Fleet Operations Platform - Database Documentation

Version: 1.0

Last Updated: January 9, 2026

Database: MySQL 8.0

Status: Production-Ready

Executive Summary

This document provides a comprehensive overview of the Fleet Operations Platform database architecture. The system manages client projects, marketing campaigns, vendor operations, vehicle fleet management, driver assignments, and financial transactions.

The database is designed with:

• **Role-based access control** for secure data management

• **Audit trails** through soft deletes and timestamps

• **Scalable relationships** supporting multiple business modules

• **Data integrity** through foreign key constraints

Table of Contents

Database Overview

Total Tables: 14

The database consists of 14 interconnected tables organized into the following modules:

- **User Management:** 1 table (users)
- **Client & Projects:** 2 tables (clients, projects)
- **Campaign Management:** 2 tables (campaigns, promoter_activities)
- **Vendor & Fleet:** 4 tables (vendors, vehicles, drivers, promoters)
- **Financial Management:** 3 tables (invoices, payments, expenses)
- **Reporting:** 1 table (reports)

Common Fields (All Tables)

Every table inherits these standard fields from the base model:

Field	Type	Purpose
\`id\`	Integer	Unique identifier (Primary Key)
\`created_at\`	DateTime	Record creation timestamp
\`updated_at\`	DateTime	Last modification timestamp
\`is_active\`	Boolean	Soft delete flag (True = Active, False = Deleted)

Note: Soft deletes ensure data is never permanently removed from the database. Deleted records have `is_active=0` and are hidden from queries but remain for audit purposes.

Core Tables

1. Users Table

Purpose: Manages system user accounts and authentication

Table Name: `users`

Column	Type	Description
id	Integer (PK)	Unique user identifier
email	String(255)	Login email (unique)
name	String(255)	Full name
phone	String(20)	Contact number
password_hash	String(255)	Encrypted password
password_hint	String(255)	Admin reference for password recovery
role	Enum	User role (see Role Types below)
vendor_id	Integer (FK)	Links to vendors table if role=vendor

Available Roles:

- `admin` - Full system access
- `client_servicing` - Client relationship management
- `operations_manager` - Campaign & operations oversight
- `accounts` - Financial operations
- `vendor` - Vendor portal access (linked to vendor_id)
- `client` - Client portal access
- `sales`, `purchase`, `operator` - Extended roles
- `driver`, `promoter`, `anchor` - Field staff roles
- `vehicle_manager`, `godown_manager` - Resource management

Foreign Keys:

- `vendor_id` → vendors.id (SET NULL on delete)

Business Use:

- Authenticates users logging into the platform
- Controls access to features based on role
- Links vendor users to their vendor record

2. Clients Table

Purpose: Stores client company information

Table Name: `clients`

Column	Type	Description
id	Integer (PK)	Unique client identifier

Column	Type	Description
name	String(255)	Client/company name
company	String(255)	Company legal name
email	String(255)	Primary contact email
phone	String(20)	Contact number
address	Text	Full address
contact_person	String(255)	Key contact name

Relationships:

• One client can have **many projects** (One-to-Many)

Business Use:

• Central repository for client information

• Links to all projects for that client

• Used in client selection for project creation

3. Projects Table

Purpose: Tracks client projects

Table Name: `projects`

Column	Type	Description
id	Integer (PK)	Unique project identifier
name	String(255)	Project name
description	Text	Project details
client_id	Integer (FK)	Links to client
budget	Float	Project budget
start_date	Date	Project start date
end_date	Date	Project end date
status	String(50)	Status (active/completed/cancelled)
assigned_cs	String(255)	Client servicing manager name

Foreign Keys:

• `client_id` → clients.id (CASCADE on delete)

Relationships:

• Belongs to **one client** (Many-to-One)

• Can have **many campaigns** (One-to-Many)

Business Use:

• Organizes work by client projects

• Each project can have multiple campaigns

• Tracks project budget and timeline

4. Campaigns Table

Purpose: Manages marketing campaigns under projects

Table Name: `campaigns`

Column	Type	Description
id	Integer (PK)	Unique campaign identifier
name	String(255)	Campaign name
description	Text	Campaign details
project_id	Integer (FK)	Parent project
campaign_type	Enum	Type: l_shape, btl, roadshow, sampling, other
status	Enum	Status: planning, upcoming, running, hold, completed, cancelled
start_date	Date	Campaign start
end_date	Date	Campaign end
budget	Float	Campaign budget
locations	Text	Campaign locations (comma-separated)

