**Name: Andi Dwi Saputro**

**Date: April 21st, 2020**

---

1. **Apakah trigger bisa juga untuk update ? mana yang lebih baik, insert update atau trigger on update ? Apa bedanya dan bagaimana mengimplementasikannya ?**
   - Trigger bisa juga untuk proses update baik dengan timeline before maupun after.

   - Menurut saya untuk pemrosesan insert dan update yang diperlukan untuk menghindari duplikasi key pada entry data/rows maka penggunaan query **insert on update duplicate key update** adalah yang terbaik. Namun untuk kasus reporting dan pencatatan logs update pada database lebih baik adalah dengan menggunakan **trigger on update**.

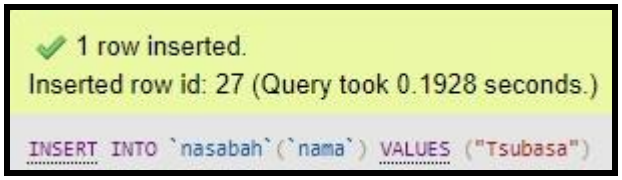
   - Perbedaan insert update dengan trigger on update

| Insert Update | Trigger on Update |
|---|---|
| Menghindari duplikasi key pada rows saat proses insert/update | Digunakan untuk reporting atau pembuatan logs |
| Perubahan value pada field (e.g: UUID) pasti terjadi pada table source / table source akan mempopulate hasil insert atau update data terbaru yang memiliki update duplicate key. | Perubahan value pada field (e.g UUID) tidak terjadi pada table source melainkan hanya pada table logs, untuk mempopulate perubahan value pada field (e.g: UUID) berdasarkan generated value dari table logs ke table source harus mengunakan query **insert update duplicate key.** |
| Dapat diimplementasikan pada table yang memiliki constraint | Tidak dapat diimplementasikan table yang memiliki constraint |

   Implementasinya:

   a. Insert Update

      Menambahkan nasabah baru



| id_nasabah | nama |
|---|---|
| 2 | Dudung |
| 4 | Ruslan |
| 5 | Andi |
| 6 | Asep |
| 7 | Ujang |
| 9 | Anton |
| 10 | Peter Shawn |
| 13 | Jon Bon Jovi |
| 14 | La Campanela |
| 15 | Jon Bon Jovi |
| 16 | Abon Jovi |
| 17 | SpringBoot |
| 25 | Monacar |
| 26 | Vivaldi Winter |
| 27 | Tsubasa |

Melakukan query insert on duplicate key update / insert update


Disini terlihat ada 2 rows yang di insert atau affected berdasarkan query diatas, karena sebuah row dengan id_nasabah 27 sudah ada pada table nasabah, kemudia statement query melakukan update nama dari Tsubasa menjadi Tsubasa Ozora, jadi ada 2 kali proses eksekusi statement insert dan update.

Result:

| id_nasabah | nama |
|---|---|
| 2 | Dudung |
| 4 | Ruslan |
| 5 | Andi |
| 6 | Asep |
| 7 | Ujang |
| 9 | Anton |
| 10 | Peter Shawn |
| 13 | Jon Bon Jovi |
| 14 | La Campanela |
| 15 | Jon Bon Jovi |
| 16 | Abon Jovi |
| 17 | SpringBoot |
| 25 | Monacar |
| 26 | Vivaldi Winter |
| 27 | Tsubasa Ozora |

