FleetTrack – Logistics Fleet & Shipment Management System (F.L.S.M.S.)

1) Story

FleetTrack Logistics operates:

- 40 delivery trucks
- 15 drivers
- 300+ shipment orders per month

Currently, they manage:

- Shipments in Excel
- Driver assignments manually
- Fuel expenses in notebooks
- Revenue tracking via spreadsheets

This causes:

- Double-assigned vehicles
- Drivers scheduled twice on same day
- Overloaded trucks
- No audit trail for assignment changes
- Incorrect revenue summaries
- No operational visibility

FleetTrack wants a **Java-based backend system** that:

- Manages vehicles, drivers, shipments, assignments, and payments
- Prevents scheduling conflicts at DB + service level
- Enforces vehicle capacity rules
- Ensures transactional shipment booking
- Generates SQL-heavy operational reports
- Uses strict layered architecture

You are hired to build **Version 1**.

---

2) What You Will Build

A console-based Java application that:

- Manages Vehicles, Drivers, Shipments, Assignments, Payments
- Persists data using MySQL/PostgreSQL via JDBC
- Uses layered architecture
- Implements transactional shipment booking + payment
- Prevents driver & vehicle double assignment at DB level
- Uses Log4j logging
- Includes JUnit tests
- Uses Git with proper branching discipline

---

3) Mandatory Tech / Concept Coverage

Core Java (Must Use)

- OOP (encapsulation, abstraction, interfaces)
- Collections (List/Map for assignment validation logic)
- Enums:
    - Role
    - ShipmentStatus
    - PaymentStatus
    - PaymentMode
- Java Time API (LocalDateTime, LocalDate)
- BigDecimal for financial values
- Custom exceptions
- Defensive validation
- No business logic in controllers

SQL + JDBC (Must Use)

- 3NF schema design
- CRUD via PreparedStatement
- Transactions during shipment booking
- JOIN + GROUP BY for reports
- Constraints:
    - PRIMARY KEY
    - FOREIGN KEY
    - UNIQUE

    - NOT NULL
    - CHECK
    - Composite UNIQUE for assignment safety

Git (Must Use)

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

JUnit (Must Use)

Test minimum:

- Over-capacity rule
- Driver double-assignment rule
- Vehicle double-assignment rule
- Payment mismatch rule
- Inactive vehicle assignment rule
- Revenue calculation
- Minimum 12 tests

Log4j (Must Use)

- Console + file logging
- INFO for flow
- ERROR with stack trace
- No System.out for business logging

---

4) Functional Requirements

A) Roles

ADMIN

- Add/update/deactivate vehicles
- Add/update drivers
- View reports

OPERATOR

- Create shipment
- Assign vehicle + driver
- Process payment
- Track shipment

Authentication: simple username + role enum.

B) Vehicle Management

Each vehicle:

- vehicleId
- vehicleNumber (UNIQUE)
- capacityKg (DECIMAL)
- active
- createdAt

Rules:

- capacityKg > 0
- Cannot assign inactive vehicle

C) Driver Management

Each driver:

- driverId
- name
- licenseNumber (UNIQUE)
- active

Rules:

- Cannot assign inactive driver

D) Shipment (Core Domain)

Each shipment:

- shipmentId

- customerName
- origin
- destination
- weightKg
- shipmentCost
- shipmentDate (DATE) ✅ **New mandatory field**
- status (CREATED, ASSIGNED, DELIVERED)
- createdAt

Rules:

- weightKg must be positive
- weightKg must be ≤ vehicle capacity
- Cannot use inactive vehicle or driver

E) Assignment (Scheduling Safety – DB Enforced)

Each assignment:

- assignmentId
- shipmentId (UNIQUE)
- vehicleId
- driverId
- assignmentDate (DATE) ✅ **NEW (same as shipmentDate)**

Critical DB Constraints (Non-negotiable)

- UNIQUE(vehicle_id, assignment_date)
- UNIQUE(driver_id, assignment_date)

These prevent:

- Vehicle double-booking on same date
- Driver double-booking on same date

F) Transactional Booking + Payment Flow

Design Rule (Mandatory)

Shipment becomes ASSIGNED only if payment SUCCESS.

