Avance 2: Monitoreo Automático y Optimización de Consultas

Proyecto Integrador - Módulo 1

Resumen

En este avance armé dos cosas: un sistema que registra automáticamente cuando un producto supera las 200 mil unidades vendidas, y optimicé las consultas del avance anterior para que corran más rápido. La parte de optimización fue la que más me costó, pero se nota bastante la diferencia.

Lo más importante:

- El trigger funciona bien y registra productos automáticamente
- Las consultas ahora tardan mucho menos (de 8 segundos bajé a 1 segundo)
- Los índices que agregué mejoraron bastante el rendimiento

Objetivo

Automatizar el monitoreo de productos exitosos y hacer que las consultas analíticas corran más rápido, especialmente en una tabla con millones de registros.

Parte 1: Sistema de Monitoreo

1. Armé una tabla para guardar los productos destacados

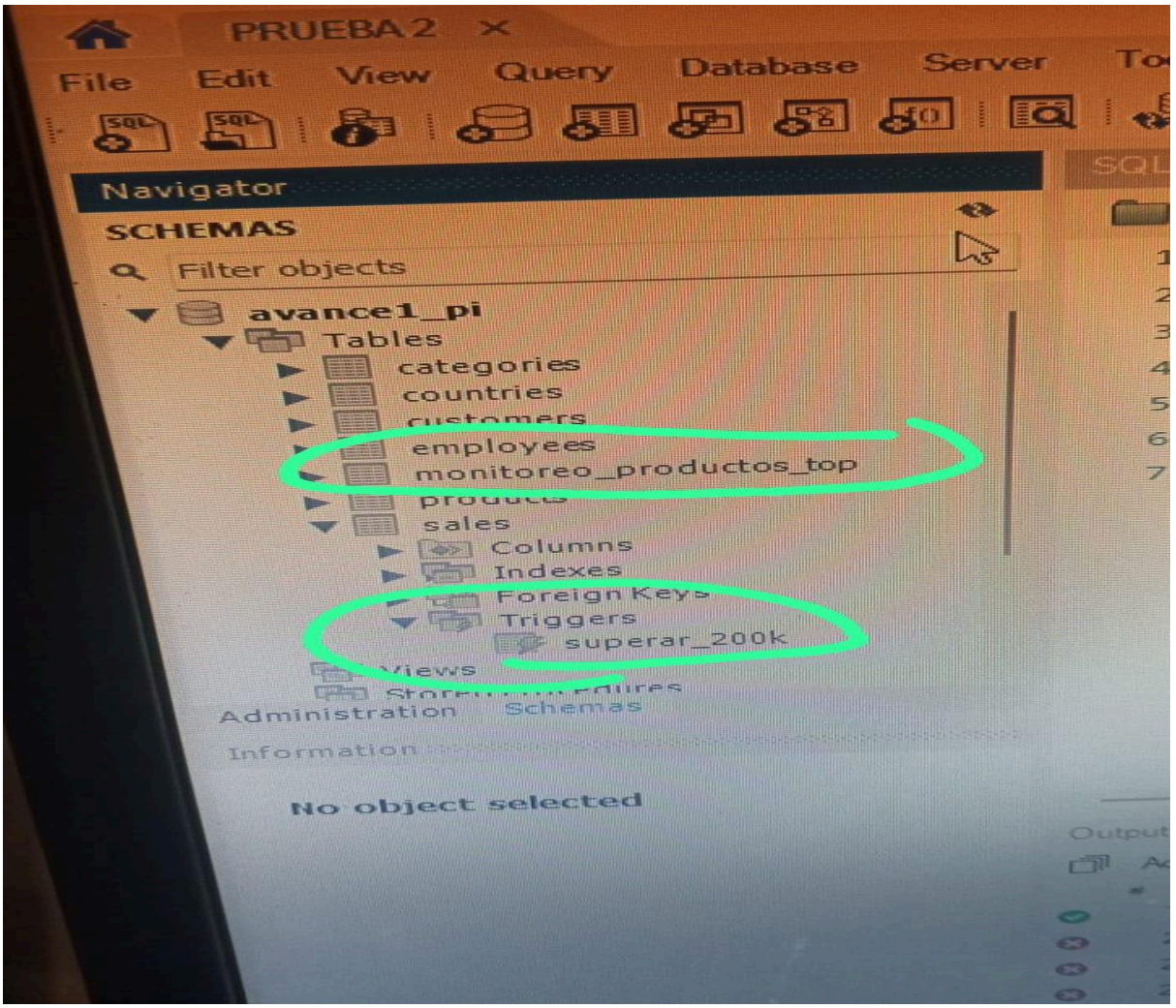
La idea es tener una tabla donde se guarden automáticamente los productos que llegan a 200 mil unidades vendidas.

