Fondasi Basis Data: Menguasai CRUD pada Tingkat Database

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

November 2, 2026

Abstrak

CRUD adalah singkatan dari Create, Read, Update, dan Delete, empat operasi dasar yang membentuk seluruh interaksi dengan sistem penyimpanan data. Namun CRUD tidak berhenti pada baris dan kolom tabel: ia berlapis, dan lapisan paling dasar adalah database itu sendiri. Makalah ini membahas CRUD pada tingkat database secara menyeluruh, mulai dari membangun wadah penyimpanan baru dengan `CREATE DATABASE`, menelusuri katalog dengan `SHOW DATABASES` dan `USE`, mengubah karakteristik wadah dengan `ALTER DATABASE`, hingga menghapusnya dengan `DROP DATABASE` beserta segala konsekuensinya. Setiap perintah dilengkapi diagram alur, simulasi keluaran, dan skenario kegagalan yang lazim terjadi. Makalah ini menjadi fondasi sebelum pembaca melangkah ke CRUD pada tingkat tabel dan baris.

1 Pendahuluan: Mengapa Database Punya Lapisan

1.1 Analogi Sistem Berkas

Sebuah komputer menata berkasnya dalam hierarki. Di puncak ada sistem berkas, di bawahnya ada folder, dan di dalam folder ada berkas. Basis data meniru struktur itu.

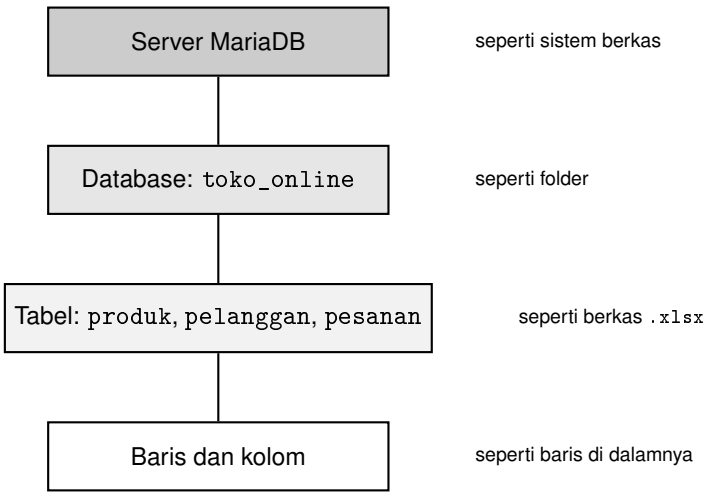


Figure 1. Hierarki server, database, tabel, dan baris sebagai analogi sistem berkas.

Pada makalah sebelumnya dibahas tipe data di tingkat kolom. Makalah ini naik satu lapis ke atas dan membahas wadah di mana seluruh kolom itu tinggal: database. Tanpa database yang dibuat dengan benar, tabel tidak punya tempat untuk hidup. Tanpa pemahaman cara menghapusnya dengan aman, satu perintah `DROP` bisa menghilangkan data setahun.

1.2 Empat Operasi, Satu Siklus Hidup

CRUD pada tingkat database membentuk satu siklus hidup yang utuh, seperti pada Gambar **??**.

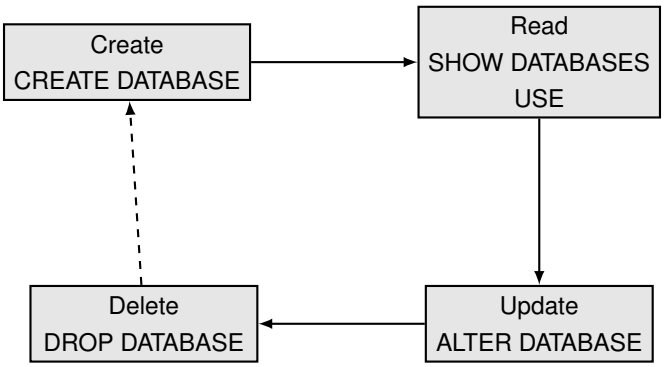


Figure 2. Siklus hidup CRUD pada tingkat database.