Foreign Keys:

• `project_id` → projects.id (CASCADE on delete)

Relationships:

• Belongs to **one project** (Many-to-One)

• Can have **many expenses** (One-to-Many)

• Can have **many reports** (One-to-Many)

• Can have **many invoices** (One-to-Many)

• Can have **many promoter activities** (One-to-Many)

Business Use:

• Core operational unit for marketing activities

• Links to all campaign-related data (expenses, reports, invoices)

• Status tracking for campaign lifecycle

Campaign-Vendor Relationship:

• Campaigns are linked to vendors through **invoices**

• When creating a campaign, placeholder invoices establish vendor assignments

• Example: Campaign ID 10 → Invoice (campaign_id=10, vendor_id=5)

5. Vendors Table

Purpose: Manages vendor/supplier information

Table Name: `vendors`

Column	Type	Description
id	Integer (PK)	Unique vendor identifier
name	String(255)	Vendor name
company	String(255)	Company name
email	String(255)	Contact email
phone	String(20)	Contact number
address	Text	Full address
contact_person	String(255)	Key contact name

Relationships:

• Can have **many vehicles** (One-to-Many)

• Can have **many drivers** (One-to-Many)

• Can have **many invoices** (One-to-Many)

• Can have **many payments** (One-to-Many)

• Can be linked to **many user accounts** (One-to-Many through users.vendor_id)

Business Use:

• Central vendor master data

• Links to all vendor-owned resources (vehicles, drivers)

• Foundation for vendor portal access (users with role=vendor)

6. Vehicles Table

Purpose: Tracks fleet vehicles

Table Name: `vehicles`

Column	Type	Description
id	Integer (PK)	Unique vehicle identifier
vehicle_number	String(50)	Registration number (unique)
vehicle_type	String(100)	Type (truck, van, car, etc.)
capacity	String(100)	Load capacity
vendor_id	Integer (FK)	Owning vendor
rc_validity	Date	Registration certificate expiry
insurance_validity	Date	Insurance expiry
permit_validity	Date	Permit expiry

Foreign Keys:

• `vendor_id` → vendors.id (SET NULL on delete)

Relationships:

• Belongs to **one vendor** (Many-to-One)

• Can be assigned to **one driver** (One-to-One through drivers.vehicle_id)

Business Use:

• Fleet management

• Compliance tracking (RC, insurance, permit dates)

• Assignment to drivers for operations

7. Drivers Table

Purpose: Manages driver information and assignments

Table Name: `drivers`

Column	Type	Description
id	Integer (PK)	Unique driver identifier
name	String(255)	Driver name
phone	String(20)	Contact number
email	String(255)	Email address
license_number	String(100)	Driving license number
license_validity	Date	License expiry date
vendor_id	Integer (FK)	Associated vendor

Column	Type	Description
vehicle_id	Integer (FK)	Assigned vehicle

Foreign Keys:

• `vendor_id` → vendors.id (SET NULL on delete)

• `vehicle_id` → vehicles.id (SET NULL on delete)

Relationships:

• Belongs to **one vendor** (Many-to-One)

• Assigned to **one vehicle** (Many-to-One)

• Can have **many expenses** (One-to-Many)

Business Use:

• Driver roster management

• License compliance tracking

• Links driver to vendor and assigned vehicle

• Expense tracking per driver

8. Promoters Table

Purpose: Manages field promoters/anchors

Table Name: `promoters`

Column	Type	Description
id	Integer (PK)	Unique promoter identifier
name	String(255)	Promoter name
phone	String(20)	Contact number
email	String(255)	Email address
specialty	String(255)	Specialization area
language	String(100)	Preferred language

Relationships:

• Can have **many activities** (One-to-Many through promoter_activities)

Business Use:

• Promoter/anchor master data

• Links to daily field activities

• Tracks specializations and language preferences

9. Promoter Activities Table

Purpose: Tracks daily promoter field activities

Table Name: `promoter_activities`

Column	Type	Description
id	Integer (PK)	Unique activity identifier
promoter_id	Integer (FK)	Promoter performing activity
promoter_name	String(255)	Denormalized name for quick access
campaign_id	Integer (FK)	Related campaign
village_name	String(255)	Activity location
activity_date	Date	Date of activity
people_attended	Integer	Attendance count
activity_count	Integer	Number of activities performed
before_image	String(500)	Photo before activity
during_image	String(500)	Photo during activity
after_image	String(500)	Photo after activity
specialty	String(255)	Activity specialty
language	String(100)	Language used
remarks	Text	Additional notes
created_by_id	Integer (FK)	User who created record

Foreign Keys:

• `promoter_id` → promoters.id (CASCADE on delete)

• `campaign_id` → campaigns.id (CASCADE on delete)

• `created_by_id` → users.id

Relationships:

• Belongs to **one promoter** (Many-to-One)

• Belongs to **one campaign** (Many-to-One)

• Created by **one user** (Many-to-One)

Business Use:

• Daily activity logging for field promoters

• Photo documentation (before/during/after)

• Attendance and activity metrics

• Links activities to campaigns for reporting

10. Expenses Table

Purpose: Tracks operational expenses

Table Name: `expenses`

Column	Type	Description
id	Integer (PK)	Unique expense identifier
campaign_id	Integer (FK)	Related campaign
driver_id	Integer (FK)	Driver who incurred expense
expense_type	String(100)	Type (fuel, food, toll, etc.)
amount	Float	Expense amount
description	Text	Expense details
bill_url	String(500)	Bill document URL
bill_image	String(500)	Bill photo path
status	Enum	Status: pending, approved, rejected
submitted_date	Date	Submission date
approved_date	Date	Approval date

Foreign Keys:

• `campaign_id` → campaigns.id (CASCADE on delete)

• `driver_id` → drivers.id (SET NULL on delete)

Relationships:

• Belongs to **one campaign** (Many-to-One)

• Belongs to **one driver** (Many-to-One)

Business Use:

• Expense tracking per campaign

• Driver expense management

• Approval workflow (pending → approved/rejected)

• Bill documentation storage

11. Invoices Table

Purpose: Manages vendor invoices

Table Name: `invoices`

Column	Type	Description
id	Integer (PK)	Unique invoice identifier
invoice_number	String(100)	Invoice number (unique)
invoice_file	String(500)	Invoice document path
amount	Float	Invoice amount
invoice_date	Date	Invoice date
status	Enum	Status: pending, submitted, approved, rejected, paid
vendor_id	Integer (FK)	Billing vendor
campaign_id	Integer (FK)	Related campaign

Foreign Keys:

• `vendor_id` → vendors.id (CASCADE on delete)

• `campaign_id` → campaigns.id (SET NULL on delete)

Relationships:

• Belongs to **one vendor** (Many-to-One)

• Belongs to **one campaign** (Many-to-One)

• Can have **one payment** (One-to-One)

Business Use:

• Vendor billing management

• Links vendors to campaigns

• Invoice lifecycle tracking (submission → approval → payment)

• Foundation for payment processing

Special Note: Placeholder invoices (invoice_number starts with "PLACEHOLDER-") are auto-created when vendors are assigned to campaigns. These establish the campaign-vendor relationship and can be replaced with real invoices later.

12. Payments Table

Purpose: Tracks payments against invoices

Table Name: `payments`

Column	Type	Description
id	Integer (PK)	Unique payment identifier
amount	Float	Payment amount
payment_date	Date	Payment date
status	Enum	Status: pending, processing, completed, failed
payment_method	Enum	Method: bank_transfer, cheque, upi, cash, other
transaction_reference	String(255)	Transaction ID/reference
remarks	Text	Additional notes
invoice_id	Integer (FK)	Related invoice (unique)
vendor_id	Integer (FK)	Receiving vendor

Foreign Keys:

- `invoice_id` → invoices.id (CASCADE on delete, UNIQUE)
- `vendor_id` → vendors.id (CASCADE on delete)

Relationships:

- Belongs to **one invoice** (One-to-One)
- Belongs to **one vendor** (Many-to-One)

Business Use:

- Payment processing against invoices
- Each invoice can have only one payment (one-to-one)
- Tracks payment method and status
- Transaction reference for reconciliation

