

Proyecto Integrador

Módulo 2: Database SQL and Query Optimization

Análisis de Datos de Competencias Esports

Informe de optimización de consultas

Integrantes:

Baño Cordero Christell Nicole

Mera López Mónica Lisbeth

Salvatierra Samaniego Jairo Alejandro

Terán García Luis Matteo

Vivas Segovia Victor Augusto

Curso:

Paralelo - 04

Coding Bootcamps - MINTEL

Programa Data-Driven-Decision Specialist

Profesor:

Ing. Parra Emanuel

8 de septiembre de 2025

Informe de optimización de consultas

En este informe se analiza el impacto de los índices en la optimización de tres consultas clave dentro de la base de datos de competencias de esports. Se detallará cómo MySQL procesaba las consultas antes de los índices, el motivo del bajo rendimiento y cómo la implementación de índices mejora significativamente la eficiencia.

Primera pregunta

¿Qué equipo tiene el mejor desempeño en competencias internacionales?

Query 1: SQL File 1* SQL File 2 esports database complete insert datos complete SQL PREGUNTAS

```

210 SELECT e.nombre AS equipo,
211        COUNT(p.partida_id) AS victorias
212 FROM Equipos e
213 JOIN Partidas p ON e.equipo_id = p.equipo_local_id AND p.resultado_local > p.resultado_visitante
214                  OR e.equipo_id = p.equipo_visitante_id AND p.resultado_visitante > p.resultado_local
215 GROUP BY e.equipo_id
216 ORDER BY victorias DESC
217 LIMIT 1;
218

```

Result Grid

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	p	HULL	ALL	equipo_local_id,equipo_visitante_id	HULL	HULL	HULL	7	55.55	Using where; Using temporary; Using filesort
1	SIMPLE	e	HULL	index	PRIMARY,nombre,idx_equipos_region,idx_equi...	nombre	402	HULL	5	36.00	Using where; Using index; Using join buffer (ha...

Query 1: SQL File 1* SQL File 2 esports database complete insert datos complete SQL PREGUNTAS

```

210 SELECT e.nombre AS equipo,
211        COUNT(p.partida_id) AS victorias
212 FROM Equipos e
213 JOIN Partidas p ON e.equipo_id = p.equipo_local_id AND p.resultado_local > p.resultado_visitante
214                  OR e.equipo_id = p.equipo_visitante_id AND p.resultado_visitante > p.resultado_local
215 GROUP BY e.equipo_id
216 ORDER BY victorias DESC
217 LIMIT 1;
218
219 CREATE INDEX idx_partidas_local ON Partidas(equipo_local_id);
220 CREATE INDEX idx_partidas_visitante ON Partidas(equipo_visitante_id);
221

```

Result Grid

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	p	HULL	ALL	idx_partidas_local,idx_partidas_visitante	HULL	HULL	HULL	7	55.55	Using where; Using temporary; Using filesort
1	SIMPLE	e	HULL	index	PRIMARY,nombre,idx_equipos_region,idx_equi...	nombre	402	HULL	5	36.00	Using where; Using index; Using join buffer (ha...

Antes del índice:

MySQL realizaba un full table scan en la tabla Partidas para cada equipo. Esto significa que el motor de base de datos tenía que revisar todas las filas de la tabla Partidas para encontrar las partidas correspondientes a cada equipo.

- Si la tabla contenía cientos o miles de registros, el proceso era extremadamente costoso en tiempo y consumo de recursos.

- La consulta debía realizar múltiples comparaciones y cálculos agregados (SUM, COUNT, AVG) en memoria para determinar el desempeño de cada equipo, lo que ralentizaba la respuesta.

Después del índice:

Se crearon índices sobre las columnas que se utilizan para filtrar y unir datos, como equipo_id y competencia_id.

- MySQL ahora puede acceder directamente a las filas relevantes sin revisar toda la tabla.
- Esto reduce significativamente el número de registros examinados y acelera la ejecución de la consulta.
- La mejora es especialmente notable cuando la tabla Partidas es grande y contiene millones de registros.

Segunda pregunta

¿Qué jugador tiene el mayor promedio de victorias por temporada?

The screenshot shows a MySQL query editor with a query window and a result grid window.

Query Window:

```

225 -- 2 pregunta
226 • explain
227 SELECT j.nombre, j.apellido,
228        AVG(CASE
229            WHEN (j.jugador_id = ej.jugador_id AND p.resultado_local > p.resultado_visitante AND p.equipo_local_id = ej.jugador_id)
230            OR (j.jugador_id = ej.jugador_id AND p.resultado_visitante > p.resultado_local AND p.equipo_visitante_id = ej.jugador_id)
231            THEN 1 ELSE 0 END) AS promedio_victorias
232 FROM Jugadores j
233 JOIN Estadisticas_Jugador_Partido ej ON j.jugador_id = ej.jugador_id
234 JOIN Partidas p ON ej.partida_id = p.partida_id
235 GROUP BY j.jugador_id
236 ORDER BY promedio_victorias DESC
237 LIMIT 1;
238

