Facultad de Informática
UNIVERSIDAD NACIONAL DE LA PLATA

Evaluación de performance en bases de datos relacionales

La Plata, Buenos Aires, Argentina 16 de febrero de 2017

Autor

Graneros Nicolás

Directores

Mg Pablo Thomas

Mg Rodolfo Bertone

Índice de contenido

Capítulo 1: Introducción

1.1 Motivación

El término bases de datos fue escuchado por primera vez en un simposio celebrado en California en 1963.

En una primera aproximación, se puede decir que una base de datos es un conjunto de información relacionada que se encuentra agrupada o estructurada [1].

Desde el punto de vista informático, una base de datos es un sistema formado por un conjunto de datos almacenados en discos que permiten el acceso directo a ellos y un conjunto de programas que manipulen ese conjunto de datos.

Por su parte, un sistema de Gestión de Bases de datos es un tipo de software muy específico dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan; o lo que es lo mismo, una agrupación de programas que sirven para definir, construir y manipular una base de datos, permitiendo así almacenar y posteriormente acceder a los datos de forma rápida y estructurada.

Desde fines del siglo pasado, las bases de datos están teniendo un impacto decisivo sobre los sistemas de información.

Los orígenes de las bases de datos se remontan a la Antigüedad donde ya existían bibliotecas y toda clase de registros. Además también se utilizaban para recoger información sobre las cosechas y censos. Sin embargo, su búsqueda era lenta y poco eficaz y no se contaba con la ayuda de máquinas que pudiesen reemplazar el trabajo manual.

Posteriormente, el uso de las bases de datos se desarrolló a partir de las necesidades de almacenar grandes cantidades de información o datos. Sobre todo, desde la aparición de las primeras computadoras, el concepto de bases de datos ha estado siempre ligado a la informática.

Por su parte, a principios de los años ochenta comenzó el auge de la comercialización de los sistemas relacionales, y SQL comenzó a ser el estándar de la industria, ya que las bases de datos relacionales con su sistema de tablas (compuesta por filas y columnas) pudieron competir con las bases jerárquicas y de red, como consecuencia de que su nivel de programación era sencillo y su nivel de programación era relativamente bajo.

Con el pasar de los tiempos y la necesidad de almacenar cada vez más información, las bases de datos se vieron obligadas a ser más eficientes tanto en la persistencia de datos como en la recuperación de los mismos, es por esto que se

quiere poner a prueba que la semántica de las consultas sql cumplen un rol fundamental a la hora de recuperar datos.

1.2 Objetivo

El objetivo central de esta tesina de grado es el análisis de distintos casos de ejecución de consultas SQL, en un ambiente dedicado con diferentes sistemas de gestión de motores de base de datos (MSSQL Server y MySql). La complejidad que pueden alcanzar algunas consultas puede ser tal, que el diseño de una consulta puede tomar un tiempo considerable, obteniendo no siempre una respuesta óptima.

El propósito fundamental de este análisis es poner a prueba los aspectos teóricos de la ejecución de consultas SQL con lo que es la práctica real de las mismas. Se espera que los diferentes casos de estudios, en otras palabras cláusulas SQL distintas, retornen tiempos de ejecución diferentes para un mismo requerimiento.

Por último, se comparara los casos de estudio entre los diferentes sistemas de gestión de base de datos y se determinara cual se comporta mejor en este ambiente dedicado.

Capítulo 2: Marco Contextual

2.1 Base de datos relacionales.

El término base de datos, fue usado por primera vez en 1963 en un simposio en California, USA. En 1970 Edgar Frank Codd, de los laboratorios IBM en San José (California), propuso el modelo relacional, este no tardó en consolidarse como un nuevo paradigma en los modelos de base de datos [8].

En 1980 las base de datos relacionales logran posicionarse en el mercado de base de datos con sus sistema de tablas, filas, columnas, además se dan diversas investigaciones paralelas como las base de datos orientada a Objetos.

SQL (por sus siglas en inglés *Structured Query Language*; en español *lenguaje de consulta estructurada*) es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en ellas. Una de sus características es el manejo del álgebra y el cálculo relacional que permiten efectuar consultas con el fin de recuperar, de forma sencilla, información de las bases de datos, así como hacer cambios en ellas.

SQL es un lenguaje de acceso a bases de datos que explota la flexibilidad y potencia de los sistemas relacionales y permite así gran variedad de operaciones [2].Es un lenguaje declarativo de "alto nivel" o "de no procedimiento" que, gracias a su fuerte base teórica y su orientación al manejo de conjuntos de registros y no a registros individuales permite una alta productividad en codificación y la orientación a objetos. De esta forma, una sola sentencia puede equivaler a uno o más programas que se utilizarían en un lenguaje de bajo nivel orientado a registros.

Como ya se dijo anteriormente, y suele ser común en los lenguajes de acceso a bases de datos de alto nivel, SQL es un lenguaje declarativo. O sea, que especifica qué es lo que se quiere y no cómo conseguirlo, por lo que una sentencia no establece explícitamente un orden de ejecución.

El orden de ejecución interno de una sentencia puede afectar en gran medida a la eficiencia del Sistema de gestión de base datos, por lo que se hace necesario que éste lleve a cabo una optimización antes de su ejecución. Muchas veces, el uso de índices acelera una instrucción de consulta, pero ralentiza la actualización de los datos. Dependiendo del uso de la aplicación, se priorizará el acceso indexado o una

rápida actualización de la información. La optimización difiere sensiblemente en cada motor de base de datos y depende de muchos factores.

2.2 Sistema de manejo de bases de datos relacional Microsoft SQL.

Microsoft SQL Server es un sistema de manejo de bases de datos del modelo relacional, desarrollado por la empresa Microsoft [9].

El lenguaje de desarrollo utilizado (por línea de comandos o mediante la interfaz gráfica de Management Studio) es Transact-SQL (TSQL), una implementación del estándar ANSI del lenguaje SQL, utilizado para manipular y recuperar datos, crear tablas y definir relaciones entre ellas.

Puede ser configurado para utilizar varias instancias en el mismo servidor físico, la primera instalación lleva generalmente el nombre del servidor, y los siguientes nombres específicos (con un guion invertido entre el nombre del servidor y el nombre de la instalación).

El código fuente original de SQL Server que fue utilizado en las versiones previas a la versión 7.0, pero fue actualizado en las versiones 7.0 y 2000, y reescrito en la versión 2005. Generalmente, cada dos o tres años, una nueva versión es lanzada y, entre estos lanzamientos, se proponen services packs con mejoras y correcciones de bugs, y hotfixes por problemas urgentes en el sistema de seguridad o bugs críticos.

Este sistema incluye una versión reducida, llamada MSDE con el mismo motor de base de datos pero orientado a proyectos más pequeños, que en sus versiones 2005 y 2008 pasa a ser el SQL Express Edition, que se distribuye en forma gratuita.

2.2.1 Índices

Un índice es una estructura de disco asociada con una tabla que acelera la recuperación de filas de la tabla. Contiene claves generadas a partir de una o varias columnas de la tabla. Dichas claves están almacenadas en una estructura (árbol b o alguna variante) que permite que SQL Server busque de forma rápida y eficiente la fila o filas asociadas a los valores de cada clave [5].

Una tabla puede contener los siguientes tipos de índices:

Agrupado

- Los índices agrupados ordenan y almacenan las filas de los datos de la tabla de acuerdo con los valores de la clave del índice. Son columnas incluidas en la definición del índice. Sólo puede haber un índice clúster por cada tabla, porque las filas de datos sólo pueden estar ordenadas de una forma.
- La única ocasión en la que las filas de datos de una tabla están ordenadas es cuando la tabla contiene un índice agrupado. Cuando una tabla tiene un índice agrupado, la tabla se denomina tabla agrupada. Si una tabla no tiene un índice agrupado, sus filas de datos están almacenadas en una estructura sin ordenar denominada montón.

No agrupado

- Los índices no agrupados tienen una estructura separada de las filas de datos. Un índice no agrupado contiene los valores de clave de índice no agrupado y cada entrada de valor de clave tiene un puntero a la fila de datos que contiene el valor clave.
- El puntero de una fila de índice no agrupado hacia una fila de datos se denomina localizador de fila. La estructura del localizador de filas depende de si las páginas de datos están almacenadas en un montón o en una tabla agrupada. Si están en un montón, el localizador de filas es un puntero hacia la fila. Si están en una tabla agrupada, el localizador de fila es la clave de índice agrupada.
- Puede agregar columnas sin clave al nivel hoja de un índice no agrupado con el fin de eludir los límites existentes para las claves de índice, 900 bytes y columnas de 16 claves, así como para ejecutar consultas indizadas y totalmente cubiertas

Los índices bien diseñados pueden reducir las operaciones de E/S de disco y consumen menos recursos del sistema, con lo que mejoran el rendimiento de la consulta. Los índices pueden ser útiles para diversas consultas que contienen instrucciones *SELECT, UPDATE, DELETE o MERGE*.

El optimizador de consultas normalmente selecciona el método más eficaz cuando ejecuta consultas. No obstante, si no hay índices disponibles, el optimizador de consultas debe utilizar un recorrido de la tabla. Su tarea consiste en diseñar y crear los índices más apropiados para su entorno de forma que el optimizador de consultas disponga de una selección de índices eficaces entre los que elegir. SQL Server proporciona el Asistente para la optimización de motor de base de datos como ayuda en el análisis del entorno de la base de datos y en la selección de los índices adecuados.

2.2.2 Plan de ejecución

Simbología del plan de ejecución

 Cluster Index seek (Búsqueda en índice cluster): El operador Clustered Index Seek Utiliza la estructura de árbol b del índice para buscar directamente los registros coincidentes, el tiempo que toma es proporcional número de tuplas coincidentes.

 Cluster Index scan (Examen de índice cluster): El operador Clustered Index Scan lee todo el índice cluster buscando coincidencias, el tiempo que toma en realizar esta operación es proporcional al tamaño del índice.

En general, un Cluster Index seek es preferible a Cluster Index Scan (cuando el número de registros coincidentes es mucho menor que el número total de registros), ya que el tiempo que se tarda en realizar una búsqueda de índice es constante independientemente del número total de tuplas que posea la tabla

Sin embargo, en ciertas situaciones un Clustered Index Scan puede ser más rápido que un Cluster Index Seek (a veces significativamente más rápido), normalmente cuando la tabla es muy pequeña o cuando un gran porcentaje de los registros coincide con el filt3ro del where.

 Top: El operador Top recorre la entrada y sólo devuelve el primer número o porcentaje especificado de filas, basándose en un criterio de ordenación si es posible.

 Nested Loop: El operador Nested Loops realiza las operaciones lógicas combinación interna, combinación externa izquierda, semicombinación izquierda y anti semicombinación.

Las combinaciones de bucles anidados buscan en la tabla interna cada fila de la tabla externa, normalmente mediante un índice. Microsoft SQL Server decide, en función de los costos anticipados, si debe ordenar o no la entrada externa para mejorar la ubicación de las búsquedas en el índice de la entrada interna.

 Select : El operador Select incluye los datos devueltos al final de un plan de consulta. Normalmente, se trata del elemento raíz de un plan de ejecución.

 Merge Join:El operador Merge Join realiza las siguientes operaciones lógicas de combinación interna (inner join), combinación externa izquierda (left outer join), semicombinación izquierda (left semi join), antisemicombinación izquierda (left anti semi join), combinación externa derecha (right outer join), semicombinación derecha (right semi join), antisemicombinación derecha (right anti semi join) y unión (union).

 Filter :El operador Filter recorre la entrada y sólo devuelve las filas que cumplen la expresión del filtro (predicado) que aparece en la columna

 Nonclustered Index Seek (Búsqueda de índice no clustered) :Idem a Clustered Index Seek, pero con la diferencia que utiliza un índice Nonclustered.

 Nonclustered Index Scan(Examen de índice no clustered) :Idem a Clustered Index Scan, pero con la diferencia que utiliza un índice Nonclustered.

 Stream Aggregate: El operador Stream Aggregate agrupa las filas por una o varias columnas y, a continuación, calcula una o varias expresiones agregadas devueltas por la consulta. Los operadores posteriores de la consulta pueden hacer referencia al resultado de este operador, devolverse al cliente, o ambas cosas.

2.3 Sistema de manejo de bases de datos relacional MYSQL.

MySQL es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual GPL/Licencia comercial por Oracle Corporation y está considerada como la base datos open source más popular del mundo , y una de las más populares en general junto a Oracle y Microsoft SQL Server, sobre todo para entornos de desarrollo web.

MySQL fue inicialmente desarrollado por MySQL AB (empresa fundada por David Axmark, Allan Larsson y Michael Widenius). MySQL A.B. fue adquirida por Sun Microsystems en 2008, y ésta a su vez fue comprada por Oracle Corporation en 2010, la cual ya era dueña desde 2005 de Innobase Oy, empresa finlandesa desarrolladora del motor InnoDB para MySQL.

Al contrario de proyectos como Apache, donde el software es desarrollado por una comunidad pública y los derechos de autor del código están en poder del autor individual, MySQL es patrocinado por una empresa privada, que posee el copyright de la mayor parte del código. Esto es lo que posibilita el esquema de doble licenciamiento anteriormente mencionado. Los SGBD se distribuyen en varias versiones, una *Community*, distribuida bajo la Licencia pública general de GNU, versión 2, y varias versiones *Enterprise*, para aquellas empresas que quieran incorporarlo en productos privativos. Las versiones *Enterprise* incluyen productos o servicios adicionales tales como herramientas de monitorización y soporte oficial.

2.3.1 Índices

Al igual que en el SGBD de Microsoft, los índices en MySQL permiten localizar y devolver registros de una forma sencilla y rápida. Son especialmente útiles cuando se quiere buscar elementos de entre los millones y hasta billones de registros que puede contener una tabla en cierto momento [10].

MySQL presenta distintos tipos de índices que se pueden crear y las condiciones que deben cumplir cada uno de ellos:

- INDEX (NON-UNIQUE): este tipo de índice se refiere a un índice normal, no único. Esto implica que admite valores duplicados para la columna (o columnas) que componen el índice. No aplica ninguna restricción especial a los datos de la columna (o columnas) que componen el índice sino que se emplea simplemente para mejorar el tiempo de ejecución de las consultas.

- UNIQUE: este tipo de índice se refiere a un índice en el que todas las columnas deben tener un valor único. Esto implica que no admite valores duplicados para la columna (o columnas) que componen el índice. Aplica la restricción de que los datos de la columna (o columnas) deben tener un valor único.
- PRIMARY: este tipo de índice se refiere a un índice en el que todas las columnas deben tener un valor único (al igual que en el caso del índice UNIQUE) pero con la limitación de que sólo puede existir un índice PRIMARY en cada una de las tablas. Aplica la restricción de que los datos de la columna (o columnas) deben tener un valor único.
- FULLTEXT: estos índices se emplean para realizar búsquedas sobre texto (CHAR, VARCHAR y TEXT). Estos índices se componen por todas las palabras que están contenidas en la columna (o columnas) que contienen el índice. No aplica ninguna restricción especial a los datos de la columna (o columnas) que componen el índice sino que se emplea simplemente para mejorar el tiempo de ejecución de las consultas. Este tipo de índices sólo están soportados por InnoDB y MyISAM en MySQL 5.7.
- SPATIAL: estos índices se emplean para realizar búsquedas sobre datos que componen formas geométricas representadas en el espacio. Este tipo de índices sólo están soportados por InnoDB y MyISAM en MySQL 5.7.

Es importante destacar que todos estos índices pueden construirse empleando una o más columnas. Del mismo modo, el orden de las columnas que se especifique al construir el orden es relevante para todos los índices menos para el FULLTEXT (ya que este índice mira en TODAS las columnas que componen el índice).

MySQL presenta distintos tipos de estructuras para almacenar los índices

- B-TREE: este tipo de índice se usa para comparaciones del tipo =, >, <, >=, <=, BETWEEN y LIKE (siempre y cuando se utilice sobre constantes que no empiecen por %). Para realizar búsquedas empleando este tipo de índice, se empleará cualquier columna (o conjunto de columnas) que formen el prefijo del índice. Por ejemplo: si un índice está formado por las columnas [A, B, C], se podrán realizar búsquedas sobre: [A], [A, B] o [A, B, C].
- HASH: este tipo de índice sólo se usa para comparaciones del tipo = o <=>. Para este tipo de operaciones son muy rápidos en comparación a otro tipo de estructura. Para realizar búsquedas empleando este tipo de índice, se emplarán todas las columnas que componen el índice.

Un índice puede ser almacenado en cualquier tipo de estructura pero, en función del uso del mismo, puede interesar crear el índice en un tipo determinado de estructura o en otro. Por norma general, un índice siempre se creará con la estructura de B-TREE, ya que es la estructura más empleada por la mayoría de operaciones.

2.3.2 Plan de ejecución

La sentencia EXPLAIN devuelve una tabla con una serie de filas con información sobre cada una de las tablas empleadas en la consulta a la que acompaña. EXPLAIN se puede emplear en las siguientes consultas: SELECT, DELETE, INSERT, REPLACE y UPDATE.

La tabla que devuelve la sentencia EXPLAIN lista las tablas en el orden en el que MySQL las leería procesando la consulta (la primera fila que aparece sería la primera tabla que se lee y la última fila sería la última tabla que sería leída). Este orden es importante conocerlo ya que indicará el plan de ejecución de la consulta y, por tanto, información relevante para realizar nuestras optimizaciones.

De forma adicional, EXPLAIN se puede combinar con EXTENDED para obtener más información del plan de ejecución. EXTENDED proporciona una columna más a la salida producida por EXPLAIN. Esta columna será la de "filtered" e indicará el porcentaje aproximado de filas que han sido filtradas por la condición empleada en la tabla. Para emplear esta sentencia, se deberá poner EXPLAIN EXTENDED seguido de la consulta que será analizada.

Como se mencionó anteriormente, la sentencia EXPLAIN devuelve una tabla con una serie de filas con información sobre cada una de las tablas empleadas en la consulta a la que acompaña. Esta tabla estará compuesta por una serie de columnas que proporcionará información relevante sobre el plan de ejecución de dicha consulta.

A continuación, se describe cada una de esas columnas:

- id: muestra el número secuencial que identificará cada una de las tablas que se procesan en la consulta realizada.
- select_type: muestra el tipo de SELECT que se ha ejecutado. En función del tipo de SELECT nos permitirá identificar qué tipo de consulta se trata. Existen varios tipos distintos: SIMPLE, PRIMARY, UNION, DERIVED…
- table: muestra el nombre de la tabla a la que se refiere la información de la fila. También, puede ser alguno de los siguientes valores:
    - tabla resultante de la unión de las tablas cuyos campos id son M y N.
    - tabla resultante de una tabla derivada cuyo id es N. Una tabla derivada puede ser, por ejemplo, una subconsulta en la cláusula FROM.

    - tabla resultante de una subconsulta materializada cuyo id es N.

- partitions: muestra las particiones cuyos registros coinciden con los de la consulta. Esta columna sólo se muestra cuando se emplea EXPLAIN PARTITIONS.
- type: muestra el tipo de unión que se ha empleado.
- possible_keys: muestra qué índices puede escoger MySQL para buscar las filas de la tabla. Si esta columna es NULL, significa que no hay índices que puedan ser usados para mejorar la consulta. En este caso, podría ser interesante examinar las columnas empleadas en el WHERE para analizar si hay alguna columna que pueda emplearse para construir un índice.
- key: muestra el índice que MySQL ha decido usar para ejecutar la consulta. Es posible que el nombre del índice no esté presente en la lista de possible_keys.
- key_len: muestra el tamaño del índice que MySQL ha decidido usar para ejecutar la consulta.
- ref: muestra qué columnas o constantes son comparadas con el índice especificado en la columna key.
- rows: muestra el número de filas que MySQL cree que deben ser examinadas para ejecutar la consulta. Este número es un número aproximado.
- filtered: muestra el porcentaje estimado de filas que serán filtradas por la condición de la consulta. Esta columna sólo se muestra cuando se emplea EXPLAIN EXTENDED.
- Extra: muestra información adicional sobre cómo MySQL ejecuta la consulta.

2.4 Base de datos orientadas a objetos.

Una base de datos orientada a objetos es una base de datos donde los elementos son objetos. El objetivo de una base de datos orientada a objetos son los mismos que los de las bases de datos relaciones, pero con la ventaja de representar los modelos de datos con un marco mucho más eficiente, manteniendo la integridad y relación entre ellos.

Se debe recordar que un objeto es una estructura que tiene asociado un estado y un comportamiento (propiedades y métodos). Estas bases tienen las características de todo lo que es orientado a objeto que son Herencia, Polimorfismo, Abstracción y Encapsulamiento.

Un objeto puede heredar comportamiento de otro tipo de objetos (herencia) y puede adaptarse para responder de diferentes maneras ante la solicitud de una acción (polimorfismo), lo importante es que permite representar cosas de la vida real con relativa facilidad (abstracción) y que todo esto se puede implementar de

manera que no nos importe el código, sino sólo la manera de comunicarnos con estos objetos pensando en ellos como una sola unidad (encapsulamiento).

Las bases de datos orientados a objetos han adoptado muchos de los objetos creados para los lenguajes de programación orientados a objetos.

La utilización de una base de datos orientada a objetos simplifica la conceptualización ya que la utilización de objetos permite representar de una manera más natural la información que se quiere guardar.

Para modelar la estructura o vista lógica de la base de datos, se utiliza el Diagrama de clases que permite presentar las clases con sus respectivas relaciones estructurales y de herencia, además del Diagrama de Objetos cuando no está muy claro y preciso cómo serían las instancias de las clases o para especificar más el Diagrama de Clases.

Las bases de datos orientadas a objetos poseen las siguientes características y ventajas con respecto a la una base de datos relacional

- *Mandatorias*: son las que el Sistema debe satisfacer a orden de tener un sistema de BDOO y estos son: Objetos complejos, Identidad de Objetos, Encapsulación, Tipos o clases, Sobre paso con unión retardada, Extensibilidad, Completación Computacional, Persistencia y Manejador de almacenamiento secundario, Concurrencia, Recuperación y Facilidad de Query.
- *Opcional*: Son las que pueden ser añadidas para hacer el sistema mejor pero que no son Mandatorias, estas son de: herencia múltiple, chequeo de tipos e inferencia de distribución y diseño de transacciones y versiones.
- *Abiertas*: Son los puntos donde el diseñador puede hacer un número de opciones y estas son el paradigma de la programación, la representación del sistema o el tipo de sistema y su uniformidad. Hemos tomado una posición no muy a la expectativa para tener una palabra final más bien para proveer un punto de orientación para un debate futuro.

Está su flexibilidad, y soporte para el manejo de tipos de datos complejos. Ya que puedo tener clases y subclases creadas por ejemplo una base de clientes puede tener una subclase de la referencia de este cliente y esta heredara todos sus atributos y característica de la clase original.

La segunda ventaja de una base orientada a objetos, es que manipula datos complejos en forma rápida y ágilmente. La estructura de la base de datos está dada por referencias (o apuntadores lógicos) entre objetos.

Así como tienen sus ventajas, las bases de datos orientadas a objetos tienen sus desventajas de las cuales se destacan las siguientes.

Al considerar la adopción de la tecnología orientada a objetos, la inmadurez del mercado de BDOO constituye una posible fuente de problemas. Hay muy pocos manejadores de base de datos en el mercado que soporten este tipo de arquitectura Algunos de los pocos OOSGBD que existen son:

- Db4o (Database for Objects)
- Informix
- Bdoviedo3
- Versant

Quizá esta sea una de las causas por las cuales las base de datos orientadas a objetos aún no tengan ese crecimiento que en algún momento tantas expectativas generaron. Es por esta causa que la tesina de grado se decidió hacerla con base de datos relacionales.

Las base de datos orientadas a objetos permiten que los objetos hagan referencia directamente a otro mediante apuntadores suaves también conocido como propiedades de navegación. Esto hace que las BDOO pasen más rápido del objeto A al objeto B que las base de datos relacionales, las cuales deben utilizar comandos JOIN para lograr esto. Incluso el JOIN optimizado es más lento que un recorrido de los objetos. Así, incluso sin alguna afinación especial, una base de datos orientada a objetos es en general más rápida en esta mecánica de caza-apuntadores.

Las base de datos orientadas a objetos hacen que el agrupamiento sea más eficiente. La mayoría de los sistemas de bases de datos permiten que el operador coloque cerca las estructuras relacionadas entre sí, en el espacio de almacenamiento en disco. Esto reduce en forma radical el tiempo de recuperación de los datos relacionados, puesto que todos los datos se leen con una lectura de disco en vez de varias.

Capítulo 3: SGBD y descripción general de la base de datos utilizada

3.1 Versión de los sistemas de gestión de base de datos utilizados.

En esta tesina de grado se utilizaron las siguientes versiones.

- Microsoft SQL Server 2008 R2 – Express Edition.
- MySQL Server Version 5.5.53.

También se utilizó la herramienta MySQL Worbench en su versión 6.3 , la cual es una herramienta visual de diseño de bases de datos que integra desarrollo de software, administración de bases de datos, diseño de bases de datos, creación y mantenimiento para el sistema de base de datos MySQL

3.2 Arquitectura del ambiente

- Procesador Intel Core I5 M460 2.53 ghz
- Memoria Ram 8gb
- Sistema Operativo Windows 7.