If payment fails:

- Shipment remains CREATED
- No assignment is inserted

Transaction Steps (Service Layer Only)

1. Validate vehicle exists + active
2. Validate driver exists + active
3. Validate weight ≤ vehicle capacity
4. Validate driver/vehicle not already assigned on shipmentDate
5. Insert shipment (status = CREATED)
6. Insert payment
7. If payment SUCCESS:

    - Insert assignment (with assignmentDate)
    - Update shipment status = ASSIGNED
    - Commit
8. If payment FAILED:
    - Commit shipment as CREATED
    - Do NOT insert assignment
9. On system exception → rollback everything

Transaction boundary exists only inside `ShipmentService`.

G) Payment

Each payment:

- paymentId
- shipmentId (UNIQUE)
- amount
- mode (CASH, CARD, ONLINE)
- status (SUCCESS, FAILED)
- paidAt

Rules:

- amount must equal shipmentCost
- BigDecimal comparison
- On FAILED → shipment remains CREATED

H) Shipment Attempt Audit (NEW – For Reporting Integrity)

To support "Overloaded shipment attempt report", create:

shipment_attempts

- attemptId
- customerName
- weightKg
- vehicleId
- shipmentDate
- reason (CAPACITY_EXCEEDED / INACTIVE_VEHICLE / DOUBLE_ASSIGNMENT)
- attemptedAt

Whenever validation fails before booking, log attempt.

I) Reporting (SQL Heavy)

Reports must use JOIN + GROUP BY.

Menu options:

1. Daily revenue summary
    - shipmentDate
    - total shipments ASSIGNED
    - total revenue
2. Vehicle utilization report
    - vehicle

    - shipments count
    - revenue generated
3. Driver performance report
    - driver
    - total shipments
    - total revenue
4. Overloaded shipment attempt report
    - group by vehicle
    - count failed capacity attempts
5. Shipment history by customer
    - sorted by shipmentDate DESC

---

5) Exception Handling Rules

Create:

- ValidationException
- CapacityExceededException
- DriverAssignmentException
- EntityNotFoundException
- InactiveEntityException
- DatabaseOperationException

Rules:

- Validate early in service

- Catch SQLException in DAO
- Wrap inside DatabaseOperationException
- Log stack trace at ERROR
- UI prints clean message only

---

6) JUnit Testing Requirements

Minimum required tests:

- shouldThrowCapacityExceededWhenWeightExceedsVehicleCapacity()
- shouldPreventDriverDoubleAssignmentOnSameDate()
- shouldPreventVehicleDoubleAssignmentOnSameDate()
- shouldNotAssignInactiveVehicle()
- shouldFailPaymentWhenAmountMismatch()
- shouldKeepShipmentCreatedWhenPaymentFails()
- shouldRollbackTransactionWhenUnexpectedFailureOccurs()
- shouldCalculateShipmentCostCorrectly()
- shouldReturnDailyRevenueAggregatedCorrectly()

Minimum 12 tests.

---

7) Evaluation Rubric

Criteria	Weight
Architecture & separation	25%
SQL + JDBC correctness	20%

Transaction safety	15%
JUnit quality	15%
Logging & exception handling	10%
Git discipline	10%
Code quality	5%