13. Reports Table

Purpose: Campaign execution reports

Table Name: `reports`

Column	Type	Description
id	Integer (PK)	Unique report identifier
campaign_id	Integer (FK)	Related campaign
report_date	Date	Report date
locations_covered	Text	Locations visited
km_travelled	Float	Distance traveled

Column	Type	Description
photos_url	Text	Photo gallery URLs
gps_data	Text	GPS coordinates
notes	Text	Additional notes

Foreign Keys:

• `campaign_id` → campaigns.id (CASCADE on delete)

Relationships:

• Belongs to **one campaign** (Many-to-One)

Business Use:

• Daily/periodic campaign execution reports

• Tracks locations and distance

• Photo documentation

• GPS tracking for verification

Table Relationships

Relationship Diagram (Text-Based)

```
■■■■■■■■■■■■
■ Clients ■
■■■■■■■■■■■■
■ (One-to-Many)
▼
■■■■■■■■■■■■
■ Projects ■
■■■■■■■■■■■■
■ (One-to-Many)
▼
■■■■■■■■■■■■■
■ Campaigns ■■■■■■■■■■■■■■■
■■■■■■■■■■■■■ ■
■ ■ ■ ■
■ ■ ■ (One-to-Many) ■
■ ■ ■■■■■■■■■■■■■■■■■■■
■ ■ ■ ■
■ ■ ▼ ■
■ ■ ■■■■■■■■■■■■ ■
■ ■ ■ Expenses ■ ■
■ ■ ■■■■■■■■■■■■ ■
■ ■ ■
■ ▼ ■
■ ■■■■■■■■■■■■ ■
■ ■ Reports ■ ■
■ ■■■■■■■■■■■■ ■
■ ■
▼ ■
■■■■■■■■■■■■ ■
■ Invoices ■■■■■■■■■■■■ ■
■■■■■■■■■■■■ ■ ■
■ ■ ■
■ (One-to-One) ■ ■
▼ ■ ■
■■■■■■■■■■■■■ ■ ■
■ Payments ■ ■ ■
■■■■■■■■■■■■■ ■ ■
■ ■
■■■■■■■■■■■■ ■ ■
■ Vendors ■■■■■■■■■■■■■■ ■
■■■■■■■■■■■■ ■
■ ■ ■
■ ■■■■■■■■■■■■■ ■
■ ■ ■
■ (Many) ■ (Many) ■
▼ ▼ ■
■■■■■■■■■■■■■ ■■■■■■■■■■■■ ■
■ Vehicles ■ ■ Drivers ■ ■
■■■■■■■■■■■■■ ■■■■■■■■■■■■ ■
■ ■
■ (FK) ■
■■■■■■■■■

■■■■■■■■■■■■■■
■ Promoters ■
■■■■■■■■■■■■■■
■ (One-to-Many)
▼
■■■■■■■■■■■■■■■■■■■■■■■■
■ PromoterActivities ■
■■■■■■■■■■■■■■■■■■■■■■■■
■
■■■■ Campaign (FK)

■■■■■■■■■■
■ Users ■■■■ Vendors (FK: vendor_id)
■■■■■■■■■■
```

Key Relationship Patterns

1. Client → Project → Campaign Hierarchy

- **Type:** Cascading One-to-Many
- **Flow:** Clients own Projects, Projects own Campaigns
- **Delete Behavior:** Deleting a client soft-deletes all projects and campaigns
- **Business Logic:** All campaigns must belong to a project and client

2. Campaign → Resources (Hub Pattern)

- **Type:** One-to-Many (Campaign as hub)
- **Connected Tables:**
- Expenses (campaign operational costs)
- Reports (campaign execution reports)
- Invoices (vendor billing)
- Promoter Activities (field activities)
- **Business Logic:** Campaign is the central operational unit

3. Vendor → Resources (Ownership Pattern)

- **Type:** One-to-Many (Vendor owns resources)
- **Connected Tables:**
- Vehicles (vendor fleet)
- Drivers (vendor employees)
- Invoices (vendor billing)
- Payments (vendor receivables)
- **Business Logic:** Vendors own and manage their resources

4. Campaign ↔ Vendor (Indirect Many-to-Many)

- **Type:** Many-to-Many through Invoices
- **Implementation:**
- Campaigns can have multiple invoices from different vendors
- Vendors can invoice multiple campaigns
- Invoice table acts as junction table
- **Business Logic:** Campaigns are assigned to vendors via invoice creation

