Fondasi Basis Data: Menguasai CRUD pada Tingkat Tabel

Muhammad Riduwan Khafidi
Program Studi Teknik Rekayasa Perangkat Lunak
Politeknik Negeri Batam
4342611034@students.polibatam.ac.id

November 5, 2026

Abstrak

Tabel adalah unit struktural terkecil yang utuh di dalam basis data relasional. Setelah database dibuat, seluruh pekerjaan berikutnya berkutat pada tabel: mendefinisikan kolom, menetapkan batasan, menambahkan indeks, mengubah struktur saat kebutuhan berubah, dan pada akhirnya menghapus tabel yang sudah tidak terpakai. Makalah ini membahas CRUD pada tingkat tabel secara menyeluruh, mulai dari `CREATE TABLE` dengan seluruh klausa batasannya, `SHOW TABLES` dan `DESCRIBE` sebagai jendela pengamatan struktur, `ALTER TABLE` dengan dua belas varian operasinya, hingga `DROP TABLE` dan `TRUNCATE TABLE` beserta perbedaan konsekuensinya. Setiap perintah dilengkapi diagram anatomi, simulasi keluaran, dan skenario yang lazim dijumpai di dunia kerja. Makalah ini melanjutkan makalah CRUD Database dan menjadi fondasi sebelum pembaca melangkah ke CRUD pada tingkat baris.

1 Pendahuluan: Tabel sebagai Unit Struktural

1.1 Analogi Lemari Arsip

Database adalah ruangan. Tabel adalah lemari arsip di dalamnya. Setiap lemari punya laci-laci bernomor, dan setiap laci diberi label yang jelas: nomor, nama, tanggal, isi. Sebelum kertas dimasukkan ke dalam laci, lemarinya harus dibangun dulu, lacinya ditentukan jumlahnya, dan labelnya dipasang.

Dalam basis data, membangun lemari itu disebut `CREATE TABLE`. Menentukan jumlah dan nama laci disebut mendefinisikan kolom. Memberi aturan bahwa laci nomor tidak boleh kosong disebut `NOT NULL`. Memberi aturan bahwa satu nomor hanya boleh muncul sekali disebut `PRIMARY KEY`. Semua itu terjadi sebelum satu baris data pun masuk.

1.2 Batas antara Struktur dan Isi

Penting membedakan dua hal yang sering tertukar:

- Struktur tabel: kolom apa saja yang ada, tipe datanya apa, batasannya apa, indeksnya apa. Ini wilayah `CREATE`, `ALTER`, dan `DROP`.

- Isi tabel: baris-baris data yang tersimpan. Ini wilayah `INSERT`, `SELECT`, `UPDATE`, dan `DELETE`.

Makalah ini fokus pada struktur. Isi akan dibahas di makalah berikutnya. Namun keduanya bertemu di satu titik: `TRUNCATE TABLE` menghapus seluruh isi tetapi mempertahankan struktur, sedangkan `DROP TABLE` menghapus keduanya sekaligus.

1.3 Siklus Hidup CRUD Tabel

Gambar **??** merangkum empat operasi yang akan dibahas sepanjang makalah ini.

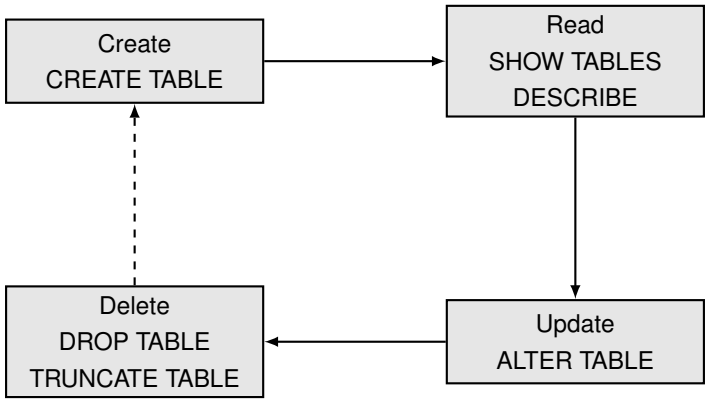


Figure 1. Siklus hidup CRUD pada tingkat tabel.

Garis putus-putus dari `DROP` kembali ke `CREATE` menandakan skenario umum di dunia kerja: struktur tabel berubah drastis sehingga lebih mudah menghapus dan membuat ulang daripada meng-ubah satu per satu.

2 Konteks: Berada di Dalam Database

Sebelum semua perintah tabel bisa dijalankan, kita harus berada di dalam sebuah database. Ini berbeda dengan perintah database yang bisa dijalankan dari mana saja.

```
USE toko_online;
SELECT DATABASE();
```

```
+--------------+
| DATABASE()   |
+--------------+
| toko_online  |
+--------------+
```

Jika `SELECT DATABASE()` mengembalikan `NULL`, artinya belum ada database yang dipilih, dan perintah tabel akan gagal dengan pesan `No database selected`.

3 Create: Mendirikan Tabel

3.1 Sintaks Dasar

Bentuk minimal `CREATE TABLE` membutuhkan setidaknya satu kolom. Tanpa kolom, MariaDB menolak.

```
CREATE TABLE nama_tabel (
    nama_kolom1 tipe_data [batasan],
    nama_kolom2 tipe_data [batasan],
    ...
    [batasan tingkat tabel]
);
```

Jika dicoba tanpa kolom, hasilnya seperti ini.

```
CREATE TABLE kosong;
```

```
ERROR 1113 (42000): A table must have at least 1 column
```

3.2 Anatomi Satu Kolom

Setiap kolom terdiri dari tiga bagian: nama, tipe data, dan batasan opsional. Gambar **??** memvisualkan anatomi tersebut pada satu tabel contoh.

