EduTrack – Course Enrollment & Fee Management System (C.E.F.M.S.)

1) Story

EduTrack Academy is a fast-growing professional training institute offering certification programs like **Java Full Stack**, **Data Science**, **Cloud Engineering**, and **DevOps**.

Today, they run operations using a mix of **Excel sheets**, **manual batch lists**, and **separate payment notes**, which causes frequent operational and revenue issues:

Current problems

- Duplicate enrollments (same student enrolled twice in the same batch)
- Batch capacity overflow (no reliable utilization tracking)
- Students added without validated payments
- No consistent visibility into **pending payments**
- No daily revenue summary or course-wise financial reporting
- No audit trail when course fees are changed (“who changed fee, when, and why?”)

EduTrack wants a **Java-based backend system** that uses a relational DB, supports **role-based actions (Admin vs Counselor)**, logs all operations, enforces strict business rules (capacity, duplicates, inactive course/batch), and produces **SQL-heavy business reports**.

You are hired as the backend team to build the **first working version (v1)**.

---

2) What You Will Build

A **console-based Java application** (no UI frameworks) that:

- Manages **Students, Courses, Batches, Enrollments, Payments**
- Persists data in **MySQL/PostgreSQL** using **SQL + JDBC**

- Implements a **transactional enrollment + payment** workflow (commit/rollback)
- Tracks batch capacity using **database-driven utilization** (ACTIVE enrollments)
- Maintains an **audit trail for course fee changes**
- Provides business reports via SQL joins + GROUP BY
- Includes **JUnit tests** for core rules and edge cases
- Uses **Log4j2** for flow + DB error logging
- Uses Git with **feature branches + disciplined commits**

---

3) Mandatory Tech / Concept Coverage (Non-Negotiable)

Core Java (Must Use)

- OOP: encapsulation, abstraction, interfaces
- Collections: List/Map usage for validations & flows
- Enums:
    - `Role` (ADMIN, COUNSELOR)
    - `EnrollmentStatus` (CREATED, ACTIVE, CANCELLED)
    - `PaymentMode` (CASH, CARD, UPI)
    - `PaymentStatus` (SUCCESS, FAILED)
- Java Time API: `LocalDateTime` (created/paid timestamps), `LocalDate` (batch start date)
- `BigDecimal` for all fees and amounts (no double)
- Custom exceptions
- Defensive input validation

- No business logic in controller/UI layer

SQL + JDBC (Must Use)

- Properly normalized schema (min 3NF)
- CRUD with `PreparedStatement`
- Transactions during enrollment checkout
- Joins + aggregation queries for reporting
- Constraints: PK, FK, UNIQUE, NOT NULL, CHECK (where supported)

Architecture + Quality (Must Use)

- Layered design: `controller → service → dao → db`
- Controller never calls DAO directly
- No SQL inside services (DAO only)
- Transaction control only in service layer
- Centralized exception strategy
- Config via `db.properties`
- Clean code principles: SRP, no god classes

Git (Must Use)

- Feature branches
- Meaningful commit messages
- No direct commits to main
- Minimum 20 commits

JUnit (Must Use)

- Service-layer tests
- 12+ meaningful tests (rules + edge cases)
- Clear naming conventions (`should...`)

Log4j2 (Must Use)

- Console + file output
- INFO for normal flow
- ERROR with stack traces for failures
- Separate logger categories recommended (flow vs db)
- No `System.out.println()` except menus/receipts/report display

---

4) Functional Requirements

A) User Roles + Authentication

Authentication is intentionally simple: **username + role selection**.

Roles

ADMIN

- Add/update/deactivate courses
- Update course fee **with audit trail**
- Create/update/deactivate batches
- View reports

COUNSELOR

- Register student
- Search student

- Enroll student into batch (transactional)
- Accept payment during enrollment checkout
- View enrollment history
- Print enrollment receipt

Role gating

- Show/hide menu actions depending on role
- Do not allow ADMIN-only actions from counselor menus

B) Student Management

Each student must have:

- `studentId` (auto)
- `name`
- `email` (UNIQUE)
- `phone` (UNIQUE)
- `createdAt`

Operations:

- Register student
- Search by phone/email
- View student enrollment history (sorted latest first)

Rules:

- Email and phone must be unique

- Basic validation: non-empty fields; normalize phone if required

---

C) Course Management + Fee Audit (Schema-aligned)

