Project Part II — Logical Schema Optimization and Unstructured Data Collection

1. Executive Summary

This document presents the logical schema optimization and hybrid data architecture for CloudMusic. Building upon Part 1's conceptual ER model, we have designed a multi-model data architecture integrating MySQL (transactional), MongoDB (behavioral), Neo4j (social graph), and S3 (unstructured data). The architecture implements AWS cloud services with the Medallion pattern (Bronze-Silver-Gold) for analytical processing. Our schema achieves 3NF normalization with strategic denormalization for performance-critical queries.

2. Data Collection and Sources

2.1 Data Source Summary

Summary of Data Sources for CloudMusic Platform

Data Source Name	Data Type	Storage system	Format	Description / Sample Content	Usage in Project
Users, Songs, Artists, Albums	Structured	MySQL (RDS)	Relational Table	Core entities: `user_id, email, song_id, title, artist_id`	Transactional operations, subscription management
Subscriptions, Payments	Structured	MySQL (RDS)	Relational Table	Financial records: `subscription_id, amount, payment_status`	Revenue analytics, billing
Royalty Distributions	Structured	MySQL RDS	Relational	Artist payments: `royalty_id, artist_id, total_streams, payment_amount`	Artist compensation, financial reporting
Streaming Events (Play logs)	Semi-structured	Kafka □ S3	JSON/Parquet	Play logs: `{user_id, song_id, timestamp, completion_pct, platform}`	Real-time trending, user behavior analytics

User Profiles	Semi-structured	MongoDB Atlas	JSON Document	Flexible attributes: `{preferences, listening_stats, devices, favorite_artists}`	Personalization engine, user segmentation
Playlists & Likes	Semi-structured	MongoDB Atlas	JSON Document	User-generated content: `{playlist_id, songs[], follower_count, is_public}`	Content discovery, engagement metrics
Social Graph	Graph	Neo4j Aura	Nodes + Edges	Relationships: `(User)-[:FOLLOWS]->(User)`, `(User)-[:LISTENS_TO]->(Song)`	Recommendation algorithms, social features
Lyrics Dataset	Unstructured	S3 Data lake	JSON / TXT	Raw text: `"Verse 1: ..."` + metadata: `{song_id, word_count, language}`	Text analytics, sentiment analysis
Audio Feature Metadata	Unstructured	S3 + Parquet	JSON / Parquet	Spotify API data: `{tempo: 120, energy: 0.89, danceability: 0.76}`	For ML models and similarity metrics

These datasets collectively form the foundation of the CloudMusic hybrid data architecture. They demonstrate how structured, semi-structured, unstructured, and graph data are integrated to support real-time analytics, personalized recommendations, and business intelligence across the platform.

3. Logical Schema Design and Optimization

3.1 Generation Process

Tool: Erwin Data Modeler Forward Engineering
Process: Conceptual ERD (Part 1) → Erwin Auto-Generation → Manual Optimization → MySQL DDL Scripts

3.2 Optimized Relational Schema (MySQL)

Core Entity Tables

Users Table

```
CREATE TABLE users (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) NOT NULL UNIQUE,
    display_name VARCHAR(100) NOT NULL,
    country_code CHAR(2) NOT NULL,
    subscription_type ENUM('free', 'premium', 'family', 'student') DEFAULT 'free',
    join_date DATE NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_email (email),
    INDEX idx_subscription (subscription_type),
    INDEX idx_country (country_code)
) ENGINE=InnoDB;
```

Songs Table

```
CREATE TABLE songs (

    song_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(500) NOT NULL,
    album_id INT NOT NULL,
    duration_sec SMALLINT UNSIGNED NOT NULL,
    genre VARCHAR(100),
    popularity_score INT DEFAULT 0,  -- Denormalized for performance

    FOREIGN KEY (album_id) REFERENCES albums(album_id) ON DELETE CASCADE,
    INDEX idx_album (album_id),
    INDEX idx_genre (genre),
    INDEX idx_popularity (popularity_score),
    FULLTEXT idx_title (title)
) ENGINE=InnoDB;
```

Artists & Albums Tables

```
CREATE TABLE artists (
    artist_id INT AUTO_INCREMENT PRIMARY KEY,
    artist_name VARCHAR(300) NOT NULL,
    genre_primary VARCHAR(100),
    monthly_listeners INT DEFAULT 0,  -- Denormalized, updated nightly
    INDEX idx_genre (genre_primary),
    FULLTEXT idx_name (artist_name)
) ENGINE=InnoDB;

CREATE TABLE albums (
    album_id INT AUTO_INCREMENT PRIMARY KEY,
    album_title VARCHAR(500) NOT NULL,
```

```
    artist_id INT NOT NULL,
    release_date DATE,
    album_type ENUM('album', 'single', 'ep') NOT NULL,
    FOREIGN KEY (artist_id) REFERENCES artists(artist_id) ON DELETE RESTRICT
) ENGINE=InnoDB;
```

