Optimización de Consultas SQL: Estrategias para Consultas Frecuentes.

Dilan Alessandro Corredor Diaz
Escuela de Ingenieria de Sistemas
Universidad Industrial de Santander
Bucaramanga,Colombia
dilancorr@gmail.com

***Abstract*—This paper focuses on addressing SQL query performance optimization, specifically focusing on the most frequently used queries in database environments. The main objective is to identify effective strategies to improve the efficiency and response time of these queries, through the application of various optimization techniques.**

Common queries such as complex SELECT, JOINs, subqueries and other recurring patterns will be addressed. Key optimization strategies will be discussed, including the use of appropriate indexes, query rewriting, materialized views, and advanced JOIN optimization techniques.

In addition, specific tools and advanced techniques for performance analysis will be explored, with the goal of providing practical examples and case studies showing the impact of these optimizations on SQL query response times and efficiency.

***Resumen*— Este paper se centra en abordar la optimización del rendimiento de consultas SQL, específicamente enfocado en las consultas más frecuentemente utilizadas en entornos de bases de datos. El objetivo principal es identificar estrategias efectivas para mejorar la eficiencia y el tiempo de respuesta de estas consultas, a través de la aplicación de diversas técnicas de optimización.**

Se abordarán consultas comunes como SELECT complejos, JOINs, subconsultas y otros patrones recurrentes. Se analizarán estrategias clave de optimización, incluyendo la utilización de índices adecuados, reescritura de consultas, vistas materializadas y técnicas avanzadas de optimización de JOINs.

Además, se explorarán herramientas específicas y técnicas avanzadas para el análisis de rendimiento, con el objetivo de proporcionar ejemplos prácticos y casos de estudio que muestren el impacto de estas optimizaciones en los tiempos de respuesta y la eficiencia de las consultas SQL.

Keywords—Optimización SQL, Rendimiento consultas, Índices bases datos, JOINs optimización, Eficiencia consultas, Tiempos respuesta SQL

I. INTRODUCCIÓN

En el contexto actual de sistemas de bases de datos, la optimización del rendimiento de consultas SQL se ha convertido en un componente crítico para garantizar la eficiencia y la capacidad de respuesta en entornos de datos cada vez más complejos y dinámicos. La capacidad de mejorar la eficiencia de las consultas más frecuentes se vuelve fundamental para mantener la agilidad operativa y la capacidad de escalabilidad en aplicaciones y sistemas que dependen de bases de datos relacionales.

Las consultas SQL, son el núcleo de la interacción de los sistemas con las bases de datos relacionales, representan un área crucial para la optimización del rendimiento. Las consultas más frecuentes, tales como las SELECT complejas que involucran múltiples tablas, JOINs, subconsultas anidadas o agregaciones extensas, pueden afectar significativamente el tiempo de respuesta y, por ende, la experiencia del usuario final.

Este proyecto se enfoca en abordar esta problemática, reconociendo que la optimización del rendimiento de consultas SQL no solo tiene un impacto directo en la velocidad de respuesta de las aplicaciones, sino que también influye en la escalabilidad de estas, no solo mejorando la experiencia del usuario, sino que también contribuyendo a la capacidad de escalabilidad de los sistemas en entornos de datos en constante expansión, sin miedo a llegar a comprometer el rendimiento causado por el constante aumento de volumen de datos.

A través de la implementación de estas estrategias y técnicas, se busca desarrollar una guía práctica dirigida a profesionales de bases de datos y desarrolladores. Este recurso tiene como objetivo brindar un enfoque detallado y sólido para mejorar significativamente el rendimiento y la eficiencia de las consultas SQL en sistemas y aplicaciones que experimentan una creciente demanda y se encuentran inmersos en entornos de datos en constante expansión.

El objetivo es proporcionar un conjunto de prácticas recomendadas, estrategias probadas y soluciones prácticas, todo ello respaldado por ejemplos concretos y análisis de resultados. Esta guía no solo se centrará en la teoría de la optimización de consultas, sino que también ofrecerá una perspectiva práctica sobre cómo implementar estas estrategias.

Al proporcionar una comprensión clara y práctica de cómo mejorar el rendimiento de las consultas SQL más comunes, aspiramos a capacitar a los profesionales de bases de datos y desarrolladores para que puedan abordar los desafíos específicos de la optimización de consultas. Convirtiéndose así en una herramienta valiosa para enfrentar los retos de rendimiento y eficiencia en sistemas que necesitan adaptarse y escalar en un entorno de datos en constante evolución.