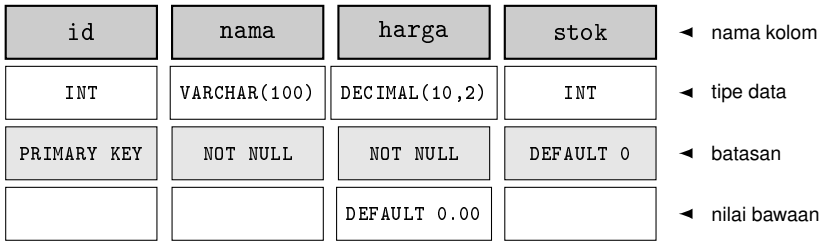


Figure 2. Anatomi kolom pada sebuah tabel.

Bagian atas adalah nama kolom yang dipakai untuk merujuk kolom di query. Bagian tengah adalah tipe data yang sudah dibahas mendalam pada makalah sebelumnya. Bagian bawah adalah batasan yang menentukan aturan main kolom itu.

3.3 Batasan (Constraint) yang Perlu Dikuasai

MariaDB mendukung beberapa batasan utama di tingkat kolom dan tabel. Tabel **??** merangkumnya.

Table 1. Batasan (constraint) pada tabel MariaDB.

Batasan	Tingkat	Fungsi
`NOT NULL`	kolom	kolom tidak boleh kosong
`DEFAULT`	kolom	nilai bawaan jika tidak diisi
`PRIMARY KEY`	kolom atau tabel	kunci unik dan tidak boleh kosong
`UNIQUE`	kolom atau tabel	nilai harus unik, boleh kosong
`CHECK`	kolom atau tabel	nilai harus memenuhi kondisi
`AUTO_INCREMENT`	kolom	isi otomatis dengan angka naik
`FOREIGN KEY`	tabel	merujuk ke primary key tabel lain
`REFERENCES`	kolom	bagian dari sintaks foreign key

Batasan di tingkat kolom ditulis langsung setelah tipe data. Batasan di tingkat tabel ditulis sebagai klausa tersendiri di akhir definisi, dan bisa merujuk lebih dari satu kolom. Contoh batasan tingkat tabel adalah `PRIMARY KEY (a, b)` untuk composite primary key.

3.4 Varian `CREATE TABLE`

MariaDB menyediakan beberapa varian yang berguna di skenario berbeda. Tabel **??** merangkumnya.

Table 2. Varian `CREATE TABLE`.

Varian	Kegunaan
`CREATE TABLE nama (...)`	membuat tabel baru dari nol
`CREATE TABLE IF NOT EXISTS`	aman untuk skrip berulang
`CREATE TEMPORARY TABLE`	tabel sementara, hilang saat sesi ditutup
`CREATE TABLE baru LIKE lama`	menyalin struktur tanpa data
`CREATE TABLE baru AS SELECT`	menyalin struktur dan data dari hasil query

Varian `LIKE` sangat berguna untuk mengkloning tabel di lingkungan pengujian. Varian `AS SELECT` dipakai untuk membuat tabel laporan atau tabel ringkasan dari tabel sumber.

3.5 Contoh Lengkap

Berikut adalah perintah `CREATE TABLE` yang mendirikan tabel `produk` dengan tujuh kolom, masing-masing dengan alasan pemilihan tipe dan batasannya.

```
CREATE TABLE produk (
    id          BIGINT UNSIGNED  DEFAULT (UUID_SHORT()) PRIMARY KEY,
    nama        VARCHAR(150)     NOT NULL,
    harga       DECIMAL(12,2)    NOT NULL DEFAULT 0.00,
    stok        INT UNSIGNED     NOT NULL DEFAULT 0,
    deskripsi   TEXT,
    is_active   BOOLEAN          NOT NULL DEFAULT TRUE,
```

```
    created_at  DATETIME         DEFAULT CURRENT_TIMESTAMP
) COMMENT='Katalog produk toko online';
```

Keluaran jika berhasil.

```
Query OK, 0 rows affected
```

Kalimat `0 rows affected` konsisten dengan sifat operasi DDL (Data Definition Language) yang tidak menyentuh baris data. Angka nol itu mengacu pada baris, bukan pada tabel yang baru saja dibuat.

3.6 Penjelasan Setiap Kolom

- `id BIGINT UNSIGNED DEFAULT (UUID_SHORT()) PRIMARY KEY`: kunci utama 8 byte dengan nilai yang selalu bertambah secara monotonik, menghindari fragmentasi indeks.
- `nama VARCHAR(150) NOT NULL`: nama produk wajib diisi, panjang bervariasi, dibatasi 150 karakter agar indeks tetap ringan.
- `harga DECIMAL(12,2) NOT NULL DEFAULT 0.00`: uang wajib exact numeric, presisi 12 digit, 2 desimal.
- `stok INT UNSIGNED NOT NULL DEFAULT 0`: jumlah barang dalam stok, tidak boleh negatif, diisi nol secara bawaan.
- `deskripsi TEXT`: deskripsi panjang yang boleh kosong.
- `is_active BOOLEAN NOT NULL DEFAULT TRUE`: penanda aktif atau tidak, otomatis aktif saat dibuat.
- `created_at DATETIME DEFAULT CURRENT_TIMESTAMP`: waktu pembuatan baris, otomatis terisi.