Garis putus-putus dari `DROP` kembali ke `CREATE` menandakan bahwa setelah database dihapus, biasanya akan dibuat database baru sebagai pengganti, misalnya saat migrasi atau pengujian.

2 Menyiapkan Konteks: Masuk ke Server

Sebelum semua perintah CRUD database bisa dijalankan, kita harus berada di dalam server MariaDB. Di terminal Linux, masuk melalui klien `mariadb` atau `mycli`.

```
mariadb -u root -p
```

Setelah memasukkan kata sandi, prompt berubah menjadi seperti berikut.

```
MariaDB [(none)]>
```

Kurung siku berisi `none` menandakan belum ada database yang dipilih. Prompt inilah tempat seluruh perintah pada makalah ini diketik.

3 Create: Membangun Wadah Baru

3.1 Sintaks Dasar

Perintah `CREATE DATABASE` memiliki dua bentuk yang setara. `SCHEMA` adalah sinonim dari `DATABASE` di MariaDB dan MySQL, warisan dari standar SQL.

```
CREATE DATABASE nama_database;
CREATE SCHEMA nama_database;
```

Kedua baris menghasilkan efek yang persis sama. Konvensi yang lazim dipakai adalah `DATABASE`, sedangkan `SCHEMA` muncul lebih sering di lingkungan PostgreSQL.

3.2 Varian Lengkap

Selain bentuk dasar, `CREATE DATABASE` menerima beberapa klausa opsional yang penting untuk produksi.

```
CREATE DATABASE [IF NOT EXISTS] nama_database
    [CHARACTER SET nama_charset]
    [COLLATE nama_collation]
    [COMMENT 'teks keterangan'];
```

Setiap klausa punya peran yang berbeda, seperti dijelaskan pada Tabel **??**.

Table 1. Klausa opsional pada `CREATE DATABASE`.

Klausa	Fungsi
`IF NOT EXISTS`	Mencegah error jika database dengan nama sama sudah ada.
`CHARACTER SET`	Menentukan pengodean karakter bawaan untuk seluruh tabel di dalamnya.
`COLLATE`	Menentukan aturan pembandingan dan pengurutan teks.
`COMMENT`	Menyimpan catatan dokumentasi pada katalog.

3.3 Contoh Nyata dan Keluarannya

```
CREATE DATABASE toko_online
    CHARACTER SET utf8mb4
    COLLATE utf8mb4_unicode_ci
    COMMENT 'Database produksi toko online';
```

Jika berhasil, MariaDB menampilkan konfirmasi singkat.

```
Query OK, 1 row affected
```

Jika diulang tanpa `IF NOT EXISTS`, MariaDB menolak.

```
ERROR 1007 (HY000): Can't create database 'toko_online';
database exists
```

Dengan `IF NOT EXISTS`, pengulangan tidak menghasilkan error, hanya peringatan halus.

```
CREATE DATABASE IF NOT EXISTS toko_online;
```

```
Query OK, 1 row affected, 1 warning
```

3.4 Alur Keputusan Sebelum Membuat Database

Membuat database bukan sekadar mengetik perintah. Ada beberapa pertanyaan yang sebaiknya dijawab lebih dulu, seperti pada Gambar **??**.