II. ESTADO DEL ARTE

La optimización de consultas SQL se ha convertido en un área fundamental en la gestión de bases de datos, enfocándose en la eficiencia y velocidad de las consultas. Permitiendo impactar positivamente en el rendimiento general de los sistemas y ayudando a facilitar la escalabilidad a entornos que tengan mayor volumen de datos.

- **Enfoque en Consultas Frecuentes:** Se busca mejorar la eficiencia de consultas SQL comunes como SELECT complejos, JOINs extensos, subconsultas y agregaciones, cruciales para la experiencia del usuario y operatividad de las aplicaciones.
- **Estrategias de Optimización:** Se trabajan algunas estrategias como índices optimizados, reescritura de consultas, vistas materializadas y optimización de JOINs para reducir tiempos de ejecución y mejorar el manejo de datos.
- **Impacto en Experiencia del Usuario y Escalabilidad:** La eficiencia en consultas mejora la experiencia del usuario al reducir tiempos de respuesta y es crucial para la escalabilidad, permitiendo manejar datos crecientes de manera ágil.
- **Herramientas y Recursos Disponibles:** La comunidad ofrece libros como " Optimización de consultas en bases de datos relacionales" y "MySQL 8: Optimización de consultas", junto con recursos en línea que brindan soluciones prácticas para mejorar el rendimiento de consultas SQL.

III. MARCO TEÓRICO

La optimización del rendimiento de consultas SQL es esencial en la gestión de bases de datos, ya que busca mejorar la eficiencia y velocidad de ejecución de las consultas. Este marco teórico busca explorar los conceptos fundamentales y estrategias clave empleadas en este campo.

Identify applicable funding agency here. If none, delete this text box.

Introducción a las Consultas SQL: Las consultas SQL son la columna vertebral de la interacción con bases de datos relacionales. Aquí podemos ver una explicación más detallada sobre los fundamentos y comandos utilizados básicamente a la hora de realizar consultas:

SELECT: Utilizado para recuperar datos de una o varias tablas, La estructura básica de una consulta SELECT implica la especificación de columnas o * para seleccionar todas las columnas, la tabla de la que se extraerán los datos y posiblemente condiciones de filtrado con cláusulas WHERE.

- Ejemplo: Supongamos que tenemos una tabla llamada empleados con columnas nombre, apellido, y salario. Para obtener los nombres y apellidos de todos los empleados cuyos salarios sean mayores a $3000, utilizaríamos:

SELECT nombre, apellido FROM empleados WHERE salario > 3000;

INSERT: Permite añadir nuevos registros a una tabla, también añade un nuevo conjunto de datos a la tabla especificada, indicando las columnas y los valores que se desean insertar en esas columnas.

- Ejemplo: Si queremos agregar un nuevo empleado a la tabla empleados, podríamos hacerlo de la siguiente manera:

INSERT INTO empleados (nombre, apellido, salario) VALUES ('Juan', 'Pérez', 3500);

UPDATE: Modifica los registros existentes en una tabla, permite cambiar valores en una o varias columnas de una tabla según una condición específica. Es útil para realizar actualizaciones masivas en registros existentes.

- Ejemplo: Si queremos aumentar el salario de todos los empleados que ganan menos de $4000 en un 10%, podemos hacerlo así:

UPDATE empleados SET salario = salario * 1.10 WHERE salario < 4000;

DELETE: Elimina registros de una tabla que cumplan con ciertas condiciones, Es una acción irreversible y borra permanentemente los datos seleccionados de la tabla.

- Ejemplo: Supongamos que queremos eliminar a un empleado que ya no trabaja en la empresa, se haría de la siguiente manera:

DELETE FROM empleados WHERE nombre = 'Juan' AND apellido = 'Pérez';

Optimización de Consultas SQL: La optimización de consultas SQL es crucial para mejorar el rendimiento y la eficiencia de las operaciones realizadas en una base de datos. A continuación, mencionamos algunas de las técnicas y estrategias comunes en la optimización de consultas.