Perhatikan bahwa semua kolom yang wajib diisi diberi `NOT NULL`, dan semua kolom yang punya nilai wajar diberi `DEFAULT`. Hasilnya, `INSERT` hanya perlu menyebut kolom yang benar-benar esensial.

3.7 Alur Keputusan Sebelum `CREATE TABLE`

Gambar **??** menyajikan alur singkat sebelum mengeksekusi `CREATE TABLE`.

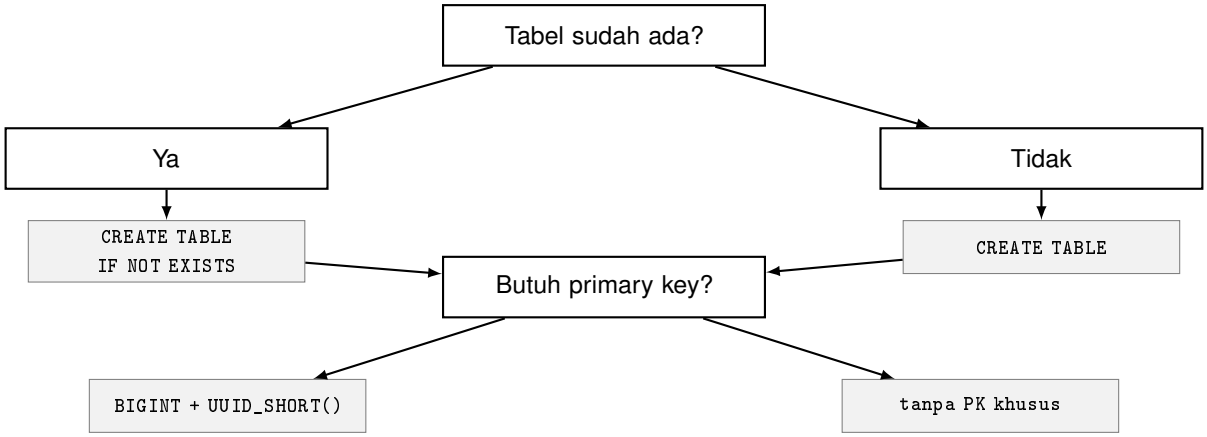


Figure 3. Alur keputusan sebelum `CREATE TABLE`.

4 Read: Mengintip Struktur Tabel

Operasi Read pada tingkat tabel tidak menampilkan isi baris (itu `SELECT`), melainkan menampilkan struktur. Ada empat perintah yang bekerja sama untuk tujuan ini, masing-masing dengan tingkat kedalaman yang berbeda.

4.1 Melihat Daftar Tabel

```
SHOW TABLES;
```

```
+-----------------------+
| Tables_in_toko_online |
+-----------------------+
| pelanggan             |
| pesanan               |
| produk                |
+-----------------------+
3 rows in set (0.001 sec)
```

Untuk menyaring berdasarkan pola, gunakan `LIKE`.

```
SHOW TABLES LIKE 'prod%';
```

4.2 Melihat Kolom dengan `DESCRIBE`

```
DESCRIBE produk;
```

```
+------------+-----------------+------+-----+------------------+
| Field      | Type            | Null | Key | Default          |
+------------+-----------------+------+-----+------------------+
| id         | bigint unsigned | NO   | PRI | NULL             |
| nama       | varchar(150)    | NO   |     | NULL             |
| harga      | decimal(12,2)   | NO   |     | 0.00             |
| stok       | int unsigned    | NO   |     | 0                |
| deskripsi  | text            | YES  |     | NULL             |
| is_active  | tinyint(1)      | NO   |     | 1                |
| created_at | datetime        | YES  |     | current_timestamp|
+------------+-----------------+------+-----+------------------+
7 rows in set (0.002 sec)
```

Perhatikan bahwa `BOOLEAN` muncul sebagai `tinyint(1)`. Ini mengonfirmasi bahwa `BOOLEAN` hanyalah alias, bukan tipe tersendiri. Kolom `Key` menampilkan `PRI` untuk primary key, `UNI` untuk unique, dan `MUL` untuk indeks biasa.

Singkatan `DESC` bisa dipakai sebagai alternatif.

```
DESC produk;
SHOW COLUMNS FROM produk;
```

Ketiga perintah menghasilkan keluaran yang sama.

4.3 Melihat Definisi Lengkap dengan `SHOW CREATE TABLE`

Jika `DESCRIBE` hanya menampilkan ringkasan, `SHOW CREATE TABLE` menampilkan perintah `CREATE TABLE` yang akan dijalankan untuk membangun tabel persis seperti sekarang.

```
SHOW CREATE TABLE produk\G
```

```
*************************** 1. row ***************************
       Table: produk
Create Table: CREATE TABLE `produk` (
  `id` bigint(20) unsigned NOT NULL DEFAULT uuid_short(),
  `nama` varchar(150) NOT NULL,
  `harga` decimal(12,2) NOT NULL DEFAULT 0.00,
  `stok` int(10) unsigned NOT NULL DEFAULT 0,
  `deskripsi` text DEFAULT NULL,
  `is_active` tinyint(1) NOT NULL DEFAULT 1,
  `created_at` datetime DEFAULT current_timestamp(),
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
  COLLATE=utf8mb4_unicode_ci
  COMMENT='Katalog produk toko online'
```

Perhatikan tulisan `\G` di akhir perintah. Ini mengganti titik koma dan membuat keluaran ditampilkan secara vertikal, jauh lebih mudah dibaca untuk tabel lebar.

