FORE SCHOOL OF MANAGEMENT

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. **Efficient Data Management –** 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. **Normalization for Optimization** – 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. **Performance Evaluation** – Stress testing the database before and after normalization to compare execution efficiency.

4. **Scalability Analysis** – 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
- ○ **Data integrity maintenance** under load
- ○ **Concurrency handling and transaction efficiency**

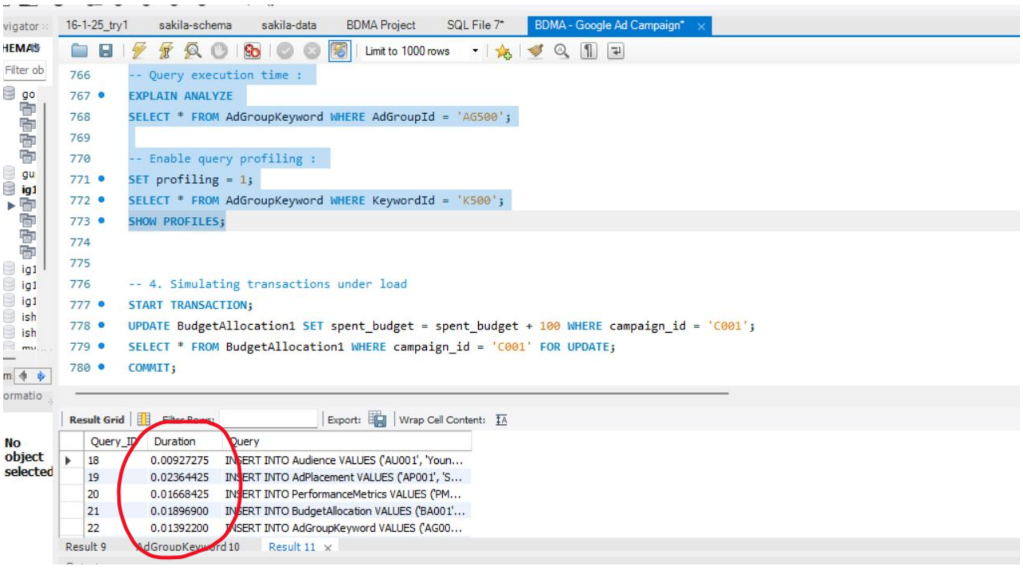
Comparison of Stress Testing Results :

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

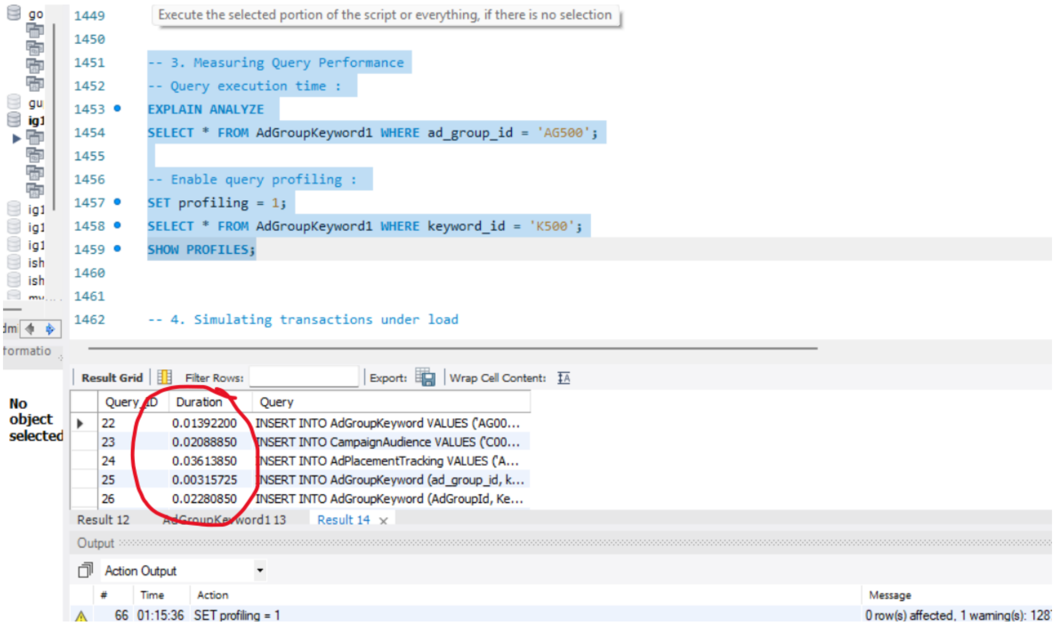
Before normalization :



```
-- Query execution time :
EXPLAIN ANALYZE
SELECT * FROM AdGroupKeyword WHERE AdGroupId = 'AG500';

-- Enable query profiling :
SET profiling = 1;
SELECT * FROM AdGroupKeyword WHERE KeywordId = 'K500';
SHOW PROFILES;


-- 4. Simulating transactions under load
START TRANSACTION;
UPDATE BudgetAllocation1 SET spent_budget = spent_budget + 100 WHERE campaign_id = 'C001';
SELECT * FROM BudgetAllocation1 WHERE campaign_id = 'C001' FOR UPDATE;
COMMIT;
```

