

# Big Data Management Analytics Project 2: Google Ad Campaign Management System Project Report

# **Submitted to:**

Prof. Amarnath Mitra
Chair, PGDM-BDA
FORE School of Management, New Delhi

# **Submitted by:**

Isha Gupta – 055014 Sneha Gupta – 055047

**Group** – 18

Section - K

#### **PROJECT INFORMATION**

## **Google Ad Campaign Management System**

The project is about the **Google Ad Campaign Management System**, which helps businesses efficiently manage and optimize their advertising campaigns. This system is designed to store and process key data related to advertisements, targeting, performance tracking, and budget allocation.

For this purpose, we are using a **relational database management system (RDBMS)** to structure and manage campaign-related data efficiently. The system consists of multiple tables, each serving a specific role in managing ad campaigns.

#### **Testing and Optimization**

To ensure the database performs efficiently under heavy loads, **stress testing** was conducted by inserting large volumes of data into the tables. This helped identify potential performance bottlenecks and improve query execution speed.

A total of 6 stress testing queries were performed on both normalized and non-normalized database tables for comparison.

#### **DESCRIPTION OF DATA**

The **Google Ad Campaign Management System** stores structured data related to advertising campaigns, ad groups, keywords, placements, audiences, budgets, and performance metrics. The data is categorized into multiple tables to ensure efficient storage, retrieval, and analysis. Below is a description of the key datasets used in the system:

- 1. Campaign Data Stores details about ad campaigns, including their objectives and duration.
  - Campaign ID (VARCHAR) Unique identifier for each campaign
  - Campaign Name (VARCHAR) Name assigned to the campaign
  - Start Date & End Date (DATE) Duration of the campaign
  - **Objective** (VARCHAR) Goal of the campaign (e.g., brand awareness, conversions)
- 2. Ad Group Data Stores information about different ad groups, which are collections of related ads.
  - Ad Group ID (VARCHAR) Unique identifier for each ad group
  - Ad Group Name (VARCHAR) Name of the ad group
  - Associated Campaign ID (VARCHAR) Links ad groups to their respective campaigns
- 3. Keyword Data Contains keywords used for targeting ads based on user search queries.
  - **Keyword ID** (VARCHAR) Unique identifier for each keyword
  - **Keyword Text** (VARCHAR) The actual keyword used for targeting
  - Match Type (VARCHAR) Type of keyword matching (e.g., broad, exact, phrase)
- **4.** Ad Data Stores details of individual ads, including their content and associated ad groups.
  - Ad ID (VARCHAR) Unique identifier for each ad
  - Ad Group ID (VARCHAR) Links ads to their respective ad groups

- Ad Content (TEXT) Text, image, or video content of the ad
- Ad Type (VARCHAR) Format of the ad (e.g., text, display, video)

**5.** Ad Placement Data - Contains information about different platforms and placements where ads are displayed.

- Placement ID (VARCHAR) Unique identifier for each ad placement
- Platform (VARCHAR) The platform where the ad is displayed (e.g., Google Search, YouTube)
- Ad Format (VARCHAR) Specifies the type of placement (e.g., sidebar, in-stream video)
- **6. Performance Metrics Data -** Tracks key performance indicators such as impressions, clicks, conversions, and CTR (Click-Through Rate).
  - Performance ID (VARCHAR) Unique identifier for performance records
  - Impressions (INT) Number of times the ad was displayed
  - Clicks (INT) Number of clicks received by the ad
  - Conversions (INT) Number of successful actions (e.g., purchases, sign-ups)
  - Click-Through Rate (CTR) (DECIMAL) Percentage of impressions that resulted in clicks
- **7.** Budget Allocation Data Stores budget-related information, including total allocated budget and spent budget for each campaign.
  - Budget ID (VARCHAR) Unique identifier for each budget record
  - Total Budget (DECIMAL) The allocated budget for a campaign
  - Spent Budget (DECIMAL) The amount spent so far in the campaign
- 8. Bridge Tables for Many-to-Many Relationships
  - AdGroupKeyword1 Links ad groups and keywords
  - CampaignAudience1 Links campaigns with audience segments
  - AdPlacementTracking1 Links ads with their placements

## **PROJECT OBJECTIVES | PROBLEM STATEMENTS**

- 1. <u>Efficient Data Management</u> Structuring campaign-related data to avoid redundancy and ensure faster queries.
  - (**PROBLEM**: The presence of duplicated data in campaign tables leads to excessive storage consumption and maintenance complexity.)
- 2. <u>Normalization for Optimization</u> Applying First Normal Form (1NF) and Second Normal Form (2NF) to improve data consistency and integrity.
  - (**PROBLEM**: Inefficient schema designs slow down ad campaign insights retrieval, impacting decision-making.)
- 3. <u>Performance Evaluation</u> Stress testing the database before and after normalization to compare execution efficiency.

4. <u>Scalability Analysis</u> – Identifying bottlenecks and recommending improvements for handling large-scale advertising data.