- Creación y uso eficiente de índices.
- Uso de JOINs explícitos.
- Modificar consultas para mejorar el plan de ejecución.
- Uso de Vistas Materializadas
- Seleccionar el tipo correcto de JOIN (INNER, LEFT, RIGHT, FULL) según los datos necesarios.
- Evitar el uso de OR dentro de la cláusula JOIN

Planificación y Ejecución de Consultas: el proceso de planificación y ejecución de consultas es esencial para garantizar un rendimiento óptimo en las operaciones de bases de datos. Aquí podemos ver algunos de los aspectos más importantes a tener en cuenta:

- Técnicas de optimización adicionales.
- Ejecución Eficiente.

- Selección de planes basados en estadísticas y estructuras de índices.
- Generación de Planes de Ejecución Óptimos.

Estructuras de Datos y Rendimiento: Las estructuras de datos juegan un papel crucial en el rendimiento de consultas SQL, ya que determinan cómo se almacenan, organizan y acceden a los datos en una base de datos. Aquí se detalla un poco de su influencia en el rendimiento:

- La estructura de las tablas, la cantidad de filas y columnas, y la forma en que se relacionan entre sí.
- Uso de Índices, Tablas Particionadas y Técnicas de Almacenamiento.
- La elección de algoritmos para acceder y procesar datos influye en la eficiencia de la consulta.
- Uso de estructuras optimizadas en memoria.

Desafíos y Tendencias Actuales: Esto nos lleva por último a los desafíos y también las ultimas tendencias en cuanto a la optimización de consultas SQL, en lo cual podemos apreciar que es un área que esta en constante evolución, algunos ejemplos de esto pueden ser los siguientes:

- La creciente diversidad de datos (datos no estructurados, semiestructurados) exige adaptarse a modelos de datos más flexibles, como NoSQL y grafos.
- Manejo Eficiente de Grandes Volúmenes de Datos y Optimización en Tiempo Real.
- La aplicación de técnicas de IA.
- La explotación de sistemas distribuidos y paralelos para dividir y procesar consultas en múltiples nodos.
- La necesidad de respuestas instantáneas en aplicaciones de tiempo real (por ejemplo, sistemas de transacciones financieras) demanda consultas altamente eficientes y optimización en tiempo de ejecución.

IV. DESARROLLO

En esta parte, vamos a llevar a cabo distintas técnicas de optimización de consultas para las consultas a base de datos más comunes y también a las consultas que pueden llegar a ser muy ineficaces en cuanto a tiempos de respuesta, también haremos comparaciones entre consultas optimizadas y no optimizadas para poder ver la diferencia que puede haber en los tiempos de respuesta en base de datos con distintos tamaños y de esta forma comprobar la eficacia de las técnicas de optimización y a su vez dar una pequeña guía para los desarrolladores buscando así facilitar el trabajo a la hora de realizar endpoints o distintas consultas típicas a la base de datos.

Para realizar lo mencionado anteriormente, primero vamos a crear una base de datos de prueba en el terminal, luego creamos algunas tablas mediante el uso de las consultas que explicamos anteriormente con el fin de usar esta base de datos para realizar las distintas pruebas de optimización y rendimiento, las tablas inicialmente quedarían de la siguiente manera:

EMPLEADOS:

Empleado_id	Nombre	Apellido	Cargo_id	Salario	Departamento_id
1	Juan	Pérez	1	5000	1
2	María	Gómez	2	3500	2
3	Carlos	López	3	4200	2
4	Laura	Martínez	2	3800	1
5	Pedro	Diaz	4	2500	3

CARGOS:

Cargo_id	Nombre_cargo
1	Gerente
2	Desarrollador
3	Analista
4	Asistente

DEPARTAMENTOS:

Departamento_id	Nombre_departamento
1	Ventas
2	Desarrollo
3	Recursos Humanos
4	Finanzas

Con esto podemos empezar con las técnicas de optimización de consultas, para esta primera fase veremos algunas de las técnicas más útiles de optimización y que resultados de mejora nos dan al tener una base de datos muy pequeña, esto con el fin de en las siguientes fases apreciar como los tiempos de respuestas al comparar las consultas pueden ser abismales.

FASE I

- **Uso de SELECT Especifico:** En lugar de seleccionar todas las columnas con 'SELECT *', elige solo las columnas necesarias haciendo de esta forma que se reduzca la cantidad de datos transferidos y se mejore la velocidad de la consulta, para este caso queremos traer el nombre y apellido de todos los empleados que sean del departamento de ventas, es decir el de id 1.