Query_ID	Duration	Query
18	0.00927275	INSERT INTO Audience VALUES ('AU001', 'Youn...
19	0.02364425	INSERT INTO AdPlacement VALUES ('AP001', 'S...
20	0.01668425	INSERT INTO PerformanceMetrics VALUES ('PM...
21	0.01896900	INSERT INTO BudgetAllocation VALUES ('BA001'...
22	0.01392200	INSERT INTO AdGroupKeyword VALUES ('AG00...

After normalization :



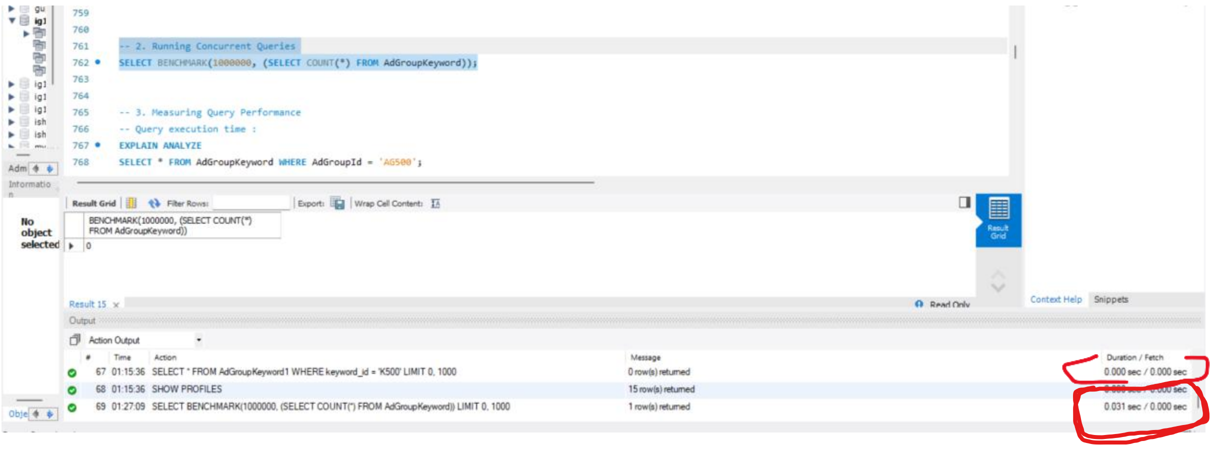
```
-- 3. Measuring Query Performance
-- Query execution time :
EXPLAIN ANALYZE
SELECT * FROM AdGroupKeyword1 WHERE ad_group_id = 'AG500';

-- Enable query profiling :
SET profiling = 1;
SELECT * FROM AdGroupKeyword1 WHERE keyword_id = 'K500';
SHOW PROFILES;


-- 4. Simulating transactions under load
```

Query_ID	Duration	Query
22	0.01392200	INSERT INTO AdGroupKeyword VALUES ('AG00...
23	0.02088850	INSERT INTO CampaignAudience VALUES ('C00...
24	0.03613850	INSERT INTO AdPlacementTracking VALUES ('A...
25	0.00315725	INSERT INTO AdGroupKeyword (ad_group_id, k...
26	0.02280850	INSERT INTO AdGroupKeyword (AdGroupId, Ke...

```
#   Time      Action              Message
66  01:15:36  SET profiling = 1   0 row(s) affected, 1 warning(s): 128
```

