**Assignment 2 Documentation**

**Author:** Hrynkiv Illia

**Date:** October 22, 2025

### **Introduction**

The objective of this project is to demonstrate the process of identifying and optimizing an inefficient ("bad") AI-generated SQL query. The task involves applying optimization techniques such as **Indexing** and **Query Rewriting** to achieve an identical result with significantly improved performance.

Performance analysis is conducted by comparing the EXPLAIN ANALYZE execution plans before and after optimization.

### **Baseline Analysis (The Non-Optimized Query)**

The "bad" AI-generated query fulfills the following business request: "Show the total score for all players from 'UA' and count how many matches each of them played on the 'Dust2' map."

**Problem 1: Correlated Subquery**

* The SELECT COUNT(\*) expression in the main SELECT clause is a **Correlated Subquery**.

The database must execute this subquery **one time for every row** returned *after* the GROUP BY. So if the query finds 100,000 players from 'UA', this subquery will run **10,0000 times**.

**Problem 2: Non-SARGable Predicate**

The WHERE LEFT(p.country\_code, 2) = 'UA' condition is **non-SARGable**.

This forces the database to perform a **Full Table Scan**, meaning it must load every single row from the table and apply the condition one by one. This is extremely slow on large tables.

### **The Optimization Process**

To resolve the identified bottlenecks, two primary optimization methods were applied. *Method 1: Strategic Indexing.* Three new indexes were created to support the rewritten query:

idx\_player\_country: Allows the optimizer to instantly find rows for country\_code = 'UA', turning a **Full Table Scan** into a fast **Index Seek**.

idx\_match\_map: Helps the optimizer speed up the JOIN to matches and the CASE statement processing.

idx\_mr\_player\_score: This is a **Covering Index**, the most critical optimization. It allows the optimizer to get all required data **directly from the index** without ever touching the main table.

#### **Query Rewriting**

The "bad" query was logically rewritten to eliminate both problems:

The Correlated Subquery was replaced by a JOIN to the matches table and a **Conditional Aggregation**.

**Change:** (SELECT COUNT(\*)...) → SUM(CASE WHEN m.map\_name = 'Dust2' THEN 1 ELSE 0 END)

The non-SARGable predicate was replaced with the SARGable equivalent.

**Change:** LEFT(p.country\_code, 2) = 'UA' → p.country\_code = 'UA'

This allows MySQL to use the newly created idx\_player\_country index.

**Conclusion**

The final query returns the **exact same result** as the "bad" query but does so in a significantly more efficient manner.