Each course must have:

- `courseId` (auto)
- `courseCode` (UNIQUE)
- `name`
- `fee` (BigDecimal)
- `active` (boolean)
- `createdAt`, `updatedAt`

Operations:

- Add course
- Update course fee
- Deactivate course (soft delete)
- Search by code/name
- List active courses

Mandatory audit rule (critical):
Any course fee update must write to `course_fee_adjustments`:

- oldFee, newFee, deltaFee, reason, changedAt

Rules:

- Fee must be > 0
- Cannot enroll into an inactive course

D) Batch Management (Capacity + Active State)

Each batch must have:

- `batchId` (auto)
- `courseId` (FK)
- `batchName` (unique per course)
- `startDate`
- `capacity`
- `active`
- `createdAt`, `updatedAt`

Operations:

- Create batch
- Update batch (start date/capacity)
- Deactivate batch
- List batches by course
- View utilization

Rules:

- capacity must be > 0
- Cannot enroll into an inactive batch
- Utilization is computed as:
    - `ACTIVE enrollments count` vs `capacity`

E) Enrollment + Payment (Core Transactional Workflow)

Enrollment includes:

- `enrollmentId`
- `studentId`
- `batchId`
- `feeAtEnrollment` (fee locked at time of enrollment)
- `status` (CREATED, ACTIVE, CANCELLED)
- `createdAt`

Payment includes:

- `paymentId`
- `enrollmentId` (UNIQUE FK, one payment per enrollment in v1)
- `amount`
- `mode` (CASH/CARD/UPI)
- `status` (SUCCESS/FAILED)
- `paidAt`
- `createdAt`

Business Rules (Must Enforce)

1. Student cannot enroll twice in same batch
    - Enforced via DB UNIQUE(student_id, batch_id) + service validation.
2. Batch capacity must not be exceeded
    - Count ACTIVE enrollments for that batch:

        - `COUNT(*) WHERE batch_id=? AND status='ACTIVE'` must be < capacity.

3. Course must be active
4. Batch must be active
5. Payment amount must equal `feeAtEnrollment` (v1 strict match)
6. Payment status decides enrollment activation:
    - If payment SUCCESS → enrollment becomes ACTIVE
    - If payment FAILED → enrollment remains CREATED (pending)

Transaction Steps (Service Layer Only)

`enrollAndPay(student, batch, payment)` must:

- Begin transaction
- Validate: course active, batch active, capacity available, no duplicate
- Lock feeAtEnrollment from course.fee
- Insert enrollment with status CREATED
- Insert payment (SUCCESS/FAILED)
- If SUCCESS: update enrollment status to ACTIVE
- Commit
- If anything fails → rollback

Important consistency rule (reports depend on this):

- `ACTIVE` means **paid successfully**
- `CREATED` means **pending payment** (either no payment yet or FAILED payment)
- `CANCELLED` means voided enrollment (v1 optional; implement if time permits)

F) Reporting Requirements (SQL-Heavy)

Reports must use SQL joins + GROUP BY and return DTO rows (no string formatting inside DAO).

Menu options:

1. **Daily Revenue Summary (date)**
    - Total successful payments
    - Total revenue
    - Course-wise revenue breakdown (optional in same report or separate)
2. **Course-wise Enrollment Count (date range)**
    - Count ACTIVE enrollments per course
    - Optional: include CREATED count for pending pipeline visibility
3. **Batch Capacity Utilization (as-of today)**
    - For each batch: capacity vs activeEnrollmentCount vs remainingSeats
4. **Students with Pending Payments**
    - Enrollments with status CREATED
    - Include student + course + batch + feeAtEnrollment
    - Include payment status if exists (FAILED) or “NO_PAYMENT”
5. **Course Revenue Report (date range)**
    - SUM(payments.amount) where status SUCCESS
    - Group by course

---

5) Exception Handling Rules

Create custom exceptions:

- `ValidationException`
- `EntityNotFoundException`
- `CapacityExceededException`
- `DuplicateEnrollmentException`
- `DatabaseOperationException`

Rules:

- Validate early in service layer (clean messages)
- DAO catches SQLException and wraps it into `DatabaseOperationException`
- Log stack traces at ERROR level
- Console UI must show friendly messages (no stack trace dumping)

---

6) JUnit Testing Requirements (Must Have)

Minimum 12 meaningful tests across services/utilities.