	CONSULTA	**RESPUESTA**
ORIGINAL	SELECT * FROM empleados WHERE departamento_id = 1;	0,0034 s
OPTIMIZADA	SELECT nombre, apellido FROM empleados WHERE departamento_id = 1;	0,0028 s

- **Uso de Índices para Consultas Frecuentes:** Identifica las columnas que se utilizan con frecuencia en cláusulas WHERE y crea índices en esas columnas para acelerar las búsquedas, esto ya que los índices permiten un acceso más rápido a los datos, para este ejemplo realizamos una búsqueda del nombre y salario del empleado el cual tenga el cargo de desarrollador o el cargo_id 2 y lo optimizamos creando un índice para ese WHERE.

	CONSULTA	**RESPUESTA**
ORIGINAL	SELECT nombre, salario FROM empleados WHERE cargo_id = 2;	0,0030 s
OPTIMIZADA	CREATE INDEX idx_cargo_id ON empleados(cargo_id); SELECT nombre, salario FROM empleados WHERE cargo_id = 2;	0,0019 s

- **Uso de Clausula LIMIT para Restringir Resultados:** La cláusula LIMIT es una instrucción que se utiliza en consultas SQL para restringir el número de filas devueltas por una consulta, es especialmente útil ya que permite limitar solamente a traer las primeras filas de una consulta que puede llegar a ser más grande, ayudando así a disminuir la cantidad de datos a cargar y el tiempo de respuesta, para este ejemplo limitaremos a 2 la cantidad de empleados que nos arrojara la consulta de todos los empleados.

	CONSULTA	**RESPUESTA**
ORIGINAL	SELECT * FROM empleados;	0,0029 s
OPTIMIZADA	SELECT * FROM empleados LIMIT 2;	0,0012 s

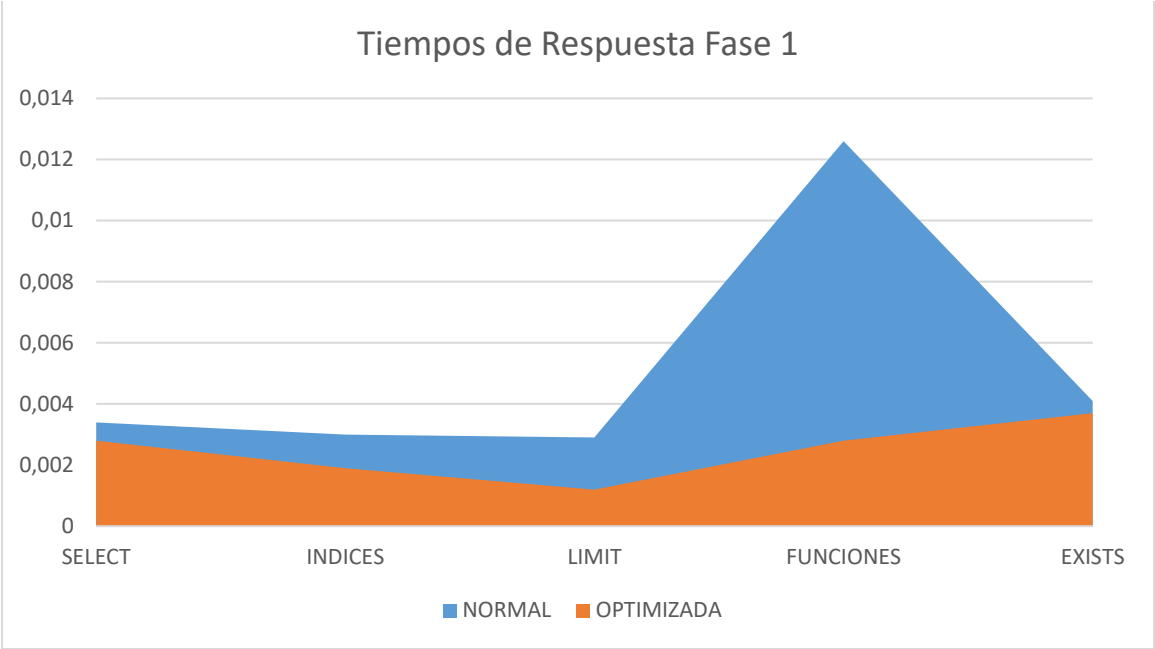
- **Evitar el Uso de Funciones en Clausulas WHERE:** el uso de funciones como podría ser ROUND() o YEAR(), puede impedir que el optimizador de consultas utilice índices eficientemente, lo que resulta en un rendimiento inferior de la consulta, para este ejemplo vamos a traer el nombre y salario del empleado que tenga un salario entre 3499.5 y 3500.5 mediante el uso de ROUND().

