DATABASE SECURITY

1. Least Privilege & Role

Prinsip **least privilege** menyatakan bahwa setiap akun hanya diberi hak minimum yang diperlukan untuk menyelesaikan tugasnya. Pada basis data, penerapan prinsip ini mengurangi risiko **akses berlebih**, **modifikasi tidak sah**, dan **insiden kebocoran**. Dengan **role-based access control (RBAC)**, hak akses dikelola melalui *role* (peran), lalu *role* tersebut diberikan ke akun pengguna—membuat administrasi lebih konsisten dan mudah diaudit.

Desain Peran (RBAC)

a. r_datareader: hanya SELECT pada seluruh objek di skema perpustakaan.
b. r_datawriter: SELECT, INSERT, UPDATE, DELETE pada skema perpustakaan.

Pembeda utama: *reader* tidak boleh menulis; *writer* boleh menulis namun tidak memiliki hak administratif.

Implementasi

1) Membuat Role dan Mengalokasikan Privilege

```
CREATE ROLE IF NOT EXISTS r_datareader;
CREATE ROLE IF NOT EXISTS r_datawriter;

GRANT SELECT ON perpustakaan.* TO r_datareader;
GRANT SELECT, INSERT, UPDATE, DELETE ON perpustakaan.* TO r_datawriter;
```

2) Membuat Akun dan Mengaitkan Role

```
CREATE USER IF NOT EXISTS 'reader'@'localhost' IDENTIFIED BY 'Pwd!Reader_2025';
CREATE USER IF NOT EXISTS 'writer'@'localhost' IDENTIFIED BY 'Pwd!Writer_2025';

GRANT r_datareader TO 'reader'@'localhost';
GRANT r_datawriter TO 'writer'@'localhost';

-- Agar role aktif otomatis saat login
SET DEFAULT ROLE r_datareader TO 'reader'@'localhost';
SET DEFAULT ROLE r_datawriter TO 'writer'@'localhost';
```

2. Password Policy & Account Lock

Dalam operasi basis data, banyak insiden kebocoran dimulai dari **pencurian kredensial**: password lemah, reuse password, atau brute-force login. Praktek ini untuk memahami dan menerapkan **pertahanan lapis awal** di MySQL—yakni **kebijakan password yang kuat** dan **mekanisme pemblokiran sementara** saat terjadi percobaan login gagal beruntun.

Fitur **FAILED_LOGIN_ATTEMPTS** dan **PASSWORD_LOCK_TIME** (MySQL 8.0.19+) mensimulasikan skenario ketika user memasukkan password salah beberapa kali sehingga akun terkunci sementara. Berikut perintah untuk implementasi password policy dan account lock

```
INSTALL COMPONENT 'file://component_validate_password';
SET PERSIST validate_password.length=12;
SET PERSIST validate_password.mixed_case_count=1;
SET PERSIST validate_password.number_count=1;
SET PERSIST validate_password.special_char_count=1;

ALTER USER 'ops_user'@'localhost'  -- sesuaikan dengan nama user yang Anda buat
  FAILED_LOGIN_ATTEMPTS 5
  PASSWORD_LOCK_TIME 2
  PASSWORD EXPIRE INTERVAL 180 DAY
  PASSWORD HISTORY 5;
```

Penjelasan:

A) **Aktivasi validator password**
   INSTALL COMPONENT 'file://component_validate_password';
   Fungsi: memasang komponen bawaan MySQL yang memeriksa kekuatan password saat CREATE/ALTER USER

B) **Menetapkan aturan kompleksitas (server-enforced)**
   SET PERSIST validate_password.length = 12; → minimal password 12 karakter.
   SET PERSIST validate_password.mixed_case_count = 1; → **wajib** mengandung **huruf besar dan huruf kecil** (masing-masing minimal 1).
   SET PERSIST validate_password.number_count = 1; → **wajib** ada **minimal 1 digit**.
   SET PERSIST validate_password.special_char_count = 1; → **wajib** ada **minimal 1 karakter spesial** (!@#$%...).