3.3 Base de datos del problema analizado

3.3.1 Descripción General

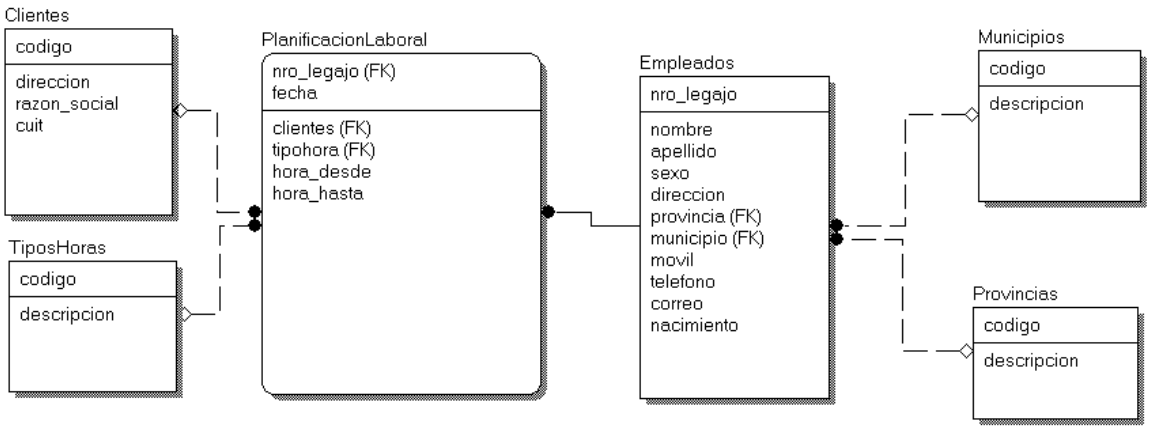
El objetivo de la base de datos que se utiliza para el análisis, es representar un sistema que administre la planificación laboral de los empleados de cierta empresa, dicha empresa posee una alta cantidad de empleados, lo cual produce una alta cantidad de tuplas en las tablas de su sistema.

Su tamaño es variable dependiendo del sistema de gestión de base de datos, en Microsoft SQL Server posee un total de 5890,88 MB mientras que en MySQL Server posee un total 3500 MB, es importante destacar que

ambas base de datos son exactamente iguales en ambos SGBD, y al momento de su creación no poseen índices de usuario, en base a estos valores se puede afirmar que en este ambiente dedicado, MySQL Server realiza una mejor uso del espacio de almacenamiento. Esta diferencia de almacenamiento está dada por la forma en que cada SGDB procesa y almacena los datos.

La base datos posee un total de 7 tablas, donde el tamaño de la base de datos esta dado principalmente por las tablas de Empleados y la tabla de PlanificacionLaboral.

3.4 Modelo Lógico de datos



3.5 Tabla clientes

Tabla que contiene datos referentes a los clientes, posee una clave numérica que identifica al cliente.

Estructura física:

```
[codigo] [int] NOT NULL,
[razon_social] [varchar](600) NOT NULL,
[direccion] [varchar](600) NOT NULL,
[cuit] [varchar](12) NOT NULL,
PRIMARY KEY [codigo]
```

Tamaño sin índices: 8 Kb MSSQL

16 kb MySQL

Total de tuplas: 40

3.6 Tabla Empleados

Tabla que contiene datos referentes a los empleados, posee una clave numérica que identifica al empleado.

Estructura física:

```
[nro_legajo] [int] NOT NULL,
[nombre] [varchar](100) NOT NULL,
[apellido] [varchar](100) NULL,
[sexo] [char](1) NULL,
[direccion] [varchar](600) NULL,
[municipio] [varchar](600) NULL,
[provincia] [varchar](600) NULL,
[movil] [varchar](50) NULL,
[telefono] [varchar](50) NULL,
[correo] [varchar](50) NULL,
[nacimiento] [date] NULL,
PRIMARY KEY [nro_legajo]
```

Tamaño sin índices: 135.211 MB Microsoft SQL Server

130.7 MB MySQL Server

Total de tuplas: 1000000

3.7 Tabla Municipios

Tabla que contiene datos referentes a los municipios, posee una clave varchar que identifica al municipio.

Estructura física:

```
[codigo] [varchar](600) NOT NULL,
[descripcion] [varchar](600) NOT NULL,
CONSTRAINT [PK_Municipios]
PRIMARY KEY [codigo]
```

Tamaño sin índices: 16 kb MSSQL

16 kb MySQL

Total de tuplas: 135

3.8 Tabla Provincias

Tabla que contiene datos referentes a las provincias, posee una clave varchar que identifica a la provincia.

Estructura física:

```
[codigo] [varchar](600) NOT NULL,
[descripcion] [varchar](60) NOT NULL,
PRIMARY KEY [codigo]
```

Tamaño sin índices: 8 kb MSSQL

16 kb MySQL

Total de tuplas: 23

3.9 Tabla TiposHoras

Tabla que contiene datos referentes a los tipos de hora que realiza el empleado como por ejemplo hora normal, ausencia, curso, etc posee una clave numérica que identifica el tipo de hora.

Estructura física:

```
[codigo] [int] NOT NULL,
[descripcion] [varchar](60) NOT NULL,
PRIMARY KEY [codigo]
```

Tamaño sin índices: 8 Kb MSSQL

16 kb MySQL

Total de tuplas: 4

3.10 Tabla Planificación Laboral

Tabla que contiene datos referentes a la planificación laboral, posee una clave compuesta que identifica la planificación laboral del empleado.

Estructura física:

```
[nro_legajo] [int] NOT NULL,
[fecha] [date] NOT NULL,
[cliente] [int] NOT NULL,
[tipohora] [int] NOT NULL,
[hora_desde] [int] NOT NULL,
[hora_hasta] [int] NOT NULL,
PRIMARY KEY [nro_legajo] ,[fecha]
```

Tamaño sin índices: 2.346.836 MB MSSQL

1.900.000 MB MySQL

Total de tuplas: 46.000.000

Capítulo 4: Casos de estudio aplicados en los distintos SGDB

El primer requerimiento planteado, consiste en obtener todos los empleados (nro_legajo) que no hayan tenido planificación laboral. Para resolver dicho requerimiento existen diferentes alternativas, las que se definen como casos de estudio, y serán analizados a continuación. Es importante tener en cuenta que las tablas involucradas en la resolución de este requerimiento son las de mayor tamaño existente en la base de datos.

El segundo requerimiento se base en, obtener la cantidad distinta de empleados que trabajaron en una fecha dada una fecha. El objetivo principal de este requerimiento se centra en analizar la performance del producto cartesiano y la cláusula inner join. Se decidió mostrar la cantidad ya que el buffer de salida, es decir la proyección del Select, de la consulta se desbordaba cuando superaba la cantidad de 100 tuplas devueltas.

Para cada caso de estudio se realizó una muestra de 10 ejecuciones con la finalidad de analizar el tiempo de respuesta 10 veces. De esta forma se puede sacar una media de ejecución, que a fines estadísticos presenta una medición más cercana con una desviación estándar. Previo a cada ejecución se realizó una limpieza del cache, para que de esta forma el analizador de consultas resuelva la consulta como si fuera la primera vez que se ejecuta en el entorno y obtener los resultados lo más transparente posible.

La unidad de medida utilizada para cada ejecución fue el segundo con decimales, es decir centésimas, ya que este permite apreciar de una mejor manera las diferencias entre cada ejecución con respecto a otras unidades de medida como el milisegundo o microsegundo.

4.1 Caso de estudio uno aplicado sobre el DBMS Microsoft SQL Server

4.1.1 Cláusula IN

```
SELECT EM.NRO_LEGAJO
FROM EMPLEADOS EM
WHERE EM.NRO_LEGAJO NOT IN (SELECT  NRO_LEGAJO
                            FROM PLANIFICACIONLABORAL)
```

Q1

Plan de ejecución

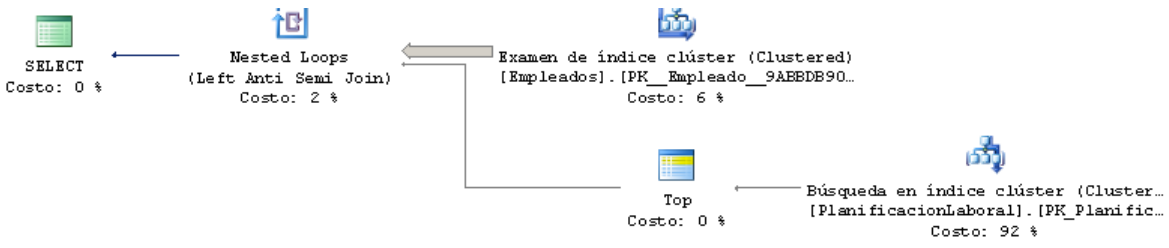


Figura 1

Sé observa que dentro del plan de ejecución el costo mayor para la resolución de la consulta `Q1` está definido sobre la tabla PlanificacionLaboral, el cual utiliza el operador *Clustered Index Seek* que utiliza la capacidad de búsqueda de los índices para recuperar filas de un índice cluster.

Resultados

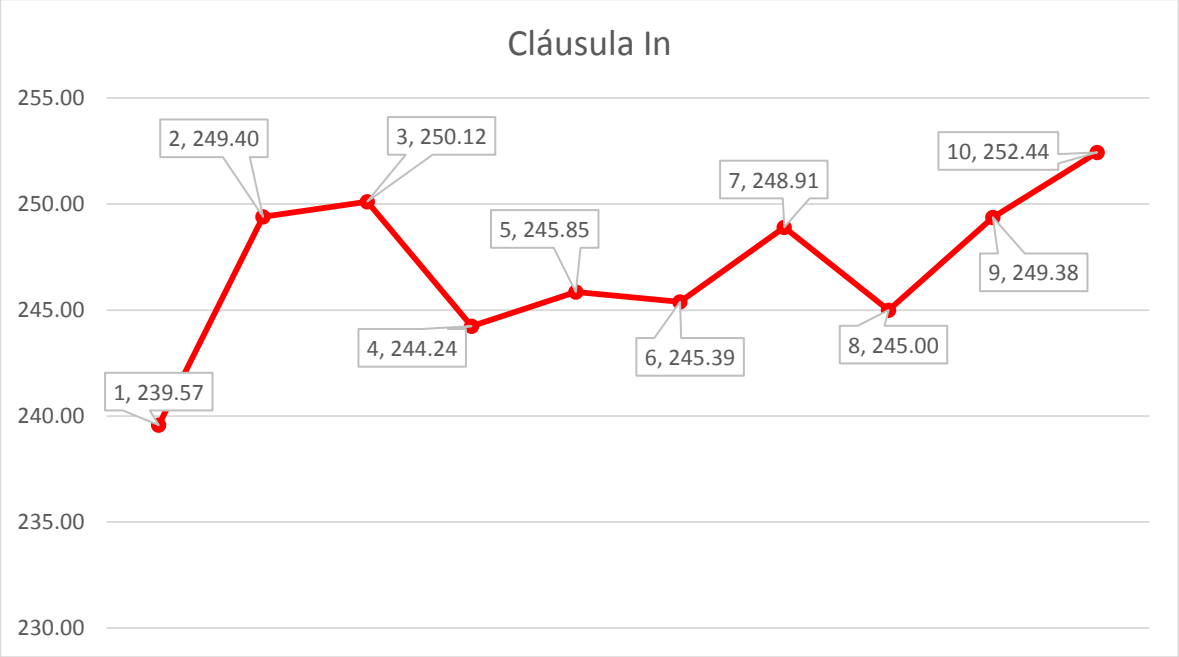


Gráfico 1

De los resultados se obtiene lo siguiente

Media aritmetica: 247.03 segundos

Desviacion estandar: 3.72

La media aritmética proviene del cálculo promedio de las 10 ejecuciones y será utilizado para decir que en promedio la consulta `Q1` tarda un total de 247.03 segundos en ejecutarse, mientras que la desviación estándar obtenida para un muestra de 10 ejecuciones, muestra cuanto pueden alejarse los valores respecto del promedio.

4.1.2 Cláusula IN con Distinct

Este ejemplo plantea una variante menor al anterior (Q1), la cláusula principal, IN esta presente para resolver el requerimiento,pero se contempla la consulta con la clausula Distinct al grupo devuelto por la tabla PlanificacionLaboral.

Dentro de la tabla PlanificacionLaboral,el dato del campo nro_legajo se puede repetir ya que cada empleadopudo trabajar mas de un dia a lo largo de toda su historia en la empresa. Por este motivo se decide agregar la Cláusula Distinct ,con el objetivo de evaluar si mejora o empeora el rendimiento de la consulta en gral, al quitar del resultado los elementos repetidos.

```
SELECT EM.NRO_LEGAJO
FROM EMPLEADOS EM
WHERE EM.NRO_LEGAJO NOT IN (SELECT DISTINCT NRO_LEGAJO
                            FROM PLANIFICACIONLABORAL)
```

Q2

Plan de ejecución

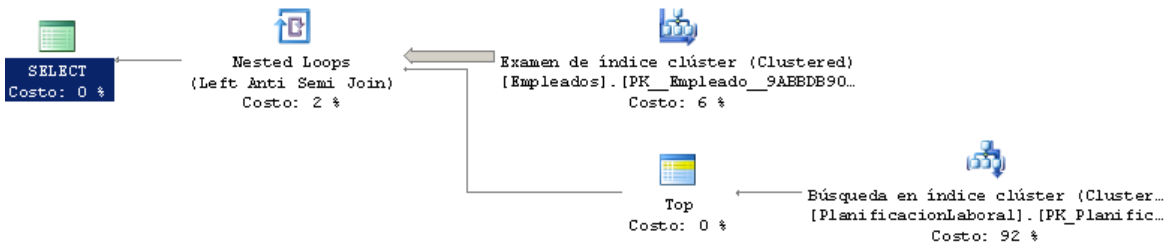


Figura 2

Comparando el plan de ejecución de la consulta Q1 contra la consulta Q2 no se ven diferencias, por lo que no se espera grandes cambios en el tiempo de ejecución de la misma.

Resultados

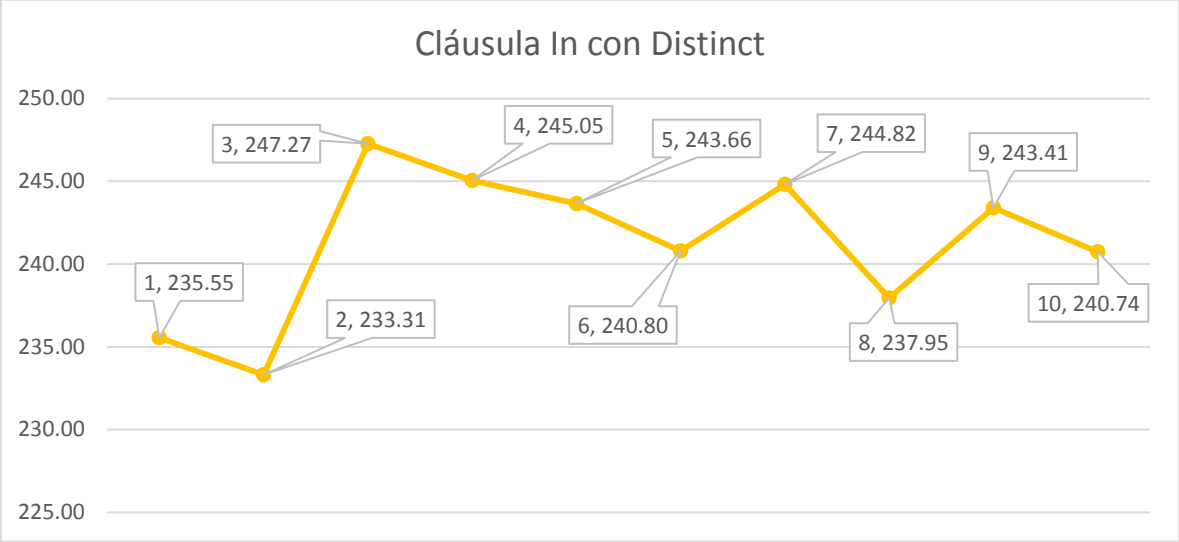


Gráfico 2

De los resultados se obtiene lo siguiente.

Media aritmetica: 241.26 segundos

Desviacion estandar: 4.48

A través de estos datos se puede deducir que, en este entorno, el Distinct no produce grandes cambios con respecto de no utilizarlo.

4.1.3 Cláusula Exists

En este caso se decide utilizar otra cláusula para resolver lo solicitado. Es el turno del exists.

```
SELECT NRO_LEGAJO
FROM EMPLEADOS EM
WHERE NOT EXISTS (SELECT PL.NRO_LEGAJO
                  FROM PLANIFICACIONLABORAL PL
                  WHERE EM.NRO_LEGAJO = PL.NRO_LEGAJO)
```

Q3

Plan de ejecución

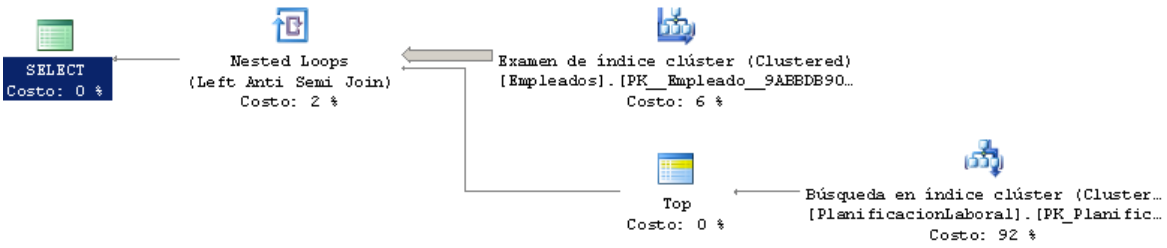


Figura 3

Observando el plan de ejecución se puede decir que el SGBD, en este entorno, opta por resolver de la misma forma la consulta, sin importar la cláusula que se utilice.

Resultados

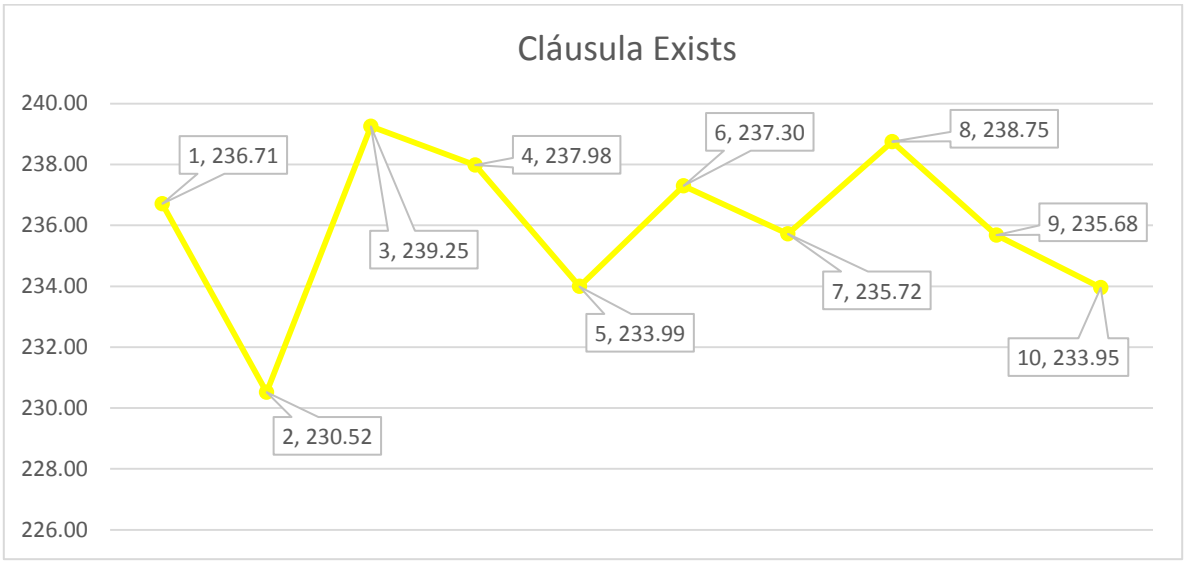


Gráfico 3

De los resultados se obtiene lo siguiente.

Media aritmetica: 235.99 segundos

Desviacion estandar: 2.63

4.1.4 Cláusula Except

La cláusula except es una opción más para resolver el requerimiento planteado, como se muestra a continuación

```
SELECT NRO_LEGAJO
FROM EMPLEADOS

EXCEPT

SELECT DISTINCT NRO_LEGAJO
FROM PLANIFICACIONLABORAL
```

Q4

Plan de ejecución

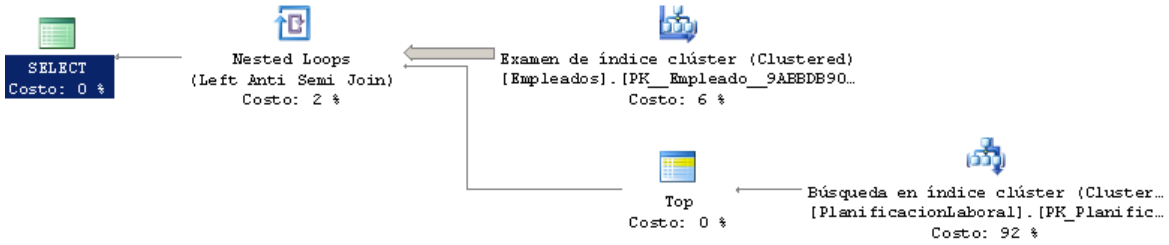


Figura 4

Al igual que los planes de ejecución de *Q1, Q2 y Q3* no se observa ninguna diferencia por lo que se puede esperar que no hayan grandes cambios en el tiempo de ejecución.

Resultados

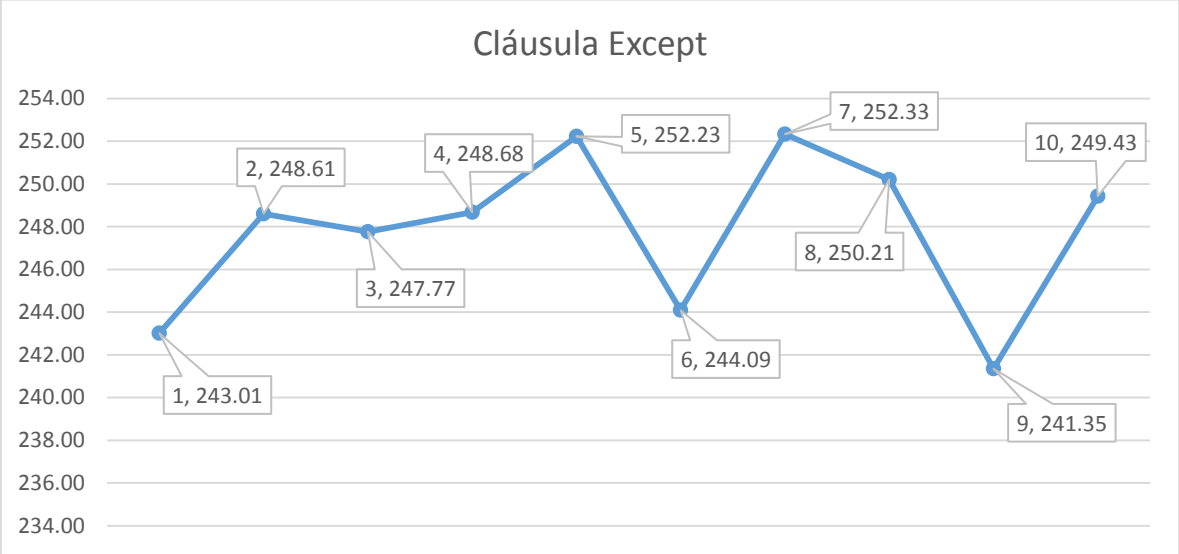


Gráfico 4

De los resultados se obtiene lo siguiente

Media aritmetica: 235.99 segundos

Desviacion estandar：2.63

4.1.5 Cláusula Left Join

```
SELECT EM.NRO_LEGAJO
FROM EMPLEADOS EM LEFT JOIN PLANIFICACIONLABORAL PL
ON (EM.NRO_LEGAJO = PL.NRO_LEGAJO)
WHERE PL.NRO_LEGAJO IS NULL
```

Q5

Plan de ejecución

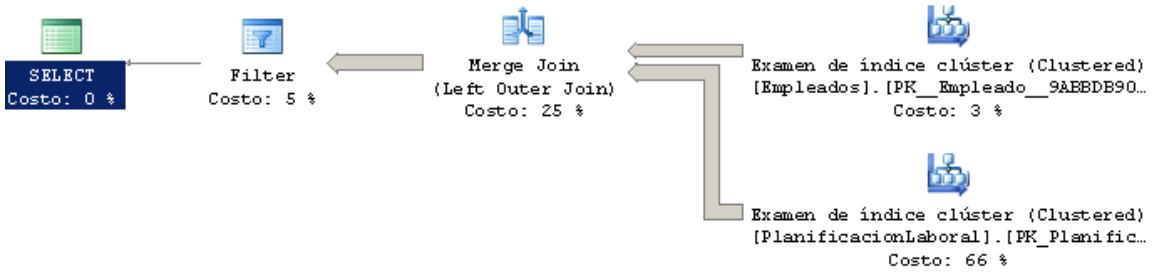


Figura 5

En este caso se observa un cambio completo en el plan de ejecución con respecto a las consultas `Q1, Q2, Q3 y Q4.`

Vemos que el costo de las búsqueda de datos en tabla PlanificacionLaboral bajo considerablemente.

