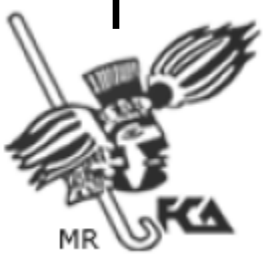
Universidad Nacional Autónoma de México

Facultad de Contaduría y Administración

Optimización del generador de concordancias del Corpus Histórico del Español en México (CHEM) mediante una comparación entre manejadores de bases de datos relacionales y administración de desempeño de bases de datos.

Tesis profesional

Julio César Vargas Mejía



México, D.F. **2013**

Universidad Nacional Autónoma de México

Facultad de Contaduría y Administración

Optimización del generador de concordancias del Corpus Histórico del Español en México (CHEM) mediante una comparación entre manejadores de bases de datos relacionales y administración de desempeño de bases de datos.

Tesis profesional
Que para obtener el título de:
Licenciado en Informática

Presenta:
Julio César Vargas Mejía

Asesor:
Mtro. Carlos Francisco Méndez Cruz

México, D.F. **2013**

AGRADECIMIENTOS

A mis padres quienes han hecho un esfuerzo infinito por sacarnos adelante a mis hermanos y a mí. Difícilmente podría describir aquí todo lo que han hecho por nosotros y lo agradecido que estoy con ellos.

A mi mamá porque siempre me ha apoyado, motivado, cuidado y ayudado en todo lo que ha podido. Sin duda sólo ella puede tener esa dedicación, amor y cariño que me llevó a terminar la licenciatura y poder superarme en todos los aspectos de mi vida. Sin su esfuerzo y ejemplo no sería nada de lo que soy ahora, gracias mamá. A mi papá quien siempre ha trabajado tanto como ha podido para permitirnos a mis hermanos y a mí finalizar nuestra carrera. Con su ejemplo tengo la motivación para esforzarme en todo lo que me proponga, gracias por tu dedicación, por cuidarnos y protegernos siempre.

A mi hermano Isaac quien no sólo ha sido la motivación para estudiar una carrera orientada a la tecnología sino también ha sido mí modelo a seguir para querer superarme y desear ser un hombre como el. Su ejemplo me motiva a salir adelante, gracias Isaac. A mi hermano Aldo quien ha compartido conmigo conocimientos, experiencias y, muchos buenos y malos momentos desde la infancia. Sin su presencia en mi vida todo hubiera sido muy difícil, gracias Aldo. Nunca encontraré las palabras para agradecer a ambos toda la ayuda que me han brindado a lo largo de mi vida.

A mis abuelos, tías, tíos, primas y primos porque siempre han estado al pendiente de mi. Su apoyo, cariño y ejemplo han sido de gran importancia en mi vida y espero algún día poder recompensárselos, gracias a todos ustedes.

A mi novia quien ha estado a mi lado desde los primeros semestres de la licenciatura animándome, apoyándome, procurándome y queriéndome en todo momento.

Al maestro Carlos Méndez, amigo y asesor de quien aprendí y compartí conocimientos que permitieron el desarrollo de esta tesis. Estoy muy agradecido con el no solo por darme la oportunidad de ser su becario y alumno sino porque fue quien me motivó a querer desempeñarme en el área de bases de datos, gracias Carlos. A mis amigos y profesores con quienes aprendí y compartí conocimientos, experiencias y excelentes momentos en la licenciatura, gracias a todos.

También gracias a la Universidad Nacional Autónoma de México, institución donde tuvo lugar mi formación académica, desde el bachillerato en el Colegio de Ciencias y Humanidades plantel Azcapotzalco hasta mis estudios superiores en la Facultad de Contaduría y Administración. Me siento orgulloso de ser egresado de esta universidad y estoy muy agradecido por todos conocimientos obtenidos en ella.

Finalmente al Instituto de Ingeniería y al Grupo de Ingeniería Lingüística por permitirme desarrollar esta tesis. Gracias a ello pude ampliar mis conocimientos y obtener otros nuevos. Por último al Consejo Nacional de Ciencia y Tecnología (CONACYT), a la Dirección General de Asuntos del Personal Académico (DGAPA) y al Programa de Apoyo a Proyectos de Investigación e Innovación Tecnológica (PAPIIT) por las becas otorgadas durante el desarrollo de esta tesis.

ÍNDICE

1. INTRODUCCIÓN 1

1.1. Antecedentes 1
1.2. Planteamiento del problema 3
1.3. Objetivos 5
1.3.1. Generales 5
1.3.2. Específicos 6
1.4. Hipótesis 6
1.5. Alcance de la investigación 7
1.6. Metodología de investigación 7

2. CORPUS HISTÓRICO DEL ESPAÑOL EN MÉXICO (CHEM) 9

2.1. Introducción 9
2.2. Arquitectura del CHEM 11
2.3. El generador de unigramas del CHEM (indexador de documentos) 12
2.4. El generador de concordancias 14
2.5. Herramientas del CHEM 15
2.5.1. Búsquedas 15
2.5.2. Estadísticas de asociación de palabras 17
2.6. Otros corpus del GIL 19

3. BASES DE DATOS 20

3.1. Introducción 20
3.2. Definición de base de datos 20
3.3. Sistema manejador de bases de datos 22
3.4. Modelo entidad relación 24
3.5. Objetos de bases de datos 24
3.6. El lenguaje de consulta estructurada SQL 26
3.6.1. Consultas de SQL 27

4. ADMINISTRACIÓN DE RENDIMIENTO 31

4.1. Rendimiento del sistema operativo 37
4.1.1. Consumo de memoria 40
4.1.2. Log de transacciones de la base de datos 42
4.1.3. Bloqueos y contención 44
4.1.4. Otros aspectos que deben ser considerados 46
4.2. Rendimiento de bases de datos 47
4.2.1. Técnicas de optimización 48
4.2.1.1. Particionamiento 49
4.2.1.2. Sistema de archivos o dispositivos en crudo 50
4.2.1.3. Indexación 50
4.2.1.4. Desnormalización 51
4.2.1.5. Clustering 52
4.2.1.6. Intercalado de datos 53
4.2.1.7. Espacio vacío 53

4.2.1.8. Compresión 54
4.2.1.9. Ubicación de archivos 54
4.2.1.10. Tamaño de paginado 55
4.2.1.11. Reorganización 56

5. COMPARACIÓN DE MANEJADORES 59

5.1. Investigación preliminar sobre los RDBMS 59
5.1.1. MySQL 59
5.1.2. PostgreSQL 61
5.1.3. Oracle 62
5.2. Estrategia de evaluación para la comparación de los RDBMS 63
5.2.1. Variables a controlar 64
5.2.2. Análisis de la estructura actual del CHEM 66
5.2.3. Adecuación de la base de datos y carga de datos 67
5.2.4. Obtención de medidas 82
5.2.5. Velocidad actual del sistema 85
5.3. Pruebas en MySQL 87
5.4. Pruebas en Oracle 90
5.5. Análisis de resultados 93

6. ADMINISTRACIÓN DE RENDIMIENTO EN LA BASE DE DATOS DEL CHEM 99

6.1. Ubicación de archivos de los RDBMS 99
6.2. Particionamiento 104

7. CONCLUSIONES 114

8. REFERENCIAS 122

8.1. Bibliográficas 122
8.2. Web 122

ÍNDICE DE FIGURAS

FIGURA 1-1 RESULTADO DE LA PALABRA DE PETICIÓN "COMO" DE 1500 A 1600. 3
FIGURA 2-1 EJEMPLO DE DOCUMENTO ETIQUETADO CON XML 11
FIGURA 2-2 IMAGEN REPRESENTATIVA DEL FUNCIONAMIENTO DEL GENERADOR DE UNIGRAMAS. 13
FIGURA 2-3 EJEMPLO DE LA DISTRIBUCIÓN DE PALABRAS Y DOCUMENTOS EN LA BASE DE DATOS DEL CHEM 13
FIGURA 2-4 CONCORDANCIAS DE LA PALABRA EN INGLES *DEAL* (TOMADA DE BIBER Y CONRAD, 1998: P. 27) 14
FIGURA 2-5 EJEMPLO DE LA SALIDA DEL GENERADOR DE CONCORDANCIAS 15
FIGURA 2-6 VARIANTES ORTOGRÁFICAS DE LA PALABRA ASÍ: ASY, ASSI, ASÍ, ASI 16
FIGURA 2-7 RESULTADO DE LA BÚSQUEDA DE LA PALABRA "INDIO" 16
FIGURA 2-8 VARIANTES MORFOLÓGICAS DE LA PALABRA HERVIR: HERVIRÁN, HERVIRÁ 17
FIGURA 2-9 PALABRAS ASOCIADAS ANTES DE LA PALABRA PIMIENTA 17
FIGURA 2-10 PALABRAS ASOCIADAS DESPUÉS DE LA PALABRA PIMIENTA 18
FIGURA 2-11 PALABRAS MÁS FRECUENTES QUE SE ENCUENTRAN CON LA PALABRA PIMIENTA 18
FIGURA 3-1 EJEMPLO DE PROYECCIÓN 28
FIGURA 3-2 EJEMPLO DE SELECCIÓN 29
FIGURA 3-3 EJEMPLO DE JUNTA DE DOS TABLAS 29
FIGURA 4-1 EJEMPLO DE CONFIGURACIÓN DEL LOG DE TRANSACCIONES DE POSTGRESQL. 44
FIGURA 5-1 REPRESENTACIÓN DE PALABRAS, DOCUMENTOS Y PALABRAS POR DOCUMENTO 66
FIGURA 5-2 EJEMPLO DE LA CREACIÓN DE UNA TABLA CON TIPO DE DATO BOOLEANO 68
FIGURA 5-3 EJEMPLO DE LA ESTRUCTURA LÓGICA DE UN ÍNDICE *BTREE* (BASADA EN FIGURA 3-1 DE HTTP://DOCS.ORACLE.COM/CD/E11882_01/SERVER.112/E25789/INDEXIOT.HTM) 70
FIGURA 5-4 EJEMPLO DEL CAMBIO DE LA CODIFICACIÓN DE CARACTERES EN MYSQL. 80

ÍNDICE DE TABLAS

TABLA 3-1 RUBROS Y SENTENCIAS DEL LENGUAJE SQL..... 27
TABLA 4-1 EJEMPLO DE BLOQUEO SIN SALIDA. 46
TABLA 5-1 PALABRAS DE PRUEBA PARA CADA PERIODO DE 100 AÑOS..... 84
TABLA 5-2 PALABRAS DE PRUEBA EN EL SERVIDOR. 85
TABLA 5-3 PROMEDIOS DE POSTGRESQL PARA GENERAR LAS CONCORDANCIAS. 86
TABLA 5-4 PROMEDIO DE BÚSQUEDA DE POSTGRESQL EN EL SERVIDOR. 87
TABLA 5-5 PROMEDIOS DE MYSQL EN LAS BÚSQUEDAS DE LA INTERFAZ DEL CHEM. 88
TABLA 5-6 COMPARACIÓN DE TIEMPOS DE RESPUESTA DE POSTGRESQL Y MYSQL EN LA INTERFAZ DEL CHEM. 89
TABLA 5-7 COMPARACIÓN DE TIEMPOS DE RESPUESTA DE POSTGRESQL Y MYSQL EN EL SERVIDOR. 89
TABLA 5-8 PROMEDIOS DE ORACLE EN LAS BÚSQUEDAS DE LA INTERFAZ DEL CHEM. 91
TABLA 5-9 COMPARACIÓN DE TIEMPOS DE RESPUESTA DE POSTGRESQL Y ORACLE EN LA INTERFAZ DEL CHEM. 92
TABLA 5-10 COMPARACIÓN DE TIEMPOS DE RESPUESTA DE POSTGRESQL Y ORACLE EN EL SERVIDOR. 93
TABLA 5-11 TIEMPOS PROMEDIO DE LOS TRES RDBMS EN LA INTERFAZ GRÁFICA DEL CHEM. 94
TABLA 5-12 PROMEDIOS DE LOS TRES RDBMS EN EL SERVIDOR. 98
TABLA 6-1 TIEMPO PROMEDIO DE LOS RDBMS SIN CAMBIAR DE UBICACIÓN LOS ÍNDICES. 100
TABLA 6-2 TIEMPO PROMEDIO DE LOS RDBMS DESPUÉS DE CAMBIAR LOS ÍNDICES DE UBICACIÓN. 102
TABLA 6-3 TIEMPOS PROMEDIO DE CADA RDBMS ANTES Y DESPUÉS DE CAMBIAR DE UBICACIÓN LOS ÍNDICES. 103
TABLA 6-4 TIEMPOS PROMEDIO DE LOS RDBMS DESPUÉS DE CREAR NUEVOS ÍNDICES. 103
TABLA 6-5 DISTRIBUCIÓN DE LOS DATOS DEL CHEM. 111
TABLA 6-6 TIEMPOS PROMEDIO DE LOS RDBMS DE LA CONSULTA EN TODO EL RANGO DE AÑOS. 112
TABLA 6-7 TIEMPOS PROMEDIO DE LOS RDBMS DE 1951 A 1975. 112

ÍNDICE DE GRÁFICAS

GRÁFICA 5-1 TIEMPOS PROMEDIO DE 1500 A 1600 DE LOS TRES RDBMS EN LA INTERFAZ GRÁFICA DEL CHEM. 95
GRÁFICA 5-2 TIEMPOS PROMEDIO DE 1601 A 1700. 96
GRÁFICA 5-3 TIEMPOS PROMEDIO DE 1701 A 1800. 97
GRÁFICA 5-4 TIEMPOS PROMEDIO DE 1801 A 1900 97
GRÁFICA 5-5 TIEMPOS PROMEDIO DE LOS TRES RDBMS EN EL SERVIDOR. 98

1. Introducción

1.1.Antecedentes

A lo largo de los años la Universidad Nacional Autónoma de México (UNAM) ha dedicado recursos y esfuerzos por promover la investigación a partir de un conjunto de institutos y comunidades académicas que se dedican a ello. Dentro de los diferentes institutos de investigación que existen en la UNAM se encuentra el Instituto de Ingeniería (IIUNAM). Este instituto es "el centro de investigación en diversas áreas de la ingeniería más productivo del país"[1].

La misión del IIUNAM es "contribuir al desarrollo del país y al bienestar de la sociedad a través de la investigación en ingeniería, la formación de recursos humanos y la vinculación con la sociedad." como se cita en su sitio web[2]. Está conformado por diferentes áreas de investigación, evidentemente en ingeniería, donde podemos ubicar al Grupo de Ingeniería Lingüística (GIL).

El GIL se dedica a la investigación en ingeniería lingüística. "Ésta área del conocimiento humano tiene como finalidad el desarrollo de sistemas informáticos que puedan reconocer, comprender, interpretar y generar lenguaje humano[3]". Además, el GIL está dedicado a la lingüística computacional y al procesamiento de lenguaje natural.

El GIL ha realizado investigación en corpus lingüísticos electrónicos desde hace tiempo. Esta labor lo ha llevado a contar hoy en día con varios corpus disponibles en línea. Más adelante definiré qué es un corpus lingüístico electrónico (véase sección 2.1). Por el momento, comentaré que uno de estos corpus en línea es el Corpus Histórico del Español en México (CHEM) [4]. Éste es un corpus diacrónico del español que contiene textos desde

[1] Obtenido de http://www.iingen.unam.mx/es-mx/SobreNosotros/Paginas/default.aspx, visitada el 9 de septiembre
[2] http://www.iingen.unam.mx/es-mx/SobreNosotros/Paginas/MisionYValores.aspx
[3] Tomado de http://www.iling.unam.mx/index.php?ID_HIST_INFORMACION=5
[4] Para ver éste corpus en línea visitar: http://www.corpus.unam.mx/chem/

el siglo XVI hasta el siglo XXI y puede consultarse a través de una interfaz en internet disponible para investigadores del lenguaje.

El CHEM tuvo una versión prototipo que fue liberada en el año 2005 con algunas características que permitían búsquedas y manejo de archivos de texto plano. En el 2010 se liberó la primera versión del CHEM en la que se incluyeron mecanismos de búsqueda más elaborados, y los documentos del corpus fueron etiquetados con XML. Este sistema en línea está instalado actualmente en un servidor Linux, distribución OpenSUSE, y ha sido desarrollado con tecnología de programación orientada a objetos en Java y bases de datos relacionales.

La principal herramienta disponible en la interfaz de consulta del CHEM permite, a grandes rasgos, buscar palabras en los textos y recuperar un conjunto de palabras antes y después. Adicionalmente es posible visualizar los documentos completos que se encuentran almacenados en él sistema. La arquitectura del CHEM y sus mecanismos de consulta serán explicados detalladamente en el capítulo 2.

A pesar de los avances y mejoras que se dieron entre el prototipo y la primera versión del CHEM, algunos de los requerimientos para mejorar este proyecto siguen vigentes, en especial en lo que tiene que ver con la velocidad de las consultas. Es aquí donde surge la importancia y razón de ser de este trabajo de investigación.

A finales del 2011, se presentó la oportunidad de desarrollar esta investigación en donde se pretende medir el rendimiento de diferentes manejadores de bases de datos con la arquitectura actual del CHEM. Esto se realizará con base en diversas consultas en un mismo ambiente. Además se buscarán mejoras, a nivel de la base de datos, en términos del tiempo de consulta a los datos del CHEM.

Entonces, es así como se da inicio al desarrollo de esta tesis en la que se comprobará qué manejador responde más rápido y eficientemente a las consultas del CHEM, en términos cuantitativos. Por otro lado, se hará una optimización de las consultas que usa el generador de concordancias esperando reducir el tiempo de respuesta del manejador de

bases de datos. Dicho lo anterior, esta tesis lleva por nombre: “Optimización del generador de concordancias del Corpus Histórico del Español en México (CHEM) mediante una comparación entre manejadores de bases de datos relacionales y administración de desempeño de bases de datos”. En la siguiente sección expongo los problemas que esta tesis tratará de resolver.

1.2.Planteamiento del problema

El mecanismo que realiza las búsquedas en la interfaz de consulta del CHEM se llama generador de concordancias. Este mecanismo tiene como función hacer una extracción de palabras alrededor de la palabra de petición del usuario. Este número de palabras puede ser definido desde la misma interfaz de consulta, por ejemplo, diez palabras antes y diez palabras después. Las peticiones pueden ser hechas por palabra normalizada, palabra ortográfica, por lema o una combinación de estas, según decida el usuario. Estos tipos de consultas serán explicados a detalle en la sección 2.5.1. En la Figura 1-1 puede verse un ejemplo de la salida de este mecanismo a partir de la palabra de petición “como”.

Figura 1-1 Resultado de la palabra de petición “como” de 1500 a 1600.

Hoy en día las consultas en el CHEM funcionan de forma adecuada, aunque dependiendo de la búsqueda los tiempos de respuesta pueden variar, siendo algunos muy amplios. Por ejemplo, la búsqueda de la conjunción "Y", la cual es la palabra más frecuente en el periodo de 1500 a 1600, tarda en promedio 00:00:00.648 segundos. Esto es un poco más de medio segundo, tiempo que es considerable si pensamos que el CHEM tiene un número relativamente reducido de palabras (un cuarto de millón), en comparación con otros corpus del mismo tipo, los cuales llegan a tener cientos de millones de ellas[5].

Lo anterior ha provocado una preocupación en los desarrolladores del sistema por mejorar el mecanismo de consultas, pensando en que el CHEM puede llegar a contar también con millones de palabras en un futuro. Es factible pensar que a medida que el CHEM crezca este tiempo se incrementará, imaginemos cuánto se tardará cuando llegue a 10 millones de palabras, por dar un ejemplo.

A manera de comparación Mark Davies reporta que para el Corpus del Español, un corpus de poco más de 100 millones de palabras, su sistema de consulta puede recuperar las palabras de uno a dos segundos. Si bien este corpus y el CHEM no son del todo comparables, ya que sus arquitecturas de cómputo son distintas, esto puede dar idea de las necesidades de optimización del CHEM.

Para darnos otra idea, una consulta en el Corpus Diacrónico del Español (CORDE) de la Real Academia Española de la palabra "donde" se genera para 10,390 documentos en menos de 10 segundos. Este corpus cuenta con alrededor de 125 millones de palabras.

De hecho, el GIL ha puesto en línea recientemente un corpus de 2 millones de palabras, el Corpus del Español Mexicano Contemporáneo (CEMC) de El Colegio de México A. C[6]. Además, próximamente se pondrá en línea otro corpus que cuenta con cerca de 4 millones de palabras (véase sección 2.6). Por lo tanto es esencial la búsqueda de opciones

[5] Por ejemplo, el Corpus del Español (http://www.corpusdelespanol.org/) tiene 101,311,682 palabras en 13,926 textos, y el Corpus Diacrónico del Español (CORDE) de la Real Academia Española (http://corpus.rae.es/ayuda_c.htm#_Toc30228217) tiene alrededor de 125 millones de palabras.
[6] Este corpus puede consultarse en http://www.corpus.unam.mx/cemc/.

de optimización para la arquitectura de corpus del GIL. De esta manera, esta tesis no sólo aportará beneficios al CHEM, sino a diversos corpus.

1.3.Objetivos

Como es bien sabido, existen diferentes manejadores de bases de datos relacionales y no relacionales. El CHEM está montado en el manejador de bases de datos objeto-relacional PostgreSQL v. 8.4.2. Por tanto, para fines de esta investigación, se probarán diferentes manejadores relacionales con el fin de determinar cuál ofrece mejor velocidad de respuesta en las consultas.

También, una tarea común en bases de datos, en cuanto a mejorar el rendimiento se refiere, es precisamente la optimización de la base de datos, *tuning* por su término en inglés. De esta manera se pretenderá aplicar dicha técnica para mejorar el rendimiento de la base de datos del CHEM.

Para que esta tesis tenga un rumbo, es importante tener bien definido lo que se pretende lograr. Por ello, a continuación se hace mención de los objetivos de esta tesis.

1.3.1. Generales

Desarrollar una investigación práctica que aporte conocimientos, pruebas, experiencia y sobre todo una alternativa de solución, que ayude al grupo de desarrolladores del CHEM a optimizar su mecanismo de consultas. Esto implica determinar qué manejador de base de datos realiza las consultas del CHEM en el menor tiempo posible.

Además, se tratará de demostrar que las técnicas de optimización de bases de datos ayudarán a mejorar el rendimiento de la base de datos del CHEM.

Al final, este trabajo intenta dar información teórica y práctica al GIL, en específico a los desarrolladores del CHEM, para que puedan tomar la decisión de implementar o no la solución que se presente aquí dependiendo de los resultados obtenidos.

Para lograr estos objetivos serán necesarios diversos objetivos específicos que menciono a continuación.

1.3.2. Específicos

Los objetivos específicos que propongo son:

1 Seleccionar bajo criterios específicos un conjunto de consultas que permitan realizar las pruebas de rendimiento.

2 Comparar la velocidad de respuesta de los manejadores PostgreSQL, MySQL y Oracle en las mismas condiciones: sistema operativo, procesador, versión del CHEM, cantidad de información en la base de datos y hora de realización de las pruebas; todo ello con las mismas consultas.

3 Seleccionar las técnicas de optimización de bases de datos que resulten más factibles de implementar en la base de datos del CHEM.

4 Realizar pruebas en cada manejador para determinar si la optimización de la base de datos mejora la velocidad de respuesta de las consultas.

