MGT4850: FleetFlow Logistics Business Analysis

By: Divya Pateliya

Solution Overview:

FleetFlow Logistics is a regional delivery company that tracks vehicles, deliveries, and maintenance activities across multiple spreadsheets and paper records. This makes it difficult to answer basic operational questions reliably ("Which vehicles are overused?", "Which routes are busiest?", "What maintenance has been done on a specific truck?"). In order to address this issue, I have implemented a lightweight Python Flask app that uses SQLite3 to quickly pilot a solution. This prototype runs as a single file (fleetflow.db), requires no database server, supports standard SQL and ACID transactions, and has effectively zero licensing cost.

Furthermore, I created a simple database schema around the main core entities:

- **vehicles**: each vehicle's type, capacity, status, license plate, and odometer.
- **routes**: origin, destination, distance, and whether the route is currently active.
- **deliveries**: records linking vehicles and routes, with delivery dates, customer details, and status.
- **maintenance_logs**: service records, including service type, vendor, odometer at service, and cost.
- **audit_log**: a trail of insert/update/delete actions across the main tables.

Depending on the success of this pilot, if FleetFlow wants to expand to multiple depots and many concurrent users, this design can be migrated to a server-based database (e.g., PostgreSQL or a managed cloud database) while keeping the same logical schema. The current pilot focuses on being simple, reliable, and inexpensive to get FleetFlow out of spreadsheets and into a structured, auditable system.

CRUD Support for Operations:

Here is a simple breakdown of the CRUD support my solution gives:

Module	**Key Functions (CRUD)**
Vehicles	- **Create** new vehicles with ID, type, capacity, license plate. - **Read** current fleet overview including status and odometer. - **Update** vehicle status (e.g., *active* → *maintenance*). - **Delete** retired vehicles (only if no dependent deliveries).
Deliveries	- **Create** new delivery records linked to vehicles and routes. - **Read** delivery lists by date, vehicle, or route. - **Update** delivery status (*pending, in_transit, completed, cancelled*). - **Delete** incorrect or test entries.
Maintenance Logs	- **Create** maintenance entries (service date, vendor, cost, odometer, type). - **Read** all maintenance records per vehicle. - **Update** service details if corrected. - **Delete** old or duplicate logs.
Reports	- **Read** only: aggregated data views. • Vehicle Utilization Report – number of deliveries per vehicle. • Deliveries per Route Report – delivery count per route.
Audit Log	- **Read** only: All transactions on the DB

MGT4850: FleetFlow Logistics Business Analysis
By: Divya Pateliya

Costs & Hosting:

Category	Details	Estimated Cost
Software Stack	- SQLite: Lightweight, file-based database (no server needed). - Flask (Python): Open-source web framework. - Bootstrap: Used via CDN for styling, no installation required.	$0 (open-source and license-free)
Infrastructure Options	Option 1: On-Premise PC – Run locally on an existing Option 2: Cloud VM (e.g., DigitalOcean, AWS, Azure) – Host Flask app + SQLite remotely.	Option 1: No additional cost. Option 2: ~$10–$15 per month.
Data Volume & Storage	SQLite file (fleetflow.db) stores all data in a few MBs even after tens of thousands of records.	Negligible (<$0.10/month in cloud storage).
Backup & Maintenance	Simple file-copy backup strategy (nightly copy to network drive or cloud bucket).	Minimal setup time; negligible ongoing cost.
Future Scaling	Migration path to PostgreSQL/MySQL or managed cloud database; deploy Flask on container or app service.	Future cost ≈ $20–$50 / month depending on usage.
Training	Staff orientation and one-time setup of environment (Python, Flask).	~3–5 hours of staff time.

Time to implement:

Phase	Key Activities	Est. Time
1. Design & Planning	- Define data entities (vehicles, routes, deliveries, maintenance). - Establish relationships, constraints, and data integrity rules.	2–3 days
2. Development	- Implement SQLite schema and sample data loader. - Build Flask CRUD routes for vehicles, deliveries, and maintenance logs. - Integrate audit logging and reports.	7–10 days
3. Testing & Data Validation	- Populate database with test data. - Run end-to-end tests (add, edit, delete records). - Verify reports and foreign key constraints.	3–5 days
4. Deployment & Training	- Deploy on local or cloud environment. - Train staff on app usage and basic troubleshooting. - Document setup and backup process.	2–3 days
5. Documentation & Handover	- Write README, backup guide, and business memo. - Provide AI usage disclosure for transparency.	1–2 days

Maintenance and Audit Plan:

The proposed solution is designed for simplicity and reliability. Daily data entry and updates are handled directly through the web interface, minimizing errors and technical overhead. Routine database backups, performed nightly or weekly, ensure data safety with virtually no maintenance cost. The integrated audit log provides continuous accountability, automatically recording every change made to the system.

Risks, Limitations, and Scaling:

Since we are using SQLite3-based solution, the application is ideal for a single-location pilot but not for high-concurrency, multi-site use. As the company's operations expand, potential risks include data entry errors, limited simultaneous users, and local hardware dependency. To mitigate these scaling issues, the company can migrate to a server-based database such as PostgreSQL or a cloud platform once transaction volume or user count increases.