Keluaran `SHOW CREATE TABLE` menampilkan beberapa informasi tambahan yang tidak muncul di `DESCRIBE`, yaitu `ENGINE`, `CHARSET`, `COLLATE`, dan `COMMENT`.

4.4 Melihat Indeks dengan `SHOW INDEX`

```
SHOW INDEX FROM produk;
```

```
+--------+------------+----------+--------------+-------------+
| Table  | Non_unique | Key_name | Seq_in_index | Column_name |
+--------+------------+----------+--------------+-------------+
| produk |          0 | PRIMARY  |            1 | id          |
+--------+------------+----------+--------------+-------------+
1 row in set
```

Kolom `Non_unique` bernilai `0` menandakan indeks unik. Primary key selalu unik. Indeks biasa akan bernilai `1`.

4.5 Menelusuri Katalog dengan `information_schema`

Untuk kebutuhan otomatisasi, hasil `SHOW` bisa diperoleh lewat query SQL ke `information_schema`, yang merupakan katalog metadata server.

```
SELECT table_name, engine, table_rows,
       data_length, index_length
FROM information_schema.tables
WHERE table_schema = 'toko_online';
```

```
+------------+--------+------------+-------------+--------------+
| table_name | engine | table_rows | data_length | index_length |
+------------+--------+------------+-------------+--------------+
| pelanggan  | InnoDB |         14 |       16384 |        16384 |
| pesanan    | InnoDB |         52 |       32768 |        16384 |
| produk     | InnoDB |          8 |       16384 |        16384 |
+------------+--------+------------+-------------+--------------+
```

Perintah `SHOW` cocok untuk dipakai secara manual di terminal. Query `information_schema` cocok untuk dipakai di dalam aplikasi karena hasilnya bisa difilter, diurutkan, dan digabung dengan query lain.

4.6 Perbandingan Empat Cara Melihat Struktur

Gambar **??** merangkum kapan sebaiknya memakai perintah yang mana.

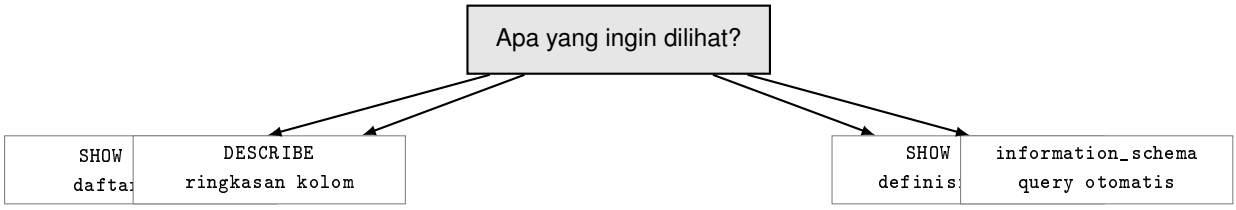


Figure 4. Empat jalur untuk melihat struktur tabel sesuai kebutuhan.

5 Update: Mengubah Struktur dengan `ALTER TABLE`

Inilah bagian terpanjang dari CRUD tabel. `ALTER TABLE` memiliki lebih dari sepuluh varian operasi, dan satu perintah bisa menggabungkan beberapa operasi sekaligus. Semua contoh di bawah menggunakan tabel `produk` dari bagian sebelumnya.

5.1 Kerangka Umum

```
ALTER TABLE nama_tabel
    operasi_1,
    operasi_2,
    ...
    operasi_n;
```

Setiap operasi dipisahkan koma. MariaDB akan menjalankan semuanya dalam satu transaksi DDL, artinya jika salah satu gagal, seluruh perubahan dibatalkan.

5.2 Menambah Kolom: `ADD COLUMN`

```
ALTER TABLE produk
    ADD COLUMN berat_gram INT UNSIGNED DEFAULT 0;
```

Kolom baru otomatis ditempatkan di akhir. Untuk menempatkannya di posisi tertentu, gunakan `FIRST` atau `AFTER`.

```
ALTER TABLE produk
    ADD COLUMN sku VARCHAR(20) NOT NULL AFTER id;
ALTER TABLE produk
    ADD COLUMN catatan TEXT FIRST;
```

Gambar **??** mengilustrasikan perubahan struktur sebelum dan sesudah `ADD COLUMN`.

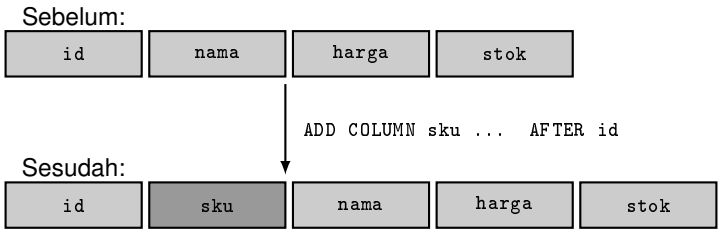


Figure 5. Efek `ADD COLUMN ... AFTER` pada struktur tabel.

