilege\_list | ALL ON table\_name TO role\_name;  Contoh : **GRANT SELECT ON barang TO postgres;**    **In this syntax:**   * First, specify the privilege\_list that can be [SELECT](https://www.postgresqltutorial.com/postgresql-select/), [INSERT](https://www.postgresqltutorial.com/postgresql-insert/), [UPDATE](https://www.postgresqltutorial.com/postgresql-update/), [DELETE](https://www.postgresqltutorial.com/postgresql-delete/), [TRUNCATE](https://www.postgresqltutorial.com/postgresql-truncate-table/), etc. You use the ALL option to grant all privileges on a table to the role. * Second, specify the name of the table after the ON keyword. * Third, specify the name of the role to which you want to grant privileges.   **Bagaimana menampilkan grant pada sebuah tabel ?**  **select** grantee, privilege\_type **from** information\_schema.role\_table\_grants **where** table\_name='barang';  **atau dapat dilihat di DBEVER**  klik kanan tabel -pilih view tabel -pilih permission |

|  |
| --- |
| 1. **Grant all privileges on all sequences in the schema:**   GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA schema\_name TO username;  **GRANT** **ALL** **PRIVILEGES** **ON** **ALL** **SEQUENCES** **IN** **SCHEMA** public **TO** postgres;  **GRANT** **SELECT** **ON** **SEQUENCE** public.barang\_id\_seq **TO** postgres; |

|  |
| --- |
| 1. **Grant all privileges on the database:**   **Contoh all tabel:**  Syntax:GRANT ALL PRIVILEGES ON DATABASE database\_name TO username;  **Contoh :GRANT** **ALL** **PRIVILEGES** **ON** **DATABASE** dibimbing1 **TO** postgres; |

<https://tableplus.com/blog/2018/04/postgresql-how-to-grant-access-to-users.html>

https://www.postgresql.org/docs/current/sql-grant.html

#### Revoke

Menghapus atau mencabut kembali izin akses user yang sebelumnya telah diberikan.

|  |
| --- |
| Syntax  REVOKE [ GRANT OPTION FOR ]  { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER }  [, ...] | ALL [ PRIVILEGES ] }  ON { [ TABLE ] ***table\_name*** [, ...]  | ALL TABLES IN SCHEMA ***schema\_name*** [, ...] }  FROM ***role\_specification*** [, ...]  [ GRANTED BY ***role\_specification*** ]  [ CASCADE | RESTRICT ] |

|  |
| --- |
| Revoke Insert schema:  REVOKE INSERT ON barang FROM PUBLIC;  Revoke Insert tabel:  **REVOKE** **INSERT** **ON** barang **FROM** postgres;  Revoke All schema:  REVOKE ALL PRIVILEGES ON barang FROM public;  Revoke All table barang FROM USER POSTGRES:  REVOKE ALL PRIVILEGES ON barang FROM postgres; |

## Transaction Control Language (TCL)

Transaction Control atau singkatan dari TCL yaitu perintah untuk kendali terhadap suatu perubahan data di Database contoh menyinpan data secara permanent yang kita kenal sebagai commit dan untuk mengembalikan ke kondisi sebelumnya yaitu rollback dalam suatu transaksi.

#### Commit

mengakhiri semua transaksi dan menjadikannya sebagai perubahan permanen.

Grafik perbandingan

|  |  |  |
| --- | --- | --- |
| **Dasar untuk Perbandingan** | **MELAKUKAN** | **ROLLBACK** |
| Dasar | COMMIT memvalidasi modifikasi yang dibuat oleh transaksi saat ini. | ROLLBACK menghapus modifikasi yang dibuat oleh transaksi saat ini. |
| Efek | Setelah mengeksekusi pernyataan COMMIT, transaksi tidak dapat ROLLBACK. | Setelah ROLLBACK dieksekusi, basis data mencapai keadaan sebelumnya, yaitu sebelum eksekusi pernyataan pertama transaksi. |
| Kejadian | COMMIT terjadi ketika transaksi dijalankan dengan sukses. | ROLLBACK terjadi ketika tra nsaksi dibatalkan di tengah-tengah eksekusi. |

|  |
| --- |
|  |

https://i-3.co.id/mengenal-fitur-commit-rollback-dan-save-point-pada-postgresql/

#### Rollback

Membatalkan seluruh perubahan data DML yang masih menggantung dengan menggunakan statement ROLLBACK. Hal-hal yang mengikuti ROLLBACK:

* Seluruh perubahan data dibatalkan.
* Keadaan data sebelumnya dikembalikan seperti semula.
* Penguncian terhadap baris-baris data yang diubah akan dibebaskan, baris-baris data tersebut sekarang tersedia kembali bagi user lain untuk melakukan perubahan data berikutnya.

Contoh:

Ketika berusaha menghapus satu record pada table Barang, secara tidak sengaja menghapus seluruh table. Betulkan kesalahan tersebut, lalu jalankan perintah yang sesuai, dan buatlah seluruh perubahan data menjadi permanen.

|  |
| --- |
| **update** barang **set** nama ='barang update 101' **where** id =2;  **rollback**;  cara menjalankan : blok semua querynya    Sebenarnya pada query baris pertama, sudah dilakukan update pada column nama, namun karena ada fungsi rollback di baris kedua. Sehingga column nama akan di set dengan value semula.  Output :  Anggap saja : output berikut data awal nya.    Maka ketika dijalankan query rollback diatas: outputnya akan sama: |

http://www.kuliahkomputer.com/2020/06/contoh-commit-savepoint-dan-rollback.html

#### Savepoint

Perintah yang di gunakan untuk menyimpan sebuah titik chek point dari suatu database, sehingga jika terjadi kesalahan kita dapat memanggil kembali chek point tersebut agar data yang sudah kita inputkan dapat kembali seperti saat sebelum save point tersebut dieksekusi. save point ini sama seperti **membackup data.**

Savepoint adalah sebuah titik restore, dimana dengan menggunakan kode SQL ROLLBACK TO SAVEPOINT, kita bisa kembali kedalam keadaan ketika kita menyimpan SAVEPOINT tersebut. SAVEPOINT tidak bisa di eksekusi.  Karena bersifat pasif, dan hanya bisa aktif jika ada perintah SQL yang lain (seperti ROLLBACK TO SAVEPOINT).

|  |
| --- |
| **SAVEPOINT BARANG\_updatetest;** |

|  |
| --- |
| Anggap data id = 2    Ketika ingin membuat savepoint dan runningnya : semua query di block. Diawali oleh begin dan diakhir oleh commit.  Contoh:  **BEGIN**;  **SAVEPOINT** BARANG\_UPDATE;  **update** barang **set** nama ='barang update 1011' **where** id =2;  **ROLLBACK** **TO** BARANG\_UPDATE;  **commit**;  **Maka outputnya adalah : nama tetapa akan di set ke semula :”barang update 10”. Itu dikarenakan ada fungsi ROLLBACK** **TO** BARANG\_UPDATE**;** |

#### Set Transaction

**Apa itu Transaction ?**

Transaction adalah urutan operasi SQL yang diterapkan pada database (yang bisa lebih dari satu pernyataan SQL) yang diperlakukan sebagai satu logika operasi, semua pernyataan yang akan diterapkan harus **berhasil** semuanya, jika ada satu saja pernyataan yang **gagal** maka keseluruhan operasi harus ikut di **gagalkan** .

**Contoh:**

Ketika kita ingin melakukan transfer uang dari akun A ke akun B sebanyak 50000 maka logikanya kita harus melakukan update pada akun A untuk mengurangi uang sebanyak 50000 dan kita juga harus melakukan update pada akun B untuk menambah uang sebanyak 50000. **Jika salah satu proses ada yang gagal maka kita harus membatalkan semua operasi yang telah dilakukan.**

**Penggunaan transaction pada postgresql menggunakan command line ?**

1. Untuk memulai transaction bisa kita lakukan command **begin** terlebih dahulu (bisa juga **begin transaction**)

|  |
| --- |
| BEGIN; |

1. lalu kita bisa masukkan satu persatu logika operasi sql yang akan kita terapkan:

|  |
| --- |
| **BEGIN**;  **UPDATE** barang **SET** harga = 4500 **WHERE** id = 3;  **UPDATE** barang **SET** harga = 6000 **WHERE** id = 4;  **commit**; |

https://dhimas-hary.medium.com/mengenal-transaction-di-sql-dan-contoh-penggunaannya-557de028aa97

#### Debugging

**Kasus**:

debug query yang proses loadingnya lambat untuk tarik data 3 bulan, 6 bulan, 12 bulan.

Cara nya ? silahkan di pelajari diinternet .