Required tests (at least these):

- `shouldThrowCapacityExceededWhenBatchIsFull()`
- `shouldThrowDuplicateEnrollmentWhenStudentAlreadyEnrolled()`
- `shouldNotAllowEnrollmentForInactiveCourse()`
- `shouldNotAllowEnrollmentForInactiveBatch()`
- `shouldLockFeeAtEnrollmentEvenIfCourseFeeChangesLater()`
- `shouldFailPaymentWhenAmountMismatch()`
- `shouldKeepEnrollmentCreatedWhenPaymentFails()`

- `shouldActivateEnrollmentWhenPaymentSucceeds()`
- `shouldReturnPendingPaymentsReportCorrectly()`
- `shouldReturnBatchUtilizationAggregatedCorrectly()`

Bonus:

- `shouldRollbackTransactionWhenPaymentInsertFails()` (controlled DAO failure / fake DAO)

---

7) Evaluation Rubric (Same)

- Architecture & separation: 25%
- SQL schema + JDBC correctness: 20%
- Transaction safety: 15%
- JUnit quality: 15%
- Logging & exception handling: 10%
- Git discipline + README quality: 10%
- Code cleanliness: 5%

Starter Kit

None

```
edutrack-cefms/
├─ README.md
│  └─ HINT: Setup steps, schema init, how to run, sample user flows,
│           transaction boundary (enrollment+payment), and assumptions.
│
├─ schema.sql
│  └─ HINT: All tables + constraints + seed data (4+ courses, 6+ batches, 8+ students).
│           Include course_fee_adjustments audit table.
│
├─ pom.xml
│  └─ HINT: Dependencies: JDBC driver, Log4j2, JUnit5, Surefire plugin.
```

```
|
├─ src/
|  ├─ main/
|  |  ├─ java/com/edutrack/cefms/
|  |  |  ├─ App.java
|  |  |  |  └─ HINT: Entry point + login (username + role).
|  |  |  |         Route to role-based menus. Global exception handling.
|  |  |  |
|  |  |  ├─ config/
|  |  |  |  ├─ AppConfig.java
|  |  |  |  |  └─ HINT: Manual wiring: Services + DAOs (simple DI).
|  |  |  |  └─ DbConfig.java
|  |  |  |     └─ HINT: Read db.properties, expose settings.
|  |  |  |
|  |  |  ├─ controller/
|  |  |  |  ├─ StudentController.java
|  |  |  |  |  └─ HINT: Register/search students; view enrollment history.
|  |  |  |  ├─ CourseController.java
|  |  |  |  |  └─ HINT: Admin actions for course CRUD + fee updates (audit).
|  |  |  |  ├─ BatchController.java
|  |  |  |  |  └─ HINT: Admin actions for batch create/update/deactivate/list/search.
|  |  |  |  ├─ EnrollmentController.java
|  |  |  |  |  └─ HINT: Counselor flow:
|  |  |  |  |       select student → select batch → validate capacity/active →
|  |  |  |  |       create enrollment → take payment → activate enrollment.
|  |  |  |  └─ ReportController.java
|  |  |  |     └─ HINT: Print reports (SQL heavy) from ReportService DTOs.
|  |  |  |
|  |  |  ├─ dao/
|  |  |  |  ├─ StudentDao.java
|  |  |  |  ├─ CourseDao.java
|  |  |  |  ├─ CourseFeeAdjustmentDao.java
|  |  |  |  ├─ BatchDao.java
|  |  |  |  ├─ EnrollmentDao.java
|  |  |  |  ├─ PaymentDao.java
|  |  |  |  ├─ ReportDao.java
|  |  |  |  └─ impl/
|  |  |  |     ├─ JdbcStudentDao.java
|  |  |  |     ├─ JdbcCourseDao.java
|  |  |  |     ├─ JdbcCourseFeeAdjustmentDao.java
|  |  |  |     ├─ JdbcBatchDao.java
|  |  |  |     ├─ JdbcEnrollmentDao.java
|  |  |  |     ├─ JdbcPaymentDao.java
|  |  |  |     └─ JdbcReportDao.java
|  |  |  |        └─ HINT: JDBC only. Catch SQLException → DatabaseOperationException.
|  |  |  |
|  |  |  ├─ service/
|  |  |  |  ├─ StudentService.java
|  |  |  |  ├─ CourseService.java
|  |  |  |  ├─ BatchService.java
|  |  |  |  ├─ EnrollmentService.java
|  |  |  |  └─ ReportService.java
|  |  |  |     └─ HINT: Validate inputs, orchestrate DAOs, handle transaction boundaries.
|  |  |  |
|  |  |  ├─ model/
|  |  |  |  ├─ Student.java
|  |  |  |  ├─ Course.java
```

```
|  |  |  |  ├─ Batch.java
|  |  |  |  ├─ Enrollment.java
|  |  |  |  ├─ Payment.java
|  |  |  |  └─ report/
|  |  |  |     ├─ DailyRevenueRow.java
|  |  |  |     ├─ CourseRevenueRow.java
|  |  |  |     ├─ CourseEnrollmentCountRow.java
|  |  |  |     ├─ BatchUtilizationRow.java
|  |  |  |     └─ PendingPaymentRow.java
|  |  |  |
|  |  |  ├─ exception/
|  |  |  |  ├─ ValidationException.java
|  |  |  |  ├─ EntityNotFoundException.java
|  |  |  |  ├─ CapacityExceededException.java
|  |  |  |  ├─ DuplicateEnrollmentException.java
|  |  |  |  └─ DatabaseOperationException.java
|  |  |  |
|  |  |  ├─ util/
|  |  |  |  ├─ DbConnectionFactory.java
|  |  |  |  ├─ InputUtil.java
|  |  |  |  ├─ ValidationUtil.java
|  |  |  |  └─ MoneyUtil.java
|  |  |  |
|  |  |  └─ constants/
|  |  |     ├─ Role.java
|  |  |     ├─ EnrollmentStatus.java
|  |  |     ├─ PaymentMode.java
|  |  |     └─ PaymentStatus.java
|  |  |
|  |  └─ resources/
|  |     ├─ db.properties
|  |     |  └─ HINT: Commit template only (no passwords).
|  |     └─ log4j2.xml
|  |        └─ HINT: console + file appenders; separate loggers for flow vs db.
|  |
|  └─ test/java/com/edutrack/cefms/service/
|     ├─ EnrollmentServiceTest.java
|     ├─ BatchServiceTest.java
|     ├─ CourseServiceTest.java
|     └─ ValidationUtilTest.java
|        └─ HINT: 12+ tests, include required ones.
```