1.4.Hipótesis

Con relación a lo anterior surgen algunas preguntas que tendrán que ser resueltas, por ejemplo, ¿qué técnicas de *tuning* existen en bases de datos para mejorar el rendimiento? ¿Es posible mejorar el rendimiento de la base de datos del CHEM usando *tuning* de base de datos? ¿Será posible que haya un manejador de bases de datos que se adapte mejor a las demandas del CHEM en cuestión de velocidad? Éstas son algunas cuestiones que el presente trabajo de investigación pretende resolver y que se expresan en las siguientes hipótesis.

- Existe un manejador de bases de datos que responde más rápido a las consultas del CHEM.
- MySQL es el manejador de bases de datos que responde más rápido a las consultas del CHEM.
- Existe una técnica de optimización de bases de datos que mejore la velocidad de respuesta de las consultas del CHEM.

1.5.Alcance de la investigación

Anteriormente, mencioné lo que se espera lograr con esta tesis, por lo que ahora me dispongo a presentar un listado en el cual detallaré cuáles serán los parámetros a tomar en cuenta para esta investigación.

- Solo se hará una investigación con manejadores de bases de datos relacionales. Estoy consciente de que existen otros tipos de manejadores de bases de datos como por ejemplo los orientados a objetos, y también sé que sería interesante hacer una comparación de éstos con manejadores relacionales. Sin embargo, debido a que la arquitectura actual del CHEM se encuentra actualmente en un manejador relacional sólo se probaran los manejadores relacionales que he mencionado anteriormente. Las comparaciones con manejadores de bases de datos orientadas a objetos, puede que se desarrollen en un futuro en el Instituto de Ingeniería.
- Debido a que la arquitectura actual del CHEM está montada sobre un servidor Linux se utilizarán los manejadores de bases de datos que puedan ser instalados en dicha plataforma.
- Un punto crucial de esta tesis es que sólo se trabajará sobre la parte de la arquitectura del CHEM que genera las concordancias. Es decir, no se tomaran en cuenta todos los otros componentes del CHEM.
- No se buscarán distintas opciones de indexado de documentos electrónicos, las pruebas se realizarán con el indexado actual del CHEM.
- Es importante recalcar que esta tesis no llegará hasta la implementación de la optimización en los corpus en producción debido a que no es el objetivo primordial, sino simplemente se dará la información necesaria para saber cómo aplicar la solución. El GIL tiene pensado reunir el resultado de varias tesis para decidir cuál será la solución a implementar.

1.6.Metodología de investigación

A continuación se presenta la metodología que se seguirá para la realización de esta investigación.

1. Determinar manejadores de bases de datos con los que se pretende hacer la comparación y optimización.
2. Investigación documental sobre los manejadores seleccionados.
3. Decidir la estrategia de evaluación de rendimiento del manejador de bases de datos actual.
4. Decidir el entorno de las pruebas y los factores a controlar (servidor, SO, horario, consulta, etc.).
5. Analizar la estructura actual de la base de datos del CHEM.
6. Recuperar y analizar la velocidad de respuesta del manejador de bases de datos actual.
7. Adecuar la base de datos del CHEM a los diferentes manejadores de bases de datos mediante la modificación del script para creación de objetos de bases de datos.
8. Crear la base de datos del CHEM en los diferentes manejadores, mediante la ejecución del script.
9. Realizar pruebas de rendimiento en los distintos manejadores de bases de datos mediante la ejecución de diferentes consultas.
10. Analizar resultados.
11. Elaborar conclusiones.

Por otro lado, para la realización de la administración de rendimiento se hará lo siguiente:

1. Realizar investigación documental acerca de administración de rendimiento en bases de datos en general.
2. Realizar investigación documental sobre administración de rendimiento de los manejadores de bases de datos seleccionados.
3. Analizar las consultas actuales del CHEM.
4. Emplear la administración de rendimiento en la base de datos del CHEM.
5. Realizar pruebas.
6. Analizar resultados.

2. Corpus Histórico del Español en México (CHEM)

2.1.Introducción

Como había mencionado, el Corpus Histórico del Español en México (CHEM) fue desarrollado en la UNAM por el Grupo de Ingeniería Lingüística (GIL), pero después de tanto mencionarlo cabe la pregunta de ¿qué es un corpus? Un corpus es una "recopilación de un conjunto de textos de materiales escritos y/o hablados, agrupados bajo un conjunto de criterios mínimos para realizar ciertos análisis lingüísticos"[7].

La definición anterior no describe lo que es un corpus electrónico, aunque su definición no varía mucho. Podemos describir un corpus electrónico como un "conjunto de textos elegidos y anotados con ciertas normas y criterios para el análisis lingüístico, de forma que se sirve de la tecnología y las herramientas computacionales para generar resultados más exactos"[8].

Actualmente, existen diversos corpus electrónicos en internet como lo son el Corpus del Español[9], el Corpus de Referencia del Español Actual de la Real Academia Española[10] y el Corpus objeto de estudio de esta tesis: el CHEM[11].

Ahora bien, el CHEM lo podemos definir como un corpus diacrónico, ya que comprende diferentes estados de lengua. Hoy en día ofrece la posibilidad de consultar documentos del español de México desde el siglo XVI al XXI, así como la posibilidad de interactuar con herramientas de exploración de dichos documentos. Los documentos del CHEM pueden ser diferenciados por diversos géneros textuales, pero además por diferentes áreas temáticas, diferentes lugares, tipos de hablantes, etcétera.

Uno de los objetivos del CHEM es permitir a los investigadores del lenguaje observar la evolución que ha tenido el idioma español en nuestro país. Esto se puede realizar a través

[7] Tomado del material de la materia lingüística de corpus que se encuentra en: http://www.iling.unam.mx/CursoCorpus/1_1_Definicion.html, visitada el 14/11/2011.
[8] Tomado del material de la materia lingüística de corpus que se encuentra en: http://www.iling.unam.mx/CursoCorpus/1_1_Definicion.html, visitada el 14/11/2011.
[9] Disponible en http://www.corpusdelespanol.org/
[10] Disponible en http://corpus.rae.es/creanet.html
[11] Disponible http://www.corpus.unam.mx/chem/

de diferentes formas de búsqueda en los documentos del CHEM, pero esto lo veremos más adelante a detalle.

Luego, ya que he planteado un panorama general de lo que es el CHEM, cabe preguntar ¿qué es lo que hace posible la exploración de documentos del corpus? Una de las características del CHEM es que los documentos que son digitalizados mantienen las características textuales originales de los documentos físicos (manuscritos). Para que sea posible la asociación de la ortografía antigua con la actual, se utilizó tecnología XML de etiquetado de documentos. XML es un lenguaje de marcado o etiquetado. Este lenguaje ha sido utilizado para marcar cada una de las palabras de los documentos del CHEM logrando así tener documentos homogéneos digitalizados disponibles para su exploración.

Los documentos que se encuentran en el CHEM están divididos en dos partes: el encabezado y el cuerpo del documento. El encabezado de los documentos contiene información general como puede ser el título del documento, lugar de procedencia, fecha de impresión, género literario, zona geográfica, área temática, etc. Por otro lado, el cuerpo del documento es como tal el contenido en sí pero con las palabras ya etiquetadas con XML. En la Figura 2-1 se puede ver un fragmento de un documento del CHEM.

```
<documento      xmlns="http://www.ii.unam.mx"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://www.ii.unam.mx h/esquema.xsd">

    <encabezado id="1831000001" permisos="todos">

        <titulo>Recetario, 2a edición, circa 1831</titulo>

        <parametros     generoLiterario="prosa" registro="estándar">
            <corpus><CHEM   zonaGeografica="Altiplano Central" areaTematica="cocina"/></corpus>
        </parametros>

        <hablante id="0"/>

        <referencia id="recetario1831">
            <fechaOriginal>1831</fechaOriginal>
        </referencia>

        <responsables>
            <transcriptor nombres="T" apellidos="Careaga" fecha="noviembre 2007"/>
            <etiquetador nombres="A" apellidos="Medina" fecha="agosto 2010"/>
            <revisor nombres="" apellidos="" fecha="noviembre 2007"/>
        </responsables>

    </encabezado>

<cuerpo tipoFuente="Times New Roman">
<seccion>
<tituloSecc alineamiento="centro">
<g n="advertencia" o="advertencia" c="NPFS" l="advertencia" f="adbertensja">ADVERTENCIA</g>
</tituloSecc>
<g n="conocida" o="conocida" c="NPFS" l="conocido" f="konosida">Conocida</g><e/>
<g c="DA3FS0" l="la1" f="la">la</g><e/>
<g c="NCFS" l="utilidad" f="utilidad">utilidad</g><e/>
<g c="SPS000" l="de2" f="de">de</g><e/>
<g c="DD3FS0" l="estar" f="esta">esta</g><e/>
<g c="NCFS" l="obra" f="obrita">obrita</g><e/>
<g c="CC" l="y2" f="i">y</g><e/>
<g c="DA3FS0" l="la1" f="la">la</g><e/>
<g c="NCFS" l="aceptación" f="aseptasjon">aceptacion</g><e/>
<g c="PR3FS00" l="que1" f="ke">que</g><e/>
<g c="VAIP3S0" l="haber1" f="a">ha</g><e/>
```

Figura 2-1 Ejemplo de documento etiquetado con XML

2.2.Arquitectura del CHEM

La funcionalidad del CHEM, al igual que la de algunos otros corpus electrónicos y, en general, al igual que muchos sistemas, es a través de una interfaz en internet que funciona con el apoyo de bases de datos relacionales. El acceso a esta interfaz está restringido por tipos de usuarios los cuales son: usuarios registrados, usuarios de la UNAM y usuarios anónimos. La diferencia entre estos tipos de usuarios está en qué documentos y qué

herramientas de exploración pueden utilizar. Dependiendo de ciertos rangos de privilegios, estos usuarios pueden tener acceso o no a las funcionalidades del sistema.

La arquitectura del CHEM (Medina y Méndez, 2006) incluye varios componentes que describo en los siguientes apartados.

2.3.El generador de unigramas del CHEM (indexador de documentos)

Un aspecto fundamental en el CHEM es el mantener la naturaleza original de los documentos del corpus, es decir, mantener el contenido del documento tal y como éste haya sido creado. Para lograr este fin los documentos del CHEM fueron procesados y etiquetados con XML. Para agilizar su exploración es utilizado un indexador de documentos.

Para que el indexador pueda trabajar con los documentos, éstos deben ser previamente etiquetados con XML. Cabe mencionar que los documentos se indexan una sola vez. Una vez etiquetados, el primer paso del indexador es extraer el encabezado del documento y almacenar esta información en una tabla de documentos de la base de datos. Después, extrae los unigramas (palabras) con su posición en bytes (denominado offset) y acumula la frecuencia total de la palabra (el número de veces que se repite la palabra). Estos datos (palabras, posición y frecuencia) se almacenan en una tabla de unigramas dentro de la base de datos. En la Figura 2-2 se muestra el funcionamiento del generador de unigramas.

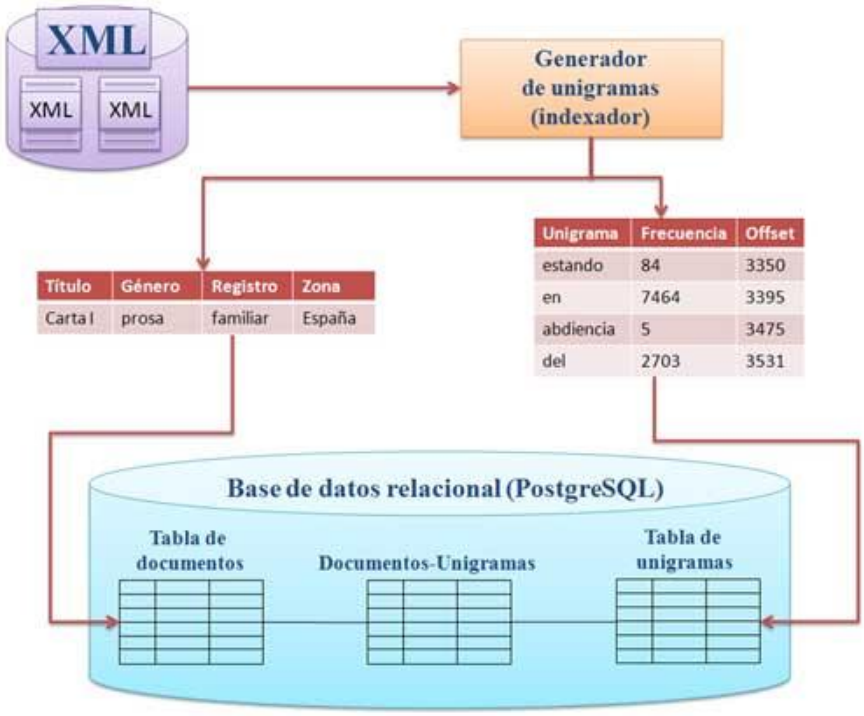


Figura 2-2 Imagen representativa del funcionamiento del generador de unigramas[12].

Luego entonces, a partir de la tabla de documentos y unigramas se crea una tabla transitiva para determinar qué palabras se encuentran en que documentos. La siguiente figura muestra gráficamente estas tablas.

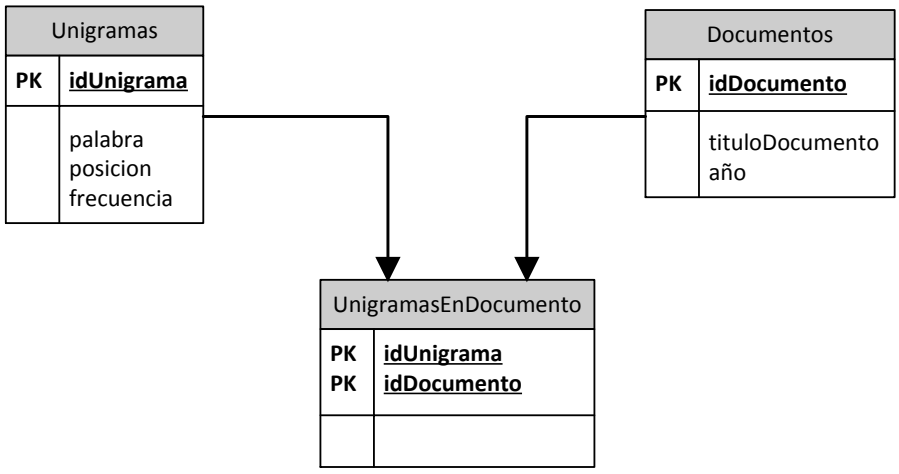


Figura 2-3 Ejemplo de la distribución de palabras y documentos en la base de datos del CHEM

[12] Tomada de Medina y Méndez, 2006.

2.4.El generador de concordancias

Antes de comenzar a describir el generador de concordancias es importante entender que es, en sí, una concordancia. Una concordancia es una ocurrencia de una palabra seleccionada, con el contexto que la rodea. Es común que cada ocurrencia de la palabra seleccionada se presente en una línea, con la palabra en medio y su contexto en cada lado. Además, se puede definir un conjunto de concordancias como una lista de las ocurrencias de elementos en un documento, organizada en un orden determinado por el usuario, donde cada ocurrencia está rodeada por una porción apropiada de su contexto original (Oakes, 1998: p.150). La Figura 2-4 muestra un ejemplo de concordancias de la palabra en ingles *deal*.

and secret plans prepared to	deal	with the mass sit-down	1
of companies and put one property	deal	through each. Mr.	2
. In particular, a good	deal	of concern has been	3
hangs a tale – and a great	deal	of money. Neville	4
where his new measures to	deal	with Britain's	5
just a matter of working a good	deal	harder before we really	6
. "I'm mixed up in a	deal	involving millions	7

Figura 2-4 Concordancias de la palabra en ingles *deal* (tomada de Biber y Conrad, 1998: p. 27)

Ahora bien, el generador de concordancias del CHEM recibe una palabra, la cual es ingresada por el usuario, y a partir de ella hace una petición a la base de datos para encontrar la palabra. La base de datos hace la consulta y regresa la lista de documentos donde se encontró la palabra junto con las posiciones en donde aparece la palabra de petición. Ya con el resultado que regresa la base de datos, el generador de concordancias localiza la palabra y extrae las concordancias en formato XML, mostrando un determinado número de palabras antes y después de la palabra buscada. Finalmente se muestra el resultado de la petición al usuario en la pantalla del CHEM, es decir, las concordancias de la palabra buscada (ver Figura 2-5).

Figura 2-5 Ejemplo de la salida del generador de concordancias

2.5.Herramientas del CHEM

2.5.1. Búsquedas

El CHEM actualmente cuenta tres opciones de búsqueda: por ortografía normalizada, por coincidencia ortográfica exacta y por lema. En un futuro, se implementaran más tipos de búsquedas.

La búsqueda por ortografía normalizada busca una palabra en sus distintas variantes ortográficas. Por ejemplo, la Figura 2-6 muestra las variantes de la palabra “así”.

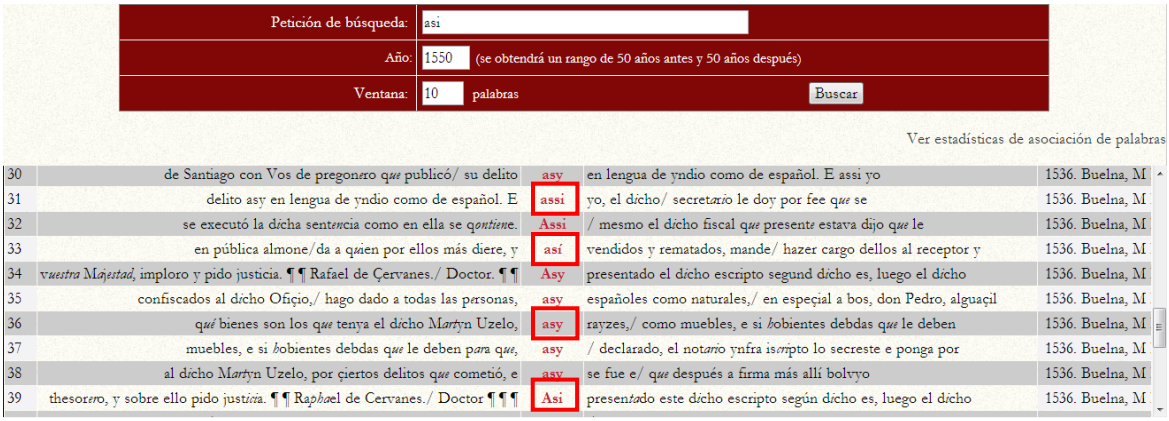


Figura 2-6 Variantes ortográficas de la palabra así: asy, assi, así, asi

La búsqueda ortográfica exacta permite recuperar concordancias de la palabra tal y como la escribió el usuario. La Figura 2-7 genera una búsqueda de la palabra “indio”. Nótese que para generar una búsqueda ortográfica exacta la palabra esta debe estar entrecomillada.

Figura 2-7 Resultado de la búsqueda de la palabra “indio”

Finalmente, la búsqueda por lema (conjunto de variantes morfológicas) recupera las concordancias de las variantes de la palabra que se está buscando. A continuación, en la Figura 2-8 se muestra el resultado de la búsqueda da la palabra [hervir]. Nótese que la búsqueda por lema se genera con la palabra entre corchetes.

Figura 2-8 Variantes morfológicas de la palabra hervir: hervirán, hervirá

2.5.2. Estadísticas de asociación de palabras

Este es un componente que trabaja de la mano con el generador de concordancias. Cada vez que se generan concordancias se obtiene la frecuencia de aparición de la palabra de búsqueda con las palabras que la acompañan antes y después. A partir de ello se genera una tabla de contingencia para obtener cuatro medidas estadísticas de asociación de palabras. Finalmente, se crea una tabla que muestra la palabra de búsqueda y las palabras más asociadas a ella en los documentos del CHEM. Los siguientes ejemplos muestran gráficamente la asociación de palabras antes, después y en todas las concordancias de la palabra “pimienta”.


Estadísticas de palabras inmediatamente antes de **pimienta** en el rango de años de 1800 a 1900

Palabras	I	-2logλ	χ^2	Y	Promedio normalizado
y	3.13	486.9007	2189.2599	0.7106	0.8071
bastantita	6.4593	47.337	N/S	0.9556	0.5243

N/S = No significativo

Figura 2-9 Palabras asociadas antes de la palabra pimienta

Estadísticas de palabras inmediatamente después de **pimienta** en el rango de años de 1800 a 1900

Palabras	I	-2logλ	χ^2	Y	Promedio normalizado
gorda	6.3342	114.6084	5621.1398	0.9482	0.8346
molida	5.7197	137.8872	4244.3533	0.9116	0.7625
y	2.7578	275.7194	997.704	0.639	0.5633
despolvoreada	6.1716	43.889	N/S	0.9378	0.5118
molidos	5.0265	57.7871	1054.0887	0.8631	0.5099

N/S = No significativo

Figura 2-10 Palabras asociadas después de la palabra pimienta

Estadísticas de palabras dentro de toda la concordancia en el rango de años de 1800 a 1900

Palabras	I	-2logλ	χ^2	Y	Promedio normalizado
clavo	6.6391	1741.2797	102327.0802	0.976	0.9918
canela	6.4563	1205.7742	62931.1132	0.963	0.8087
sal	5.4876	1436.6815	35098.3107	0.9229	0.7283
con	4.5941	1498.2302	18058.3261	0.8875	0.6539
un	4.8103	1300.8701	18818.9313	0.8885	0.6355
molidos	6.3765	314.5078	15837.1845	0.9522	0.56
nuez	6.2068	323.5169	14346.9064	0.9424	0.5489
poco	5.0834	718.4222	13260.0752	0.8846	0.5473
en	4.2286	1048.8554	9875.7906	0.8454	0.5453
cominos	6.0265	341.7008	13210.016	0.9323	0.5396
moscada	6.2587	270.8688	12507.8713	0.9449	0.5394
azafran	6.1325	273.4028	11478.3546	0.9376	0.5308
agengibre	6.4593	142.3336	7648.0275	0.9562	0.5193
clavos	6.2323	190.1678	8627.5696	0.9427	0.5168
gorda	6.4295	129.3668	6803.8014	0.9542	0.5138
molida	6.0251	201.9515	7831.0902	0.9307	0.5059
le	4.6928	632.6678	8711.0062	0.8566	0.5024
bastantita	6.6824	63.3259	N/S	0.9717	0.5014
tabasco	6.6824	63.3259	N/S	0.9717	0.5014

N/S = No significativo

Figura 2-11 Palabras más frecuentes que se encuentran con la palabra pimienta

2.6.Otros corpus del GIL

El Grupo de Ingeniería Lingüística también ha desarrollado otros corpus electrónicos aparte del CHEM. Estos se lista a continuación:

- Corpus Lingüístico en Ingeniería (CLI).
- Corpus de las Sexualidades en México (CSMX).
- Corpus del Español Mexicano Contemporáneo (CEMC).
- Corpus Básico Científico del Español de México (COCIEM)
- Corpus de Contextos Definitorios (CORCODE).
- Corpus anotados con relaciones discursivas (RST Spanish Treebank).
- Corpus del Español Colonial Mexicano (COREECOM).

De estos corpus el CSMX, el CEMC y el COCIEM están basados computacionalmente en la arquitectura del CHEM. De ellos el CEMC cuenta con cerca de dos millones de palabras y el COCIEM cuenta con más de cuatro millones de palabras. Debido a la cantidad de información que manejan estos corpus será importante implementar lo que aquí se presente siempre y cuando sea positivo.