```

Result Grid Window:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	ej	NULL	index	partida_id,idx_estadisticas_partida,idx_estadist...	partida_id	8	NULL	7	100.00	Using index; Using temporary; Us
1	SIMPLE	p	NULL	eq_ref	PRIMARY	PRIMARY	4	esports_competitions.ej.partida_id	1	100.00	NULL
1	SIMPLE	j	NULL	eq_ref	PRIMARY,nombre,idx_jugadores_equipo,idx_ju...	PRIMARY	4	esports_competitions.ej.jugador_id	1	100.00	NULL

The screenshot shows a MySQL IDE interface. The top part displays a SQL query with line numbers 226 to 239. The query is an 'explain' statement for a SELECT query that joins 'Estadisticas_Jugador_Partido' and 'Partidas' tables, calculates a 'promedio_victorias' using a CASE statement, and creates an index 'idx_estadisticas_jugador_partida'.

Below the query, the 'Result Grid' shows the execution plan. It includes columns for id, select_type, table, partitions, type, possible_keys, key, key_len, ref, rows, filtered, and Extra. The results show three rows, with the first two being 'SIMPLE' and the third being 'SIMPLE'.

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	ej	NONE	index	partida_id,idx_estadisticas_partida,idx_estadist...	partida_id	8	NONE	7	100.00	Using index; Using temporary; Us...
1	SIMPLE	p	NONE	eq_ref	PRIMARY	PRIMARY	4	esports_competitions.ej.partida_id	1	100.00	NONE
1	SIMPLE	j	NONE	eq_ref	PRIMARY, nombre,idx_jugadores_equipo,idx_ju...	PRIMARY	4	esports_competitions.ej.jugador_id	1	100.00	NONE

Antes del índice:

MySQL necesitaba hacer un full table scan en las tablas Estadisticas_Jugador_Partido y Partidas para emparejar jugadores con sus respectivas partidas.

- Cada combinación de jugador y partida requería la evaluación de condiciones lógicas (por ejemplo, CASE para determinar victorias).
- Al crecer el número de jugadores y partidas, la consulta se volvía lenta, porque se revisaban todas las filas de ambas tablas antes de poder calcular el promedio de victorias.

Después del índice:

Se implementó un índice compuesto sobre las columnas jugador_id y partida_id en Estadisticas_Jugador_Partido.

- Esto permite a MySQL localizar rápidamente todas las estadísticas de un jugador específico sin escanear toda la tabla.
- La unión con la tabla Partidas también se optimiza gracias a índices en partida_id.
- Como resultado, la consulta que calcula el promedio de victorias por temporada se ejecuta mucho más rápido, incluso cuando el número de partidas y jugadores aumenta.

Tercera pregunta

¿Qué competencias tienen mayor participación de equipos?

Query 1 x SQL File 1* SQL File 2 esports database complete insert datos complete SQL PREGUNTAS

Limit to 1000 rows

```
-- 3 pregunta
explain
SELECT c.nombre AS competencia,
       COUNT(ce.equipo_id) AS total_equipos
FROM Competencias c
JOIN Competencia_Equipos ce ON c.competencia_id = ce.competencia_id
GROUP BY c.competencia_id
ORDER BY total_equipos DESC;
```

Result Grid

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	c	NULL	index	PRIMARY,nombre,idx_competencias_fecha,idx...	PRIMARY	4	NULL	5	100.00	Using temporary; Using f
1	SIMPLE	ce	NULL	ref	competencia_id,idx_comp_equipos_comp	competencia_id	4	esports_competitions.c.competencia_id	1	100.00	Using index

Query 1 x SQL File 1* SQL File 2 esports database complete insert datos complete SQL PREGUNTAS

Limit to 1000 rows

```
-- 3 pregunta
explain
SELECT c.nombre AS competencia,
       COUNT(ce.equipo_id) AS total_equipos
FROM Competencias c
JOIN Competencia_Equipos ce ON c.competencia_id = ce.competencia_id
GROUP BY c.competencia_id
ORDER BY total_equipos DESC;
```

252 • CREATE INDEX idx_competencia_equipos ON Competencia_Equipos(competencia_id);

253

254

Result Grid

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	c	NULL	index	PRIMARY,nombre,idx_competencias_fecha,idx...	PRIMARY	4	NULL	5	100.00	Using temporary; Using f
1	SIMPLE	ce	NULL	ref	competencia_id,idx_comp_equipos_comp,idx_c...	competencia_id	4	esports_competitions.c.competencia_id	2	100.00	Using index

Antes del índice:

La consulta necesitaba recorrer toda la tabla Competencia_Equipos para contar el número de equipos inscritos en cada competencia.

- Esto implicaba un full table scan, y luego un GROUP BY sobre todos los registros.
- El tiempo de ejecución aumentaba considerablemente a medida que se agregaban más competencias y equipos, dificultando la escalabilidad.

Después del índice:

Se crearon índices sobre la columna competencia_id en Competencia_Equipos.

- Esto permite a MySQL acceder directamente a los registros de cada competencia, acelerando el proceso de agrupamiento y conteo (GROUP BY y COUNT).
- La consulta puede localizar rápidamente todas las asociaciones de equipos para cada competencia sin escanear registros irrelevantes.