C) **Menerapkan kebijakan ke akun**
   FAILED_LOGIN_ATTEMPTS 5 → Jika 5 kali berturut-turut salah password, akun diblokir sementara.
   PASSWORD_LOCK_TIME 2 → Lama blokir sementara = 2 hari. (Bisa pecahan: 0.01 ≈ ±15 menit.)
   PASSWORD EXPIRE INTERVAL 180 DAY → Password kadaluwarsa tiap 180 hari → user dipaksa ganti saat login.
   PASSWORD HISTORY 5 → Dilarang memakai 5 password terakhir (mencegah reuse).

Catatan: Praktek ini **bukan** lock permanen. Status blokir sementara **tidak muncul** sebagai account_locked='Y'. Klien akan menampilkan pesan seperti *"Account is blocked for X day(s)…"*.

D) Verifikasi

```
SELECT user, host, account_locked
FROM mysql.user
WHERE user IN ('reader','writer','ops_user');
```

user	host	account_locked
reader	%	Y
writer	%	Y
ops_user	localhost	N

Fungsi: cek apakah akun **pernah di-LOCK permanen** (Y/N).
Ingat: blokir sementara **tetap 'N'** di kolom ini.

E) Membuka blokir & pemeliharaan

ALTER USER 'writer'@'localhost' ACCOUNT UNLOCK; → **hapus blokir sementara** (unlock manual oleh admin).
ALTER USER 'writer'@'localhost' PASSWORD_LOCK_TIME 0; → **matikan/akhiri** durasi blokir sementara (praktis untuk lab/demo).
ALTER USER 'writer'@'localhost' IDENTIFIED BY 'WriTer!2025_99'; → ganti password user agar **sesuai kebijakan**.
Jika muncul ERROR 1819: password baru **tidak memenuhi** aturan validate_password.*.

3. Data Masking via View (Privasi)

Dalam pengelolaan basis data, sering diperlukan pemberian akses baca bagi pihak tertentu (misalnya asisten, QA, atau tim analitik) tanpa menampilkan **Personally Identifiable Information (PII)** atau **data identitas pribadi** secara penuh. Memberi akses langsung ke tabel sumber berpotensi menimbulkan kebocoran data.
Data masking melalui VIEW merupakan pendekatan yang menampilkan data versi *yang disamarkan* pada saat query dijalankan, tanpa mengubah data asli. Dengan demikian, pengguna tetap memperoleh informasi yang dibutuhkan, tetapi detail sensitif terlindungi

A. Skenario Kasus

Skema perpustakaan memiliki tabel anggota berisi kolom nama, email, dan no_hp. Admin ingin memberi akses baca untuk akun reader agar bisa melakukan analisis agregat, namun **tanpa** menampakkan identitas lengkap anggota.

B. Konsep Utama

1) **Data Masking**: proses menyamarkan sebagian nilai kolom sensitif di hasil query (mis. “Budi Santoso” → “B***”).
2) **VIEW:** objek logika yang menyajikan hasil SELECT tertentu. Dengan **privilege** yang tepat, VIEW dapat menjadi **lapisan privasi** di atas tabel asli.
3) **Least Privilege:** pengguna hanya mendapat hak yang benar-benar diperlukan. Dalam konteks ini, reader hanya diizinkan membaca VIEW termasking.

C. Langkah Praktek

1) Membuat view masking

```
CREATE OR REPLACE VIEW perpustakaan.v_anggota_masked AS
        SELECT
          -- tampilkan 1 huruf pertama nama, sisanya disamarkan
          CONCAT(SUBSTR(nama, 1, 1), '***') AS nama,

          -- tampilkan sebagian awal alamat lalu disamarkan
          CONCAT(SUBSTR(alamat, 1, 2), '***') AS alamat,

          -- tanggal pendaftaran tetap ditampilkan (umum, bukan PII langsung)
          DATE(tanggal_daftar) AS tanggal_daftar FROM perpustakaan.anggota;
```

2) Pengaturan Privilege

```
REVOKE SELECT ON `perpustakaan`.* FROM 'reader';
GRANT SELECT ON perpustakaan.v_anggota_masked TO 'reader';
```

3) Verifikasi

Masuk sebagai reader dan jalankan:

```
SELECT * FROM perpustakaan.v_anggota_masked LIMIT 5;  -- harus tampil data termasking

SELECT * FROM perpustakaan.anggota LIMIT 5;  -- harus ditolak
```

Output

nama	alamat	tanggal_daftar
A***	Su***	2024-07-01
B***	Si***	2024-07-02
C***	Gr***	2024-07-03
D***	Ma***	2024-07-04
E***	Ke***	2024-07-05

Bagaimana jika ingin menyamarkan tanggal_daftar?

