Database Setup - Instructions

Below is a **comprehensive design** for the **Transaction Database** and the **Reporting Database** in MySQL, along with **step-by-step instructions** for setting up the file storage server and documenting data flows (export, archive). These instructions assume a single recommended approach for each item—no alternate options.

1. Design Transaction DB Schema

The **Transaction DB** is where all real-time operations occur: campaigns, user activities, doctor engagement, etc. We will use MySQL 8.0 or later.

1.1 Database & Tables

1. **Database Creation**

```
-- On your MySQL server:
CREATE DATABASE inclinic_transaction;
USE inclinic_transaction;
```

2. **users Table**

Stores all registered users who log into the system (field reps, brand managers, admins). Doctors are not stored here because they do not log in.

```
CREATE TABLE users (
  user_id INT AUTO_INCREMENT PRIMARY KEY,
  email VARCHAR(255) NOT NULL UNIQUE,
  full_name VARCHAR(255) NOT NULL,
  role ENUM('FIELD_REP', 'BRAND_MANAGER', 'ADMIN') NOT NULL,
  status ENUM('ACTIVE', 'INACTIVE') DEFAULT 'ACTIVE',
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME ON UPDATE CURRENT_TIMESTAMP
);
```

- **Fields**:

- email: Linked to Google OAuth.
- role: Defines permissions (FIELD_REP, BRAND_MANAGER, ADMIN).
- status: If user is deactivated.
- **Indices**: Primary key on user_id, unique index on email.

3. **campaigns Table**

Represents each campaign. A single transaction server may host multiple campaigns, but each campaign is independent.

```
CREATE TABLE campaigns (
  campaign_id INT AUTO_INCREMENT PRIMARY KEY,
  campaign_name VARCHAR(255) NOT NULL,
  therapy_area VARCHAR(255) NOT NULL,
  start_date DATE NOT NULL,
  end_date DATE NOT NULL,
  status ENUM('ACTIVE', 'COMPLETED', 'ARCHIVED') DEFAULT 'ACTIVE',
  created_by INT NOT NULL,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME ON UPDATE CURRENT_TIMESTAMP,
  FOREIGN KEY (created_by) REFERENCES users(user_id)
    ON DELETE CASCADE ON UPDATE CASCADE
);
```

- **Fields**:
- therapy_area for informational grouping.
- status can be ACTIVE, COMPLETED, or ARCHIVED.
- **Associations**: created_by links to users.user_id.

4. **campaign_content Table**

Links each campaign with its educational content (PDFs or Vimeo videos).

```
CREATE TABLE campaign_content (
  content_id INT AUTO_INCREMENT PRIMARY KEY,
  campaign_id INT NOT NULL,
```

```
  content_type ENUM('PDF', 'VIDEO') NOT NULL,
  content_title VARCHAR(255) NOT NULL,
  file_path VARCHAR(500),
    -- For PDFs, e.g., "pdfs/<filename>.pdf" on the file storage server
  video_url VARCHAR(500),
    -- For Vimeo videos, e.g., "https://vimeo.com/<video_id>"
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME ON UPDATE CURRENT_TIMESTAMP,
  FOREIGN KEY (campaign_id) REFERENCES campaigns(campaign_id)
    ON DELETE CASCADE ON UPDATE CASCADE
);
```

- **Fields**:
- content_type: Distinguishes PDF vs. VIDEO.
- file_path: Relative path or URL to the stored PDF on the file storage server.
- video_url: Link to Vimeo video.

5. **doctor_shares Table**

Logs each time a field rep shares content with a doctor (identified by phone number).

```
CREATE TABLE doctor_shares (
  share_id INT AUTO_INCREMENT PRIMARY KEY,
  campaign_id INT NOT NULL,
  content_id INT NOT NULL,
  rep_id INT NOT NULL,
  doctor_phone VARCHAR(20) NOT NULL,
  share_timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (campaign_id) REFERENCES campaigns(campaign_id),
  FOREIGN KEY (content_id) REFERENCES campaign_content(content_id),
  FOREIGN KEY (rep_id) REFERENCES users(user_id)
);
```

- **Fields**:
- doctor_phone: used for logging the share event; doctors don’t log in.

