Advanced MySQL Topics

Maulana Hafidz Ismail

10 Oktober 2025

Contents

1 Functions and Triggers

1.1 Pengantar Functions dan Stored Procedures

- **Tujuan Utama**: **Mengen kapsulasi kode** dan meningkatkan **reusability** (`reuse kode`) dalam proyek `database`.
- **Manfaat**: Kode lebih `konsisten`, `terorganisir`, dan mudah `dipelihara`.
- **Nama Lain Fungsi**: Fungsi dalam MySQL juga disebut `stored function`.

1.1.1 Perbedaan Kunci: Functions vs. Stored Procedures

- **Pengembalian Nilai**: `Functions` **selalu mengembalikan satu nilai**; `stored procedures` **tidak selalu mengembalikan nilai**.
- **Parameter**: `Functions` hanya menerima `IN` parameters; `stored procedures` dapat menerima `IN`, `OUT`, dan `INOUT` parameters.
- **Cara Memanggil**: `Functions` dipanggil dalam pernyataan `SELECT` atau ekspresi; `Procedures` dipanggil dengan `CALL`.
- **Pembuatan**: `Functions` menggunakan `CREATE FUNCTION` dan wajib memiliki klausa `RETURNS`; `Procedures` menggunakan `CREATE PROCEDURE`.

```
-- Sintaks Stored Procedure
CREATE PROCEDURE procedure_name AS
    BEGIN
        -- logic here
    END;

-- Sintaks Pemanggilan Function
SELECT MOD(x, y);
```

Listing 1: Sintaks Dasar

—

1.2 Variabel dan Parameter dalam MySQL

1.2.1 Variabel MySQL

- **Variabel**: `Placeholder` yang menyimpan nilai untuk dioper antar pernyataan SQL.
- **Deklarasi**:
  - **Lokal**: Menggunakan `DECLARE variable_name datatype;` (Hanya berlaku di dalam `stored procedure`).

  - **Sesi**: Menggunakan `SET @variable_name = value;` (Dapat diakses di seluruh sesi).

- **Mengatur Nilai**: Menggunakan perintah `SET` atau operator penetapan `:=`.

```
CREATE PROCEDURE example_procedure()
BEGIN
    DECLARE total_cost DECIMAL(10,2);
    SET total_cost = 100.00;
    SET @discount = 10; -- Variabel Sesi
    total_cost := total_cost - @discount;
END;
```

Listing 2: Contoh Variabel di Stored Procedure

1.2.2 Mode Parameter dalam Stored Procedure

1. `IN` (Input): Digunakan untuk **mengirim nilai ke** prosedur. Nilai hanya dapat dibaca.

2. `OUT` (Output): Digunakan untuk **mengembalikan nilai dari** prosedur ke variabel di luar. Nilai parameter dapat diubah di dalam prosedur.

3. `INOUT` (Input/Output): Digunakan untuk **mengirim nilai dan mengembalikan nilai baru** yang telah diubah.

```
-- OUT Parameter: Mengembalikan nilai minimum cost
CREATE PROCEDURE get_lowest_cost(OUT lowest_cost DECIMAL
    (10,2))
BEGIN
  SELECT MIN(cost) INTO lowest_cost FROM orders;
END;
CALL get_lowest_cost(@lowest_cost);
SELECT @lowest_cost AS LowestCost;

-- INOUT Parameter: Mengkuadratkan angka
CREATE PROCEDURE square_number(INOUT number INT)
BEGIN
  SET number = number * number;
END;
SET @num = 5;
CALL square_number(@num);
SELECT @num AS SquaredValue; -- Output 25
```