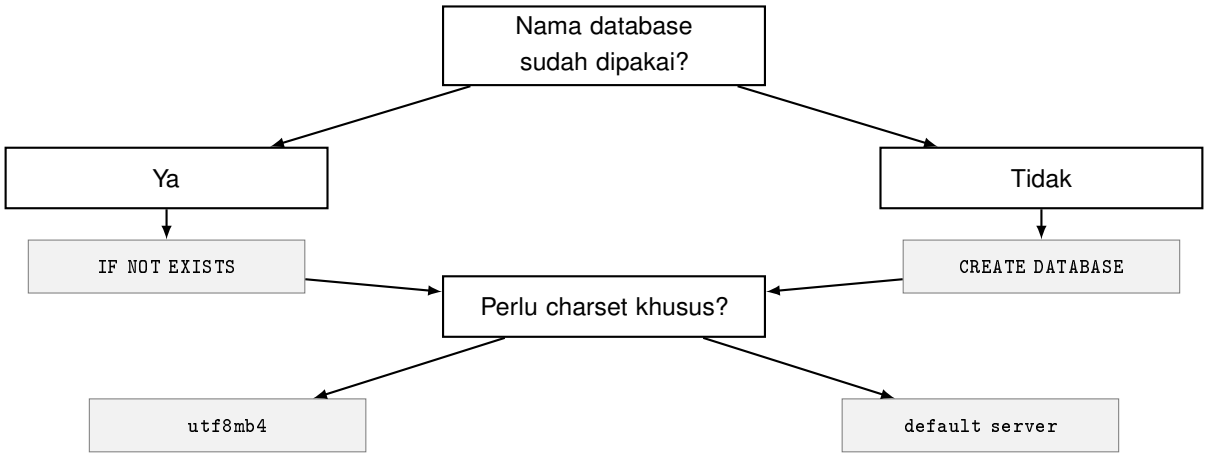


Figure 3. Alur keputusan sebelum menjalankan `CREATE DATABASE`.

3.5 Mengapa `utf8mb4` dan Bukan `utf8`

MariaDB memiliki dua charset yang namanya mirip tetapi berperilaku berbeda. `utf8` di MariaDB hanyalah alias untuk `utf8mb3`, yang hanya mendukung karakter hingga tiga byte. Emoji dan beberapa aksara Asia membutuhkan empat byte. Jika Anda membuat database dengan `utf8` lalu mencoba menyimpan emoji, karakter tersebut akan terpotong atau ditolak.

`utf8mb4` mendukung seluruh rentang Unicode, termasuk emoji. Untuk database baru pada tahun 2020 ke atas, `utf8mb4` adalah pilihan bawaan yang tepat. Collation yang umum dipakai adalah `utf8mb4_unicode_ci`, dengan `ci` berarti *case-insensitive*, sehingga pembandingan tidak membedakan huruf besar dan kecil.

4 Read: Menelusuri Katalog Database

Operasi Read pada tingkat database terdiri dari dua perintah yang saling melengkapi: melihat daftar seluruh database, dan memilih salah satu untuk dijadikan konteks kerja.

4.1 Melihat Seluruh Database

```
SHOW DATABASES;
```

Keluaran perintah ini berupa tabel satu kolom.

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| toko_online        |
+--------------------+
5 rows in set (0.001 sec)
```

Empat baris pertama adalah database sistem yang dibuat otomatis oleh MariaDB saat instalasi. Penjelasannya ada pada Tabel **??**.

Table 2. Database sistem bawaan MariaDB.

Database	Isi
`information_schema`	Katalog metadata tentang seluruh database, tabel, dan kolom.
`mysql`	Tabel pengguna, hak akses, dan konfigurasi server.
`performance_schema`	Statistik kinerja dan profil eksekusi query.
`sys`	Tampilan ringkas dari `performance_schema`.

`information_schema` adalah database khusus yang tidak menyimpan data pengguna, melainkan metadata. Isinya dihasilkan langsung oleh server setiap kali di-query, sehingga tidak bisa di-`DROP` atau di-`UPDATE` secara langsung.

4.2 Varian Filter

`SHOW DATABASES` juga menerima klausa `LIKE` untuk menyaring hasil berdasarkan pola.

```
SHOW DATABASES LIKE 'toko%';
SHOW DATABASES WHERE \`Database\` = 'toko_online';
```

Keluaran pertama hanya menampilkan database yang namanya diawali `toko`. Tanda persen berperan sebagai wildcard, sama seperti pada `LIKE` di SQL umumnya.

4.3 Memilih Database Aktif

Setelah tahu nama database, langkah berikutnya adalah masuk ke dalamnya dengan `USE`.