Minimal bootstrap classes (copy/paste)

Java

```java
package com.edutrack.cefms;

import com.edutrack.cefms.controller.*;
```

```java
import com.edutrack.cefms.util.InputUtil;

public class App {

    public static void main(String[] args) {

        while (true) {
            System.out.println("\n=== EduTrack C.E.F.M.S. ===");
            System.out.println("1. Students");
            System.out.println("2. Courses & Batches");
            System.out.println("3. Enrollments");
            System.out.println("4. Reports");
            System.out.println("0. Exit");

            int choice = InputUtil.readInt("Choose: ");

            switch (choice) {
                case 1 -> new StudentController().menu();
                case 2 -> new CourseController().menu();
                case 3 -> new EnrollmentController().menu();
                case 4 -> new ReportController().menu();
                case 0 -> {
                    System.out.println("Bye!");
                    return;
                }
                default -> System.out.println("Invalid option.");
            }
        }
    }
}
```

DbConnectionFactory.java (single place for connections)

Java

```java
package com.cityfest.tbems.util;

import java.io.InputStream;
import java.sql.Connection;
```

```java
import java.sql.DriverManager;
import java.util.Properties;

public final class DbConnectionFactory {
    private static final Properties props = new Properties();

    static {
        try (InputStream in =
DbConnectionFactory.class.getClassLoader().getResourceAsStream("db.properties")) {
            if (in == null) throw new IllegalStateException("db.properties not found in
resources/");
            props.load(in);
        } catch (Exception e) {
            throw new ExceptionInInitializerError("Failed to load db.properties: " +
e.getMessage());
        }
    }

    private DbConnectionFactory() {}

    public static Connection getConnection() {
        try {
            return DriverManager.getConnection(
                    props.getProperty("db.url"),
                    props.getProperty("db.username"),
                    props.getProperty("db.password")
            );
        } catch (Exception e) {
            throw new RuntimeException("DB connection failed: " + e.getMessage(), e);
        }
    }
}
```