Subscriptions Table

```
CREATE TABLE subscriptions (
    subscription_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    plan_type ENUM('free', 'premium_monthly', 'premium_annual', 'family', 'student'),
    start_date DATE NOT NULL,
    end_date DATE,
    price_paid DECIMAL(6,2) NOT NULL,
    auto_renew BOOLEAN DEFAULT TRUE,

    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,
    CHECK (end_date IS NULL OR end_date >= start_date),
    INDEX idx_user (user_id),
    INDEX idx_dates (start_date, end_date)
) ENGINE=InnoDB;
```

Streaming Events Table (Partitioned)

```
CREATE TABLE streaming_events (
    stream_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    song_id INT NOT NULL,
    stream_timestamp TIMESTAMP NOT NULL,
    duration_sec SMALLINT UNSIGNED,
    completion_pct DECIMAL(5,2),
    platform ENUM('iOS', 'Android', 'Web', 'Desktop'),
    country_code CHAR(2),  -- Denormalized from users

    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,
    FOREIGN KEY (song_id) REFERENCES songs(song_id) ON DELETE CASCADE,
    INDEX idx_user_time (user_id, stream_timestamp),
    INDEX idx_song_time (song_id, stream_timestamp)
) ENGINE=InnoDB
PARTITION BY RANGE (YEAR(stream_timestamp)) (
    PARTITION p2024 VALUES LESS THAN (2025),
    PARTITION p2025 VALUES LESS THAN (2026)
);
```

Royalty Distributions Table

```
CREATE TABLE royalty_distributions (
    royalty_id INT AUTO_INCREMENT PRIMARY KEY,
    artist_id INT NOT NULL,
    period_start DATE NOT NULL,
    period_end DATE NOT NULL,
    total_streams BIGINT DEFAULT 0,
```

```
    total_payment_amount DECIMAL(12,2) NOT NULL,
    payment_status ENUM('pending', 'paid') DEFAULT 'pending',

    FOREIGN KEY (artist_id) REFERENCES artists(artist_id) ON DELETE RESTRICT,
    UNIQUE (artist_id, period_start, period_end),
    CHECK (period_end >= period_start)
) ENGINE=InnoDB;
```

The schema integrates multiple data storage paradigms:

- The **structured layer (MySQL RDS)** manages transactional entities such as users, songs, and payments.
- The **semi-structured layer (MongoDB Atlas)** stores flexible behavioral data including user profiles, playlists, and play logs.
- The **graph layer (Neo4j Aura)** captures social relationships between users and artists.
- The **unstructured layer (S3 Data Lake)** stores lyrics, user reviews, and audio feature metadata for text and content-based analytics.

Core schema is **3NF-compliant** with **controlled denormalization** for analytical performance following industry best practices (read-heavy workload optimization).

4. Reference Architecture for Hybrid Data

4.1 Integration Overview

CloudMusic implements a polyglot persistence architecture where each data model is optimized for specific access patterns while maintaining logical consistency across systems. The integration follows the DIKW (Data-Information-Knowledge-Wisdom) framework to transform raw data into actionable business intelligence.