Resultados

Gráfico 5

De los resultados se obtiene lo siguiente

Media aritmetica: 75.20 segundos

Desviacion estandar：1.49

Comparación

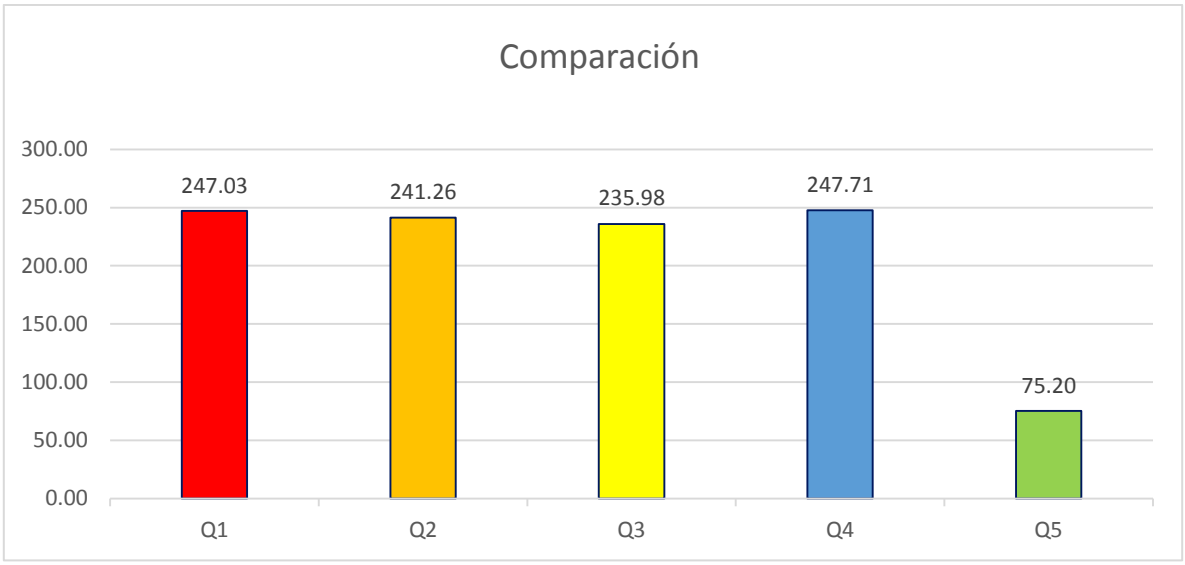


Gráfico 6

Como se observa en el gráfico 6 el tiempo de ejecución de las clausulas Not In, Not Exists y Except no presenten una diferencia importante en cuanto a sus tiempos de ejecución, esto se debe a que para cada cláusula el plan de ejecución es idéntico. Pero en el caso de la cláusula Left Join el plan de ejecución es totalmente diferente, dando la pauta de que los tiempos al ejecutar dicha consulta sean distintos al de las clausulas anteriores.

Al observar los resultados, el tiempo de ejecución de la cláusula Left Join es más eficiente con respecto a las otras, ya que el tiempo promedio de ejecución es 3 veces menor que las otras cláusulas anteriormente mencionadas.

Esta reducción del tiempo de ejecución está dado por la resolución en paralelo de los operadores *examen de índice cluster* para la tabla empleados y planificación laboral.

4.2 Caso de estudio dos aplicado sobre el DBMS Microsoft SQL Server

4.2.1 Producto Cartesiano

```
SELECT COUNT(DISTINCT EM.nro_legajo)
FROM Empleados EM , PlanificacionLaboral PL
WHERE EM.nro_legajo = PL.nro_legajo
and PL.fecha BETWEEN '20160101' AND '20160130'
```

Q6

Plan de ejecución

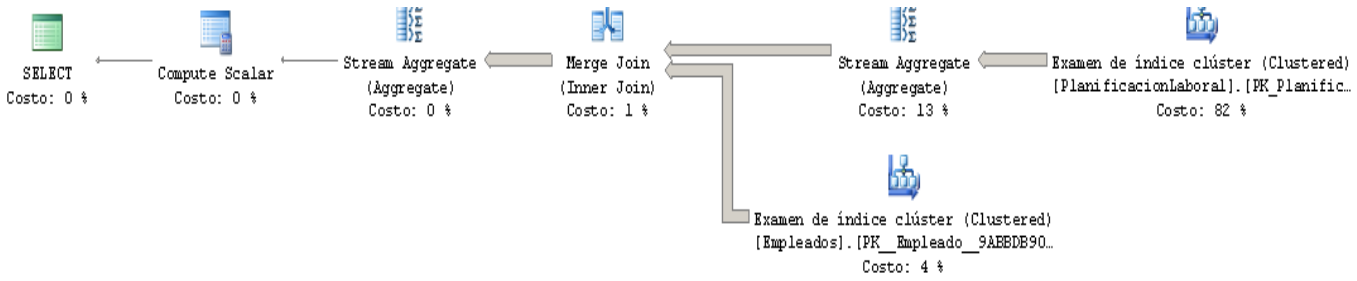


Figura 6

Como se observa en el plan de ejecución, el costo mayor de la consulta recae sobre el análisis del índice cluster de la tabla PlanificacionLaboral.

Resultados

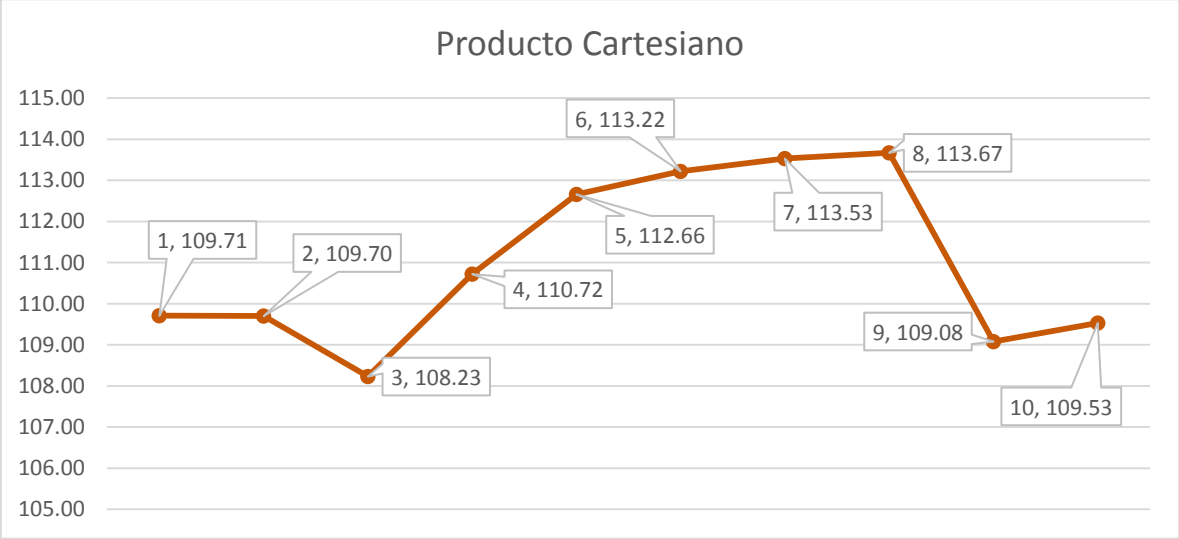


Gráfico 7

De los resultados se obtiene lo siguiente

Media aritmetica: 111 segundos

Desviacion estandar：2.05

4.2.2 Cláusula Inner Join

```
SELECT COUNT(DISTINCT EM.nro_legajo)
FROM Empleados EM inner join PlanificacionLaboral PL
on EM.nro_legajo = PL.nro_legajo
WHERe PL.fecha BETWEEN '20160101' AND '20160131'
```

Q7

Plan de ejecución

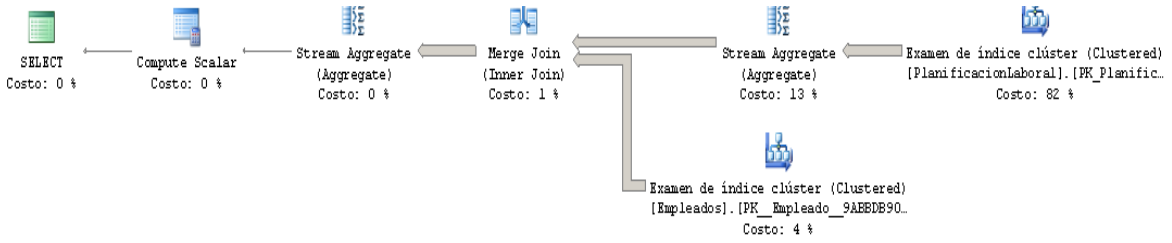


Figura 7

Como se observa en la figura 7 el plan de ejecución de la consulta Q7 es idéntico al plan de ejecución de la consulta Q6. Esta da la pauta de que los tiempos de ejecución de ambas consultas serán similares.

Resultados

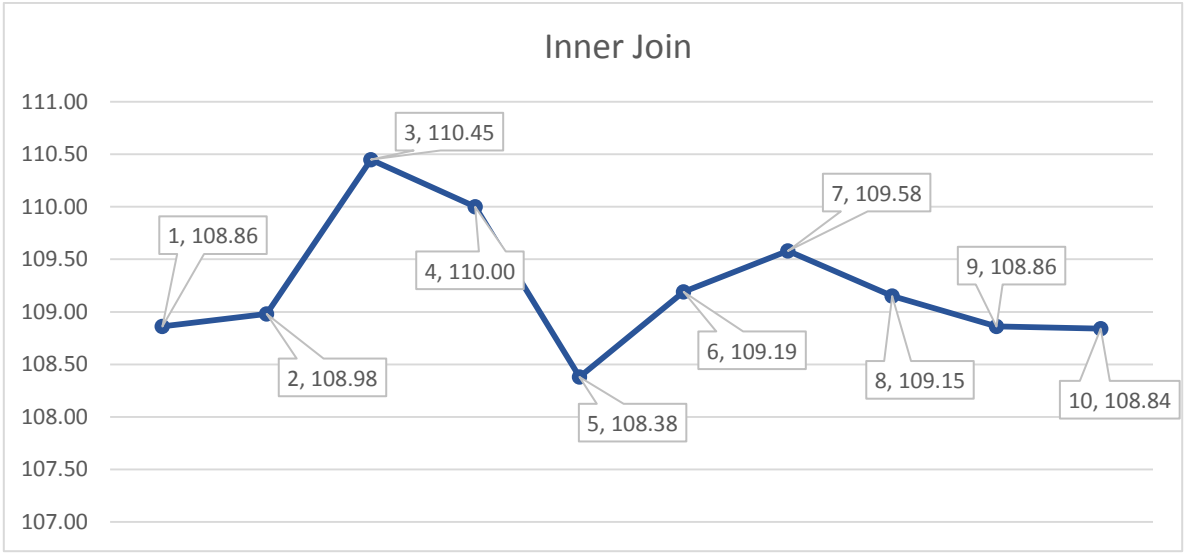


Gráfico 8

De los resultados se obtiene lo siguiente

Media aritmetica: 109.23 segundos

Desviacion estandar：0.61

Comparación

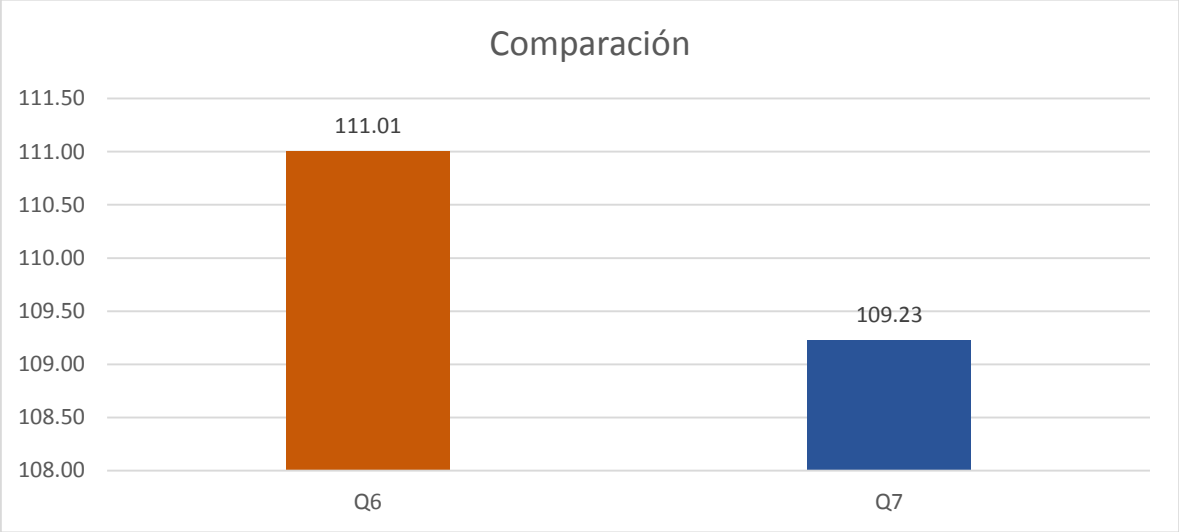


Gráfico 8

Conclusión: Al observar el gráfico de comparación de ambas Cláusulas, la diferencia en promedio entre las consultas Q6 y Q7 es de apenas 2 segundos, a partir de estos resultados se puede afirmar, en este ambiente dedicado, que se está frente a un caso donde los aspectos teóricos difieren de la práctica, ya que en teoría la cláusula inner join es más eficiente que el producto cartesiano.

4.3 Caso de estudio uno aplicado sobre el DBMS Microsoft SQL Server utilizando índices

Luego de todas las pruebas realizadas, se evalúo como afecta al rendimiento de las consultas al incorporar un índice a la tabla PlanificaciónLaboral.

Se incorporó un índice NonClustered a la tabla *PlanificacionLaboral ,* el cual se lo llamo *Tesina001,* dicho índice está compuesto por el campo *nro_legajo*. Se utilizó este campo ya que las consultas o subconsultas realizadas a la tabla *PlanificacionLaboral* incluyen solamente el campo *nro_legajo*

A continuación se detalla la sintaxis de creación del índice *Tesina001* sobre la tabla *PlanificacionLaboral.*

```
CREATE NONCLUSTERED INDEX [Tesina001] ON
[dbo].[PlanificacionLaboral]
([nro_legajo] ASC)
ON [PRIMARY]
```

Tamaño el índice Tesina001: 455,80 Mb

4.3.1 Cláusula IN

La sintaxis de la consulta es idéntica a la consulta Q1, ya que la incorporación de un índice a la tabla no afecta a la sintaxis de la consulta como se muestra a continuación.

```
SELECT EM.NRO_LEGAJO
FROM EMPLEADOS EM
WHERE EM.NRO_LEGAJO NOT IN (SELECT  NRO_LEGAJO
                            FROM PLANIFICACIONLABORAL)
```

QI1

Plan de ejecución

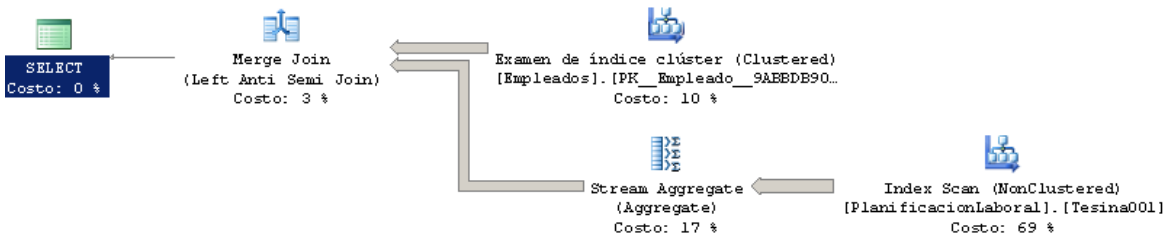


Figura 8

Como se observa en la Figura 8 el plan de ejecución de la consulta cambia con respecto a la consulta `Q1`. La incorporación del índice produce un decremento en el costo de la búsqueda de los registros de la tabla *PlanificacionLaboral.*

Resultados

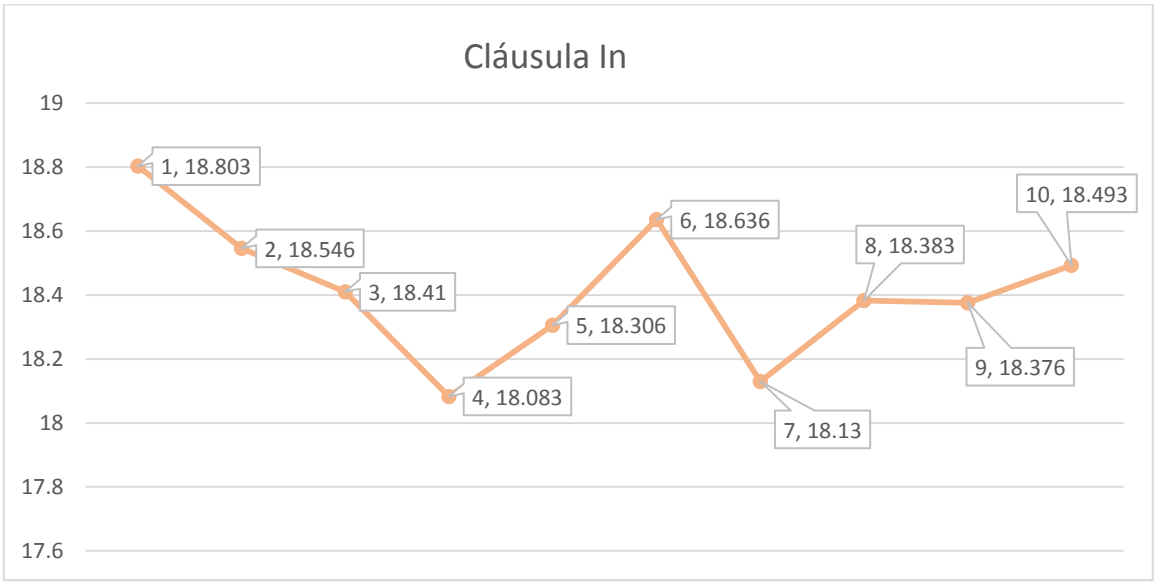


Gráfico 9

De los resultados se obtiene lo siguiente

Media aritmetica: 18.41 segundos

Desviacion estandar: 0.21

Comparación `Q1` & `QI1`

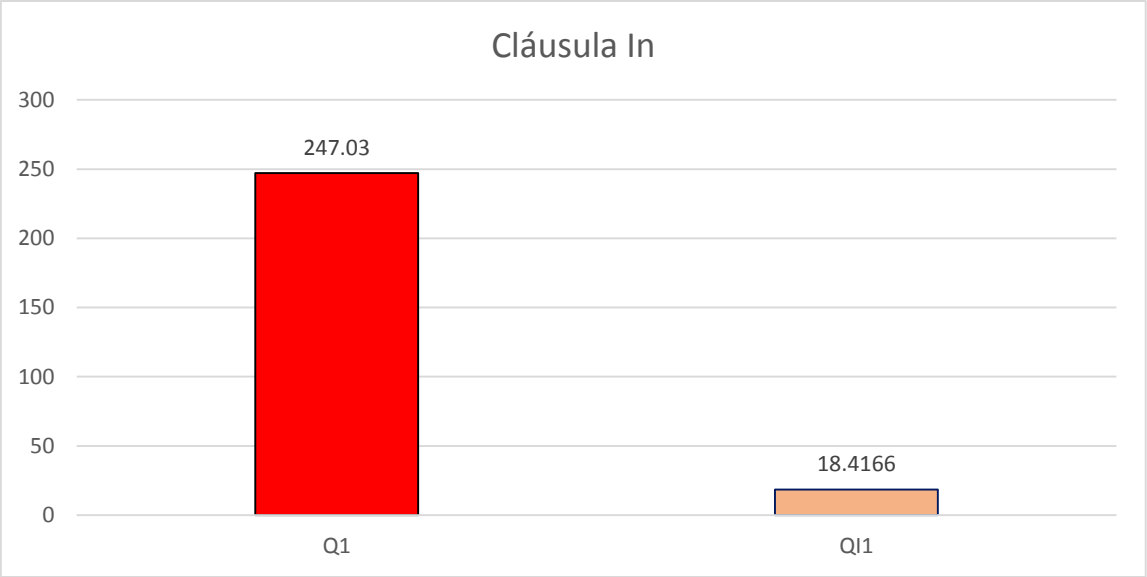


Gráfico 10

El Gráfico 9 revela que la incorporación del índice produjo los resultados esperados, el tiempo de ejecución promedio con índices es 13 veces menor al tiempo promedio obtenido sin índice.

4.3.2 Cláusula IN con Distinct

```
SELECT EM.NRO_LEGAJO
 FROM EMPLEADOS EM
 WHERE EM.NRO_LEGAJO NOT IN (SELECT DISTINCT  NRO_LEGAJO
                                 FROM PLANIFICACIONLABORAL)
```

QI2

Plan de ejecución

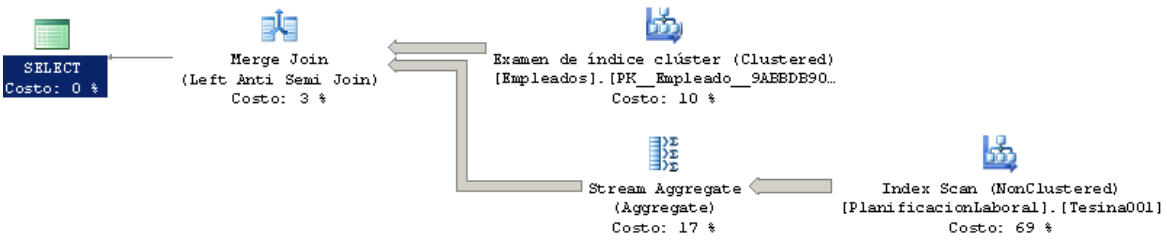


Figura 10

Como se observa en la Figura 10 los planes de la consulta `QI1` y `QI2` son idénticamente igual, con lo que se puede afirmar que el analizador de consultas desestima la utilización de la cláusula distinct dentro de la subconsulta.

Resultados

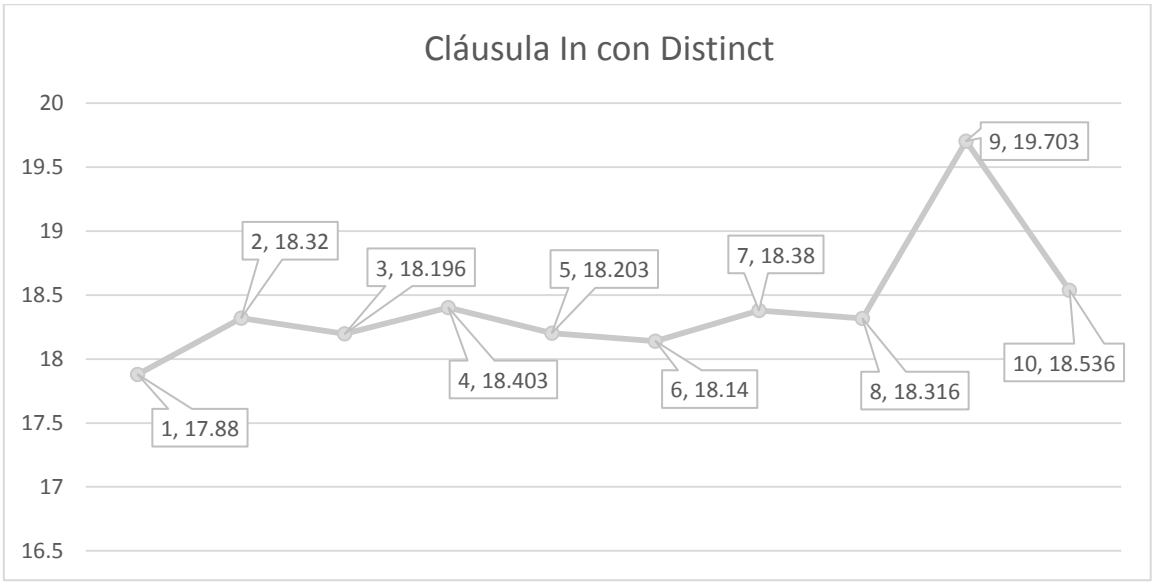


Gráfico 11

De los resultados se obtiene lo siguiente

Media aritmetica: 18.40 segundos

Desviacion estandar：0.48

Al igual que en las pruebas anterior, es notable que el tiempo de ejecución de cada prueba realizada sobre la consulta `QI2` es menor que los tiempos obtenidos en las pruebas de `Q2`.El Gráfico que se muestra a continuación demuestra la diferencia.