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

Before normalization (0.031 sec)

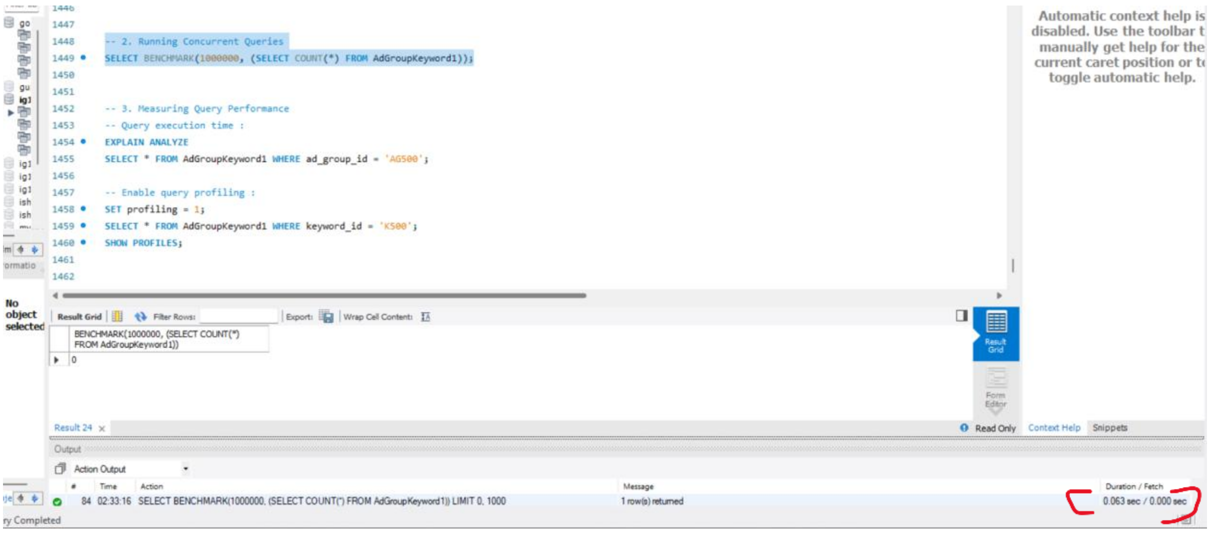


```
-- 2. Running Concurrent Queries
SELECT BENCHMARK(1000000, (SELECT COUNT(*) FROM AdGroupKeyword));


-- 3. Measuring Query Performance
-- Query execution time :
EXPLAIN ANALYZE
SELECT * FROM AdGroupKeyword WHERE AdGroupId = 'AG500';
```

BENCHMARK(1000000, (SELECT COUNT(*) FROM AdGroupKeyword))
0

```
#   Time      Action                                                                     Message            Duration / Fetch
67  01:15:36  SELECT * FROM AdGroupKeyword1 WHERE keyword_id = 'K500' LIMIT 0, 1000      0 row(s) returned  0.000 sec / 0.000 sec
68  01:15:36  SHOW PROFILES                                                              15 row(s) returned 0.000 sec / 0.000 sec
69  01:27:09  SELECT BENCHMARK(1000000, (SELECT COUNT(*) FROM AdGroupKeyword)) LIMIT 0, 1000  1 row(s) returned  0.031 sec / 0.000 sec
```

After normalization (0.063 sec)



```
-- 2. Running Concurrent Queries
SELECT BENCHMARK(1000000, (SELECT COUNT(*) FROM AdGroupKeyword1));


-- 3. Measuring Query Performance
-- Query execution time :
EXPLAIN ANALYZE
SELECT * FROM AdGroupKeyword1 WHERE ad_group_id = 'AG500';

-- Enable query profiling :
SET profiling = 1;
SELECT * FROM AdGroupKeyword1 WHERE keyword_id = 'K500';
SHOW PROFILES;
```