## Type Data

#### Numeric Data Types

|  |  |  |
| --- | --- | --- |
| **DATA TYPE** | **FROM** | **TO** |
| bigint | -9,223,372,036,854,775,808 | 9,223,372,036,854,775,807 |
| int | -2,147,483,648 | 2,147,483,647 |
| smallint | -32,768 | 32,767 |
| tinyint | 0 | 255 |
| bit | 0 | 1 |
| decimal | -10^38 +1 | 10^38 -1 |
| numeric | -10^38 +1 | 10^38 -1 |
| money | -922,337,203,685,477.5808 | +922,337,203,685,477.5807 |
| smallmoney | -214,748.3648 | +214,748.3647 |

#### Approximate Numeric Data Types

|  |  |  |
| --- | --- | --- |
| **DATA TYPE** | **FROM** | **TO** |
| float | -1.79E + 308 | 1.79E + 308 |
| real | -3.40E + 38 | 3.40E + 38 |

#### Date and Time Data Types

|  |  |  |
| --- | --- | --- |
| **DATA TYPE** | **FROM** | **TO** |
| datetime | Jan 1, 1753 | Dec 31, 9999 |
| smalldatetime | Jan 1, 1900 | Jun 6, 2079 |
| date | Stores a date like June 30, 1991 | |
| time | Stores a time of day like 12:30 P.M. | |

#### Character Strings Data Types

|  |  |
| --- | --- |
| **Sr.No.** | **DATA TYPE & Description** |
| 1 | **char**  Maximum length of 8,000 characters.( Fixed length non-Unicode characters) |
| 2 | **varchar**  Maximum of 8,000 characters.(Variable-length non-Unicode data). |
| 3 | **varchar(max)**  Maximum length of 2E + 31 characters, Variable-length non-Unicode data (SQL Server 2005 only). |
| 4 | **text**  Variable-length non-Unicode data with a maximum length of 2,147,483,647 characters. |

#### Unicode Character Strings Data Types

|  |  |
| --- | --- |
| **Sr.No.** | **DATA TYPE & Description** |
| 1 | **nchar**  Maximum length of 4,000 characters.( Fixed length Unicode) |
| 2 | **nvarchar**  Maximum length of 4,000 characters.(Variable length Unicode) |
| 3 | **nvarchar(max)**  Maximum length of 2E + 31 characters (SQL Server 2005 only).( Variable length Unicode) |
| 4 | **ntext**  Maximum length of 1,073,741,823 characters. ( Variable length Unicode ) |

## Operator

#### SQL Arithmetic Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + (Addition) | Adds values on either side of the operator. | a + b will give 30 |
| - (Subtraction) | Subtracts right hand operand from left hand operand. | a - b will give -10 |
| \* (Multiplication) | Multiplies values on either side of the operator. | a \* b will give 200 |
| / (Division) | Divides left hand operand by right hand operand. | b / a will give 2 |
| % (Modulus) | Divides left hand operand by right hand operand and returns remainder. | b % a will give 0 |

#### SQL Comparison Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Checks if the values of two operands are equal or not, if yes then condition becomes true. | (a = b) is not true. |
| != | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. | (a != b) is true. |
| <> | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. | (a <> b) is true. |
| > | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. | (a > b) is not true. |
| < | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. | (a < b) is true. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | (a >= b) is not true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. | (a <= b) is true. |
| !< | Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true. | (a !< b) is false. |
| !> | Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true. | (a !> b) is true. |

#### SQL Logical Operators

|  |  |
| --- | --- |
| **Sr.No.** | **Operator & Description** |
| 1 | **ALL**  The ALL operator is used to compare a value to all values in another value set. |
| 2 | **AND**  The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause. |
| 3 | **ANY**  The ANY operator is used to compare a value to any applicable value in the list as per the condition. |
| 4 | **BETWEEN**  The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value. |
| 5 | **EXISTS**  The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criterion. |
| 6 | **IN**  The IN operator is used to compare a value to a list of literal values that have been specified. |
| 7 | **LIKE**  The LIKE operator is used to compare a value to similar values using wildcard operators. |
| 8 | **NOT**  The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. **This is a negate operator.** |
| 9 | **OR**  The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause. |
| 10 | **IS NULL**  The NULL operator is used to compare a value with a NULL value. |
| 11 | **UNIQUE**  The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates). |

#### Expressions

Syntax

|  |
| --- |
| SELECT column1, column2, columnN  FROM table\_name  WHERE [CONDITION|EXPRESSION];  **Contoh**  SELECT (15 + 6) AS ADDITION  SELECT COUNT(\*) AS "RECORDS" FROM CUSTOMERS;  SELECT CURRENT\_TIMESTAMP;  SELECT GETDATE(); |

# Perintah Dasar SQL

|  |  |
| --- | --- |
| **SELECT** | SELECT kolom1, kolom2, … FROM nama\_tabel; SELECT \* FROM nama\_tabel; |
| **SELECT DISTINCT** | digunakan untuk mengembalikan hanya nilai yang berbeda dari dalam sebuah tabel, dengan kata lain semua record duplikat (record dengan nilai yang sama) yang terdapat pada tabel akan di anggap sebagai satu record/nilai  SELECT DISTINCT kolom1, kolom2, … FROM nama\_tabel; |
| **WHERE** | SELECT kolom1, kolom2, … FROM nama\_tabel WHERE kondisi; |
| **(operator) AND, OR dan NOT** | SELECT kolom1, kolom2, … FROM nama\_tabel WHERE kondisi1 AND kondisi2 AND kondisi3; SELECT kolom1, kolom2, … FROM nama\_tabel WHERE kondisi1 OR kondisi2 OR kondisi3 …; |
| **ORDER BY** | SELECT kolom1, kolom2, … FROM nama\_tabel ORDER BY column DESC;  SELECT nis, nama FROM siswa ORDER BY tahun\_lahir DESC; |
| **INSERT INTO** | INSERT INTO nama\_tabel VALUES (nilai1, nilai2, nilai3, …); INSERT INTO nama\_tabel (kolom1, kolom2) VALUES (nilai1, nilai2); |
| **UPDATE** | UPDATE nama\_tabel SET kolom1 = nilai1, kolom2 = nilai2, … WHERE kondisi; |
| **DELETE** | DELETE FROM table\_name WHERE condition; |
| **MIN()** | Fungsi MIN() merupakan perintah dasar SQL yang di gunakan untuk mendapatkan nilai terkecil dari suatu kolom  SELECT MIN(nama\_kolom) FROM nama\_tabel WHERE kondisi; |
| **MAX()** | SELECT MAX(nama\_kolom) FROM nama\_tabel WHERE kondisi; |
| **COUNT()** | mendapatkan jumlah hitungan record yang memenuhi suatu kriteria.  SELECT COUNT(nama\_kolom) FROM nama\_tabel WHERE kondisi; |
| **AVG()** | di gunakan untuk mendapatkan rata-rata record yang memenuhi suatu kriteria.  SELECT AVG(nama\_kolom) FROM nama\_tabel WHERE kondisi; |
| **SUM()** | merupakan perintah dasar SQL yang di gunakan untuk mendapatkan jumlah record yang memenuhi suatu kriteria  SELECT SUM(nama\_kolom) FROM nama\_tabel WHERE kondisi; |
| **INNER JOIN** | INNER JOIN merupakan perintah dasar SQL yang di gunakan untuk menggabungkan beberapa tabel dan mengambil nilai yang cocok (identik) di antara kedua tabel tersebut.  SELECT nama\_kolom1, nama\_kolom2, … FROM tabel1 INNER JOIN tabel2 ON tabel1.nama\_kolom = tabel2.nama\_kolom; |
| **LEFT JOIN** | LEFT JOIN merupakan perintah dasar SQL yang di gunakan untuk menggabungkan beberapa tabel dan mengambil nilai yang cocok (identik) di antara kedua tabel tersebut dan nilai lain dari tabel pada ruas kiri meskipun tak ada nilai yang cocok dengan tabel pada ruas kanan.  SELECT nama\_kolom1, nama\_kolom2, … FROM tabel1 LEFT JOIN tabel2 ON tabel1.nama\_kolom = tabel2.nama\_kolom; |

# Kapan Menggunakan NoSQL

***Laju pengembangan dengan database NoSQL bisa jauh lebih cepat dibandingkan dengan database SQL***

Karena database NoSQL sering kali memungkinkan developer untuk mengontrol struktur data, mereka sangat cocok dengan praktik pengembangan Agile modern berdasarkan sprint, iterasi cepat, dan dorongan kode yang sering. Ketika pengembang harus meminta administrator database SQL untuk mengubah struktur database dan kemudian membongkar dan memuat ulang data, itu dapat memperlambat pengembangan.