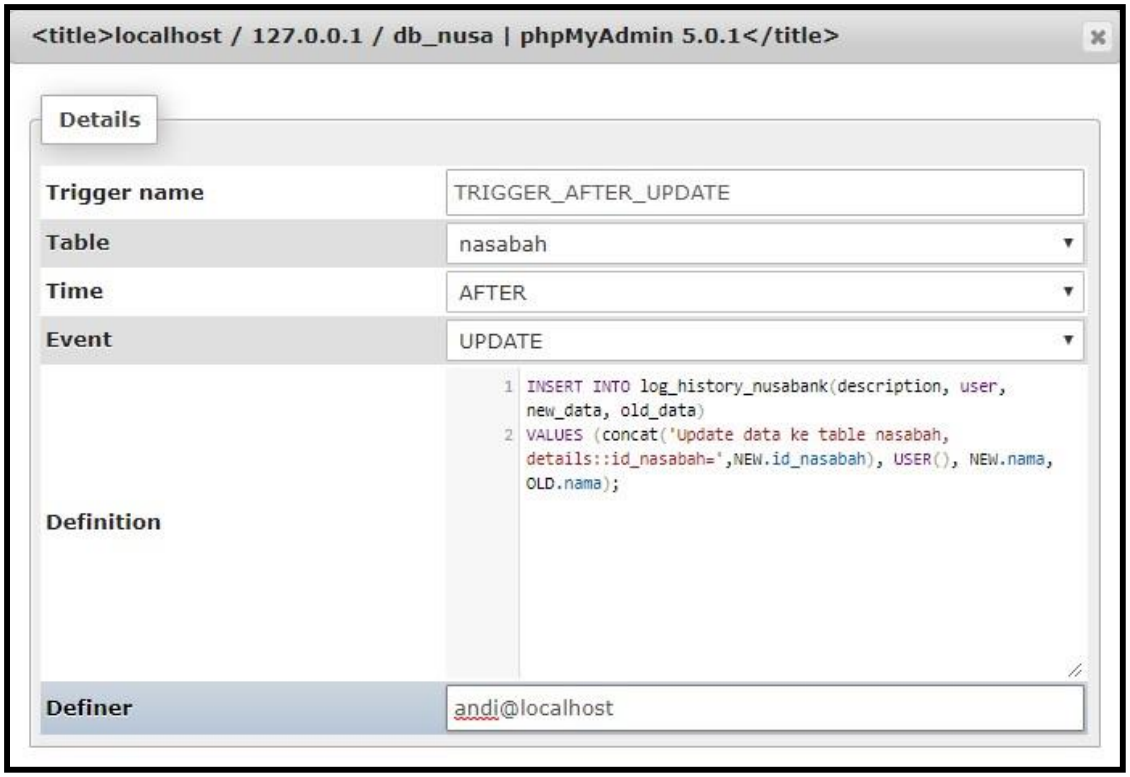
b. Trigger on Update

- **Membuat table log_history_nusabank**
  Berikut struktur tablenya:

| | # | Name | Type | Collation | Attributes | Null | Default |
|---|---|---|---|---|---|---|---|
| ☐ | 1 | **description** | text | latin1_swedish_ci | | No | *None* |
| ☐ | 2 | **log_time** | timestamp | | | No | current_timestamp() |
| ☐ | 3 | **user** | varchar(60) | latin1_swedish_ci | | No | *None* |
| ☐ | 4 | **old_data** | varchar(45) | latin1_swedish_ci | | No | *None* |
| ☐ | 5 | **new_data** | varchar(45) | latin1_swedish_ci | | No | *None* |

  - Description: untuk menampung keterangan detail mengenai perubahan yang terjadi
  - Log_time: untuk mencatat keterangan waktu kapan perubahan terjadi
  - User: merupakan user database administrator yang melakukan proses data manipulation tersebut
  - Old_data: merupakan value data yang sebelumnya
  - New_data: merupakan value data terbaru yang diubah atau ditambah.

- **Membuat trigger after update**




- **Uji coba trigger dengan query update**


- **Result pada log table**


2. **Apa-apa saja jenis trigger ? kapan baiknya digunakan dan bagaimana menggunakannya ? Hubungkan dengan kasus, apakah kita sebaiknya mendelete row atau hanya menandai bahwa row tsb statusnya terdelete ? apa buruknya dari mendelete row ? bagaiman baiknya kita mendelete row ?**

➢ **Jenis Trigger**

a. **Row-level dan Statement-level Trigger**
Ketika membuat sebuah trigger, kita dapat menentukan berapa kali action trigger dieksekusi:

- Row-level trigger dieksekusi untuk setiap row yang dimanipulasi pada suatu transaksi. Dengan kata lain, row-level trigger mengerjakan trigger action 1x/row. Penerapan trigger ini ditunjukkan oleh adanya clause FOR EACH ROW. Row-level trigger berguna jika kode di dalam trigger body bergantung pada tiap baris yang terpengaruh oleh suatu triggering statement.
- Statement-level trigger dieksekusi 1x pada saat transaction, tanpa memperhatikan jumlah row yang terlibat. Misalnya, jika tedapat suatu transaction yang memasukkan 1000 rows ke dalam table, maka statement-level trigger hanya akan dieksekusi sekali saja. Statement-level trigger berguna jika kode dalam trigger body tidak tergantung ada rows yang terpengaruh oleh triggering statement. Secara default trigger yang dibuat adalah statement-level trigger.

**b. Before and After Trigger**

Merupakan konsep untuk menentukan timing pada suatu action trigger, akankah dieksekusi sebelum atau sesudah triggering statement diekeskusi.

