Subscription Tier Optimization Engine

Database Management Systems Project Report

Project Information

Project Title: Subscription Tier Optimization Engine

Course: Database Management Systems (DBMS)

Database: subsys_db (MySQL)

Application Type: Web-based Analytics Dashboard

Programming Language: Python 3.8+

Framework: Streamlit

1. Executive Summary

Project Title: Subscription Tier Optimization Engine

Course: Database Management Systems (DBMS)

Database: subsys_db (MySQL)

Application Type: Web-based Analytics Dashboard (Streamlit + MySQL)

This project implements a small analytics platform for a subscription-based product. It tracks feature usage, user activity, and subscription tiers, then uses SQL queries and a Streamlit dashboard to answer questions such as:

- Which features are most engaging?
- Which market segments and tiers show the highest activity?
- Which users are good candidates for an upgrade?

The system demonstrates core DBMS concepts: relational design, foreign keys, triggers, stored procedures, user-defined functions, and analytical SQL queries.

2. Problem Statement and Objectives

Subscription products often struggle with:

- Identifying which features actually drive user engagement
- Understanding how engagement differs across tiers and segments
- Finding high-engagement users who are ready for upgrades

Main objective:

Design a relational database and a simple dashboard that together provide basic but useful analytics for subscription optimization.

Specific objectives:

1. Model users, tiers, features, segments, and activities in a normalized MySQL schema.
2. Populate the database with realistic sample data.
3. Implement SQL queries for engagement, retention, and segmentation.
4. Use Streamlit to visualise these results in an interactive way.

---

3. System Design (Short)

3.1 Main Entities

- `User` – subscriber details and current tier/segment
- `Tier` – subscription plans (Free, Silver, Gold, etc.)
- `Feature` – platform features (Analytics Dashboard, API Access, etc.)
- `User_Activity` – logs of feature usage with duration and timestamp
- `Market_Segment` – user segments (Students, Tech Enthusiasts, etc.)
- `System_Log` – audit entries generated by a trigger

Relationships use foreign keys with `ON DELETE CASCADE` or `ON DELETE SET NULL` to keep data consistent.

3.2 Advanced Database Elements

- **Trigger**: `trg_user_insert` – writes to `System_Log` when a new user is inserted.
- **Stored procedure**: `recommend_upgrade()` – returns users whose average activity duration is above a threshold (e.g., > 50).
- **Functions**: e.g. `avg_session_duration(uid)` – computes a user's average session duration.

---

4. Key SQL Analytics

Only the most important queries are listed here. All of them are implemented in `subsys_project_full(1).sql` and used by `frontend.py`.

4.1 Tier-based Engagement

Average engagement per tier and associated pricing.

```
SELECT
  t.tier_name AS tier,
  ROUND(AVG(a.duration), 2) AS avg_engagement,
  t.price AS tier_price,
  COUNT(DISTINCT u.user_id) AS user_count
FROM Tier t
JOIN User u ON t.tier_id = u.current_tier_id
JOIN User_Activity a ON a.user_id = u.user_id
GROUP BY t.tier_id
ORDER BY avg_engagement DESC;
```

4.2 Top Features

Top 5 features by total time spent.

```
SELECT
  f.feature_name,
  SUM(a.duration) AS total_time,
  COUNT(a.activity_id) AS total_sessions
FROM Feature f
JOIN User_Activity a ON f.feature_id = a.feature_id
```

```
GROUP BY f.feature_id
ORDER BY total_time DESC
LIMIT 5;
```

4.3 Segment Ranking (Window Function)

```
SELECT
  m.segment_name,
  ROUND(AVG(a.duration), 2) AS avg_engagement,
  RANK() OVER (ORDER BY AVG(a.duration) DESC) AS rank_seg
FROM Market_Segment m
JOIN User u ON m.segment_id = u.current_segment_id
JOIN User_Activity a ON a.user_id = u.user_id
GROUP BY m.segment_id;
```

4.4 Activity Segmentation

Classifies users as Highly Active, Moderately Active, or Low Activity.

```
SELECT
  activity_level,
  COUNT(DISTINCT user_id) AS total_users
FROM (
  SELECT
    user_id,
    CASE
      WHEN SUM(duration) > 100 THEN 'Highly Active'
      WHEN SUM(duration) BETWEEN 50 AND 100 THEN 'Moderately Active'
      ELSE 'Low Activity'
    END AS activity_level
  FROM User_Activity
  GROUP BY user_id
) AS sub
GROUP BY activity_level
ORDER BY total_users DESC;
```

4.5 Above-Average Features (Correlated Subquery)

```
SELECT
  f.feature_name,
  ROUND(AVG(a.duration), 2) AS avg_duration
FROM Feature f
JOIN User_Activity a ON f.feature_id = a.feature_id
GROUP BY f.feature_id, f.feature_name
HAVING AVG(a.duration) > (SELECT AVG(duration) FROM User_Activity);
```

---

5. Streamlit Dashboard (Summary)

The frontend (`frontend.py`) is a single Streamlit app with a sidebar menu. Main sections:

1. **Feature Insights** – bar chart of top features by total duration.
2. **User Insights** – engagement by market segment (with ranking).
3. **Upgrade Recommendations** – uses `CALL recommend_upgrade()`.
4. **Correlation & Retention** – tier price vs engagement, and feature retention rates.
5. **Activity Segmentation** – pie chart of activity levels.
6. **System Logs** – latest rows from `System_Log` (trigger output).
7. **Session Analytics** – calls `avg_session_duration(uid)` for a given user.
8. **Above-Average Features** – features with engagement above the global average.

Each section reuses the same pattern: open a MySQL connection, run a query, load into a Pandas DataFrame, and display with Plotly plus a data table.

---

6. Testing and Results (Short)

- Inserted sample users, tiers, features, and activities from `subsys_project_full(1).sql`.
- Verified that foreign keys and `ON DELETE` actions behave as expected.

- Confirmed that the trigger writes log rows on `User` insert.
- Ran all main analytical queries and checked that results match the sample data.
- Manually tested each dashboard tab to ensure charts render and no runtime errors occur.

From the sample dataset we can see, for example:

- Certain features (like Data Export and Cloud Backup) generate higher total engagement.
- Some segments and tiers show clearly higher activity, indicating upgrade and targeting opportunities.

---

7. Conclusion

The Subscription Tier Optimization Engine project shows how a small but well-structured MySQL schema, combined with a simple Streamlit frontend, can support basic subscription analytics:

- The database design enforces consistency through foreign keys and constraints.
- Triggers, functions, and procedures move part of the business logic into the database.
- Analytical queries reveal patterns in feature usage, segments, and tiers.
- The dashboard turns these queries into readable charts for quick interpretation.

Overall, the project meets its goal of demonstrating key DBMS concepts in a practical, subscription-focused use case.