***Struktur berbagai bentuk data lebih mudah ditangani dan dikembangkan dengan database NoSQL***

Database NoSQL seringkali lebih cocok untuk menyimpan dan memodelkan data terstruktur, semi-terstruktur, dan tidak terstruktur dalam satu database.Database NoSQL sering menyimpan data dalam bentuk yang mirip dengan objek yang digunakan dalam aplikasi, sehingga mengurangi kebutuhan untuk penerjemahan dari bentuk data yang disimpan ke dalam bentuk data yang diambil dalam kode program.

***Jumlah data di banyak aplikasi tidak dapat dilayani dengan harga terjangkau oleh database SQL***

Database NoSQL dibuat untuk menangani big data sebagai bagian dari arsitektur fundamentalnya. Rekayasa tambahan tidak diperlukan seperti saat database SQL digunakan untuk menangani aplikasi skala web. Jalur menuju skalabilitas data sangat mudah dan dipahami dengan baik,Basis data NoSQL sering kali didasarkan pada strategi skala-keluar, yang membuat penskalaan ke volume data besar jauh lebih murah daripada saat menggunakan pendekatan peningkatan skala yang dilakukan basis data SQL.

# Praktek DDL dan DML

|  |  |
| --- | --- |
| **CREATE DATABASE** | CREATE DATABASE DatabaseName;  Contoh  CREATE DATABASE testDB; |
| **DROP DATABASE** | DROP DATABASE DatabaseName;  Contoh  DROP DATABASE testDB; |
| **SELECT Database** | USE DatabaseName;  SHOW DATABASES  USE dbName; |
| **CREATE TABLE** | CREATE TABLE table\_name(  column1 datatype,  column2 datatype,  column3 datatype,  .....  columnN datatype,  PRIMARY KEY( one or more columns )  );  Contoh  CREATE TABLE CUSTOMERS(  ID INT NOT NULL,  NAME VARCHAR (20) NOT NULL,  AGE INT NOT NULL,  ADDRESS CHAR (25) ,  SALARY DECIMAL (18, 2),  PRIMARY KEY (ID)  ) |
| **DROP or DELETE Table** | DROP TABLE table\_name;  contoh  DESC CUSTOMERS; |
| **INSERT** | INSERT INTO TABLE\_NAME (column1, column2, column3,…columnN)  VALUES (value1, value2, value3,…valueN);  Contoh  INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)  VALUES (1, ‘Ramesh’, 32, ‘Ahmedabad’, 2000.00 );  INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)  VALUES (2, ‘Khilan’, 25, ‘Delhi’, 1500.00 );  INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)  VALUES (3, ‘kaushik’, 23, ‘Kota’, 2000.00 );  INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)  VALUES (4, ‘Chaitali’, 25, ‘Mumbai’, 6500.00 );  INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)  VALUES (5, ‘Hardik’, 27, ‘Bhopal’, 8500.00 );  Insert banyak  INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)  VALUES (7, 'maya', 22, 'MP', 4500.00 ),  (8, 'maya', 22, 'MP', 4500.00 ); |
| **SELECT** | SELECT column1, column2, columnN FROM table\_name;  Contoh  SELECT \* FROM table\_name;  SELECT ID, NAME, SALARY FROM CUSTOMERS; |
| **WHERE** | SELECT column1, column2, columnN  FROM table\_name  WHERE [condition]  Contoh  SELECT ID, NAME, SALARY  FROM CUSTOMERS  WHERE SALARY > 2000;  SELECT ID, NAME, SALARY  FROM CUSTOMERS  WHERE NAME = ‘Hardik’; |
| **AND** & **OR** | SELECT column1, column2, columnN  FROM table\_name  WHERE [condition1] AND [condition2]…AND [conditionN];  SELECT ID, NAME, SALARY  FROM CUSTOMERS  WHERE SALARY > 2000 AND age < 25;  SELECT ID, NAME, SALARY  FROM CUSTOMERS  WHERE SALARY > 2000 OR age < 25; |
|  |  |
| **UPDATE** | UPDATE table\_name  SET column1 = value1, column2 = value2…., columnN = valueN  WHERE [condition];  UPDATE CUSTOMERS  SET ADDRESS = ‘Pune’  WHERE ID = 6;  UPDATE CUSTOMERS  SET ADDRESS = ‘Pune’, SALARY = 1000.00; |
| DELETE | DELETE FROM table\_name  WHERE [condition];  DELETE FROM CUSTOMERS  WHERE ID = 6;  Hati hati:delete semua record  DELETE FROM CUSTOMERS; |
|  |  |
| **LIKE** | * The percent sign (%) * The underscore (\_)   SELECT \* FROM CUSTOMERS  WHERE SALARY LIKE ‘200%’;   |  |  | | --- | --- | | **No.** | **Statement & Description** | | 1 | **WHERE SALARY LIKE ‘200%’**  Finds any values that start with 200. | | 2 | **WHERE SALARY LIKE ‘%200%’**  Finds any values that have 200 in any position. | | 3 | **WHERE SALARY LIKE ‘\_00%’**  Finds any values that have 00 in the second and third positions. | | 4 | **WHERE SALARY LIKE ‘2\_%\_%’**  Finds any values that start with 2 and are at least 3 characters in length. | | 5 | **WHERE SALARY LIKE ‘%2’**  Finds any values that end with 2. | | 6 | **WHERE SALARY LIKE ‘\_2%3’**  Finds any values that have a 2 in the second position and end with a 3. | | 7 | **WHERE SALARY LIKE ‘2\_\_\_3’**  Finds any values in a five-digit number that start with 2 and end with 3. | |
| **TOP** | TOP digunakan untuk menentukan jumlah record yang akan ditampilkan  SELECT TOP number|percent column\_name(s)  FROM table\_name  WHERE [condition]  SELECT TOP 3 \* FROM CUSTOMERS; |
| **LIMIT** | Membatasi jumlah data yang ditampilkan  SELECT \* FROM CUSTOMERS  LIMIT 3;  SELECT \* FROM CUSTOMERS  WHERE ROWNUM <= 3;  [SELECT](http://localhost/phpmyadmin/url.php?url=http://dev.mysql.com/doc/refman/5.5/en/select.html) \* FROM `siswa` LIMIT 2 OFFSET 2 |
| **Order by** | sort the data  SELECT column-list  FROM table\_name  [WHERE condition]  [ORDER BY column1, column2, .. columnN] [ASC | DESC];  SELECT \* FROM CUSTOMERS  ORDER BY NAME, SALARY;  SELECT \* FROM CUSTOMERS  ORDER BY NAME DESC; |
| **GROUP BY** | SELECT column1, column2  FROM table\_name  WHERE [ conditions ]  GROUP BY column1, column2  ORDER BY column1, column2  SELECT NAME, SUM(SALARY) FROM CUSTOMERS  GROUP BY NAME;  SELECT NAME, SUM(SALARY) FROM CUSTOMERS  GROUP BY NAME; |
| **DISTINCT** | Mencegah duplicate records  ELECT DISTINCT column1, column2,…..columnN  FROM table\_name  WHERE [condition]  SELECT DISTINCT SALARY FROM CUSTOMERS  ORDER BY SALARY; |
| **Order by with CASE** | SELECT \* FROM CUSTOMERS  ORDER BY (CASE ADDRESS  WHEN ‘DELHI’ THEN 1  WHEN ‘BHOPAL’ THEN 2  WHEN ‘KOTA’ THEN 3  WHEN ‘AHMEDABAD’ THEN 4  WHEN ‘MP’ THEN 5  ELSE 100 END) ASC, ADDRESS DESC;  SELECT (CASE ADDRESS  WHEN 'padang' THEN 1  WHEN 'jakarta' THEN 2  WHEN 'KOTA' THEN 3  WHEN 'AHMEDABAD' THEN 4  WHEN 'MP' THEN 5  ELSE 100 END) address FROM CUSTOMERS |

Latihan sql “**latihan1\_nama\_sql**”