Before trigger menjalankan trigger body sebelum even atau triggering statement. Oleh sebab itu, trigger ini cocok utk mendeteksi bagaimana event boleh dilanjutkan atau tidak. After trigger menjalankan trigger action setelah event terjadi.

Penggunaan before dan after trigger sangat cocok untuk pembuatan history logs.

**c. Instead of Trigger**

Instead of Trigger merupakan trigger yang hanya akan dieksekusi bagi View dan diaktivasi jika terjadi perubahan pada source/base table. Proses yang dilakukan oleh triggering statement akan digantikan oleh action pada trigger body.

**d. System event & User event Trigger**

Penggunaan trigger dapat dikelompokkan menurut evt yang terjadi:

- System events
  a. Database startup & shutdown
  b. Server error msg events
  c. User events
- User events (logon/off)
  a. DDL statements
  b. DML statements

➢ **Kapan baiknya digunakan dan bagaimana menggunakannya**

Penggunaan pada trigger sebaiknya saat kita memerlukan auditing atau report berdasarkan log history perubahan yang ada pada table didatabase kita.

Untuk menggunakan trigger berikut saya paparkan berdasarkan contoh kasus nusabank untuk melakukan pencatatan atau auditing history perubahan pada table nasabah ke table log_history_nasabah.

berikut step-stepnya menggunakan phpmyadmin:

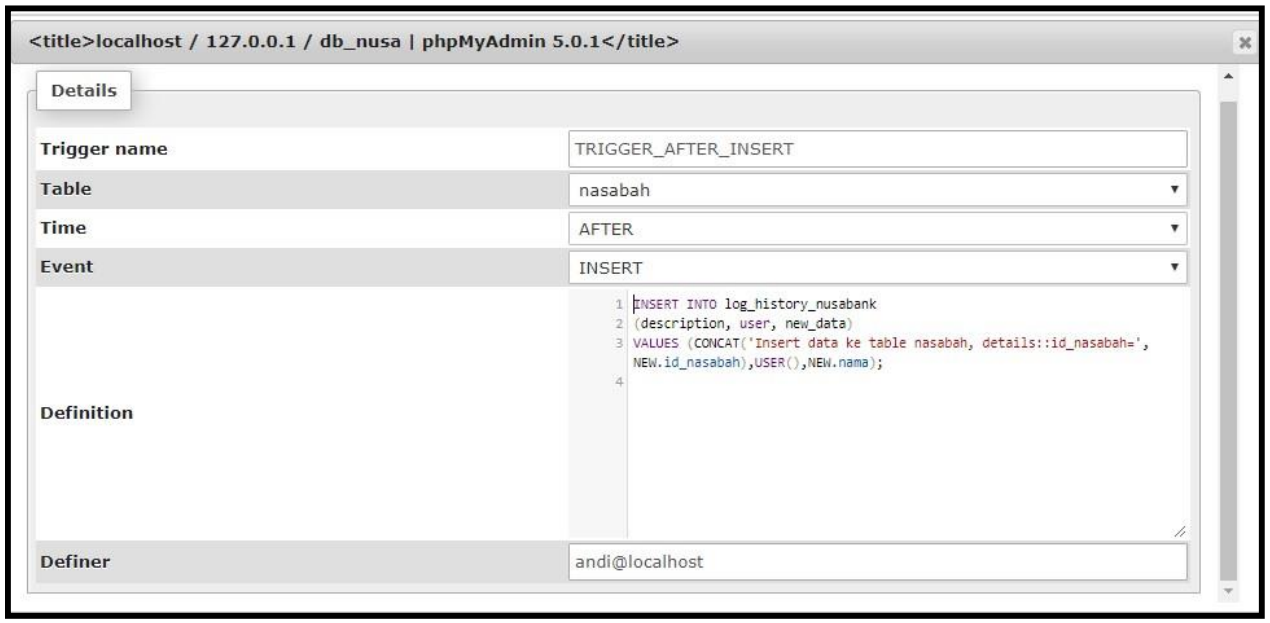
1. **Membuat table log_history_nusabank**

   Berikut struktur tablenya:

| # | Name | Type | Collation | Attributes | Null | Default |
|---|---|---|---|---|---|---|
| 1 | description | text | latin1_swedish_ci | | No | *None* |
| 2 | log_time | timestamp | | | No | current_timestamp() |
| 3 | user | varchar(60) | latin1_swedish_ci | | No | *None* |
| 4 | old_data | varchar(45) | latin1_swedish_ci | | No | *None* |
| 5 | new_data | varchar(45) | latin1_swedish_ci | | No | *None* |