	CONSULTA	**RESPUESTA**
ORIGINAL	SELECT nombre, salario FROM empleados WHERE ROUND(salario) = 3500;	0,0126 s
OPTIMIZADA	SELECT nombre, salario FROM empleados WHERE salario BETWEEN 3499.5 AND 3500.5;	0,0028 s

- **Uso de Subconsultas y EXISTS:** las subconsultas y el operador EXISTS son herramientas poderosas en SQL que permiten realizar consultas complejas, filtrar datos y realizar comparaciones entre conjuntos de datos, con el podemos verificar la existencia de registros en un subconjunto de datos. Para este ejemplo queremos encontrar empleados que trabajen en un departamento especifico.

	CONSULTA	**RESPUESTA**
ORIGINAL	SELECT nombre FROM empleados WHERE departamento_id IN (SELECT departamento_id FROM departamentos WHERE nombre_departamento = 'Desarrollo')	0,0041 s
OPTIMIZADA	SELECT e.nombre, e.salario FROM empleados e JOIN departamentos d ON e.departamento_id = d.departamento_id WHERE d.nombre_departamento = 'Desarrollo'	0,0037 s

A raíz de estos resultados, podemos obtener una tabla en la cual comparamos los tiempos de respuesta obtenidos entre las consultas originales y las consultas luego de aplicar distintas técnicas de optimización, con esto podemos apreciar la gran diferencia que puede presentar la optimización en una consulta, incluso aunque sea una base de datos con muy pocos datos el porcentaje de mejora puede llegar a ser muy alto.



FASE II

Para la fase dos, utilizaremos las mismas consultas, pero incrementaremos el volumen de datos de cada una de las tablas y modificaremos un poco las consultas optimizadas para poder tener una comparación mas justa de la diferencia entre estas, el tamaño de la base de datos será algo mas realistas, teniendo alrededor de 100 filas en cada una de las tablas.

Con el fin de poblar las tablas de la base de datos usaremos PHP, con el cual podremos conectarnos a la base de datos y generar mediante un bucle la cantidad de registros deseados en la tabla especificada, diferenciándolos entre ellos al estar concatenados con un contador, para hacer más fácil la distinción, la estructura de este script seria la siguiente:

```
for ($i = 1; $i <= 100; $i++) {

    $nombreDepartamento = 'Departamento ' . $i;
    $sql = "INSERT INTO departamentos (nombre_departamento) VALUES ('$nombreDepartamento')";
    if ($conexion->query($sql) === TRUE) {
        echo "Departamento $i insertado correctamente <br>";
    } else {
        echo "Error al insertar departamento $i: " . $conexion->error . "<br>";
    }

}
```

Ya que en la fase 1 se detalló las técnicas de optimización y que se haría en cada una de ellas, en esta fase y en la siguiente solamente referenciaremos la técnica utilizada y se mostraran los tiempos de respuesta comparando la consulta original con la optimizada.

- **Uso de SELECT Especifico:**

	CONSULTA	RESPUESTA
ORIGINAL	SELECT * FROM empleados WHERE departamento_id > 50	0,0035 s
OPTIMIZADA	SELECT nombre,apellido FROM empleados WHERE departamento_id > 50	0,0027 s

- **Uso de Índices para Consultas Frecuentes:**

	CONSULTA	RESPUESTA
ORIGINAL	SELECT nombre, salario FROM empleados WHERE departamento_id > 50;	0,0034 s
OPTIMIZADA	CREATE INDEX idx_depa_id ON empleados(departamento_id); SELECT nombre, salario FROM empleados WHERE departamento_id > 50;	0,0023 s