Comparación `Q2` & `QI2`

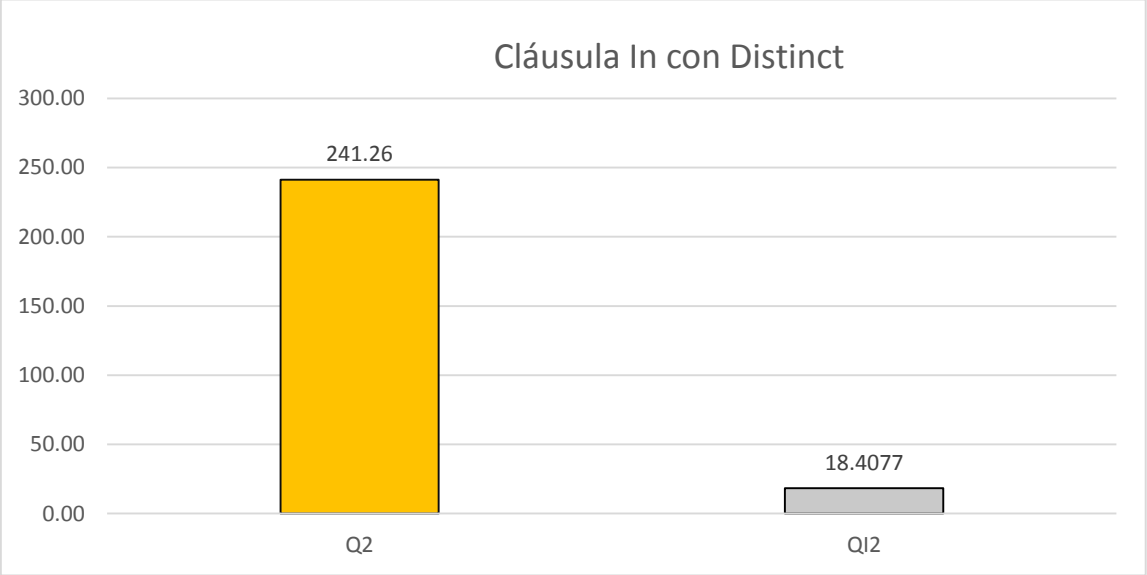


Gráfico 12

4.3.3 Cláusula Exists

```
SELECT NRO_LEGAJO
FROM EMPLEADOS EM
WHERE NOT EXISTS (SELECT PL.NRO_LEGAJO
                  FROM PLANIFICACIONLABORAL PL
```

QI3

Plan de ejecución

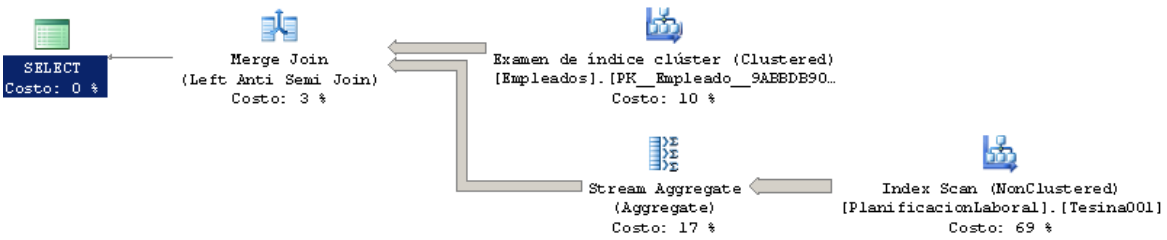


Figura 11

El plan de ejecución de la consulta `QI3` no muestra diferencia con respecto a los planes de ejecución de `QI1 Y QI2`.Por lo tanto se espera que los tiempos de ejecución de la consulta `QI3` no sean muy diferente a los tiempos de las consultas `QI1 Y QI2`.

Resultados

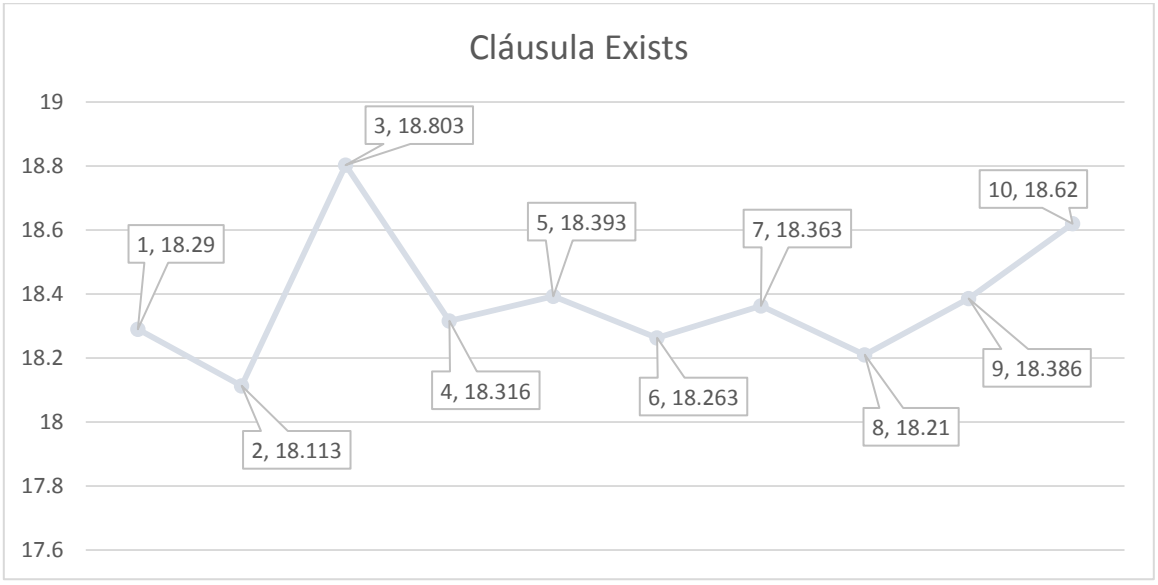


Gráfico 13

De los resultados se obtiene lo siguiente

Media aritmetica: 18.37 segundos

Desviacion estandar：0.20

Comparación Q3 & QI3

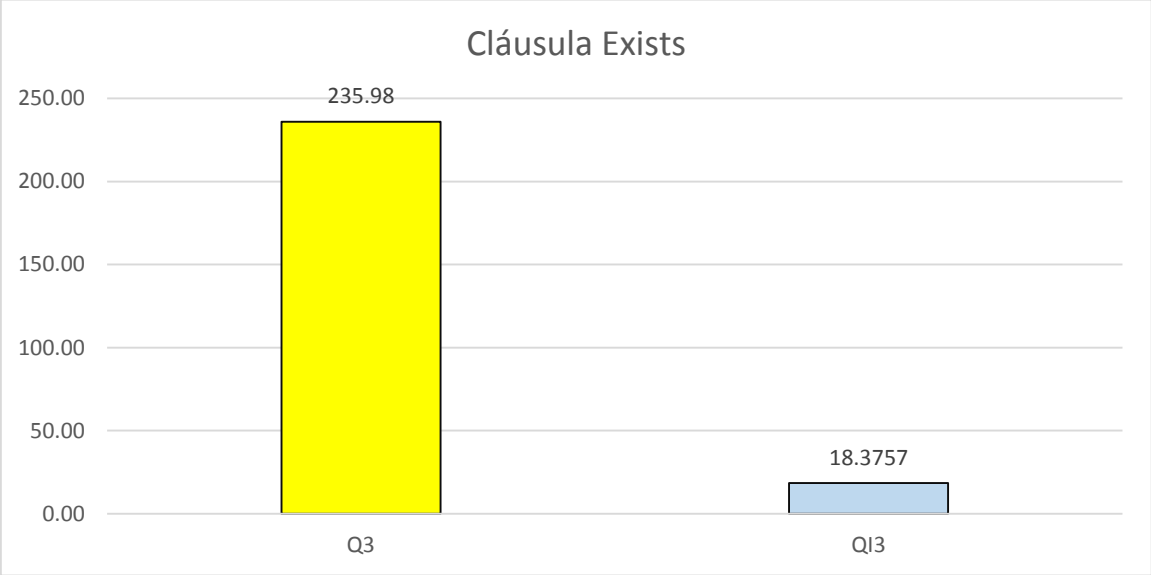


Gráfico 14

De los resultados anteriores demuestran que la incorporación del índice Tesina001 mejoró considerablemente el tiempo de ejecución de la cláusula Exists.

4.3.4 Cláusula Except

```
SELECT NRO_LEGAJO
FROM EMPLEADOS

EXCEPT

SELECT DISTINCT NRO_LEGAJO
FROM PLANIFICACIONLABORAL
```

QI4

Plan de ejecución

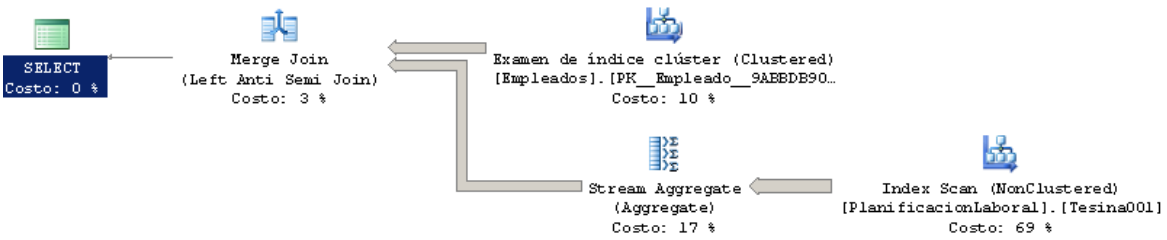


Figura 12

Al igual que los casos anteriores el SGBD, en este entorno dedicado, opta por resolver de la misma forma la consulta, sin importar la cláusula.

Resultados

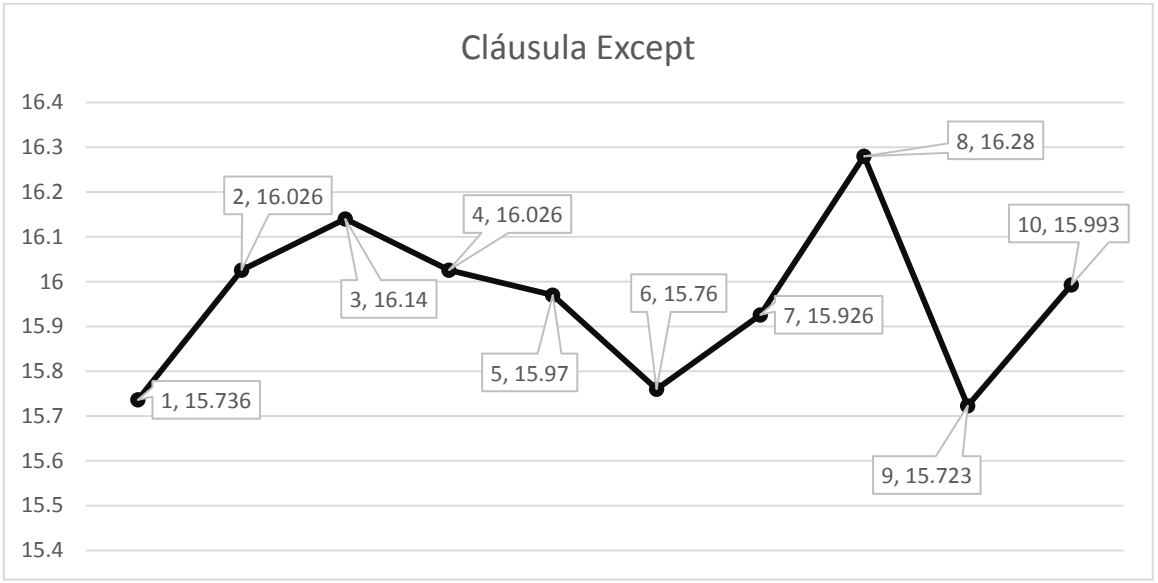


Gráfico 15

De los resultados se obtiene lo siguiente

Media aritmetica: 15.95 segundos

Desviacion estandar: 0.18

Comparación Q4 & QI4

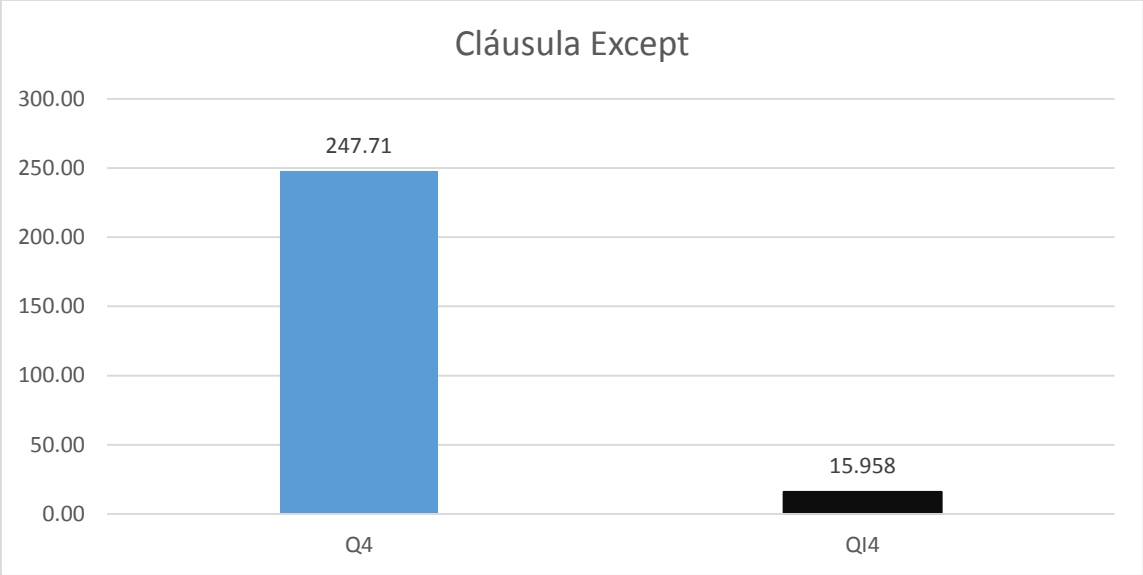


Gráfico 16

La incorporación del índice Tesina001, al igual que en los casos anteriores redujo, también en la cláusula Except, el tiempo promedio de ejecución.

4.3.5 Cláusula Left Join

```
SELECT EM.NRO_LEGAJO
FROM EMPLEADOS EM LEFT JOIN PLANIFICACIONLABORAL PL
ON (EM.NRO_LEGAJO = PL.NRO_LEGAJO)
WHERE PL.NRO_LEGAJO IS  NULL
```

QI5

Plan de ejecución

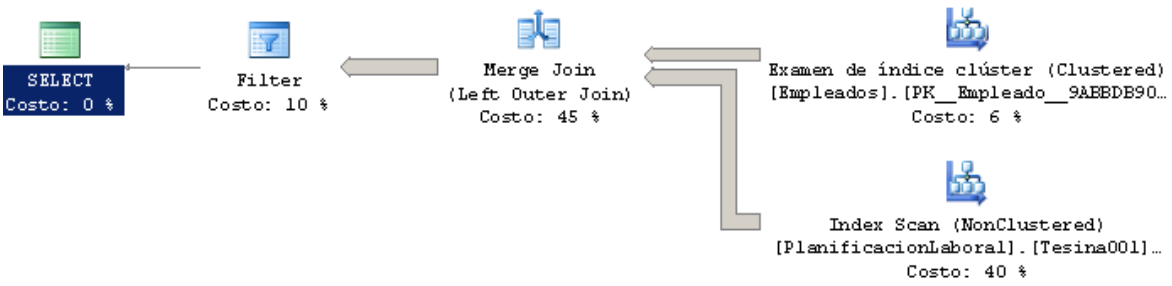


Figura 13

Al igual que el caso Q5, el plan de ejecución de la consulta QI5, es diferente con respecto al de las otras consultas.

Resultados

Gráfico 17

De los resultados se obtiene lo siguiente

Media aritmetica: 16.5 segundos

Desviacion estandar: 0.30

Comparación Q5 & QI5

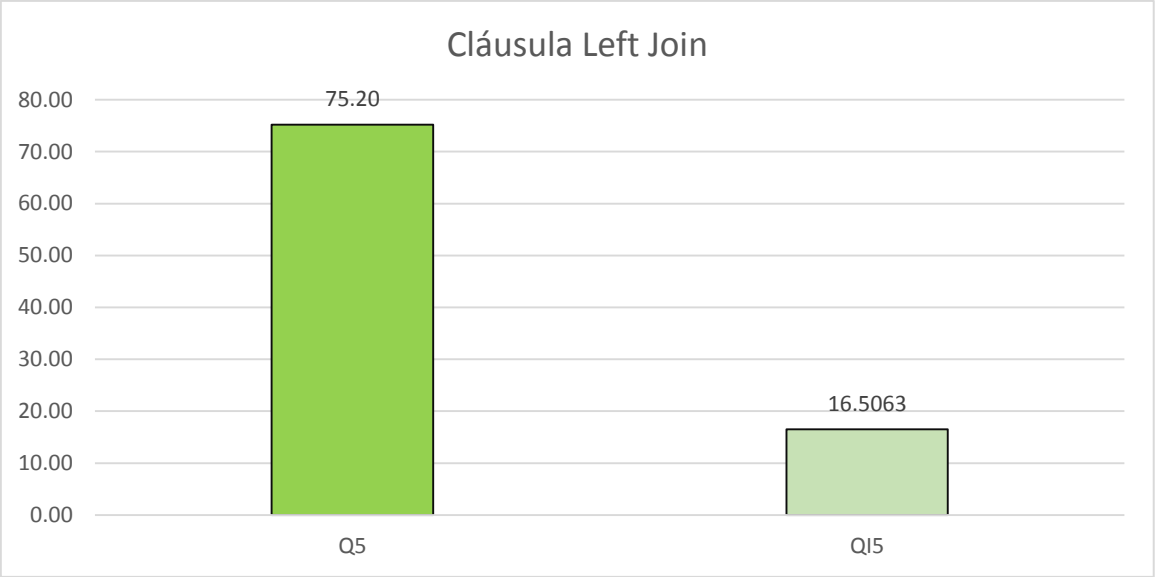


Gráfico 17

Al comparar Q5 y QI5 se observa exactamente lo mismo que en los otros casos,la incoportacion del indice Tesina001 produce una reduccion en el tiempo de ejecución.Esto no quiere decir que se la Cláusula mas optima para resolver el requerimiento ya que todos los casos de estudios probados con el indice Tesina001 mostraron valores muy similares en el tiempo de ejecucion.

Comparación

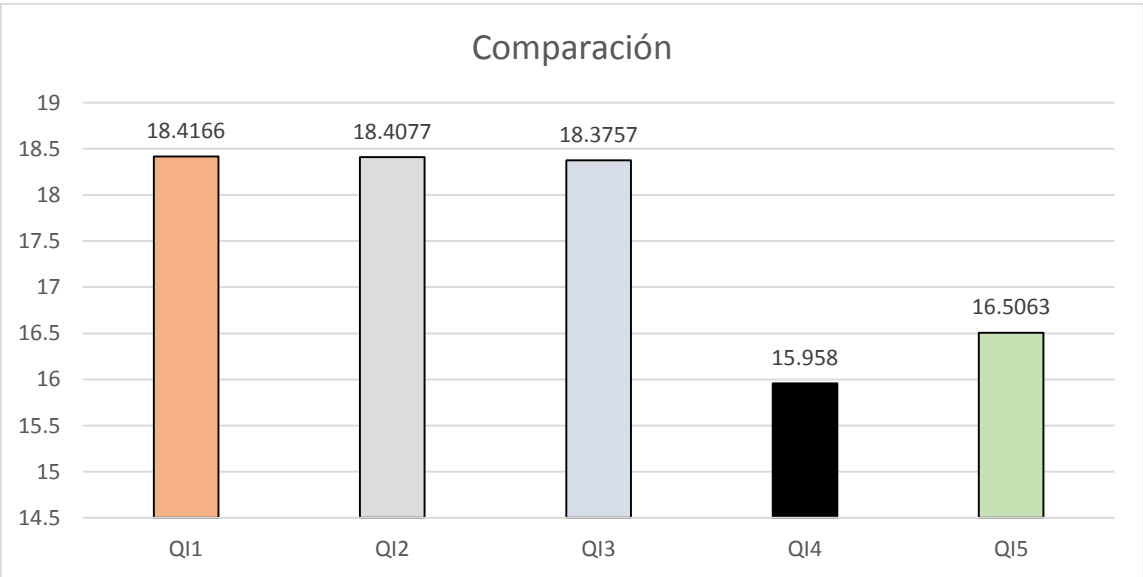


Gráfico 18

De los resultados obtenidos se puede decir lo siguiente, luego de incorporar el índice Tesina001 los tiempos de ejecución de cada consulta se vio reducido en promedio 16 veces. Pero a diferencia de los casos sin índices estas pruebas no nos permiten decidir que Cláusula es más eficiente o mejor qué otra ya que los tiempos de ejecución obtenidos en cada prueba resultaron ser muy similares.

4.4 Caso de estudio dos aplicado sobre el DBMS Microsoft SQL Server utilizando índices

4.4.1 Producto Cartesiano

```
SELECT COUNT(DISTINCT EM.nro_legajo)
FROM Empleados EM , PlanificacionLaboral PL
WHERE EM.nro_legajo = PL.nro_legajo
and PL.fecha BETWEEN '20160101' AND '20160130'
```

Q6

Plan de ejecución

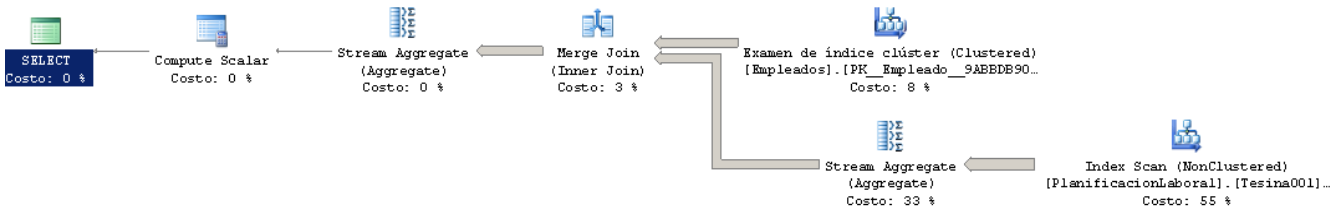


Figura 14

Tras haber aplicado el índice se observó el plan de ejecución del producto cartesiano, como se puede apreciar el SGBD comienza a hacer uso del mismo, lo cual indica una alta posibilidad de que el tiempo de ejecución de la consulta QI6 se menor al de la consulta Q6.

Resultados

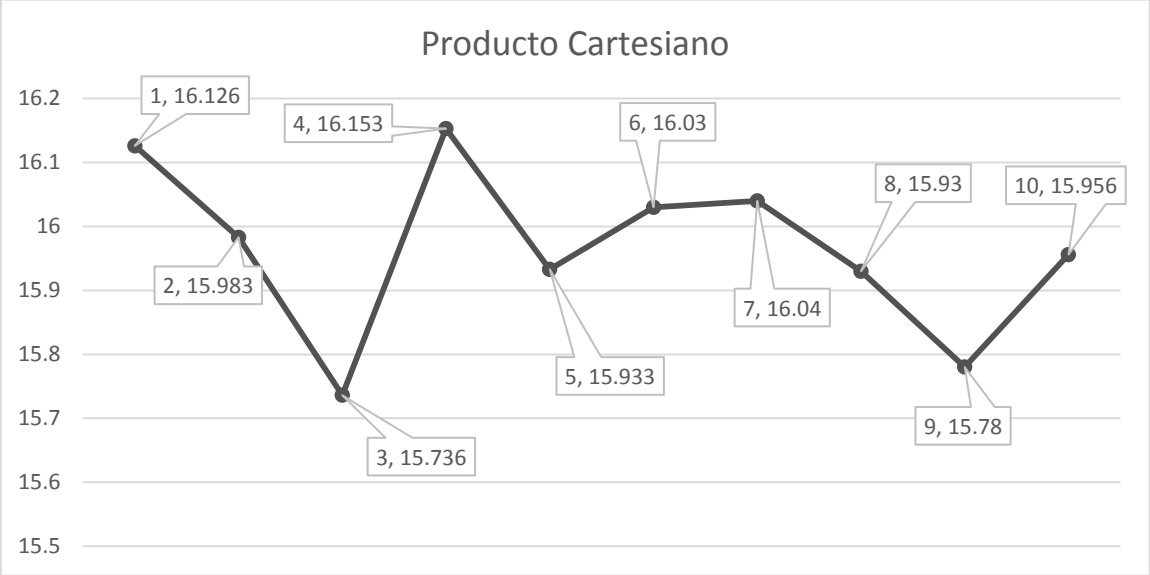


Gráfico 19

De los resultados se obtiene lo siguiente

Media aritmetica: 15.96 segundos

Desviacion estandar: 0.13

Comparación Q6 & QI6

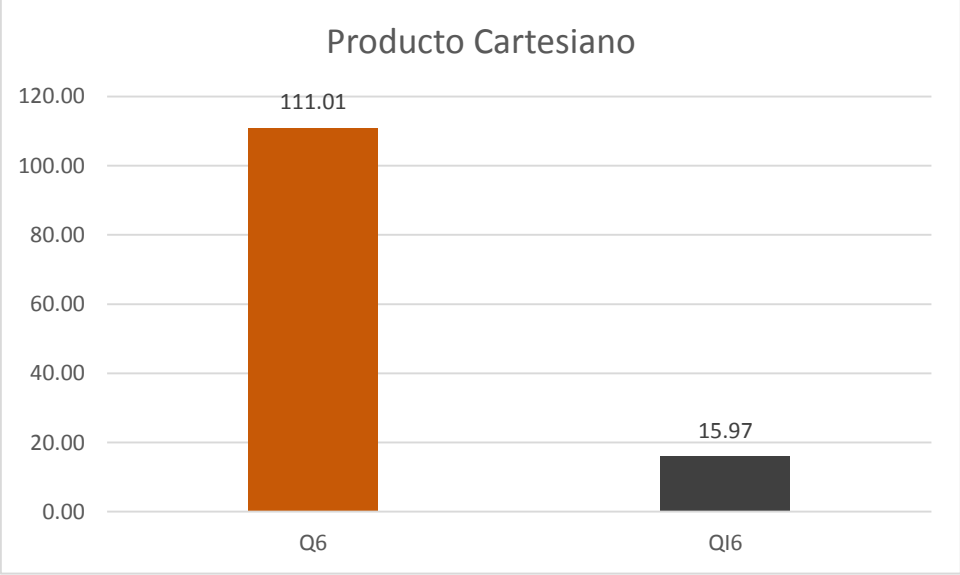


Gráfico 19

Los resultados obtenidos demuestran lo que se pensaba al observar el plan de ejecución, el tiempo ejecución promedio de producto cartesiano es 7 veces menor con respecto al tiempo obtenido sin índice.