5. Driver → Vehicle (Assignment)

- **Type:** Many-to-One (optional)
- **Implementation:** Driver has `vehicle_id` FK
- **Business Logic:** A driver can be assigned to one vehicle; vehicle can have multiple drivers over time

6. Invoice → Payment (One-to-One)

- **Type:** One-to-One (UNIQUE constraint on invoice_id)
- **Implementation:** Payment has unique `invoice_id` FK
- **Business Logic:** Each invoice has exactly one payment record

Role-Based Data Access

Access Control Matrix

Role	Tables Accessible	Scope	Permissions
Admin	All tables	All data	Full CRUD + Delete
Operations Manager	Campaigns, Projects, Reports, Expenses, Promoter Activities	All data	Full CRUD
Client Servicing	Clients, Projects, Campaigns	All data	Full CRUD

Role	Tables Accessible	Scope	Permissions
Accounts	Invoices, Payments, Expenses	All data	Full CRUD
Vendor	Vehicles, Drivers, Invoices, Payments	Own vendor_id only	Read + Limited Update
Client	Projects, Campaigns	Own client_id only	Read-only
Driver	Expenses	Own driver_id only	Create + Read
Promoter	Promoter Activities	Own promoter_id only	Create + Read

Data Scoping Mechanism

Vendor Users

```
-- Vendor users see only their own data
WHERE vendor_id = current_user.vendor_id AND is_active = 1
```

Example: When vendor user logs in:

• Dashboard shows only vehicles WHERE `vendor_id = user.vendor_id`

• Invoices filtered by `vendor_id = user.vendor_id`

• Payments filtered by `vendor_id = user.vendor_id`

Admin Users

```
-- Admin sees all active data
WHERE is_active = 1
```

Example: Admin dashboard shows:

• All campaigns across all clients

• All vendors and their resources

• All invoices and payments

Client Users

```
-- Client users see only their client's data
WHERE client_id = current_user.client_id AND is_active = 1
```

Example: Client portal shows:

• Projects WHERE `client_id = user.client_id`

• Campaigns under those projects

Permission Enforcement

Permissions are enforced at **three levels**:

Database Level: Foreign keys ensure referential integrity

API Level: FastAPI endpoints check user role before queries

Frontend Level: UI elements hidden based on user permissions

Example Flow:

```
User Request → JWT Token Validation → Role Check → Query Scoping → Data Return
```

Data Flow Architecture

High-Level Data Flow

```
■■■■■■■■■■■■■■■■
■ Frontend ■ (React)
■ (Browser) ■
■■■■■■■■■■■■■■■■
■ HTTP/REST API
▼
■■■■■■■■■■■■■■■■
■ FastAPI ■ (Python Backend)
■ Application ■
■■■■■■■■■■■■■■■■
■ SQLAlchemy ORM
▼
■■■■■■■■■■■■■■■■
■ MySQL ■
■ Database ■
■■■■■■■■■■■■■■■■
```

Create Operation Flow

Example: Creating a New Campaign

Frontend (React):

```
// User submits campaign form
POST /api/v1/campaigns
{
name: "Summer Campaign",
project_id: 5,
vendor_ids: [1, 2],
status: "planning"
}
```

API Layer (FastAPI):

```
# Validate JWT token
# Check user has CAMPAIGN_CREATE permission
# Validate project exists and is active
```

Service Layer:

```
# Create campaign record
# Create placeholder invoices for each vendor
# Commit transaction
```

Database:

```
INSERT INTO campaigns (...) VALUES (...);
INSERT INTO invoices (campaign_id, vendor_id, ...) VALUES (...);
COMMIT;
```

Response:

```
{
"id": 42,
"name": "Summer Campaign",
"status": "planning",
"vendor_names": ["Vendor A", "Vendor B"]
}
```

Read Operation Flow

Example: Vendor Dashboard

Frontend Request:

```
GET /api/v1/vendor-dashboard
```

API Layer:

```
# Extract vendor_id from JWT token
current_user.vendor_id = 10
```

Query with Scoping:

```
# Get campaigns assigned to this vendor
query = select(Campaign).join(Invoice).where(
Invoice.vendor_id == current_user.vendor_id,
Campaign.is_active == 1
)
```