3. Bases de datos

3.1.Introducción

En este capítulo describiré algunos de los conceptos más importantes que deben ser mencionados para entender el propósito de esta investigación. Cabe mencionar que hablar de bases de datos implica saber sobre muchos aspectos, tanto teóricos como técnicos, pero para fines de esta tesis sólo se dará una vista general de los temas y se hará énfasis en aquellos que sean significativos en esta investigación.

3.2.Definición de base de datos

Antes de comenzar a explicar lo que es una base de datos y un sistema manejador de bases de datos, es importante resaltar la importancia de los datos y la información ya que son la fuente principal de alimentación de las bases de datos. De acuerdo con Peter Rob, (2001, p.5) datos son hechos (o acontecimientos) los cuales, como tal, no han sido procesados para revelar un significado especial. Por otro lado, el autor define la información como el resultado de procesar los datos.

Los datos deben ser procesados o trabajados para que puedan interpretarse como información ya que si sólo observamos los datos difícilmente podremos tener una amplia visión de lo que representan. Para llegar a tener información verídica, oportuna y objetiva, los datos deben tener un contexto, de esta forma podremos procesar los datos para obtener información que nos ayude como fundamento para la toma de decisiones o para crear reportes estadísticos.

Los datos son el fundamento y fuente de la información y a su vez la información es la base del conocimiento. El conocimiento puede ser definido como el cuerpo de la información y hechos acerca de un tema específico (Rob, 2001). Para que la información sea útil, ésta requiere que los datos sean precisos, es decir, deben ser generados correctamente y su formato debe ser de fácil acceso para su procesamiento.

Para tener un control eficiente de datos, generalmente se requiere del uso de sistemas de bases de datos. Hoy en día, los sistemas de bases de datos son vitales para el funcionamiento de instituciones y organizaciones modernas.

Ya que se ha descrito la relevancia de los datos y la información como elementos principales de las bases de datos, tenemos que existen diversas definiciones de lo que es una base de datos, aunque no hay mucha diferencia entre las que tenemos a continuación:

"Colección de datos interrelacionados almacenados en conjunto sin redundancias prejudiciales o innecesarias; su finalidad es servir a una aplicación o más, de la mejor manera posible; los datos se almacenan de modo que resulten independientes de los programas que los usan; se emplean métodos bien determinados para incluir nuevos datos y para modificar o extraer datos almacenados" (Martin, 1975, citado en De Miguel, Piattini y Marcos, 1999: p. 26).

"Conjunto de datos de la empresa memorizado en un ordenador, que es utilizado por numerosas personas y cuya organización está regida por un modelo de datos"[13] (Flory, 1982, citado en De Miguel, Piattini y Marcos, 2001: p. 26).

"Colección integrada y generalizada de datos estructurada, atendiendo a las relaciones naturales de modo que suministre todos los caminos de acceso necesarios a cada unidad de datos con el objeto de poder atender todas las necesidades de los diferentes usuarios" (Deen, 1988, citado en De Miguel, Piattini y Marcos, 2001: p. 26).

"Colección de datos relacionados" (Elmasri y Navathe, 2007, p. 4).

Para mi gusto, podemos definir una base de datos como lo hace Rob (2001: p. 6): una estructura computacional integrada y compartida que almacena datos de usuario final. Los datos que se almacenan pueden ser hechos, los cuales son de interés para algún usuario, y metadatos o datos acerca de datos. Estos últimos proveen una descripción de las características de los datos y un conjunto de relaciones que los ligan con la base de

[13] Un modelo de datos es "un conjunto de conceptos, reglas y convenciones bien definidos que nos permiten aplicar una serie de abstracciones a fin de describir y manipular los datos de un cierto mundo real que deseamos almacenar en la base de datos" (De Miguel y Piattini, 2000: p. 23).

datos. En otras palabras, los metadatos presentan una visión más amplia de los datos en la base de datos.

Las definiciones descritas anteriormente mantienen la esencia de lo que es una base de datos, aunque en algún punto puedan verse un tanto variadas. Para enfatizar y profundizar más en lo que son las bases de datos, a continuación menciono algunas de sus características.

- **Persistente.** Esta característica se refiere a que el almacenamiento de la base de datos y por consecuencia de los datos, debe ser estable, por ejemplo en un disco magnético o una cinta magnética. Es común que estos medios de almacenamiento se vayan depurando conforme pasa el tiempo, dependiendo de las necesidades de mantener o no un histórico de la información.
- **Compartir.** Significa que la base de datos puede tener diversos usuarios y funciones. Una base de datos permite tener varias funciones o usos a la vez ejecutados por diferentes usuarios; a este hecho se le conoce como concurrencia.
- **Interrelación.** Quiere decir que los datos que residen en la base de datos se pueden conectar entre sí para mostrar una visión más detallada. Hablando técnicamente, la base de datos cuenta con entidades y relaciones entre éstas. Esta característica se detallará más adelante en este mismo capítulo.

3.3.Sistema manejador de bases de datos

Anteriormente, definí las bases de datos y mencioné algunas de las características más importantes. En este apartado describiré lo que es un sistema manejador de bases de datos (DBMS por sus siglas en inglés – *Data Base Management System*) y las ventajas que ofrece ante la demanda de almacenar grandes cantidades de información.

Un DBMS es un conjunto de programas computacionales los cuales se encargan de ayudar a controlar las bases de datos y el acceso a los datos almacenados en ella. Como dice Peter Rob (2001, p. 6), en cierto sentido, una base de datos se asemeja a un

archivador electrónico bien organizado en el cual, con la ayuda de software, conocido como DBMS, se maneja mejor el contenido del archivador. Otra definición puede ser que un DBMS "es un conjunto de componentes que soportan la creación, el uso y el mantenimiento de bases de datos" (Catherine, 2009: p. 6).

Además, como es crucial mantener datos claros y concisos para tener información precisa, es importante tener el método adecuado de manejo de los datos y para lograrlo es común que sea con el apoyo de un DBMS ya que este puede hacer esta tarea de forma eficiente y efectiva. Es por eso que a continuación hago mención de algunas otras ventajas que ofrece el uso de un DBMS (Rob, 2001: p. 7 y 8):

- Mejora la manera de compartir los datos. El DBMS crea un ambiente en el cual los usuarios tienen acceso a más datos y los datos están mejor controlados.
- Mejora la seguridad de los datos. Entre más usuarios demanden acceso a los datos, mayor es el riesgo de violaciones a la seguridad de los datos. Un DBMS provee un esquema de seguridad para la mejor aplicación de privacidad de datos y políticas de seguridad.
- Mejora la integración de datos. Un acceso amplio a datos mejor controlados provee una vista más integrada de las operaciones de la organización.
- Minimiza inconsistencia de datos. La inconsistencia de datos existe cuando diferentes versiones de los mismos datos aparecen en diferentes lugares. El DBMS permite tener concentrados los datos para que los usuarios puedan tener acceso de diversos lugares logrando que todos vean una sola versión de la información.
- Mejora el acceso a los datos. El DBMS puede obtener respuestas rápidas a diversas cónsulas de múltiples usuarios.
- Mejora la toma de decisiones. Datos bien controlados con un acceso inmediato a ellos hace posible generar mejor calidad de información, y por consiguiente mejor toma de decisiones.

Los DBMS se pueden clasificar como relacionales y no relacionales. Esta investigación está desarrollada sobre bases de datos relacionales. Entre los DBMS más conocidos

tenemos Informix, DB2, Sybase, Adabas, Oracle, Microsoft SQL server, PostgreSQL, MySQL, SQLite, NexusDB entre muchos otros. Todos y cada uno tienen mecanismos únicos que los diferencian uno del otro. Para fines de esta investigación se decidió trabajar solo con tres manejadores: PostgreSQL, MySQL y Oracle.

3.4.Modelo entidad relación

Como mencionaba anteriormente en las características de bases de datos, las unidades almacenadas en la base de datos pueden estar interrelacionadas. Para entender la forma en que pueden estar relacionadas es necesario plantear de manera conceptual el modo en que los datos deben comportarse para poder obtener información. Esto se logra a través de un modelo de datos.

Entre los modelos de datos existentes podemos mencionar el modelo entidad/relación (E/R) el cual fue propuesto por Peter P. Chen. De acuerdo con él, "El modelo entidad/relación puede ser usado como una base para una vista unificada de los datos con el enfoque más natural del mundo real que consiste en entidades e interrelaciones" (Chen, 1976, citado en De Miguel, Piattini y Marcos, 2001, p. 47).

El modelo E/R permite hacer un planteamiento de la estructura de la base de datos donde los diseñadores, mediante una abstracción de la realidad, proyectan la estructura final de la base de datos donde serán almacenados los datos. El modelo E/R está apoyado de dos conceptos: entidad y relación. Según Peter Chen (Chen, 1976, citado en De Miguel, Piattini y Marcos, 2001: p. 48) una entidad es "una cosa que se puede identificar claramente" y una relación "una vinculación entre entidades".

3.5. Objetos de bases de datos

Las bases de datos relacionales tienen como objetivo almacenar la información a través de tablas y relacionar éstas mismas con el fin de transformar datos en información confiable. En un nivel lógico, la base de datos almacena un conjunto de objetos los cuales son: tablas, vistas, índices, funciones, procedimientos almacenados, *triggers*, secuencias, entre otros. A continuación expondré en qué consisten cada uno de ellos.

Las tablas son objetos de dos dimensiones formados por filas y columnas. Éstas almacenan datos y puede que estén relacionadas con otras tablas. Las columnas de las tablas están definidas por tipos de datos. Estos tipos de datos pueden ser numéricos[14], de caracteres[15], fechas[16] y otros más. Las tablas y las relaciones entre ellas pueden ser ajustadas, controladas y definidas por restricciones de integridad. Estas restricciones son de llave primaria, llave foránea, valores no nulos, valores únicos y valores restringidos (CHECK).

Las vistas son una proyección de lo que almacenan las tablas. Una vista está formada por una o todas las columnas de una o muchas tablas. Éstas tienen muchos propósitos y uno de ellos es mostrar información de forma más general. Otro de los propósitos es ocultar información a los usuarios ya que en ocasiones es necesario que se excluya información que puede ser confidencial.

Los índices son objetos de bases de datos los cuales ordenan los datos de las columnas de las tablas. Éstos tienen como función mejorar el acceso a los datos. Se pueden crear índices por cada campo de una tabla o por varios de ellos.

Los procedimientos almacenados y funciones pueden ser definidos como un conjunto de programas que existen dentro de la base de datos. La base de datos tiene la capacidad de administrar y ejecutar la lógica de los procedimientos y funciones a través de los comandos con los que cuenta. La diferencia entre un procedimiento y una función radica en que el primero no regresa valor. Lo anterior tiene impacto en la sintaxis, pero ésta será explicada en la sección 5.2.3 de esta tesis.

Una de las razones por las que surgieron los procedimientos y funciones fue para mover código de las aplicaciones, que se desarrollan en algún lenguaje de programación de propósito general, al servidor de base de datos. De esta manera se evita que los clientes que usan las bases de datos relacionales tengan que ejecutar varios comandos de

[14] Tales como *INTEGER* o *NUMERIC*.
[15] Por ejemplo: *CHAR* y *VARCHAR*.
[16] Como *DATE* o *DATETIME*.

SQL desde la aplicación para lograr algún objetivo. Otra de las características de los procedimientos es que se crean como objetos independientes en la base de datos, es decir, no están asociados a ninguna tabla o vista que resida en la base de datos.

El uso de procedimientos almacenados ofrece ventajas en algunas áreas como lo son desarrollo, integridad, seguridad, rendimiento y consumo de memoria (Oracle, 2005a). También tienen la ventaja de que pueden consultar o modificar datos de diversas tablas o vistas.

Los *triggers* son programas que residen en la base de datos y que se ejecutan dependiendo de alguna acción en la misma. Por ejemplo, puede que un *trigger* se ejecute y cumpla alguna tarea si se hace una inserción o una actualización en la base de datos. Un uso muy práctico que tienen es almacenar la información de los usuarios que ejecutan instrucciones en la base de datos. Estas tareas generalmente son para auditar el uso de las mismas.

Finalmente las secuencias son objetos que tienen como función incrementar valores numéricos de forma secuencial. La creación de estos valores empieza, termina y se incrementa con valores asignados por el usuario. Se usan generalmente para asignar valores de llave primaria ya que ésta tiene como restricción el tener un valor único y no nulo.

3.6. El lenguaje de consulta estructurada SQL

El lenguaje de consulta estructurada (SQL por sus siglas en inglés - *Structured Query Language*) tiene su historia dentro de la muy conocida organización IBM quienes desarrollaron este lenguaje como parte de uno de sus proyectos.

En sus inicios este lenguaje tuvo diferentes variantes ya que distintos fabricantes lo empleaban para su beneficio hasta que un grupo de organizaciones lograron convertirlo en un estándar. Estas organizaciones fueron el Instituto Nacional de Estándares de América (ANSI por sus siglas en inglés – *American National Standards Institute*), La Organización Internacional de Estándares (ISO por sus siglas en inglés – *International*

Organization for Standardization) y la Comisión Electrónica Internacional (IEC por sus siglas en inglés – *International Electrotechnical Commission*).

El primer estándar publicado para el lenguaje SQL fue el SQL-86, aunque tuvo que ser mejorado para así publicar SQL-92 años más tarde. Posteriormente salieron dos publicaciones más (SQL: 1999 y SQL: 2003) con mejoras con respecto a la versión anterior.

SQL fue creado con el propósito de tener un lenguaje para la definición, manipulación y control de bases de datos. En la siguiente tabla se muestra un resumen de las sentencias más importantes de SQL (basada en Catherine, 2009: p.81).

Tabla 3-1 Rubros y sentencias del lenguaje SQL

Tipo de Sentencia	***Sentencia***	***Propósito***
Definición de bases de datos. (DDL por sus siglas en inglés – *Data Definition Language*)	CREATE DATABASE, CREATE SCHEMA, CREATE TABLE, CREATE VIEW	Definir una nueva base de datos, tabla y vista.
	ALTER TABLE	Modificar la definición de una tabla.
Manipulación de bases de datos. (DML por sus siglas en inglés – *Data Manipulation Language*)	SELECT	Extraer los contenidos de las tablas.
	UPDATE, DELETE, INSERT	Modificar, eliminar y agregar filas.
Control de bases de datos. (DCL por sus siglas en inglés – *Data Control Language*)	COMMIT, ROLLBACK	Terminar y deshacer transacciones.
	GRANT, REVOKE	Agregar y eliminar derechos de acceso.
	CREATE TRIGGER	Definir una regla de bases de datos.

Uno de los usos de SQL es de forma individual donde el usuario puede ejecutar un conjunto de sentencias mediante un editor especial (cliente) el cual está incluido en el conjunto de programas que provee el DBMS. Otro de los usos de SQL es mediante un programa o aplicación el cual envía instrucciones de forma automatizada al DBMS y este envía los resultados (en caso de que existan) de regreso al programa para procesarlos.

3.6.1.Consultas de SQL

Como podemos ver en la tabla anterior (Tabla 3-1), el lenguaje SQL está dividido en diferentes lenguajes que ayudan al control y administración de las bases de datos

relacionales. Para fines de esta investigación, es importante explicar el funcionamiento y comportamiento de la sentencia SELECT.

La sentencia SELECT se usa para obtener información de una o más tablas o vistas, las cuales residen en la base de datos. Con esta instrucción, podemos obtener información de las siguientes maneras.

- Proyección. Con la sentencia SELECT podemos obtener un conjunto de columnas de una tabla. Es posible obtener tantas columnas como se requieran o sea necesario obtener. En la Figura 3-1 puede verse un ejemplo de proyección donde se obtienen las columnas C1, C4 y C6.

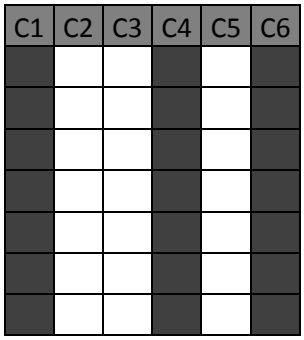


Figura 3-1 Ejemplo de proyección

- Selección. Podemos obtener un conjunto de filas de una tabla. Se pueden aplicar diferentes criterios para obtener un conjunto de filas. En la
- Figura **3-2** puede observarse un ejemplo de selección de la segunda, quinta y sexta filas de la tabla.

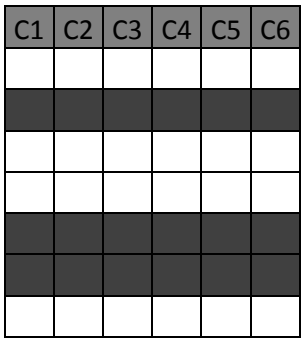


Figura 3-2 Ejemplo de selección

- Junta o *join*. Nos permite extraer datos almacenados en diferentes tablas. Para lograr esto es necesario especificar una llave entre ellas. En la Figura 3-3 puede verse la junta de la Tabla 1 y la Tabla 2 en donde se obtiene una columna de cada una. De la Tabla 1 la columna C6 y de la Tabla 2 la columna C2. El resultado es una nueva tabla formada de estas dos columnas.

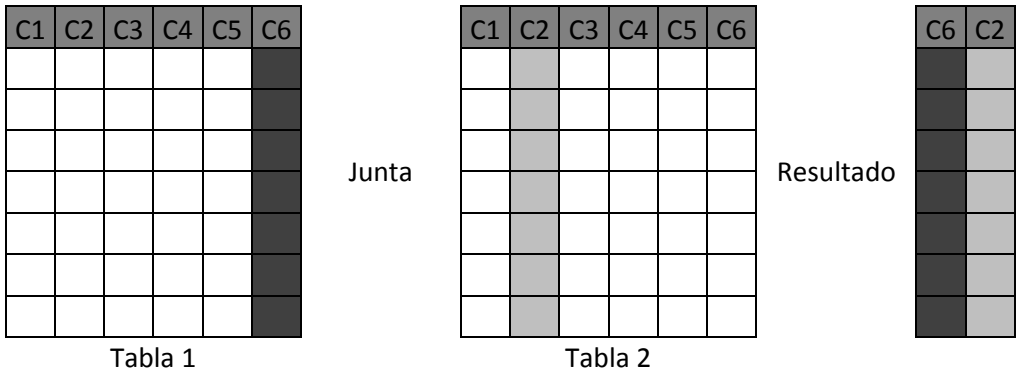


Figura 3-3 Ejemplo de junta de dos tablas

La sintaxis básica de la sentencia SELECT es la siguiente:

```
SELECT campo1, campo2, campo3, campo4
FROM Tabla1;
```

Dónde SELECT identifica las columnas que se desean mostrar u obtener y FROM indica la tabla que contiene las columnas deseadas. Cabe mencionar que la instrucción anterior selecciona columnas específicas; sin embargo, se pueden obtener todas las columnas de la siguiente manera:

```
SELECT *
FROM Tabla1;
```

La instrucción anterior hace una proyección de todas las columnas de la Tabla1. La sentencia SELECT tiene muchos usos y maneras de usarse, y aunque es muy interesante cómo se hace, estos detalles no se mencionarán en este trabajo. De cualquier manera existen muchos libros y documentación disponible en línea donde se puede obtener información detallada. En la siguiente liga se puede ver el uso de la sentencia SELECT en Oracle:

http://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_10002.htm.

4. Administración de rendimiento

Una de las tareas más importantes de un administrador de bases de datos es la administración del rendimiento[17] y el monitoreo de bases de datos. Estas tareas son importantes hoy en día debido a que la demanda de acceso a la información ha crecido y está presente en muchas de las actividades tanto laborales como cotidianas.

La administración de rendimiento y el monitoreo no son actividades únicas aplicables al área de bases de datos. Existe una gran cantidad de campos laborales donde es posible aplicar estas tareas como por ejemplo: el área de administración de servidores, telecomunicaciones y desarrollo de sistemas. Hasta en la vida cotidiana es posible utilizarlas cuando nuestra computadora tiene problemas con el acceso a páginas de internet a través de un navegador o cuando al querer ejecutar un programa observamos retrasos en el tiempo de respuesta.

Para poder pensar en rendimiento de bases de datos es necesario pensar en eficiencia. Como es bien sabido, la información que se almacena en las bases de datos es requerida por diversos usuarios durante todo el día. Los usuarios necesitan información y el DBMS provee la información requerida. La velocidad, y en general la eficiencia a la que el DBMS satisface la demanda de información de los usuarios, es un claro ejemplo de rendimiento de bases de datos.

El rendimiento de bases de datos en muchas ocasiones depende de los siguientes factores: carga de trabajo, rendimiento, recursos, optimización y contención. A continuación explico cada uno de ellos (Mullins, 2002).

La carga de trabajo es una combinación de transacciones en línea, programas automáticos (*jobs*), las consultas que se ejecutan cotidianamente en el DBMS, análisis de *data warehouse*, comandos de sistema, ente otros, los cuales son enviados desde un cliente. Algunas veces cuando las actividades cotidianas tienden a tener un solo comportamiento o una rutina, la carga de trabajo puede ser predecible pero otras veces no.

[17] También llamada administración del desempeño.

El rendimiento se refiere a la capacidad que tiene una computadora para procesar datos. A su vez, puede estar compuesto de la velocidad de datos de entrada y salida, velocidad del CPU, capacidades de paralelismo en el servidor, y la eficiencia del sistema operativo y de los programas que se utilicen.

Los recursos son tanto el hardware y software con los que cuenta el equipo de cómputo. Ejemplos de estos recursos son el DBMS y los programas instalados, los dispositivos de almacenamiento, memoria RAM, controladores de caché, etcétera.

La optimización es uno de los puntos más importantes. Todos los sistemas pueden ser optimizados en algún momento. En las bases de datos relacionales se pueden optimizar las consultas que se ejecuten en el DBMS así como los parámetros que tiene la base de datos para establecer accesos y controles más eficientes.

Finalmente la contención[18] es el resultado de la demanda de un recurso en particular. Es un estado donde dos o más clientes, quienes forman parte de la carga de trabajo, intentan usar un mismo recurso de diferentes maneras, por ejemplo, un cliente intenta borrar información mientras otro cliente está actualizando la misma información. Entre más contención exista más bajo es el rendimiento.

De esta forma, se puede mejorar el rendimiento de bases de datos mediante la optimización de los recursos usados y la minimización de la contención, permitiendo procesar la mayor carga de trabajo posible (Mullins, 2002).

Una vez que se definieron e identificaron los aspectos importantes que tienen impacto en el rendimiento de una base de datos, es importante analizar la conveniencia de tener un plan para el rendimiento. La planeación del rendimiento de bases de datos es uno de los puntos más relevantes en la implementación de aplicaciones. Es por ello que el administrador de bases de datos necesita prevenir el mal desempeño de la base de datos en todas las aplicaciones que se creen en un ambiente de producción. Un plan simple,

[18] La concurrencia es una de las principales causas de la contención.

pero completo, que permita monitorear el rendimiento de las aplicaciones, puede ayudar a optimizar la base de datos y las sentencias SQL que se utilicen.

En muchos casos, los problemas de rendimiento de aplicaciones de bases de datos se deben al mal uso del lenguaje SQL o a la mala programación de las mismas aplicaciones. Esto no implica que desde un inicio esté mal la implementación de SQL en las aplicaciones sino que éstas pueden tener un bajo desempeño con el paso del tiempo. Esto tiende a pasar por el crecimiento de las bases de datos, nuevos accesos a éstas, incremento de usuarios, cambios en las reglas del negocio, etcétera.