Listing 3: Contoh Parameter OUT dan INOUT

—

1.3 User-Defined Functions (UDF)

1.3.1 Pembuatan Fungsi dan DETERMINISTIC

- **Sintaks**: Menggunakan `CREATE FUNCTION`, `RETURNS data_type`, dan `RETURN value`.
- **DETERMINISTIC**: Klausa ini menandakan bahwa fungsi **selalu mengembalikan hasil yang sama** untuk input yang sama. Ini vital untuk **Optimisasi Kinerja** (caching hasil) dan memungkinkan **Penggunaan dalam Indeks**. Tanpa ini, kinerja akan lebih lambat.

```
-- UDF Dasar (Diskon 10% tetap)
CREATE FUNCTION find_total_cost(cost DECIMAL)
    RETURNS DECIMAL(5,2) DETERMINISTIC
    BEGIN
        RETURN cost * 0.90;
    END;

-- UDF Kompleks (Diskon berdasarkan IF ELSE)
CREATE FUNCTION GetTotalCost(cost DECIMAL)
    RETURNS DECIMAL(5,2) DETERMINISTIC
    BEGIN
        DECLARE final_cost DECIMAL(5,2);
        IF cost >= 500 THEN
            SET final_cost = cost * 0.80; -- 20%
        ELSEIF cost >= 100 THEN
            SET final_cost = cost * 0.90; -- 10%
        ELSE
            SET final_cost = cost;
        END IF;
        RETURN final_cost;
    END;
```

Listing 4: Sintaks UDF Dasar dan Kompleks

```
SELECT GetTotalCost(500);
DROP FUNCTION function_name;
```

Listing 5: Pengujian dan Penghapusan Fungsi

—

1.4 Delimiter

1.4.1 Pengertian dan Fungsi Delimiter

- **Delimiter**: Karakter atau `string` yang menandai **akhir dari pernyataan SQL** (defaultnya adalah titik koma; `;`).

- **Mengapa Diubah**: Diperlukan saat membuat `stored programs` (`Function`/`Procedure`) yang berisi **beberapa pernyataan SQL** (masing-masing diakhiri `;`). Perubahan (*misalnya*, menjadi `//`) menghindari kesalahan sintaks dengan memastikan seluruh blok `BEGIN...END` dikirim dan dikompilasi sebagai satu unit.

```
DELIMITER // -- Ubah Delimiter

CREATE FUNCTION calculate_discount(price DECIMAL)
RETURNS DECIMAL(5,2)
BEGIN
    RETURN price * 0.90;
END // -- Blok diakhiri oleh delimiter baru

DELIMITER ; -- Kembalikan Delimiter ke default
```

Listing 6: Contoh Penggunaan DELIMITER

—

1.5 MySQL Trigger

1.5.1 Definisi dan Klasifikasi Trigger

- **TRIGGER**: Program tersimpan yang **diaktifkan secara otomatis** ketika peristiwa (`INSERT`, `UPDATE`, `DELETE`) terjadi pada tabel.
- **Klasifikasi**:
  - **Waktu**: `BEFORE` atau `AFTER`.
  - **Level**: `ROW LEVEL` (diaktifkan untuk setiap baris yang terpengaruh).
- **Manfaat**: Menjaga `log audit`, memastikan `integritas data` (validasi), dan mengotomatiskan tugas.

1.5.2 Sintaks dan Modifier OLD/NEW

- **Modifier** `OLD` dan `NEW`: Digunakan untuk mengakses nilai kolom. `OLD` adalah nilai **sebelum** operasi; `NEW` adalah nilai **setelah** operasi.

```
CREATE TRIGGER trigger_name
BEFORE|AFTER INSERT|UPDATE|DELETE
ON table_name FOR EACH ROW
BEGIN
    -- logic here (misal: validasi input negatif)
    IF NEW.order_quantity < 0 THEN
        SET NEW.order_quantity = 0; -- Memodifikasi nilai
            sebelum INSERT
    END IF;
```

```
END;

-- Menghapus Trigger
DROP TRIGGER IF EXISTS schema_name.trigger_name;
```

Listing 7: Sintaks Pembuatan Trigger

—

1.6 MySQL Scheduled Events

1.6.1 Definisi dan Tipe Events

- **Scheduled Event**: Tugas yang dieksekusi **sesuai jadwal tertentu**.
- **Tipe**:
  1. **One Time Events**: Terjadi hanya sekali (`ON SCHEDULE AT timestamp`).
  2. **Recurring Events**: Terjadi secara berkala (`ON SCHEDULE EVERY unit`).

