🏥 Hospital Management System (HMS) — Complete Crash Course

This crash course provides a full understanding of how the **Hospital Management System (HMS)** database works — including its **structure**, **encryption/hashing system**, **triggers**, **views**, and **data workflows**.
It is designed for developers, database administrators, and students who need to maintain, extend, or audit the system securely.

📘 1. System Overview

The HMS database is a secure **MySQL 8.0+** system designed to manage hospital data while **protecting sensitive information** through a combination of **AES-256 encryption** and **SHA-256 hashing**.

🔐 Core Security Principles

- **AES-256 Encryption** (reversible): Protects patient, doctor, and billing data that must remain confidential but retrievable.
- **SHA-256 Hashing** (irreversible): Protects authentication data and identifiers (passwords, email hashes, etc.).
- **Referential Integrity**: Enforced through multiple **foreign key constraints** to maintain data consistency.
- **Triggers**: Automatically handle encryption, hashing, and validation logic during data insertion.
- **Views**: Provide decrypted, readable representations of data for authorized users.

🧱 2. Database Architecture

🗂️ Primary Tables

Table	Description
patient	Stores personal details of patients (fully encrypted).
doctor	Contains doctors' professional information (contact encrypted).
staff	Includes non-doctor hospital staff (receptionists, cashiers, etc.).
bill	Manages financial charges linked to patients.
payment	Tracks payments made towards bills.
appointment	Records scheduled appointments between patients and doctors.
user	Handles authentication data (usernames encrypted, passwords hashed).
secret_config	Holds the AES encryption key. (Critical for decrypting all sensitive data.)

🧩 3. Table Relationships (Foreign Keys)

HMS uses strict foreign key dependencies to preserve consistency:

Relationship	Description
`bill.patient_id → patient.patient_id`	A bill must belong to an existing patient.
`payment.bill_id → bill.bill_id`	A payment must be tied to a valid bill.
`payment.received_by → staff.staff_id`	Each payment is received by a valid staff member.
`appointment.patient_id → patient.patient_id`	Each appointment must correspond to an existing patient.
`appointment.doctor_id → doctor.doctor_id`	Each appointment must have a valid doctor.

Foreign keys ensure **no orphaned records** (e.g., a bill cannot exist without a patient).

🔑 4. Encryption and Hashing System

🔒 AES-256 Encryption (Reversible)

- Used for personally identifiable information (PII) and financial data.
- Implemented with:

- ```
  AES_ENCRYPT(<plaintext>, get_enc_key())
  AES_DECRYPT(<ciphertext>, get_enc_key())
  ```

- The function `get_enc_key()` retrieves a 32-byte key from `secret_config`, giving full **AES-256** strength.

SHA-256 Hashing (Irreversible)

- Used for **passwords**, **email_hash**, and **phone_hash**.
- Generated with:

  ```
  SHA2(<value>, 256)
  ```

- Provides 64-character hexadecimal hashes, ensuring no plaintext recovery.

Key Storage

The encryption key is stored in:

```
CREATE TABLE secret_config (
  id TINYINT PRIMARY KEY,
  enc_key VARBINARY(32)
);
```

It is initialized as:

```
INSERT INTO secret_config VALUES (1,
LEFT(SHA2('xHMEeykkS!Y$dj1T7H6UeL*@qTBKFkS$',256),32));
```

> ⚠ This key must be kept secure — losing it makes decryption impossible.

---

5. Trigger System (Automatic Encryption & Hashing)

Triggers ensure that all inserts are **automatically protected**, so developers never manually encrypt or hash data.
```

Trigger	Table	Operation	Function
`trg_patient_bi`	`patient`	Before Insert	Encrypts all fields + hashes email/phone
`trg_doctor_bi`	`doctor`	Before Insert	Encrypts email/phone, stores hashes
`trg_staff_bi`	`staff`	Before Insert	Encrypts email/phone, stores hashes
`trg_bill_bi`	`bill`	Before Insert	Encrypts total
`trg_payment_bi`	`payment`	Before Insert	Encrypts amount
`trg_appointment_bi`	`appointment`	Before Insert	Encrypts reason
`trg_user_bi`	`user`	Before Insert	Encrypts username, hashes password
`trg_payment_after_insert`	`payment`	After Insert	Automatically updates bill status after payments

Example: Patient Trigger

```
CREATE TRIGGER trg_patient_bi
BEFORE INSERT ON patient
FOR EACH ROW
BEGIN
  SET NEW.first_name = AES_ENCRYPT(NEW.first_name, get_enc_key());
  SET NEW.email_hash = SHA2(LOWER(CAST(NEW.email AS CHAR)),256);
  SET NEW.email = AES_ENCRYPT(NEW.email, get_enc_key());
END;
```

This ensures every sensitive field is encrypted automatically before insertion.

---

6. Decrypted Views

Views allow authorized users to access decrypted, human-readable data securely.

View	Description
v_patient_clear	Shows decrypted patient details.
v_bill_clear	Displays readable totals and bill information.
v_payment_clear	Reveals payment details with decrypted amounts.
v_user_clear	Shows readable usernames (passwords remain hashed).

Example

```
SELECT * FROM v_patient_clear;
```

Output will display normal names, emails, and phone numbers — even though they're stored encrypted in the main tables.

---

7. Stored Procedure

`sp_book_appointment`

This procedure safely books an appointment, checking for doctor availability and data validity before inserting.

Key Features:

- Validates patient and doctor existence.
- Prevents double-booking a doctor at the same time.
- Automatically encrypts the appointment reason.