Algunos de los problemas por los que el rendimiento de bases de datos a nivel de consultas puede ser bajo son los siguientes (Mullins, 2002):

- Escaneo total de tablas.
- Falta de índices necesarios o mala elección de creación de índices.
- No usar los índices disponibles en la base de datos.
- Estadísticas anticuadas de base de datos.
- Junta de tablas en orden poco óptimo.
- Junta de tablas en el código de la aplicación en lugar de código SQL.
- Junta de tablas inapropiado o innecesario.
- Código SQL eficiente dentro de código de aplicación ineficiente.
- Elaboración ineficiente de subconsultas.
- Ordenamientos y agrupamientos innecesarios.

Una tarea difícil de hacer es encontrar las consultas que tienen grandes costos o consumo de recursos cuando se habla de cientos de éstas. Lo anterior se complica aún más cuando una consulta se ejecuta en un ambiente donde corren una gran cantidad de programas. El rendimiento de un ambiente en producción puede ser afectado si hay más de un cliente que ejecute consultas que requieran muchos recursos.

Existen otros factores que provocan un bajo rendimiento en las bases de datos por lo que es recomendable hacer un análisis periódico del rendimiento de la base de datos y el

servidor donde se encuentra ésta. Posterior al análisis, para asegurar un óptimo rendimiento de la base de datos, es ideal llevar a cabo un plan que abarque rendimiento de bases de datos y rendimiento del servidor.

Algunos factores que pueden ser revisados en el plan de rendimiento son los siguientes:

- Asignación de memoria.
- Opciones de logs (log del caché, segmentos de rollback de las bases de datos).
- Eficiencia de E/S (separación de tablas e índices en disco, tamaño de la base de datos, archivos fragmentados y extendidos).
- Carga de trabajo de la base de datos y de las aplicaciones.

Pero no es suficiente con tener un plan de rendimiento que nos ayude algunas veces a optimizar nuestro ambiente. Manejar el rendimiento difiere de monitorear el rendimiento porque el primero combina el monitoreo junto con el plan para resolver los problemas cuando sea necesario.

La administración del rendimiento consiste en tres componentes que deben ser aplicados en conjunto y de forma consecutiva. Estos son: monitorear, analizar y corregir (Mullins, 2002). A continuación defino los componentes anteriores, que deben ser contemplados en el plan de rendimiento.

El monitoreo consiste en una revisión del sistema en donde se analiza cómo funciona el sistema completo (sistema operativo, aplicaciones, bases de datos, etcétera), permitiendo identificar problemas o componentes que pueden ser mejorados.

El monitoreo permite la recolección de información para mejorar el rendimiento del sistema, pero antes de intentar optimizar o cambiar cualquier cosa, se debe hacer un análisis. Analizar, segundo componente, nos ayuda a tomar las mejores decisiones para la optimización del rendimiento de nuestro ambiente.

Llevar a cabo las correcciones es el último componente de la administración del rendimiento. Actualmente existen muchas herramientas que permiten administrar el rendimiento del sistema pero muchas veces no hacen una optimización completa de nuestro sistema. Es por eso que la persona encargada de hacer el análisis debe tomar en cuenta cada uno de los aspectos que pueden ser optimizados para hacer las mejorar apropiadas en el sistema.

De cualquier manera, por mucho que se planee la administración de rendimiento siempre van a existir y a ocurrir problemas. En otras palabras, es imposible prevenir todos los problemas que pueden existir en cuanto a rendimiento se refiere porque los sistemas y aplicaciones cambian con el tiempo.

Una buena práctica que nos permite prevenir lo anterior, en la medida de lo posible, es usar la predicción del rendimiento de nuestro sistema. Esta predicción es otra tarea de rendimiento y consiste en anotar y analizar la tendencia del uso de recursos y las estadísticas de rendimiento a lo largo del tiempo. El historial de rendimiento y tendencia de consumo de recursos permite o ayuda a predecir la necesidad de actualizar hardware y/o software. También el historial puede mostrar los periodos cuando el rendimiento de la base de datos es menos de lo normal, causado por el incremento de actividad de los usuarios.

Una aplicación de bases de datos necesita interacción con los recursos del sistema para operar debidamente. Por lo anterior, el mejoramiento de aplicaciones de bases de datos puede dividirse en los siguientes tres componentes: optimización de sistema operativo, de bases de datos, y de aplicaciones.

La optimización del sistema tiene la mayor importancia e impacto ya que tanto bases de datos como aplicaciones residen en él. No importa la cantidad de optimizaciones o mejoras que se hagan en las bases de datos o en las aplicaciones, si el servidor en el que corren tiene pocos recursos o es inapropiado para dar soporte a los anteriormente mencionados.

El DBMS debe ser optimizado para tener un mejor rendimiento. La forma en la que el DBMS está instalado, su consumo de memoria, almacenamiento en disco, otros recursos y su configuración pudieran impactar en el rendimiento de las aplicaciones de bases de datos.

Finalmente, el rendimiento de bases de datos puede ser degradado por el diseño físico de la base de datos, normalización, almacenamiento en disco, número de tablas, diseño de índices y el uso de lenguaje de definición de datos. El almacenamiento físico de los archivos de la base de datos puede tener un impacto en el rendimiento de aplicaciones cuando éstas necesiten acceso al archivo para obtener información. Entre más información sea almacenada en un solo archivo de bases de datos más bajo es el rendimiento de las aplicaciones.

Con el paso del tiempo, los datos almacenados en la base de datos cambiarán. Las operaciones del DML que se generen en la base de datos pueden degradar la eficiencia de ésta. También puede ser que los archivos de datos de la base de datos necesiten expandirse conforme se agrega información o quizás sea necesario agregar nuevos archivos de datos. La creación de archivos nuevos o el crecimiento de éstos degradan el rendimiento de la base de datos.

Los índices también necesitan ser monitoreados, analizados y, de ser posible, optimizados para mejorar el acceso a los datos y asegurar que no tienen un impacto negativo en la base de datos. Por ejemplo, Oracle recomienda lo siguiente en cuanto a índices se refiere[19]:

Crear índices cuando:

- Una columna contiene una gran cantidad de valores.
- Una columna contiene un gran número de valores nulos.

[19] Para más información visitar la siguiente página de documentación oficial de Oracle: http://docs.oracle.com/cd/E11882_01/server.112/e25494/indexes002.htm

- Existen columnas que son usadas con frecuencia en la cláusula *WHERE* o *JOIN* de una consulta.
- La tabla es muy grande y se requiere obtener menos del 15% de registros de la tabla.

No crear un índice cuando:

- Las columnas no son usadas con frecuencia como condición en la cláusula *WHERE*.
- La tabla no contiene muchos registros.
- La tabla se actualiza constantemente.
- Las columnas indexadas forman parte de una expresión.

Por otro lado, para el rendimiento de las aplicaciones, éstas se deben diseñar apropiadamente y estar siendo monitoreadas para saber la eficiencia que tienen. Muchas veces los problemas de rendimiento son causados por mala implementación de código de las aplicaciones. La implementación de código SQL puede ser una tarea difícil. Es necesario que los desarrolladores sepan formular, monitorear y optimizar consultas o bien contar con la ayuda de un administrador de bases de datos para tener orientación de las consultas que vayan a ser implementadas.

4.1.Rendimiento del sistema operativo

Como ya había mencionado, un bajo rendimiento del sistema operativo puede causar que las aplicaciones y las bases de datos tengan un desempeño similar o igual de bajo al del sistema operativo sin importar qué tantas tareas de rendimiento se les apliquen a éstas.

El sistema operativo se conforma de software que se ejecuta en un determinado hardware con características específicas, mismo que necesitan las aplicaciones y bases de datos. El administrador de bases de datos, de ser posible, debe tener los accesos y permisos para hacer cambios en la arquitectura del sistema con el fin de mejorar el rendimiento de la base de datos.

Algunos de los aspectos que deben ser revisados son los siguientes:

- Cantidad de memoria para el sistema operativo.
- Cantidad de memoria swap[20] en caso de que se trate de un sistema operativo Unix/Linux.
- En la mayoría de los casos, se desarrollan diferentes manejadores de bases de datos para correr en distintas versiones de sistemas operativos. Dependiendo del sistema operativo se debe escoger una versión del DBMS para ser instalado.
- Los valores de los parámetros del sistema operativo que utiliza el DBMS.

En cuanto a esta tesis se refiere, estos aspectos fueron considerados en la realización de los experimentos ya que fue importante tomar en cuenta el entorno de hardware y software en el que se realizaron las pruebas. Estos aspectos serán descritos más adelante en la sección 5.2.1 del capítulo 5.

Aunque los aspectos anteriores son importantes, no son los únicos. Existen más componentes que se deben tomar en cuenta a nivel de hardware para que las bases de datos trabajen mejor. Uno de estos componentes es el almacenamiento en disco. Éste es uno de los factores que pueden generar un cuello de botella en el rendimiento de la base de datos ya que genera costo de lectura y escritura (E/S) de datos en los archivos. Lo anterior se produce como resultado de los múltiples accesos a los archivos a través del mecanismo del disco para hacer esta tarea.

Una solución a los accesos simultáneos de datos en disco es el uso de memorias de estado sólido. Éstas cuentan con un mecanismo de acceso que trabaja más rápido que los discos duros comunes. Otra solución es la fragmentación de archivos de datos a lo largo de los discos duros (en caso de que exista más de uno) para que los accesos no se concentren en un solo disco. Esta capacidad también debe ser soportada por el DBMS para poder distribuir los archivos de datos.

[20] La memoria swap la utiliza el sistema operativo en caso de que se agote la memoria RAM.

Es importante tomar en cuenta la arquitectura del DBMS y la interacción que tendrá con el sistema operativo. La arquitectura de los manejadores de bases de datos que ya existen varían unos de otros y cada uno de ellos está construido de diferentes maneras.

Por ejemplo, la arquitectura de PostgreSQL está compuesta de la siguiente manera[21].

- Un proceso maestro llamado "*postmaster*". Este proceso maestro corre todos los procesos dependientes necesarios para que la base de datos funcione.
- Una o varias conexiones con procesos de usuario, como el uso de la consola *psql* para operaciones SQL, *pg_dump* para respaldos o conexiones desde distintas aplicaciones.
- Los procesos *background* que corren en el servidor para que funcionen la base de datos de PostgreSQL. Algunos de estos son los procesos de memoria y procesos de utilerías como el que permite escribir al log de transacciones (*archiver process*) o el proceso que ayuda a limpiar/depurar la base de datos (*autovacuum launcher*).

Por otro lado la arquitectura de Oracle está compuesta de la siguiente forma:

- Archivos de estructura de la base de datos. Estos son los archivos de control de la base de datos, los archivos de redo log y archivos de base de datos que contienen información de ella.
- Estructuras de memoria. Segmentos de memoria que utiliza el software de Oracle: SGA y PGA.
- Procesos. Existen varios de éstos que corren en modo *background* para que el DBMS funcione. Sus características no serán explicadas aquí porque no es relevante para esta investigación[22].

[21] Para más información visitar la documentación oficial de PostgreSQL: http://www.postgresql.org/docs/7.1/static/arch-pg.html y http://www.postgresql.org/docs/9.0/interactive/runtime-config-resource.html.
[22] Sin embargo, en la documentación oficial de Oracle se pueden encontrar los procesos y para qué sirve cada uno de ellos: http://docs.oracle.com/cd/B28359_01/server.111/b28318/process.htm.

- Segmentos de *rollback*. Estos segmentos permiten a los usuarios deshacer las operaciones que se ejecuten en el DBMS.
- Archivos de Redo log. Estos archivos registran los cambios que se hacen en los datos de la base de datos.

Estos son ejemplos de la arquitectura de dos manejadores de bases de datos que fueron tomados en cuenta para esta tesis. Es claro que sus arquitecturas son diferentes y por ello es importante que el administrador de la base de datos tenga entendimiento total de ellas para saber cómo va a influir su comportamiento en el sistema operativo y cómo se puede mejorar el rendimiento en éste.

Por otro lado, los DBMS proveen un conjunto de parámetros que pueden ser configurados para que el administrador decida el comportamiento conveniente en el sistema operativo. En algunos casos, los parámetros de los DBMS pueden ser cambiados dinámicamente. Esto quiere decir que en algunos casos los cambios pueden tomar lugar sin necesidad de reiniciar el servidor de base de datos. En otros casos es necesario que el DBMS se reinicie para que los cambios hagan efecto.

Algunas de las maneras en que pueden ser modificados dichos parámetros son la ejecución de procedimientos almacenados, edición de archivos de configuración del DBMS, o la ejecución de comandos SQL.

En los apartados siguientes expongo algunos aspectos que pueden modificarse con el fin de optimizar un sistema operativo.

4.1.1. Consumo de memoria

Otro de los aspectos importantes a tomar en cuenta en nuestras tareas de optimización del rendimiento del sistema operativo es el consumo de memoria. Todos los programas instalados en la computadora, incluyendo el sistema operativo, utilizan memoria. Los DBMS no son la excepción y de hecho existen tareas de rendimiento que se basan en el uso de la memoria.

Los DBMS utilizan memoria RAM para muchas funciones, pero en especial para el uso de sus propios componentes y para mantener datos en caché permitiendo que su acceso sea más rápido. El propósito de que los datos se encuentren en caché es porque la velocidad de acceso a esos datos es más rápida que si se consultaran o leyeran desde disco. De esta forma entre más memoria RAM sea asignada al DBMS, mejor será el rendimiento de éste[23] (Mullins, 2002), aunque es evidente que debe haber una configuración de por medio para que esta característica funcione.

Al igual que muchos programas, los DBMS suben los datos más demandados a la memoria caché. Por ello, entre más tiempo esté un recurso en memoria caché, mayor es la posibilidad de que las peticiones subsecuentes para ese recurso eviten operaciones de lectura directamente en disco (Mullins, 2002).

Para que los datos sean almacenados en memoria caché es necesario que sean leídos desde disco, es decir, el DBMS procesa una petición de lectura de sus archivos de datos, regresa el resultado de la petición y posteriormente agrega el resultado a la memoria. De esta forma si esos datos tienen una alta demanda de acceso podrán ser leídos desde la memoria caché. Puede que esta característica se encuentre en los manejadores de bases de datos bajo el concepto de *buffer pool*.

Otro de los beneficios de guardar datos en caché es que cada que se ejecuta una orden, el DBMS transforma el código en un procedimiento que él mismo entiende. Esta transformación de código genera un plan de ejecución el cual se almacena en memoria y tiene un beneficio relevante en el rendimiento de las aplicaciones. Como éstas son programadas para cumplir tareas repetitivas, ejecutan las mismas consultas hacia el DBMS y al ser las mismas, los datos se mandan llamar desde la memoria.

Cabe mencionar que la memoria caché no es la única memoria que utilizan los DBMS. Existe otra memoria conocida como *sort cache* o caché de ordenamiento, la cual se encarga de almacenar ordenamientos de datos ya que puede ser que los datos no se

[23] Por ejemplo, para la instalación de Oracle, se pide como requisito tener la misma cantidad de memoria swap que de memoria RAM.

encuentren ordenados a nivel de disco. Los ordenamientos más comunes en los DBMS se producen por las instrucciones de agrupado, ordenamiento y junta de tablas.

Los anteriores son algunos usos que los DBMS le dan a la memoria, aunque no son los únicos. La memoria es esencial, no solo para los DBMS, sino también para todos los programas que corren en un sistema operativo. Otros aspectos que los DBMS controlan mediante el uso de la memoria son los siguientes:

- Conexiones de usuarios. Todas las conexiones de usuario requieren memoria para establecer la comunicación entre el cliente y DBMS.
- Dispositivos. Las bases de datos necesitan comunicación con los dispositivos de la PC porque es en donde almacenan los archivos necesarios que utilizan para su funcionamiento.
- Abrir la base de datos y sus objetos para su uso. La interacción que tienen los usuarios con las bases de datos y sus objetos requiere el uso de memoria para mantener las conexiones entre las bases de datos y objetos con los usuarios o clientes.
- Bloqueos. Estos se generan cuando dos o más usuarios intentar acceder a los mismos datos.

Aunque los usos que los DBMS le dan a la memoria parecen ser muchos, para que el DBMS trabaje correctamente sólo se necesita cierta cantidad de memoria necesaria. La mayoría de los DBMS utilizan memoria para algunas de las características anteriormente descritas, aunque algunas pueden no estar presentes. Para saber la cantidad adecuada de memoria que los manejadores requieren basta con consultar sus manuales de instalación. También es recomendable revisar las características que tienen los DBMS para manipular el rendimiento de la memoria caché a través de los parámetros que tienen disponibles para ello.

4.1.2. Log de transacciones de la base de datos

Cada DBMS cuenta con un log de transacciones el cual puede tener impacto en el rendimiento. Este componente guarda el registro consecutivo de los cambios que se

hacen a los datos de la base de datos en archivos de diferente tamaño según cada manejador. Como resultado del almacenamiento de las operaciones y cambios que se hacen en los datos, este mecanismo permite llevar a cabo instrucciones de *rollback* y de recuperación de bases de datos a un estado en particular de los datos.

El almacenamiento y comportamiento del log de transacciones depende del DBMS. Por un lado, en algunos casos se puede especificar en dónde se almacenarán los archivos de registro del log y por otro lado es posible habilitar o deshabilitar el almacenamiento del log.

Por ejemplo en PostgreSQL, en su archivo de configuración *postgresql.conf* existe un apartado para configurar la escritura del log. Este apartado se denomina como *Write Ahead Log* y cuenta con parámetros para indicar la ruta de almacenamiento de los logs, el comando para almacenarlos, etc. La Figura 4-1 muestra el ejemplo de configuración de log de PostgreSQL.

Es importante resaltar que la generación de archivos del log de transacciones crece dependiendo de la actividad de la base de datos, por lo que es necesario respaldar y depurar el log periódicamente. Es importante hacer lo anterior ya que el log utiliza espacio en disco y memoria para ser escrito, lo que implica el consumo de recursos de la base de datos. Esta configuración dependerá de las necesidades del administrador y del valor de la información de la base de datos.

```
#------------------------------------------------------------------------------
# WRITE AHEAD LOG
#------------------------------------------------------------------------------

# - Settings -

#fsync = on             # turns forced synchronization on or off
#synchronous_commit = on        # immediate fsync at commit
#wal_sync_method = fsync        # the default is the first option
                        # supported by the operating system:
                        #   open_datasync
                        #   fdatasync
                        #   fsync
                        #   fsync_writethrough
                        #   open_sync
#full_page_writes = on          # recover from partial page writes
#wal_buffers = 64kB         # min 32kB
                        # (change requires restart)
#wal_writer_delay = 200ms       # 1-10000 milliseconds

#commit_delay = 0           # range 0-100000, in microseconds
#commit_siblings = 5            # range 1-1000

# - Checkpoints -

#checkpoint_segments = 3        # in logfile segments, min 1, 16MB each
#checkpoint_timeout = 5min      # range 30s-1h
#checkpoint_completion_target = 0.5 # checkpoint target duration, 0.0 - 1.0
#checkpoint_warning = 30s       # 0 disables

# - Archiving -

#archive_mode = off     # allows archiving to be done
                # (change requires restart)
#archive_command = ''       # command to use to archive a logfile segment
#archive_timeout = 0        # force a logfile segment switch after this
                # number of seconds; 0 disables
```

Figura 4-1 Ejemplo de configuración del log de transacciones de PostgreSQL.

Aunque existe una gran cantidad de información acerca del log de transacciones, para propósitos de esta tesis solo se tomó en cuenta como parte de los elementos que deben ser considerados para la administración de rendimiento de bases de datos.

4.1.3. Bloqueos y contención

Otros de los aspectos que pueden tener un impacto en el rendimiento de las bases de datos son los bloqueos y los bloqueos sin salida (*deadlocks*, por su término en inglés). Estos dos procesos han sido implementados en los DBMS como una medida de seguridad para garantizar la consistencia de los datos.

Los bloqueos se presentan cuando un usuario intenta modificar datos que están siendo modificados por otro usuario. Por ejemplo, si el usuario 1 está modificando el campo 1 y 2 del registro número 100 de una tabla y el usuario 2 intenta modificar los mismos dos campos de la misma tabla del mismo registro, el DBMS bloquea la transacción[24] del usuario 2 hasta que el usuario 1 finaliza la suya. Posteriormente, cuando el usuario 1 finaliza su transacción y suelta los registros que el usuario 2 intentaba modificar, el DBMS desbloqueará la transacción del usuario 2 y aplicará los cambios que éste intentaba hacer.

Por otro lado los *deadlocks* se presentan cuando dos usuarios intentan modificar registros que están siendo utilizados por el otro usuario. Para ejemplificar lo descrito anteriormente se presenta la siguiente tabla.

[24] Las transacciones en las bases de datos se refieren a operaciones que pueden alterar la estructura o contenido de los objetos de la base de datos.

Tabla 4-1 Ejemplo de bloqueo sin salida

Transacción 1	***Tiempo***	***Transacción 2***	***Descripción***
Usuario 1 modifica campo 1 del registro número 100.	12:00 P.M.	Usuario 2 modifica campo 2 del registro número 100.	Inician dos transacciones.
Usuario 1 modifica campo 2 del registro número 100.	12:05 P.M.		Se bloque la sesión del usuario 1.
	12:08 P.M.	Usuario 2 intenta modificar el campo 1 del registro número 100.	Se bloque la sesión del usuario 2
En este punto ambas transacciones están bloqueadas. A este evento se le conoce como *deadlock*			
	12:10 P.M.		El DBMS finaliza las transacciones con *rollback* para asegurar la consistencia de los datos. Esta acción depende de cada DBMS. Puede que se finalice solo una de las dos transacciones y la otra no.

Cada vez que se presenta un bloqueo en el manejador de bases de datos se consumen recursos de memoria de éste y el problema incrementa cuando se presenta una cantidad considerable de éstos.

4.1.4. Otros aspectos que deben ser considerados

Existen otros aspectos que pueden tener impacto en el rendimiento de la base de datos y que deben ser considerados.

Uno de estos aspectos es el almacenamiento del diccionario de datos del DBMS. El diccionario de datos almacena información de los objetos de la base de datos. Entre la información que almacena se encuentran los nombres de las tablas, nombres de los campos que se encuentran en dichas tablas, los tipos de dato de los campos, nombres de usuarios, nombres de índices, etcétera.

Se recomienda que el almacenamiento del diccionario de datos se haga en un dispositivo diferente a donde se almacenarán los archivos de datos. De esta forma se podrá controlar de manera independiente a los objetos que sean creados en la base de datos.

Otros aspectos que también pueden ser considerados para el rendimiento de la base de datos son la configuración de *triggers* anidados[25], opciones de seguridad, o configuraciones para bases de datos distribuidas.

La mayoría de los aspectos de rendimiento que se plantearon en esta sección pueden variar entre los diferentes manejadores de bases de datos que existen. En esta tesis se dará explicación de los cambios que pueden ser implementados en los DBMS seleccionados para este trabajo en lo que respecta a optimizar la base de datos del CHEM.

Finalmente, es importante resaltar que cualquier cambio que se intente hacer para mejorar el rendimiento de bases de datos, si no se hace bien puede provocar la degradación del rendimiento de la misma.