DB Schema

None

```sql
-- ============================================================
-- EduTrack C.E.F.M.S. - Database Schema (MySQL 8+)
-- Fixes included:
--   ✅ batches.active + updated_at
--   ✅ courses.updated_at
--   ✅ course_fee_adjustments audit trail
--   ✅ CHECK constraints for enums and amounts
-- ============================================================

CREATE DATABASE IF NOT EXISTS edutrack_cefms;
USE edutrack_cefms;

-- ============================================================
-- Students
-- ============================================================
CREATE TABLE students (
  student_id BIGINT PRIMARY KEY AUTO_INCREMENT,
  name       VARCHAR(120) NOT NULL,
  email      VARCHAR(120) NOT NULL UNIQUE,
  phone      VARCHAR(20)  NOT NULL UNIQUE,
  created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP
```

```
);

-- ============================================================
-- Courses
-- ============================================================
CREATE TABLE courses (
  course_id   BIGINT PRIMARY KEY AUTO_INCREMENT,
  course_code VARCHAR(30) NOT NULL UNIQUE,
  name        VARCHAR(120) NOT NULL,
  fee         DECIMAL(12,2) NOT NULL,
  active      BOOLEAN NOT NULL DEFAULT TRUE,
  created_at  DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated_at  DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- Fee adjustment audit trail (Admin updates)
CREATE TABLE course_fee_adjustments (
  adjustment_id BIGINT PRIMARY KEY AUTO_INCREMENT,
  course_id     BIGINT NOT NULL,
  old_fee       DECIMAL(12,2) NOT NULL,
  new_fee       DECIMAL(12,2) NOT NULL,
  delta_fee     DECIMAL(12,2) NOT NULL,
  reason        VARCHAR(200) NOT NULL,
  changed_at    DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT fk_fee_adj_course FOREIGN KEY (course_id) REFERENCES courses(course_id) ON DELETE
RESTRICT
);

-- ============================================================
-- Batches
-- ============================================================
CREATE TABLE batches (
  batch_id    BIGINT PRIMARY KEY AUTO_INCREMENT,
  course_id   BIGINT NOT NULL,
  batch_name  VARCHAR(60) NOT NULL,
  start_date  DATE NOT NULL,
  capacity    INT NOT NULL,
  active      BOOLEAN NOT NULL DEFAULT TRUE,
  created_at  DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated_at  DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  CONSTRAINT fk_batch_course FOREIGN KEY (course_id) REFERENCES courses(course_id) ON DELETE
RESTRICT,
  CONSTRAINT uq_course_batch UNIQUE (course_id, batch_name)
);

CREATE INDEX idx_batches_course ON batches(course_id);
CREATE INDEX idx_batches_start ON batches(start_date);

-- ============================================================
-- Enrollments
-- One student cannot enroll twice in same batch (DB enforced).
-- status: CREATED (payment pending/failed), ACTIVE (paid), CANCELLED
-- ============================================================
CREATE TABLE enrollments (
  enrollment_id      BIGINT PRIMARY KEY AUTO_INCREMENT,
  student_id         BIGINT NOT NULL,
  batch_id           BIGINT NOT NULL,
  fee_at_enrollment  DECIMAL(12,2) NOT NULL,
  status             VARCHAR(20) NOT NULL,
  created_at         DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  UNIQUE (student_id, batch_id),
```

```
  CONSTRAINT fk_enr_student FOREIGN KEY (student_id) REFERENCES students(student_id) ON DELETE
RESTRICT,
  CONSTRAINT fk_enr_batch   FOREIGN KEY (batch_id) REFERENCES batches(batch_id) ON DELETE RESTRICT
);

CREATE INDEX idx_enr_batch ON enrollments(batch_id);
CREATE INDEX idx_enr_status ON enrollments(status);

-- ============================================================
-- Payments (1 payment per enrollment in v1)
-- If payment FAILED -> enrollment remains CREATED
-- ============================================================
CREATE TABLE payments (