```
CALL sp_book_appointment(1, 2, '2025-11-10', '09:30:00', 'Flu follow-up', @id);
SELECT @id; -- returns new appointment_id
```

---

8. Payment Logic (Automatic Bill Updates)

After a payment is inserted:

1. The trigger `trg_payment_after_insert` decrypts all payments for that bill.
2. It compares the **sum of payments** with the **bill total**.

3. Automatically updates bill status:
    - **OPEN** → if partial payment.
    - **PAID** → if fully paid.
    - **Error** → if payments exceed total.

This ensures billing integrity without manual intervention.

🧮 9. User Authentication and Password Verification

All passwords are stored as **SHA-256 hashes**.
Verification is done by hashing the user's input and comparing hashes.

Example Login Query

```
SELECT *
FROM user
WHERE AES_DECRYPT(username, get_enc_key()) = 'admin.admin'
  AND password = SHA2('admin',256);
```

Security Benefits

- Passwords are never decrypted or stored in plaintext.
- Even if the database is compromised, hashes cannot be reversed.

🔍 10. Searching and Matching Encrypted Data

You can search encrypted users or patients using their hashed identifiers (not the ciphertext).

Example: Search for patient by email

```
SELECT *
FROM patient
WHERE email_hash = SHA2('ananda85@gmail.com',256);
```

This ensures fast lookups without decrypting any data.

---

11. Developer Guidelines

Task	Recommendation
Inserting data	Always insert plaintext; triggers encrypt/hash automatically.
Reading data	Use decrypted views (`v_patient_clear` , `v_user_clear` , etc.).
Searching	Use hash columns for filters (`email_hash` , `phone_hash`).
Changing key	Update `secret_config` and re-encrypt existing data carefully.
Backups	Exclude `secret_config` from public dumps.

---

12. Typical Workflows

Example: Adding a new patient

```
INSERT INTO patient (first_name, last_name, dob, sex, email, phone, address,
emergency_contact)
VALUES ('Nicha', 'Wattanakul', '1997-09-12', 'F', 'nicha97@gmail.com', '0899988877',
'Bangkok', '0811119999');
```

➡ Automatically encrypted and hashed.
Check it with:

```
SELECT * FROM v_patient_clear WHERE first_name='Nicha';
```

Example: Adding a new user (doctor)

```
INSERT INTO user (username, password, role)
VALUES ('pawat.kit', 'mypassword123', 'doctor');
```

- The username becomes AES-encrypted.
- The password becomes a SHA-256 hash.
- Login comparison uses SHA2() for verification.

⚠ 13. Security Best Practices

- **Never store the AES key in plaintext** outside `secret_config`.
- **Limit access** to decrypted views only for trusted roles (e.g., admin).
- **Use SSL/TLS** for all database connections.
- **Backup policy**: Exclude `secret_config` in any exported SQL dump unless it's securely stored elsewhere.
- **Hash salting (optional)**: You may extend hashing with salt for enhanced password protection.

🗒 14. Summary Diagram (Conceptual)

```
┌──────────────────────────────────────────────┐
│            Hospital Management DB            │
├──────────────────────────────────────────────┤
│ patient, doctor, staff → AES encrypted       │
│ bill, payment → AES encrypted                │
│ user.password → SHA-256 hashed               │
│ email_hash / phone_hash → SHA-256 hashed     │
├──────────────────────────────────────────────┤
│ secret_config → holds AES key                │
│ get_enc_key() → retrieves AES key            │
├──────────────────────────────────────────────┤
│ triggers → automatic encryption / hashing    │
│ views → decrypted read-only access           │
└──────────────────────────────────────────────┘
```

🗐 15. Key Takeaways

1. **AES-256** secures readable confidential data.
2. **SHA-256** protects passwords and identifiers irreversibly.
3. **Triggers** enforce consistency and prevent human error.
4. **Views** provide safe, decrypted read access.
5. **Foreign keys** guarantee referential integrity.
6. **Secret keys** are the lifeblood — protect them above all else.

☑ Quick Reference Commands

Action	SQL
Decrypt data	`SELECT * FROM v_patient_clear;`
Verify password	`password = SHA2('input',256)`
Find by email	`email_hash = SHA2('email',256)`
Recreate AES key	`UPDATE secret_config SET enc_key = LEFT(SHA2('newkey',256),32);`

🧭 Final Notes

This HMS database is a robust foundation for a **secure healthcare management platform**.
It demonstrates **proper separation between encryption and hashing**, real-world **trigger automation**, and **auditable views** — all essential principles for any sensitive information system.

HMS Encryption and Hashing Summary

This document lists which fields in each table are **AES-encrypted (reversible)** and which are **SHA-256 hashed (irreversible)**.

PATIENT

- 🔓 Encrypted:
  - first_name
  - last_name
  - dob
  - sex
  - email
  - phone
  - address
  - emergency_contact
- 🔐 Hashed:
  - email_hash
  - phone_hash

DOCTOR

- 🔒 Encrypted:
  - email
  - phone
- 🔐 Hashed:
  - email_hash
  - phone_hash

STAFF

- 🔒 Encrypted:
  - email
  - phone
- 🔐 Hashed:
  - email_hash
  - phone_hash

BILL

- 🔒 Encrypted:
  - total
- 🔐 Hashed:
  - (none)

PAYMENT

- 🔒 Encrypted:
  - amount
- 🔐 Hashed:
  - (none)

APPOINTMENT

- 🔒 Encrypted:
  - reason
- 🔐 Hashed:
  - (none)

USER

- Encrypted:
  - username
- Hashed:
  - password