4.5 Cláusula Inner Join

```
SELECT COUNT(DISTINCT EM.nro_legajo)
FROM Empleados EM inner join PlanificacionLaboral PL
on EM.nro_legajo = PL.nro_legajo
WHERe PL.fecha BETWEEN '20160101' AND '20160131'
```

QI7

Plan de ejecución

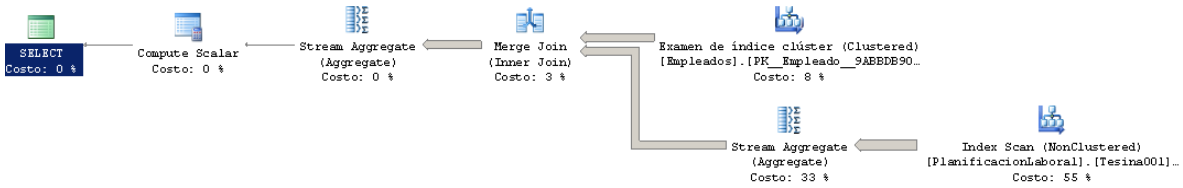


Figura 14

El plan de ejecución de la cláusula inner join es idéntico al del producto cartesiano, por lo que se esperó que los tiempo de ejecución sean similares en ambos casos.

Resultados

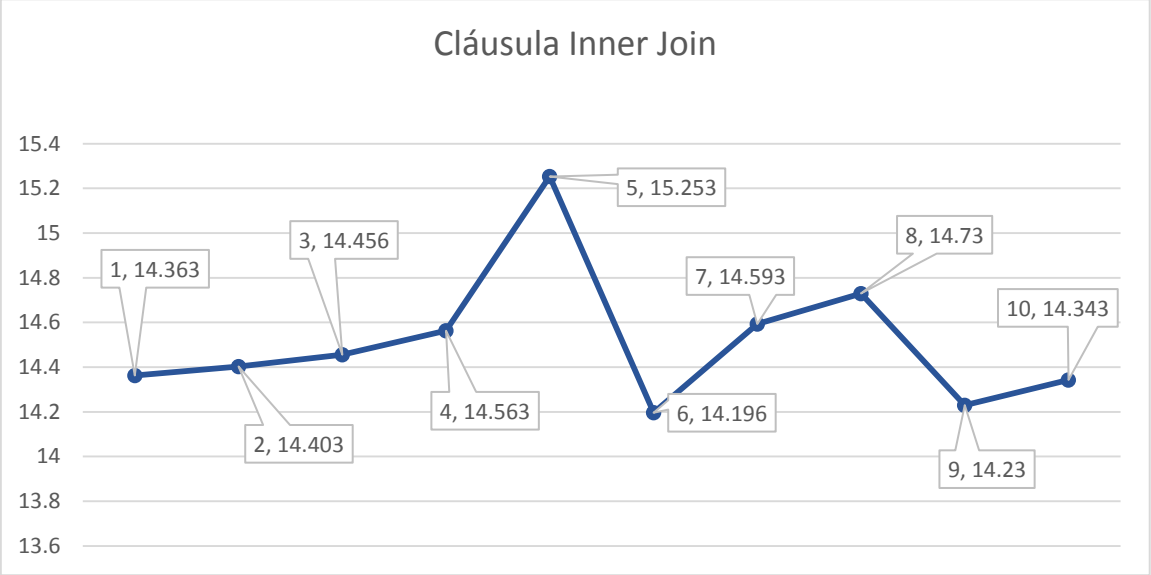


Gráfico 20

Comparación Q7 & QI7

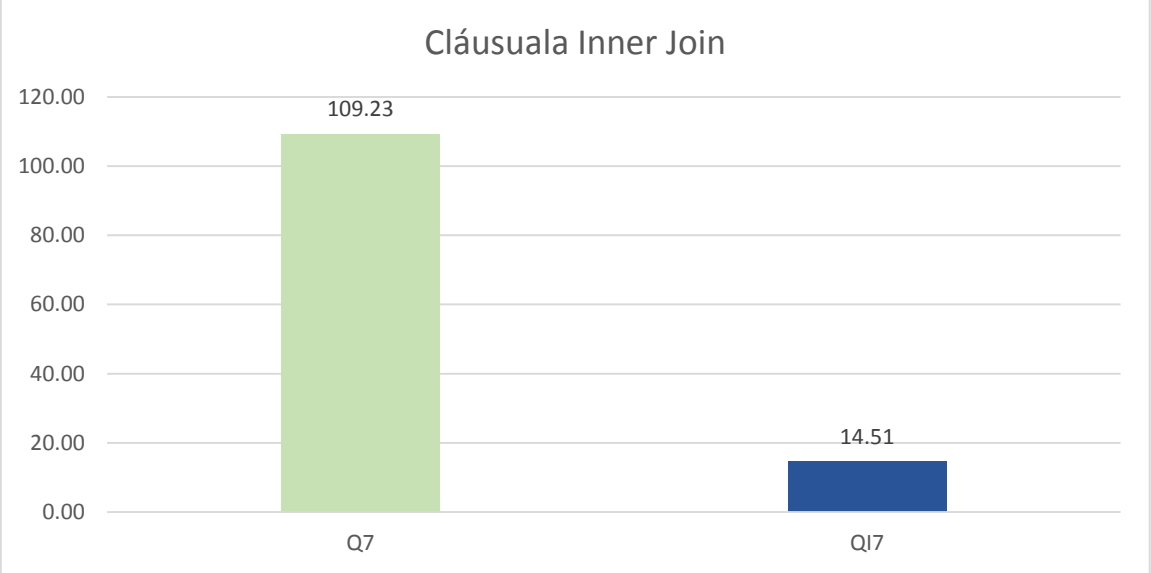


Gráfico 21

Los resultados demuestran que el tiempo de ejecucion en promedio de la Cláusula inner al utilizar indice es 7 veces menos que sin indice.

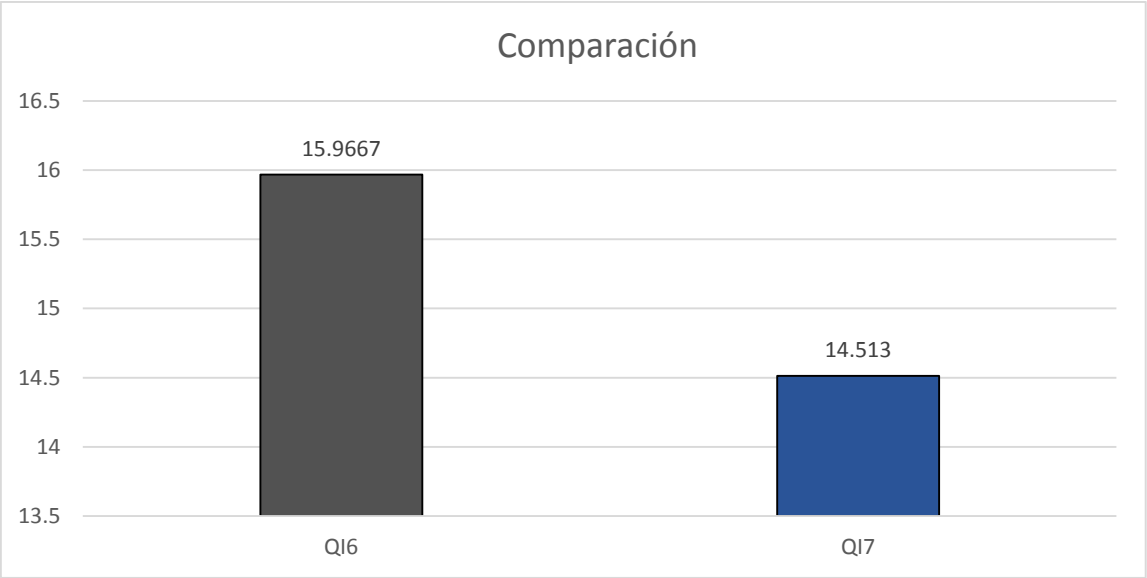


Gráfico 22

Los resultados demuestran que hay una mínima diferencia entre usar un producto cartesiano y la cláusula inner join.Esto se debe a que el optimizador de consulta siempre opta por resolver de la manera más eficiente cada consulta ,en este caso omitiendo el producto cartesiano.

4.5 Caso de estudio uno aplicado sobre el DBMS MySQL

Luego de analizar todos los casos de estudios en el SGBD de Microsoft se procedió a realizar las mismas pruebas con los mismos casos de estudios en el SGBD MySQL.

De la misma forma que en el DMBS de Microsoft, se realizaron 10 pruebas con el objetivo de mostrar que para cada prueba el tiempo es variable y de esta forma poder obtener un promedio del tiempo de ejecución de cada caso de estudio, previo a la ejecución de cada prueba se realizó una limpieza del cache simulando de esta forma que la consulta se ejecuta por primera vez en cada prueba.

La unidad de medida utilizada para cada prueba, también es el segundo permitiendo de esta manera comparar los resultado obtenidos por ambos SGBD y a partir de ellos obtener las conclusiones pertinentes.

4.5.1 Cláusula IN

```
SELECT EM.NRO_LEGAJO
FROM EMPLEADOS EM
WHERE EM.NRO_LEGAJO NOT IN (SELECT  NRO_LEGAJO
                                FROM PLANIFICACIONLABORAL)
```

C1

Plan de ejecución

id	select_type	table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	EM	index	NULL	FK_Empleados_Municipios	603	NULL	999682	100.00	Using where; Using index
2	DEPENDENT SUBQUERY	PLANIFICACIONLABORAL	index_subquery	PRIMARY	PRIMARY	4	func	22	100.00	Using index

Figura 15

Como se observa en la Figura 15, a diferencia del plan de ejecución del DMBS de Microsoft, este plan de ejecución es mucho más limitado en cuanto representación Gráfica y, además , es mas más complejo a la hora de interpretarlo.

Principalmente, esta interpretación, se centra en la cantidad de filas filtradas al momento de resolver la consulta `C1`. En este caso se observa que el SGBD realiza un filtrado del 100% de las filas, columna filtered. Esto se debe a que no existe índice alguno que pueda ayudar a resolver la consulta. A continuación se presentan los resultados para cada una de las 10 pruebas realizadas.

Resultados

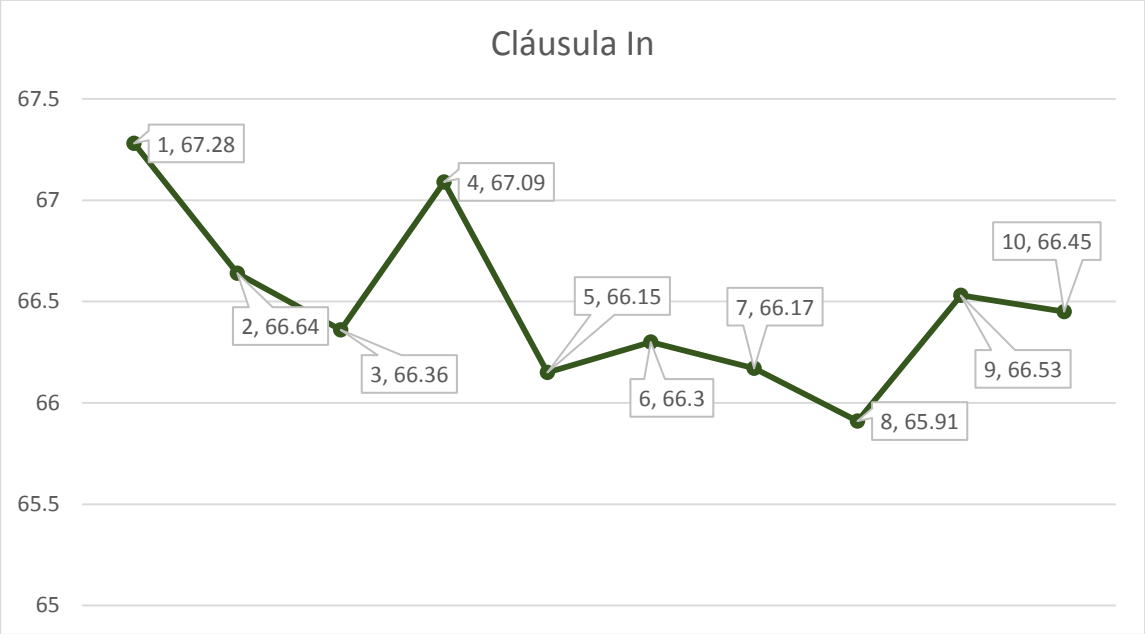


Gráfico 23

De los resultados se obtiene lo siguiente

Media aritmetica: 66.48 segundos

Desviacion estandar：0.42

Al igual que con las pruebas realizadas en el SGBD de Microsoft la media aritmética proviene del cálculo promedio de las 10 ejecuciones y será utilizado para decir que en promedio la consulta `C1` tarda un total de 66.48 segundos en ejecutarse, mientras que la desviación estándar obtenida para un muestra de 10 ejecuciones, enseña cuanto pueden alejarse los valores respecto del promedio.

4.5.2 Cláusula IN con Distinct

```
SELECT EM.NRO_LEGAJO
FROM EMPLEADOS EM
WHERE EM.NRO_LEGAJO NOT IN (SELECT DISTINCT  NRO_LEGAJO
                                FROM PLANIFICACIONLABORAL)
```

C2

Plan de ejecución

id	select_type	table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	EM	index	NULL	FK_Empleados_Municipios	603	NULL	999682	100.00	Using where; Using index
2	DEPENDENT SUBQUERY	PLANIFICACIONLABORAL	index_subquery	PRIMARY	PRIMARY	4	func	22	100.00	Using index

Figura 16

Como se observa en la Figura 16 los planes de la consulta `C1` y `C2` son idénticamente igual, con lo que se puede afirmar que el analizador de consultas desestima la utilización de la cláusula distinct dentro de la sub consulta. Ídem a los que sucede en el DMBS de Microsoft.

Resultados

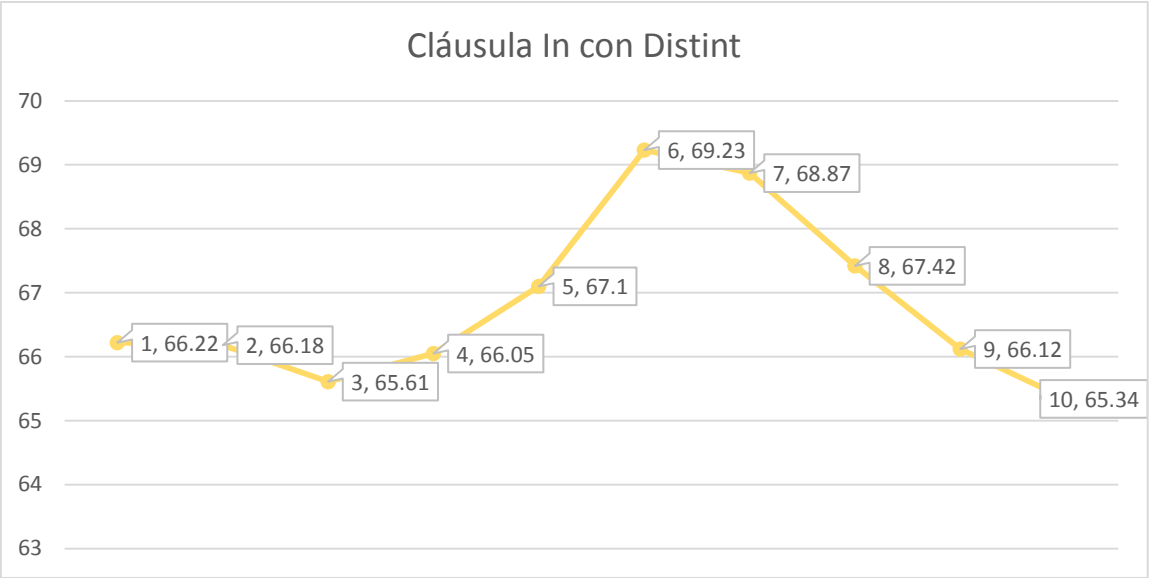


Gráfico 24

De los resultados se obtiene lo siguiente

Media aritmetica: 66.81 segundos

Desviacion estandar: 1.33

Como se supuso en el plan de ejecución de la consulta `C2` los tiempo de ejecución de no variaron significativamente con respecto de usar la cláusula distinct o no en la sub consulta.

4.5.3 Cláusula Exists

```
SELECT NRO_LEGAJO
FROM EMPLEADOS EM
WHERE NOT EXISTS (SELECT PL.NRO_LEGAJO
                  FROM PLANIFICACIONLABORAL PL
                  WHERE EM.NRO_LEGAJO = PL.NRO_LEGAJO)
```

Q3

Plan de ejecución

id	select_type	table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	EM	index	NULL	FK_Empleados_Municipios	603	NULL	999682	100.00	Using where; Using index
2	DEPENDENT SUBQUERY	PLANIFICACIONLABORAL	index_subquery	PRIMARY	PRIMARY	4	func	22	100.00	Using index

Figura 17

Como se observa en la Figura 17 el cambio de Cláusula Exists por IN no produce cambios en cuanto al plan de ejecución, por lo que se es de esperar que los resultados obtenidos en cada prueba sean similares al de la Cláusula IN

Resultados

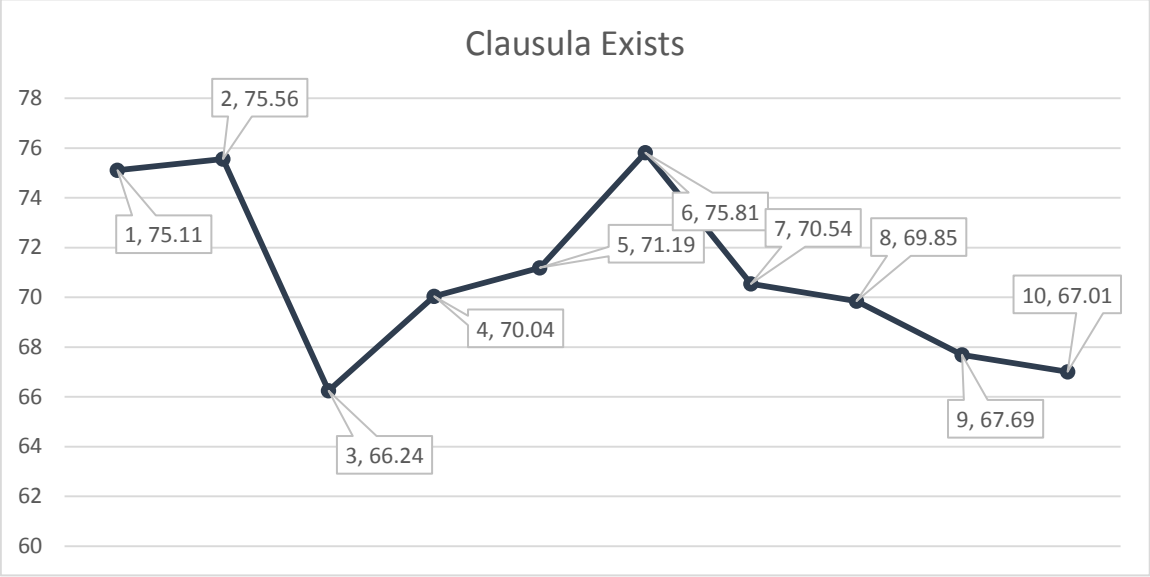


Gráfico 25

De los resultados se obtiene lo siguiente

Media aritmetica: 70.90 segundos

Desviacion estandar: 3.53

El DBMS MySQL, no implementa la cláusula Except , es por este motivo que no se pudieron realizar las pruebas con dicha cláusula.

4.5.4 Cláusula Left Join

```
SELECT EM.NRO_LEGAJO
FROM EMPLEADOS EM LEFT JOIN PLANIFICACIONLABORAL PL
ON (EM.NRO_LEGAJO = PL.NRO_LEGAJO)
WHERE PL.NRO_LEGAJO IS  NULL
```

C4

Plan de ejecución

id	select_type	table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	EM	index	NULL	FK_Empleados_Municipios	603	NULL	999682	100.00	Using index
1	SIMPLE	PL	ref	PRIMARY	PRIMARY	4	tesina.EM.nro_legajo	22	100.00	Using where; Using index;...

Figura 18

El plan de ejecución de la cláusula Left Join es similar, tanto en la parte Filtered como rows , al de las Cláusulas anteriormente analizadas, por lo que es de esperarse unos resultados similares a los anteriores.

Resultados

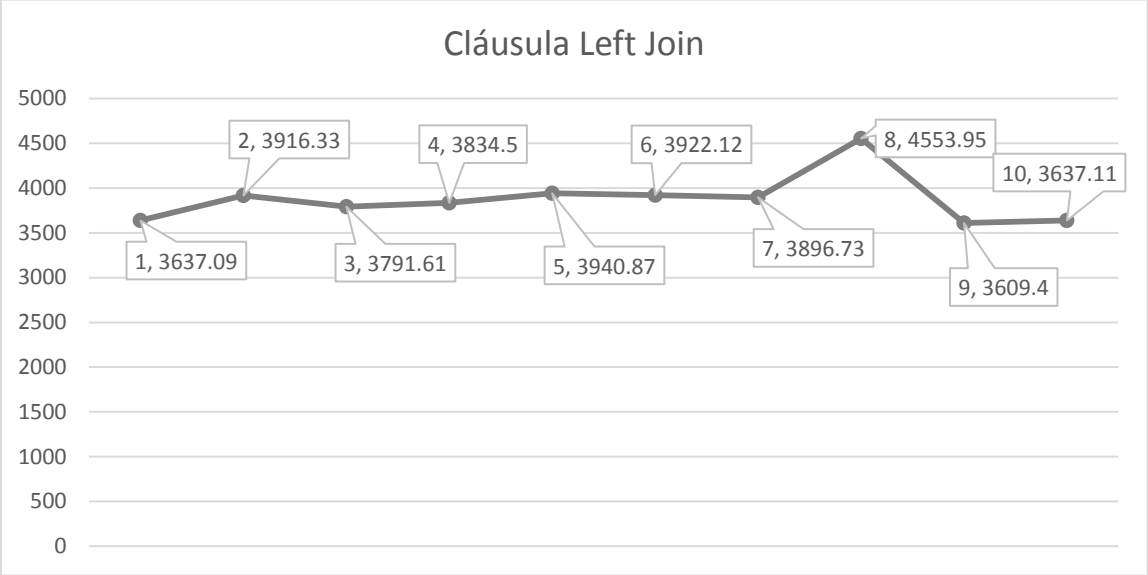


Gráfico 26

Del Gráfico se obtiene lo siguiente

Media aritmetica: 3873.97 segundos

Desviacion estandar: 271.21

Al ver los resultados, aparecen unos valores que sorprenden, sin importar lo que informaba el plan de ejecución, los tiempos de ejecución de cada prueba rondan en los 65 minutos. En promedio esta cláusula demoró aproximadamente 64 veces más que las anteriores.