Starter Kit

None

```
fleettrack-flsms/
├─ README.md
│  └─ HINT: Include:
│       - Setup steps (DB creation + schema.sql execution)
│       - db.properties template usage (no passwords committed)
│       - How to run (mvn test, mvn exec:java or java -jar)
│       - Sample console flows (Create Shipment → Payment → Assignment)
│       - Critical design decisions:
│           ✅ shipment_date is mandatory
│           ✅ assignment_date = shipment_date (v1)
│           ✅ DB enforces no double assignment:
│               UNIQUE(vehicle_id, assignment_date)
│               UNIQUE(driver_id, assignment_date)
│           ✅ Shipment becomes ASSIGNED only on payment SUCCESS
│       - Assumptions: one assignment per shipment (v1), one payment per shipment (v1)
│
├─ schema.sql
│  └─ HINT: Must include:
│       - Tables + constraints exactly as fixed schema
│       - Composite UNIQUE constraints on assignments for scheduling safety
│       - shipment_attempts table (for overloaded attempt report)
│       - Seed data: 8+ vehicles, 10+ drivers, few sample shipments (optional)
│
├─ pom.xml  (or build.gradle)
│  └─ HINT: Dependencies:
│       - JDBC driver (mysql-connector-j OR postgresql)
│       - Log4j2 (api + core)
│       - JUnit5 + surefire
│       - (Optional) Mockito for service tests with fake DAOs
│
├─ src/
│  ├─ main/
│  │  ├─ java/
│  │  │  └─ com/
│  │  │     └─ fleettrack/
│  │  │        └─ flsms/
│  │  │           ├─ App.java
│  │  │           │  └─ HINT: Entry point.
│  │  │           │       - Simple login: username + role
│  │  │           │       - Shows main menu based on role gating
```

```
|  |  |           |       - Global exception handling wrapper:
|  |  |           |           catch domain exceptions -> user-friendly message
|  |  |           |           log stack traces for unexpected errors
|  |  |           |
|  |  |           ├─ config/
|  |  |           |  ├─ AppConfig.java
|  |  |           |  |  └─ HINT: Manual wiring (simple DI):
|  |  |           |  |       - Create DAOs -> Services -> Controllers
|  |  |           |  |       - Keep all object creation here (no "new" scattered)
|  |  |           |  |
|  |  |           |  └─ DbConfig.java
|  |  |           |     └─ HINT: Reads db.properties; exposes driver/url/user.
|  |  |           |          Do NOT store secrets in code.
|  |  |           |
|  |  |           ├─ controller/
|  |  |           |  ├─ VehicleController.java
|  |  |           |  |  └─ HINT: Admin menu:
|  |  |           |  |       - add/update/deactivate/search vehicles
|  |  |           |  |       - Only collect input + call VehicleService
|  |  |           |  |
|  |  |           |  ├─ DriverController.java
|  |  |           |  |  └─ HINT: Admin menu:
|  |  |           |  |       - add/deactivate/list active drivers
|  |  |           |  |
|  |  |           |  ├─ ShipmentController.java
|  |  |           |  |  └─ HINT: Operator flow:
|  |  |           |  |       - create shipment booking
|  |  |           |  |       - select vehicle + driver + shipment_date
|  |  |           |  |       - payment step
|  |  |           |  |       - print shipment receipt (console)
|  |  |           |  |       - track shipment status (CREATED/ASSIGNED/DELIVERED)
|  |  |           |  |
|  |  |           |  └─ ReportController.java
|  |  |           |     └─ HINT: Report menus:
|  |  |           |          - daily revenue summary
|  |  |           |          - vehicle utilization
|  |  |           |          - driver performance
|  |  |           |          - overloaded shipment attempt report
|  |  |           |          - shipment history by customer
|  |  |           |          - Formatting only; SQL stays in ReportDao
|  |  |           |
|  |  |           ├─ dao/
|  |  |           |  ├─ VehicleDao.java
|  |  |           |  |  └─ HINT: CRUD + findByVehicleNumber + listActive
|  |  |           |  |
|  |  |           |  ├─ DriverDao.java
|  |  |           |  |  └─ HINT: CRUD + findByLicense + listActive
|  |  |           |  |
|  |  |           |  ├─ ShipmentDao.java
|  |  |           |  |  └─ HINT:
|  |  |           |  |       - Insert shipment with shipment_date
|  |  |           |  |       - Update shipment status (CREATED/ASSIGNED/DELIVERED)
|  |  |           |  |       - Find by id / list by date / list by customer
|  |  |           |  |
|  |  |           |  ├─ AssignmentDao.java
|  |  |           |  |  └─ HINT:
|  |  |           |  |       - Insert assignment with assignment_date
```

```
|  |  |           |  |       - Queries to check conflicts:
|  |  |           |  |           existsVehicleAssignment(vehicleId, date)
|  |  |           |  |           existsDriverAssignment(driverId, date)
|  |  |           |  |       - DB UNIQUE constraints are the final safety net
|  |  |           |  |
|  |  |           |  ├─ PaymentDao.java
|  |  |           |  |  └─ HINT:
|  |  |           |  |       - Insert payment (one per shipment in v1)
|  |  |           |  |       - Fetch by shipmentId
|  |  |           |  |
|  |  |           |  ├─ ShipmentAttemptDao.java
|  |  |           |  |  └─ HINT:
|  |  |           |  |       - Insert attempt row when booking fails validation
|  |  |           |  |       - Used for "Overloaded shipment attempt report"
|  |  |           |  |
|  |  |           |  ├─ ReportDao.java
|  |  |           |  |  └─ HINT:
|  |  |           |  |       - SQL-heavy queries only (JOIN/GROUP BY/SUM/COUNT)
|  |  |           |  |       - Returns DTO rows; controller prints them
|  |  |           |  |
|  |  |           |  └─ impl/
|  |  |           |     ├─ JdbcVehicleDao.java
|  |  |           |     |  └─ HINT: PreparedStatements only; wrap SQLException.
|  |  |           |     ├─ JdbcDriverDao.java
|  |  |           |     ├─ JdbcShipmentDao.java
|  |  |           |     ├─ JdbcAssignmentDao.java
|  |  |           |     ├─ JdbcPaymentDao.java