1.6.2 Sintaks Scheduled Event

```
-- Event Satu Kali (12 jam dari sekarang)
CREATE EVENT GenerateRevenueReport
ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 12 HOUR
DO
BEGIN
    -- logic here
END;

-- Event Berulang (setiap 1 hari)
CREATE EVENT IF NOT EXISTS daily_restock
ON SCHEDULE EVERY 1 DAY
STARTS CURRENT_TIMESTAMP() + INTERVAL 20 DAY
ENDS CURRENT_TIMESTAMP() + INTERVAL 40 DAY -- Opsional
DO
BEGIN
  -- logic untuk memeriksa dan memperbarui stok
END;

-- Menghapus Event
DROP EVENT IF EXISTS event_name;
```

Listing 8: Sintaks Pembuatan Scheduled Event

1.7 Keuntungan dan Kerugian dari Triggers

1.7.1 Keuntungan (Advantages)

- **Validasi Data**: Memungkinkan `validasi data` sebelum dimasukkan atau diperbarui, yang membantu menjaga integritas data.
- **Otomatisasi Tugas**: Menyediakan cara alternatif untuk `menjalankan tugas terjadwal` atau melakukan tugas secara otomatis berdasarkan tindakan tertentu pada tabel.
- **Peningkatan Kinerja Query**: Meningkatkan kinerja kueri SQL karena kode trigger **tidak perlu dikompilasi setiap kali** kueri dieksekusi (setelah kompilasi pertama).
- **Mengurangi Kode Sisi Klien**: Mengurangi kebutuhan untuk membuat kode validasi yang berulang di `sisi klien`, yang menghemat waktu dan usaha pengembangan.

1.7.2 Kerugian (Disadvantages)

- **Keterbatasan Validasi**: Tidak semua jenis validasi batasan (`constraint validation`) dapat dilakukan menggunakan trigger.
- **Sulit Dipecahkan Masalahnya (`Troubleshooting`)**: Sulit untuk melakukan `troubleshooting` karena trigger beroperasi secara implisit di `lapisan database` (*database layer*).
- **Beban Server**: Dapat `meningkatkan beban` pada server database karena trigger diaktifkan secara otomatis dan menjalankan logika tambahan.

2 Database Optimization

2.1 Konsep Database Optimization

- **Database Optimization** adalah proses meningkatkan kinerja sistem `database` untuk **mengurangi waktu** yang dibutuhkan untuk menjalankan `query`, memproses, dan mengirimkan hasil.
- **Pentingnya**: Memastikan `query SQL` diproses dan data dikembalikan dengan cepat.

2.1.1 Jenis Pernyataan SQL

- **Data Retrieval Statements** (`SELECT`): Mengembalikan data dari `database`.
- **Data Change Statements** (`INSERT`, `UPDATE`, `DELETE`): Mengubah data dalam `database`.

—

2.2 Teknik Optimasi Kueri SQL

2.2.1 Optimasi SELECT Statements (Data Retrieval)

1. **Targetkan Kolom**: Targetkan **hanya kolom yang diperlukan**. Hindari menggunakan `*` jika tidak membutuhkan semua kolom.
2. **Hindari Fungsi di `WHERE`**: Hindari menggunakan `fungsi` dalam `predikat` (kondisi klausa `WHERE`) pada kolom yang `diindeks` karena **mencegah `database` menggunakan indeks**.
3. **Wildcard Efisien**: Hindari penggunaan `wildcard` di awal dalam `predikat` (*misalnya*, `LIKE '%kata'`) karena **tidak dapat menggunakan indeks**. Solusi: Tambahkan kolom terbalik (*reverse_full_name*) dan buat indeks padanya untuk memungkinkan `LIKE 'kata%'`.
4. **JOIN**: Gunakan `INNER JOIN` jika memungkinkan, karena lebih efisien daripada `OUTER JOIN` (`LEFT/RIGHT JOIN`) yang juga mengembalikan catatan yang tidak cocok.
5. **Set Operations**: Gunakan `UNION ALL` daripada `UNION` jika duplikat diizinkan, karena `UNION ALL` menghindari operasi pengurutan yang mahal. Gunakan `DISTINCT` dan `UNION` hanya saat diperlukan.