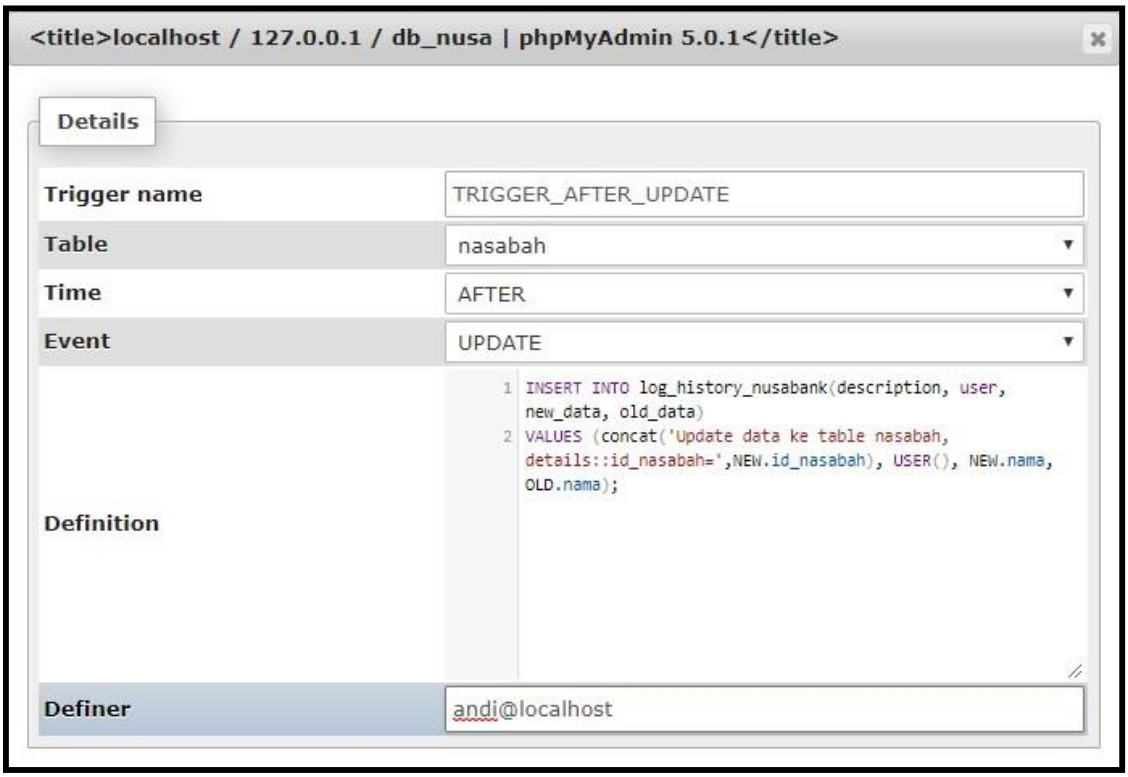
- Description: untuk menampung keterangan detail mengenai perubahan yang terjadi
- Log_time: untuk mencatat keterangan waktu kapan perubahan terjadi
- User: merupakan user database administrator yang melakukan proses data manipulation tersebut
- Old_data: merupakan value data yang sebelumnya
- New_data: merupakan value data terbaru yang diubah atau ditambah.