```
USE toko_online;
```

```
Database changed
```

Prompt langsung berubah menandakan konteks kerja sudah berpindah.

```
MariaDB [toko_online]>
```

Setelah `USE`, semua perintah tabel seperti `CREATE TABLE`, `SELECT`, dan `DROP TABLE` akan berlaku pada database `toko_online` tanpa perlu menyebut nama database setiap kali.

Jika nama database salah, MariaDB menolak.

```
ERROR 1049 (42000): Unknown database 'toko_onlinee'
```

Untuk memeriksa database mana yang sedang aktif tanpa melihat prompt, gunakan fungsi `DATABASE()`.

```
SELECT DATABASE();
```

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

4.4 Diagram Perpindahan Konteks

Gambar **??** mengilustrasikan bagaimana `USE` memindahkan konteks kerja tanpa mengubah data apa pun.



Figure 4. Perpindahan konteks kerja dengan `USE`.

Perpindahan konteks bersifat sementara. Jika sesi ditutup lalu dibuka kembali, konteks kembali kosong dan `USE` harus dijalankan ulang.

5 Update: Mengubah Karakteristik Database

Pada tingkat database, operasi Update tidak mengubah data baris (itu tugas `UPDATE` di tingkat tabel), melainkan mengubah karakteristik wadah: charset, collation, dan pada beberapa mesin basis data, mode baca-saja.

5.1 Sintaks Dasar

```
ALTER DATABASE nama_database
    [CHARACTER SET nama_charset]
    [COLLATE nama_collation];
```

`ALTER SCHEMA` adalah sinonim yang setara.

5.2 Mengubah Charset dan Collation

Misalnya sebuah database lama dibuat dengan charset bawaan yang keliru. Charset-nya bisa diperbaiki.

```
ALTER DATABASE toko_online
    CHARACTER SET utf8mb4
    COLLATE utf8mb4_unicode_ci;
```

```
Query OK, 1 row affected
```

Penting dipahami: `ALTER DATABASE` hanya mengubah pengaturan bawaan. Tabel yang sudah ada sebelumnya tidak ikut berubah charset-nya. Untuk mengubah tabel dan kolom yang sudah ada, perlu `ALTER TABLE` terpisah per tabel, atau konversi bertahap.

5.3 Apa yang Tidak Bisa Dilakukan

Banyak pemula mencoba `RENAME DATABASE` dan terkejut karena perintah itu tidak ada.

```
RENAME DATABASE toko_online TO toko_baru;
```

```
ERROR 1064 (42000): You have an error in your SQL syntax;
check the manual that corresponds to your MariaDB server
version for the right syntax to use near
```