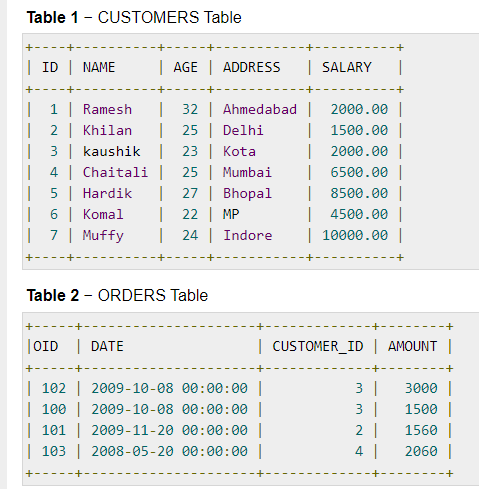
* 1. Buatlaha tabel dengan nama “kelasbe” dengan atribut id, nama,tema,alamat
  2. Input kan minimal 5 data : input satu, atau sekalian
  3. Tampilkan data berdasarkan id besar dari 3
  4. Ubah nama dengan “nama telah diubah” berdasarkan id =3
  5. Tampilkan nama yang dimulai dari huruf ‘a’
  6. Tampilkan data berdasarkan alamat , urutkan menggukan descending
  7. Tampilkan data dengan mengelompokan berdasarkan nama .
  8. Tampilkan data berdasarkan id = 3 dan id = 4 : in()

**Tugas dikumpulkan di ..**

**Tuliskan query dan Capture output pada .docx**

# Advance SQL

Table

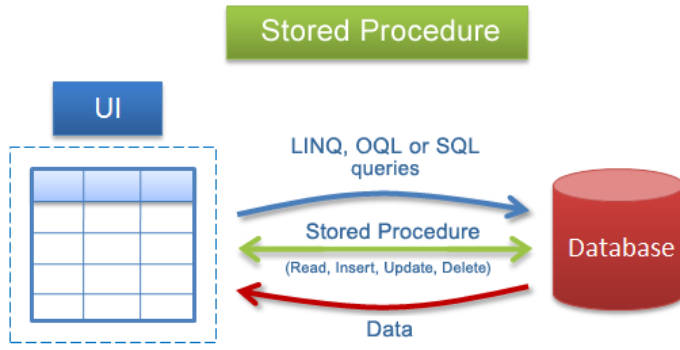


|  |  |
| --- | --- |
| Constraints | Constraint di SQL adalah sebuah opsi atau atribut yang berfungsi untuk membatasi nilai setiap data yang akan dimasukkan dalam suatu kolom di dalam tabel database SQL. Ini memastikan keakuratan dan keandalan data dalam database.   * Not Null : berfungsi untuk membatasi setiap data yang dimasukkan di suatu kolom pada tabel database SQL harus memiliki nilai. * Unique: berfungsi untuk membatasi setiap data yang dimasukkan di suatu kolom pada tabel database SQL, data yang dimasukkan harus berbeda dengan data yang ada sebelumnya sehingga tidak ada data yang sama dalam satu kolom. * Primary Key: berfungsi sebagai acuan dalam suatu tabel data * Foreign Key: berfungsi sebagai rujukan ketika memasukkan data dalam suatu kolom. Jadi jika ada data yang akan dimasukkan di kolom yang memiliki constraint Foreign Key, maka data tersebut sebelumnya sudah harus ada di tabel lain yang sudah dijadikan rujukan. * Check : berfungsi untuk melakukan pengecekan data sebelum disimpan di dalam kolom tabel SQL. Constraint check ini mirip quality controll, sehingga apabila data yang akan disimpan di kolom tidak sesuai persyaratan yang dibuat, maka data tersebut tidak dapat disimpan di kolom tersebut, sehingg SQL akan menampilkan pesan error bahwa data tersebut tidak lolos uji cek. * Default: menentukan nilai default untuk kolom. Default kolom adalah sebuah nilai yang akan dimasukkan dalam kolom oleh mesin database ketika pernyataan INSERT tidak menetapkan nilai tertentu pada sebuah kolom. |
| Drop constraint | ALTER TABLE EMPLOYEES DROP CONSTRAINT EMPLOYEES\_PK;  ALTER TABLE EMPLOYEES DROP PRIMARY KEY; |
| JOIN | used to combine records from two or more tables in a database  SELECT ID, NAME, AGE, AMOUNT  FROM CUSTOMERS, ORDERS  WHERE CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;  Belajar INNER JOIN, LEFT JOIN dan RIGHT JOIN pada SQL | YukCoding |
| **INNER JOIN** | SELECT table1.column1, table2.column2…  FROM table1  INNER JOIN table2  ON table1.common\_field = table2.common\_field;  SELECT ID, NAME, AMOUNT, DATE  FROM CUSTOMERS  INNER JOIN ORDERS  ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID; |
| **LEFT JOIN** | SELECT table1.column1, table2.column2…  FROM table1  LEFT JOIN table2  ON table1.common\_field = table2.common\_field;  SELECT ID, NAME, AMOUNT, DATE  FROM CUSTOMERS  LEFT JOIN ORDERS  ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID; |
| **RIGHT JOIN** | SELECT table1.column1, table2.column2…  FROM table1  RIGHT JOIN table2  ON table1.common\_field = table2.common\_field;  SELECT ID, NAME, AMOUNT, DATE  FROM CUSTOMERS  RIGHT JOIN ORDERS  ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID; |
| FULL JOIN | SELECT table1.column1, table2.column2…  FROM table1  FULL JOIN table2  ON table1.common\_field = table2.common\_field;  SELECT ID, NAME, AMOUNT, DATE  FROM CUSTOMERS  FULL JOIN ORDERS  ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;  If your Database does not support FULL JOIN  you can use **UNION ALL**  SELECT ID, NAME, AMOUNT, DATE  FROM CUSTOMERS  LEFT JOIN ORDERS  ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID  UNION ALL  SELECT ID, NAME, AMOUNT, DATE  FROM CUSTOMERS  RIGHT JOIN ORDERS  ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID |
| **SELF JOIN** | SELECT a.column\_name, b.column\_name…  FROM table1 a, table1 b  WHERE a.common\_field = b.common\_field;  SELECT a.ID, b.NAME, a.SALARY  FROM CUSTOMERS a, CUSTOMERS b  WHERE a.SALARY < b.SALARY; |
| CARTESIAN JOIN or CROSS JOIN | Join jenis ini menghasilkan kombinasi semua baris yang terdapat dalam tabel – tabel yang digabungkan baik yang berpasangan maupun yang tidak berpasangan  SELECT ID, NAME, AMOUNT, DATE  FROM CUSTOMERS, ORDERS; |
| **NULL** | SELECT ID, NAME, AGE, ADDRESS, SALARY  FROM CUSTOMERS  WHERE SALARY IS NOT NULL; |
| **ALIAS SYNTAX** | SELECT column1, column2….  FROM table\_name AS alias\_name  WHERE [condition];  SELECT C.ID, C.NAME, C.AGE, O.AMOUNT  FROM CUSTOMERS AS C, ORDERS AS O  WHERE C.ID = O.CUSTOMER\_ID; |
| **ALTER TABLE** | **New Column**  ALTER TABLE table\_name ADD column\_name datatype;  **DROP COLUMN**  ALTER TABLE table\_name DROP COLUMN column\_name;  **DATA TYPE**  ALTER TABLE table\_name MODIFY COLUMN column\_name datatype;  **NOT NULL**  ALTER TABLE table\_name MODIFY column\_name datatype NOT NULL;  **ADD UNIQUE CONSTRAINT**  ALTER TABLE table\_name  ADD CONSTRAINT MyUniqueConstraint UNIQUE(column1, column2…);  **ADD CHECK CONSTRAINT**  ALTER TABLE table\_name  ADD CONSTRAINT MyUniqueConstraint CHECK (CONDITION);  **ADD PRIMARY KEY**  ALTER TABLE table\_name  ADD CONSTRAINT MyPrimaryKey PRIMARY KEY (column1, column2…);  **DROP CONSTRAINT**  ALTER TABLE table\_name  DROP CONSTRAINT MyUniqueConstraint;  **DROP PRIMARY KEY**  ALTER TABLE table\_name  DROP CONSTRAINT MyPrimaryKey;  Ex  ALTER TABLE CUSTOMERS ADD SEX char(1);  ALTER TABLE CUSTOMERS DROP SEX; |
| **TRUNCATE TABLE** | untuk mengosongkan atau menghapus semuda data yang ada di **table**. Perintah **Truncate Table** mirip dengan perintah DELETE **table**, hanya lebih singkat dan sederhana tanpa menggunakan WHERE clause  TRUNCATE TABLE table\_name;  TRUNCATE TABLE CUSTOMERS; |
| SQL View | SQL View adalah tabel virtual (bukan tabel sebenarnya) yang dibuat dari beberapa tabel atau view lain. SQL View tidak memiliki data sendiri, tetapi data-datanya berasal dari tabel-tabel atau view lain yang sudah dibentuk dari kueri SQL yang telah ditentukan sebelumnya.  **Kegunaan VIEW**   * Membuat struktur data sesuai dengan kebutuhan pengguna * Menyembunyikan Kolom atau Baris sehingga pengguna hanya dapat melihat data tertentu * Menyembunyikan Sintak SQL yang rumit   **Keuntungan VIEW**   * Kita dapat membatasi akses data dengan membuat view sesuai kebutuhan * Memudahkan query yang kompleks karena tidak perlu membuat ulang query yang kompleks namun cukup memanggil view * Untuk query yang rumit serta melibatkan ratusan tabel, VIEW bisa memberikan pengaruh besar dalam performa Database SQL |
| **HAVING Clause** | Klausa HAVING digunakan untuk menentukan kondisi bagi GROUP BY. Kelompok yang memenuhi HAVING saja yang akan dihasilkan.  **Syntax**  SELECT  FROM  WHERE  GROUP BY  HAVING  ORDER BY  SELECT ID, NAME, AGE, ADDRESS, SALARY  FROM CUSTOMERS  GROUP BY age  HAVING COUNT(age) >= 2; |
| **Sub queris** | SELECT column\_name [, column\_name ]  FROM table1 [, table2 ]  WHERE column\_name OPERATOR  (SELECT column\_name [, column\_name ]  FROM table1 [, table2 ]  [WHERE])  SELECT \*  FROM CUSTOMERS  WHERE ID IN (SELECT ID  FROM CUSTOMERS  WHERE SALARY > 4500) ;  INSERT INTO CUSTOMERS\_BKP  SELECT \* FROM CUSTOMERS  WHERE ID IN (SELECT ID  FROM CUSTOMERS) ;  UPDATE CUSTOMERS  SET SALARY = SALARY \* 0.25  WHERE AGE IN (SELECT AGE FROM CUSTOMERS\_BKP  WHERE AGE >= 27 );  DELETE FROM CUSTOMERS  WHERE AGE IN (SELECT AGE FROM CUSTOMERS\_BKP  WHERE AGE >= 27 ); |
| **TUNNING DB** | tuning basis data adalah peningkatan kinerja pada desain basis data secara fisikal yang mencakup relasi dan view sesuai dengan kebutuhan pengguna.  Tujuan dari tuning basis data adalah:   * Mengurangi waktu respon dari sistem ke pengguna akhir. * Mengurangi sumber daya yang diperlukan untuk melakukan pengolahan data dalam basis data. |