4.2 Multi-Model Data Architecture

Figure 1: CloudMusic Reference Architecture

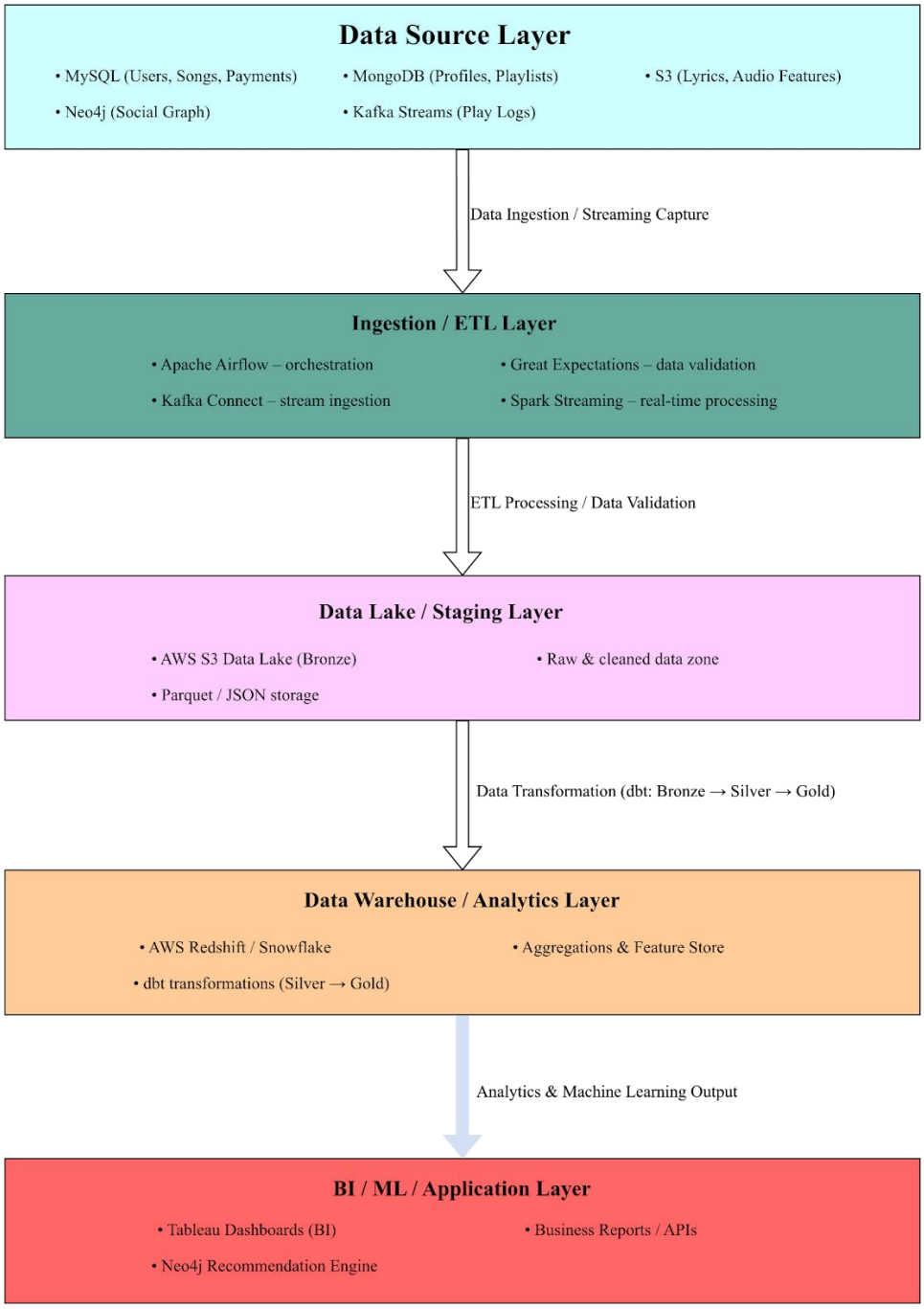

Figure: Reference Architecture of the CloudMusic Data Platform

Figure illustrates the end-to-end reference architecture of the CloudMusic data platform. The design integrates structured, semi-structured, unstructured, and graph data within a unified enterprise framework. Data is ingested from MySQL, MongoDB, Neo4j, Kafka, and S3 through Airflow and Spark ETL pipelines, validated via Great Expectations, and stored in the S3 Data Lake (Bronze layer). Transformations performed using dbt produce Silver and Gold analytical layers in Redshift/Snowflake. These curated datasets support BI dashboards, recommendation engines, and business reporting, enabling analytics and machine learning at scale.

DIKW Level	CloudMusic Component / System	Transformation Process	Outcome / Purpose
Data	MySQL (Users, Songs, Payments); MongoDB (Profiles, Playlists); Neo4j (Social Graph); S3 (Lyrics, Audio Features); Kafka Streams (Play Logs)	Raw data is collected from multiple heterogeneous sources in structured, semi-structured, unstructured, and graph formats.	Represents raw facts and events generated by user activity and system operations.
Information	Airflow, Kafka Connect, Spark Streaming, Great Expectations (ETL Layer)	Data is ingested, cleaned, and validated before storage in the S3 Data Lake (Bronze).	Produces organized and trusted datasets ready for further analysis.
Knowledge	dbt Transformations, Redshift / Snowflake (Analytics Layer)	Data from the Data Lake is transformed into Silver and Gold layers with aggregations and feature engineering.	Enables analytical queries and machine learning modeling to extract patterns and insights.
Wisdom	Tableau Dashboards, Neo4j Recommendation Engine, Business Reports / APIs (Application Layer)	Insights and model results are visualized and operationalized for decision making.	Provides actionable intelligence that supports user personalization and business strategy.