Comparación

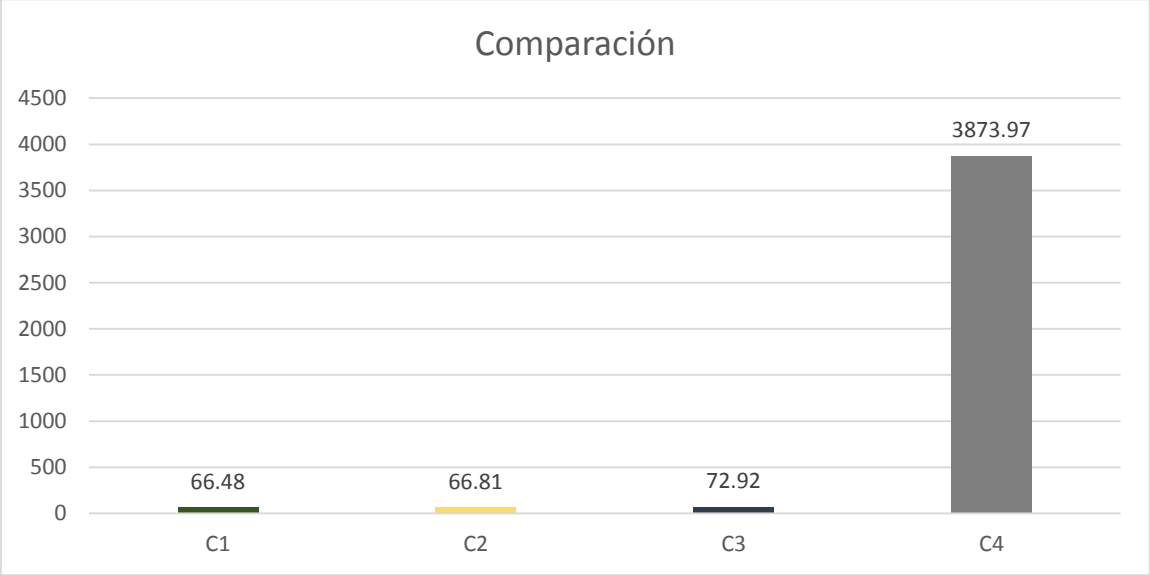


Gráfico 26

Como conclusión parcial, se puede decir que la cláusula Left Join posee una muy baja performance con respecto a las otras cláusulas, debido a que` el tiempo de ejecución aumento considerablemente.

4.6 Caso de estudio dos aplicado sobre el DBMS MySQL

4.6.1 Producto Cartesiano

```
SELECT COUNT(DISTINCT EM.nro_legajo)
FROM Empleados EM , PlanificacionLaboral PL
WHERE EM.nro_legajo = PL.nro_legajo
and PL.fecha BETWEEN '20160101' AND '20160130'
```

C5

Plan de ejecución

id	select_type	table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	EM	index	PRIMARY	FK_Empleados_Municipios	603	NULL	999682	100.00	Using index
1	SIMPLE	PL	ref	PRIMARY	PRIMARY	4	tesina.EM.nro_legajo	22	100.00	Using where; Using index

Figura 19

Resultados

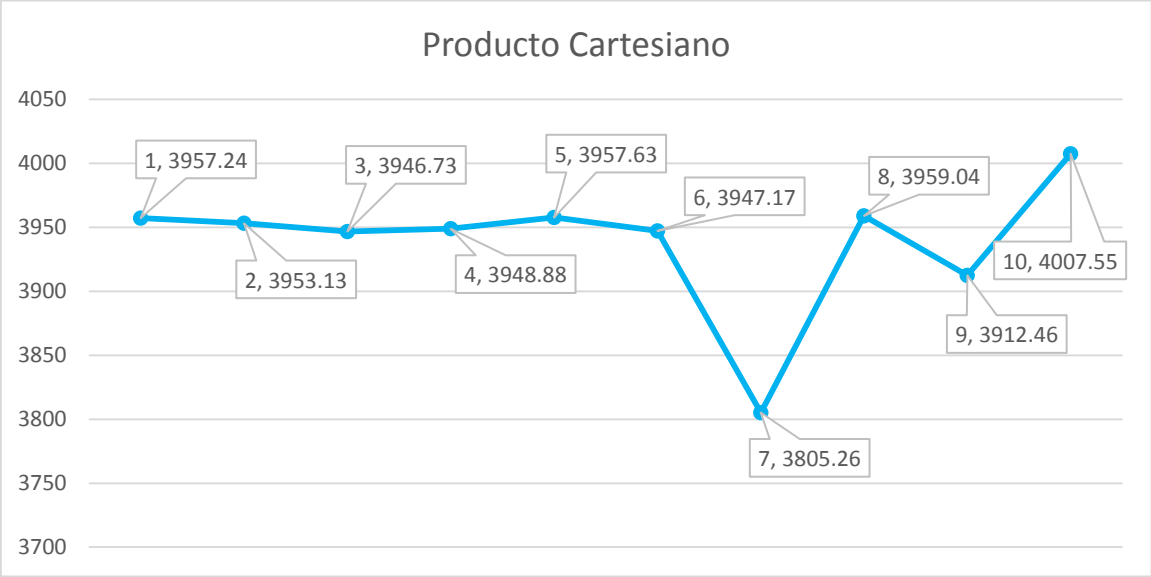


Gráfico 27

De los resultados se obtiene lo siguiente

Media aritmetica: 3939.50 segundos

Desviacion estandar: 52.48

4.6.2 Cláusula Inner Join

```
SELECT COUNT(DISTINCT EM.nro_legajo)
FROM Empleados EM inner join PlanificacionLaboral PL
on EM.nro_legajo = PL.nro_legajo
WHERe PL.fecha BETWEEN '20160101' AND '20160131'
```

C6

Plan de ejecución

id	select_type	table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	EM	index	PRIMARY	FK_Empleados_Municipios	603	NULL	999682	100.00	Using index
1	SIMPLE	PL	ref	PRIMARY	PRIMARY	4	tesina.EM.nro_legajo	22	100.00	Using where; Using index

Figura 20

El plan de ejecución de la cláusula Inner Join es similar al del producto cartesiano, por lo era esperable tener tiempo de ejecución equivalentes.

Resultados

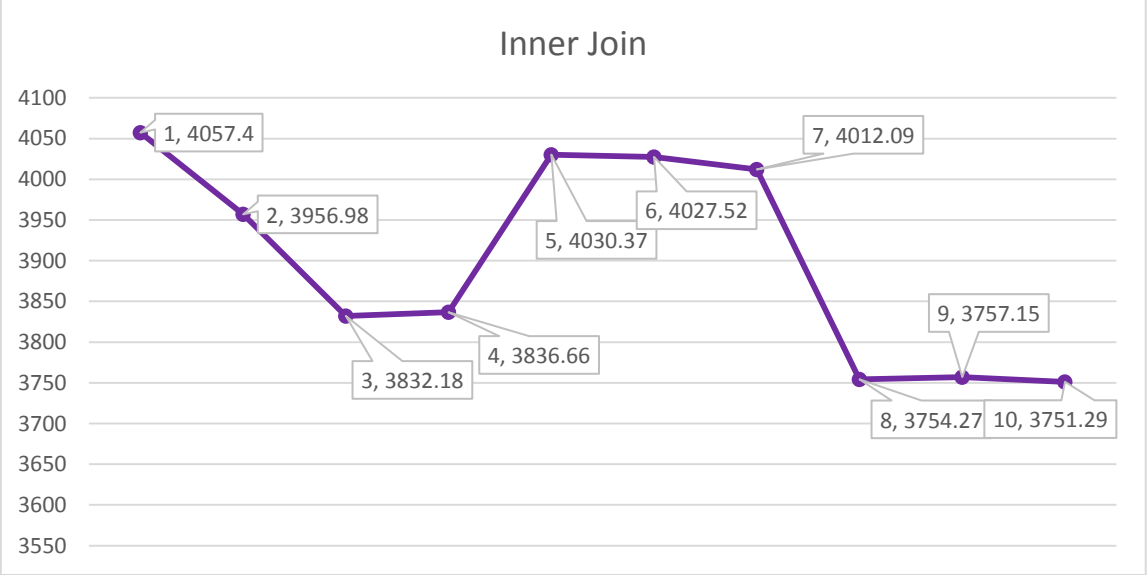


Gráfico 28

De los resultados se obtiene lo siguiente

Media aritmetica: 3901.59 segundos

Desviacion estandar: 127.44

Comparación

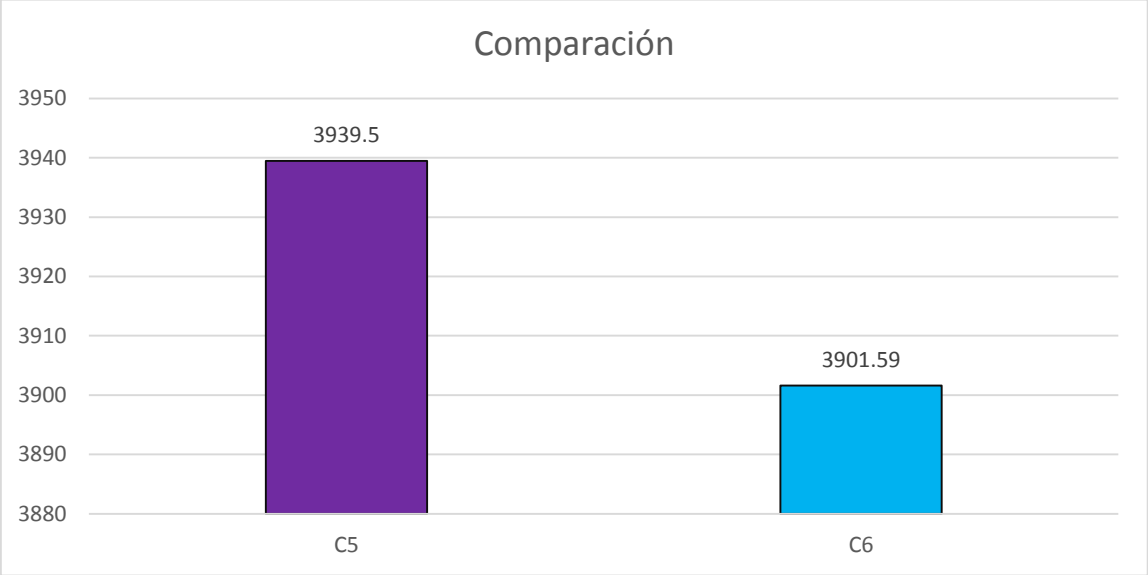


Gráfico 29

Cuando no se utiliza índices, a veces se puede percibir que MySQL tarda demasiado en responder una consulta. Es por este motivo que se decidió crear un índice con las mismas características que el índice Tesina001 utilizado por el SGBD de Microsoft.

A continuación se detalla la sentencia de creación de índices en el SGBD MySQL

```
CREATE INDEX Tesina001 ON
PlanificacionLaboral (nro_legajo);
```

Tamaño de índice: 600 Mb

4.7 Caso de estudio uno aplicado sobre el DBMS MySQL utilizando índices

4.7.1 Cláusula IN

Al igual que en el DMBS de Microsoft, la sintaxis de la consulta es idéntica a la consulta `C1`, ya que la incorporación de un índice a la tabla no afecta a la sintaxis de la consulta como se muestra a continuación.

```
SELECT EM.NRO_LEGAJO
FROM EMPLEADOS EM
WHERE EM.NRO_LEGAJO NOT IN (SELECT  NRO_LEGAJO
                            FROM PLANIFICACIONLABORAL)
```

CI1

Plan de ejecución

id	select_type	table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	EM	index	NULL	FK_Empleados_Municipios	603	NULL	1000725	100.00	Using where; Using index
2	DEPENDENT SUBQUERY	PLANIFICACIONLABORAL	index_subquery	PRIMARY,Tesina001	PRIMARY	4	func	22	100.00	Using index

Figura 21

Como se puede ver en la Figura 15 el DMBS comienza hacer uso del índice Tesina001 de la tabla *PlanificacionLaboral,* columna posible_keys

Resultados

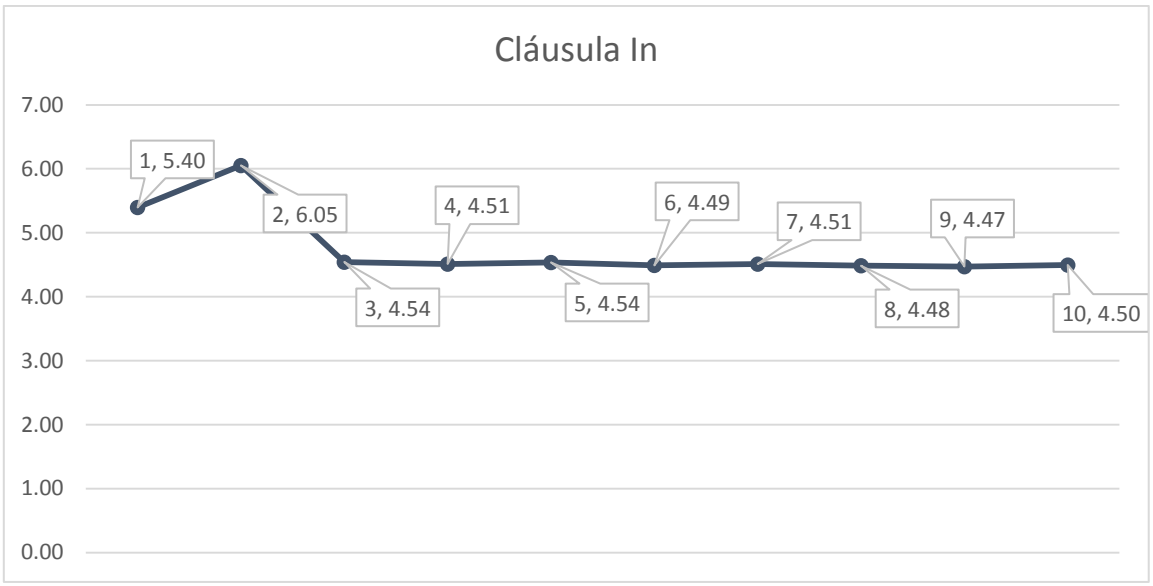


Gráfico 30

De los resultados se obtiene lo siguiente

Media aritmetica: 4.74 segundos

Desviacion estandar: 0.5

En el Gráfico 30 se observa un cambio considerable en los tiempos de ejecución de las pruebas realizadas en `CI1` respecto de `C1`. A partir de estos valores, se puede afirmar, que en este entorno, la incorporación del índice mejora la performance de la cláusula IN.

Comparación `C1` & `CI1`

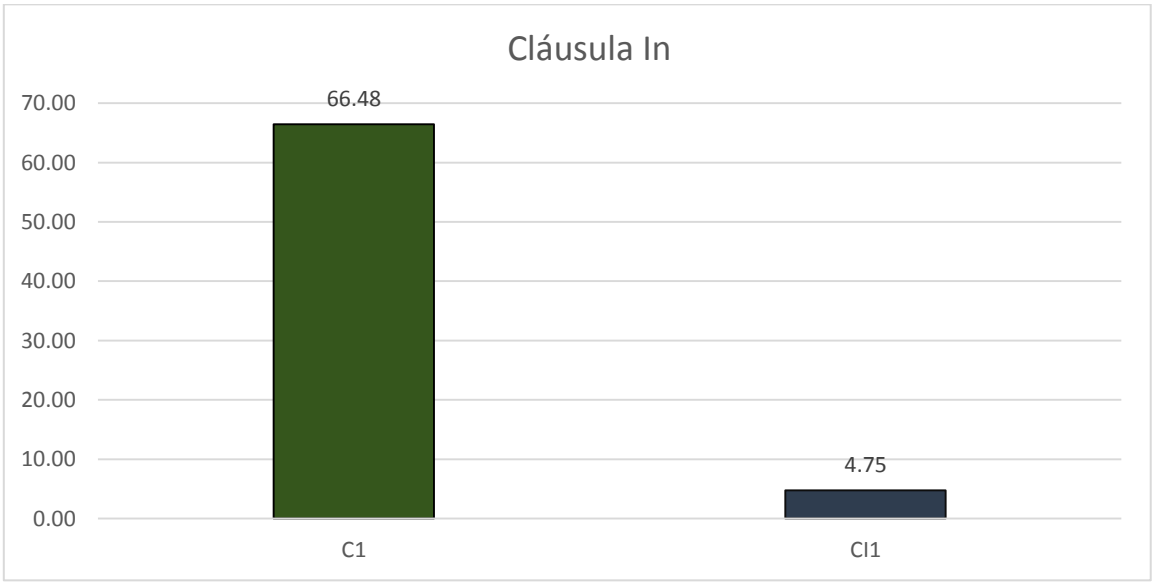


Gráfico 31

4.7.2 Cláusula IN con Distinct

```
SELECT EM.NRO_LEGAJO
FROM EMPLEADOS EM
WHERE EM.NRO_LEGAJO NOT IN (SELECT DISTINCT  NRO_LEGAJO
                            FROM PLANIFICACIONLABORAL)
```

CI2

Plan de ejecución

id	select_type	table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	EM	index	NULL	FK_Empleados_Municipios	603	NULL	999682	100.00	Using where; Using index
2	DEPENDENT SUBQUERY	PLANIFICACIONLABORAL	index_subquery	PRIMARY	PRIMARY	4	func	22	100.00	Using index

Figura 22

Como era de esperarse la incorporación de la cláusula Distinct dentro de la sub consulta no genera cambios en el plan de ejecución.

Resultados

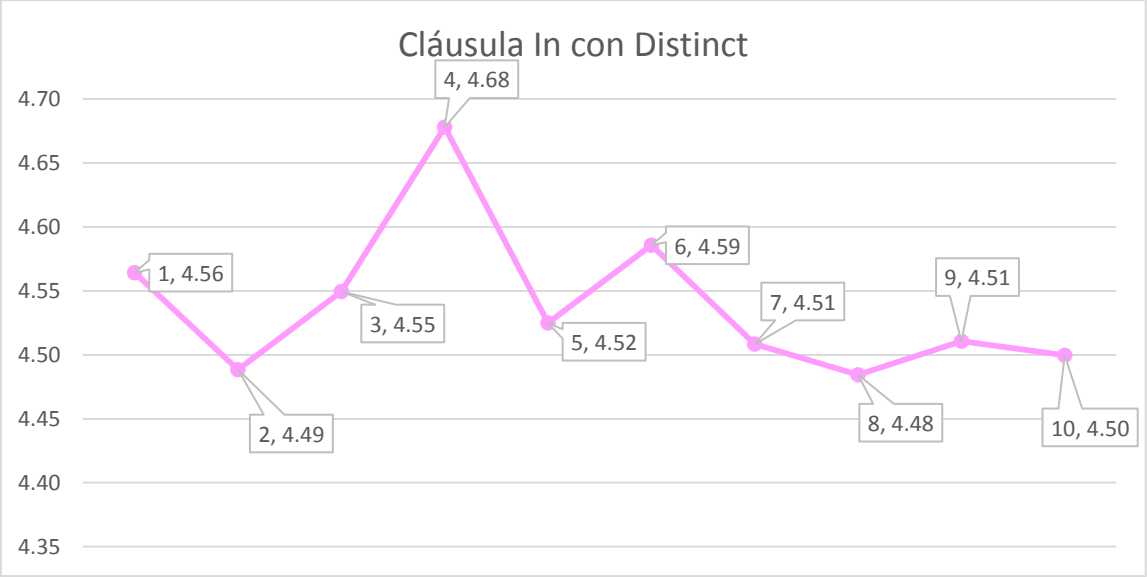


Gráfico 32

De los resultados se obtiene lo siguiente

Media aritmetica: 4.53 segundos

Desviacion estandar：0.05

Comparación C2 & CI2

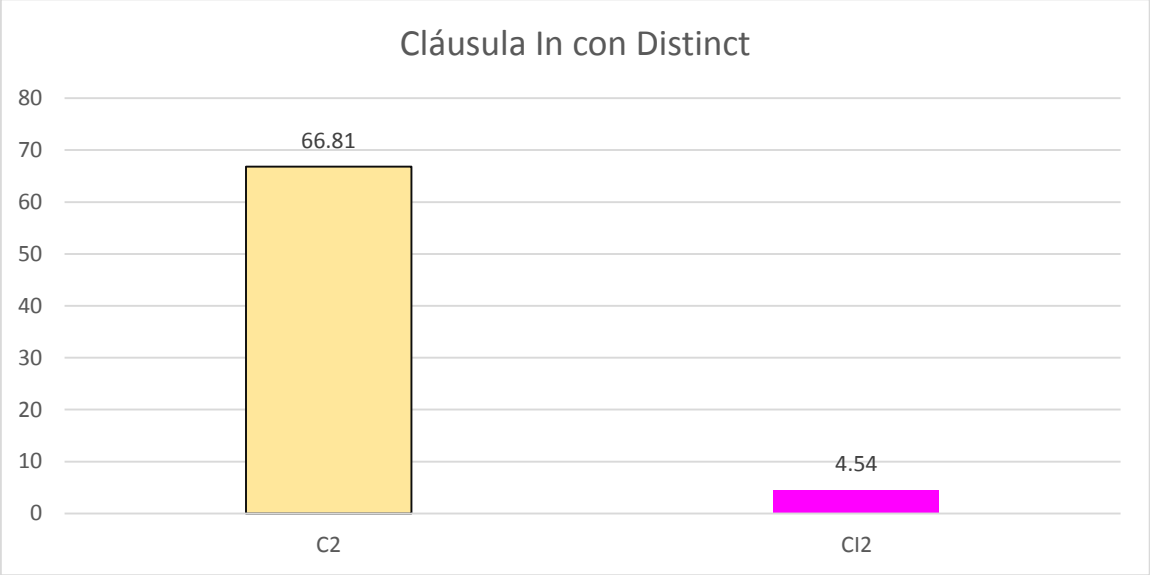


Gráfico 33

Los resultados están a la vista, teniendo en cuenta que la incorporación del distinct no produce una mejora notable en los tiempos de ejecución de cada prueba, el índice Tesina001 cumple con lo esperado, en este caso mejorar la performance.

4.7.3 Cláusula Exists

```
SELECT NRO_LEGAJO
FROM EMPLEADOS EM
WHERE NOT EXISTS (SELECT PL.NRO_LEGAJO
                  FROM PLANIFICACIONLABORAL PL
                  WHERE EM.NRO_LEGAJO = PL.NRO_LEGAJO)
```

CI3

Plan de ejecución

id	select_type	table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	EM	index	NULL	FK_Empleados_Municipios	603	NULL	999682	100.00	Using where; Using index
2	DEPENDENT SUBQUERY	PLANIFICACIONLABORAL	index_subquery	PRIMARY	PRIMARY	4	func	22	100.00	Using index

Figura 23

Resultados

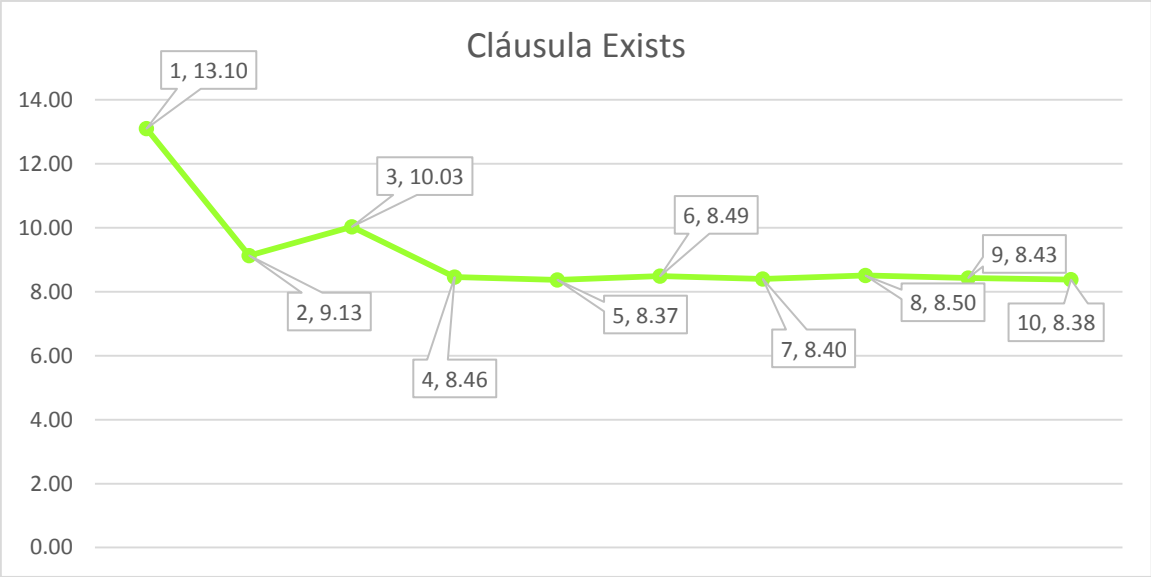


Gráfico 34

De los resultados se obtiene lo siguiente

Media aritmetica: 9.21 segundos

Desviacion estandar：1.41

Comparación `C3 & CI3`

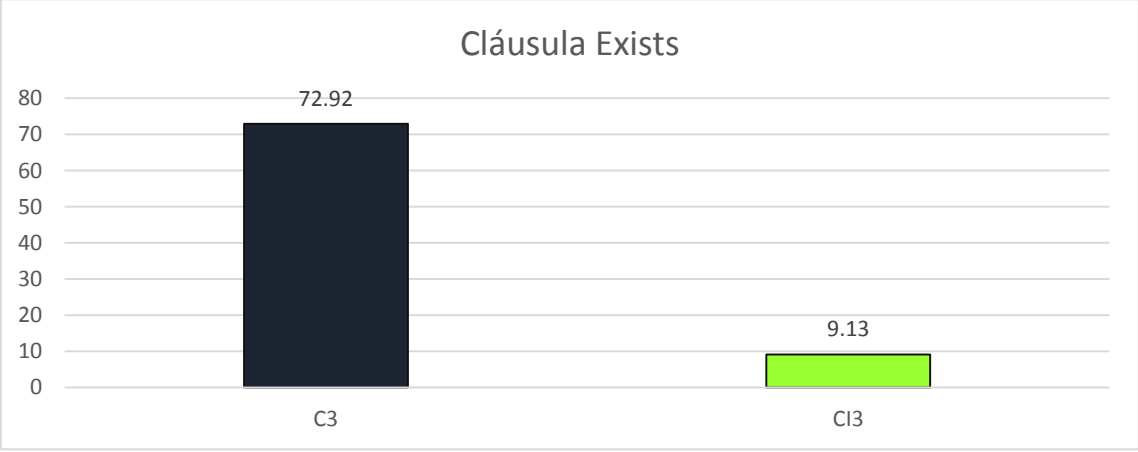


Gráfico 35

Al comparar los resultados entre C3 y CI3, está más que claro que la incorporación del índice en produce una mejorar de performance en la cláusula Exists.