```
'DATABASE toko_online TO toko_baru' at line 1
```

MariaDB dan MySQL memang tidak menyediakan perintah `RENAME DATABASE`. Ada dua alasan historis. Pertama, operasi ini berisiko jika ada tabel yang sedang dibuka oleh sesi lain. Kedua, dalam sistem terdistribusi, mengganti nama database bisa mematahkan replikasi.

Sebagai gantinya, ada tiga jalur resmi seperti pada Tabel **??**.

Table 3. Alternatif saat ingin mengganti nama database.

Pendekatan	Langkah singkat
Dump dan restore	`mysqldump` database lama, buat database baru, restore ke sana, lalu `DROP` yang lama.
Pindahkan tabel	Buat database baru, gunakan `RENAME TABLE lama.t TO baru.t` untuk setiap tabel.
Simpan tautan	Buat `VIEW` atau nama alias di aplikasi tanpa mengubah struktur.

5.4 Alur Update Database

Gambar **??** merangkum kapan `ALTER DATABASE` berguna dan kapan harus memakai pendekatan lain.

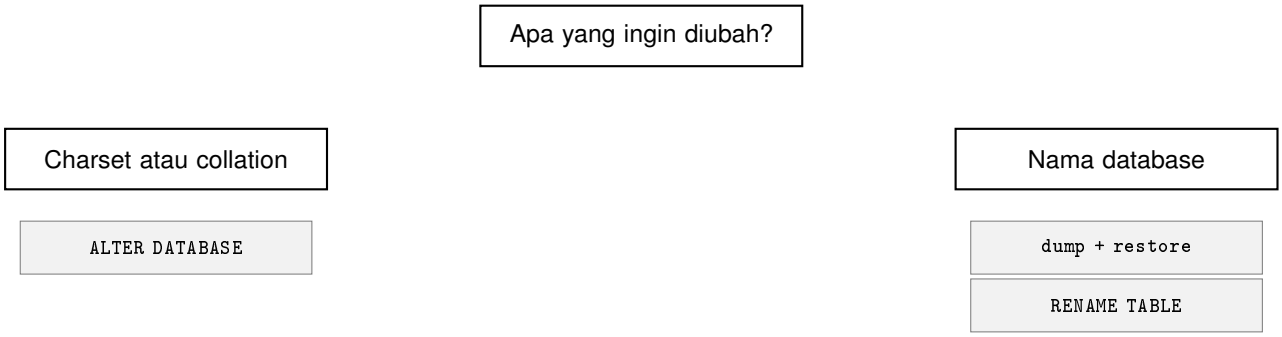


Figure 5. Alur memilih operasi update pada tingkat database.

5.5 Memeriksa Hasil Perubahan

Setelah `ALTER DATABASE`, hasilnya bisa diperiksa dengan `SHOW CREATE DATABASE`.

```
SHOW CREATE DATABASE toko_online;
```

```
+-------------+-----------------------------------------------------+
| Database    | Create Database                                     |
+-------------+-----------------------------------------------------+
| toko_online | CREATE DATABASE `toko_online`
               /*!40100 DEFAULT CHARACTER SET utf8mb4
               COLLATE utf8mb4_unicode_ci */
```

```
               /*!80016 COMMENT 'Database produksi toko online' */ |
+------------+-----------------------------------------------------+
```

Keluaran ini menampilkan perintah `CREATE DATABASE` yang persis akan dijalankan jika database dibuat ulang dari nol. Ini cara paling andal untuk memverifikasi charset, collation, dan komentar yang sedang aktif.

6 Delete: Menghapus Database

6.1 Sintaks Dasar

```
DROP DATABASE [IF EXISTS] nama_database;
```

Perintah ini menghapus seluruh database beserta semua tabel, indeks, view, trigger, dan data di dalamnya. Tidak ada tombol undo. Tidak ada tempat sampah. Tidak ada konfirmasi.

```
DROP DATABASE toko_online;
```

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

Keluaran `0 rows affected` bisa menyesatkan. Angka nol itu merujuk pada baris data, bukan pada ada atau tidaknya database. Database tetap terhapus, terlepas dari berapa banyak tabel di dalamnya.

6.2 `IF EXISTS` dan Perilakunya

Tanpa `IF EXISTS`, menghapus database yang tidak ada akan menghasilkan error.

```
ERROR 1008 (HY000): Can't drop database 'toko_online';
database doesn't exist
```

Dengan `IF EXISTS`, penghapusan yang gagal tidak dianggap error.