  payment_id     BIGINT PRIMARY KEY AUTO_INCREMENT,
  enrollment_id  BIGINT NOT NULL UNIQUE,
  amount         DECIMAL(12,2) NOT NULL,
  mode           VARCHAR(20) NOT NULL,  -- CASH, CARD, UPI
  status         VARCHAR(20) NOT NULL,  -- SUCCESS, FAILED
  paid_at        DATETIME NULL,
  created_at     DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT fk_pay_enr FOREIGN KEY (enrollment_id) REFERENCES enrollments(enrollment_id) ON
DELETE RESTRICT
);

CREATE INDEX idx_pay_status ON payments(status);

-- ============================================================
-- CHECK constraints (MySQL 8+ enforces CHECK)
-- ============================================================
ALTER TABLE courses
  ADD CONSTRAINT chk_course_fee CHECK (fee > 0);

ALTER TABLE batches
  ADD CONSTRAINT chk_batch_capacity CHECK (capacity > 0);

ALTER TABLE enrollments
  ADD CONSTRAINT chk_enr_fee CHECK (fee_at_enrollment > 0),
  ADD CONSTRAINT chk_enr_status CHECK (status IN ('CREATED','ACTIVE','CANCELLED'));

ALTER TABLE payments
  ADD CONSTRAINT chk_pay_amount CHECK (amount >= 0),
  ADD CONSTRAINT chk_pay_mode CHECK (mode IN ('CASH','CARD','UPI')),
  ADD CONSTRAINT chk_pay_status CHECK (status IN ('SUCCESS','FAILED'));

-- ============================================================
-- Seed Data
-- ============================================================

-- Courses (4+)
INSERT INTO courses (course_code, name, fee, active) VALUES
('JAVA-FS', 'Java Full Stack',        60000.00, TRUE),
('DS-101',  'Data Science',           75000.00, TRUE),
('CLOUD',   'Cloud Engineering',      80000.00, TRUE),
('DEVOPS',  'DevOps Engineering',     65000.00, TRUE);

-- Batches (6+)
INSERT INTO batches (course_id, batch_name, start_date, capacity, active) VALUES
(1, 'JAVA-FS-B1', '2026-03-01', 30, TRUE),
(1, 'JAVA-FS-B2', '2026-04-01', 25, TRUE),
(2, 'DS-101-B1',  '2026-03-10', 20, TRUE),
(2, 'DS-101-B2',  '2026-04-15', 20, TRUE),
```

```
(3, 'CLOUD-B1',   '2026-03-20', 18, TRUE),
(4, 'DEVOPS-B1',  '2026-03-25', 22, TRUE);

-- Students (8+)
INSERT INTO students (name, email, phone) VALUES
('Aarav Sharma',   'aarav@example.com',   '9990000001'),
('Samiksha Patil', 'samiksha@example.com','9990000002'),
('Tushar Kulkarni','tushar@example.com',  '9990000003'),
('Siddhant Jain',  'siddhant@example.com','9990000004'),
('Rutuja Desai',   'rutuja@example.com',  '9990000005'),
('Sandhya Rao',    'sandhya@example.com', '9990000006'),
('Sanket Joshi',   'sanket@example.com',  '9990000007'),
('Shrinath Mehta', 'shrinath@example.com','9990000008');

-- Enrollments + payments (some ACTIVE, some pending/failed)
-- Enrollment 1: ACTIVE
INSERT INTO enrollments (student_id, batch_id, fee_at_enrollment, status) VALUES
(1, 1, 60000.00, 'ACTIVE');
INSERT INTO payments (enrollment_id, amount, mode, status, paid_at) VALUES
(1, 60000.00, 'UPI', 'SUCCESS', '2026-02-16 10:00:00');

-- Enrollment 2: CREATED with FAILED payment (pending)
INSERT INTO enrollments (student_id, batch_id, fee_at_enrollment, status) VALUES
(2, 3, 75000.00, 'CREATED');
INSERT INTO payments (enrollment_id, amount, mode, status, paid_at) VALUES
(2, 75000.00, 'CARD', 'FAILED', '2026-02-16 11:30:00');

-- Enrollment 3: CREATED (no payment yet) -> pending payment
INSERT INTO enrollments (student_id, batch_id, fee_at_enrollment, status) VALUES
(3, 5, 80000.00, 'CREATED');

-- Enrollment 4: ACTIVE
INSERT INTO enrollments (student_id, batch_id, fee_at_enrollment, status) VALUES
(4, 6, 65000.00, 'ACTIVE');
INSERT INTO payments (enrollment_id, amount, mode, status, paid_at) VALUES
(4, 65000.00, 'CASH', 'SUCCESS', '2026-02-16 12:15:00');

-- Fee adjustment seed example (audit)
INSERT INTO course_fee_adjustments (course_id, old_fee, new_fee, delta_fee, reason) VALUES
(1, 60000.00, 62000.00, 2000.00, 'Annual fee revision for 2026 intake');

UPDATE courses SET fee = 62000.00 WHERE course_id = 1;
```