4.7.4 Cláusula Left Join

```
SELECT EM.NRO_LEGAJO
FROM EMPLEADOS EM LEFT JOIN PLANIFICACIONLABORAL PL
ON (EM.NRO_LEGAJO = PL.NRO_LEGAJO)
WHERE PL.NRO_LEGAJO IS  NULL
```

CI4

Plan de ejecución

id	select_type	table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	EM	index	NULL	FK_Empleados_Municipios	603	NULL	999682	100.00	Using index
1	SIMPLE	PL	ref	PRIMARY	PRIMARY	4	tesina.EM.nro_legajo	22	100.00	Using where; Using index;...

Figura 24

Al igual que en las pruebas sin índices el plan de ejecución de la cláusula Left Join es similar, tanto en la parte Filtered como rows , al de las Cláusulas anteriormente analizadas, por lo que es de esperarse unos resultados similares a los anteriores.

Resultados

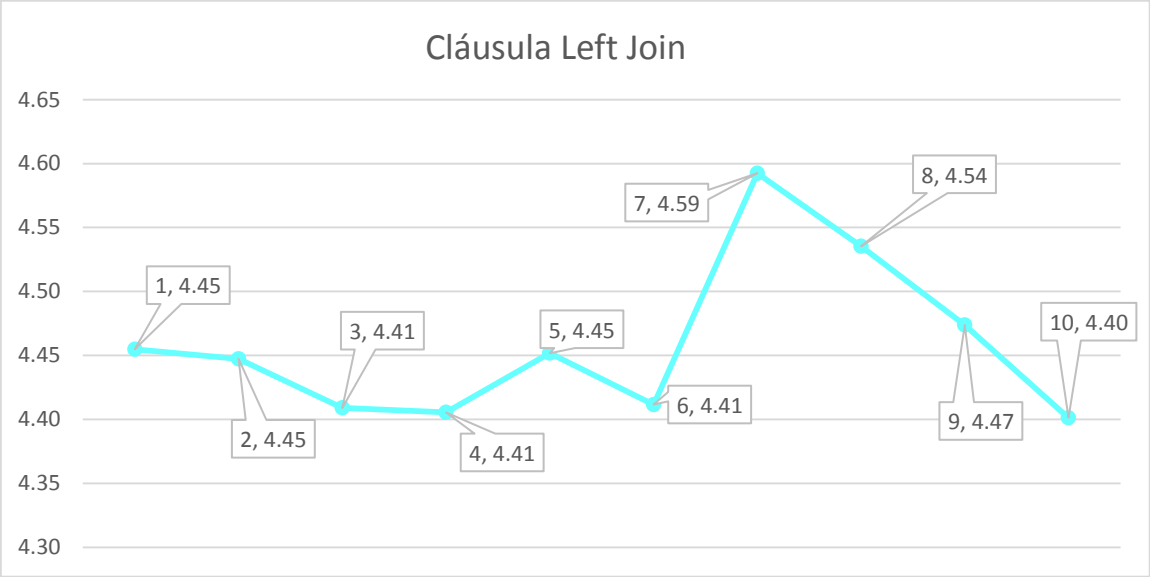


Gráfico 36

De los resultados se obtiene lo siguiente

Media aritmetica: 4.45 segundos

Desviacion estandar: 0.05

Comparación `C4 & CI4`

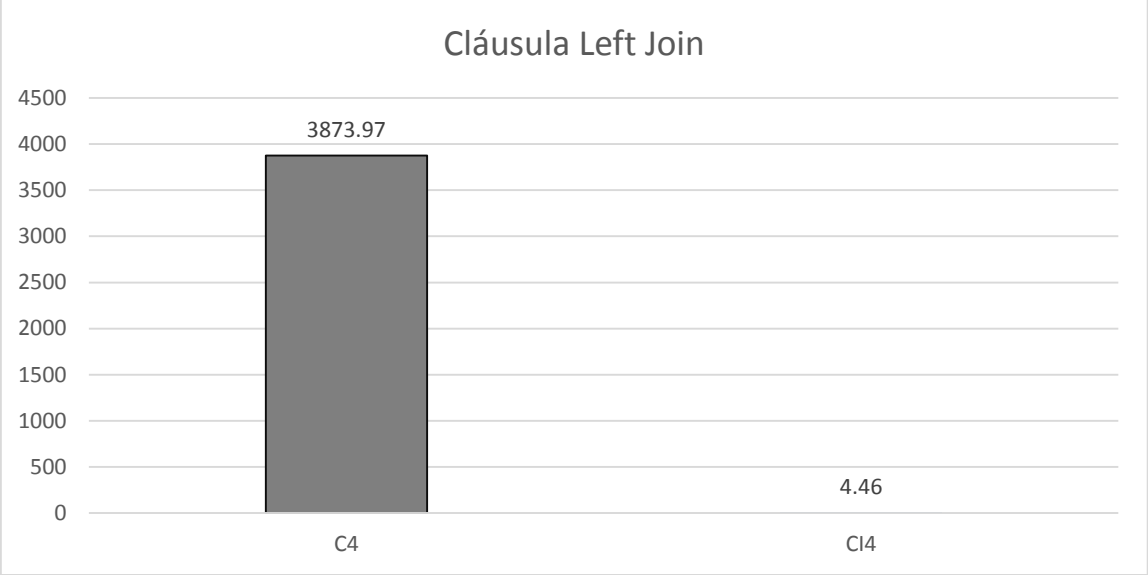


Gráfico 37

Los resultados demuestran la importancia de la incorporación del índice Tesina001 en la tabla PlanificacionLaboral. El tiempo de ejecución de la cláusula Left Join se decremento en promedio unas 868 veces.

Comparación general

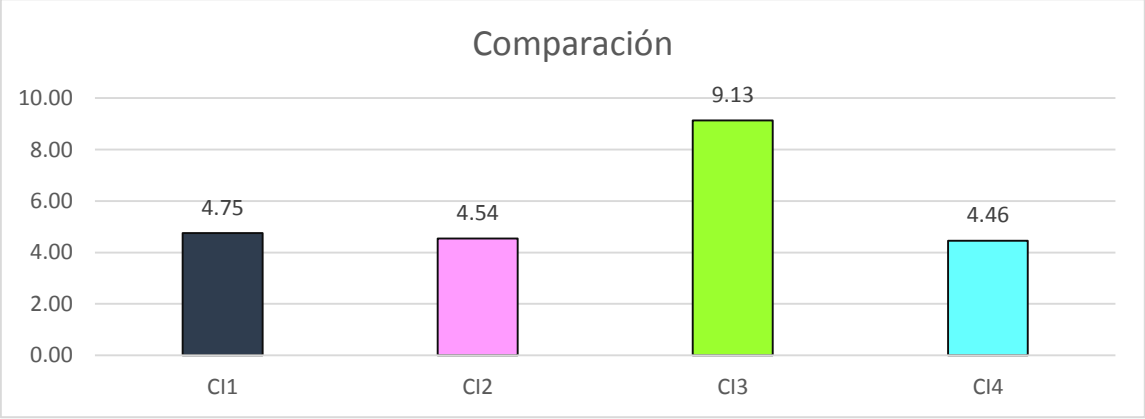


Gráfico 38

La comparación general entre las diferentes cláusulas demuestra que no hay grandes diferencias en los tiempos de ejecución al utilizar diferentes Cláusulas cuando se tiene un índice, al menos en este entorno dedicado.

4.8 Caso de estudio dos aplicado sobre el MySQL utilizando índices

4.8.1 Producto Cartesiano

```
SELECT COUNT(DISTINCT EM.nro_legajo)
FROM Empleados EM , PlanificacionLaboral PL
WHERE EM.nro_legajo = PL.nro_legajo
and PL.fecha BETWEEN '20160101' AND '20160130'
```

CI5

Plan de ejecución

id	select_type	table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	EM	index	PRIMARY	FK_Empleados_Municipios	603	NULL	1000725	100.00	Using index
1	SIMPLE	PL	ref	PRIMARY,Tesina001	Tesina001	4	tesina.EM.nro_legajo	23	100.00	Using where; Using index

Como se observó en los casos anteriores, la incorporación del índice influye en el plan de ejecución del producto cartesiano. Por lo tanto se esperó que lo tiempos de ejecución también se vean afectados

Resultados

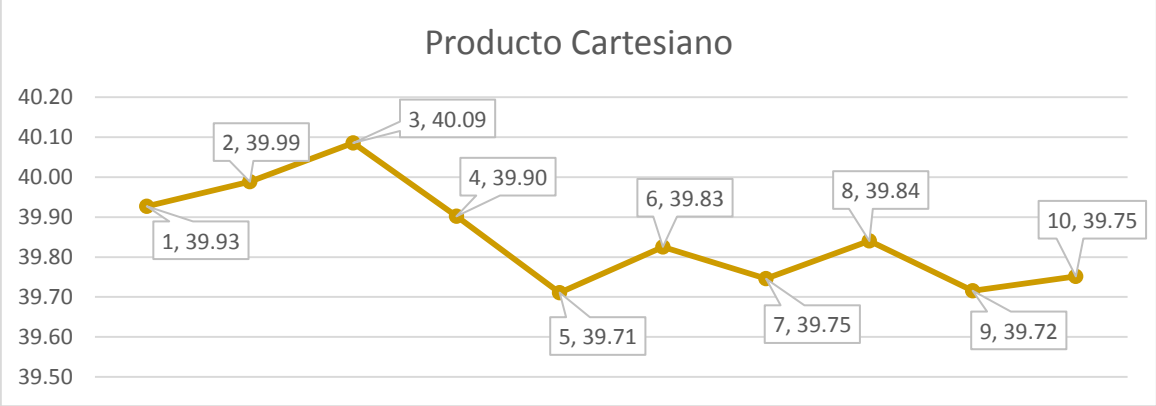


Gráfico 39

De los resultados se obtiene lo siguiente

Media aritmetica: 39.85 segundos

Desviacion estandar: 0.11

Comparación `C5` & `CI5`

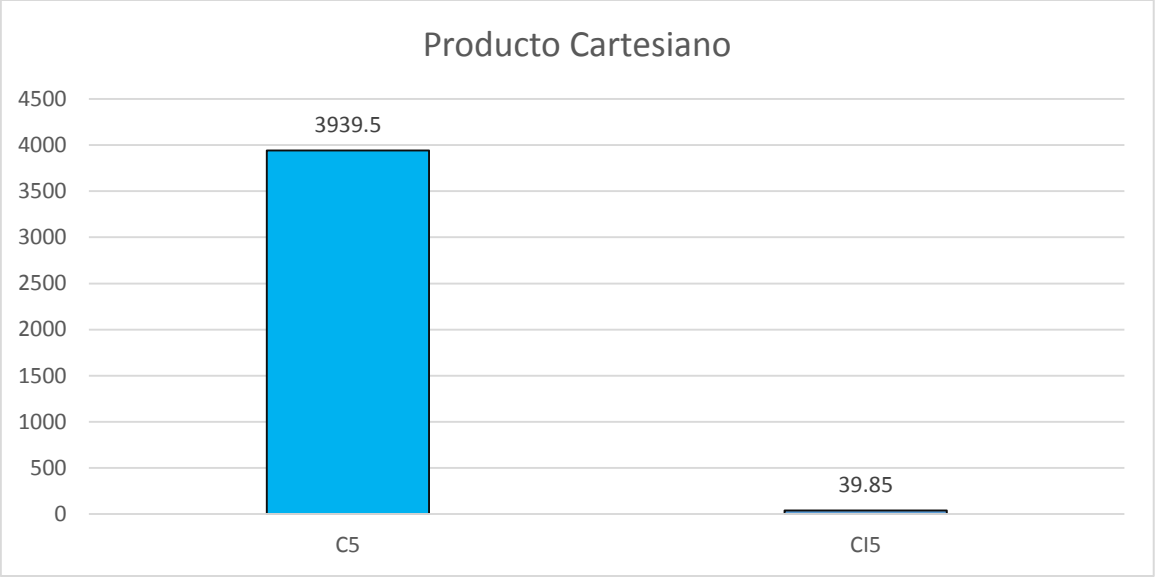


Gráfico 40

Los resultados estan a la vista, el tiempo de ejecucion del producto cartesiano al incorporar un indice ,en este caso Tesina001, es en promedio casi 100 veces menor.

4.8.3 Cláusula Inner Join

```
SELECT COUNT(DISTINCT EM.nro_legajo)
FROM Empleados EM inner join PlanificacionLaboral PL
on EM.nro_legajo = PL.nro_legajo
WHERe PL.fecha BETWEEN '20160101' AND '20160131'
```

CI6

Plan de ejecución

id	select_type	table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	EM	index	PRIMARY	FK_Empleados_Municipios	603	NULL	1000725	100.00	Using index
1	SIMPLE	PL	ref	PRIMARY,Tesina001	Tesina001	4	tesina.EM.nro_legajo	23	100.00	Using where; Using index

Figura 25

Como se mencionó antes, tanto el Producto Cartesiano como el Inner Join no presentan diferencias en su plan de ejecución, este se debe a que el optimizador de consulta internamente se encarga de resolver dicha consulta de la forma que mejor le parece sin importar que tipo de cláusula se utilizó.

Resultados

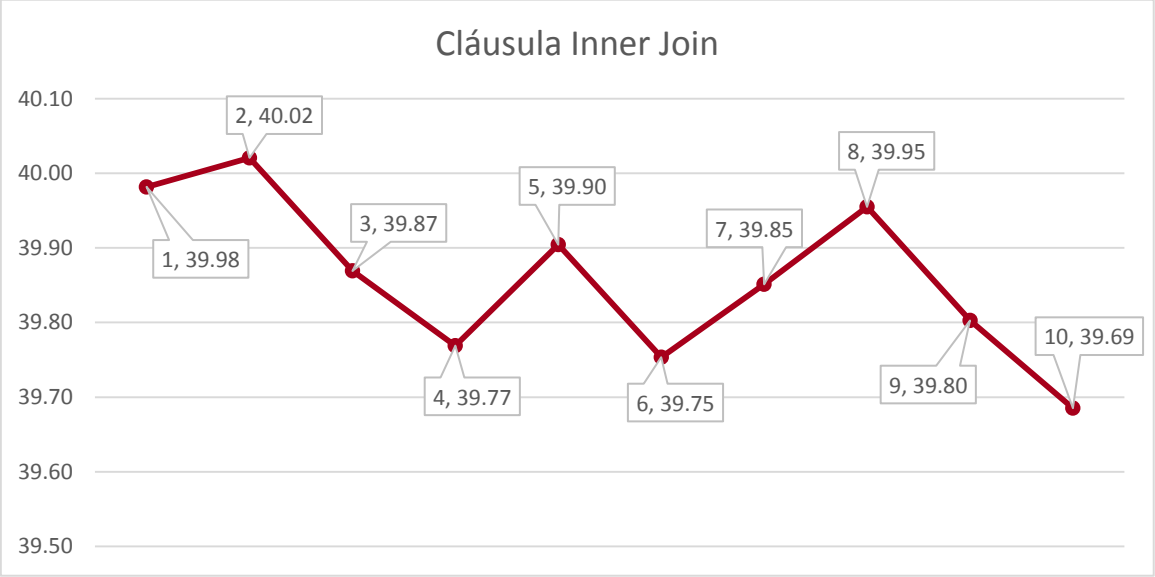


Gráfico 41

De los resultados se obtiene lo siguiente

Media aritmetica: 39.85 segundos

Desviacion estandar: 0.1

Comparación C5 & CI5

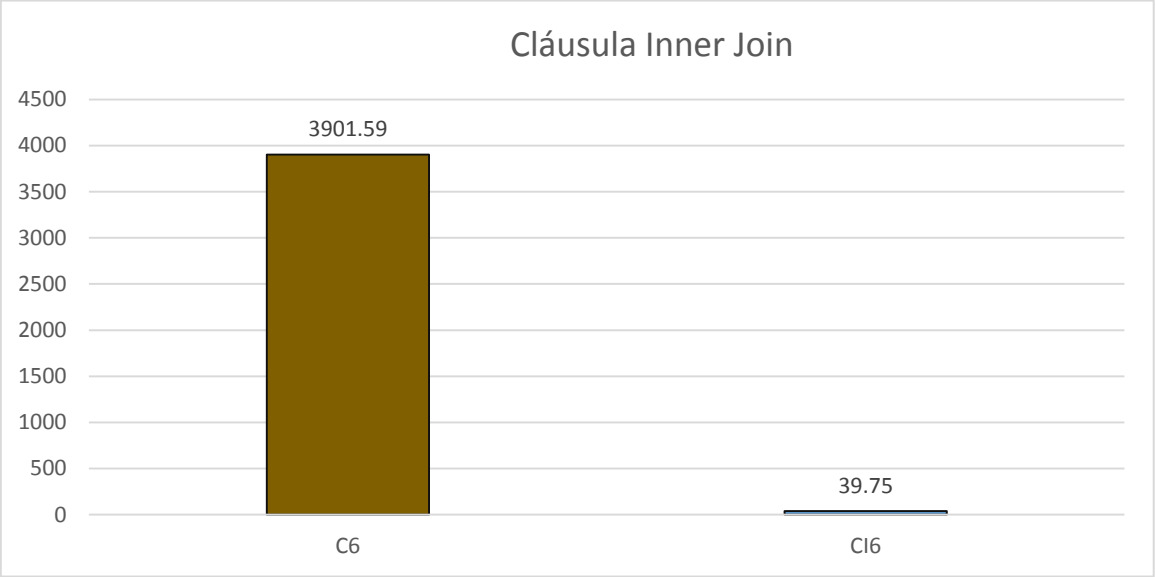


Gráfico 42

Como se observó en los otros casos, el índice Tesina001 produjo en gran cambio en el tiempo de ejecución de la cláusula Inner Join.

Comparación general

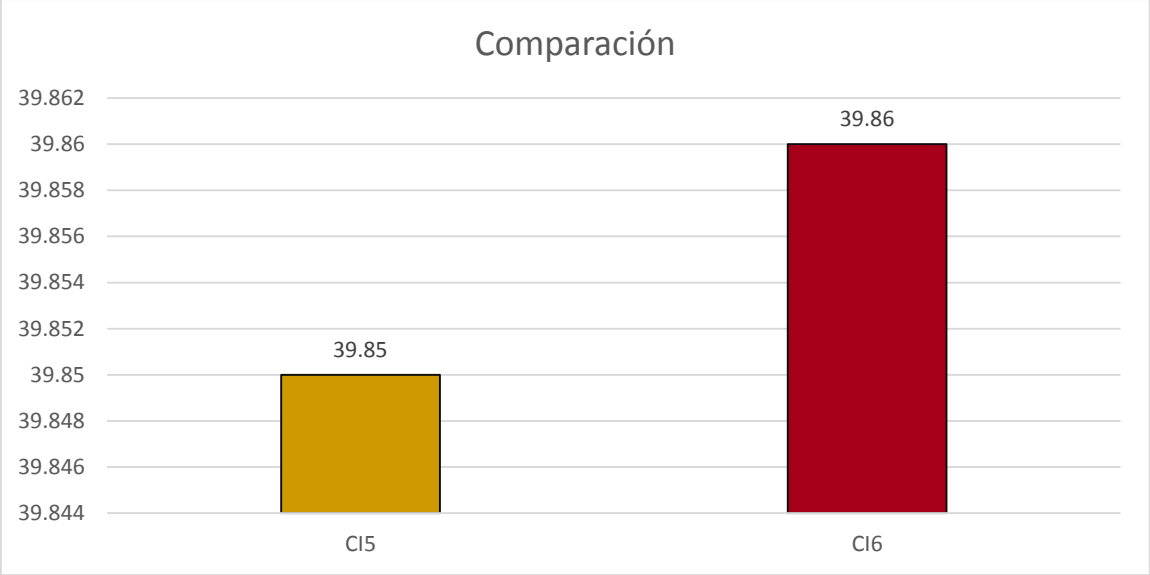


Gráfico 43

La comparación general entre el producto cartesiano y la cláusula Inner Join no muestra diferencias,como se menciono antes, esto se debe a que el optimizador de consultas opta por resolver de la forma mas efeciente la consulta.

Capítulo 5

Comparación de cláusulas en los distintos SGBD.

A continuación, se mostrarán comparaciones entre los distintos SGBD de cada una de las cláusulas sin índices que fueron utilizadas como caso de estudio.