- **Uso de Clausula LIMIT para Restringir Resultados:**

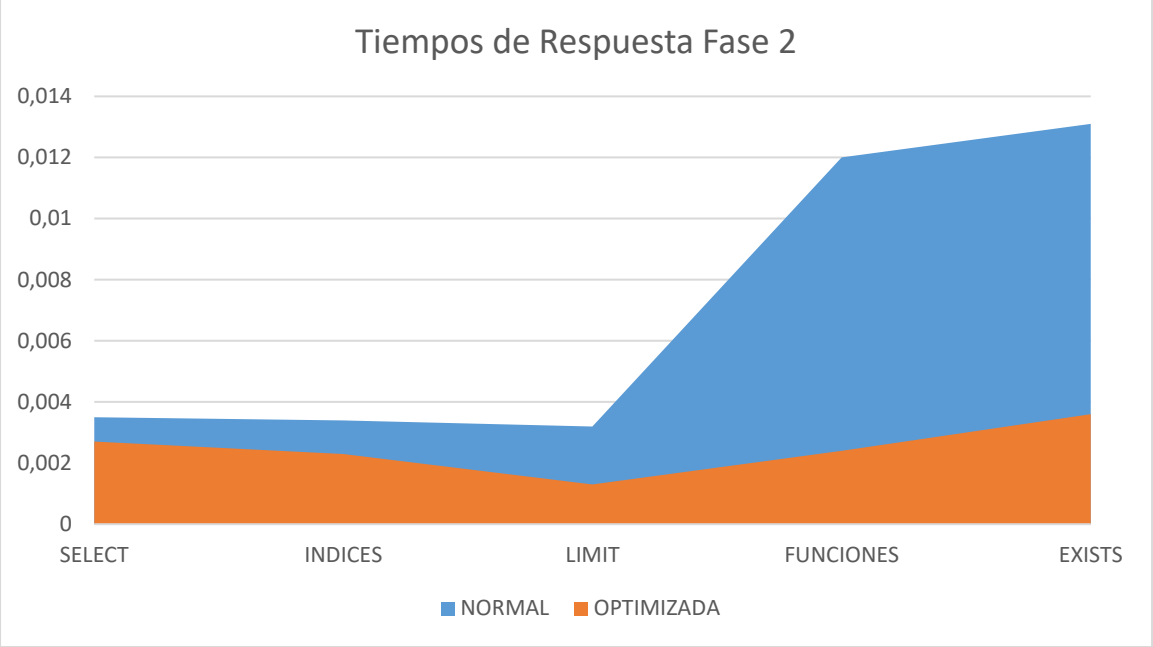
	CONSULTA	RESPUESTA
ORIGINAL	SELECT * FROM empleados;	0,0032 s
OPTIMIZADA	SELECT * FROM empleados LIMIT 20;	0,0013 s

- **Evitar el uso de Funciones en Clausulas WHERE:**

	CONSULTA	RESPUESTA
ORIGINAL	SELECT nombre, salario FROM empleados WHERE ROUND(salario * 1.1) > 50;	0,0120 s
OPTIMIZADA	SELECT nombre, salario FROM empleados WHERE salario > 50 / 1.1;	0,0024 s

- **Uso de Subconsultas y EXISTS:**

	CONSULTA	RESPUESTA
ORIGINAL	SELECT nombre FROM empleados WHERE departamento_id IN (SELECT departamento_id FROM departamentos WHERE nombre_departamento like 'Departamento%')	0,0131 s
OPTIMIZADA	SELECT e.nombre, e.salario FROM empleados e JOIN departamentos d ON e.departamento_id = d.departamento_id WHERE d.nombre_departamento like 'Departamento%'	0,0036 s



FASE III

Para esta ultima fase, incrementamos el tamaño de la base de datos a 10.000 registros por tabla, esto para evaluar los tiempos de respuesta que podría dar en un sistema real en producción al utilizar consultas sencillas a comparación con las consultas optimizadas:

- **Uso de SELECT Especifico:**

	CONSULTA	**RESPUESTA**
ORIGINAL	SELECT * FROM empleados WHERE departamento_id > 5000	0,0116 s
OPTIMIZADA	SELECT nombre,apellido FROM empleados WHERE departamento_id > 50	0,0038 s

- **Uso de Índices para Consultas Frecuentes:**

	CONSULTA	**RESPUESTA**
ORIGINAL	SELECT nombre, salario FROM empleados WHERE departamento_id > 50;	0,0134 s
OPTIMIZADA	CREATE INDEX idx_depa_id ON empleados(departamento_id); SELECT nombre, salario FROM empleados WHERE departamento_id > 50;	0,0022 s