# Store Prosedure

## Apa itu Stored Procedure?

Stored Procedure merupakan berbagai pernyataan atau prosedur SQL yang tersimpan di dalam database. Stored Procedure bisa menerima parameter sehingga Anda dapat menulis pernyataan IF, melakuan pengaturan pada variabel, dan lain sebagainya.



## Kelebihan Stored Procedure

Lalu apa saja kelebihan yang akan Anda dapatkan ketika menggunakan Stored Procedure? Berikut ini adalah beberapa kelebihannya:

* **Performa lebih baik**: Stored Procedure akan dikompilasi yang selanjutnya dicache dan akan digunakan kembali. Dengan begitu dapat membaut waktu respon yang lebih cepat pada saat Stored Procedure sama ingin dieksekusi kembali
* **Modular**: jika ada beberapa aplikasi yang ingin memakai query sama, maka Anda bisa menduplikasikan kode yang tidak dibutuhkan pada aplikasi. Cara terbaik yang bisa Anda lakukan adalah dengan menempatkan kode di dalam database
* **Keamanan menjadi lebih baik**: sebuah aplikasi yang menggunakan pernyataan SQL serta dibuat dengan cara dinamis diketahui cenderung rentan ketika adanya [serangan injeksi SQL](https://www.nesabamedia.com/pengertian-sql-injection/). Sementara itu, Stored Procedure memiliki kemampuan yang mampu sepenuhnya menghindari injeksi SQL dengan baik

## Manfaat Stored Procedure

Manfaat yang Anda peroleh dengan menggunakan Stored Procedure.

* Stored Procedure bisa Anda gunakan kapan saja seperti pembuatan prosedur menggunakan bahasa **pemrograman seperti Java/VB** maupun bahasa pemrograman yang lain. Ketika ada program yang memiliki banyak proses untuk dikerjakan, lebih baik program tersebut harus dipecah menjadi bagian yang lebih kecil (procedure). Fungsi dari program utama tersebut nantinya akan memanggil bagian-bagian lain dari program tersebut
* Stored Procedure membuat kinerja menjadi lebih **cepat serta efisien** karena berbasis *server side*. Jika Anda ingin membuat sebuah program yang lebih besar, pembuatan program menggunakan *server side* terasa lebih mudah jika dibandingkan *client side*. *Server side* bersifat netral pada seluruh aplikasi. Selain itu, database administrator tidak harus tahu secara mendalam mengenai [bahasa pemrograman](https://www.nesabamedia.com/pengertian-bahasa-pemrograman/) seperti Java, PHP, dan lain-lain

## Kekurangan

* berpotensi meningkatkan beban server dan
* penulisannya tidak mudah (memerlukan pengetahuan spesifik).

## Syntax

**Sintaks** stored procedure diperlihatkan sebagai berikut:

|  |
| --- |
| **CREATE** **OR** **REPLACE** **PROCEDURE** nama\_prosedure(**IN**OUT parameterName typeData) |

Untuk **memanggil** stored procedure, digunakan perintah CALL :

|  |
| --- |
| Call store\_prosedure\_name |

## Parameter stored Procedure

* **IN**

Parameter yang merupakan mode default ini mengindikasikan bahwa sebuah parameter dapat di-pass ke stored procedure tetapi nilainya tidak dapat diubah (dari dalam stored procedure).

* **OUT**

Mode ini mengindikasikan bahwa stored procedure dapat mengubah parameter dan mengirimkan kembali ke program pemanggil.

* **INOUT**

Mode ini pada dasarnya merupakan kombinasi dari mode IN dan OUT .

Contoh

|  |
| --- |
|  |

## Pencabangan

IF-ELSEIF-ELSE digunakan untuk memberikan beberapa kondisi dalam pengambilan keputusan.

|  |
| --- |
| **CREATE** **OR** **REPLACE** **PROCEDURE** public.bilangan(**IN** **input** **INT**, **OUT** hasil **VARCHAR**(20))  **LANGUAGE** plpgsql  **AS** **$procedure$**  **begin**  **IF** **input** > 15 **THEN**  **SET** hasil = 'Angka lebih dari 15';  **ELSIF** **input** > 10 **THEN**  **SET** hasil = 'Angka lebih dari 10';  **ELSE**  **SET** hasil = 'Angka kurang dari 10';  **END** **IF**;  **END**;  **$procedure$**  ;  Call :  **call** bilangan(4,'ok'); |

## Variabel

|  |
| --- |
| **CREATE** **OR** **REPLACE** **PROCEDURE** public.bilangan2(**IN** **input** **integer**, **OUT** hasil **character** **varying**, **OUT** out\_int **integer**, **OUT** out\_text **text**)  **LANGUAGE** plpgsql  **AS** **$procedure$**    **declare** var1\_int **INTEGER** := 10;  **declare** var2\_text **TEXT** := 'this is text type variable';  **declare** var3\_date **DATE** := **now**();  **begin**  **IF** **input** > 15 **THEN**  hasil = 'Angka lebih dari 15';  out\_int = var1\_int;  out\_text = var2\_text;  **ELSIF** **input** > 10 **THEN**  hasil = 'Angka lebih dari 10';  out\_int = var1\_int;  out\_text = var2\_text;  **ELSE**  hasil = 'Angka kurang dari 10';  out\_int = var1\_int;  out\_text = var2\_text;  **END** **IF**;  **END**;  **$procedure$**  ;  Call:  **call** bilangan2(4,'ok',**null**,''); |

## Case

Case adalah bentuk percabangan case sederhana yang dapat memeriksa serangkaian kondisi yang memiliki nilai unik.

|  |
| --- |
| **CREATE** **OR** **REPLACE** **PROCEDURE** public.bilanganCase(**IN** **input** **integer**, **OUT** hasil **character** **varying**, **OUT** out\_int **integer**, **OUT** out\_text **text**)  **LANGUAGE** plpgsql  **AS** **$procedure$**    **declare** var1\_int **INTEGER** := 10;  **declare** var2\_text **TEXT** := 'this is text type variable';  **declare** var3\_date **DATE** := **now**();  **begin**  **CASE**  **WHEN** **input** = 10 **THEN**  hasil = 'Angka 10';  **WHEN** **input** = 20 **THEN**  hasil = 'Angka 20';  **WHEN** **input** = 30 **THEN**  hasil = 'Angka 30';  **ELSE**  hasil = 'Bukan angka 10, 20, atau 30';  **END** **CASE**;  **END**;  **$procedure$**  ;  Call :  **call** bilangancase(10,'ok',**null**,'');  Output |

## Print to Console

Digunakan untuk mencetak ke log console

|  |
| --- |
| **raise** **notice** 'Value: %', nama\_variabel; |

## LOOP

Syntax

|  |
| --- |
| **LOOP**  <execution block starts>    <**EXIT** **condition** based **on** requirement>    <execution\_block\_ends>  **END** **LOOP**; |

|  |
| --- |
| **CREATE** **OR** **REPLACE** **PROCEDURE** public.**loop**()  **LANGUAGE** plpgsql  **AS** **$procedure$**  **declare** a **int** := 1 ;  **begin**  **LOOP**  a:=+a+1;  **raise** **notice** 'Value: %', a;  **EXIT** **WHEN** a>5;    **END** **LOOP**;    **END**;  **$procedure$**  ;  Call:  **call** **loop**(); |

## For Loop

Syntax

|  |
| --- |
| [ <<**label**>> ]  **for** loop\_cnt **in** [ **reverse** ] **from**.. **to** [ **by** step ] **loop**  statements  **end** **loop** [ **label** ]; |

Contoh :

|  |
| --- |
| **for** cnt **in** 1..10 **loop**  **raise** **notice** 'cnt: %', cnt;  **end** **loop**;  **Output** |

## NOT EXIST & EXIST

Pengecekan data ada atau tidak.