|  |  |           |     ├─ JdbcShipmentAttemptDao.java
|  |  |           |     └─ JdbcReportDao.java
|  |  |           |        └─ HINT:
|  |  |           |             - Catch SQLException -> throw DatabaseOperationException
|  |  |           |             - No printing here
|  |  |           |
|  |  |           ├─ service/
|  |  |           |  ├─ VehicleService.java
|  |  |           |  |  └─ HINT: Validate vehicleNumber format + capacity > 0; manage active flag.
|  |  |           |  |
|  |  |           |  ├─ DriverService.java
|  |  |           |  |  └─ HINT: Validate license uniqueness; manage active flag.
|  |  |           |  |
|  |  |           |  ├─ ShipmentService.java
|  |  |           |  |  └─ HINT: Core transaction boundary lives here.
|  |  |           |  |       Transaction flow (must match spec):
|  |  |           |  |       1) validate vehicle active
|  |  |           |  |       2) validate driver active
|  |  |           |  |       3) validate weight <= capacity
|  |  |           |  |       4) validate no double assignment on shipment_date
|  |  |           |  |       5) insert shipment (CREATED)
|  |  |           |  |       6) insert payment
|  |  |           |  |       7) if SUCCESS -> insert assignment (assignment_date = shipment_date)
|  |  |           |  |                     -> update shipment status ASSIGNED
|  |  |           |  |       8) commit; rollback on unexpected failures
|  |  |           |  |       Also: insert shipment_attempts when validation fails (capacity/double
assignment)
|  |  |           |  |
|  |  |           |  └─ ReportService.java
|  |  |           |     └─ HINT: Validates date input/range; calls ReportDao; returns DTO rows.
```

```
|  |  |           |
|  |  |           ├─ model/
|  |  |           |  ├─ Vehicle.java
|  |  |           |  |  └─ HINT: capacityKg as BigDecimal; active boolean; createdAt LocalDateTime.
|  |  |           |  ├─ Driver.java
|  |  |           |  ├─ Shipment.java
|  |  |           |  |  └─ HINT: shipmentDate as LocalDate (not datetime).
|  |  |           |  ├─ Assignment.java
|  |  |           |  |  └─ HINT: assignmentDate as LocalDate (mirrors shipmentDate).
|  |  |           |  ├─ Payment.java
|  |  |           |  ├─ ShipmentAttempt.java
|  |  |           |  └─ report/
|  |  |           |     ├─ DailyRevenueRow.java
|  |  |           |     ├─ VehicleUtilizationRow.java
|  |  |           |     ├─ DriverPerformanceRow.java
|  |  |           |     ├─ OverloadAttemptRow.java
|  |  |           |     └─ CustomerShipmentHistoryRow.java
|  |  |           |     └─ HINT: DTOs only; keep printing out of DAO.
|  |  |           |
|  |  |           ├─ exception/
|  |  |           |  ├─ ValidationException.java
|  |  |           |  |  └─ HINT: bad inputs (negative weight, invalid date, empty fields).
|  |  |           |  ├─ CapacityExceededException.java
|  |  |           |  |  └─ HINT: thrown when weight exceeds capacity.
|  |  |           |  ├─ DriverAssignmentException.java
|  |  |           |  |  └─ HINT: thrown when driver/vehicle already assigned on same date.
|  |  |           |  ├─ EntityNotFoundException.java
|  |  |           |  ├─ InactiveEntityException.java
|  |  |           |  └─ DatabaseOperationException.java
|  |  |           |     └─ HINT: DAO wraps SQLException -> this exception; service decides rollback.
|  |  |           |
|  |  |           ├─ util/
|  |  |           |  ├─ DbConnectionFactory.java
|  |  |           |  |  └─ HINT: single connection creator from db.properties.
|  |  |           |  |       Service layer should control autoCommit for transactions.
|  |  |           |  ├─ InputUtil.java
|  |  |           |  |  └─ HINT: safe readInt/readString/readDate; re-prompt on invalid input.
|  |  |           |  ├─ ValidationUtil.java
|  |  |           |  |  └─ HINT: vehicle number format, license format, weight > 0, date parsing.
|  |  |           |  ├─ MoneyUtil.java
|  |  |           |  |  └─ HINT: BigDecimal parsing/scale=2; compareTo for equality checks.
|  |  |           |  └─ DateUtil.java
|  |  |           |     └─ HINT: standardize date format (ISO yyyy-MM-dd).
|  |  |           |
|  |  |           └─ constants/
|  |  |              ├─ Role.java
|  |  |              |  └─ HINT: ADMIN / OPERATOR.
|  |  |              ├─ ShipmentStatus.java
|  |  |              |  └─ HINT: CREATED, ASSIGNED, DELIVERED.
|  |  |              ├─ PaymentMode.java
|  |  |              |  └─ HINT: CASH, CARD, ONLINE.
|  |  |              └─ PaymentStatus.java
|  |  |                 └─ HINT: SUCCESS, FAILED.
|  |  |
|  |  └─ resources/
|  |     ├─ db.properties
|  |     |  └─ HINT: Commit TEMPLATE ONLY:
```

```
|  |     |       db.url=...
|  |     |       db.username=...
|  |     |       db.password=YOUR_PASSWORD
|  |     └─ log4j2.xml
|  |        └─ HINT:
|  |             - Console + rolling file appender
|  |             - INFO for app flow
|  |             - ERROR logs include stack traces (exceptions)
|  |             - Optional: separate logger name for DAO layer
|  |
|  └─ test/
|     └─ java/
|        └─ com/fleettrack/flsms/service/
|           ├─ ShipmentServiceTest.java
|           |  └─ HINT:
|           |       - capacity exceeded test
|           |       - driver/vehicle double assignment test (same shipment_date)
|           |       - payment mismatch test
|           |       - shouldKeepShipmentCreatedWhenPaymentFails()
|           |       - rollback behavior (bonus with fake dao)
|           |
|           ├─ VehicleServiceTest.java
|           |  └─ HINT: validate capacity > 0; vehicleNumber uniqueness.
|           |
|           ├─ DriverServiceTest.java
|           |  └─ HINT: license uniqueness + deactivate behavior.
|           |
|           └─ ValidationUtilTest.java
|              └─ HINT: date parsing, money parsing, weight validation, formats.
```