Database Query:

```
SELECT campaigns.*
FROM campaigns
JOIN invoices ON campaigns.id = invoices.campaign_id
WHERE invoices.vendor_id = 10
AND campaigns.is_active = 1;
```

Response:

```
{
"campaigns": [...],
"invoices": [...],
"payments": [...]
}
```

Update Operation Flow

Example: Approving an Expense

Frontend:

```
PATCH /api/v1/expenses/123
{ status: "approved", approved_date: "2026-01-09" }
```

API Layer:

```
# Check user has EXPENSE_UPDATE permission
# Validate expense exists
```

Database:

```
UPDATE expenses
SET status = 'approved',
approved_date = '2026-01-09',
updated_at = NOW()
WHERE id = 123 AND is_active = 1;
```

Delete Operation Flow (Soft Delete)

Example: Deleting a Campaign

Frontend:

```
DELETE /api/v1/campaigns/42
```

Service Layer:

```
# Set is_active = False for campaign
# Set is_active = False for placeholder invoices
```

Database:

```
UPDATE campaigns
SET is_active = 0, updated_at = NOW()
WHERE id = 42;

UPDATE invoices
SET is_active = 0, updated_at = NOW()
WHERE campaign_id = 42
AND invoice_number LIKE 'PLACEHOLDER-%';
```

Note: Campaign is not deleted from database, only hidden from queries. This preserves audit trail and allows recovery if needed.

Schema Management

Alembic Migration System

What is Alembic?

Alembic is a database migration tool that tracks and applies changes to the database schema over time.

Why Use Alembic?

• **Version Control for Database:** Every schema change is recorded

• **Safe Deployments:** Changes are tested and applied systematically

• **Rollback Support:** Can revert to previous schema versions

• **Team Collaboration:** All developers share same schema state

Migration Workflow

```
Developer → Create Migration → Test Locally → Commit to Git → Deploy to Production
```

Example Migration

Scenario: Adding `vehicle_id` column to drivers table

Create Migration File:

```
alembic revision -m "add_driver_vehicle_id"
```

Migration Code:

```
def upgrade():
op.add_column('drivers',
sa.Column('vehicle_id', sa.Integer(), nullable=True))
op.create_foreign_key('fk_drivers_vehicle_id',
'drivers', 'vehicles', ['vehicle_id'], ['id'])

def downgrade():
op.drop_constraint('fk_drivers_vehicle_id', 'drivers')
op.drop_column('drivers', 'vehicle_id')
```

Apply Migration:

```
alembic upgrade head
```

Migration Safety

■ Production Data is Safe:

• Alembic only modifies schema (table structure), not data

• All migrations are tested in staging before production

• Downgrade option available for rollback

• Migrations are atomic (all or nothing)

■ Migrations Do NOT:

• Delete existing data

• Modify business logic

• Affect running application

Recent Migrations Applied

Date	Migration	Purpose
2026-01-08	\`add_driver_vehicle_id\`	Added vehicle assignment to drivers
2026-01-08	\`fix_payment_tables\`	Fixed payment table structure
2026-01-08	\`vendor_dashboard\`	Added vendor dashboard support
2026-01-06	\`add_promoter_language\`	Added language field to promoters

Business Process Flows

Process 1: Campaign Creation & Vendor Assignment

```
Client Request → Project Created → Campaign Created → Vendors Assigned
↓
Placeholder Invoices Created
↓
Vendor Dashboard Shows Campaign
```

Database Operations:

Admin selects client and creates project

• `INSERT INTO projects (client_id, name, ...)`

Admin creates campaign under project

• `INSERT INTO campaigns (project_id, name, ...)`

Admin selects vendors for campaign

• `INSERT INTO invoices (campaign_id, vendor_id, invoice_number='PLACEHOLDER-...')`

Vendor logs in and sees campaign

• `SELECT campaigns WHERE campaign_id IN (SELECT campaign_id FROM invoices WHERE vendor_id = X)`

Process 2: Expense Approval Workflow

```
Driver Submits Expense → Pending Status → Manager Reviews → Approved/Rejected
↓
Dashboard Stats Updated
```

Database Flow:

Driver submits expense

• `INSERT INTO expenses (driver_id, campaign_id, status='pending')`

Manager views pending expenses

- `SELECT * FROM expenses WHERE status='pending' AND is_active=1`

Manager approves expense

- `UPDATE expenses SET status='approved', approved_date=NOW()`

Dashboard recalculates totals

- `SELECT SUM(amount) FROM expenses WHERE status='approved' AND DATE(created_at)=TODAY()`