- **Association**: Ties into campaigns, campaign_content, and users.

6. **pdf_events Table**

Tracks open, download, complete events for PDFs.

```
CREATE TABLE pdf_events (
  pdf_event_id INT AUTO_INCREMENT PRIMARY KEY,
  share_id INT NOT NULL,
  event_type ENUM('OPEN', 'DOWNLOAD', 'COMPLETE') NOT NULL,
  event_timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (share_id) REFERENCES doctor_shares(share_id)
    ON DELETE CASCADE
);
```

- **Fields**:
- event_type: Possible values are OPEN, DOWNLOAD, or COMPLETE.
- **Association**: Ties back to the share record.

7. **video_events Table**

Tracks play, watch time, and complete for Vimeo videos.

```
CREATE TABLE video_events (
  video_event_id INT AUTO_INCREMENT PRIMARY KEY,
  share_id INT NOT NULL,
  event_type ENUM('PLAY', 'WATCH_TIME', 'COMPLETE') NOT NULL,
  watch_seconds INT DEFAULT NULL,
    -- store # of seconds watched for WATCH_TIME events
  event_timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (share_id) REFERENCES doctor_shares(share_id)
    ON DELETE CASCADE
);
```

- **Fields**:
- event_type: PLAY, WATCH_TIME, or COMPLETE.
- watch_seconds: optional detail for partial watch progress.

8. **server_routing Table** *(Optional)*

If you're routing campaigns to different transaction servers, store references here:

```
CREATE TABLE server_routing (
  routing_id INT AUTO_INCREMENT PRIMARY KEY,
  campaign_id INT NOT NULL,
  transaction_server_name VARCHAR(100) NOT NULL,
  updated_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (campaign_id) REFERENCES campaigns(campaign_id)
);
```

Indexes & Optimization

- Ensure indexing on campaign_id, rep_id, share_id, etc., to speed up queries.

2. Design Reporting DB Schema

The **Reporting DB** mirrors essential fields from the Transaction DB but may also include aggregated or denormalized tables for faster analytics in Power BI. Use the same MySQL engine (or a separate MySQL instance).

1. **Database Creation**

```
CREATE DATABASE inclinic_reporting;
USE inclinic_reporting;
```

2. **Mirrored Tables (Core Data)**

- **campaigns_rep**: Minimal columns (campaign_id, name, therapy_area, start/end date, status).
- **campaign_content_rep**: Flattened references to PDF or video links.
- **doctor_shares_rep**: Same structure as transaction side, so you can see who shared what to which doctor.
- **pdf_events_rep** and **video_events_rep**: Mirrors of the raw events for analytics.

3. **Aggregated Tables / Materialized Views**

- **campaign_summary**: Summaries of PDF/video usage per campaign (e.g., total opens, completes, watchers, etc.).
- **rep_performance**: Field rep-level aggregates (total shares, successful completions).
- **pdf_vs_video_engagement**: Comparisons of PDF events to video events across campaigns.

Example aggregated schema for campaign_summary:

```
CREATE TABLE campaign_summary (
  summary_id INT AUTO_INCREMENT PRIMARY KEY,
  campaign_id INT NOT NULL,
  total_shares INT DEFAULT 0,
  pdf_opens INT DEFAULT 0,
  pdf_completes INT DEFAULT 0,
  pdf_downloads INT DEFAULT 0,
  video_plays INT DEFAULT 0,
  video_completes INT DEFAULT 0,
  last_updated DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (campaign_id) REFERENCES campaigns_rep(campaign_id)
);
```

- These aggregated tables can be updated each time the data export job runs or be updated by stored procedures.