Minimal bootstrap classes

Java

```java
public class App {

    public static void main(String[] args) {

        while (true) {
            System.out.println("\n=== FleetTrack F.L.S.M.S. ===");
            System.out.println("1. Vehicles");
            System.out.println("2. Drivers");
            System.out.println("3. Shipments");
            System.out.println("4. Reports");
            System.out.println("0. Exit");
```

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

            switch (choice) {
                case 1 -> new VehicleController().menu();
                case 2 -> new DriverController().menu();
                case 3 -> new ShipmentController().menu();
                case 4 -> new ReportController().menu();
                case 0 -> {
                    System.out.println("Bye!");
                    return;
                }
            }
        }
    }
}
```

DbConnectionFactory.java (single place for connections)

Java

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

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

```
CREATE DATABASE IF NOT EXISTS fleettrack_flsms;
USE fleettrack_flsms;

-- ============================================================
-- Vehicles
-- ============================================================
CREATE TABLE vehicles (
  vehicle_id      BIGINT PRIMARY KEY AUTO_INCREMENT,
  vehicle_number  VARCHAR(30) NOT NULL UNIQUE,
  capacity_kg     DECIMAL(10,2) 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
);

-- ============================================================
-- Drivers
-- ============================================================
CREATE TABLE drivers (
  driver_id       BIGINT PRIMARY KEY AUTO_INCREMENT,
  name            VARCHAR(120) NOT NULL,
  license_number  VARCHAR(50) NOT NULL UNIQUE,
  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
);

-- ============================================================
-- Shipments
-- NOTE: shipment_date is mandatory to enforce "same date" scheduling rules
-- ============================================================
CREATE TABLE shipments (
  shipment_id     BIGINT PRIMARY KEY AUTO_INCREMENT,
  customer_name   VARCHAR(120) NOT NULL,
  origin          VARCHAR(120) NOT NULL,
  destination     VARCHAR(120) NOT NULL,
  shipment_date   DATE NOT NULL,                -- ✅ critical fix
  weight_kg       DECIMAL(10,2) NOT NULL,
  shipment_cost   DECIMAL(12,2) NOT NULL,
```

```
  status          VARCHAR(20) NOT NULL,          -- CREATED, ASSIGNED, DELIVERED
  created_at      DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated_at      DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

CREATE INDEX idx_shipments_date ON shipments (shipment_date);
CREATE INDEX idx_shipments_customer ON shipments (customer_name);