Process 3: Vendor Invoice & Payment Processing

```
Vendor Uploads Invoice → Accounts Approves → Payment Initiated → Payment Completed
↓
Vendor Dashboard Updated
```

Database Flow:

Vendor uploads invoice

- `INSERT INTO invoices (vendor_id, campaign_id, status='submitted')`

Accounts approves invoice

- `UPDATE invoices SET status='approved'`

Accounts creates payment

- `INSERT INTO payments (invoice_id, vendor_id, status='pending')`

Payment processed

- `UPDATE payments SET status='completed', payment_date=NOW()`
- `UPDATE invoices SET status='paid'`

Process 4: Daily Promoter Activity Logging

```
Promoter Visits Village → Takes Photos → Records Attendance → Submits Activity
↓
Campaign Report Generated
```

Database Flow:

Promoter logs activity

- `INSERT INTO promoter_activities (promoter_id, campaign_id, village_name, ...)`

Photos uploaded and paths stored

- `UPDATE promoter_activities SET before_image='/uploads/...'`

Manager views campaign activities

- `SELECT * FROM promoter_activities WHERE campaign_id=X ORDER BY activity_date DESC`

Key Performance & Scalability Notes

Indexing Strategy

Indexed Columns:

• All primary keys (`id`)

• Foreign keys for fast joins

• `email` in users (unique index for login)

• `vehicle_number` in vehicles (unique index)

• `invoice_number` in invoices (unique index)

• `activity_date` in promoter_activities (range queries)

• `vendor_id` in multiple tables (filtering)

Query Optimization

Soft Delete Filtering:

All queries automatically add `WHERE is_active = 1` to exclude deleted records.

Relationship Loading:

• Uses `joinedload` for eager loading related data

• Prevents N+1 query problems

• Example: Loading campaign with project and client in one query

Pagination:

• Large datasets use LIMIT/OFFSET for pagination

• Default limit: 1000 records per query

Data Retention

Soft Deletes:

• No data is permanently deleted

• Deleted records have `is_active = 0`

• Can be recovered if needed

• Preserved for audit and compliance

Audit Trail:

• `created_at`: When record was created

• `updated_at`: Last modification time

• These fields are automatically maintained by the system

Security Considerations

Data Protection

Password Storage:
• Passwords are hashed using bcrypt
• Original passwords never stored
• Password hints stored for admin reference

Role-Based Access:
• Every API endpoint checks user role
• Database queries scoped by role
• Vendors see only their data
• Clients see only their projects

Foreign Key Integrity:
• Database enforces referential integrity
• Cannot delete parent records with children
• Cascade deletes where appropriate

Soft Deletes:
• Accidental deletes are recoverable
• Audit trail maintained
• Data never truly lost

Compliance

• **Data Integrity:** Foreign keys prevent orphaned records
• **Audit Trail:** All records timestamped with creation/update times
• **Access Control:** Role-based permissions enforced
• **Backup & Recovery:** Soft deletes enable easy recovery

Appendix: Table Summary Reference

#	Table	Purpose	Key Relationships
1	users	User authentication & roles	→ vendors
2	clients	Client company data	→ projects
3	projects	Client projects	← clients, → campaigns
4	campaigns	Marketing campaigns	← projects, → expenses/reports/invoices
5	vendors	Vendor/supplier data	→ vehicles/drivers/invoices/payments
6	vehicles	Fleet vehicles	← vendors, → drivers
7	drivers	Driver information	← vendors, ← vehicles, → expenses
8	promoters	Field promoters	→ promoter_activities
9	promoter_activities	Daily field activities	← promoters, ← campaigns
10	expenses	Operational expenses	← campaigns, ← drivers
11	invoices	Vendor invoices	← vendors, ← campaigns, → payment
12	payments	Invoice payments	← invoices, ← vendors
13	reports	Campaign reports	← campaigns

Document Version History

Version	Date	Changes
1.0	2026-01-09	Initial comprehensive documentation

Contact & Support

For technical questions about this database documentation:

• **Technical Lead:** Development Team

• **Database Administrator:** Operations Team

End of Document