3. Set Up File Storage Server (PDF Structure)

We'll store PDFs on a **dedicated EC2 instance** for simplicity:

1. **Provision an EC2 Instance** named pdf-file-server:
- Use Ubuntu 20.04.
- Restrict inbound traffic so only transaction servers can access it (port 22 for SSH from admin IP, port 80 or 443 for internal usage if needed).

2. **Directory Structure** (on the PDF File Server):

```
/var/www/pdf_files
  ├── campaigns
```

```
│   ├── campaign_<id>
│   │   ├── pdf_12345.pdf
│   │   ├── pdf_67890.pdf
│   │   └── ...
└── ...
```

- Each campaign can have its own subfolder or you can store PDFs by a unique ID.

3. **Install & Configure a Web Server** (e.g., Nginx) if you want direct HTTP access:

- Place PDF directory under /var/www/pdf_files.
- Only allow transaction servers to request these PDFs:
- You can use security group rules or Nginx IP allowlist.

4. **Permissions**:

- Ownership: www-data:www-data if using Nginx/Apache.
- Restrict write access to an administrative user or a script from the transaction server for uploading new PDFs.

5. **Reference in Transaction DB**:

- Each PDF is stored at a path like: http://pdf-file-server-internal/pdfs/campaign_<id>/pdf_<id>.pdf
- This path is saved in campaign_content.file_path.

4. Document Data Flows (Export, Archive)

4.1 Export Flow: Transaction → Reporting

1. **Scheduled Job**:

- On the transaction server, create a cron job (e.g., runs every 3 hours):

```
crontab -e
# Example: 0 */3 * * * /usr/bin/python /path/to/export_script.py
```

2. **Export Script** (e.g., export_script.py):

Export script details

- Connect to **Transaction DB** (inclinic_transaction).
- Query all new or updated records from campaigns, campaign_content, doctor_shares, pdf_events, video_events since last export.
- Insert them into **Reporting DB** (inclinic_reporting) mirrored tables (e.g., *_rep).
- Update or recalculate aggregated tables (e.g., campaign_summary).

3. **Mark Exported Records**:

- Optionally store an exported_at timestamp in transaction DB tables for each row.
- This way, subsequent runs only pick up records with exported_at IS NULL.

4.2 Archive & Cleanup Flow

1. **Archiving Completed Campaigns**:

- When a campaign's status is set to COMPLETED, schedule an **archive script** that runs after a grace period (e.g., 7 days).
- Moves older campaign data from transaction DB to an **Archive DB** or deletes them if fully exported.

2. **Archive Script** (e.g., archive_script.py):

- Identify campaigns with status = COMPLETED.
- Copy relevant rows from the transaction DB to an archive DB (optional if you want a dedicated place for historical data).
- Delete those rows from the transaction DB to keep it lean.

3. **Optimization**:

- Once data is removed, you can run OPTIMIZE TABLE commands on MySQL for large tables to reclaim space:

```
OPTIMIZE TABLE doctor_shares;
OPTIMIZE TABLE pdf_events;
OPTIMIZE TABLE video_events;
```

Summary

1. **Transaction DB**:

- Contains live records for campaigns, user roles, share logs, PDF/video events.
- Well-indexed schema for quick inserts/queries.

2. **Reporting DB**:

- Mirrored plus aggregated tables for Power BI.
- Populated by scheduled exports from the transaction DB.

3. **File Storage Server**:

- Dedicated EC2 instance hosting PDFs.
- Directory structured by campaign, accessible only to transaction servers.

4. **Data Flows**:

- **Export** from transaction DB to reporting DB on a schedule (via Python script).
- **Archive** completed campaigns to keep transaction DB small.
- **Optimization** commands used after large deletions.

Following these **step-by-step instructions** ensures that your engineers can **implement** the entire design—creating the schema, setting up the file storage, and documenting the data export and archival processes—without guesswork.