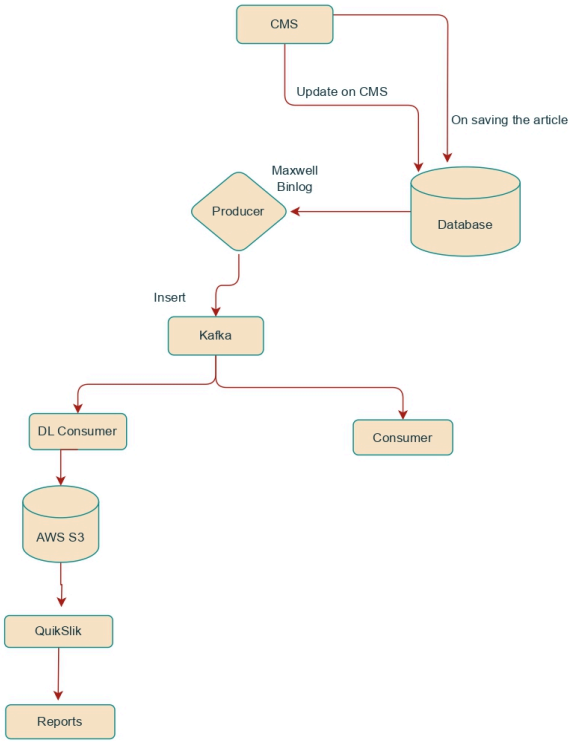
Subject: Proposal to Introduce a Data Lake for organisation.

Context: As our Django application and database continue to grow, we're beginning to see performance issues, especially around reporting, analytics and heavy DB queries. Currently, all structured and semi-structured data is stored directly in the database, including large text content. This setup increases read latency and slows down the application, particularly when generating complex or historical reports**.**

Problem:

- MySQL is being used both for transactional operations and reporting-heavy queries.
- Reports (weekly/monthly) are getting slower as data grows.
- Risk of overloading the primary database and degrading application performance.

Proposed Solution: Data Lake Architecture we propose introducing a Data Lake architecture to offload heavy, non-transactional workloads and optimize performance of the system. The goal is to store all historical and reporting-relevant data in a scalable, cost-effective storage layer, and query it efficiently without burdening the main application DB.



Milestone 1: Reporting Enablement via Selective Data Lake Integration

1. **Selective Table Dumping:**
   Begin by identifying and exporting only the critical MySQL tables required for analytics and reporting purposes.

2. **CDC-Based Synchronization:**
   Implement Change Data Capture (CDC) using MySQL binlogs to ensure real-time or near real-time sync of selected tables into the Data Lake.

3. **Reporting Layer Setup:**
   Build the initial reporting and visualization layer using Amazon QuickSight, querying data from the Data Lake.

Milestone 2: Full Data Lake Migration & Platform Optimization

1. **Complete Table Migration:**
   Expand the pipeline to include all application tables, offloading the entire database to the Data Lake for scalable storage and historical analysis.

2. **Live Database Load Reduction:**
   Minimize the operational overhead on the production MySQL database by shifting heavy read/reporting queries to the Data Lake.

3. **Data Access Abstraction:**
   Introduce a wrapper or abstraction layer in all dependent applications to route read and write operations intelligently — either from the Live DB or the Data Lake, based on data freshness and use case.

Task List for Milestone 1 -

🔹 **1. One-Time Historical Data Load**

- **Identify key reporting tables from the MySQL database.**
- **Perform necessary ETL (transform columns, add partition fields like `created_at`).**
- **Convert to Parquet format with proper partitioning.**
- **Upload to Amazon S3 in a structured data lake layout.**

🔹 **2. CDC Consumer Pipeline for Incremental Sync**

- **Create a Kafka consumer to read MySQL binlogs (CDC stream).**
- **Parse only the selected tables (schema-aware).**
- **Write CDC events in Parquet format with partitioning.**
- **Upload CDC files to the same S3 path used for historical data.**

🔹 **3. Data Lake Setup (Storage Layer)**

- **Configure and organize S3 buckets to store Parquet data partitioned by time.**
- **Define folder structure for each table.**
- **Set access policies for data lake services (Glue, Athena, QuickSight).**

🔹 **4. Metadata & Catalog Layer**

- **Define Glue Data Catalog tables for each selected dataset.**

- **Set up Glue Crawler or create DDL schemas manually to register partitions.**

🔹 **5. Analytics & Reporting Layer (QuickSight)**

- **Connect Amazon QuickSight to Athena/Glue tables.**
- **Build dashboards and visuals based on reporting use cases.**
- **Set access controls and refresh strategies for reports.**

🔹 **6. Testing & Validation**

- **Validate data consistency between MySQL and S3 (row counts, sampling).**
- **Test CDC replay correctness and schema evolution handling.**
- **Run performance benchmarks on Athena + QuickSight queries.**