```
DROP DATABASE IF EXISTS toko_online;
```

```
Query OK, 0 rows affected, 1 warning
```

`IF EXISTS` sering dipakai di skrip migrasi atau pengujian yang dijalankan berulang, karena hasilnya selalu sukses tanpa perlu mengecek keberadaan database lebih dulu.

6.3 Konsekuensi Berantai

Gambar **??** menunjukkan bagaimana satu perintah `DROP DATABASE` menghapus semua lapisan di bawahnya secara berantai.

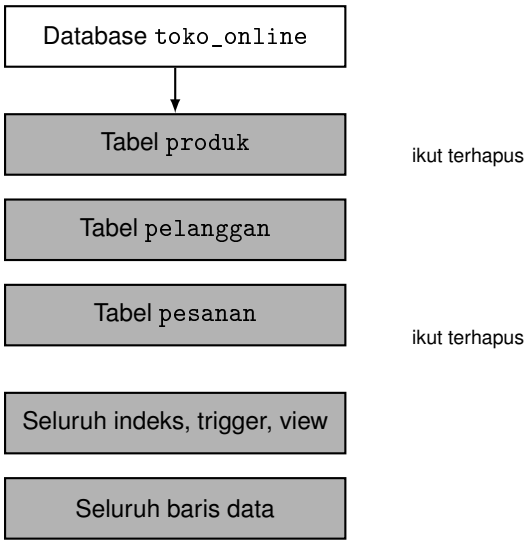


Figure 6. Konsekuensi berantai satu perintah `DROP DATABASE`.

6.4 Prosedur Aman Sebelum `DROP`

Penghapusan database yang tidak disengaja adalah salah satu insiden paling sering dilaporkan di forum basis data. Gambar **??** menyajikan prosedur minimal yang sebaiknya dilalui.

1. Pastikan sesi tidak sedang memakai database: `SELECT DATABASE();`

↓

2. Buat cadangan: `mysqldump -u root -p toko_online > backup.sql`

↓

3. Verifikasi berkas cadangan: `ls -lh backup.sql`

↓

4. Uji restore di server sementara jika waktu memungkinkan

↓

5. Baru jalankan `DROP DATABASE toko_online;`

Figure 7. Prosedur aman sebelum menghapus database.

Perintah cadangan pada langkah kedua adalah penyelamat utama. Berkas `backup.sql` bisa dipulihkan dengan satu perintah.

```
mariadb -u root -p toko_online < backup.sql
```

Selama berkas cadangan ada, penghapusan database tidak lagi bersifat final.

6.5 Menghapus Banyak Database Sekaligus

MariaDB tidak memiliki sintaks `DROP DATABASE db1, db2, db3`. Untuk menghapus beberapa database, perintah harus dijalankan berulang. Pada skrip, ini biasanya dilakukan dalam satu sesi.

```
DROP DATABASE IF EXISTS db_lama;
DROP DATABASE IF EXISTS db_uji;
DROP DATABASE IF EXISTS db_arsip;
```

7 Studi Kasus: Siklus Hidup Satu Database

Mari ikuti satu database dari lahir sampai mati, dengan setiap perintah dan keluarannya.

```
-- Langkah 1: masuk ke server
-- (dari terminal: mariadb -u root -p)

-- Langkah 2: lihat kondisi awal
SHOW DATABASES;

-- Langkah 3: buat database baru
CREATE DATABASE perpustakaan
    CHARACTER SET utf8mb4
    COLLATE utf8mb4_unicode_ci
    COMMENT 'Database perpustakaan kampus';

-- Langkah 4: masuk ke database
USE perpustakaan;

-- Langkah 5: verifikasi konteks
SELECT DATABASE();

-- Langkah 6: periksa struktur yang dibuat
SHOW CREATE DATABASE perpustakaan;

-- Langkah 7: ubah collation (contoh update)
ALTER DATABASE perpustakaan
    COLLATE utf8mb4_general_ci;

-- Langkah 8: hapus database
DROP DATABASE perpustakaan;