**Not exists**

|  |
| --- |
| **If** **not** **Exists**(**select** id **from** barang b **where** id = resid) **Then**  **raise** **notice** 'id tidak ada';  **return**;  **else**  **raise** **notice** 'id ada';  **end** **if**; |

**Contoh:**

|  |
| --- |
| Pada praktek |

## Function

Tujuan adalah : returns column table

#### Return table: return Query

Contoh : table barang

Contoh select with parameter like:

|  |
| --- |
| **CREATE** **OR** **REPLACE** **FUNCTION** public.getbarang(rqnama **character** **varying**)  **RETURNS** **TABLE**(resid **bigint**, resnama **character** **varying**, resstok **integer**, resharga **double** **precision**, ressatuan **character** **varying**)  **LANGUAGE** plpgsql  **AS** **$function$**  **begin**  **return** query  **SELECT**  id,  nama,  stok,  harga,  satuan  **FROM** barang  **WHERE** nama **ILIKE** rqNama;  **end**;**$function$**  ;  **Call :**  **select** \* **from** getbarang('%barang%');  **Output** |

#### Return table with For LOOP

Contoh select with parameter like:

|  |
| --- |
| **CREATE** **OR** **REPLACE** **FUNCTION** public.getbarang1(rqnama **character** **varying**)  **RETURNS** **TABLE**(resid **integer**, resnama **character** **varying**, resstok **character** **varying**, resharga **integer**, ressatuan **character** **varying**)  **LANGUAGE** plpgsql  **AS** **$function$**  **DECLARE**  var\_r record;  **BEGIN**  **FOR** var\_r **IN**(**SELECT**  id,  nama,  stok,  harga,  satuan  **FROM** barang  **WHERE** nama **ILIKE** rqNama)  **LOOP**  resid := var\_r.id ;  resnama := var\_r.nama ;  resstok := var\_r.stok ;  resharga := var\_r.harga ;  ressatuan := var\_r.satuan ;  **RETURN** **NEXT**;  **END** **LOOP**;  **END**; **$function$**  ;  **Call** :  **select** \* **from** getbarang1('%barang%');  **Output:** |

#### Return By ID

|  |
| --- |
| **CREATE** **OR** **REPLACE** **FUNCTION** public.getbyidfunction(rqid **integer**)  **RETURNS** **TABLE**(resid **bigint**, resnama **character** **varying**, ressatuan **character** **varying**, resstok **integer**, resharga **double** **precision**)  **LANGUAGE** plpgsql  **AS** **$function$**  **BEGIN**  **RETURN** QUERY  **select** e.id, e.nama, e.satuan,e.stok,e.harga  **FROM** public.barang **AS** e  **WHERE** id = rqId;  **END**;  **$function$**  ;  **Call** :  **select** \* **from** getbyidfunction(22);  **Output** |

#### Select + Case + COALESCE + For Loop

|  |
| --- |
| **CREATE** **OR** **REPLACE** **FUNCTION** public.getbarang2(rqnama **character** **varying**)  **RETURNS** **TABLE**(resid **integer**, resnama **character** **varying**, resstok **character** **varying**, resharga **integer**, ressatuan **character** **varying**)  **LANGUAGE** plpgsql  **AS** **$function$**  **DECLARE**  var\_r record;  **BEGIN**  **FOR** var\_r **IN**(**SELECT**  id,  **CASE** nama  **WHEN** 'barang 10' **THEN** 'result barang 10'  **WHEN** 'barang 11' **THEN** 'result barang 11'  **ELSE** nama **END** ,  stok,  **COALESCE**(harga,0) **as** harga,  satuan  **FROM** barang  **WHERE** nama **ILIKE** rqNama)  **LOOP**  resid := var\_r.id ;  resnama := var\_r.nama ;  resstok := var\_r.stok ;  resharga := var\_r.harga ;  ressatuan := var\_r.satuan ;  **RETURN** **NEXT**;  **END** **LOOP**;  **END**; **$function$**  ;  **Call :**  **select** \* **from** getbarang2('%barang%');  Output: |

## Praktek Store Prosedure and Function

Studi kasus table barang; CRUD

#### Insert

Insert contoh 1

|  |
| --- |
| **CREATE** **OR** **REPLACE** **PROCEDURE** public.savebarang(rqharga **float**, rqnama **character** **varying**, rqsatuan **character** **varying**, rqstok **integer**, **INOUT** resid **integer**)  **LANGUAGE** plpgsql  **AS** **$procedure$**  **begin**    **INSERT** **into** public.barang  (id,harga,nama,satuan,stok,created\_date,updated\_date)  **SELECT** **nextval**('barang\_id\_seq'),  rqharga,  rqnama,  rqsatuan,  rqstok,  **NOW**(),**NOW**()  **RETURNING** id **INTO** resid;      **commit**;  **END**;  **$procedure$**  ;  **How to call :**  **call** savebarang(1000, 'baju', 'pcs', 200, **null**)  output: |

Insert with out :

|  |
| --- |
| **CREATE** **OR** **REPLACE** **PROCEDURE** public.savebarangout(**IN** rqharga **double** **precision**, **IN** rqnama **character** **varying**, **IN** rqsatuan **character** **varying**, **IN** rqstok **integer**, **INOUT** resid **integer**, **OUT** resharga **double** **precision**, **OUT** resnama **character** **varying**, **OUT** ressatuan **character** **varying**, **OUT** resstok **integer**)  **LANGUAGE** plpgsql  **AS** **$procedure$**  **begin**    **INSERT** **into** public.barang  (id,harga,nama,satuan,stok,created\_date,updated\_date)  **SELECT** **nextval**('barang\_id\_seq'),  rqharga,  rqnama,  rqsatuan,  rqstok,  **NOW**(),**NOW**()  **RETURNING** id **INTO** resid;  resharga = rqharga;  resnama =rqnama;  ressatuan =rqsatuan;  resstok =rqstok;    **commit**;  **END**;  **$procedure$**  ;  **Call:**  **call** savebarangout(1000.989, 'baju 3', 'pcs', 200, **null**, **null**, **null**, **null**, **null**)  **output** |

Insert with out dan eror code

|  |
| --- |
| **CREATE** **OR** **REPLACE** **PROCEDURE** public.savebarangoutwitheror(**IN** rqharga **double** **precision**, **IN** rqnama **character** **varying**, **IN** rqsatuan **character** **varying**, **IN** rqstok **integer**, **INOUT** resid **integer**, **INOUT** resharga **double** **precision**, **INOUT** resnama **character** **varying**, **INOUT** ressatuan **character** **varying**, **INOUT** resstok **integer**, **INOUT** reserordesc **character** **varying**, **INOUT** reserorcode **integer**)  **LANGUAGE** plpgsql  **AS** **$procedure$**  **begin**  **if** rqnama **is** **null** **Then**  **raise** **notice** 'nama kosong';  reserordesc = 'nama wajib diisi';  reserorcode = 404;  **return**;  **else**  **raise** **notice** 'nama ada';  **end** **if**;    **INSERT** **into** public.barang  (id,harga,nama,satuan,stok,created\_date,updated\_date)  **SELECT** **nextval**('barang\_id\_seq'),  rqharga,  rqnama,  rqsatuan,  rqstok ,  **NOW**(),**NOW**()  **RETURNING** id **INTO** resid;  resharga = rqharga;  resnama =rqnama;  ressatuan =rqsatuan;  resstok =rqstok;    reserordesc = 'sukses';  reserorcode = 200;  **commit**;  **END**;  **$procedure$**  ;  Cara pangiil :  **call** savebarangoutwitheror(123, **null**, 'unit', 100, **null**,**null**,**null**,**null**,**null**,**null**,**null**);    **select** \* **from** barang b **where** id = 9; |