4.2.Rendimiento de bases de datos

El rendimiento de bases de datos está enfocado particularmente en los siguientes aspectos:

- Diseño de la base de datos.
- Parámetros del DBMS.
- Construcción física de los objetos de la base de datos, especialmente tablas e índices.
- Archivos de datos donde se almacenan los datos de la base.

El diseño y selección de los aspectos anteriores son esenciales para que la base de datos trabaje adecuadamente. Estos aspectos deben ser monitoreados para poder aplicar los cambios necesarios en caso de que el comportamiento de la base de datos sea

[25] Esta característica permite que un *trigger* mande llamar a otro *trigger*. Algunos DBMS permiten configurar el número de *triggers* que pueden ser anidados. Esto se debe a que si el número de *triggers* anidados es muy alto, puede provocar deficiencias en el rendimiento de la base de datos.

ineficiente. En caso de que los valores que se tomen para estos aspectos no sean los adecuados, no importará la cantidad de mejoras que se intenten hacer en las consultas SQL o las mejoras que se hagan en nuestro sistema, la base de datos no trabajará adecuadamente.

4.2.1. Técnicas de optimización

Los DBMS cuentan con características diferentes unos de otros por lo que es necesario saber cuáles son las que soportan para poder aplicar las técnicas de rendimiento apropiadas en la base de datos. Los siguientes son algunos aspectos que pueden ser optimizados.

- **Particionamiento**. Técnica que permite dividir una tabla en diferentes archivos de datos.
- **Sistema de archivos o dispositivos en crudo**. Opción que nos permite almacenar los datos en discos de sistema operativo o en dispositivos en crudo[26].
- **Indexación**. Selección de índices apropiados.
- **Desnormalización**. Es una variación del diseño lógico de la base de datos la cual nos ayuda a optimizar el rendimiento de las consultas.
- **Clustering**. Forzar el almacenamiento físico de los datos de forma secuencial.
- **Intercalado de datos**. Combinación de datos de múltiples tablas en un solo archivo.
- **Espacio vacío**. Espacio reservado para el crecimiento de los datos.
- **Compresión**. Técnica que permite reducir requerimientos de almacenamiento.
- **Ubicación de archivos**. Poner los archivos de datos en el lugar indicado.
- **Tamaño de paginado**. Usar el tamaño adecuado para el almacenamiento de datos y E/S.
- **Reorganización**. Quitar ineficiencias de la base de datos restructurando objetos de la base de datos.

[26] Por su término en inglés *row devices*, son dispositivos que no tienen ningún formato y que son controlados directamente por el DBMS.

A continuación explicaré cómo se pueden optimizar los aspectos anteriores. Cabe mencionar que para fines de esta tesis no serán optimizados todos los rubros mencionados sino sólo aquellos que sean soportados en los DBMS seleccionados.

4.2.1.1. Particionamiento

En un nivel lógico, una tabla es un objeto de la base de datos que almacena datos, aunque lo anterior sólo es una convención ya que físicamente ambos están almacenados en disco. El hecho de identificar qué tablas existen en qué archivos de datos es una tarea que realiza el DBMS.

En este punto, la distribución de objetos de la base de datos en los archivos de datos se puede hacer de la siguiente manera:

- Una tabla en un archivo.
- Una tabla en múltiples archivos.
- Múltiples tablas en varios archivos.

El método más común que utilizan los DBMS en cuanto al almacenamiento de objetos en archivos de datos se refiere, es el de almacenar una tabla o tablas en un archivo. De esta forma cada vez que se insertan nuevas filas, esta información se va almacenando en un archivo.

Otra de las formas de almacenar objetos de la base de datos es el almacenamiento de tablas en diversos archivos de datos. Esta característica es utilizada cuando las tablas contienen grandes cantidades de datos.

El último de los métodos es el almacenamiento de múltiples tablas en diversos archivos de datos. Este método es útil para almacenar tablas con una pequeña cantidad de datos como el diccionario de datos o catálogos cuyos datos no cambian constantemente.

La característica principal del particionamiento es que se puede implementar paralelismo. El paralelismo es el proceso de acceder a la base de datos de diferentes

formas en paralelo. Para poder acceder a los datos en paralelo se utilizan diferentes CPUs de forma simultánea mediante una consulta SQL. De esta forma es posible reducir el tiempo que tarda una consulta en recuperar los datos que ésta busca.

4.2.1.2. Sistema de archivos o dispositivos en crudo

Es posible almacenar los datos de la base de datos en el sistema de archivos del sistema operativo o en dispositivos en crudo. Un dispositivo en crudo es aquel que no tiene ningún formato y que puede ser utilizado directamente por el DBMS.

Cuando se utiliza el sistema de archivos para el almacenamiento de datos, el sistema operativo utiliza memoria cache para almacenar los datos y él mismo decide cuando enviar los datos a disco. Por otro lado, cuando se utilizan dispositivos en crudo, el DBMS envía directamente los datos a disco a través del cache del DBMS. Esto provoca que la escritura de los datos sea más rápida mejorando así el rendimiento de la base de datos.

Esta característica, aunque presenta ventajas muy interesantes para mejorar el rendimiento de la base de datos, no será posible aplicarla en esta investigación ya que no se cuenta con dispositivos en crudo para hacer pruebas de este tipo.

4.2.1.3. Indexación

El propósito de los índices en la base de datos es para que los accesos a los datos sean más rápidos y eficientes. Como ya había mencionado en la sección 4, existen diversas situaciones en las que es recomendable la creación de índices. Sin la creación de índices, el acceso a los datos tendría que ser mediante un escaneo de todas las filas existentes en las tablas. En este caso el rendimiento de la base de datos es malo si las tablas contienen millones de registros.

Por otro lado, la creación de índices no implica que se tenga que crear una gran cantidad de índices. De hecho, llenar la base de datos de índices degrada el rendimiento de la base de datos. Es por lo anterior que se recomienda hacer un análisis de la base de datos para determinar dónde crear índices. De esta forma no sólo el rendimiento de la base de datos será mejor, sino que el rendimiento las aplicaciones que acceden a la base de datos será también mejor.

La mejora del rendimiento de la base de datos a través de la creación de índices se debe a que el I/O que genera el DBMS es menor al que se genera cuando se hace un escaneo completo de columnas o de la tabla. Sin embargo, el constante cambio de los datos que se indexan puede degradar el rendimiento ya que se genera I/O para actualizar los índices.

4.2.1.4. Desnormalización

La desnormalización es el proceso de agregar un hecho en varios lugares. Aunque parece ir en contra de un diseño correcto de datos, este método ayuda a que el acceso a los datos sea más rápido.

Este método se implementa cuando un diseño normalizado no funciona apropiadamente o como se espera que trabaje. De hecho, la única razón por la que un diseño relacional debe ser desnormalizado es para mejorar el rendimiento de la base de datos.

Algunos de los casos en los que es aceptable implementar desnormalización son los siguientes:

- Cuando está prohibido hacer junta de tablas que generen un alto costo de recursos. Algunas organizaciones prohíben a los desarrolladores hacer consultas con junta de tablas que consuma muchos recursos.
- Cuando se generan reportes críticos y estos tienen un costo alto para generarse.
- Cuando se usan dos tablas en más de un ambiente.
- Para generar relaciones uno a uno o uno a muchos en una sola tabla.

Otras técnicas que pueden ser útiles para desnormalizar un modelo es almacenar datos redundantes en tablas para evitar consultas que necesiten varias juntas. También, almacenar grupos de datos en un registro puede ayudar a disminuir el I/O y el uso de disco. Finalmente, almacenar datos derivables, como operaciones de columnas, reduce el costo de cálculos del DBMS y por lo tanto, del uso de algoritmos con altos costos.

4.2.1.5. Clustering

La principal característica de esta técnica es que permite el almacenamiento físico de las tablas en el disco ordenadas por una o más columnas. El *clustering* de tablas es forzado por el DBMS a través de un índice. Este índice obliga a que las filas de las tablas estén ordenadas por determinadas columnas en orden ascendente. Esta misma característica restringe a que sólo exista un ordenamiento por tabla debido a que los datos sólo se pueden almacenar de una forma físicamente.

La ventaja de aplicar *clustering* en las tablas de la base de datos es que las consultas que requieren un ordenamiento secuencial generan menos I/O de disco, lo que hace que el rendimiento sea mejor.

Es recomendable que las columnas que formaran parte de la definición del *cluster* sean columnas que son usadas frecuentemente en la cláusula *WHERE*. De esta forma el acceso a los datos será más rápido. Por otro lado, no se recomienda que la llave primaria de una tabla forme parte de la definición del *cluster* ya que por definición es única. Sin embargo, si ésta es usada con frecuencia puede que tenga un beneficio en el rendimiento.

Una vez que se ha definido el índice que hará el *clustering* de la tabla, el DBMS puede comportarse de dos formas. Por un lado puede que cuando nuevos registros sean insertados en la tabla, el DBMS maneje el espacio físico para insertar en orden las nuevas filas. Por otro lado es posible que al insertar nuevos registros el espacio en disco sea insuficiente para almacenar los nuevos registros conforme a la secuencia por lo que el DBMS pondrá los nuevos registros en otro espacio.

Es importante revisar las opciones de *clustering* que ofrecen los DBMS en caso de que se presente la situación mencionada anteriormente. Si se llega a presentar, con el paso del tiempo la tabla dejara de estar ordenada y será necesaria una reorganización.

Cuando el DBMS tiene que acomodar registros nuevos, y no hay espacio disponible para almacenar los registros, el DBMS crea un espacio nuevo para almacenar los datos. A esto se le conoce como división de páginas (page *splitting*). Para llevar a cabo esta tarea el

DBMS puede hacer dos procedimientos diferentes. El primero consiste en agregar un espacio vacío en medio del paginado lleno. A continuación recorre la mitad de los registros del área llena al espacio vacío y finalmente ajusta los nuevos registros en su lugar así como indicadores para saber la continuación de los registros. El segundo solo crea un espacio vacío en el espacio lleno donde los registros subsecuentes serán insertados.

Aunque lo anterior es un proceso que permite mantener el orden de los registros, es importante hacer reorganizaciones de los espacios que se utilicen ya que los procesos anteriores pueden generar muchos espacios para los datos, generando así una desorganización de los registros.

4.2.1.6. Intercalado de datos

Otra buena técnica para mejorar el acceso a los datos es almacenar las tablas que más son combinadas en el mismo archivo de datos. De esta forma, el DBMS podrá obtener la información de las tablas más rápido que si estuvieran en archivos de datos diferentes o incluso que si estuvieran almacenados en discos diferentes.

4.2.1.7. Espacio vacío

Dejar espacio vacío en los índices o *tablespaces*[27] sirve para almacenar nuevos datos. Las ventajas de reservar espacio vacío para los datos que se van agregando son evitar reorganizar los datos con frecuencia, reducir la contención, e incrementar la rapidez con la que se insertan los datos.

Sin embargo, esta técnica también tiene algunas desventajas. Una de ellas es que los requerimientos de espacio en disco son mayores, los escaneos en las tablas toman más tiempo, y si existen pocos registros almacenados, serán necesarios más recursos de I/O para dar lectura al espacio de datos.

Esta característica puede ser buena o mala dependiendo de las necesidades de la base de datos. Dependiendo del ambiente y uso de la base de datos, el espacio vacío se debe

[27] Un espacio de tablas, o por su término en inglés *tablespace*, es un espacio lógico en la base de datos que sirve para almacenar tablas, índices, vistas, entre otros objetos de base de datos. Físicamente los *tablespaces* se almacenan en archivos de datos.

medir con respecto a la frecuencia de inserciones y modificaciones que se necesiten, el impacto de tener acceso a datos que no tengan *clustering*, y la probabilidad de que los registros sean cambiantes o migrados.

4.2.1.8. Compresión

La compresión, tal y como suena, ayuda a comprimir la base de datos, lo que provoca que ésta necesite menos espacio en disco para ser almacenada. Algunos DBMS ya cuentan con mecanismos para comprimir la base de datos y también existen programas de diferentes fabricantes que permiten realizar esta tarea.

Cuando una base de datos es comprimida, trabajan los mecanismos para comprimir los datos cuando son insertados. Cuando estos necesitan ser consultados, los mismos mecanismos descomprimen los datos para ser leídos. La desventaja de tener una base de datos comprimida es que leer y escribir los datos requieren más CPU de lo normal aunque como ya mencioné, ayuda a reducir las exigencias de almacenamiento. De nuevo, depende del ambiente de la base de datos y las necesidades el implementar o no esta técnica.

Otra de las ventajas que ofrece este método de optimización, es que se pueden leer más datos. Esto se debe a que una cantidad mayor de datos es almacenada en los archivos de datos. A su vez, los datos estarán almacenados en cache lo que permite una lectura de datos mayor que si estuvieran descomprimidos.

4.2.1.9. Ubicación de archivos

La ubicación de los archivos de datos de la base de datos es otro de los aspectos que aunque no lo parezca, puede tener un alto impacto en el rendimiento. Las bases de datos procesan mucho I/O por lo que es importante, en la medida de lo posible, minimizar el costo de lectura y escritura de disco.

Lo primero que se tiene que hacer para asignar una ubicación a los archivos de datos, es separar los índices de los datos, es decir, almacenar los índices en un archivo de datos y los datos como tal en otro archivo. Esto se recomienda debido a que si los índices se encuentran en el mismo archivo donde residen los datos, será más lento dar lectura a los

datos con sus respectivos índices, reduciendo la velocidad y por lo tanto el rendimiento de la base de datos.

Similar al aspecto anterior, también es recomendable colocar las tablas que más tienen acceso en archivos de datos diferentes. Al igual que con los índices, la lectura de las tablas será más rápida que si las tablas están en un mismo archivo de datos.

Otra consideración para ubicar archivos de datos en diferentes discos es cuando una tabla esta particionada en diferentes archivos, como se mencionó en la sección 4.2.1.1. Tras separar los archivos de datos de la tabla en diferentes dispositivos, se da pie a que operaciones de paralelismo se presenten. De esta forma, una consulta que requiera obtener datos de tablas las cuales tienen varios archivos de datos en diferentes discos reducirá la latencia[28] del disco para obtener los datos.

Sin embargo, los archivos de datos no son los únicos que deberían ser ubicados en discos separados. Como ya mencioné en la sección 4.1.2, existe un componente de la base de datos denominado log de transacciones. Este histórico de transacciones genera archivos en disco de determinado tamaño. La configuración para que estos archivos sean almacenados en discos diferentes donde se almacenan los archivos de datos no sólo mejorará el rendimiento de la base de datos (por la escritura de un disco diferente) sino que permitirá respaldar este log sin necesidad de dar de baja el servicio de la base de datos.

El objetivo de ubicar y distribuir los archivos de la base de datos en diferentes dispositivos de almacenamiento es para optimizar el acceso a los datos y para reducir la contención de los discos.

4.2.1.10. Tamaño de paginado

El almacenamiento de los datos en las bases de datos suele estar definido por lo que se denomina tamaño de paginado o tamaño de bloques de almacenamiento. Este tamaño de paginado sirve para almacenar las filas de las tablas en disco.

[28] La latencia es el tiempo que toma a un disco recuperar información que está escrita en sí mismo.

Para llevar a la práctica la técnica de rendimiento de paginado, se debe tener un control bastante amplio y preciso del almacenamiento de los registros de la base de datos. Esto se debe a que las filas en las tablas miden una cantidad de bytes, más aparte un pequeño número de bytes de encabezado por fila. De esta forma se debe medir el peso en bytes de un registro por su tamaño en sí, más el encabezado, para posteriormente determinar el tamaño del paginado para los datos. Más aún, también debe ser considerado un espacio vacío para almacenar los nuevos registros que vayan a ser insertados.

Elegir el tamaño adecuado de paginado para la base de datos ayuda a mejorar el rendimiento de I/O. Por lo general, esta característica puede ser configurada desde el DBMS. Por ejemplo, en Oracle el tamaño mínimo de bloques de datos es de 2 KB y el default es de 8 KB. Este valor es configurado en una variable de la base de datos denominada *DB_BLOCK_SIZE* y es un parámetro que no puede ser modificado una vez que se estableció su tamaño. Cabe mencionar que el tamaño de bloques de datos depende del tamaño de bloques del sistema operativo.

4.2.1.11. Reorganización

Aunque es el administrador de base de datos y los usuarios, tanto de aplicaciones como de la base de datos, quienes indican los cambios que deben hacerse, el DBMS es quien hace las operaciones y procesamientos correspondientes para la modificación de los datos. La manera en la que el DBMS controla los datos físicamente puede causar problemas de desempeño en la base de datos.

Este tipo de problemas puede presentarse cuando una base de datos lleva mucho tiempo en un ambiente de producción. Lo anterior puede ser causado si el número de transacciones ha aumentado o si la cantidad de datos se ha expandido demasiado.

Una desorganización de las bases de datos ocurre cuando el almacenamiento lógico y físico tiene áreas de almacenamiento dispersas, físicamente no son continuos, o están tan mal organizados que no pueden ser suficientemente útiles para un ambiente en producción. Algunas de las razones por lo que esto ocurre pueden ser las siguientes.

Una razón puede ser por no tener los datos con *clustering*. Si el DBMS no tiene habilitada esta capacidad o está mal implementada, los datos no se almacenarán en un orden al momento de ser insertados o cambiados. Lo anterior provoca que las consultas tarden más que si los datos estuvieran bajo *clustering*.

Otra de las razones puede ser por la fragmentación. Puede que existan áreas en los archivos que son muy pequeñas para almacenar datos y para ser usadas productivamente. Como consecuencia, aparte de desperdiciarse espacio para datos, el rendimiento de la base disminuye, ya que el sistema requiere más I/O para obtener datos.

En ocasiones, cuando se presentan migraciones de datos o el simple cambio de contenido de las filas, cabe la posibilidad de que se termine el espacio en los archivos de datos. En estos casos, el DBMS busca espacio en otros archivos de datos para almacenar los registros. La consecuencia de que los registros tengan que ser reubicados es que al momento de ser consultados es necesario más I/O, debido a que tiene que leer los registros en diferentes archivos de datos que si se hiciera una sola lectura en un solo archivo de datos.

Otra situación que provoca la desorganización de bases de datos, es cuando el DBMS tiene que crear espacios nuevos para acomodar nuevos datos. Esta situación fue explicada en la sección 4.2.1.5.

Cuando los archivos de datos que almacenan determinados *tablespaces* se llenan, algunos DBMS pueden expandir el espacio del archivo de datos. Aunque es una solución contra la falta de espacio, esto provoca que los espacios que se agreguen para expandir el archivo no sean continuos al archivo original provocando desorganizar físicamente los datos.

Aunque los problemas anteriores son concretos, las soluciones a ellos pueden ser diferentes dependiendo del DBMS. Para corregir la desorganización de las bases de datos se pueden ejecutar programas de reorganización que pueden ser parte del conjunto de herramientas del DBMS. En caso de que el DBMS no cuente con herramientas para

cumplir esta tarea, la manera más efectiva, aunque no la más fácil, es recreando los objetos y datos de la base de datos, es decir, exportando o respaldando la base de datos, luego borrarla, y después restaurarla a partir del respaldo que se creó.

Para determinar el momento apropiado para reorganizar una base de datos, ésta misma cuenta con información en las tablas del sistema o en tablas especiales. Esta característica puede no está presente en algunos DBMS.

Las características que fueron mencionadas en este apartado pueden ser aplicadas o no en los diversos DBMS. Para mejorar el rendimiento de las bases de datos y de las aplicaciones es recomendable que se apliquen tantas de estas tareas de rendimiento como sean posibles y sean soportadas por el DBMS que se está utilizando.

5. Comparación de manejadores

En este capítulo expondré los tres manejadores de bases de datos relacionales que fueron seleccionados para realizar esta tesis. Además, explicaré la situación actual del ambiente de producción de la base de datos del CHEM, así como las variables que se tomaron en cuenta para medir el desempeño de la misma. Posteriormente explicaré la migración y puesta en marcha de la base de datos en los diferentes manejadores, y finalmente expondré la manera en que se hicieron las pruebas y los resultados de éstas.

5.1. Investigación preliminar sobre los RDBMS

Para realizar esta tesis, después de buscar la información sobre cuestiones teóricas pertinentes (descrita en los capítulos anteriores), fue necesario seleccionar los RDBMS[29] en los que iba a poner a funcionar la base de datos del CHEM. Los RDBMS que fueron investigados y que se tomaron en cuenta fueron MySQL y Oracle, por ser éstos de los manejadores más usados en el mercado, así como el mismo PostgreSQL donde se encuentra actualmente la base de datos del CHEM. Otros manejadores, como SQL Server, fueron descartados ya que no funcionan en plataforma Linux/Unix.

A continuación expondré algunas características de estos RDBMS, empezando por MySQL, siguiendo con PostgreSQL y finalizando con Oracle.

5.1.1. MySQL

La página oficial de MySQL define a este RDBMS como la base de datos de código abierto más popular del mundo[30], la cual es desarrollada, distribuida y soportada por Oracle. La versión seleccionada para realizar las pruebas fue la 5.5, debido a que es la versión instalada y utilizada en el servidor del GIL. Algunas de las características generales de MySQL son las siguientes.

- Es un manejador de bases de datos relacionales.

[29] Manejador de base de datos relacional, RDBMS por sus siglas en inglés – *Relational Database Management System*.
[30] http://dev.mysql.com/doc/refman/5.5/en/what-is-mysql.html, visitada el 04 de abril de 2013.

- Es de código abierto, esto quiere decir que quien sea puede utilizarlo y modificarlo a conveniencia si así se requiere. Para modificar su código existen lineamientos de lo que está y no está permitido[31].
- Es rápido, seguro, escalable y fácil de utilizar. Puede ser utilizado en computadoras de cualquier tipo ya que no exige muchos requerimientos de software.

En un principio, el objetivo primordial de MySQL era manejar bases de datos gigantes de forma rápida y efectiva. Esta característica lo llevo a obtener una gran demanda en el mercado. Por otro lado, MySQL cuenta con características específicas que se agregan cada que una versión nueva sale al mercado. Estas características se refieren al funcionamiento interno de este RDBMS.

Algunas de estas características[32] son la agrupación de subprocesos, la cual permite manejar los hilos para las conexiones de los diferentes clientes. En MySQL *Enterprise*[33] se implementó un componente de auditoria para monitorear las conexiones y consultas de los clientes del RDBMS. También cuenta con dos métodos nuevos de autenticación. Uno es a través de *plugins* que se instalan en el RDBMS y en el cliente para habilitar las conexiones. El otro es por medio de usuarios proxy, los cuales fungen como máscaras para que los usuarios se conecten y se comporten en el RDBMS como si fueran el usuario proxy que utilizan.

También cuenta con escalabilidad multinúcleos, lo que permite al RDBMS usar los diferentes núcleos de los procesadores para aprovechar el procesamiento máximo que puedan soportar. Otra de las características que fueron presentadas en la versión 5.5 es la replicación síncrona de transacciones, lo cual permite hacer cambios en la base de datos maestra y en una base de datos de réplica.

[31] Para más información acerca de los lineamientos que utiliza MySQL visitar http://www.gnu.org/licenses/license-list.html.
[32] Para ver el listado completo de las características de esta versión de MySQL revisar el sitio oficial en http://dev.mysql.com/doc/refman/5.5/en/mysql-nutshell.html.
[33] Esta versión de MySQL tiene una licencia de paga.