-- ============================================================
-- Assignments
-- One shipment -> one assignment (v1), with date-level conflict protection
-- ============================================================
CREATE TABLE assignments (
  assignment_id    BIGINT PRIMARY KEY AUTO_INCREMENT,
  shipment_id      BIGINT NOT NULL UNIQUE,       -- one assignment per shipment in v1
  vehicle_id       BIGINT NOT NULL,
  driver_id        BIGINT NOT NULL,
  assignment_date  DATE NOT NULL,                -- ✅ critical fix (v1 = shipment_date)
  created_at       DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,

  CONSTRAINT fk_assign_shipment FOREIGN KEY (shipment_id)
    REFERENCES shipments(shipment_id) ON DELETE RESTRICT,

  CONSTRAINT fk_assign_vehicle FOREIGN KEY (vehicle_id)
    REFERENCES vehicles(vehicle_id) ON DELETE RESTRICT,

  CONSTRAINT fk_assign_driver FOREIGN KEY (driver_id)
    REFERENCES drivers(driver_id) ON DELETE RESTRICT,

  -- ✅ DB-level enforcement: no double assignment on same day
  CONSTRAINT uq_vehicle_date UNIQUE (vehicle_id, assignment_date),
  CONSTRAINT uq_driver_date  UNIQUE (driver_id, assignment_date)
);

CREATE INDEX idx_assign_date ON assignments (assignment_date);

-- ============================================================
-- Payments
-- One payment per shipment (v1)
-- ============================================================
CREATE TABLE payments (
  payment_id   BIGINT PRIMARY KEY AUTO_INCREMENT,
  shipment_id  BIGINT NOT NULL UNIQUE,
  amount       DECIMAL(12,2) NOT NULL,
  mode         VARCHAR(20) NOT NULL,             -- CASH, CARD, ONLINE
  status       VARCHAR(20) NOT NULL,             -- SUCCESS, FAILED
  paid_at      DATETIME NULL,
  created_at   DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,

  CONSTRAINT fk_pay_shipment FOREIGN KEY (shipment_id)
    REFERENCES shipments(shipment_id) ON DELETE RESTRICT
);

CREATE INDEX idx_pay_status ON payments (status);

-- ============================================================
-- Shipment Attempts (Audit for failed booking attempts)
-- Used for "Overloaded shipment attempt report"
-- ============================================================
CREATE TABLE shipment_attempts (
  attempt_id       BIGINT PRIMARY KEY AUTO_INCREMENT,
```

```
  shipment_date    DATE NOT NULL,
  customer_name    VARCHAR(120) NOT NULL,
  origin           VARCHAR(120) NOT NULL,
  destination      VARCHAR(120) NOT NULL,
  weight_kg        DECIMAL(10,2) NOT NULL,
  attempted_vehicle_id BIGINT NULL,
  attempted_driver_id  BIGINT NULL,
  reason_code      VARCHAR(40) NOT NULL,         -- CAPACITY_EXCEEDED, DRIVER_DOUBLE_ASSIGNED,
VEHICLE_DOUBLE_ASSIGNED, INACTIVE_ENTITY, VALIDATION_FAILED
  notes            VARCHAR(250) NULL,
  created_at       DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,

  CONSTRAINT fk_attempt_vehicle FOREIGN KEY (attempted_vehicle_id)
    REFERENCES vehicles(vehicle_id) ON DELETE SET NULL,

  CONSTRAINT fk_attempt_driver FOREIGN KEY (attempted_driver_id)
    REFERENCES drivers(driver_id) ON DELETE SET NULL
);

CREATE INDEX idx_attempt_date ON shipment_attempts (shipment_date);
CREATE INDEX idx_attempt_reason ON shipment_attempts (reason_code);

-- ============================================================
-- CHECK constraints (MySQL 8+ enforces CHECK)
-- ============================================================
ALTER TABLE vehicles
  ADD CONSTRAINT chk_vehicle_capacity CHECK (capacity_kg > 0);

ALTER TABLE shipments
  ADD CONSTRAINT chk_shipment_weight CHECK (weight_kg > 0),
  ADD CONSTRAINT chk_shipment_cost CHECK (shipment_cost >= 0),
  ADD CONSTRAINT chk_shipment_status CHECK (status IN ('CREATED','ASSIGNED','DELIVERED'));

ALTER TABLE payments
  ADD CONSTRAINT chk_payment_amount CHECK (amount >= 0),
  ADD CONSTRAINT chk_payment_status CHECK (status IN ('SUCCESS','FAILED')),
  ADD CONSTRAINT chk_payment_mode CHECK (mode IN ('CASH','CARD','ONLINE'));