-- Langkah 9: pastikan sudah hilang
SHOW DATABASES;
```

Urutan ini menggambarkan satu siklus CRUD lengkap: `SHOW` untuk Read, `CREATE` untuk Create,

`ALTER` untuk Update, dan `DROP` untuk Delete.

8 Kesalahan Umum dan Cara Menghindarinya

Table 4. Kesalahan umum pada CRUD database.

Kesalahan	Akibat	Perbaikan
`DROP DATABASE` tanpa backup	kehilangan seluruh data secara permanen	jalankan `mysqldump` lebih dulu
`USE` tanpa memeriksa nama	konteks salah, tabel dibuat di database keliru	`SELECT DATABASE()` sesudahnya
`CREATE` tanpa `IF NOT EXISTS`	skrip gagal saat dijalankan ulang	selalu pakai `IF NOT EXISTS`
`utf8` alih-alih `utf8mb4`	emoji dan aksara CJK rusak	pakai `utf8mb4`
mengira `ALTER DATABASE` mengubah tabel	tabel lama tetap dengan charset lama	jalankan `ALTER TABLE` per tabel
mengharapkan `RENAME DATABASE`	error sintaks	pakai dump dan restore
`DROP` di dalam sesi yang sedang `USE`	tidak bisa di-`DROP` dari sesi lain	`USE` ke database lain dulu
mengira `0 rows affected` berarti tidak ada yang terhapus	salah tafsir	tetap periksa dengan `SHOW DATABASES`

Perhatikan poin terakhir. Pada beberapa forum, ada yang bingung karena `DROP DATABASE` mengembalikan `0 rows affected`. Angka nol itu adalah sisa dari protokol yang mengukur baris data, bukan objek skema. Database tetap terhapus.

9 Lembar Rujukan Cepat

Table 5. Rujukan cepat perintah CRUD pada tingkat database.

Operasi	Perintah	Keterangan
Create	`CREATE DATABASE nama`	membuat database baru
Create	`CREATE DATABASE IF NOT EXISTS nama`	aman untuk skrip berulang
Read	`SHOW DATABASES`	menampilkan seluruh database
Read	`SHOW DATABASES LIKE 'pola%'`	filter berdasarkan pola
Read	`USE nama`	memilih database aktif
Read	`SELECT DATABASE()`	memeriksa konteks aktif
Read	`SHOW CREATE DATABASE nama`	melihat definisi lengkap
Update	`ALTER DATABASE nama CHARACTER SET cs`	ubah charset
Update	`ALTER DATABASE nama COLLATE col`	ubah collation
Delete	`DROP DATABASE nama`	hapus database
Delete	`DROP DATABASE IF EXISTS nama`	hapus jika ada, tanpa error

10 Kesimpulan

CRUD pada tingkat database adalah fondasi yang sering terlewat. Padahal seluruh pekerjaan tingkat tabel bergantung pada wadah yang dibuat dengan benar. Tiga prinsip penutup:

1. Create dengan niat. Selalu tentukan charset `utf8mb4` dan collation yang konsisten, serta gunakan `IF NOT EXISTS` agar skrip tidak rapuh.
2. Read sebagai kebiasaan. Sebelum menyentuh apa pun, biasakan menjalankan `SHOW DATABASES` dan `SELECT DATABASE()`. Dua perintah ini mencegah hampir semua kesalahan konteks.
3. Delete dengan cadangan. Tidak ada undo pada `DROP DATABASE`. Satu `mysqldump` sebelum `DROP` adalah asuransi termurah yang bisa dipasang.

Dengan fondasi ini, pembaca siap melangkah ke CRUD pada tingkat tabel, di mana tabel, kolom, dan indeks akan menjadi fokus.

Referensi

References

[1] MariaDB Foundation. (2026). *SQL Statements — CREATE DATABASE, ALTER DATABASE, DROP DATABASE, SHOW DATABASES, USE*. 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] The PostgreSQL Global Development Group. (2026). *SQL Commands — CREATE DATABASE, ALTER DATABASE, DROP DATABASE*. https://www.postgresql.org/docs/current/sql-commands.html.

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

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

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

[7] The Unicode Consortium. (2024). *The Unicode Standard, Version 16.0*. https://www.unicode.org/versions/Unicode16.0.0/.

[8] Oracle Corporation. (2024). *The utf8mb3 Character Set and utf8mb4*. MySQL Reference Manual. https://dev.mysql.com/doc/refman/8.0/en/charset-unicode-utf8mb3.html.

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