5.3 Menghapus Kolom: `DROP COLUMN`

```
ALTER TABLE produk DROP COLUMN catatan;
```

Operasi ini menghapus kolom beserta seluruh datanya. Tidak ada undo. Jika kolom itu bagian dari indeks, indeks tersebut ikut terhapus atau perlu didefinisikan ulang. Jika kolom itu dirujuk oleh foreign key dari tabel lain, operasi akan ditolak.

5.4 Mengubah Definisi Kolom: `MODIFY COLUMN`

`MODIFY` mengubah tipe data, batasan, atau posisi kolom, tanpa mengubah namanya.

```
ALTER TABLE produk
    MODIFY COLUMN nama VARCHAR(200) NOT NULL;
```

Jika tipe data baru lebih kecil dan ada data yang tidak muat, MariaDB menolak operasi dengan pesan `Data truncated for column`. Jika tipe data baru lebih besar, operasi berhasil dan data lama tetap utuh.

5.5 Mengubah Nama Kolom: `CHANGE COLUMN`

`CHANGE` mengubah nama sekaligus definisi. Kedua hal harus disebutkan meskipun hanya salah satu yang berubah.

```
ALTER TABLE produk
    CHANGE COLUMN berat_gram berat INT UNSIGNED DEFAULT 0;
```

Perbedaan filosofis `MODIFY` dan `CHANGE` ditunjukkan pada Gambar **??**.

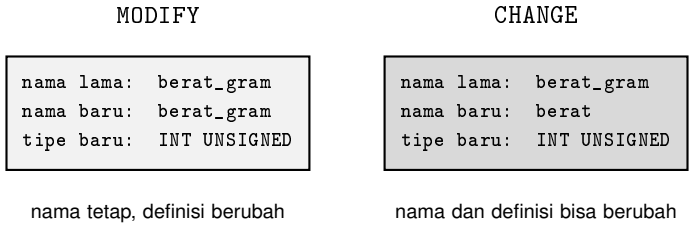


Figure 6. Perbedaan `MODIFY COLUMN` dan `CHANGE COLUMN`.

5.6 Mengganti Nama Tabel: `RENAME TO`

```
ALTER TABLE produk RENAME TO produk_lama;
ALTER TABLE produk_lama RENAME TO produk;
```

Operasi ini hanya mengganti nama, tidak menyentuh struktur maupun data. Berbeda dengan database yang tidak punya `RENAME`, tabel MariaDB mendukung `RENAME` secara langsung.

5.7 Menambah dan Menghapus Indeks

Indeks adalah struktur tambahan yang mempercepat pencarian pada kolom yang sering difilter.

```
ALTER TABLE produk
    ADD INDEX idx_harga (harga);
ALTER TABLE produk
    DROP INDEX idx_harga;
```

Untuk indeks unik, gunakan `ADD UNIQUE`.

```
ALTER TABLE produk
    ADD UNIQUE idx_sku (sku);
```

Gambar **??** menunjukkan mengapa indeks mempercepat pencarian.

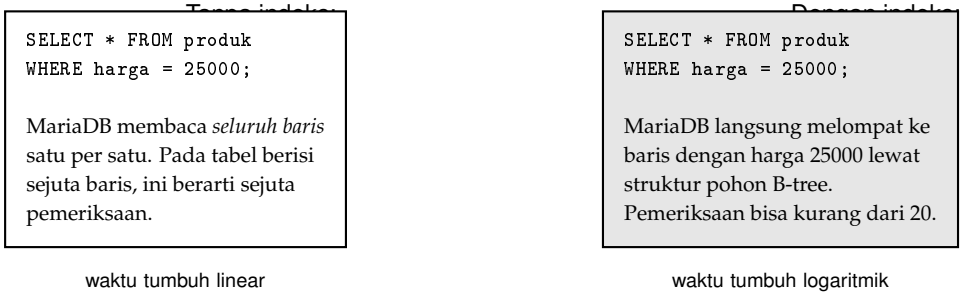


Figure 7. Efek indeks pada kecepatan pencarian.

5.8 Menambah dan Menghapus Primary Key

Tabel yang dibuat tanpa primary key bisa ditambahi setelahnya, asalkan kolomnya memenuhi syarat unik dan tidak ada nilai `NULL`.

```
ALTER TABLE produk ADD PRIMARY KEY (id);
ALTER TABLE produk DROP PRIMARY KEY;
```

Karena satu tabel hanya boleh punya satu primary key, `DROP PRIMARY KEY` selalu menghapus satu-satunya primary key yang ada. Jika kolomnya juga `AUTO_INCREMENT`, operasi akan ditolak sampai `AUTO_INCREMENT` dilepas lebih dulu.

5.9 Menambah dan Menghapus Foreign Key

```
ALTER TABLE pesanan
    ADD CONSTRAINT fk_pesanan_produk
        FOREIGN KEY (produk_id) REFERENCES produk(id);
ALTER TABLE pesanan DROP FOREIGN KEY fk_pesanan_produk;
```

Foreign key memaksa nilai `produk_id` di tabel `pesanan` merujuk ke salah satu `id` yang ada di tabel `produk`. Ini menjaga integritas referensial, tetapi juga berarti bahwa `DROP TABLE produk` akan ditolak selama masih ada tabel lain yang merujuknya.