Una de las características más importantes, al menos para esta tesis, es la mejora del particionamiento. Esta característica será retomada en el capítulo 6 donde se implementan operaciones de rendimiento para la base de datos del CHEM. Esta característica soporta dos formas de particionamiento. Una es por rango de columnas (*range partitioning*) y la otra es por listas de columnas (*list partitioning*). Esta característica soporta particionamiento para los tipos de datos *DATE*, *DATETIME*, *CHAR* y *VARCHAR*. También, como parte de las nuevas características, es posible borrar todos los registros de una partición sin afectar la estructura de la tabla. Esto se logra a través del comando *ALTER TABLE Tabla1 TRUNCATE PARTITION*.

5.1.2. PostgreSQL

Según el sitio oficial de PostgreSQL, éste es un sistema manejador de bases de datos objeto-relacionales, de código abierto, que soporta consultas complejas, llaves foráneas, *triggers*, vistas, entre otros objetos. Debido a que es un manejador de código abierto puede ser usado, modificado y distribuido por cualquier persona. Al igual que MySQL, para la realización de esta tesis, se tomará la versión que está instalada en los servidores del GIL, la cual es la versión 8.4.

Esta versión cuenta con mejoras en cuanto a la administración, las consultas y programación de bases de datos. Además, muchos de los cambios que tiene esta versión fueron especialmente orientados a la mejora de la administración y el rendimiento. Algunas de las características de esta versión son las siguientes[34]:

- Restauración de bases de datos en paralelo, acelerando la recuperación desde un respaldo ocho veces más rápido que versiones anteriores.
- Permisos a nivel de columnas, permitiendo un mayor control en la información más delicada.
- Disponibilidad de múltiples conjuntos de caracteres para soportar ambientes que tienen diferentes lenguajes.

[34] Para ver todas las características visitar: http://www.postgresql.org/about/news/1108/.

- Herramientas de monitoreo de consultas para administrar y controlar las que se ejecutan en el RDBMS.
- Herramientas para el control de consultas actuales, carga de consultas y *deadlocks*.

Otra de las mejoras de PostgreSQL para esta versión es que el uso de procedimientos almacenados es más flexible que en otras versiones. También cuenta con mejoras para el rendimiento de la base de datos.

5.1.3. Oracle

Según su página oficial, Oracle es la base de datos número uno del mundo. Éste ofrece un rendimiento líder en la industria, escalabilidad, seguridad y fiabilidad ya sea en un grupo de servidores (*cluster*) o en servidores únicos, con soporte para Unix, Linux y Windows. En esta investigación, se utilizara Oracle 11g para Linux.

Algunos de los beneficios que ofrece este manejador es protección ante fallas de servidor, errores humanos, y reduce tiempos de inactividad planeados. También asegura los datos y permite tener seguridad a nivel de registros en las tablas. Además ofrece un control y manejo de almacenamiento rentable, así como reducir las costosas operaciones de I/O, comprimir los datos y maximizar la utilización de recursos de almacenamiento para todas las bases de datos aparte de ofrecer opciones de respaldo óptimas para los ambientes en producción.

Oracle cuenta con una gran cantidad de características[35]. Entre las que más destacan son el uso, control y asignación de memoria que utiliza para la base de datos. Otra característica muy importante es el conjunto de comandos para deshacer transacciones con borrado de tablas e incluso de bases de datos. También cuenta con opciones para monitorear el rendimiento de la base de datos.

Hablando de la arquitectura de la base de datos, Oracle cuenta con diversos procesos y estructuras de memoria que le conceden no solo el mejor funcionamiento, sino que

[35] Para ver las características de Oracle 11g visitar:
http://docs.oracle.com/cd/E11882_01/server.112/e25789/toc.htm.

también le permiten al DBA tener el control de asignar la cantidad de recursos que él desee.

A diferencia de los RDBMS mencionados anteriormente, Oracle no es de licencia libre ni de código abierto. Oracle es un producto de licencia comercial para bases de datos relacionales y cuenta con una gama de programas para la administración y control de la información, todos ellos también de licencia comercial.

Para fines de esta tesis, se utilizará la versión completa de bases de datos de Oracle (*Oracle Enterprice Edition*). Es posible utilizar esta versión sin comprarla ya qué se especifica que puede ser usada si se trata de investigaciones, pruebas, prototipos, y/o demostraciones[36].

5.2.Estrategia de evaluación para la comparación de los RDBMS

Para poder comparar el rendimiento de los RDBMS seleccionados, es necesaria una estrategia de evaluación que nos permita compararlos bajo las mismas condiciones para posteriormente determinar cuál es más rápido. Lo anterior se va a determinar mediante dos etapas de pruebas donde la primera consistirá en medir el tiempo de respuesta de la base de datos desde la interfaz de consulta del CHEM, y en la segunda se obtendrá el tiempo de respuesta de los RDBMS desde la consola SQL de cada uno de ellos.

Ambas etapas de pruebas estarán basadas en una comparación de los manejadores sin implementar ninguna tarea de rendimiento, es decir, con la instalación por default de cada uno de ellos. Además, se harán diferentes consultas para determinar el tiempo que tardan en generar el resultado de cada una de ellas.

Una consideración en cuanto a la realización de las pruebas, para fines prácticos, es que la primera ronda de pruebas se hará con los datos de la base de datos del CHEM, es decir, con aproximadamente 1 millón 700 mil registros. Por otro lado, la segunda ronda de pruebas se hará con los datos del CEMC, descrito en la sección 2.6, ya que éste cuenta con un poco más de doce millones de registros. Esta prueba se realizará con el fin de hacer

[36] Para ver la licencia completa de Oracle visitar: http://www.oracle.com/technetwork/licenses/standard-license-152015.html en el apartado *Licence Rights.*

una simulación en caso de que el CHEM llegue a contar con millones de palabras y también con el fin de que esta tesis ayude a optimizar otros corpus del GIL.

Antes de iniciar con las pruebas, fue importante determinar algunos aspectos que resultaron fundamentales para interpretar el comportamiento del RDBMS. Estos aspectos son los siguientes.

- Las variables a controlar, es decir, determinar en qué ambiente serán realizadas las pruebas.
- Parámetros a medir. Qué parámetros y qué medidas se van a tomar para comparar la velocidad de los manejadores.
- Estructura actual de la base de datos del CHEM. Es necesario saber si la base de datos cuenta con características propias del manejador donde reside actualmente. Lo anterior para determinar si es posible recrear la base de datos en los RDMBS seleccionados.
- Adecuación de la base de datos y carga de datos. Poner en funcionamiento la base de datos del CHEM en MySQL y en Oracle. Posteriormente agregar los datos en ambas bases de datos.
- Detectar la velocidad actual de las consultas del CHEM. Como éstos son los valores actuales, será el punto de partida para determinar si MySQL u Oracle es o son más rápidos que PostgreSQL (manejador actual de la base de datos del CHEM).

Posterior a la descripción de los puntos anteriores, se realizaron las pruebas con MySQL y Oracle para descubrir si uno de ellos era mejor que el manejador donde está actualmente la base del CHEM. A continuación describiré detalladamente los puntos antes mencionados.

5.2.1. Variables a controlar

Para esta investigación, las variables a controlar son todos aquellos factores que pueden tener intervención en el desempeño de la base de datos. Estas variables son: las capacidades de hardware del equipo de cómputo, el sistema operativo donde los RDBMS

están instalados, el código de la consulta y la cantidad de registros en las base de datos. A continuación describiré estas variables.

El equipo donde se realizará la primera etapa de pruebas es en una PC del GIL. Esta PC cuenta con una memoria RAM de 4 GB, un procesador Intel i3 y Microsoft Windows 7 como sistema operativo. También tiene instalados los RDBMS descritos anteriormente. La versión de Oracle es una versión gratuita conocida como *Oracle Express*, la versión de MySQL es la 5.5 y la versión de PostgreSQL es 8.4.

Por otro lado, el servidor donde se hará la segunda etapa de pruebas es un *cluster* de servidores del Instituto de ingeniería. Debido a que es un *cluster*, el servidor que se asignó para esta etapa de pruebas es un servidor virtual. Este servidor tiene un arreglo de discos configurados con RAID 5[37], tiene una memoria RAM de 12 GB al igual que de área *swap*, y 4 procesadores i7. El sistema operativo que tiene instalado este servidor es Linux en su distribución OpenSUSE. En este servidor se cuenta con Oracle en su versión 11g, MySQL versión 5.5 y PostgreSQL 8.4.

Otro factor que necesariamente se debe tomar en cuenta para medir el rendimiento de los RDBMS, es la consulta que se ejecuta para generar las concordancias. Esta consulta se ejecuta cada vez que un usuario ingresa una palabra de búsqueda en la interfaz del CHEM. Lo que hace la consulta es obtener la palabra de búsqueda junto con su identificador, el identificador del documento donde se encuentra, su posición, el año del documento, la posición de la décima palabra anterior y la posición de la décima palabra posterior a la palabra de búsqueda.

Aunado a la consulta, como factor de análisis del rendimiento de los RDBMS, está el número de registros que ésta obtiene de la base de datos. Como sabemos, el CHEM

[37] Un RAID, por sus siglas en inglés *Redundant Array of Independent Disks*, es una configuración para mejorar el rendimiento de los discos, y para mejorar la fiabilidad de los datos. Existen diferentes métodos de este sistema, donde el quinto (RAID 5) tiene como característica intercalar sectores de datos a través de los discos. La paridad de los datos para estos sectores es calculada y guardada en los mismos discos. Para el sistema es como si existiera un solo disco y los datos están distribuidos a través de los múltiples discos. Tomado de http://www.google.es/patents?hl=es&lr=&vid=USPAT5805788&id=IoYiAAAAEBAJ&oi=fnd&dq=raid+5&printsec=abstract#v=onepage&q=raid%205&f=false.

cuenta con aproximadamente un millón 700 mil registros y estos son los que serán utilizados para la primera ronda de pruebas. Por otro lado, para la segunda etapa de pruebas se tienen más de doce millones de registros. Para ambos casos, la consulta es la misma.

5.2.2. Análisis de la estructura actual del CHEM

Aunque la base de datos del CHEM cuenta con varias tablas y relaciones, solo se describirá la parte de la base de datos que es de interés para esta tesis, es decir, las tablas y relaciones que utiliza el generador de concordancias del CHEM. Las tablas de nuestro interés son las de palabras y documentos.

Las palabras están almacenadas en una tabla en la base de datos. También existe una tabla de documentos y una que almacena qué palabras se encuentran en qué documentos. La Figura 5-1 ilustra lo mencionado anteriormente.

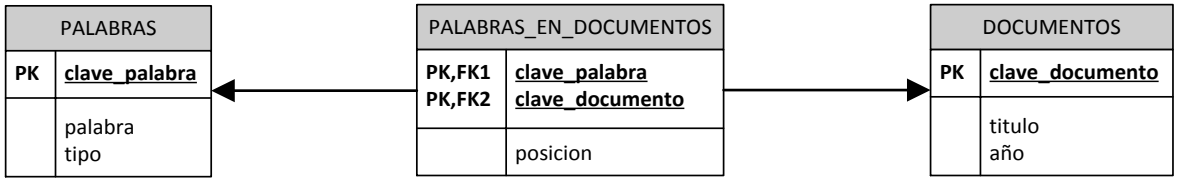


Figura 5-1 Representación de palabras, documentos y palabras por documento

Como se puede ver en la figura anterior, la tabla que tiene las palabras por documento (PALABRAS_EN_DOCUMENTOS) cuenta con un campo el cual es la posición de la palabra en el documento. Esta posición indica en dónde se encuentra la palabra por lo que puede haber palabras duplicadas en un documento, pero con diferente posición. La tabla de palabras almacena la clave de la palabra, la palabra en sí y el tipo de palabra. El tipo de las palabras se refiere a las formas que puede tener esa palabra, por ejemplo: normalizada, ortográfica o lema; como se explicó en la sección 2.5.1 donde se habla de las búsquedas que soporta el CHEM. La tabla de documentos almacena la clave del documento, el título del mismo, su año y otros campos que no son necesarios mencionar porque no serán utilizados en esta tesis.

5.2.3. Adecuación de la base de datos y carga de datos

Para poder realizar las rondas de pruebas fue necesario adecuar la base de datos del CHEM en MySQL y en Oracle. Como primer paso, fue necesario analizar si la creación de las tablas, vistas, índices, funciones, procedimientos almacenados y *triggers* eran soportados en los RDBMS. Respecto a lo anterior, se tuvo que ajustar la creación de algunos objetos en los RDBMS ya que existen formas diferentes de creación de esos objetos (sintaxis) para cada uno. A continuación describiré los ajustes, para cada uno de los objetos, que fueron necesarios para poder cargar la base de datos del CHEM en Oracle y MySQL.

Comenzaré por describir los cambios en las tablas de la base de datos. El primer problema al que me enfrenté fue que en Oracle no existe el tipo de dato *boolean*[38] por lo que tuvo que ser sustituido por el tipo de dato *INTEGER* y asignar el valor 1 en caso de verdadero y 0 en el caso contrario. Por otro lado, MySQL sí acepta la sintaxis del tipo de dato *BOOLEAN* pero en la descripción de las tablas los campos aparecen bajo el tipo de dato *TINYINT* y los mismos valores que fueron asignados en Oracle, es decir 1 para verdadero y 0 para falso, se asignaron aquí. La Figura 5-2 muestra una descripción con una tabla llamada “booleano”.

[38] Tampoco existe este tipo de dato en la definición de funciones y procedimientos almacenados.

```
mysql>
mysql> CREATE TABLE booleano(dato boolean);
Query OK, 0 rows affected (0.02 sec)

mysql> desc booleano;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| dato  | tinyint(1) | YES  |     | NULL    |       |
+-------+------------+------+-----+---------+-------+
1 row in set (0.00 sec)

mysql> INSERT INTO booleano VALUES(true);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO booleano VALUES(false);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM booleano;
+------+
| dato |
+------+
|    1 |
|    0 |
+------+
2 rows in set (0.00 sec)

mysql>
```

Figura 5-2 Ejemplo de la creación de una tabla con tipo de dato booleano

Como se puede apreciar en la figura anterior, se crea una tabla bajo el nombre de "booleano", donde ésta tiene un campo llamado "dato" de tipo booleano. Inmediatamente después se muestra la descripción de la tabla donde la columna "*TYPE*" nos indica que es un tipo de dato *TINYINT* que almacena dos valores (1 y 0). Después se presenta la inserción de un valor verdadero y uno falso, y cuando se consulta el contenido de la tabla nos muestra 1 para verdadero y 0 para falso.

También como parte de los ajustes en tablas se tuvo que asignar un rango de valores para el tipo de dato *VARCHAR*. Este tipo de dato almacena cadenas de caracteres y en la definición del mismo se puede o no asignar un rango de valores. Si no se asigna un valor se asigna automáticamente un valor por default, al menos para PostgreSQL. En el caso de MySQL y Oracle es necesario que se asigne un valor a este tipo de dato, de lo contrario se mandará un error de sintaxis.

Otro de los aspectos que tuvieron que ser modificados en cuanto a la creación de tablas en MySQL fue la creación de secuencias. Las secuencias son un objeto de base de

datos que tiene como función incrementar valores numéricos, como se explicó en la sección 3.5. En el caso de MySQL no existen las secuencias por lo que se utilizó la cláusula "*auto_increment*". Esta cláusula se comporta como una secuencia e inicia con el valor 1 a menos que se le especifique otro. En Oracle las secuencias se manejan igual que en PostgreSQL por lo que no se hicieron cambios en este aspecto.

Después de terminar la creación de tablas en los RDBMS, se decidió continuar con la creación de los índices de las tablas. Los índices son objetos de bases de datos que contienen entradas para cada valor que aparece en la columna a indexar. Recordemos que los índices permiten un acceso más rápido y directo a las filas de las tablas.

El índice más común en las bases de datos es conocido como índice *btree*[39]. Un índice *btree* es una lista ordenada de valores divididos en niveles[40]. Al asociar una llave con una fila o un conjunto de estas, este tipo de índices provee un mejor rendimiento para consultas que obtienen grandes cantidades de datos para búsquedas exactas y por rangos. La siguiente figura muestra un ejemplo de un índice *btree*.

[39] *Balanced tree* en inglés.
[40] Definición tomada de http://docs.oracle.com/cd/E11882_01/server.112/e25789/indexiot.htm visitada el 25 de Abril de 2013.

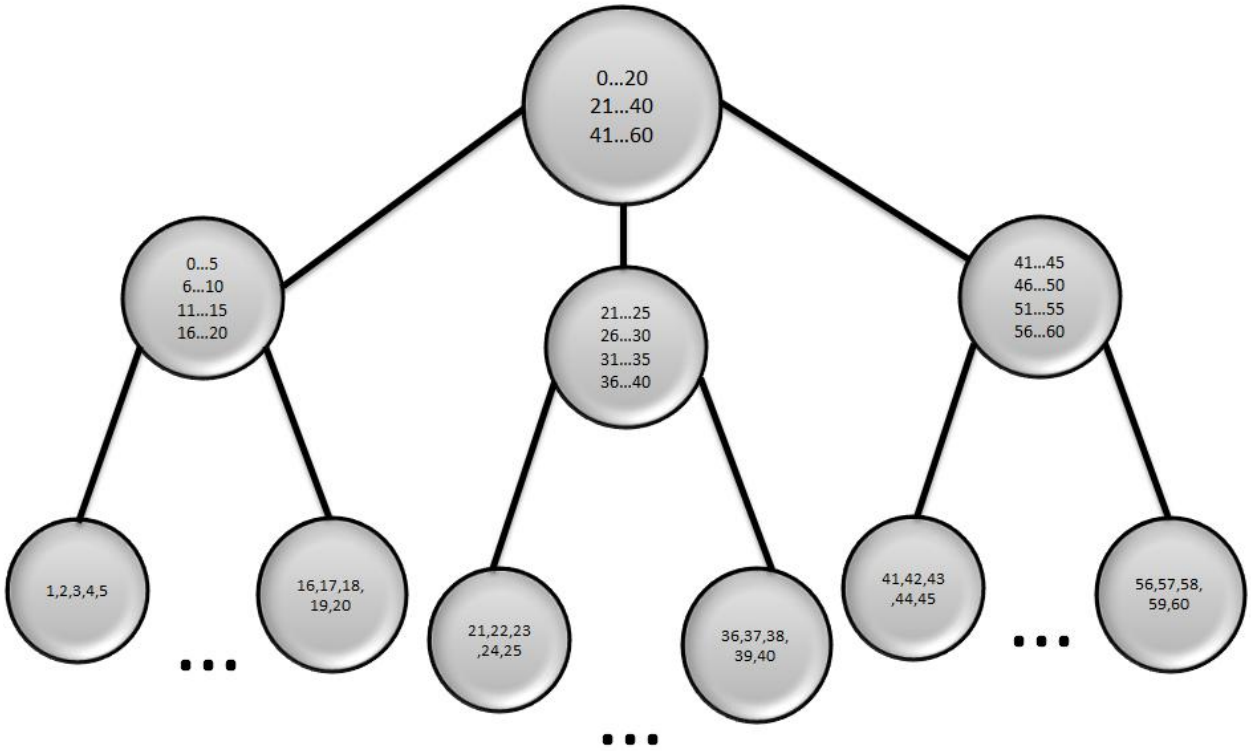


Figura 5-3 Ejemplo de la estructura lógica de un índice *btree* (basada en figura 3-1 de http://docs.oracle.com/cd/E11882_01/server.112/e25789/indexiot.htm).

Una de las características de este tipo de índices es que cuentan con un nodo padre[41]. Por otro lado también cuentan con un conjunto de ramas para las búsquedas y con grupos de hojas las cuales almacenan valores. Otra característica de los arboles es la altura, la cual se cuenta desde el nodo padre hasta las hojas que están en el nivel más bajo. En el ejemplo anterior el árbol tiene una altura de tres.

El orden de los datos del índice *btree* es de forma ascendente donde las ramas almacenan índices que apuntan a los valores almacenados en las hojas. Se dice que es un índice *btree* o también un árbol balanceado ya que las hojas se mantienen de forma automática a la misma profundidad. De esta manera recuperar los datos del índice desde cualquier posición toma aproximadamente el mismo tiempo.

En Oracle las ramas almacenan una clave que hace referencia a las hojas y los valores de estas últimas. El número de valores y apuntadores por cada rama depende del tamaño

[41] También conocido como *root*.

de bloques de la base de datos (8 KB por default). Las hojas contienen el valor indexado junto con su *rowid*[42] y están ordenadas y ligadas de derecha a izquierda.

En la Figura 5-3 el nodo padre almacena referencias a las ramas, una del 0 al 20, otra del 21 al 40 y la última del 41 al 60. La primera rama almacena valores del 0 al 20 e indica que tiene una hoja para los valores individuales del 0 al 5, la segunda hoja contiene los valores del 6 al 10 y así sucesivamente hasta el último apuntador de la rama, para este ejemplo hasta el 20. Este ordenamiento se repite para cada una de las ramas y está acomodado de derecha a izquierda.

La base de datos del CHEM cuenta con índices *btree*. Estos tuvieron que ser adaptados para MySQL y Oracle. La sintaxis de PostgreSQL para crear un índice se muestra en el código siguiente.

```
CREATE INDEX indx_numero ON numeros USING btree(numero);
```

Para crear un índice en PostgreSQL se define su nombre, la tabla en donde se creará el índice, su tipo[43] y los campos a indexar. En el ejemplo anterior se crea un índice llamado "indx_numero", con la cláusula *ON* indicamos que la tabla "numeros" tendrá el índice y con la cláusula *USING btree* se indica que es un índice de tipo *btree* usando el campo "numero".

Sin cambiar tanto en la sintaxis, para crear un índice en MySQL se define el nombre del índice, la tabla, los campos que serán indexados y por último el tipo de índice.

```
CREATE INDEX indx_numero ON numeros(numero) USING btree;
```

El ejemplo anterior crea un índice llamado "indx_numero" seguido de la cláusula *ON* la cual indica la tabla "números" y el campo "numero". Finalmente se indica que es un índice *btree*.

[42] El *rowid* es una columna que se encuentra en todas las tablas que existen en Oracle. Almacena un valor hexadecimal, existe uno por cada registro y es único por cada una de las filas almacenadas.
[43] Aunque existen varios tipos de índices en PostgreSQL, solo se explicara el índice *btree* ya que los otros tipos son especializados para otras funciones.

Por último la adecuación de índices para Oracle cambia solo en el sentido de la especificación del tipo de índice. El índice que crea por default es de tipo *btree* por lo que no es necesario especificar en la sintaxis esta cláusula. En el ejemplo siguiente se muestra la definición del índice donde se indica el nombre de este y la tabla con los campos a indexar.

```
CREATE INDEX indx_numero ON numeros(numero);
```

Para continuar con la adecuación de la base de datos fue necesario adaptar las funciones. En el caso de PostgreSQL, antes de crear una función, se debe declarar la creación del lenguaje *plpgsql.* Éste es el lenguaje que da soporte a las funciones y procedimientos almacenados en PostgreSQL. El contenido de las funciones está dividido en diferentes partes las cuales son el encabezado, la declaración de variables, la zona de ejecución, que es el cuerpo que contiene las acciones que la función debe de hacer, el valor de retorno de la función y finalmente un área de excepciones que ayuda a controlar el flujo de la función. El siguiente ejemplo muestra un ejemplo de una función en PostgreSQL.

```
CREATE FUNCTION inserta_numero(integer) RETURNS VARCHAR
    -- Encabezado
    LANGUAGE plpgsql
    AS $_$
    -- Declaracion de variables
    DECLARE
        numero_entrada ALIAS FOR $1;
        valor VARCHAR;
    -- Instrucciones a cumplir
    BEGIN
        INSERT INTO numeros (numero) VALUES(numero_entrada);
        valor := 'Numero insertado';
    -- valor de retorno
    RETURN valor;
    -- Fin de la función
END; $_$;
```

En PostgreSQL, después de crear el lenguaje, las funciones se declaran con el nombre de está, entre paréntesis[44] los tipos de datos de los parámetros que recibe de entrada y el tipo de dato que va a regresar la función. En la figura anterior se crea una función llamada "inserta_numero" la cual recibe como parámetro un INTEGER, es decir un valor numérico y regresa una cadena de caracteres.