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

-- Vehicles (8+)
INSERT INTO vehicles (vehicle_number, capacity_kg, active) VALUES
('MH12TRK1001', 1200.00, TRUE),
('MH12TRK1002', 1500.00, TRUE),
('MH12TRK1003', 800.00,  TRUE),
('MH12TRK1004', 2000.00, TRUE),
('MH12TRK1005', 1000.00, TRUE),
('RJ19TRK2001', 1800.00, TRUE),
('RJ19TRK2002', 1400.00, TRUE),
('GJ01TRK3001', 1600.00, FALSE); -- inactive

-- Drivers (10+)
INSERT INTO drivers (name, license_number, active) VALUES
('Ravi Patil',   'LIC-MH-10001', TRUE),
('Neha Sharma',  'LIC-MH-10002', TRUE),
('Amit Verma',   'LIC-MH-10003', TRUE),
('Sana Khan',    'LIC-MH-10004', TRUE),
('John Dsouza',  'LIC-MH-10005', TRUE),
('Kiran Joshi',  'LIC-RJ-20001', TRUE),
```

```
('Meera Singh',  'LIC-RJ-20002', TRUE),
('Farhan Ali',   'LIC-GJ-30001', TRUE),
('Pooja Nair',   'LIC-GJ-30002', FALSE), -- inactive
('Dev Malhotra', 'LIC-MH-10006', TRUE);

-- Sample shipments (mix of CREATED/ASSIGNED/DELIVERED)
INSERT INTO shipments (customer_name, origin, destination, shipment_date, weight_kg,
shipment_cost, status) VALUES
('BlueMart Retail',  'Pune',     'Mumbai',   '2026-02-16', 900.00,  7500.00, 'ASSIGNED'),
('AutoParts Hub',    'Mumbai',   'Nashik',   '2026-02-16', 700.00,  6200.00, 'DELIVERED'),
('FreshFoods Co',    'Pune',     'Nagpur',   '2026-02-17', 1400.00, 14000.00,'CREATED'),
('MediQuick',        'Nashik',   'Pune',     '2026-02-17', 500.00,  4200.00, 'ASSIGNED'),
('HomeStyle',        'Mumbai',   'Surat',    '2026-02-18', 1600.00, 15500.00,'CREATED');

-- Payments (one per shipment, v1)
-- shipment_id ordering matches insertion order above: 1..5
INSERT INTO payments (shipment_id, amount, mode, status, paid_at) VALUES
(1, 7500.00,  'UPI',    'SUCCESS', '2026-02-16 09:10:00'), -- NOTE: UPI isn't allowed by CHECK
(mode IN CASH/CARD/ONLINE)
-- Fix: use ONLINE instead of UPI
(2, 6200.00,  'CARD',   'SUCCESS', '2026-02-16 10:30:00'),
(3, 14000.00, 'ONLINE', 'FAILED',  '2026-02-17 11:05:00'),
(4, 4200.00,  'CASH',   'SUCCESS', '2026-02-17 14:20:00'),
(5, 15500.00, 'ONLINE', 'FAILED',  '2026-02-18 16:10:00');

-- IMPORTANT: The first payment row above uses UPI which violates the CHECK.
-- Replace it with ONLINE.
UPDATE payments
SET mode = 'ONLINE'
WHERE shipment_id = 1;

-- Assignments (only for PAID/SUCCESS shipments in v1)
-- assignment_date = shipment_date (v1 rule)
INSERT INTO assignments (shipment_id, vehicle_id, driver_id, assignment_date) VALUES
(1, 1, 1, '2026-02-16'),
(4, 2, 2, '2026-02-17'),
(2, 3, 3, '2026-02-16');  -- delivered shipment still had an assignment

-- Shipment attempts (audit examples)
INSERT INTO shipment_attempts (
  shipment_date, customer_name, origin, destination, weight_kg,
  attempted_vehicle_id, attempted_driver_id, reason_code, notes
) VALUES
('2026-02-17', 'FreshFoods Co', 'Pune', 'Nagpur', 2500.00, 1, 4, 'CAPACITY_EXCEEDED', 'Weight 2500
> vehicle capacity 1200'),
('2026-02-16', 'CityElectro',   'Pune', 'Mumbai', 600.00, 1, 1, 'VEHICLE_DOUBLE_ASSIGNED',
'Vehicle already assigned on 2026-02-16'),
('2026-02-16', 'CityElectro',   'Pune', 'Mumbai', 600.00, 2, 1, 'DRIVER_DOUBLE_ASSIGNED', 'Driver
already assigned on 2026-02-16'),
('2026-02-18', 'MegaTextiles',  'Mumbai','Surat', 900.00, 8, 9, 'INACTIVE_ENTITY', 'Attempted
inactive vehicle/driver');
```

`db.properties` (template)