5.1 Cláusula IN

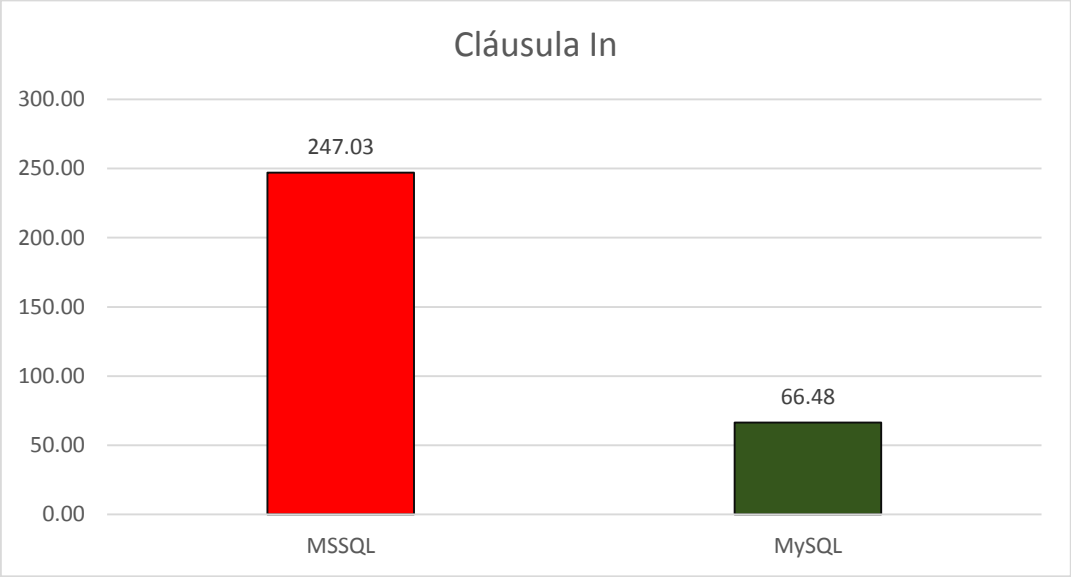


Gráfico 44

5.2 Cláusula IN con Distinct

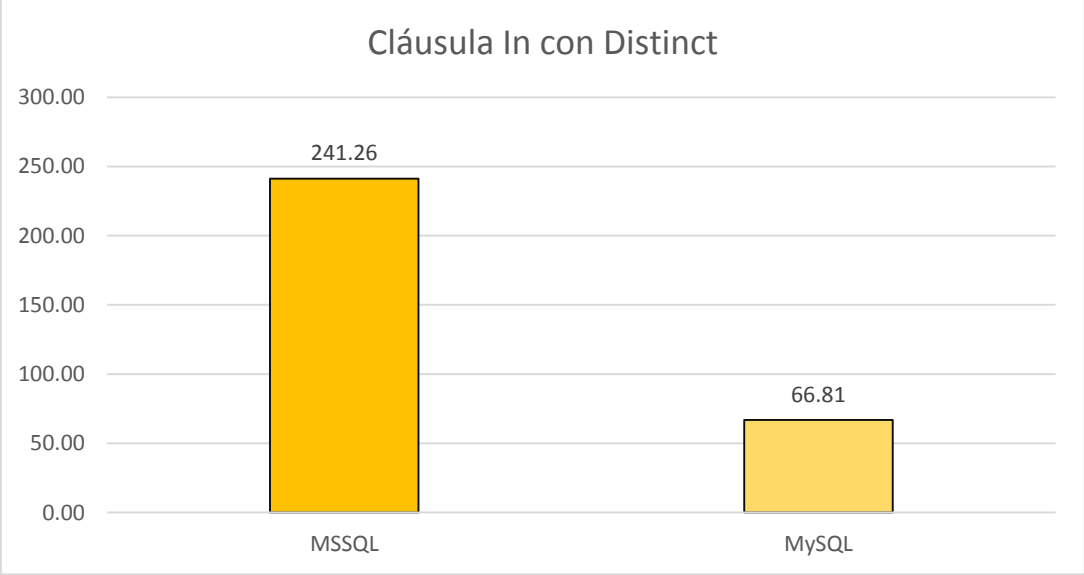


Gráfico 45

5.3 Cláusula Exists

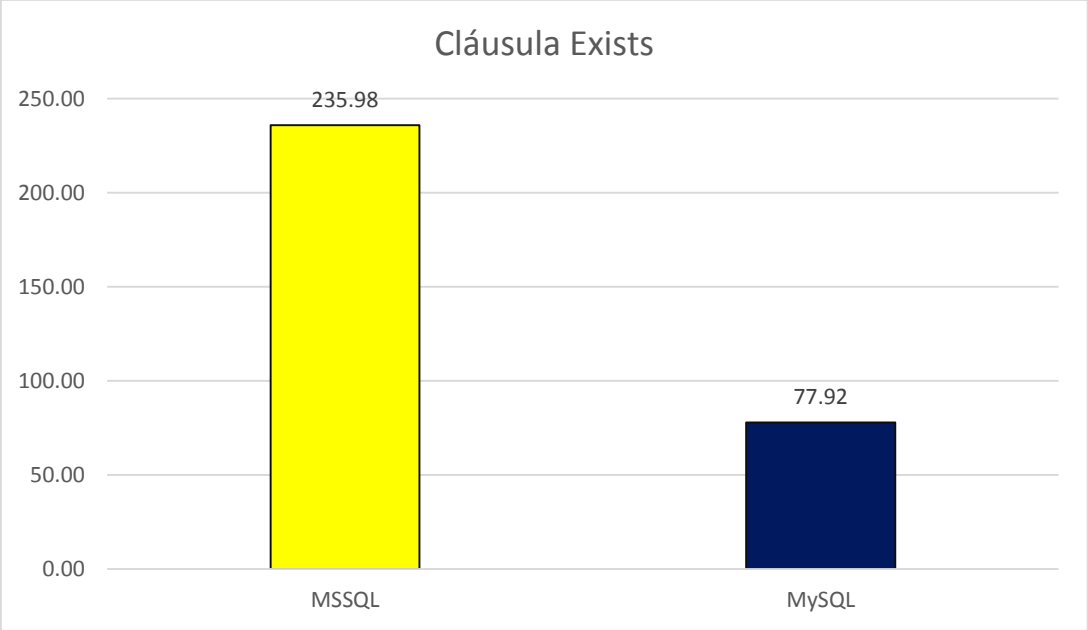


Gráfico 46

5.4 Cláusula Left Join

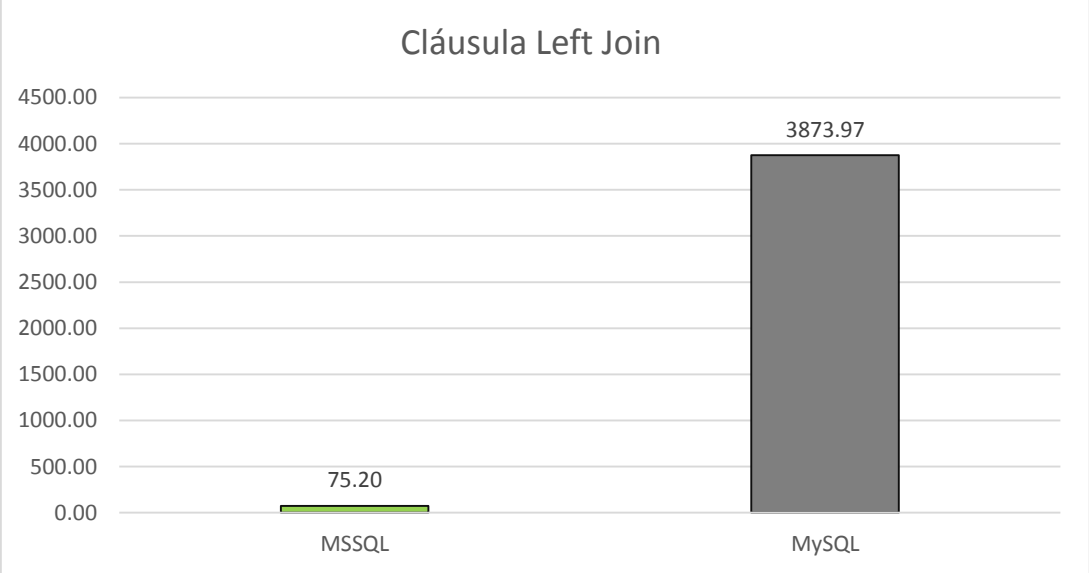


Gráfico 46

5.5 Producto Cartesiano

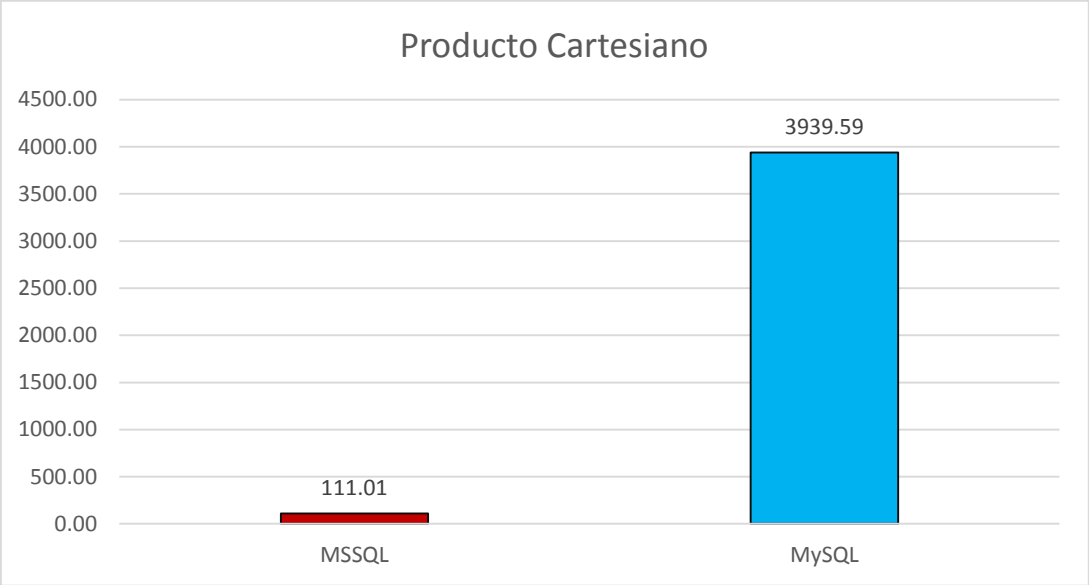


Gráfico 47

5.6 Cláusula Inner Join

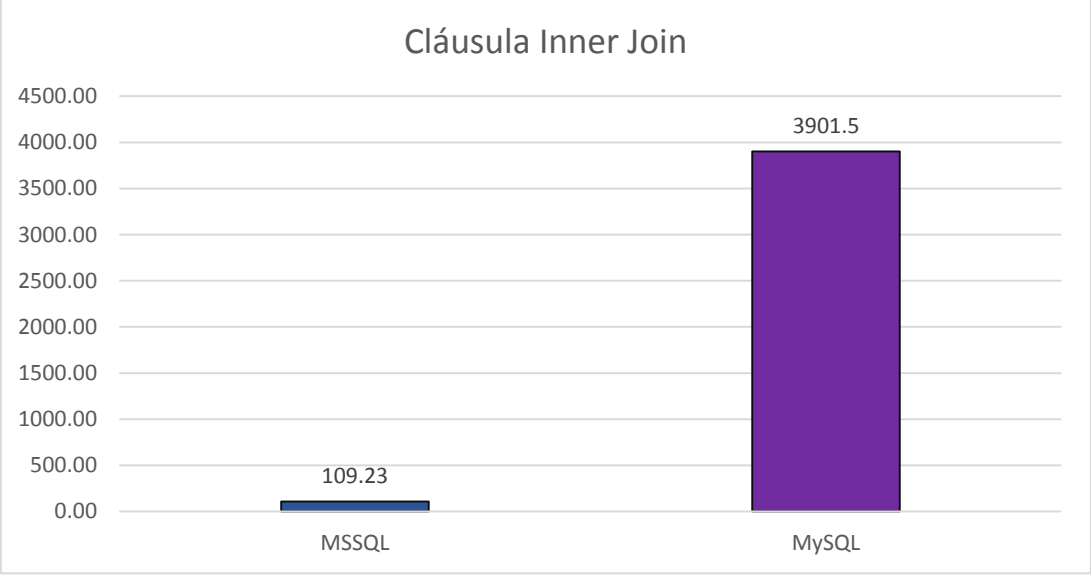


Gráfico 48

Una vez que se realizó todas las comparaciones sin índices se procedió a realizar las mismas comparaciones entre Cláusulas pero con índices.

5.7 Cláusula IN

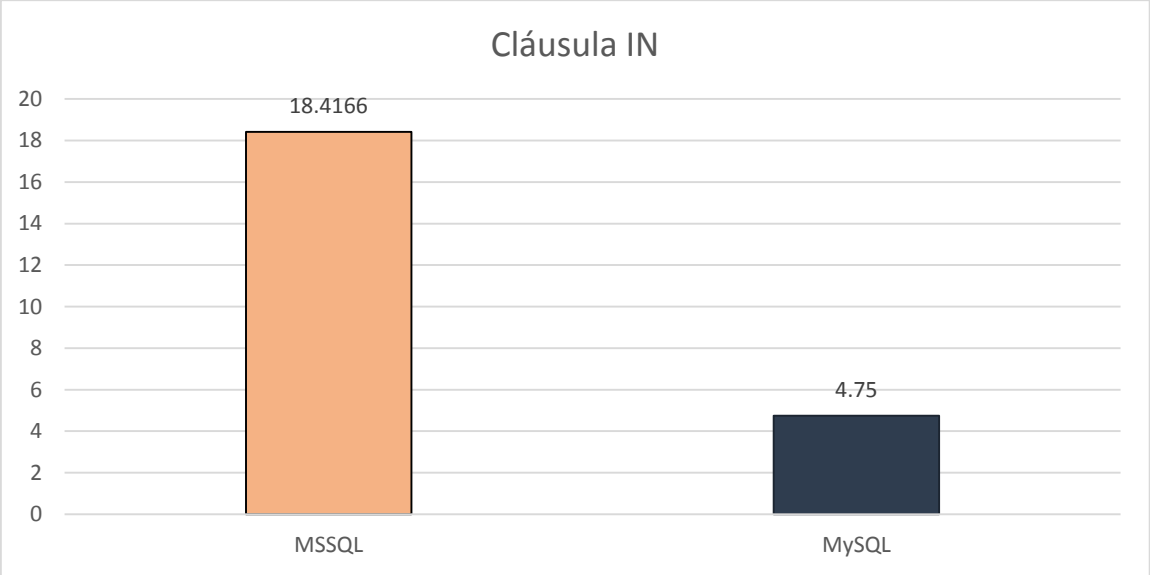


Gráfico 49

5.8 Cláusula IN con Distinct

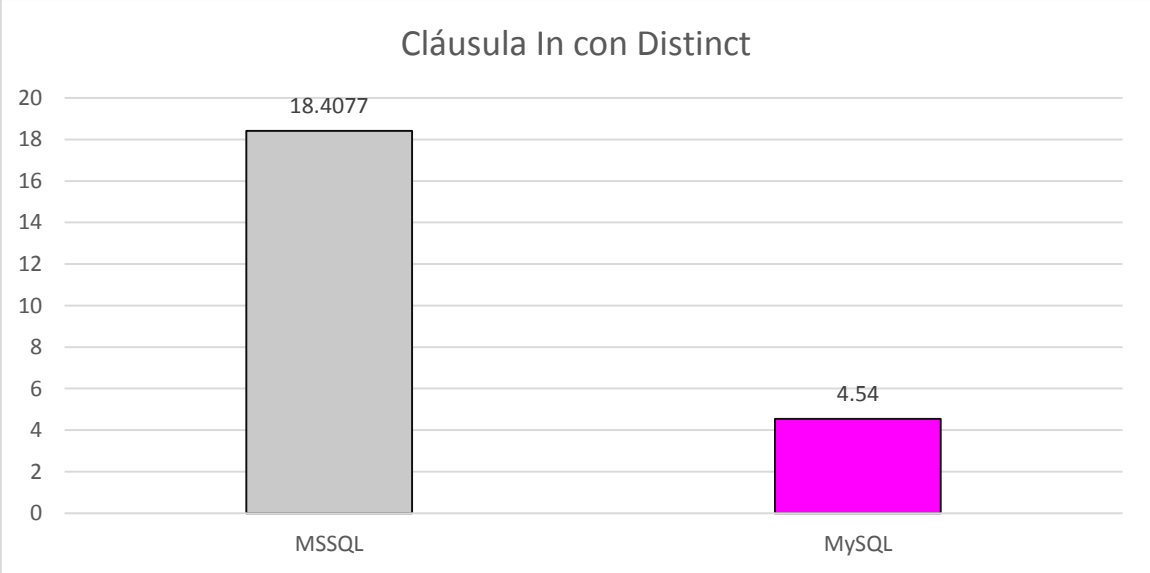


Gráfico 50

5.9 Cláusula Exists

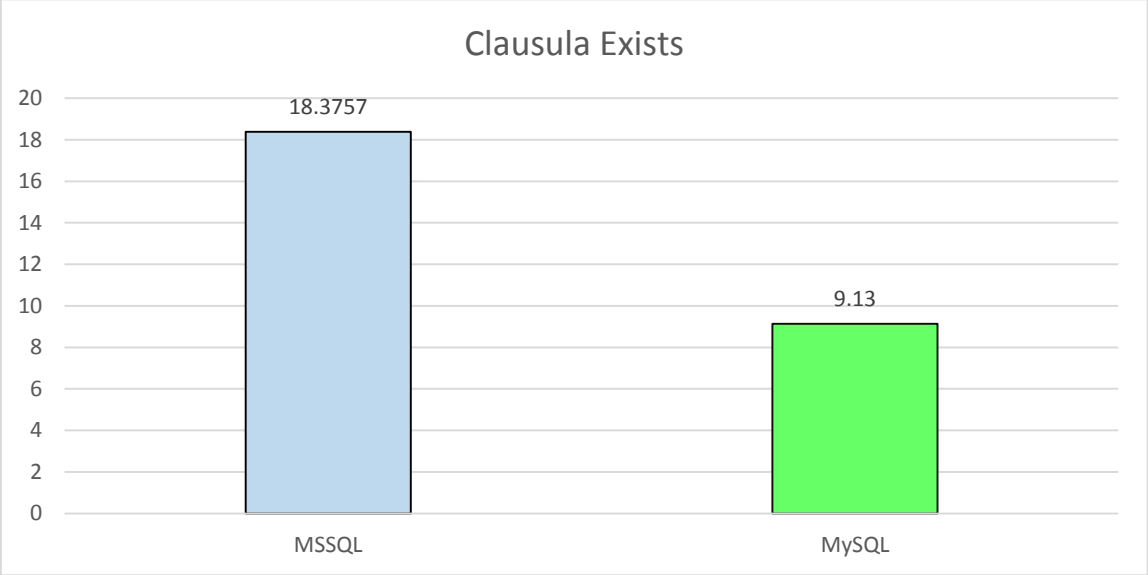


Gráfico 51

5.10 Cláusula Left Join

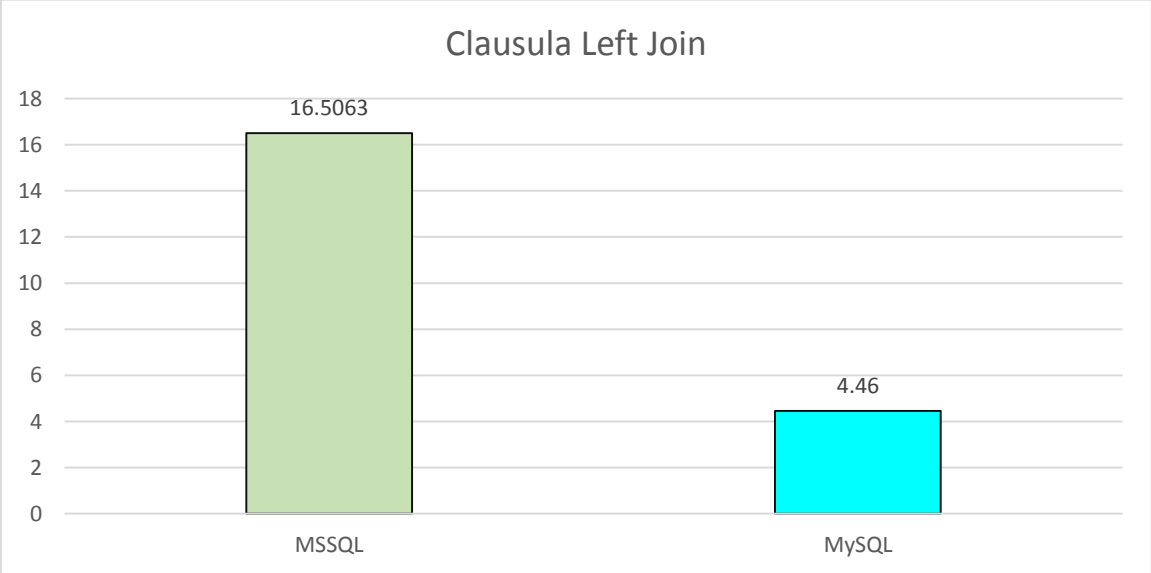


Gráfico 52

5.11 Producto Cartesiano

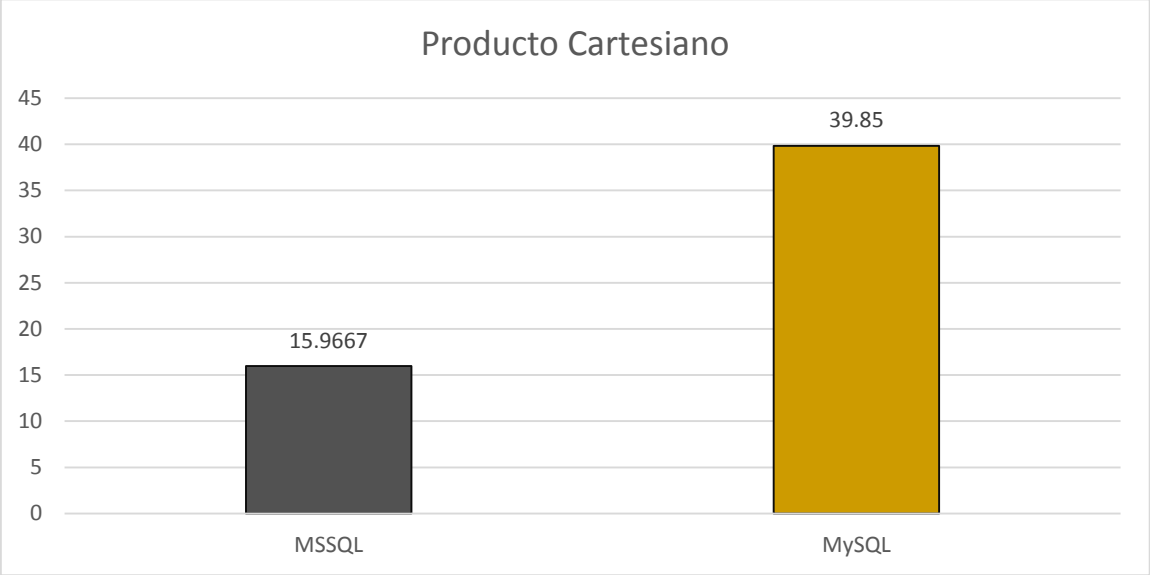


Gráfico 53

5.12 Clausula Inner join

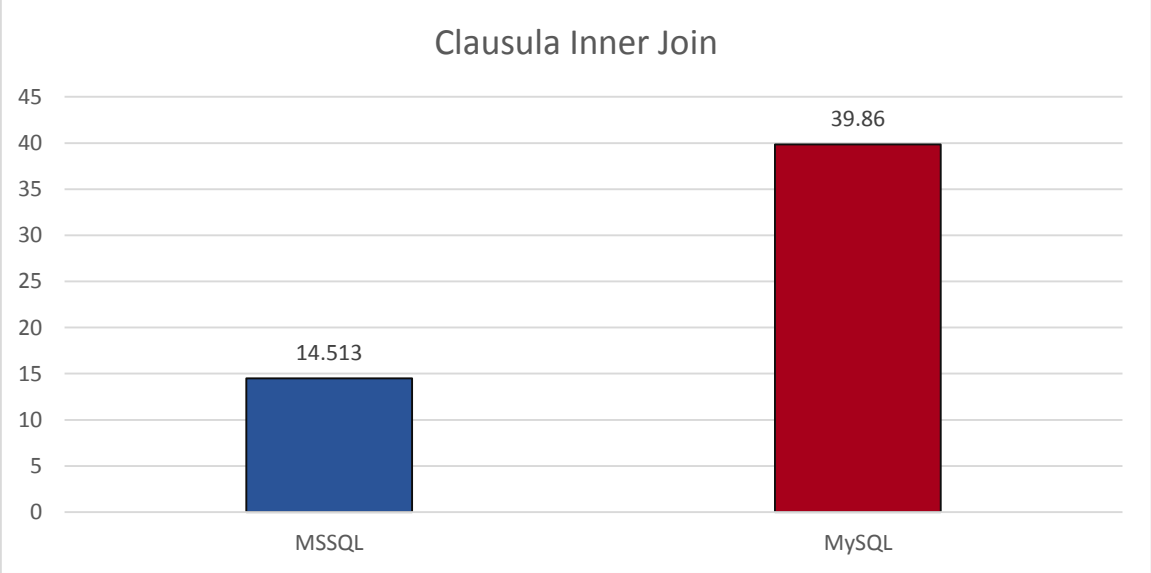


Gráfico 54

Capítulo 6

Conclusiones

En esta tesina de grado se presentó varios casos de estudio a los cuales se les evaluó la performance a través de varias pruebas en un ambiente dedicado. Dichas pruebas arrojaron una serie de resultados de los cuales se puede decir que:

Tanto en el SGBD de Microsoft como en el de Oracle, el optimizador de consulta desestimo el tipo de clausula utilizada ya que cada resultado obtenido era similar entre los casos de estudio, dicha afirmación aplica para las clausulas IN, Exists y Except. Pero un caso de estudio resalto entre todos, el Left Join.

En el SGBD de Microsoft la utilización de la cláusula Left Join presento una mejoría notable en cuanto al tiempo de ejecución tomado para recuperar los datos, esto se debió al grado de paralelismo que utilizo optimizador de consultas para resolver dicho requerimiento. Si el optimizador de consultas determina que el número de filas es demasiado alto, como en el caso de la tabla PlanificacionLaboral, este proporciona operadores de intercambio para distribuir las filas. En consecuencia, los operadores se ejecutan en paralelo.

Pero en el SGBD de Oracle, no sucedió lo mismo y los tiempos de ejecución de la clúsula Left Join se dispararon con respecto al In y Exists.

Con respecto a la cláusula In y Exists, se puede afirmar que son mucho más eficientes en cuanto tiempo de ejecución en el DMBS de Oracle.

Para los casos de estudio del producto cartesiano y la cláusula Inner Join no se observaron grandes diferencias entre sí, por lo que no se puede afirmar que en la práctica el Inner Join es más eficiente que el producto cartesiano y este es un claro ejemplo donde la teoría difiere de la práctica.

Tanto el producto cartesiano como el Inner join se ejecutaron más rápido en el DMBS de Microsoft que en el SGBD de Oracle.

Se sabe que al incorporar un índice en una tabla la performance de la consulta que utilice ese índice en teoría se vería mejorada, y eso fue lo que paso para cada uno de los casos de estudios el tiempo de ejecución se vio reducido considerablemente, pero se debe tener en cuenta que la incorporación del índice reduce el tiempo de ejecución mientras que hace más lenta las operación de Insert

Update o Delete, esto se debe a que por cada una de estas operación el índice debe ser actualizado.

Como resultado final, se obtuvo un excelente rendimiento en todos los casos de estudios en ambos DMBS, no obstante a eso se cumplió que las clausulas In , Exists y Left Join fueron más eficientes en cuanto a tiempo de ejecución en el DMBS de Oracle respecto al de Microsoft pero el producto cartesioano y el inner join resultaron ser más eficientes en el SGBD de Microsoft.

Bibliografía

[1] Introduccion a las Bases de Datos. Fundamentos y Diseño, Rodolfo Bertone/Plablo Thomas

[2] Chapple, Mike. SQL Fundamentals

[3] Bases de Datos Modelos, Lenguajes, Diseños James L. Johnson. Sistemas de Bases deDatos Elmasri / Nawathe.

[4] Fundamentos de Bases de Datos Henry Korth/ Abraham Silverschatz/ Sudargham.

[5] Microsoft, Conceptos básicos de los índices.

[6] https://technet.microsoft.com/es-es/library/ms175913(v=sql.105).aspx – Iconos del plan de ejecución.

[7] https://dev.mysql.com/doc/refman/5.6/en/execution-plan-information.html

[8] Fundamentos a las bases de datos – Cuarta edición – Abraham Silberschatz/Henry F. Korth

[9] https://es.wikipedia.org/wiki/Microsoft_SQL_Server

[10] https://downloads.mysql.com/docs/refman-5.6-en.pdf