4. **Gatekeeper via Stored Procedure**

Dalam banyak sistem, aplikasi atau pengguna dapat mengeksekusi **DML langsung** ke tabel (INSERT/UPDATE/DELETE). Pola ini berisiko karena:

a. Validasi bisnis (mis. range tahun terbit, status anggota) dapat terlewat.
b. Integritas data mudah terganggu (kolom wajib kosong/asal, nilai tidak konsisten).
c. Jejak akuntabilitas menjadi lemah (sulit menelusuri siapa mengubah apa dan lewat aturan apa).

Stored procedure diposisikan sebagai **gatekeeper** (penjaga gerbang) agar semua perubahan data **wajib melalui satu titik kendali** yang memuat aturan bisnis, validasi, dan pengamanan.

Konsep

a. **Enkapsulasi logika di server**: validasi tidak hanya di aplikasi klien, tetapi **dipaksa** di sisi DBMS.
b. **Separation of duties**: pengguna "writer" tidak lagi punya hak INSERT/UPDATE/DELETE langsung, melainkan **hanya EXECUTE** pada prosedur tertentu.
c. **SQL SECURITY DEFINER** (opsional): prosedur bisa berjalan dengan hak pembuat prosedur, namun akses ke objek internal tetap dikontrol.

Implementasi

```
DELIMITER //
CREATE PROCEDURE perpustakaan.sp_tambah_buku(
          IN p_judul VARCHAR(255),
          IN p_penulis VARCHAR(255),
          IN p_tahun INT
         )
SQL SECURITY DEFINER
        BEGIN
          -- Validasi aturan bisnis: tahun terbit wajar
          IF p_tahun < 1800 OR p_tahun > YEAR(CURDATE()) THEN
```

```
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Tahun terinvalid';
  END IF;

  -- (Opsional) Validasi lain: judul/penulis tidak kosong
  IF p_judul IS NULL OR p_judul = '' OR p_penulis IS NULL OR p_penulis = '' THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Judul/Penulis wajib diisi';
  END IF;

  -- Tulis data bila valid
  INSERT INTO perpustakaan.buku(judul_buku, penulis, tahun_terbit)
  VALUES (p_judul, p_penulis, p_tahun);
END//
DELIMITER ;

-- Cabut DML langsung dari user writer
REVOKE INSERT, UPDATE, DELETE ON perpustakaan.buku FROM 'writer';

-- Beri akses hanya eksekusi prosedur
GRANT EXECUTE ON PROCEDURE perpustakaan.sp_tambah_buku TO 'writer';
```

Penjelasan:

1) SQL SECURITY DEFINER: prosedur dieksekusi dengan hak pemilik (definer). Gunakan dengan hati-hati; definer harus aman, dan haknya tidak berlebihan.
2) SIGNAL SQLSTATE '45000': mengeluarkan error kustom saat validasi gagal—mendorong aplikasi memperbaiki input.
3) REVOKE … ON … FROM writer: menutup “jalan pintas” DML langsung.
4) GRANT EXECUTE … TO writer: membuka “jalan resmi” melalui prosedur.

Alur Akses (Sebelum vs Sesudah)

Sebelum:

Aplikasi → INSERT langsung ke buku → **tidak ada** validasi terpusat → rentan salah data.

Sesudah:

Aplikasi → CALL sp_tambah_buku(...) → **divalidasi** (range tahun, field wajib, dsb) → baru INSERT → **konsisten & terkontrol**.

Verifikasi (Login sebagai Writer)

Negatif test:

CALL sp_tambah_buku('Judul', 'Penulis', 2050); → gagal dengan pesan “Tahun terinvalid”.

Positif test:

CALL sp_tambah_buku('Dasar Basis Data', 'A. Siregar', 2021); → sukses, baris baru muncul di buku.

Hak akses:
INSERT langsung oleh writer ke buku → ditolak (karena sudah di-REVOKE).