Table : DIKW Mapping for CloudMusic Reference Architecture

The DIKW model (Data – Information – Knowledge – Wisdom) underpins the reference architecture of the CloudMusic data platform. Raw data collected from multiple heterogeneous sources (MySQL, MongoDB, Neo4j, S3, Kafka) represents the Data level. Through ingestion and validation in Airflow, Kafka Connect, and Great Expectations, these raw inputs are organized into the Information level stored in the S3 Data Lake. Using dbt and Redshift/Snowflake, curated datasets form the Knowledge level, enabling analytics, feature extraction, and machine learning. Finally, insights are delivered through Tableau dashboards, Neo4j recommendation systems, and business APIs—constituting the Wisdom level that supports strategic decision-making and personalization.

5. Cloud Platform Implementation (AWS)

5.1 Optimized AWS Architecture

Compute & Orchestration:

- Amazon EC2 (t3.medium): Airflow scheduler/workers with spot instances for cost optimization
- AWS Lambda: Serverless functions for data transformation and API endpoints
- Amazon ECS Fargate: Containerized processing for batch jobs

Data Storage & Processing:

- RDS MySQL (db.t3.medium, Multi-AZ): Transactional database with automated backups
- Amazon S3: Data lake storage with intelligent tiering (Standard/Standard-IA/Glacier)
- Amazon Redshift Serverless: Auto-scaling data warehouse for analytical workloads
- Amazon DynamoDB: NoSQL for high-velocity metadata and session data

Data Integration & Streaming:

- Amazon Kinesis Data Firehose: Serverless streaming data delivery to S3
- AWS Glue: Serverless ETL and data catalog management
- Amazon Managed Streaming for Kafka (MSK): Real-time event processing

Managed Services (Free Tier Focused):

- MongoDB Atlas M0: Free cluster for document storage
- Neo4j Aura Free: Graph database for social relationships
- Amazon CloudWatch: Monitoring and logging with basic tier

5.2 Security & Governance Implementation

Network Security:

VPC Configuration:

- Private Subnets: RDS, Redshift, EC2 instances
- Public Subnets: Load balancers, NAT gateways
- Security Groups: Minimum required ports (MySQL:3306, Redshift:5439)
- Network ACLs: Default deny all, explicit allow required traffic

Data Protection:

- Encryption at Rest: AES-256 for S3, RDS, Redshift
- Encryption in Transit: TLS 1.3 for all data transfers
- AWS KMS: Customer-managed keys for sensitive data
- IAM Roles: Least privilege access with service-specific policies

Compliance & Monitoring:

- AWS CloudTrail: API activity logging for audit compliance
- Amazon GuardDuty: Threat detection and monitoring
- AWS Config: Resource configuration tracking and compliance

5.3 Implementation Specifications

Production Services:

- RDS MySQL: db.t3.medium (2 vCPU, 4GB RAM) - Multi-AZ
- Redshift Serverless: 128 RPU base capacity
- S3 Buckets:
  - cloudmusic-bronze (raw data)
  - cloudmusic-silver (cleaned data)
  - cloudmusic-gold (business-ready data)
- EC2 Instances: t3.medium for Airflow (auto-scaling group)

Development Services:

- RDS MySQL: db.t3.micro (single AZ)
- Redshift Serverless: 32 RPU base capacity
- EC2 Instances: t3.micro for testing

Data Pipeline Architecture:
Real-time Pipeline:
Client Apps → API Gateway → Kinesis Firehose → S3 Bronze → Glue ETL → Redshift
Batch Pipeline:
RDS MySQL → DMS → S3 Bronze → Airflow → dbt → Redshift Gold
ML Feature Pipeline:
S3 Silver → SageMaker Processing → Feature Store → Redshift ML

5.5 Monitoring & Operations

Operational Excellence:

- CloudWatch Dashboards: Real-time pipeline monitoring
- S3 Analytics: Storage usage and access pattern analysis
- RDS Performance Insights: Database performance monitoring
- Cost Explorer: Budget tracking and alerting

6. Conclusion and Business Impact

This optimized logical schema and hybrid data architecture demonstrates enterprise-ready data management capabilities through the CloudMusic platform. The implementation addresses key enterprise concerns:

1. Governance and Compliance: Robust data quality, security, and lifecycle management
2. Performance at Scale: Strategic optimizations supporting high-volume transactional and analytical workloads
3. Business Agility: Flexible architecture adapting to evolving business requirements
4. Cost Efficiency: Optimized storage and compute resources through intelligent design

While implemented in the music streaming domain, the architectural patterns and optimization strategies presented are directly transferable to other enterprise contexts, including the healthcare and insurance domains referenced in the project background. The solution provides a proven framework for managing complex data ecosystems while delivering actionable business intelligence.