5.10 Menggabungkan Beberapa Operasi

Salah satu keunggulan `ALTER TABLE` adalah kemampuannya menggabungkan beberapa operasi dalam satu perintah, sehingga tabel hanya dibaca dan ditulis ulang satu kali.

```
ALTER TABLE produk
    ADD COLUMN sku VARCHAR(20) NOT NULL FIRST,
    MODIFY COLUMN nama VARCHAR(200) NOT NULL,
    ADD INDEX idx_harga (harga),
    DROP COLUMN catatan;
```

Pada tabel yang sudah besar, penggabungan seperti ini bisa menghemat puluhan menit karena MariaDB tidak perlu membangun ulang tabel untuk setiap operasi terpisah.

5.11 Ringkasan Dua Belas Varian `ALTER TABLE`

Table 3. Ringkasan varian `ALTER TABLE` yang perlu dikuasai.

Operasi	Sintaks singkat
Tambah kolom	`ADD COLUMN nama tipe`
Tambah kolom di awal	`ADD COLUMN nama tipe FIRST`
Tambah kolom di posisi tertentu	`ADD COLUMN nama tipe AFTER kolom`
Hapus kolom	`DROP COLUMN nama`
Ubah definisi kolom	`MODIFY COLUMN nama tipe_baru`
Ubah nama dan definisi	`CHANGE COLUMN lama baru tipe`
Ganti nama tabel	`RENAME TO nama_baru`
Tambah indeks	`ADD INDEX nama (kolom)`
Hapus indeks	`DROP INDEX nama`
Tambah primary key	`ADD PRIMARY KEY (kolom)`
Hapus primary key	`DROP PRIMARY KEY`
Tambah foreign key	`ADD CONSTRAINT nama FOREIGN KEY (kolom) REFERENCES tabel(kolom)`
Hapus foreign key	`DROP FOREIGN KEY nama`

6 Delete: Menghapus Tabel

MariaDB membedakan dua operasi penghapusan tabel. Keduanya berawalan kata yang sama, tetapi konsekuensinya sangat berbeda.

6.1 `DROP TABLE`: Menghapus Struktur dan Isi

```
DROP TABLE [IF EXISTS] nama_tabel;
```

Perintah ini menghapus tabel beserta seluruh baris, indeks, trigger, dan definisinya. Setelah `DROP`, nama tabel itu hilang dari katalog.

```
DROP TABLE produk;
```

```
Query OK, 0 rows affected
```

Jika tabel tidak ada, tanpa `IF EXISTS` MariaDB menolak.

```
ERROR 1051 (42S02): Unknown table 'toko_online.produk'
```

Dengan `IF EXISTS`, penghapusan yang gagal dianggap sukses.

```
DROP TABLE IF EXISTS produk;
```

Untuk menghapus beberapa tabel sekaligus, MariaDB mengizinkan penulisan dalam satu perintah.

```
DROP TABLE IF EXISTS produk, pelanggan, pesanan;
```

6.2 `TRUNCATE TABLE`: Menghapus Isi Saja

```
TRUNCATE TABLE nama_tabel;
```

Perintah ini menghapus seluruh baris tetapi mempertahankan struktur tabel. Setelah `TRUNCATE`, tabel masih ada, kolomnya masih ada, indeksnya masih ada, hanya isinya kosong.

```
TRUNCATE TABLE produk;
```

```
Query OK, 0 rows affected
```

`TRUNCATE` juga mereset `AUTO_INCREMENT` ke nilai awal. Jika tabel punya kolom `AUTO_INCREMENT` dan sebelumnya sudah berisi baris dengan ID hingga 100, setelah `TRUNCATE` baris berikutnya akan kembali ID 1.

6.3 Perbandingan `DROP`, `TRUNCATE`, dan `DELETE`

Ketiga perintah ini sering tertukar. Tabel **??** merangkum perbedaannya secara lengkap.

Table 4. Perbandingan `DROP`, `TRUNCATE`, dan `DELETE`.

Aspek	`DROP TABLE`	`TRUNCATE TABLE`	`DELETE FROM`
Yang dihapus	struktur dan semua baris	semua baris	baris tertentu atau semua
Struktur tabel	hilang	tetap	tetap
Indeks, trigger	hilang	tetap	tetap
Klausa `WHERE`	tidak bisa	tidak bisa	bisa
Bisa rollback	tidak	tidak	ya, dalam transaksi
Kecepatan	cepat	sangat cepat	lambat, per baris
`AUTO_INCREMENT`	hilang	reset ke awal	tidak reset
Trigger `DELETE`	tidak dijalankan	tidak dijalankan	dijalankan
Kapan dipakai	tabel sudah tidak diperlukan	kosongkan tabel, pakai ulang	hapus baris selektif

Gambar **??** memberikan visualisasi perbedaan yang sama.



Figure 8. Tiga perintah penghapusan dan hasil akhirnya.

Pemilihan ketiganya bergantung pada tujuan. Jika ingin menyingkirkan tabel dari skema, pakai `DROP`. Jika ingin mengosongkan tabel untuk diisi ulang, pakai `TRUNCATE`. Jika ingin menghapus sebagian baris atau butuh `WHERE`, pakai `DELETE`.

6.4 Prosedur Aman Sebelum `DROP TABLE`

Menghapus tabel dari database produksi adalah salah satu operasi paling berisiko. Gambar **??** menyajikan prosedur minimal.