En el encabezado se asigna el nombre del lenguaje que va a utilizar[45] seguido del símbolo '$_$' que indica que la función inicia en ese punto. Dentro de la declaración de variables se puede ver una, llamada "numero_entrada", y a continuación la indicación de que es un alias para $1. Lo anterior quiere decir que es el nombre de la variable que entra como parámetro. También se puede ver una variable llamada "valor" la cual es de tipo VARCHAR.

El cuerpo de la función debe iniciar con la palabra reservada BEGIN y terminar con la palabra END. Dentro del código de ejemplo se puede apreciar la inserción de un número en la tabla números, el cual es el que entra como parámetro. En seguida, la variable "valor" toma el valor de cadena que dice "Número insertado". En caso de que ningún número sea asignado como parámetro en la ejecución de la función se enviará un mensaje de error. Finalmente se retorna el valor de la variable "valor" y termina la función. Cabe resaltar que los ejemplos de funciones que se muestren en esta sección no serán tomados de la base de datos del CHEM por cuestiones de seguridad. Sin embargo, se darán ejemplos similares para comprender el tipo de adaptaciones que fueron realizadas.

La adaptación de la función anterior en MySQL sería como se muestra en el siguiente ejemplo. En el caso de MySQL las funciones están estructuradas de forma distinta a PostgreSQL. En estas funciones no hay encabezado sino que se entra directamente al cuerpo de la función. Dentro de éste está la declaración de variables, las instrucciones a cumplir por la función, el valor de retorno y el fin de la función.

[44] No necesariamente llevan un parámetro entre paréntesis.
[45] También puede ponerse al final de la definición de la función.

```
DELIMITER //
CREATE FUNCTION inserta_numero(numero_entrada integer)
RETURNS VARCHAR
-- Cuerpo de la función
BEGIN
    -- Declaracion de variables
    DECLARE valor VARCHAR;
        INSERT INTO numeros (numero) VALUES(numero_entrada);
        valor := 'Numero insertado';
    -- valor de retorno
    RETURN valor;
    -- Fin de la función
END;
DELIMITER ;
```

Antes de crear la función en MySQL es necesario cambiar el delimitador con el que se indica que termina una instrucción en la línea de comando de MySQL. Esto se debe a que el punto y coma (;) es utilizado dentro de la función. De no ser así MySQL lo interpretaría como fin de una instrucción cuando aún no lo es. El cambio del delimitador se hace como se muestra en el ejemplo, escribiendo la palabra reservada *DELIMITER* junto con el delimitador, en este caso doble diagonal (//).

Una vez que el delimitador ha sido cambiado se define la función con su nombre y sus parámetros de entrada. A diferencia de PostgreSQL el nombre de la variable de entrada se define dentro de los paréntesis en la declaración de la función. Seguido va el tipo de dato de la variable de entrada y el valor de retorno. En este caso la función se nombra "inserta_numero" y tiene como parámetro una variable llamada "numero_entrada" de tipo *INTEGER*. Después se declara una variable llamada "valor" de tipo *VARCHAR*. A continuación hay una inserción en la tabla "numeros" y la variable "valor" toma la cadena "Número insertado". Finalmente se retorna la variable "valor" y se finaliza la función con el delimitador que fue definido antes de la función. Como paso adicional se regresa el valor del delimitador a punto y coma.

Por último, en el caso de Oracle, la creación de las funciones también cambia en algunos aspectos. El código siguiente muestra un ejemplo con la misma función de los ejemplos anteriores.

```
CREATE FUNCTION inserta_numero (numero_entrada IN INTEGER)
RETURN VARCHAR2
    -- Declaracion de variables
    IS
        valor VARCHAR2(30);
    -- Instrucciones a cumplir
    BEGIN
        INSERT INTO numeros (numero) VALUES(numero_entrada);
        valor := 'Numero insertado';
    -- Valor de retorno
    RETURN valor;
    END;
/
```

En Oracle las variables que la función recibe como parámetros están diferenciadas por tres tipos. Estos tipos son *IN*, *OUT* e *INOUT*. El tipo *IN* indica que es una variable que será utilizada dentro de la función y su valor no podrá ser modificado. El tipo *OUT* indica que es un parámetro de retorno y su valor puede ser alterado dentro de la función. Finalmente el tipo *INOUT* es un parámetro de entrada y de retorno y su valor puede ser modificado dentro de la función.

Luego entonces, el ejemplo anterior muestra la definición de la función llamada "inserta_numero" seguida de la variable de entrada de tipo INTEGER, la cual es de tipo *IN*, terminando con el valor de retorno el cual es de tipo *VARCHAR2*. Cabe mencionar que la palabra reservada *RETURN* (la cual indica que la función regresa un valor) no termina con "S", es decir *RETURNS* como es en el caso de PostgreSQL y MySQL. También se puede observar que el tipo de dato de retorno es VARCHAR2. Este tipo de dato es propio de Oracle y sólo es una variación del nombre del tipo de dato VARCHAR, es decir no existe diferencia alguna entre ellos.

Después de la definición de la función se establece la parte de declaración de variables. Oracle utiliza la palabra reservada *IS* en lugar de *DECLARE* como lo hacen PostgreSQL y MySQL. A continuación se inicia la zona de ejecución de la función indicándola con la palabra *BEGIN*. La ejecución de la función es una inserción en la tabla "números" donde se inserta el valor de entrada "numero_entrada", la variable "valor" toma la cadena "Numero insertado" y se indica la cláusula de retorno de la variable "valor" (RETURN valor) y el fin de la función. Para indicar que la definición de la función ha terminado se escribe la diagonal (/).

Para finalizar con la adaptación de las funciones cabe mencionar que PostgreSQL permite crear varias funciones con el mismo nombre. La condición para que se pueda hacer lo anterior es que el número y tipo de parámetros por cada función no se debe repetir, es decir, no se puede tener una función con los mismos parámetros más de una vez. Para MySQL y Oracle no es posible hacer lo anterior. El nombre de las funciones no se puede repetir y por consecuencia no es posible crear varias de éstas con diversos parámetros. Para esta tesis fue necesario cambiar el nombre en MySQL y Oracle de las funciones de PostgreSQL que cumplían con esta característica.

Para poder ejecutar las funciones en PostgreSQL y también en MySQL basta con seguir la sintaxis de una consulta. Para ello se utiliza sólo la palabra *SELECT* y el nombre de la función. En el caso de Oracle se utiliza la palabra *EXECUTE* seguida del nombre de la función.

Por otro lado, los *triggers* son otro tipo de funciones que tuvieron que ser adecuadas a MySQL y Oracle. Un *trigger* en PostgreSQL está conformado de dos componentes. Por un lado se debe establecer la definición del *trigger* y por otro lado una función que requiere el mismo. El siguiente ejemplo muestra la definición de un *trigger* y su función en PostgreSQL.

```
-- Función

CREATE FUNCTION sp_inserta_otro_numero() RETURNS TRIGGER
    LANGUAGE plpgsql
    AS $$
    BEGIN
        NEW.otro_numero := NEW.numero + 2;
    RETURN NEW;
END;$$;

-- Trigger

CREATE TRIGGER tr_inserta_otro_numero
    BEFORE INSERT ON numeros
    FOR EACH ROW
    EXECUTE PROCEDURE sp_inserta_otro_numero();
```

El ejemplo anterior muestra la definición de una función llamada "sp_inserta_otro_numero" y la definición del *trigger* que la utiliza llamado "tr_inserta_otro_numero". Como se puede observar, la función tiene como valor de retorno un *trigger* y no recibe ningún parámetro de entrada.

Dentro de la definición de la función se tiene lo mismo que todas las funciones en PostgreSQL a excepción de que tiene una variable llamada *NEW*. Esta variable es un valor que es utilizado por los *triggers* y puede hacer referencia a un campo de una tabla. En el ejemplo se muestra el valor NEW.otro_numero, donde "otro_numero" es un campo de la tabla "números". A este valor se le asigna el campo NEW.numero más 2 lo que quiere decir que por cada inserción en el campo "numero" de la tabla "números", el valor de "otro_numero" tomara el valor de número más dos.

Por otro lado, la definición del *trigger* cuenta con un conjunto de parámetros. El primero de ellos es *BEFORE INSERT ON* y nos indica que su ejecución tomará lugar antes de una inserción en la tabla "numeros". El siguiente parámetro, *FOR EACH ROW*, quiere decir que su acción se ejecutará para cada una de las filas de la tabla. Por último, el

parámetro EXECUTE PROCEDURE es el parámetro que manda a llamar la función "sp_inserta_otro_numero".

A diferencia de los *triggers* de PostgreSQL, MySQL y Oracle tienen una sintaxis diferente. En cuestión de código sólo implementan el del *trigger*, es decir, no necesitan definir una función que sea utilizada por el *trigger*. El siguiente ejemplo muestra un *trigger* en MySQL.

```
DELIMITER //
CREATE TRIGGER tr_inserta_otro_numero BEFORE INSERT ON numeros
FOR EACH ROW BEGIN
    set NEW.otro_numero = NEW.numero + 2;
END
//
DELIMITER ;
```

La estructura de los *triggers* para MySQL sólo incluye su definición y dentro de la misma se especifican las acciones que debe cumplir. En el ejemplo anterior, al igual que en las funciones, cambia el delimitador de los comandos de MySQL por doble diagonal. La definición del *trigger* comienza con el nombre de éste, seguido de los parámetros *BEFORE INSERT ON* y *FOR EACH ROW.* A continuación tienen las palabras *BEGIN* y *END* las cuales indican que las acciones que debe ejecutar el *trigger* van entre estas dos. La acción que realiza el *trigger* de este ejemplo es la misma que la del anterior.

```
CREATE TRIGGER tr_inserta_otro_numero BEFORE INSERT ON numeros
FOR EACH ROW BEGIN
    :NEW.otro_numero := :NEW.numero + 2;
END;
/
```

Finalmente, este ejemplo muestra que el código de Oracle es similar al de MySQL y utiliza solo la definición del *trigger* con los parámetros *BEFORE INSERT ON* y *FOR EACH ROW*. También utiliza las palabras reservadas BEGIN y END para indicar que las acciones del *trigger* van dentro de éstas. La diferencia con MySQL y PostgreSQL es que la variable *NEW* utiliza dos puntos antes de ser declarada. Para indicar que la definición del *trigger* ha terminado, al igual que en las funciones, se escribe una diagonal.

Después de adaptar la estructura de la base de datos en los diferentes manejadores, sólo restaba cargar los datos. Para cargar éstos se tuvieron que ejecutar los scripts de bases de datos los cuales contenían las inserciones de los registros en las tablas. Se presentó un problema al momento de insertar los datos que contenían acento ya que la codificación de la base de datos del CHEM que tiene en PostgreSQL es *latin1*.

Para que los RDBMS aceptaran los acentos se tuvo que configurar la codificación de caracteres de éstos. Para lograr lo anterior, en MySQL se modificó la base de datos con el comando *ALTER DATABASE chem DEFAULT CHARACTER SET latin1*. En la figura Figura 5-4 se muestra la ejecución del comando anterior. Cabe resaltar que el comando *SHOW VARIABLES LIKE '%character_set%'* muestra la codificación actual de la base de datos.

```
mysql> SHOW VARIABLES LIKE '%character_set%';
+--------------------------+----------------------------------------------+
| Variable_name            | Value                                        |
+--------------------------+----------------------------------------------+
| character_set_client     | utf8                                         |
| character_set_connection | utf8                                         |
| character_set_database   | utf8                                         |
| character_set_filesystem | binary                                       |
| character_set_results    | utf8                                         |
| character_set_server     | utf8                                         |
| character_set_system     | utf8                                         |
| character_sets_dir       | /usr/share/mysql-community-server/charsets/ |
+--------------------------+----------------------------------------------+
8 rows in set (0.00 sec)

mysql> alter database chem default character set latin1;
Query OK, 1 row affected (0.00 sec)

mysql> SHOW VARIABLES LIKE '%character_set%';
+--------------------------+----------------------------------------------+
| Variable_name            | Value                                        |
+--------------------------+----------------------------------------------+
| character_set_client     | utf8                                         |
| character_set_connection | utf8                                         |
| character_set_database   | latin1                                       |
| character_set_filesystem | binary                                       |
| character_set_results    | utf8                                         |
| character_set_server     | utf8                                         |
| character_set_system     | utf8                                         |
| character_sets_dir       | /usr/share/mysql-community-server/charsets/ |
+--------------------------+----------------------------------------------+
8 rows in set (0.00 sec)

mysql> SET CHARACTER SET latin1;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE '%character_set%';
+--------------------------+----------------------------------------------+
| Variable_name            | Value                                        |
+--------------------------+----------------------------------------------+
| character_set_client     | latin1                                       |
| character_set_connection | latin1                                       |
| character_set_database   | latin1                                       |
| character_set_filesystem | binary                                       |
| character_set_results    | latin1                                       |
| character_set_server     | utf8                                         |
| character_set_system     | utf8                                         |
| character_sets_dir       | /usr/share/mysql-community-server/charsets/ |
+--------------------------+----------------------------------------------+
8 rows in set (0.01 sec)
```

Figura 5-4 Ejemplo del cambio de la codificación de caracteres en MySQL.

Como se puede ver en el ejemplo anterior, primero se ejecuta el comando *SHOW VARIABLES LIKE '%character_set%'* y la salida de éste indica que la base de datos está configurada con el conjunto de caracteres *UTF8*. Después se cambia la codificación de caracteres con el comando *ALTER DATABASE chem DEFAULT CHARACTER SET latin1*, pero hasta este punto sólo cambia la codificación para la base de datos, lo cual se controla con

la variable *character_set_database*. Para cambiar la codificación de la sesión en la base de datos se ejecuta el comando *SET CHARACTER SET latin1* y con esto se logra que las variables *character_set_client* y *character_set_connection*[46] cambien su valor a *latin1*.

De manera similar a lo anterior, Oracle también requiere de un procedimiento para cambiar los parámetros de la codificación de caracteres. En primer lugar se puede establecer la codificación de caracteres cuando se instala Oracle. Si los valores de los parámetros que configuran la codificación de caracteres no es la apropiada se pueden cambiar los valores posteriormente.

Oracle controla la codificación mediante una variable de ambiente de sistema operativo llamada *NLS_LANG*[47] y también mediante algunas variables de base de datos. En el primer caso, la variable NLS_LANG está conformada de tres partes. Estas son el lenguaje, el territorio y el juego de caracteres. La codificación que se utiliza para el soporte de acentos es *MEXICAN SPANISH_MEXICO.WE8MSWIN1252*[48] en donde MEXICAN es el territorio, SPANISH_MEXICO es el lenguaje y WE8MSWIN1252[49] es la codificación de caracteres.

Más aun, para configurar la codificación de caracteres mediante las variables de la base de datos se requiere ejecutar la instrucción *ALTER SYSTEM* para las variables *NLS_LANGUAGE*, *NLS_TERRITORY* y *NLS_CHARACTERSET*. Los valores de la ejecución del comando se muestran en el siguiente ejemplo.

```
ALTER SYSTEM SET NLS_LANGUAGE=Spanish;
ALTER SYSTEM SET NLS_TERRITORY=Mexico;
ALTER SYSTEM SET NLS_CHARACTERSET=WE8MSWIN1252;
```

[46] Estas variables son dinámicas. La primera de ellas codifica los comandos que se envíen del cliente al servidor de bases de datos. El segundo de ellos es para literales que no tienen una codificación de caracteres y para conversiones de números a cadenas de caracteres. Para más información visitar http://dev.mysql.com/doc/refman/5.0/en/server-system-variables.html#sysvar_character_set_client

[47] Cuando se instala Oracle, esta variable indica el conjunto de caracteres que debe tomar por default.

[48] Para ver más codificaciones de caracteres visitar http://docs.oracle.com/html/B10131_02/gblsupp.htm

[49] Esta codificación soporta latin1 para sistemas Windows. Se selecciono este valor ya que el cliente que se conecta a las bases de datos es Windows.

Cabe mencionar que para alterar los valores de estas variables se debe seguir un procedimiento. Este consiste en cerrar la base de datos, hacer las modificaciones que se muestran en la figura anterior y abrir la base de datos.

Finalmente, después de configurar la codificación de caracteres en MySQL y Oracle se procedió a la carga de los datos. Por un lado, de forma local se montó el millón 700 mil registros que tiene el CHEM. Por otro lado, en el servidor virtual se cargó más de 12 millones de registros en la base de datos del CHEM para ejemplificar y simular el comportamiento de esta base de datos pensando en que algún día puede llegar a contar con la misma cantidad de palabras.

A continuación describiré los criterios para hacer las pruebas en los RDBMS, después explicaré las pruebas como tal y posteriormente daré una interpretación de los resultados de las mismas.

5.2.4. Obtención de medidas

Para poder realizar las pruebas de comparación de los RDBMS fue necesario determinar dos aspectos importantes. El primero de ellos fue el número de pruebas que se iban a realizar junto con las palabras que serían buscadas. El segundo de ellos fue la medida en base a la cual se iba a medir la velocidad de los RDBMS.

En el caso de la selección de pruebas y palabras a buscar, se decidió buscar las diez palabras más frecuentes por cada periodo de 100 años. Como se explicó anteriormente, las búsquedas que realiza el CHEM se generan por periodos de este tamaño. Las palabras que fueron utilizadas para cada periodo se listan en la Tabla 5-1. Para el caso de las pruebas en el servidor virtual sólo existe un periodo de tiempo el cual va de 1925 a 1975. Para este caso se tomaron sólo las primeras 5 palabras de este periodo, éstas se muestran en la Tabla 5-2.

Por otro lado, las medidas que se tomaron para la realización de las pruebas estuvieron basadas en tiempo. Se decidió que se tomaría este último, ya que para ambas

etapas de pruebas es posible obtener dicha medida aunada a la realización de cada consulta.

En el caso de las pruebas desde la interfaz de consulta del CHEM, existe un archivo que almacena la bitácora de la ejecución de la búsqueda en el CHEM[50]. En el caso de las pruebas en el servidor, a nivel de la consola de cada RDBMS, existe un parámetro que muestra el tiempo que tardan las consultas en generar el resultado.

Para PostgreSQL es *TIMING*, para Oracle es *SET TIMING ON* y MySQL muestra por default el tiempo que tarda la consulta. Cabe mencionar que Oracle toma en cuenta el tiempo en que se muestra la salida de la consulta en pantalla. Para evitar lo anterior se define la variable SET AUTOTRACE TRACEONLY para evitar que se muestre la salida en pantalla mostrando sólo el tiempo que tarda en generarse la consulta.

[50] Este archivo guarda la hora de inicio y de fin, y los pasos del procedimiento de ejecución del CHEM. Cada que se busca una palabra, la aplicación del CHEM realiza diversas tareas y éstas son registradas en dicho archivo.

Tabla 5-1 Palabras de prueba para cada periodo de 100 años.

Periodo de tiempo	*Palabras*	*Numero de repeticiones*
1500 a 1600	y	2,924
	de	2,695
	que	2,554
	a	1,323
	el	1,265
	en	1,131
	dicho	1,065
	se	965
	la	853
	los	753
1600 a 1700	de	155
	que	139
	y	123
	a	60
	la	59
	el	55
	en	48
	se	37
	lo	32
	con	32
1700 a 1800	de	2,875
	que	2,115
	el	1,580
	y	1,525
	en	1,354
	la	1,313
	los	1,112
	se	736
	a	715
	del	635
1800 a 1900	de	1,653
	se	1,634
	y	1,438
	la	672
	en	644
	con	556
	que	549
	un	418
	el	384
	a	346

Tabla 5-2 Palabras de prueba en el servidor.

Periodo de tiempo	*Palabras*	*Numero de repeticiones*
1925 a 1975	de	114,753
	la	73,612
	que	71,751
	y	60,309
	el	54,900

Las tablas anteriores muestran las palabras por cada periodo de años y su número de repeticiones. Cabe resaltar que el periodo de tiempo que tiene más palabras es el de 1500 a 1600 con 51,283 palabras. Para este periodo de tiempo la palabra que más se repite es la “y”. Teniendo esta información se puede intuir que las pruebas que se realicen para esta palabra en este periodo de años pueden tomar más tiempo.

5.2.5. Velocidad actual del sistema

El punto de partida para realizar la comparación de los RDBMS es el tiempo que tarda actualmente la base de datos del CHEM para generar las concordancias. Para obtener el tiempo actual que toma el CHEM para generar las concordancias se llevó a cabo el siguiente procedimiento.

Como las pruebas se realizaron desde la interfaz del CHEM se buscó la palabra, en el primer caso la “y”, para el periodo de 1500 a 1600. Después se revisó el archivo bitácora que genera la aplicación del CHEM y de él se tomó el tiempo que tardó el RDBMS en regresar el resultado de la consulta. Así se tomaron las diez muestras para la primera palabra y se obtuvo un promedio del tiempo que tarda en generarse la consulta en la base de datos. Posteriormente se siguió el mismo procedimiento para el resto de las palabras.

Dicho lo anterior, la siguiente tabla muestra los promedios de búsqueda de PostgreSQL para obtener las concordancias. Es importante resaltar que los promedios que se mencionen a lo largo de esta sección están basados en segundos y milisegundos.

Tabla 5-3 Promedios de PostgreSQL para generar las concordancias.

Periodo de tiempo	*Palabras*	*Numero de repeticiones*	*Promedio de PostgreSQL*
1500 a 1600	y	2,924	0.648
	de	2,695	0.648
	que	2,554	0.507
	a	1,323	0.351
	el	1,265	0.338
	en	1,131	0.327
	dicho	1,065	0.319
	se	965	0.309
	la	853	0.316
	los	753	0.293
1600 a 1700	de	155	0.206
	que	139	0.189
	y	123	0.259
	a	60	0.179
	la	59	0.288
	el	55	0.199
	en	48	0.196
	se	37	0.192
	lo	32	0.188
	con	32	0.187
1700 a 1800	de	2,875	0.613
	que	2,115	0.499
	el	1,580	0.358
	y	1,525	0.447
	en	1,354	0.341
	la	1,313	0.339
	los	1,112	0.309
	se	736	0.272
	a	715	0.272
	del	635	0.267
1800 a 1900	de	1,653	0.393
	se	1,634	0.632
	y	1,438	0.393
	la	672	0.367
	en	644	0.249
	con	556	0.231
	que	549	0.288
	un	418	0.225
	el	384	0.212
	a	346	0.214

Cabe mencionar que aunque los promedios de PostgreSQL no rebasan el segundo, el tiempo puede ser aún menor al mostrado en la tabla anterior. Lo anterior podría ser

posible ya que en bases de datos los tiempos dependen de qué tan rápido sea el RDBMS y de las tareas de rendimiento que se apliquen para que lo sea.