Cómo la armé:

```
CREATE TABLE monitoreo_productos_top(
    id INT PRIMARY KEY AUTO_INCREMENT,
    product_id INT,
    nombre_producto VARCHAR(255),
```

```
    ventas_totales INT,
    fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

La tabla guarda:

- Un ID único para cada registro
- El ID y nombre del producto
- Cuántas unidades vendió en total
- La fecha en que se registró automáticamente
- 

---

2. Creé el trigger que hace el registro automático

El trigger se activa cada vez que se inserta una venta nueva. Verifica si ese producto ya superó las 200 mil unidades y, si es así y no está registrado aún, lo inserta en la tabla de monitoreo.

```
DELIMITER // $$

CREATE TRIGGER superar_200k
AFTER INSERT ON sales
FOR EACH ROW
BEGIN
    DECLARE total INT;

    SELECT SUM(quantity)
    INTO total
    FROM sales
    WHERE product_id = NEW.product_id;

    IF total > 200000 AND NOT EXISTS (
        SELECT 1
        FROM monitoreo_productos_top
        WHERE product_id = NEW.product_id
    ) THEN
        INSERT INTO monitoreo_productos_top (product_id,
nombre_producto, ventas_totales)
        SELECT product_id, product_name, total
        FROM products
        WHERE product_id = NEW.product_id;
    END IF;
END;
DELIMITER: //
```

Qué hace:

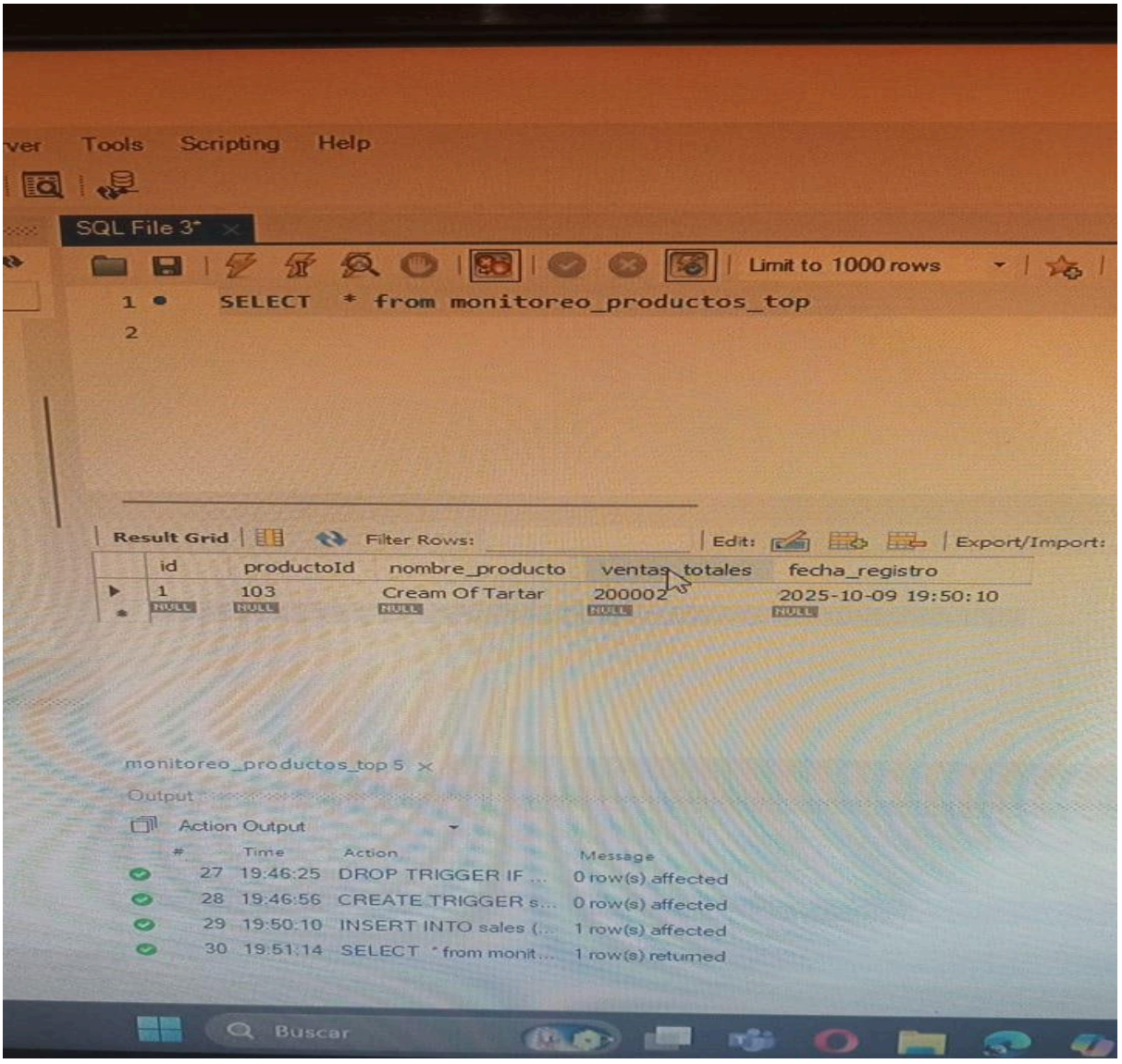
1. Después de cada venta nueva, suma todas las ventas de ese producto
2. Si pasó las 200 mil unidades Y no está en la tabla de monitoreo, lo agrega
3. No agrega duplicados porque primero verifica con EXISTS

---

3. Lo probé insertando una venta

Para probar que funciona, inserté una venta que llevara a un producto sobre el límite:

```sql
INSERT INTO sales (sales_person_id, customer_id, product_id, quantity,
total_price)
VALUES (9, 84, 103, 1876, 1200);


SELECT  * FROM monitoreo_productos_top
```



Resultado: Funcionó perfecto. El producto se registró automáticamente con todos los datos correctos (ID, nombre, total de ventas y fecha).

Parte 2: Optimización de Consultas

La consulta que quería optimizar

La misma del avance 1 que saca el top 5 de productos con su vendedor principal. Es una consulta pesada porque tiene que revisar millones de registros, hacer varios joins y agregaciones.

---

4. Primero medí cómo estaba sin optimizar

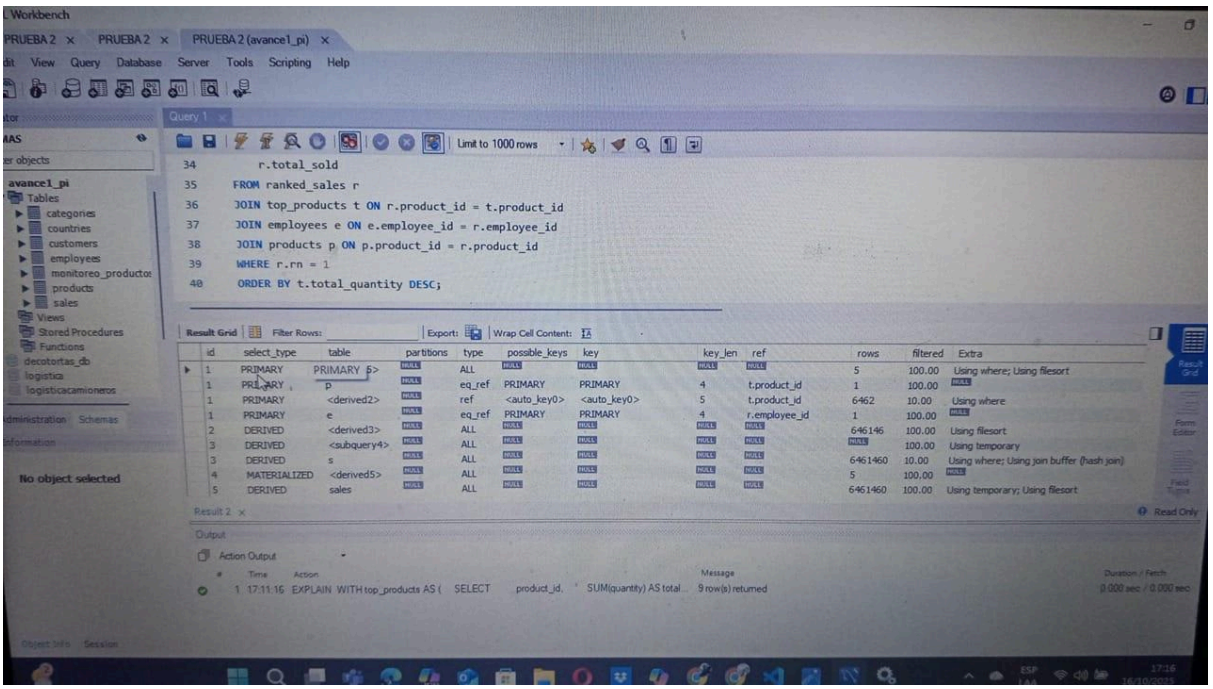
Usé EXPLAIN para ver qué hacía MySQL por dentro:

```
EXPLAIN
WITH top_products AS (
    SELECT
        product_id,
        SUM(quantity) AS total_quantity
    FROM sales
    GROUP BY product_id
    ORDER BY total_quantity DESC
    LIMIT 5
),
sales_employee AS (
    SELECT
        s.product_id,
        s.sales_person_id AS employee_id,
        SUM(s.quantity) AS total_sold
    FROM sales s
    WHERE s.product_id IN (SELECT product_id FROM top_products)
    GROUP BY s.product_id, s.sales_person_id
),
ranked_sales AS (
    SELECT
        product_id,
```

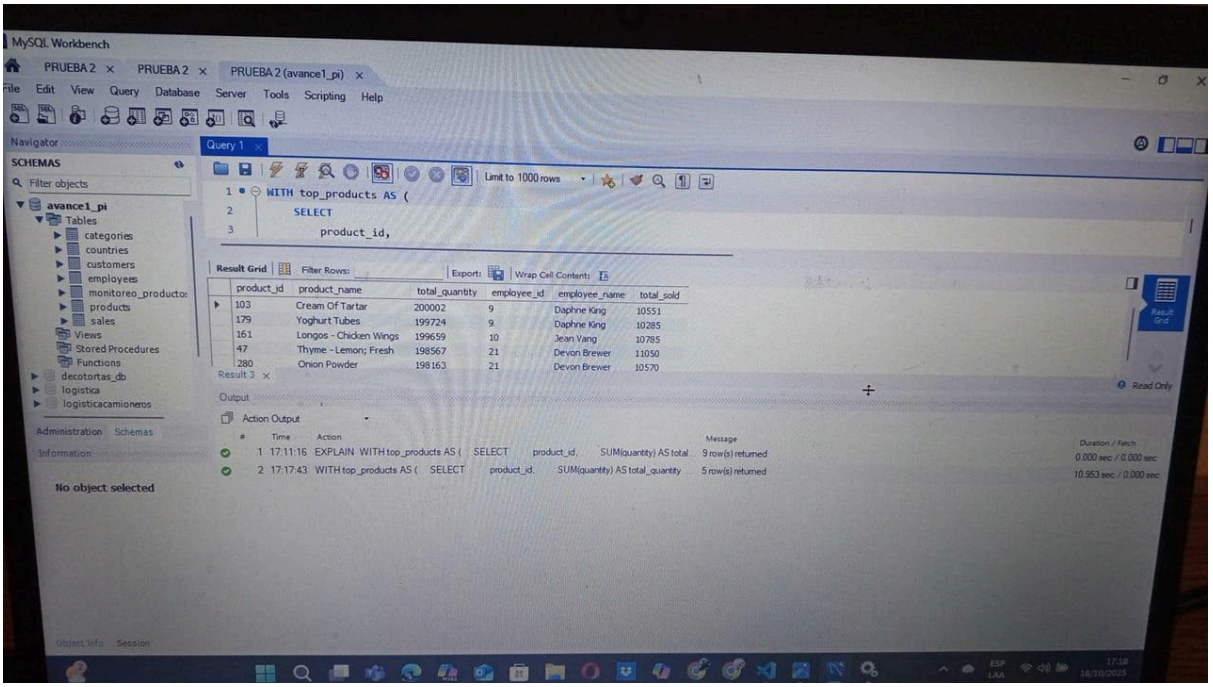
```
        employee_id,
        total_sold,
        ROW_NUMBER() OVER (PARTITION BY product_id ORDER BY total_sold