2. **Membuat Trigger After Insert**

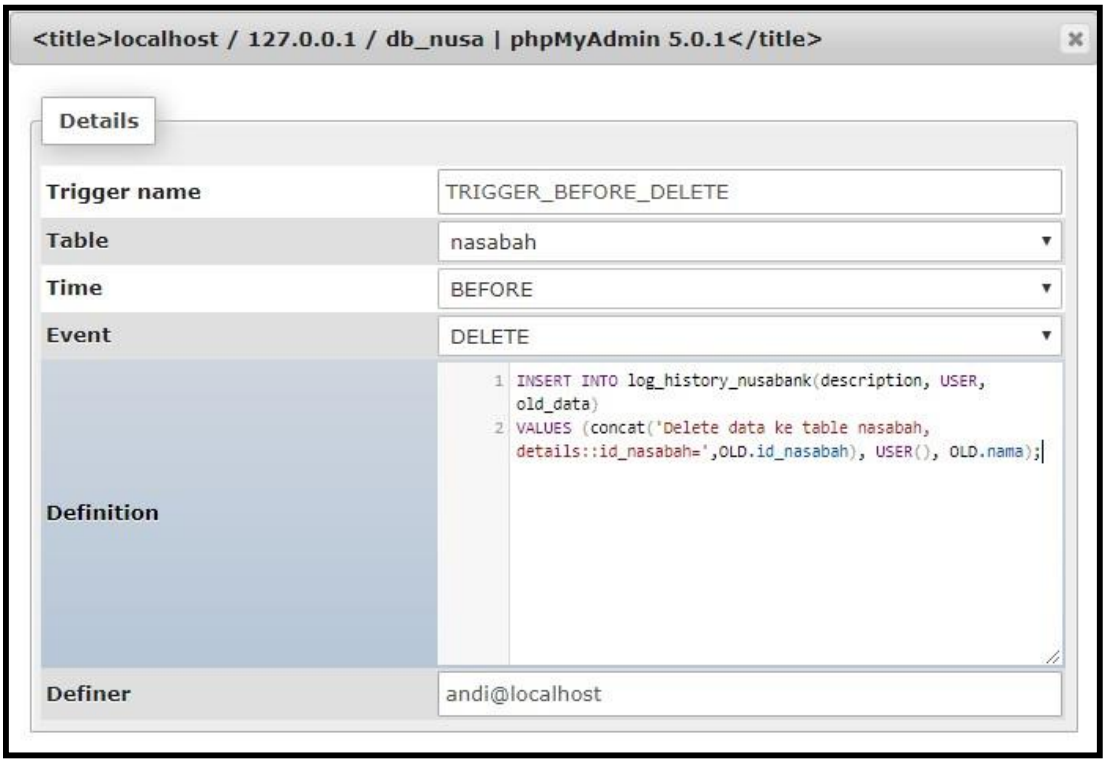



Disini trigger yang masuk akan mentrack perubahan setelah operas insert dilakukan, detilnya akan ditambahkan ke table log history, untuk sementara tracking perubahannya hanya pada id dan nama nasabah.

3. **Membuat Trigger After Update**




4. **Membuat Trigger Before Delete**



5. **Uji coba trigger**
   a. Test insert query ke table nasabah


   b. Test update query di table nasabah


   c. Delete duplikat nasabah Jon Bon Jovi
   d. Result trigger pada table log_history_nusabank

| description | log_time | user | old_data | new_data |
|---|---|---|---|---|
| Insert data ke table nasabah, details::id_nasabah=... | 2020-04-17 17:42:30 | root@localhost | | Vivaldi Winter |
| Update data ke table nasabah, details::id_nasabah=... | 2020-04-17 17:44:25 | root@localhost | Jon Bon Jovi | La Campanela |
| Delete data ke table nasabah, details::id_nasabah=... | 2020-04-17 17:45:27 | root@localhost | Jon Bon Jovi | |

- **Apakah sebaiknya kita mendelete row atau hanya menandai bahwa row tsb statusnya terdelete?**

  Untuk kasus dimana kita **benar-benar** sudah tidak membutuhkan lagi values dari row yang ingin didelete tersebut maka kita bisa cukup dengan mendelete row saja, namun apabila kita ingin memiliki **backup** atau sekiranya values dari data pada row tersebut masih ingin digunakan dikemudian hari, maka kita bisa melakukan **safely delete** ( akan saya coba jelaskan pada section pertanyaan selanjutnya) dan menandai status data terdelete pada **logs** history menggunakan **trigger on delete.**

- **Apa buruknya mendelete row dan bagaimana baiknya kita medelete row ?**

  Skenario terburuk dari mendelete row yang dilakukan sengaja atau tidak sengaja bukan dengan cara **safely delete** adalah kita tidak bisa mendapatkan backup atau tidak bisa merollback dataset yang sudah kita buat sebelumnya apalagi data tersebut memang benar-benar kredensial, memiliki relasi antar table dan memiliki integritas yang cukup tinggi. Hal ini tentu akan sangat membuat kita panic dan kerepotan.

  Cara terbaik untuk mendelete row adalah dengan metode **safely delete** yakni menggunakan **WHERE** clause supaya data yang terdelete spesifik namun hal ini masih

kurang, supaya kita dapat memiliki backup apabila kita melakukan kesalahan kita bisa mengeksekusi command **COMMIT** pada statement yang kita buat baik itu CREATE, INSERT, UPDATE, DELETE,  dsb. Sehingga kita bisa melakukan **rollback** atau recovery pengembalian sesi data sesuai dengan commit statement yang masuk terakhir kali, untuk merollback cukup menggunakan command **ROLLBACK** pada statement penggunaan COMMIT, ROLLBACK, dan TRIGGER ON DELETE dapat memberikan kita backup bersamaan dengan rincian log history perubahan yang ada pada table tersebut.