2.2.2 Optimasi Data Change Statements

- `UPDATE/DELETE`: Optimalkan `kondisi dalam klausa WHERE` (terutama dengan indeks yang sesuai).

- `INSERT`: Optimasi dilakukan dengan melakukan `batch inserts` (menyisipkan lebih dari satu baris dalam satu operasi `INSERT`).

—

2.3 Index (Indeks)

2.3.1 Konsep Dasar Index

- **Definisi**: Struktur data yang membantu mempertahankan `pointer` ke data yang terurut untuk **mempercepat pencarian data**.
- **Kapan Digunakan**: Dibuat pada kolom tabel yang **sering digunakan untuk memfilter query** `SELECT`.

2.3.2 Jenis-Jenis Index

1. **Primary Index** (Clustered Index): Dihasilkan **secara otomatis** saat tabel dibuat dengan `PRIMARY KEY` atau `UNIQUE KEY`. Data `index` disimpan dalam tabel itu sendiri.
2. **Secondary Index**: Diciptakan secara manual menggunakan pernyataan `CREATE INDEX`. Dapat dibuat menggunakan satu atau lebih kolom.

2.3.3 Operasi Query yang Menggunakan Index

Indeks dapat meningkatkan kinerja pada:

- **Filtering rows**: Klausa `WHERE`.
- `JOIN` **operations**: Kolom yang digunakan dalam `JOIN` harus memiliki tipe data yang sama untuk kinerja terbaik.
- **Fungsi** `MIN` **dan** `MAX`.
- `Sorting and grouping` **operations** (`ORDER BY` dan `GROUP BY`).

2.3.4 Sintaks Pembuatan Index

```
CREATE INDEX idx_column_name ON table_name (column_name);

-- Contoh:
CREATE INDEX IdxFullName ON Clients(FullName);
```

Listing 9: Membuat Secondary Index

—

2.4 Menganalisis Kueri dengan EXPLAIN

2.4.1 Konsep EXPLAIN

- **EXPLAIN**: Pernyataan yang digunakan untuk **melihat rencana eksekusi** `query` yang optimal yang dibuat oleh `MySQL Query Optimizer`.
- **Tujuan**: Membantu mengidentifikasi masalah kinerja dan bagaimana `database` mengakses data.

```
EXPLAIN SELECT column_name FROM table_name WHERE VALUE;
```

Listing 10: Sintaks EXPLAIN

2.4.2 Kolom Penting dalam Hasil EXPLAIN

Table 1: Penjelasan Kolom Hasil EXPLAIN

Kolom	**Penjelasan Singkat**
`ID`	Pengidentifikasi urut untuk setiap pernyataan `SELECT` (lebih berarti pada `subquery`).
`SELECT_TYPE`	Jenis `SELECT` yang dieksekusi (*misalnya*, `SIMPLE`, `PRIMARY`, `SUBQUERY`, `UNION`).
`table`	Nama tabel yang dirujuk.
`type`	**Cara pencarian dilakukan** (akses data). Nilai terburuk adalah `all` (pemindaian seluru
`possible_keys`	Kunci yang **dapat** digunakan oleh MySQL.
`key`	Indeks yang **sebenarnya** digunakan oleh MySQL (`NULL` berarti tidak ada indeks yang dig
`rows`	Jumlah catatan yang diperiksa (**semakin kecil, semakin efisien**).
`filtered`	Persentase baris yang difilter oleh kondisi (**semakin tinggi persentase, semakin baik**)
`Extra`	Informasi tambahan (*misalnya*, `Using temporary` atau `Using filesort` menunjukkan ku

2.4.3 Nilai type dan Efisiensi Akses Data

- **Akses Terbaik**: `system`, `const`, `eq_ref`.
- **Akses Memadai**: `ref` (kolom diindeks diakses dengan operator kesetaraan), `index` (seluruh indeks dipindai).
- **Akses Terburuk**: `all` (pemindaian **seluruh tabel**) → biasanya mengindikasikan kurangnya indeks yang sesuai.