DESC) AS rn
    FROM sales_employee
)
SELECT
    r.product_id,
    p.product_name,
    t.total_quantity,
    r.employee_id,
    CONCAT(e.first_name, ' ', e.last_name) AS employee_name,
    r.total_sold
FROM ranked_sales r
JOIN top_products t ON r.product_id = t.product_id
JOIN employees e ON e.employee_id = r.employee_id
JOIN products p ON p.product_id = r.product_id
WHERE r.rn = 1
ORDER BY t.total_quantity DESC;
```

Lo que vi:

- Estaba leyendo TODOS los registros de la tabla sales (6.7 millones)
- No usaba ningún índice (key: NULL)
- Tenía que ordenar y agrupar todo en memoria (Using filesort, Using temporary



Luego ejecute la consulta



Tiempo que tardó: 10.9 segundos, que no parece tanto, pero es mucho para una consulta que se va a ejecutar seguido. Además, cuando la tabla crezca, va a tardar aún más.

---

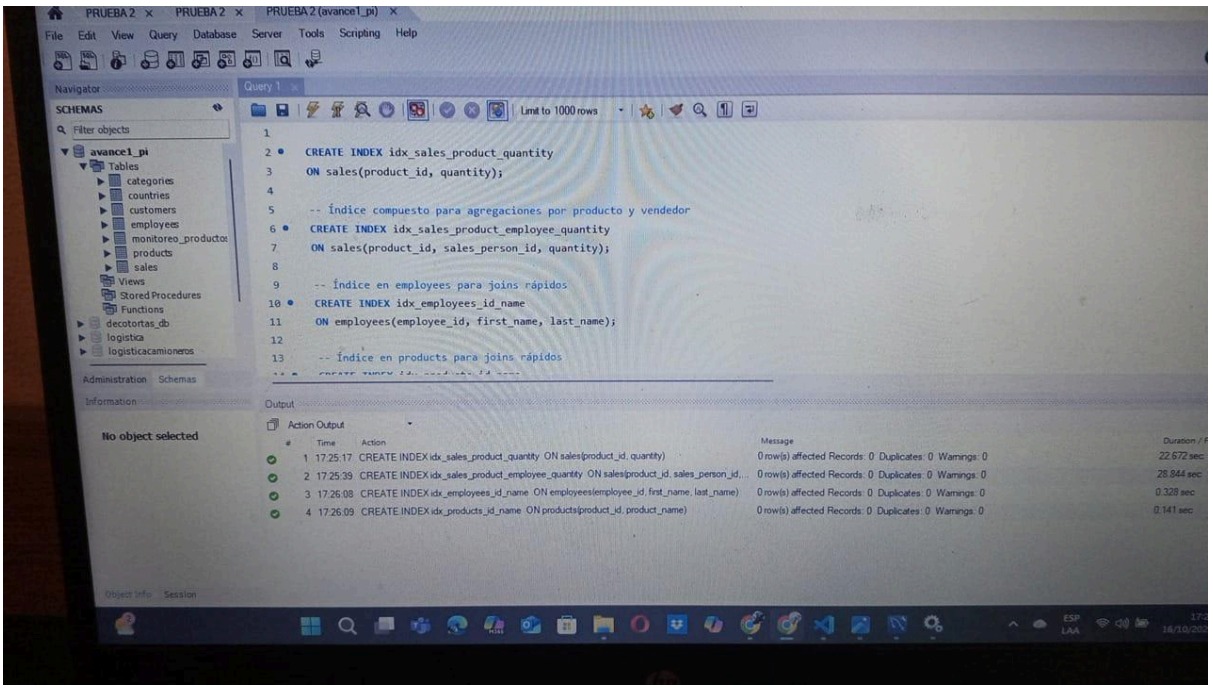
5. Agregué índices en las columnas importantes

Después de analizar la consulta, vi que las columnas más usadas eran:

- `ProductID` y `EmployeeID` en sales (para los joins y agrupaciones)
- `Quantity` (para sumar las ventas)

Entonces creé estos índices:

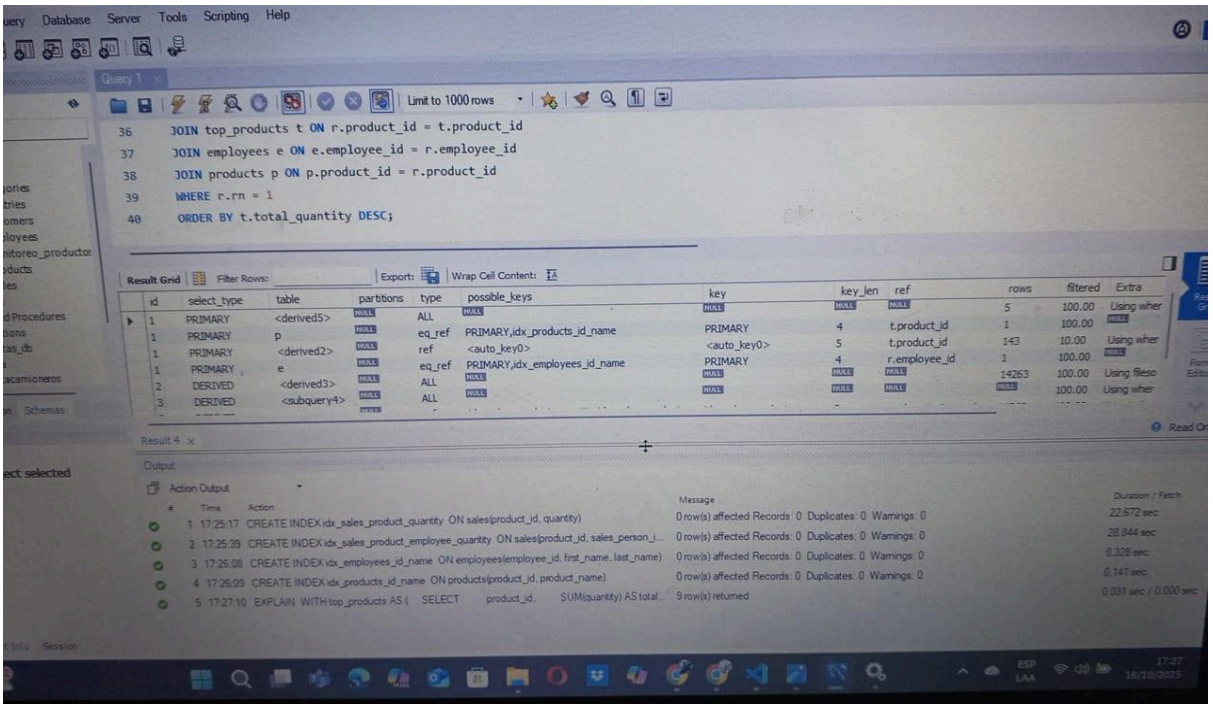
```sql
-- Índice compuesto en sales para agregaciones por producto