**(PROBLEM**: Handling high volumes of transactions without optimized indexing and query execution causes system lag.)

# **ANALYSIS OF DATA**

#### Stress testing analysis:

- To assess database performance, we conducted stress testing on two versions of our schema:
- 1. **Before Normalization** (Non-NF Schema)
- 2. After Normalization (1NF and 2NF Applied)
- For each version, we executed batch insertions, transaction simulations, and query performance measurements. Key stress testing parameters included:
- Query execution time for SELECT, INSERT, UPDATE operations
- System performance in terms of CPU, RAM, and disk usage
- o **Data integrity maintenance** under load
- Concurrency handling and transaction efficiency

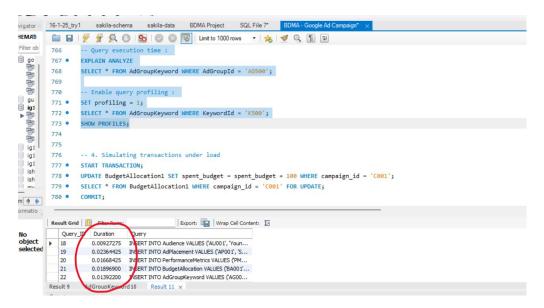
#### Comparison of Stress Testing Results:

Before Normalization (Non-NF Schema)	After Normalization (1NF & 2NF Schema)
Query execution times were relatively higher	Query execution was significantly faster (as
due to redundancy and data duplication.	seen in the profiling results).
High <b>disk space consumption</b> due to redundant	Lower disk space usage, making the database
data storage.	more scalable for large datasets.
Transactions under load led to deadlocks and	Update operations became efficient,
slower response times.	eliminating redundancy and reducing locking issues.
EXAMPLE : INSERT operations took an average	EXAMPLE : INSERT operations reduced to
of <b>0.0189 sec</b> per row.	<b>0.0113 sec</b> per row.
	Query execution time (SELECT
	AdGroupKeyword WHERE
	ad_group_id='AG500') improved significantly
	post-normalization.

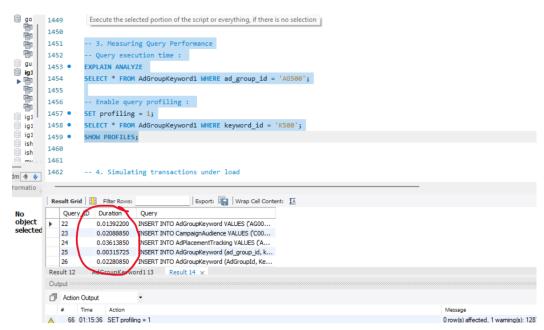
#### **OBSERVATIONS & FINDINGS**

1. **Normalization drastically improves efficiency:** Eliminating redundancy and structuring data logically optimizes both query performance and storage efficiency. (*in few cases*)

#### **Before normalization:**

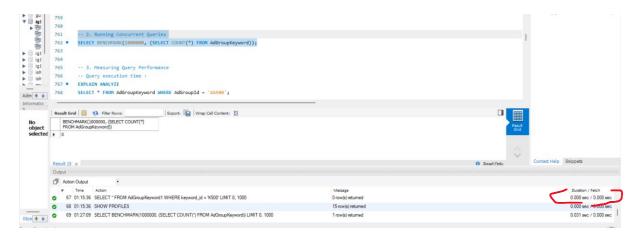


#### After normalization:

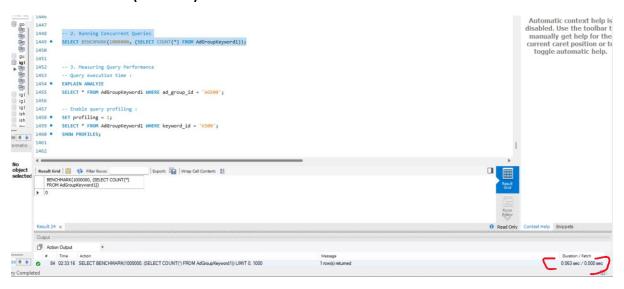


2. Not necessary that deadlocks always get reduced and concurrency handling gets better after normalization, for example :

# Before normalization (0.031 sec)

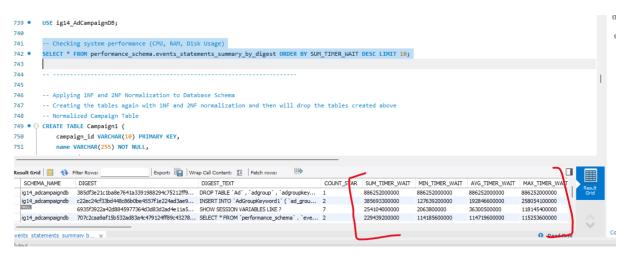


# After normalization (0.063 sec)



3. System's performance improved a lot after normalization.

#### **Before normalization:**



#### After normalization:



#### **MANAGERIAL INSIGHTS & RECOMMENDATIONS**

- 1. Adopt a fully normalized schema (at least up to 2NF or 3NF):
  - o Reduces redundancy and ensures efficient data storage.
  - Speeds up transactional operations and improves consistency.
- 2. Optimize indexing strategies:
  - Use primary keys and foreign keys efficiently.
  - o Implement composite indexes for frequently used joins.

# 3. Monitor system performance regularly:

- Use query profiling tools to measure execution times.
- Keep track of CPU, RAM, and disk usage to prevent bottlenecks.

# 4. Improve concurrency handling mechanisms:

- o Implement proper transaction isolation levels to avoid deadlocks.
- o Use optimized queries to prevent long-running locks.

# 5. Implement batch processing for large insertions:

 Large datasets should be inserted in chunks to optimize memory use and execution time.

# 6. **Optimize query execution plans:**

- o Avoid full table scans by implementing efficient indexing.
- o Use EXPLAIN ANALYZE to debug slow queries and optimize them.