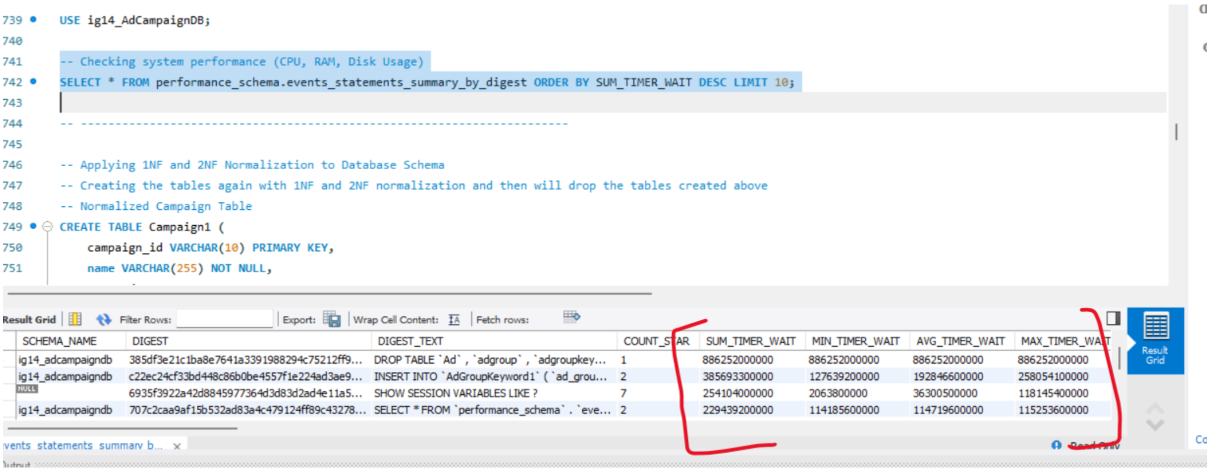
Automatic context help is disabled. Use the toolbar t manually get help for the current caret position or t toggle automatic help.

BENCHMARK(1000000, (SELECT COUNT(*) FROM AdGroupKeyword1))
0

```
#   Time      Action                                                                      Message            Duration / Fetch
84  02:33:16  SELECT BENCHMARK(1000000, (SELECT COUNT(*) FROM AdGroupKeyword1)) LIMIT 0, 1000  1 row(s) returned  0.063 sec / 0.000 sec
```

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

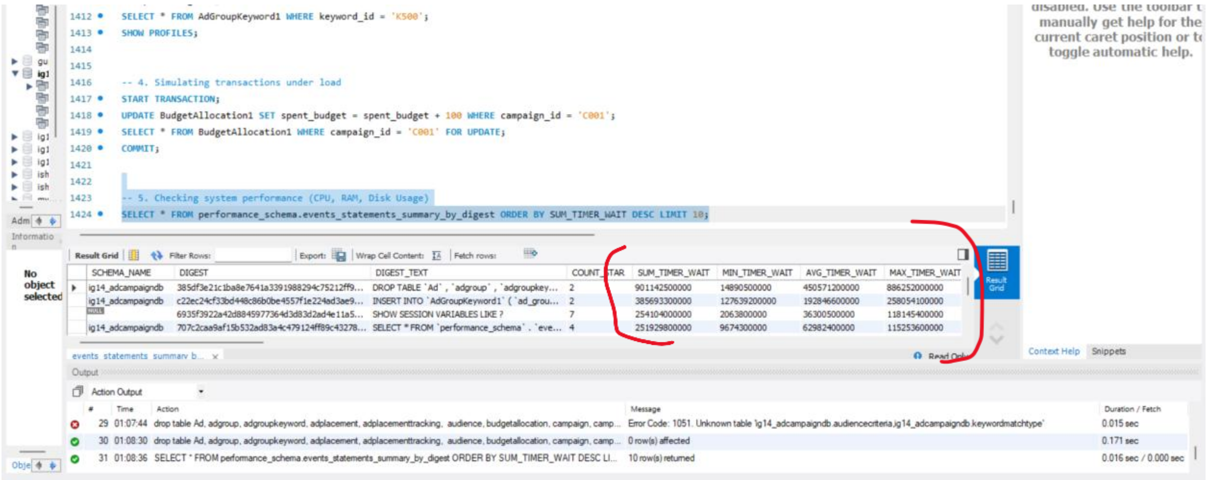
Before normalization :



```
USE ig14_AdCampaignDB;

-- Checking system performance (CPU, RAM, Disk Usage)
SELECT * FROM performance_schema.events_statements_summary_by_digest ORDER BY SUM_TIMER_WAIT DESC LIMIT 10;


-- ----------------------------------------------------------------------

-- Applying 1NF and 2NF Normalization to Database Schema
-- Creating the tables again with 1NF and 2NF normalization and then will drop the tables created above
-- Normalized Campaign Table
CREATE TABLE Campaign1 (
    campaign_id VARCHAR(10) PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
```