`db.properties` (template)

```
db.url=jdbc:mysql://localhost:3306/<dbname>
db.username=root
db.password=YOUR_PASSWORD
```

Sample menu flow (console UX)

Login

```
=== EduTrack C.E.F.M.S. ===
Enter username: vishal
Select role:
1. ADMIN
2. COUNSELOR
Choose: 2
Logged in as COUNSELOR
```

Main Menu (role gated)

ADMIN

```
=== EduTrack ===
1. Courses (Admin)
2. Batches (Admin)
3. Reports
0. Exit
```

COUNSELOR

```
=== EduTrack ===
1. Students
2. Enrollments
3. Reports
0. Exit
```

Students Menu (COUNSELOR)

```
--- Students ---
1. Register student
2. Search student by phone
3. Search student by email
4. View student enrollment history (by phone/email)
0. Back
```

Flow: Register student

- Enter name, email, phone
- Validate: not blank, email format, normalize phone (digits only), uniqueness

- Save and display studentId

Courses Menu (ADMIN)

```
--- Courses (Admin) ---
1. Add course
2. Update course fee (writes audit trail)
3. Deactivate course
4. Search course by code/name
5. List active courses
0. Back
```

Flow: Update course fee (audit mandatory)

1. Enter courseCode
2. Show current fee
3. Enter newFee + reason
4. Validate newFee > 0
5. Transaction (recommended):
    - Insert course_fee_adjustments row
    - Update courses.fee
    - Commit

Batches Menu (ADMIN)

```
--- Batches (Admin) ---
1. Create batch
2. Update batch (start date/capacity)
3. Deactivate batch
4. List batches by course
5. View batch utilization (ACTIVE enrollments vs capacity)
0. Back
```

Enrollments Menu (COUNSELOR)

```
--- Enrollments ---
1. Enroll student + payment (transactional)
2. Find enrollment by ID
3. Find enrollments by student phone/email
4. Print enrollment receipt (by enrollment ID)
0. Back
```

Core flow: “Enroll student + payment”

Step 1: Select student

- Ask phone/email → fetch student
- If not found → offer register

Step 2: Select batch

- List active courses → select course
- List active batches of that course → select batch

Step 3: Validate business rules

- course active
- batch active
- duplicate enrollment: UNIQUE(student_id, batch_id)
- capacity: `COUNT(ACTIVE enrollments) < batch.capacity`

Step 4: Fee lock

- feeAtEnrollment = courses.fee (current)
- Show fee preview

Step 5: Payment

- Choose mode: CASH/CARD/UPI
- Enter amount

- Validate amount equals feeAtEnrollment (v1)

Transactional execution (service layer)

- Begin transaction
- Insert enrollment (CREATED)
- Insert payment (SUCCESS/FAILED)
- If SUCCESS → update enrollment status ACTIVE
- Commit
- If any DB failure → rollback

Receipt

```
--- Enrollment Receipt ---
EnrollmentId: 104
Student: Aarav Sharma (9990000001)
Course: Java Full Stack
Batch: JAVA-FS-B1 (Start: 2026-03-01)
Fee Locked: 62000.00
Payment: SUCCESS (UPI) Amount: 62000.00
Status: ACTIVE
-----------------------
```

Reports Menu (ADMIN + COUNSELOR)

```
--- Reports ---
1. Daily revenue summary (date)
2. Course-wise enrollment count (date range)
3. Batch capacity utilization (as of today)
4. Students with pending payments
5. Course revenue report (date range)
0. Back
```

Definition used (important for consistency)

- “Revenue” = SUM(payments.amount) where status = SUCCESS and paid_at date in range

- **“Pending payments” =**
    - **enrollments.status = CREATED AND (**
      payment missing OR payment.status = FAILED
)