1) Update, install MySQL server, MySQL client, PHP (Adminer), unzip, Python

A screenshot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer screen

AI-generated content may be incorrect.

A screen shot of a computer screen

AI-generated content may be incorrect.

2.MySQL client is installed in your Codespace environment.

A screenshot of a computer

AI-generated content may be incorrect.

3. Start your MySQL container:

A screen shot of a computer

AI-generated content may be incorrect.

4. MySQL container with port mapping 0.0.0.0:3307->3306/tcp:

A screenshot of a computer

AI-generated content may be incorrect.

5. Connected to MySQL:

A screenshot of a computer

AI-generated content may be incorrect.

6. Create table:

A screenshot of a computer

AI-generated content may be incorrect.

7. Load CSV into table:

A computer screen shot of a computer screen

AI-generated content may be incorrect.

8. Baseline tests (scalar fields & full-text) — EXPLAIN + timing:

A screenshot of a computer

AI-generated content may be incorrect.

9. full-text test:

A screenshot of a computer

AI-generated content may be incorrect.

10. Terminal showing the ALTER TABLE statements and success.A screenshot of a computer

AI-generated content may be incorrect.

11. Terminal showing ANALYZE TABLE orders:

A computer screen shot of a program

AI-generated content may be incorrect.

12. Re-run EXPLAIN & timings (after indexing):

A screenshot of a computer

AI-generated content may be incorrect.

13. Run the Python timing script and it show output:

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**PROG8850 — Assignment 5**

**Documentation: theory, tasks, findings & case study**

**1. Overview**

**Goal.** Import a ~100k ecommerce orders dataset into MySQL, run scalar and full-text queries, measure timings, inspect query plans (EXPLAIN), create indexes, re-measure, and summarise who benefits and why.

**Deliverables (this document):**

* Theoretical background
* Step-by-step task explanation and commands
* Timing methodology and Python script usage
* Template for recording results (before/after)
* Case study analysis & stakeholder mapping
* Screenshot checklist and packaging instructions

**2. Theoretical background (short and exam-ready)**

**Index types & why they matter**

* **B-tree indexes (DEFAULT)**  
  Best for equality and range queries on scalar fields (INT, DATE, DECIMAL). Use for WHERE amount > X, order\_date BETWEEN ....
  + Benefit: reduce scanned rows from full table scan to index lookup.
  + Cost: extra storage and slower writes (INSERT/UPDATE/DELETE).
* **Composite indexes**  
  B-tree on (col1, col2) supports queries that filter on leading column(s) and can avoid multiple indexes or file sorts.
* **FULLTEXT indexes**  
  Designed for text search (MATCH...AGAINST). Efficient for natural/boolean language searches across VARCHAR/TEXT columns. MySQL supports FULLTEXT on InnoDB and MyISAM (modern MySQL favors InnoDB).

**EXPLAIN (what to look for)**

When you run EXPLAIN SELECT ..., check:

* type — best is const, eq\_ref, ref, range; ALL = full table scan (bad).
* possible\_keys / key — whether an index could and does get used.
* rows — how many rows MySQL estimates to read.
* Extra — flags like Using index, Using where, Using temporary, Using filesort. Using filesort and Using temporary often indicate slower queries.

**Query timing methodology (principles)**

* Run each query multiple times (3–10) and use the **median** time to avoid outliers.
* Disable or avoid caching effects where possible. (MySQL 8 removed the query cache; still do repeated runs and use median.)
* Use EXPLAIN to compare plan changes before/after indexing — faster times + index in key is ideal.

**3. Environment & files (what I used / what you must include)**

* MySQL (container or local). In these instructions we used a Docker MySQL exposed on 127.0.0.1:3307.
* schema\_changes.sql — contains CREATE DATABASE and table creation (or you created orders manually).
* CSV(s) in data/ directory, e.g. data/orders.csv.
* Python timing script: .devcontainer/scripts/index\_timing.py (or scripts/index\_timing.py) — runs EXPLAIN and times queries.
* timings\_summary.txt — output from timing script.
* Screenshots in screenshots/ (see checklist below).

**4. Step-by-step tasks & copy-paste commands**

Do each block in order. Use the absolute path in Codespaces: /workspaces/<repo>/....

**4.1 Prepare MySQL (Docker example)**

# create/run MySQL container (maps host 3307 -> container 3306)

docker run --name mysql-container -e MYSQL\_ROOT\_PASSWORD=Secret5555 -e MYSQL\_DATABASE=prog8850 -p 3307:3306 -v /workspaces/PROG8850Assignment5/data:/data -d mysql:8.0 --local-infile=1

**4.2 Create DB schema**

Save this as schema\_changes.sql (or use the file you already edited):

CREATE DATABASE IF NOT EXISTS prog8850;

USE prog8850;

CREATE TABLE IF NOT EXISTS orders (

order\_id INT,

user\_id INT,

product\_id INT,

amount DECIMAL(10,2),

product\_name VARCHAR(255),

description TEXT,

order\_date DATE

);

Load it:

mysql -h 127.0.0.1 -P 3307 -u root -pSecret5555 prog8850 < schema\_changes.sql

**4.3 Load CSV into orders**

Start CLI with local infile allowed (important):

mysql --local-infile=1 -h 127.0.0.1 -P 3307 -u root -pSecret5555

Inside MySQL:

USE prog8850;

-- if you mounted /data in the docker run above, path is /data/orders.csv inside container;

-- but LOAD DATA LOCAL INFILE executes from client, so use the path visible to the client:

LOAD DATA LOCAL INFILE '/workspaces/PROG8850Assignment5/data/orders.csv'

INTO TABLE orders

FIELDS TERMINATED BY ','

OPTIONALLY ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 LINES

(order\_id, user\_id, product\_id, amount, product\_name, description, order\_date);

If LOCAL INFILE fails, reconnect the client with --local-infile=1 as shown.