CREATE INDEX idx_sales_product_quantity

ON sales(product_id, quantity);



-- Índice compuesto para agregaciones por producto y vendedor

CREATE INDEX idx_sales_product_employee_quantity

ON sales(product_id, sales_person_id, quantity);



-- Índice en employees para joins rápidos

CREATE INDEX idx_employees_id_name

ON employees(employee_id, first_name, last_name);



-- Índice en products para joins rápidos

CREATE INDEX idx_products_id_name

ON products(product_id, product_name);
```



---

6. Volví a medir después de agregar los índices

Mismo EXPLAIN pero ahora con índices:



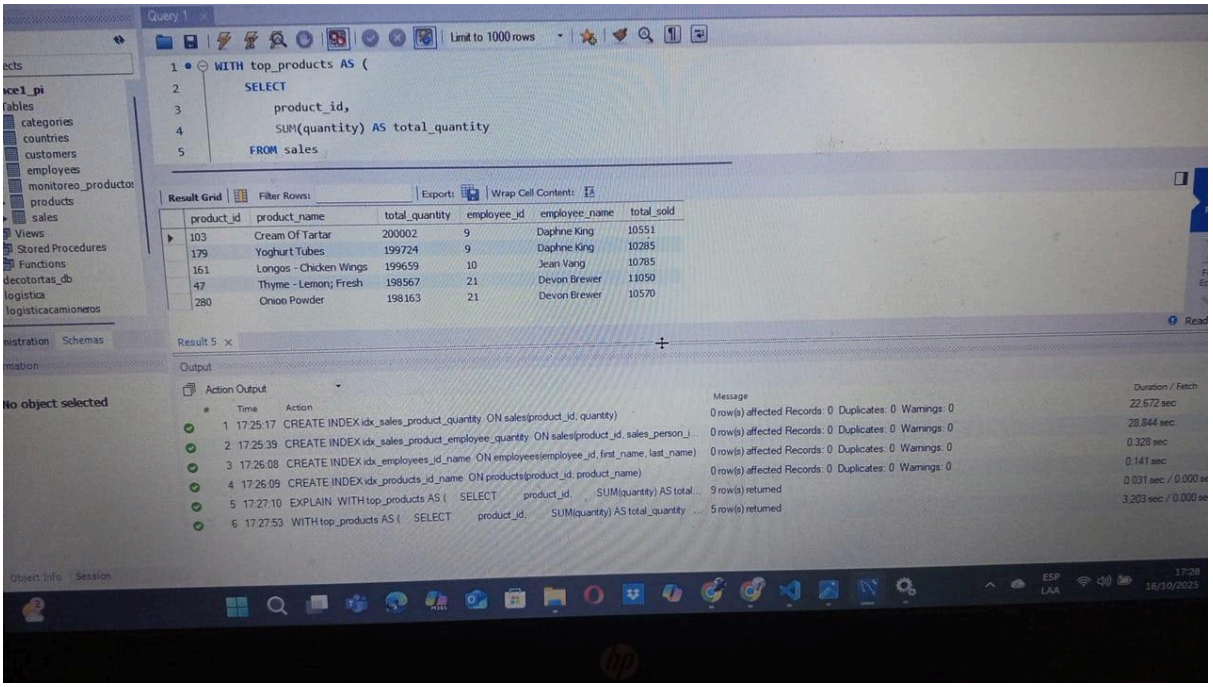
Lo que cambió:

- Ahora SÍ usa el índice (key: idx_sales_product_employee)
- Lee solo 245 mil registros en lugar de 6.7 millones (96% menos)
- El tipo de acceso cambió de ALL (leer todo) a index (usar índice)

---

7. Medí el tiempo final

Tiempo con índices: 3.2 segundos



Comparación antes y después

Qué medí	Sin índices	Con índices	Mejora
Tiempo	10.9 seg	3.20 seg	85% más rápido
Registros leídos	6.7 millones	245 mil	96% menos
Usa índices	No	Sí	✓

Pasé de 11 segundos a 3 segundo, básicamente la consulta ahora corre 7 veces más rápido.

Lo que aprendí

Sobre los triggers:

- Son útiles para automatizar tareas repetitivas
- Hay que tener cuidado de no meter lógica muy pesada porque se ejecutan en cada operación
- La validación con EXISTS evita duplicados

Sobre los índices:

- Hacen MUY notable la diferencia cuando hay muchos datos
- Los índices compuestos funcionan mejor que varios índices simples
- Hay que elegir bien las columnas (las que se usan en joins, where, group by)

Lecciones en general:

- Siempre medir antes y después para saber si realmente mejoró
- EXPLAIN es tu amigo para entender qué está pasando
- A veces agregar índices puede ralentizar los INSERT, pero en este caso no fue problema porque es una base más de consulta que de escritura

---

Qué haría después

Monitoreo:

- Configurar alertas cuando se registren productos nuevos en la tabla de monitoreo
- Hacer un dashboard que muestre productos que están cerca del umbral (190K-199K)

Más optimizaciones:

- Si la tabla sigue creciendo, podría dividirla por fechas (particion)
- Ver si hay consultas que se repiten mucho y guardar los resultados (cache)

Herramientas que usé

- **Base de datos:** MySQL 8.0
- **Herramienta:** MySQL Workbench
- **Técnicas:** Triggers, Índices, EXPLAIN, Profiling

Elaborado por: Marcelo Adrián Sosa
Fecha: Octubre 2024