1. Periksa foreign key: `SHOW CREATE TABLE produk` dan cari `REFERENCES`
↓
2. Periksa trigger: `SHOW TRIGGERS LIKE 'produk'`
↓
3. Cadangkan: `mysqldump -u root -p toko_online produk > produk.sql`
↓
4. Verifikasi berkas cadangan: `ls -lh produk.sql`
↓
5. Baru jalankan `DROP TABLE produk;`

Figure 9. Prosedur aman sebelum `DROP TABLE`.

Langkah 1 penting karena `DROP TABLE` akan gagal jika ada tabel lain yang merujuknya lewat foreign key. Pesan yang muncul seperti berikut.

```
ERROR 1217 (23000): Cannot delete or update a parent row:
a foreign key constraint fails
```

Untuk mengatasinya, hapus dulu foreign key di tabel anak, atau ubah urutan penghapusan.

7 Studi Kasus: Evolusi Tabel `produk`

Mari ikuti perjalanan sebuah tabel dari lahir, berkembang, sampai pensiun. Setiap perintah disertai keluarannya.

7.1 Langkah 1: Membuat Tabel Awal

```
CREATE TABLE produk (
    id    INT AUTO_INCREMENT PRIMARY KEY,
    nama  VARCHAR(100) NOT NULL,
    harga DECIMAL(10,2) NOT NULL
);
```

```
Query OK, 0 rows affected
```

7.2 Langkah 2: Melihat Struktur Awal

```
DESCRIBE produk;
```

```
+-------+---------------+------+-----+---------+----------------+
| Field | Type          | Null | Key | Default | Extra          |
+-------+---------------+------+-----+---------+----------------+
| id    | int           | NO   | PRI | NULL    | auto_increment |
| nama  | varchar(100)  | NO   |     | NULL    |                |
| harga | decimal(10,2) | NO   |     | NULL    |                |
+-------+---------------+------+-----+---------+----------------+
```

7.3 Langkah 3: Menambah Kolom Stok dan Status

```
ALTER TABLE produk
    ADD COLUMN stok INT UNSIGNED NOT NULL DEFAULT 0,
    ADD COLUMN is_active BOOLEAN NOT NULL DEFAULT TRUE;
```

7.4 Langkah 4: Menambah Kolom SKU di Posisi Khusus

```
ALTER TABLE produk
    ADD COLUMN sku VARCHAR(20) NOT NULL AFTER id;
```

7.5 Langkah 5: Memperpanjang Nama Produk

```
ALTER TABLE produk
    MODIFY COLUMN nama VARCHAR(150) NOT NULL;
```

7.6 Langkah 6: Menambah Indeks pada Harga

```
ALTER TABLE produk
    ADD INDEX idx_harga (harga);
```

7.7 Langkah 7: Memeriksa Seluruh Perubahan

```
SHOW CREATE TABLE produk\G
```

Keluaran akan menampilkan `CREATE TABLE` dengan semua kolom dan indeks yang sudah ditambahkan.

7.8 Langkah 8: Mengubah Nama Kolom

```
ALTER TABLE produk
    CHANGE COLUMN is_active aktif BOOLEAN NOT NULL DEFAULT TRUE;
```

7.9 Langkah 9: Menghapus Indeks yang Tidak Terpakai

```
ALTER TABLE produk DROP INDEX idx_harga;
```

7.10 Langkah 10: Mengosongkan Isi Tanpa Menghapus Struktur

```
TRUNCATE TABLE produk;
```

Setelah `TRUNCATE`, `id` yang tadinya `AUTO_INCREMENT` akan mulai lagi dari 1.

7.11 Langkah 11: Menghapus Tabel

```
DROP TABLE produk;
```

7.12 Timeline Studi Kasus

Gambar **??** merangkum sebelas langkah tersebut dalam satu garis waktu.

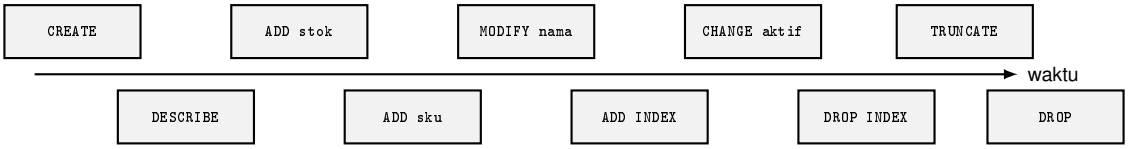


Figure 10. Timeline sebelas langkah evolusi tabel `produk`.

8 Kesalahan Umum dan Cara Menghindarinya

Table 5. Kesalahan umum pada CRUD tabel.

Kesalahan	Akibat	Perbaikan
`DROP TABLE` tanpa backup	kehilangan data permanen	`mysqldump` lebih dulu
`TRUNCATE` dikira sama dengan `DROP`	tabel masih ada, tetapi isi hilang	pahami perbedaan pada Tabel **??**
`ALTER TABLE` satu operasi per perintah	tabel dibangun ulang berkali-kali	gabungkan beberapa operasi sekaligus
`MODIFY` dikira bisa ganti nama	error karena sintaksnya beda	pakai `CHANGE` untuk ganti nama
lupa `FIRST` atau `AFTER`	kolom baru selalu di akhir	tambahkan posisi jika perlu
`ADD INDEX` untuk kolom yang jarang difilter	indeks tidak terpakai, hanya makan ruang	indeks hanya untuk kolom yang sering di `WHERE` atau `ORDER BY`
`DROP COLUMN` tanpa cek foreign key	error karena ada referensi	cek `SHOW CREATE TABLE` tabel anak
mengira `SHOW TABLES` menampilkan isi	hanya nama tabel	pakai `SELECT` untuk isi
`CREATE TABLE` tanpa primary key	sulit melakukan update selektif	selalu sediakan primary key
`AUTO_INCREMENT` untuk ID publik	ID mudah ditebak, rawan scraping	`UUID_SHORT()` atau `BINARY(16)`

9 Lembar Rujukan Cepat

Table 6. Rujukan cepat perintah CRUD pada tingkat tabel.