Req dan res digabung jadi satu : ada response status

|  |
| --- |
| **call** savebarangoutwitheror\_reqressama(123, 'rumah', 'unit', 100, **null**,**null**,**null**);  **query prosedure:**  **CREATE** **OR** **REPLACE** **PROCEDURE** public.savebarangoutwitheror\_reqressama(**INOUT** resharga **double** **precision**, **INOUT** resnama **character** **varying**, **INOUT** ressatuan **character** **varying**, **INOUT** resstok **integer**, **INOUT** resid **integer**, **INOUT** reserordesc **character** **varying**, **INOUT** reserorcode **integer**)  **LANGUAGE** plpgsql  **AS** **$procedure$**  **begin**  **if** resnama **is** **null** **Then**  **raise** **notice** 'nama kosong';  reserordesc = 'nama wajib diisi';  reserorcode = 404;  **return**;  **else**  **raise** **notice** 'nama ada';  **end** **if**;    **INSERT** **into** public.barang  (id,harga,nama,satuan,stok,created\_date,updated\_date)  **SELECT** **nextval**('barang\_id\_seq'),  resharga,  resnama,  ressatuan,  resstok ,  **NOW**(),**NOW**()  **RETURNING** id **INTO** resid;    reserordesc = 'sukses';  reserorcode = 200;  **commit**;  **END**;  **$procedure$**  ; |

#### Update

|  |
| --- |
| **CREATE** **OR** **REPLACE** **PROCEDURE** public.updatebarang(rqharga **integer**, rqnama **character** **varying**, rqsatuan **character** **varying**, rqstok **integer**, **INOUT** resid **integer**)  **LANGUAGE** plpgsql  **AS** **$procedure$**  **begin**    **update** public.barang  **set** harga = rqharga, nama = rqnama, satuan = rqsatuan, stok = rqstok  **where** id = resid **returning** id **into** resid;  **commit**;  **END**;  **$procedure$**  ;  **How to call :**  **call** updatebarang(3000, 'buku', 'lusin', 300, 22)  output: |

Update barang out

|  |
| --- |
| **CREATE** **OR** **REPLACE** **PROCEDURE** public.updatebarangout(**IN** rqharga **integer**, **IN** rqnama **character** **varying**, **IN** rqsatuan **character** **varying**, **IN** rqstok **integer**, **INOUT** resid **integer**, **INOUT** resharga **double** **precision**, **INOUT** resnama **character** **varying**, **INOUT** ressatuan **character** **varying**, **INOUT** resstok **integer**)  **LANGUAGE** plpgsql  **AS** **$procedure$**  **begin**    **update** public.barang  **set** harga = rqharga, nama = rqnama, satuan = rqsatuan, stok = rqstok  **where** id = resid **returning** id **into** resid;  resharga = rqharga;  resnama =rqnama;  ressatuan =rqsatuan;  resstok =rqstok;  **commit**;  **END**;  **$procedure$**  ;  Call :  **call** updatebarangout(3000, 'buku update 1', 'lusin', 300, 22, **null**, '', '', **null**)  Output : |

**Insert with out and eror description**

|  |
| --- |
| **CREATE** **OR** **REPLACE** **PROCEDURE** public.updatebarangoutwitheror(**IN** rqharga **integer**, **IN** rqnama **character** **varying**, **IN** rqsatuan **character** **varying**, **IN** rqstok **integer**, **INOUT** resid **integer**, **INOUT** resharga **double** **precision**, **INOUT** resnama **character** **varying**, **INOUT** ressatuan **character** **varying**, **INOUT** resstok **integer**, **INOUT** reserordesc **character** **varying**, **INOUT** reserorcode **integer**)  **LANGUAGE** plpgsql  **AS** **$procedure$**  **begin**  **If** **not** **Exists**(**select** id **from** barang b **where** id = resid) **Then**  **raise** **notice** 'id tidak ada';  reserordesc = 'id tidak ditemukan';  reserorcode = '404';  **return**;  **else**  **raise** **notice** 'id ada';  **end** **if**;    **update** public.barang  **set** harga = rqharga, nama = rqnama, satuan = rqsatuan, stok = rqstok  **where** id = resid **returning** id **into** resid;  resharga = rqharga;  resnama =rqnama;  ressatuan =rqsatuan;  resstok =rqstok;    reserordesc = 'sukses';  reserorcode = '200';  **commit**;  **END**;  **$procedure$**  ;  **CALL**:  **call** updatebarangoutwitheror(3000, 'buku update 1', 'lusin', 300, 2222, **null**, '', '', **null**,'', **null**);  **Output:** |

#### Get List

|  |
| --- |
| **CREATE** **OR** **REPLACE** **FUNCTION** public.getbarang1(rqnama **character** **varying**)  **RETURNS** **TABLE**(resid **integer**, resnama **character** **varying**, resstok **character** **varying**, resharga **integer**, ressatuan **character** **varying**)  **LANGUAGE** plpgsql  **AS** **$function$**  **DECLARE**  var\_r record;  **BEGIN**  **FOR** var\_r **IN**(**SELECT**  id,  nama,  stok,  harga,  satuan  **FROM** barang  **WHERE** nama **ILIKE** rqNama)  **LOOP**  resid := var\_r.id ;  resnama := var\_r.nama ;  resstok := var\_r.stok ;  resharga := var\_r.harga ;  ressatuan := var\_r.satuan ;  **RETURN** **NEXT**;  **END** **LOOP**;  **END**; **$function$**  ;  How to call Function:  **select** \* **from** getbarang1('%barang%')  Output: |

#### Get By Id

|  |
| --- |
| **CREATE** **OR** **REPLACE** **FUNCTION** public.getbyidfunction(rqid **integer**)  **RETURNS** **TABLE**(resid **bigint**, resnama **character** **varying**, ressatuan **character** **varying**, resstok **integer**, resharga **double** **precision**)  **LANGUAGE** plpgsql  **AS** **$function$**  **BEGIN**  **RETURN** QUERY  **select** e.id, e.nama, e.satuan,e.stok,e.harga  **FROM** public.barang **AS** e  **WHERE** id = rqId;  **END**;  **$function$**  ;  **How to call function:**  **select** \* **from** getbyidfunction(22)  Output |

#### Delete

|  |
| --- |
| **CREATE** **OR** **REPLACE** **PROCEDURE** public.deletebarang(rqid **integer**)  **LANGUAGE** plpgsql  **AS** **$procedure$**  **begin**  **delete** **from** barang **where** id = rqid;  **commit**;  **END**;  **$procedure$**  ; |

<https://www.nesabamedia.com/stored-procedure-pada-database/>

https://www.enterprisedb.com/postgres-tutorials/how-work-control-structures-postgresql-stored-procedures-using-if-case-and-loop

# Store Procedure dan function studi kasus employee

## ddl

|  |
| --- |
| -- public.employee definition  -- Drop table  -- DROP TABLE public.employee;  **CREATE** **TABLE** public.employee (  id bigserial **NOT** **NULL**,  created\_date **timestamp**(6) **NOT** **NULL**,  deleted\_date **timestamp**(6) **NULL**,  updated\_date **timestamp**(6) **NOT** **NULL**,  address **text** **NULL**,  dob **date** **NULL**,  "name" **varchar**(255) **NULL**,  status **varchar**(255) **NULL**,  **CONSTRAINT** employee\_pkey **PRIMARY** **KEY** (id)  );  **INSERT** **INTO** employee (created\_date, updated\_date, address, dob, "name", status)  **VALUES** (**now**(), **now**(), 'Alamat Karyawan 1', '1990-01-15', 'Karyawan 1', 'Active'),  (**now**(), **now**(), 'Alamat Karyawan 2', '1980-01-15', 'Karyawan 2', 'Active'),  (**now**(), **now**(), 'Alamat Karyawan 3', '1970-01-15', 'Karyawan 3', 'Active'),  (**now**(), **now**(), 'Alamat Karyawan 4', '1960-01-15', 'Karyawan 4', 'Active'),  (**now**(), **now**(), 'Alamat Karyawan 5', '1950-01-15', 'Karyawan 5', 'Active'); |

## Function get id

|  |
| --- |
| **CREATE** **OR** **REPLACE** **FUNCTION** public.getemployee6()  **RETURNS** **TABLE**(resid **bigint**, resname **character** **varying**, resaddress **text**)  **LANGUAGE** plpgsql  **AS** **$function$**  **begin**  **RETURN** QUERY  **select** id **bigint**, **name** **varchar**, address **from** employee e;  **END**;  **$function$**  ; |