**4.4 Verify:**

SELECT COUNT(\*) FROM orders;

SELECT \* FROM orders LIMIT 10;

DESCRIBE orders;

**4.5 Baseline EXPLAIN queries (before indexing)**

Inside MySQL (or via script), run and save the EXPLAIN outputs:

EXPLAIN SELECT COUNT(\*) FROM orders WHERE amount > 500;

EXPLAIN SELECT AVG(amount) FROM orders WHERE amount BETWEEN 10 AND 200;

EXPLAIN SELECT \* FROM orders WHERE order\_date BETWEEN '2020-01-01' AND '2020-12-31';

EXPLAIN SELECT \* FROM orders WHERE MATCH(description) AGAINST ('running shoes' IN NATURAL LANGUAGE MODE);

Save each EXPLAIN output to text files (or copy the MySQL console output).

**4.6 Baseline timing (before indexing)**

Run the Python timing script (ensure DB name/port correct):

# set env vars so script connects to your DB

export DB\_HOST=127.0.0.1

export DB\_PORT=3307

export DB\_USER=root

export DB\_PASSWORD=Secret5555

export DB\_NAME=prog8850

python3 ./.devcontainer/scripts/index\_timing.py | tee timings\_before.txt

This writes a summary to timings\_summary.txt and the console output to timings\_before.txt.

**4.7 Create indexes**

Scalar index(s):

ALTER TABLE orders ADD INDEX idx\_amount (amount);

ALTER TABLE orders ADD INDEX idx\_order\_date (order\_date);

-- composite example if you query by product and date:

ALTER TABLE orders ADD INDEX idx\_prod\_date (product\_id, order\_date);

Fulltext:

ALTER TABLE orders ADD FULLTEXT ft\_product\_desc (product\_name, description);

Then:

ANALYZE TABLE orders;

**4.8 Re-run EXPLAIN & timing (after indexing)**

* Re-run the same EXPLAIN commands and save outputs (into explain\_after\_\* files).
* Re-run the timing script and redirect outputs:

python3 ./.devcontainer/scripts/index\_timing.py | tee timings\_after.txt

**4.9 Clean-up (optional)**

# stop container

docker stop mysql-container

docker rm mysql-container

**5. Timing methodology & how to capture reproducible results**

* For each query, run the query at least 3 times; use the **median** of those runs.
* Use the script provided — it prints runs and median automatically.
* For better reproducibility:
  + Run ANALYZE TABLE orders; after creating indexes.
  + Run tests on a quiet machine (no other heavy load).
  + Capture EXPLAIN outputs and store them as text.

**Capture commands + files** (examples):

python3 ./.devcontainer/scripts/index\_timing.py | tee screenshots/04\_timing\_before.txt

# take screenshots of terminal showing the file

**6. Results template (fill with your measured numbers)**

Paste your actual measured numbers into the table below. Replace the placeholders.

**6.1 Timings summary (median times)**

| **Query** | **Median time (before)** | **Median time (after)** | **% improvement** |
| --- | --- | --- | --- |
| COUNT(amount > 500) | <COUNT\_before> s | <COUNT\_after> s | ((before-after)/before\*100) |
| AVG(amount between 10 and 200) | <AVG\_before> s | <AVG\_after> s | <avg\_impr>% |
| Full-text search ('running shoes') | <FT\_before> s | <FT\_after> s | <ft\_impr>% |
| Date range query (order\_date) | <DATE\_before> s | <DATE\_after> s | <date\_impr>% |

**How to compute percent improvement:**  
improvement% = (before\_time - after\_time) / before\_time \* 100

**6.2 EXPLAIN summary (examples)**

* Before indexing: type = ALL (table scan), key = NULL, rows = X
* After indexing: type = range or ref, key = idx\_amount, rows = Y (much smaller)

Paste representative EXPLAIN lines here.

**7. Case study analysis — who benefits & how**

This dataset is ecommerce orders. Typical stakeholders and how indexing helps:

* **Data analysts / BI team**
  + Interests: fast aggregate queries (count, avg, totals) and date-range analysis for dashboards.
  + Index benefit: B-tree index on order\_date and amount makes aggregations and filtered counts faster for dashboard queries and reduces report latency.
* **Marketing / Product managers**
  + Interests: searching product descriptions to find trending keywords or verifying promotions (full-text queries).
  + Index benefit: FULLTEXT index on product\_name, description speeds up relevance-based searches and enables MATCH...AGAINST queries used in analytics.
* **Customer Support**
  + Interests: quickly retrieving orders by product keywords or order amount to help a customer.
  + Index benefit: fulltext and B-tree indexes reduce time to find matching orders during support calls.
* **Fraud Detection / Finance**
  + Interests: range queries on amount to detect outliers or suspicious transactions.
  + Index benefit: idx\_amount supports threshold queries (amount > X) enabling near-real-time rule-based checks.

**Business impact**

* Faster queries mean faster dashboards and lower time-to-insight (BI) and better customer experience (support).
* Indexes trade storage and write overhead (slightly slower inserts) for substantially faster reads. For mostly read workloads this is sensible.