```
db.url=jdbc:mysql://localhost:3306/<dbname>
```

```
db.username=root
db.password=YOUR_PASSWORD
```

Sample Console Menu Flow (Console UX) — FleetTrack F.L.S.M.S.

Login

```
=== FleetTrack F.L.S.M.S. ===
Enter username: vishal
Select role:
1. ADMIN
2. OPERATOR
Choose: 2
Logged in as OPERATOR
```

Main Menu (role gated)

If ADMIN

```
=== FleetTrack ===
1. Vehicles (Admin)
2. Drivers (Admin)
3. Reports
0. Exit
```

If OPERATOR

```
=== FleetTrack ===
1. Shipments
2. Reports
0. Exit
```

Vehicles Menu (ADMIN)

```
--- Vehicles (Admin) ---
1. Add vehicle
```

```
2. Update vehicle capacity
3. Deactivate vehicle
4. View vehicle by number
5. List active vehicles
0. Back
```

Recommended flow: Add vehicle

1. Enter vehicle number
2. Enter capacityKg
3. Validate: number not blank, unique; capacity > 0
4. Save and print vehicleId

Drivers Menu (ADMIN)

```
--- Drivers (Admin) ---
1. Add driver
2. Deactivate driver
3. View driver by license
4. List active drivers
0. Back
```

Recommended flow: Deactivate driver

- Enter licenseNumber → find driver → set active=false → confirm

---

Shipments Menu (OPERATOR)

```
--- Shipments ---
1. Create shipment booking + payment (transactional)
2. Track shipment by ID
3. Mark shipment as DELIVERED
4. List shipments by date
5. List shipments by customer name
```

```
0. Back
```

Core flow: “Create shipment booking + payment”

```
Enter customer name:
Enter origin:
Enter destination:
Enter shipment date (yyyy-MM-dd):
Enter weightKg:
Enter shipment cost:
```

Select vehicle & driver

```
List active vehicles:
[1] MH12TRK1001 (1200kg)
[2] MH12TRK1002 (1500kg)
...
Choose vehicleId:

List active drivers:
[1] Ravi Patil
[2] Neha Sharma
...
Choose driverId:
```

Validations (must happen in service layer)

- vehicle active
- driver active
- weightKg <= vehicle.capacityKg
- no existing assignment for same `shipment_date`:
    - `vehicle_id + date` unique
    - `driver_id + date` unique

If validation fails:

- Insert row in `shipment_attempts` with reason_code

- Show user-friendly message:

```
Booking failed: Vehicle capacity exceeded.
(Audit saved in shipment_attempts)
```

Payment

```
Select payment mode:
1. CASH
2. CARD
3. ONLINE
Choose:
Enter amount:
Confirm payment? (y/n):
```

Transactional execution (service layer)

- Insert shipment with status CREATED
- Insert payment (SUCCESS/FAILED)
- If SUCCESS:
    - Insert assignment with assignment_date = shipment_date
    - Update shipment status = ASSIGNED
    - Commit
- If FAILED:
    - Shipment remains CREATED
    - No assignment created
    - Commit (or rollback only if unexpected DB failure)

Receipt

```
--- Shipment Receipt ---
ShipmentId: 12
Customer: BlueMart Retail
Date: 2026-02-17
Status: ASSIGNED
Vehicle: MH12TRK1002
```

```
Driver: Neha Sharma
Payment: SUCCESS (CARD) Amount: 14000.00
------------------------
```

Reports Menu (ADMIN + OPERATOR)

```
--- Reports ---
1. Daily revenue summary (date)
2. Vehicle utilization report (date range)
3. Driver performance report (date range)
4. Overloaded shipment attempt report (date range)
5. Shipment history by customer (customer name)
0. Back
```

Expected output examples

Daily revenue summary

- Input: date
- Output:
  - total successful payments
  - total revenue (SUM)
  - shipments assigned/delivered count

Vehicle utilization

- Shows count of assignments per vehicle (date range)

Driver performance

- Shows count of assignments per driver + delivered shipments (date range)

Overloaded shipment attempt report

- Uses `shipment_attempts`
- Group by reason_code / date / vehicleId etc.

Shipment history by customer

- **Show shipments sorted by created_at DESC**
- **include status + payment status**