## Function list by name

|  |
| --- |
| **CREATE** **OR** **REPLACE** **FUNCTION** public.listemployee2(rqnama **character** **varying**)  **RETURNS** **TABLE**(resid **bigint**, resname **character** **varying**, resaddress **character** **varying**)  **LANGUAGE** plpgsql  **AS** **$function$**  **DECLARE**  var\_r record;  **BEGIN**  **FOR** var\_r **IN**(**SELECT**  id,  **name**,  address  **FROM** employee  **WHERE** **name** **ILIKE** rqNama)  **LOOP**  resid :=var\_r.id ;  resname :=var\_r.**name**;  resaddress :=var\_r.address;  **RETURN** **NEXT**;  **END** **LOOP**;  **end** **$function$**  ; |

## Save

|  |
| --- |
| **CREATE** **OR** **REPLACE** **PROCEDURE** public.insert1(**INOUT** resid **bigint**, **INOUT** rqnama **character** **varying**)  **LANGUAGE** plpgsql  **AS** **$procedure$**  **begin**  **if** rqnama **is** **null** **Then**  **raise** **notice** 'nama kosong';  rqnama = 'nama wajib diisi';  **return**;  **else**  **raise** **notice** 'nama ada';  **end** **if**;  **INSERT** **into** public.employee  (id, created\_date, updated\_date, address, dob, "name", status)  **SELECT** **nextval**('employee\_id\_seq'),  **now**(),  **now**(),  'jakarta',  '1970-01-15',  rqnama,  'active'  **RETURNING** id **INTO** resid;  rqnama=rqnama;  **commit**;  **END**;  **$procedure$**  ; |

## Update

|  |
| --- |
| **CREATE** **OR** **REPLACE** **PROCEDURE** public.update\_employee(**INOUT** resid **bigint**, **INOUT** rqnama **character** **varying**)  **LANGUAGE** plpgsql  **AS** **$procedure$**  **begin**  **if** rqnama **is** **null** **Then**  **raise** **notice** 'nama kosong';  rqnama = 'nama wajib diisi';  **return**;  **else**  **raise** **notice** 'nama ada';  **end** **if**;  **update** employee  **set** **name**=rqnama  **where** id = resid;  rqnama=rqnama;  **commit**;  **END**;  **$procedure$**  ; |

|  |
| --- |
|  |

## Deleted

|  |
| --- |
| **CREATE** **OR** **REPLACE** **PROCEDURE** public.deleted\_employee1(**INOUT** resid **bigint**)  **LANGUAGE** plpgsql  **AS** **$procedure$**  **begin**  **update** employee  **set** created\_date =**now**()  **where** id = resid;  **commit**;  **END**;  **$procedure$**  ; |

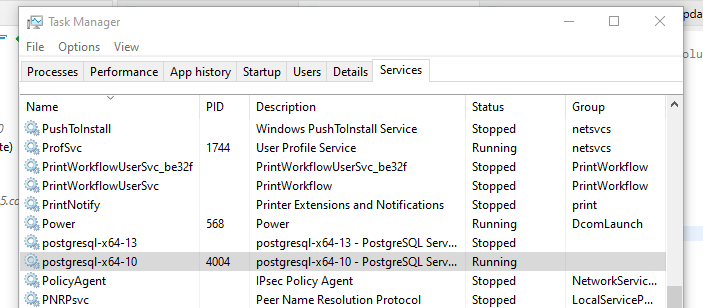
## Get by id

|  |
| --- |
| **CREATE** **OR** **REPLACE** **FUNCTION** public.getemployee6(**in** reqid **bigint** )  **RETURNS** **TABLE**(resid **bigint**, resname **varchar**, resaddress **text**)  **LANGUAGE** plpgsql  **AS** **$function$**  **begin**  **RETURN** QUERY  **select** id **bigint**, **name** **varchar**, address **from** employee e **where** id =reqid;  **END**;  **$function$**  ; |

# Issue

## Service Postgres SQL mati

Masuk ke task manager – pilih service – klik kanan service postres, pilih start.



# Latihan membuat ERD

# Entitas dan Atribut:

## Users Entity:

id (Tipe data: UUID, Primary Key)

username (Tipe data: String)

email\_address (Tipe data: String)

password (Tipe data: String)

## Merchant Entity:

id (Tipe data: UUID, Primary Key)

merchant\_name (Tipe data: String)

merchant\_location (Tipe data: String)

open (Tipe data: Boolean)

## Product Entity:

id (Tipe data: UUID, Primary Key)

product\_name (Tipe data: String)

price (Tipe data: Decimal atau Double, tergantung preferensi)

merchant\_id (Tipe data: UUID, Foreign Key mengacu ke Merchant)

## Order Entity:

id (Tipe data: UUID, Primary Key)

order\_time (Tipe data: Tanggal/Jam)

destination\_address (Tipe data: String)

user\_id (Tipe data: UUID, Foreign Key mengacu ke Users)

completed (Tipe data: Boolean)

## Order Detail Entity:

id (Tipe data: UUID, Primary Key)

order\_id (Tipe data: UUID, Foreign Key mengacu ke Order)

product\_id (Tipe data: UUID, Foreign Key mengacu ke Product)

quantity (Tipe data: Integer)

total\_price (Tipe data: Decimal atau Double, tergantung preferensi)

# Hubungan:

## Users dan Order:

* Hubungan ini adalah "satu ke banyak" (one-to-many) antara tabel Users dan Order.
* Setiap pengguna (Users) dapat memiliki banyak pesanan (Order), tetapi setiap pesanan hanya dapat terkait dengan satu pengguna.
* Ini diimplementasikan melalui kolom user\_id di tabel Order, yang merupakan Foreign Key yang mengacu pada kolom id di tabel Users.

## Merchant dan Product:

* Hubungan ini juga adalah "satu ke banyak" (one-to-many) antara tabel Merchant dan Product.
* Setiap merchant (Merchant) dapat memiliki banyak produk (Product), tetapi setiap produk hanya dapat terkait dengan satu merchant.
* Ini diimplementasikan melalui kolom merchant\_id di tabel Product, yang merupakan Foreign Key yang mengacu pada kolom id di tabel Merchant.

## Product dan Order Detail:

* Hubungan ini adalah "satu ke banyak" (one-to-many) antara tabel Product dan Order Detail.
* Setiap produk (Product) dapat terkait dengan banyak detail pesanan (Order Detail), tetapi setiap detail pesanan hanya dapat terkait dengan satu produk.
* Ini diimplementasikan melalui kolom product\_id di tabel Order Detail, yang merupakan Foreign Key yang mengacu pada kolom id di tabel Product.

## Users dan Order Detail:

* Hubungan antara tabel Users dan Order Detail tidak terjadi secara langsung. Hubungan ini diatur melalui tabel Order.
* Setiap detail pesanan (Order Detail) terkait dengan satu pesanan (Order), dan satu pesanan hanya dapat terkait dengan satu pengguna (Users).
* Ini diimplementasikan melalui kolom user\_id di tabel Order, yang merupakan Foreign Key yang mengacu pada kolom id di tabel Users. Kemudian, tabel Order Detail terkait dengan tabel Order melalui kolom order\_id.

## Order dan Order Detail:

* Hubungan ini adalah "satu ke banyak" (one-to-many) antara tabel Order dan Order Detail.
* Setiap pesanan (Order) dapat memiliki banyak detail pesanan (Order Detail), tetapi setiap detail pesanan hanya terkait dengan satu pesanan.
* Ini diimplementasikan melalui kolom id di tabel Order, yang merupakan Primary Key, dan kolom order\_id di tabel Order Detail, yang merupakan Foreign Key yang mengacu pada kolom id di tabel Order.

# ERD

A screenshot of a computer

Description automatically generated

# Referensi

<https://www.dicoding.com/blog/apa-itu-database/>

<https://dosenit.com/sql/perbedaan-sql-dan-nosql>

<https://aws.amazon.com/id/nosql/>

<https://www.sekawanmedia.co.id/pengertian-sql/>

<https://www.contohapps.com/2020/08/mengenal-database-nosql.html>

<https://idcloudhost.com/panduan-sql-fungsi-cara-kerja-serta-perintah-dasarnya/>

<https://www.tutorialspoint.com/sql/sql-data-types.htm>

<https://arfianhidayat.com/tipe-constraint-pada-sql-dan-fungsi-constraint>

<https://arfianhidayat.com/view-pada-sql>

<https://bundet.com/d/917-apa-itu-tuning-basis-data>

https://www.w3schools.com/sql/sql\_alter.asp