- **Uso de Clausula LIMIT para Restringir Resultados:**

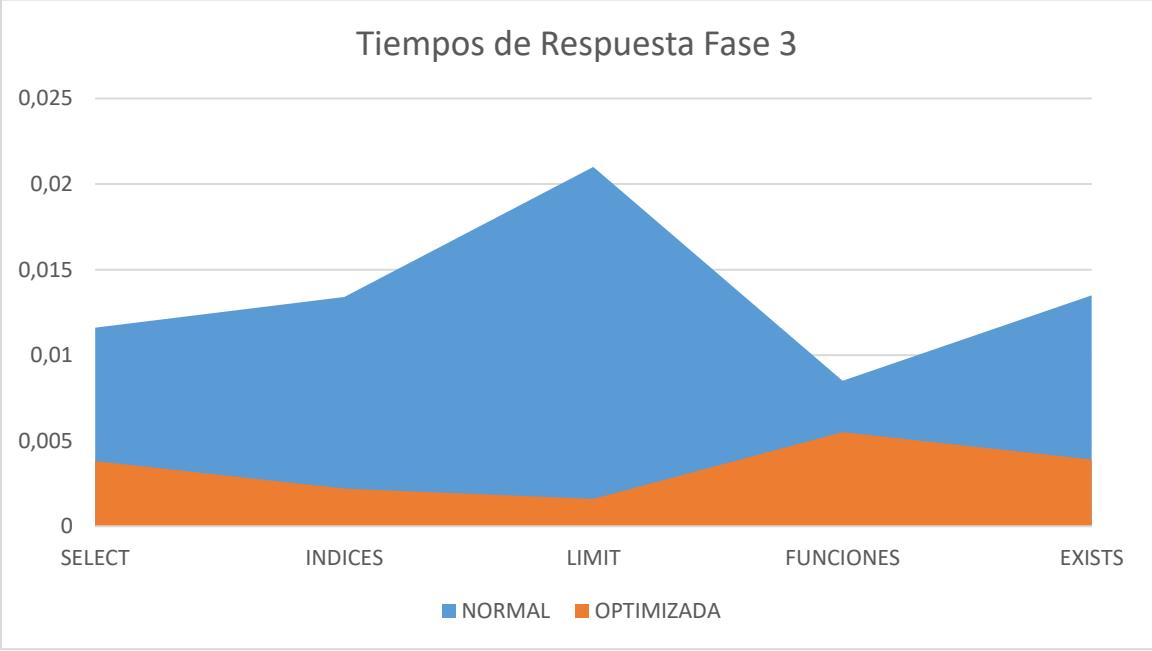
	CONSULTA	RESPUESTA
ORIGINAL	SELECT * FROM empleados;	0,0210 s
OPTIMIZADA	SELECT * FROM empleados LIMIT 100;	0,0016 s

- **Evitar el uso de Funciones en Clausulas WHERE:**

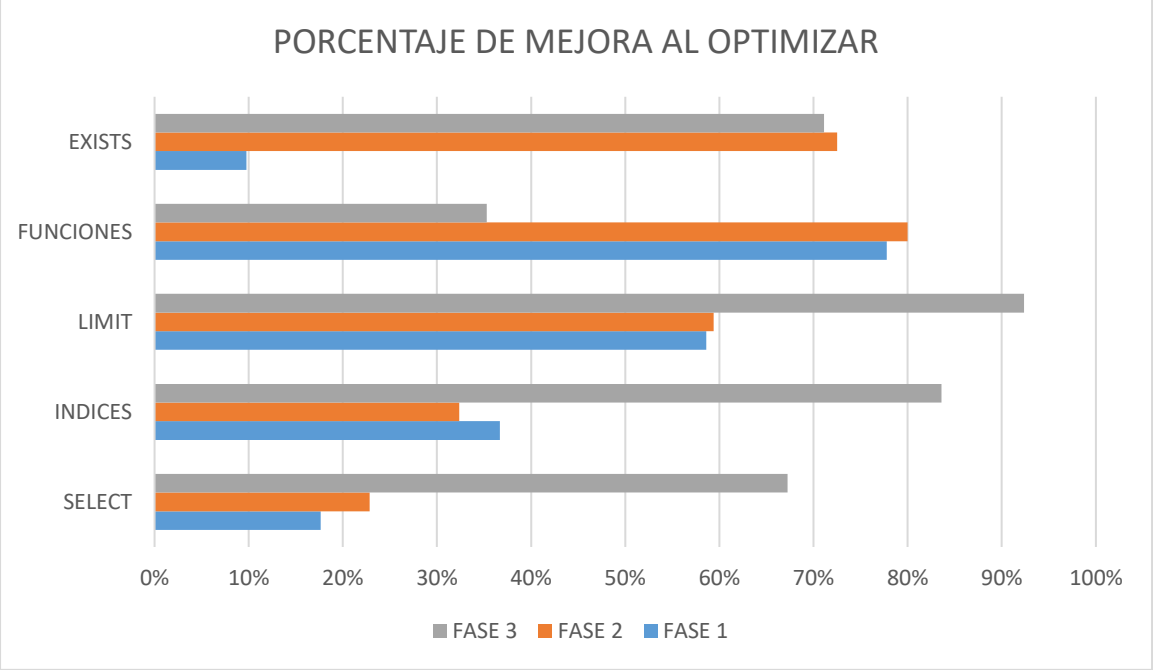
	CONSULTA	RESPUESTA
ORIGINAL	SELECT nombre, salario FROM empleados WHERE ROUND(salario * 1.1) > 5000;	0,0085 s
OPTIMIZADA	SELECT nombre, salario FROM empleados WHERE salario > 5000 / 1.1;	0,0055 s