SCHEMA_NAME	DIGEST	DIGEST_TEXT	COUNT_STAR	SUM_TIMER_WAIT	MIN_TIMER_WAIT	AVG_TIMER_WAIT	MAX_TIMER_WAIT
ig14_adcampaigndb	385df3e21c1ba8e7641a3391988294c75212ff9...	DROP TABLE `Ad` , `adgroup` , `adgroupkey...	1	886252000000	886252000000	886252000000	886252000000
ig14_adcampaigndb	c22ec24cf33bd448c86b0be4557f1e224ad3ae9...	INSERT INTO `AdGroupKeyword1` (`ad_grou...	2	385693300000	127639200000	192846600000	258054100000
NULL	6935f3922a42d8845977364d3d83d2ad4e11a5...	SHOW SESSION VARIABLES LIKE ?	7	254104000000	2063800000	36300500000	118145400000
ig14_adcampaigndb	707c2caa9af15b532ad83a4c479124ff89c43278...	SELECT * FROM `performance_schema` . `eve...	2	229439200000	114185600000	114719600000	115253600000

After normalization :



```
SELECT * FROM AdGroupKeyword1 WHERE keyword_id = 'K500';
SHOW PROFILES;


-- 4. Simulating transactions under load
START TRANSACTION;
UPDATE BudgetAllocation1 SET spent_budget = spent_budget + 100 WHERE campaign_id = 'C001';
SELECT * FROM BudgetAllocation1 WHERE campaign_id = 'C001' FOR UPDATE;
COMMIT;


-- 5. Checking system performance (CPU, RAM, Disk Usage)
SELECT * FROM performance_schema.events_statements_summary_by_digest ORDER BY SUM_TIMER_WAIT DESC LIMIT 10;
```

disabled. Use the toolbar t
manually get help for the
current caret position or to
toggle automatic help.

SCHEMA_NAME	DIGEST	DIGEST_TEXT	COUNT_STAR	SUM_TIMER_WAIT	MIN_TIMER_WAIT	AVG_TIMER_WAIT	MAX_TIMER_WAIT
ig14_adcampaigndb	385df3e21c1ba8e7641a3391988294c75212ff9...	DROP TABLE `Ad` , `adgroup` , `adgroupkey...	2	901142500000	14890500000	450571200000	886252000000
ig14_adcampaigndb	c22ec24cf33bd448c86b0be4557f1e224ad3ae9...	INSERT INTO `AdGroupKeyword1` (`ad_grou...	2	385693300000	127639200000	192846600000	258054100000
NULL	6935f3922a42d8845977364d3d83d2ad4e11a5...	SHOW SESSION VARIABLES LIKE ?	7	254104000000	2063800000	36300500000	118145400000
ig14_adcampaigndb	707c2caa9af15b532ad83a4c479124ff89c43278...	SELECT * FROM `performance_schema` . `eve...	4	251929800000	9674300000	62982400000	115253600000

#	Time	Action	Message	Duration / Fetch
29	01:07:44	drop table Ad, adgroup, adgroupkeyword, adplacement, adplacementtracking, audience, budgetallocation, campaign, camp...	Error Code: 1051. Unknown table 'ig14_adcampaigndb.audiencecriteria,ig14_adcampaigndb.keywordmatchtype'	0.015 sec
30	01:08:30	drop table Ad, adgroup, adgroupkeyword, adplacement, adplacementtracking, audience, budgetallocation, campaign, camp...	0 row(s) affected	0.171 sec
31	01:08:36	SELECT * FROM performance_schema.events_statements_summary_by_digest ORDER BY SUM_TIMER_WAIT DESC LI...	10 row(s) returned	0.016 sec / 0.000 sec

MANAGERIAL INSIGHTS & RECOMMENDATIONS

1. **Adopt a fully normalized schema (at least up to 2NF or 3NF):**
   - 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.
   - 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:**
    - Implement proper transaction isolation levels to avoid deadlocks.
    - 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:**
    - Avoid full table scans by implementing efficient indexing.
    - Use EXPLAIN ANALYZE to debug slow queries and optimize them.

******************************** **END OF REPORT** ********************************