Operasi	Perintah	Keterangan
Create	`CREATE TABLE nama (...)`	tabel baru
Create	`CREATE TABLE IF NOT EXISTS`	aman untuk skrip
Create	`CREATE TABLE baru LIKE lama`	salin struktur
Create	`CREATE TABLE baru AS SELECT`	salin struktur dan data
Read	`SHOW TABLES`	daftar tabel
Read	`DESCRIBE nama`	ringkasan kolom
Read	`SHOW CREATE TABLE nama`	definisi lengkap
Read	`SHOW INDEX FROM nama`	daftar indeks
Read	`SELECT * FROM information_schema.tables`	query katalog
Update	`ALTER TABLE ADD COLUMN`	tambah kolom
Update	`ALTER TABLE DROP COLUMN`	hapus kolom
Update	`ALTER TABLE MODIFY COLUMN`	ubah definisi
Update	`ALTER TABLE CHANGE COLUMN`	ubah nama dan definisi
Update	`ALTER TABLE RENAME TO`	ganti nama tabel
Update	`ALTER TABLE ADD INDEX`	tambah indeks
Update	`ALTER TABLE DROP INDEX`	hapus indeks
Update	`ALTER TABLE ADD PRIMARY KEY`	tambah primary key
Update	`ALTER TABLE ADD CONSTRAINT ... FOREIGN KEY`	tambah foreign key
Delete	`DROP TABLE [IF EXISTS] nama`	hapus tabel
Delete	`TRUNCATE TABLE nama`	kosongkan isi

10 Kesimpulan

CRUD pada tingkat tabel adalah wilayah dengan permukaan terluas di seluruh basis data relasional. Dari satu perintah `CREATE` hingga dua belas varian `ALTER`, semuanya bermuara pada satu prinsip: struktur tabel adalah kontrak yang bisa berubah, tetapi setiap perubahan punya biaya. Tiga prinsip penutup:

1. Create dengan niat. Setiap kolom punya alasan; setiap tipe punya konsekuensi; setiap `DEFAULT` menghemat satu baris di `INSERT`. Selalu sediakan primary key, dan pilih `IF NOT EXISTS` untuk skrip yang akan dijalankan berulang.
2. Read sebagai kebiasaan. `DESCRIBE` untuk melihat cepat, `SHOW CREATE TABLE` untuk verifikasi menyeluruh, dan `information_schema` untuk otomatisasi. Ketiganya lebih cepat daripada menebak.
3. Update dan Delete dengan sadar. Gabungkan beberapa `ALTER` dalam satu perintah untuk tabel besar. Bedakan `DROP`, `TRUNCATE`, dan `DELETE` sesuai tujuan. Selalu backup sebelum `DROP`.

Dengan fondasi ini, pembaca siap melangkah ke CRUD tingkat baris, di mana `INSERT`, `SELECT`, `UPDATE`, dan `DELETE` akan menjadi fokus.

Referensi

References

[1] MariaDB Foundation. (2026). *SQL Statements — CREATE TABLE, ALTER TABLE, DROP TABLE, TRUNCATE TABLE, SHOW TABLES, DESCRIBE*. MariaDB Knowledge Base. `https://mariadb.com/kb/en/sql-statements/`.

[2] Oracle Corporation. (2026). *MySQL 8.0 Reference Manual — Data Definition Statements*. `https://dev.mysql.com/doc/refman/8.0/en/sql-data-definition-statements.html`.

[3] Oracle Corporation. (2026). *MySQL 8.0 Reference Manual — INFORMATION_SCHEMA Tables*. `https://dev.mysql.com/doc/refman/8.0/en/information-schema.html`.

[4] The PostgreSQL Global Development Group. (2026). *SQL Commands — CREATE TABLE, ALTER TABLE, DROP TABLE, TRUNCATE*. `https://www.postgresql.org/docs/current/sql-commands.html`.

[5] Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). *Database System Concepts* (7th ed.). McGraw-Hill Education.

[6] Date, C. J. (2003). *An Introduction to Database Systems* (8th ed.). Addison-Wesley.

[7] Kleppmann, M. (2017). *Designing Data-Intensive Applications*. O'Reilly Media.

[8] Oracle Corporation. (2026). *MySQL 8.0 Reference Manual — InnoDB Storage Engine*. `https://dev.mysql.com/doc/refman/8.0/en/innodb-storage-engine.html`.

[9] MariaDB Foundation. (2026). *mariadb-dump*. MariaDB Knowledge Base. `https://mariadb.com/kb/en/mariadb-dump/`.

[10] Leach, P., Mealling, M., & Salz, R. (2005). *A Universally Unique IDentifier (UUID) URN Namespace* (RFC 4122). IETF.