- **Uso de Subconsultas y EXISTS:**

	CONSULTA	RESPUESTA
ORIGINAL	SELECT nombre FROM empleados WHERE departamento_id IN (SELECT departamento_id FROM departamentos WHERE nombre_departamento like 'Departamento%')	0,0135 s
OPTIMIZADA	SELECT e.nombre, e.salario FROM empleados e JOIN departamentos d ON e.departamento_id = d.departamento_id WHERE d.nombre_departamento like 'Departamento%'	0,0039 s



También, para finalizar podemos ver un grafico en el que se muestra el porcentaje de mejora al aplicar las técnicas de optimización en cada una de las consultas:



V. CONCLUSIONES

Gracias a esta investigación pudimos descubrir que el uso de técnicas de optimización en consultas SQL tienen un impacto significativo en los tiempos de respuesta, logrando tener márgenes de mejora en los tiempos de respuesta de hasta un 92% al aplicar distintas técnicas, lo cual puede favorecer muy positivamente tanto a aplicaciones con volúmenes de datos bajos como a aquellas que tienen grandes flujos de información.

También, podemos destacar que la optimización es de vital importancia ya que puede mejorar de gran manera la experiencia de los usuarios al mejorar la velocidad de las aplicaciones y que para lograr esto se deben aplicar estrategias de optimización especificas a las necesidades de la consulta estructura de datos, pero pese a esto en esta investigación pudimos tratar algunas de las técnicas aplicadas a las consultas que se usan de manera más frecuentes, de esta manera ayudando a dar pautas para optimizar la mayoría de casos

Las soluciones implementadas en las consultas no solamente funcionan para mejorar el rendimiento actual de la aplicación, sino que son fundamentales para mantener la eficiencia en las respuestas a medida que la base de datos escala y se enfrenta a volúmenes crecientes de información, esto lo podemos ver reflejado en el ultimo grafico de PORCENTAJE DE MEJORA AL OPTIMIZAR, ya que en este vemos que, al aumentar la cantidad de datos, la mejora en tiempos de respuesta cada vez es mayor.

Por último podemos resaltar que la optimización de estas consultas no solamente ayuda a mejorar tiempos de respuesta, ya que también puede llegar a reducir costos en los gastos de infraestructura, al reducir los tiempos de respuesta y necesitar menor cantidad de fuerza computacional para ejecutarlos.

VI. REFERENCIAS

[1] *MySQL 5: Optimización" por Arjen Lentz, Peter Brawley, y Baron Schwartz. (Anaya Multimedia)*

[2] Documentación oficial de MySQL: https://dev.mysql.com/doc/refman/5.7/es/optimization.html

[3] Wiki de PostgreSQL: https://wiki.postgresql.org/wiki/Espa%C3%B1ol

[4] SQL Shack: https://www.sqlshack.com/es/tecnicas-de-optimizacion-de-consultas-en-sql-server-consejos-y-trucos-de-aplicacion/

[5] GPS Open Source: https://www.gpsos.es/2021/09/optimizar-consultas-sql-con-estos-6-consejos/

[6] HCL Software: https://help.hcltechsw.com/commerce/9.1.0/es/developer/refs/rsdperformanceworkspaces.html