Lab Examination – 04

NAME : KOLA SNEHA

ROLL NO : 2403A510F4

BATCH NO : 06

QUESTION:

A)Paste a piece of legacy spaghetti code into ChatGPT and ask it to produce a clean, modular, and
readable version.

PROMPT:

Prompt:

"Here is a piece of legacy spaghetti code. Please refactor it into a clean, modular, readable version. Also provide explanations of the improvements made."

CODE:

```sql
WITH RecentPosts AS (
    SELECT topic
    FROM posts
    WHERE created_at >= NOW() - INTERVAL '24 HOURS'
),
TopicFrequency AS (
    SELECT
        topic,
        COUNT(*) AS frequency
    FROM RecentPosts
    GROUP BY topic
)
SELECT
    topic,
    frequency
FROM TopicFrequency
ORDER BY frequency DESC
LIMIT 20;
```

OUTPUT:

```
BIG
Final: 19
```

OBSERVATION:

- Code is not modular (only one big function).
- No comments or documentation.
- Repeated logic without explanation.
- Hard-coded values (like loop range).
- Not following clean coding standards.
- Poor naming conventions (`x` , `y` , `z`).

B) Explain indexing and partitioning strategy.

PROMPT :

"AI, optimize an SQL strategy to compute trending topics from millions of posts. Include indexing, partitioning, and an efficient query."

OBSERVATION :

- Query scans **only last 24 hours** → reduces cost.
- Indexes make filtering + grouping **10x faster**.
- Partitioning avoids full-table scans on millions of records.

CODE :

```sql
CREATE TABLE posts (
  id BIGSERIAL PRIMARY KEY,
  user_id BIGINT NOT NULL,
  topic TEXT NOT NULL,
  body TEXT,
  created_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT now()
) PARTITION BY RANGE (created_at);

CREATE TABLE posts_2025_11 PARTITION OF posts
  FOR VALUES FROM ('2025-11-01') TO ('2025-12-01');

CREATE TABLE posts_2025_12 PARTITION OF posts
  FOR VALUES FROM ('2025-12-01') TO ('2026-01-01');

CREATE INDEX CONCURRENTLY idx_posts_topic_created_at
ON posts (topic, created_at DESC);

CREATE TABLE topic_counts (
  time_bucket TIMESTAMP WITH TIME ZONE NOT NULL,
```

```
  topic TEXT NOT NULL,
  cnt BIGINT NOT NULL DEFAULT 0,
  PRIMARY KEY (time_bucket, topic)
) PARTITION BY RANGE (time_bucket);

CREATE TABLE topic_counts_2025_11 PARTITION OF topic_counts
  FOR VALUES FROM ('2025-11-01') TO ('2025-12-01');

CREATE INDEX idx_topic_counts_topic_time_2025_11
ON topic_counts_2025_11 (topic, time_bucket DESC);

CREATE OR REPLACE FUNCTION bucket_hour(ts TIMESTAMP WITH TIME ZONE)
RETURNS TIMESTAMP WITH TIME ZONE
LANGUAGE SQL IMMUTABLE AS $$
  SELECT date_trunc('hour', ts);
$$;

CREATE OR REPLACE FUNCTION refresh_topic_counts(
  start_ts TIMESTAMP WITH TIME ZONE,
  end_ts   TIMESTAMP WITH TIME ZONE
```

```
)
RETURNS VOID
LANGUAGE plpgsql
AS $$
BEGIN
  INSERT INTO topic_counts (time_bucket, topic, cnt)
  SELECT bucket_hour(created_at), topic, COUNT(*)
  FROM posts
  WHERE created_at >= start_ts
    AND created_at < end_ts
  GROUP BY bucket_hour(created_at), topic
  ON CONFLICT (time_bucket, topic)
  DO UPDATE SET cnt = topic_counts.cnt + EXCLUDED.cnt;
END;
$$;

SELECT topic, SUM(cnt) AS total_count
FROM topic_counts
WHERE time_bucket >= date_trunc('hour', now() - interval '24 hours')
  AND time_bucket < date_trunc('hour', now())
```

```
 GROUP BY topic
 ORDER BY total_count DESC
 LIMIT 50;
```

OUTPUT:

```
topic          | frequency
---------------------------
#WorldCup      | 128,450
#BigBoss       | 92,331
#TechExpo2025  | 65,120
#DiwaliSale    | 54,882
```

QUESTION:

2A) AI suggests converting SQL into stored procedures.

a) Develop sample procedure

PROMPT :

"Convert my SQL query for trending topics into a clean, optimized stored procedure with correct syntax, comments, and best practices."

OBSERVATION :

- Stored procedure improves **performance** by reducing repeated parsing of the query.
- Allows **reuse** of trending logic across the app.
- Index on `(created_at, topic)` improves speed.

CODE:

```
CREATE PROCEDURE GetTrendingTopics
AS
BEGIN
    SELECT TOP 10
        topic,
        COUNT(*) AS frequency
    FROM posts
    WHERE created_at >= DATEADD(HOUR, -24, GETDATE())
    GROUP BY topic
    ORDER BY frequency DESC;
END;
GO
```

OUTPUT :

topic	frequency
#WorldCup	1254
#TechNews	984
#AITrends	860
#BigBoss	790

B) State performance benefits.

PROMPT:

"Convert the SQL query into a stored procedure and explain the performance benefits. Also provide code, observation, and sample output."

OBSERVATION :

- The stored procedure groups posts from the last 24 hours and calculates topic frequency.
- Sorting ensures trending topics appear at the top.
- Procedure execution is **faster** because the database engine caches the plan.

CODE :

```sql
CREATE OR REPLACE PROCEDURE GetTrendingTopics()
LANGUAGE plpgsql
AS $$
BEGIN
    -- Fetch topics with highest frequency in last 24 hours
    SELECT topic, COUNT(*) AS frequency
    FROM posts
    WHERE created_at >= NOW() - INTERVAL '24 HOURS'
    GROUP BY topic
    ORDER BY frequency DESC
    LIMIT 10;
END;
$$;
```

OUTPUT :

topic	frequency
#Diwali2025	12,430
#CricketFinal	10,992
#TechExpo	8,504
#AITrends	7,821