También es importante resaltar el tiempo que tarda la palabra "y" para el periodo de 1500 a 1600, por ser éste el periodo que más palabras tiene. Aunque el promedio de esta palabra es de 648 milisegundos este tiempo podría sea menor en otro manejador o después de realizar alguna tarea de rendimiento.

Por otro lado, en el servidor virtual, el tiempo que tarda PostgreSQL para ejecutar la consulta se tomó desde la línea de comando de SQL. Estas pruebas consistieron en la ejecución de la consulta y en ir tomando el tiempo de respuesta del RDBMS. En el caso del servidor sólo se realizaron 5 pruebas para las palabras de la Tabla 5-2.

La siguiente tabla muestra el promedio de búsqueda que tarda PostgreSQL en generar las concordancias.

Tabla 5-4 Promedio de búsqueda de PostgreSQL en el servidor.

Palabra	***Numero de repeticiones***	***Promedio PostgreSQL***
de	114,753	14.368702
la	73,612	13.1715304
que	71,751	10.6987122
y	60,309	9.1148584
el	54,900	9.9195098

A diferencia de los promedios de las pruebas anteriores, el tiempo de respuesta de PostgreSQL sube hasta los 14 segundos con 36 milisegundos para la palabra de búsqueda "de". A continuación expondré los resultados de las mismas búsquedas en MySQL y Oracle.

5.3.Pruebas en MySQL

Posterior a obtener el tiempo de respuesta actual de la base de datos del CHEM con PostgreSQL, se obtuvo el tiempo promedio de respuesta de MySQL. La primera ronda de pruebas se realizó desde la interfaz del CHEM y los resultados de estas se muestran en la tabla siguiente.

Tabla 5-5 Promedios de MySQL en las búsquedas de la interfaz del CHEM.

Periodo de tiempo	*Palabras*	*Numero de repeticiones*	*Promedio MySQL*
1500 a 1600	y	2,924	1.529
	de	2,695	2.309
	que	2,554	1.364
	a	1,323	0.971
	el	1,265	0.931
	en	1,131	0.936
	dicho	1,065	0.925
	se	965	1.048
	la	853	1.065
	los	753	0.930
1600 a 1700	de	155	0.913
	que	139	0.824
	y	123	0.826
	a	60	0.811
	la	59	0.820
	el	55	0.817
	en	48	0.808
	se	37	0.893
	lo	32	0.801
	con	32	0.799
1700 a 1800	de	2,875	1.921
	que	2,115	1.677
	el	1,580	0.933
	y	1,525	1.703
	en	1,354	0.972
	la	1,313	1.006
	los	1,112	0.924
	se	736	0.910
	a	715	0.916
	del	635	0.946
1800 a 1900	de	1,653	1.355
	se	1,634	0.898
	y	1,438	1.365
	la	672	1.810
	en	644	0.869
	con	556	0.865
	que	549	1.336
	un	418	0.862
	el	384	0.860
	a	346	0.861

Como se puede observar en la Tabla 5-5 el promedio de MySQL para la palabra “y” en el primer periodo de 100 años toma un segundo y 529 milisegundos, a diferencia de PostgreSQL el cual obtuvo 648 milisegundos para la misma prueba. Con el fin de ilustrar

las diferencias de tiempo entre los RDBMS mencionados hasta este punto, la siguiente tabla muestra el promedio de respuesta de las primeras 3 palabras de cada periodo de 100 años.

Tabla 5-6 Comparación de tiempos de respuesta de PostgreSQL y MySQL en la interfaz del CHEM.

Periodo de tiempo	*Palabra*	*Promedio PostgreSQL*	*Promedio MySQL*
1500 a 1600	y	0.648	1.529
	de	0.648	2.309
	que	0.507	1.364
1600 a 1700	de	0.206	0.913
	que	0.189	0.824
	y	0.259	0.826
1700 a 1800	de	0.613	1.921
	que	0.499	1.677
	el	0.358	0.933
1800 a 1900	de	0.393	1.355
	se	0.632	0.898
	y	0.393	1.365

La tabla anterior muestra la palabra y los promedios de respuesta de PostgreSQL y MySQL por cada periodo de años. Cabe resaltar que todos los tiempos de MySQL son superiores a los de PostgreSQL, los cuales llegan a tomar hasta más de un segundo por cada búsqueda mientras que los tiempos de PostgreSQL toman poco más de medio segundo.

Las pruebas en el servidor virtual obtuvieron los siguientes promedios de búsqueda para las palabras presentadas en la Tabla 5-4.

Tabla 5-7 Comparación de tiempos de respuesta de PostgreSQL y MySQL en el servidor.

Palabra	*Promedio PostgreSQL*	*Promedio MySQL*
de	14.368702	48.776
la	13.1715304	42.982
que	10.6987122	37.884
y	9.1148584	33.654
el	9.9195098	37.892

Como podemos observar en los resultados anteriores, los tiempos promedio de respuesta de MySQL son superiores a los promedios de PostgreSQL. Tan superiores son que MySQL tarda más del triple que PostgreSQL para obtener las concordancias de la palabra “de”, con 48 segundos y 776 milisegundos.

5.4.Pruebas en Oracle

Posterior a la comparación con MySQL se realizaron las pruebas con Oracle desde la interfaz del CHEM y desde el servidor. Como ya mencioné anteriormente, las pruebas de la interfaz se llevaron a cabo con Oracle Express y las del servidor con Oracle 11g. La diferencia entre éstos es la capacidad de almacenamiento ya que Oracle Express cuenta con un tamaño limitado para el almacenamiento de datos. Lo anterior no afecta la realización de las pruebas. A continuación se muestran los promedios de Oracle Express.

Tabla 5-8 Promedios de Oracle en las búsquedas de la interfaz del CHEM.

Periodo de tiempo	*Palabras*	*Numero de repeticiones*	*Promedio Oracle*
1500 a 1600	y	2,924	0.246
	de	2,695	0.253
	que	2,554	0.178
	a	1,323	0.107
	el	1,265	0.101
	en	1,131	0.097
	dicho	1,065	0.095
	se	965	0.089
	la	853	0.107
	los	753	0.088
1600 a 1700	de	155	0.143
	que	139	0.054
	y	123	0.048
	a	60	0.044
	la	59	0.044
	el	55	0.047
	en	48	0.046
	se	37	0.043
	lo	32	0.047
	con	32	0.050
1700 a 1800	de	2,875	0.288
	que	2,115	0.209
	el	1,580	0.100
	y	1,525	0.202
	en	1,354	0.101
	la	1,313	0.225
	los	1,112	0.094
	se	736	0.085
	a	715	0.085
	del	635	0.088
1800 a 1900	de	1,653	0.188
	se	1,634	0.088
	y	1,438	0.143
	la	672	0.127
	en	644	0.070
	con	556	0.068
	que	549	0.128
	un	418	0.061
	el	384	0.063
	a	346	0.058

Es notorio que los promedios de Oracle, desde la interfaz del CHEM, están por debajo del tiempo promedio de PostgreSQL y de MySQL. Para la conjunción “y”, en el primer periodo de años, el promedio de Oracle es de menos de medio segundo (246

milisegundos). Además los promedios para el resto de las palabras también son menores. La siguiente tabla muestra algunas comparaciones de los tiempos actuales de la base de datos del CHEM (PostgreSQL) con los promedios de Oracle. Al igual que en la Tabla 5-6 se listan sólo los primeros tres resultados de cada periodo de años.

Tabla 5-9 Comparación de tiempos de respuesta de PostgreSQL y Oracle en la interfaz del CHEM.

Periodo de tiempo	*Palabra*	*Promedio PostgreSQL*	*Promedio Oracle*
1500 a 1600	y	0.648	0.246
	de	0.648	0.253
	que	0.507	0.178
1600 a 1700	de	0.206	0.143
	que	0.189	0.054
	y	0.259	0.048
1700 a 1800	de	0.613	0.288
	que	0.499	0.209
	el	0.358	0.100
1800 a 1900	de	0.393	0.188
	se	0.632	0.088
	y	0.393	0.143

Aunque los tiempos promedio de PostgreSQL parecen ser bajos, es muy notorio que los tiempos promedio de Oracle son aún menores. Estos últimos, en su mayoría, llegan a estar por debajo de la mitad del tiempo promedio que tarda PostgreSQL. Por ejemplo, para el periodo de 1700 a 1800 ambos manejadores se comportan de forma similar disminuyendo el tiempo de respuesta respecto a la cantidad de palabras. PostgreSQL tarda 613 milisegundos mientras Oracle tarda tan solo 288 milisegundos, es decir, menos de la mitad de lo que toma PostgreSQL. En el siguiente caso el primero genera las concordancias en 499 milisegundos y el segundo en 209 milisegundos. El último ejemplo muestra 358 milisegundos para PostgreSQL y 100 milisegundos para Oracle.

Más aún, las pruebas en el servidor virtual muestran resultados similares a los mostrados anteriormente. La Tabla 5-10 muestra los resultados de Oracle contra PostgreSQL.

Tabla 5-10 Comparación de tiempos de respuesta de PostgreSQL y Oracle en el servidor.

Palabra	*Promedio PostgreSQL*	*Promedio Oracle*
de	14.368702	5.19
la	13.1715304	4.53
que	10.6987122	3.942
y	9.1148584	3.576
el	9.9195098	3.962

Como se puede apreciar en la tabla anterior, Oracle tarda alrededor de 5 segundos para generar las concordancias de la preposición “de” mientras que el tiempo actual está en los 14 segundos. Para el segundo caso, “la” toma 13 segundos mientras que Oracle toma 4 segundos. Incluso los ejemplos posteriores, “que”, “y”, y “el” tienen el mismo comportamiento.

5.5. Análisis de resultados

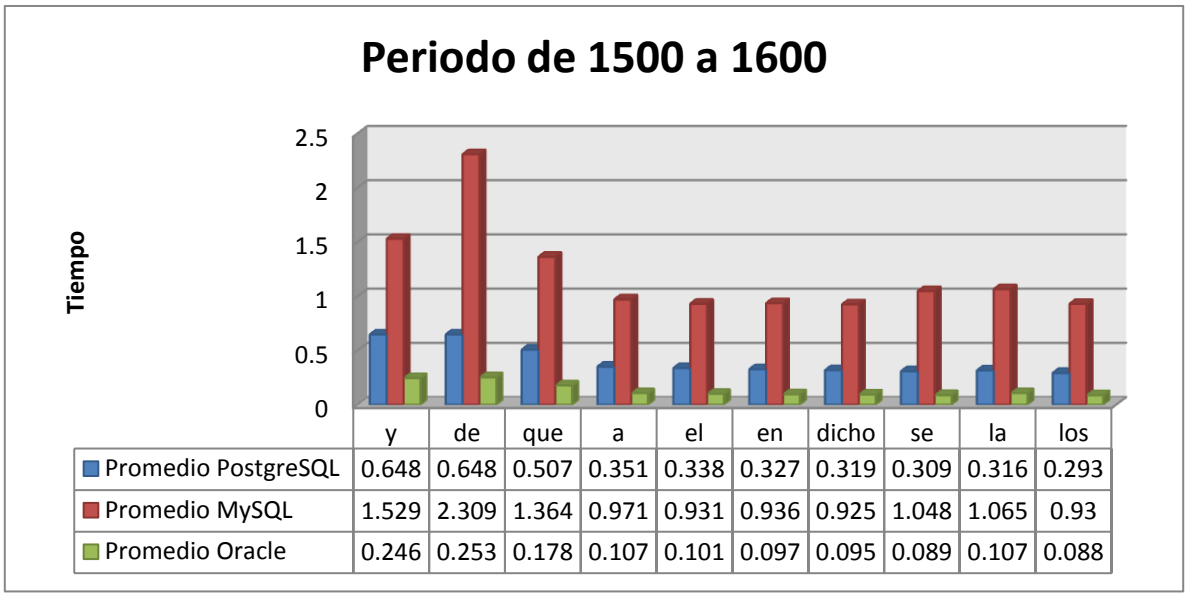
Para dejar en claro el comportamiento de los tres RDBMS se expondrán los tiempos promedio que éstos tardan en generar las concordancias. La siguiente tabla muestra los tiempos promedio obtenidos desde la interfaz del CHEM por periodos de años.

Tabla 5-11 Tiempos promedio de los tres RDBMS en la interfaz gráfica del CHEM.

Periodo de tiempo	*Palabra*	*Promedio PostgreSQL*	*Promedio MySQL*	*Promedio Oracle*
1500 a 1600	y	0.648	1.529	0.246
	de	0.648	2.309	0.253
	que	0.507	1.364	0.178
	a	0.351	0.971	0.107
	el	0.338	0.931	0.101
	en	0.327	0.936	0.097
	dicho	0.319	0.925	0.095
	se	0.309	1.048	0.089
	la	0.316	1.065	0.107
	los	0.293	0.93	0.088
1600 a 1700	de	0.206	0.913	0.143
	que	0.189	0.824	0.054
	y	0.259	0.826	0.048
	a	0.179	0.811	0.044
	la	0.288	0.82	0.044
	el	0.199	0.817	0.047
	en	0.196	0.808	0.046
	se	0.192	0.893	0.043
	lo	0.188	0.801	0.047
	con	0.187	0.799	0.05
1700 a 1800	de	0.613	1.921	0.288
	que	0.499	1.677	0.209
	el	0.358	0.933	0.1
	y	0.447	1.703	0.202
	en	0.341	0.972	0.101
	la	0.339	1.006	0.225
	los	0.309	0.924	0.094
	se	0.272	0.91	0.085
	a	0.272	0.916	0.085
	del	0.267	0.946	0.088
1800 a 1900	de	0.393	1.355	0.188
	se	0.632	0.898	0.088
	y	0.393	1.365	0.143
	la	0.367	1.81	0.127
	en	0.249	0.869	0.07
	con	0.231	0.865	0.068
	que	0.288	1.336	0.128
	un	0.225	0.862	0.061
	el	0.212	0.86	0.063
	a	0.214	0.861	0.058

El primer grupo de promedios de las pruebas de la Tabla 5-11 corresponde a la letra “y”, la cual es la más frecuente para el periodo que más palabras tiene. Como se puede

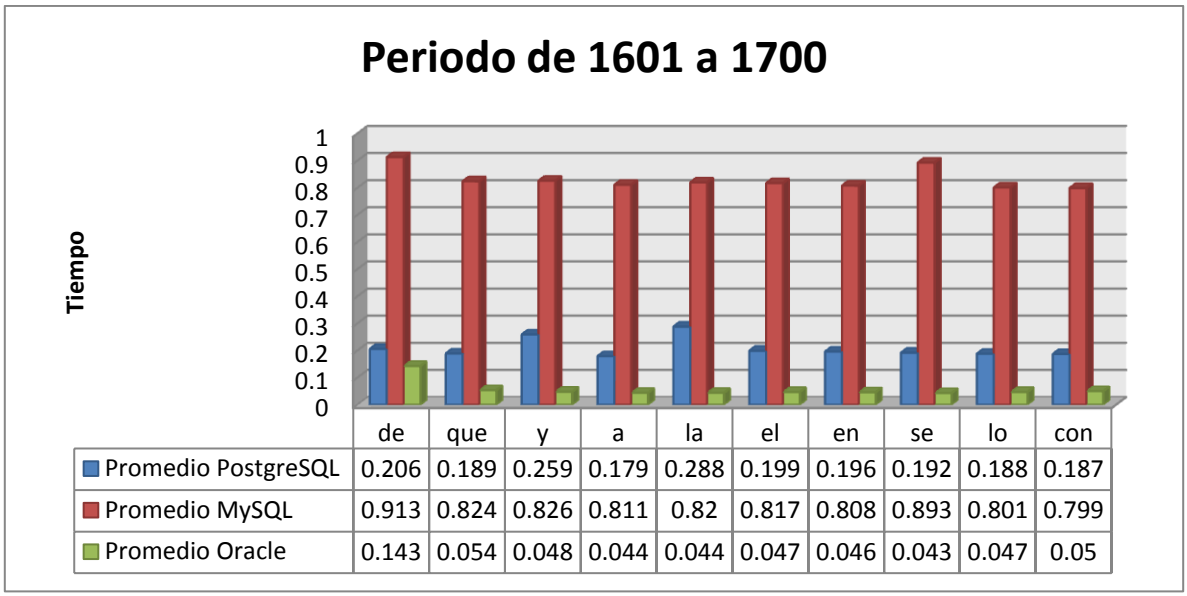
apreciar, Oracle obtiene el primer lugar en velocidad con un promedio de tiempo de 246 milisegundos seguido de PostgreSQL, el cual tiene 648 milisegundos, y finalmente MySQL, que tiene un segundo y 529 milisegundos. La siguiente gráfica muestra el comportamiento para el resto de las palabras de ese mismo periodo.



	y	de	que	a	el	en	dicho	se	la	los
Promedio PostgreSQL	0.648	0.648	0.507	0.351	0.338	0.327	0.319	0.309	0.316	0.293
Promedio MySQL	1.529	2.309	1.364	0.971	0.931	0.936	0.925	1.048	1.065	0.93
Promedio Oracle	0.246	0.253	0.178	0.107	0.101	0.097	0.095	0.089	0.107	0.088

Gráfica 5-1 Tiempos promedio de 1500 a 1600 de los tres RDBMS en la interfaz gráfica del CHEM.

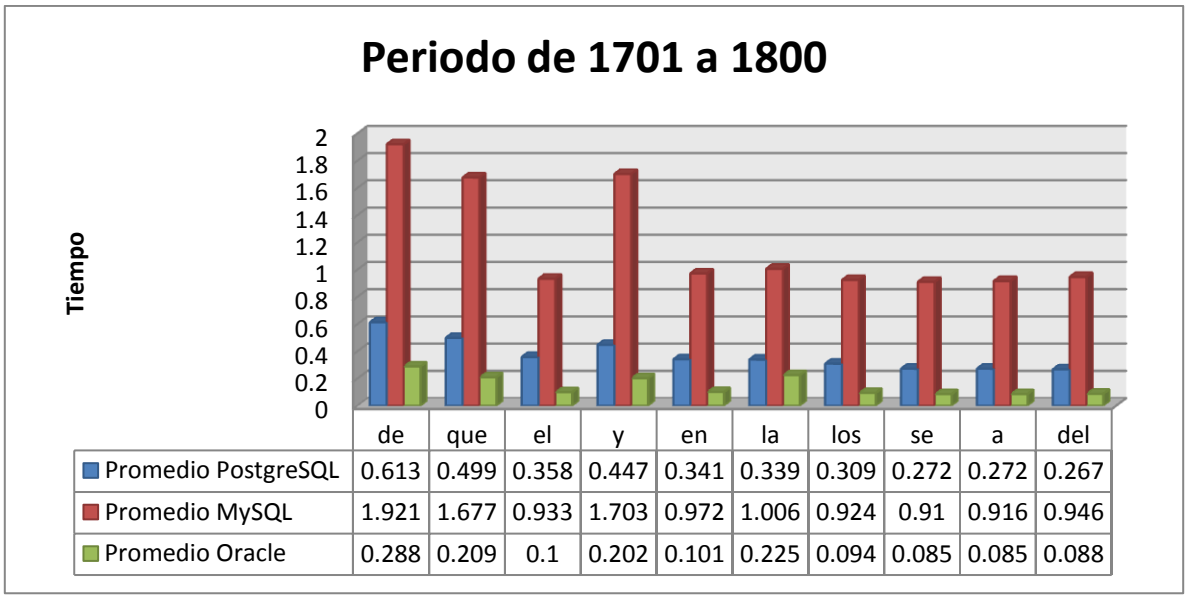
La grafica anterior nos muestra el promedio que tiene cada RDBMS por cada palabra para el primer periodo de años. Cabe resaltar que en todas las pruebas para este periodo de tiempo MySQL tarda más del doble que el tiempo actual de ejecución y más del triple del promedio de tiempo de Oracle. La siguiente grafica muestra los promedios de tiempo para el periodo de años de 1601 a 1700.



	de	que	y	a	la	el	en	se	lo	con
Promedio PostgreSQL	0.206	0.189	0.259	0.179	0.288	0.199	0.196	0.192	0.188	0.187
Promedio MySQL	0.913	0.824	0.826	0.811	0.82	0.817	0.808	0.893	0.801	0.799
Promedio Oracle	0.143	0.054	0.048	0.044	0.044	0.047	0.046	0.043	0.047	0.05

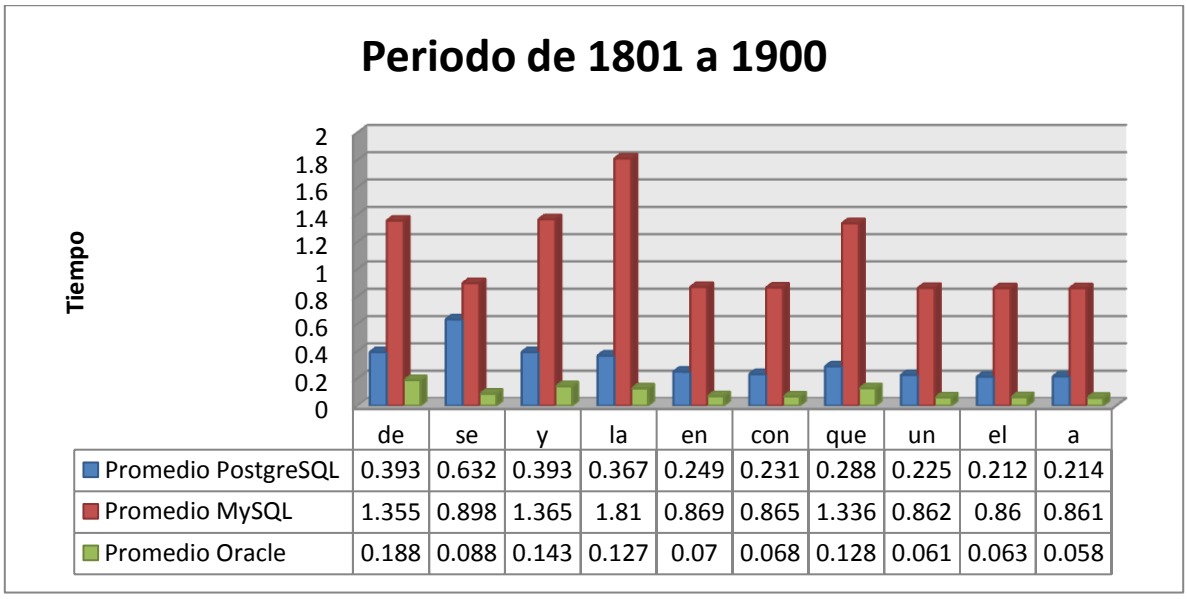
Gráfica 5-2 Tiempos promedio de 1601 a 1700.

La gráfica anterior muestra los promedios de tiempo para el periodo de años de 1601 a 1700, siendo éste el periodo que cuenta con la menor cantidad de palabras. Siguiendo con la tendencia de los resultados de la Gráfica 5-1 se puede notar como MySQL sigue tomando promedios de tiempos por encima de PostgreSQL y Oracle. Esta tendencia de promedios de MySQL no sólo se presenta en estos periodos de tiempo, sino en las gráficas siguientes, es decir, para los periodos restantes pasa lo mismo.



	de	que	el	y	en	la	los	se	a	del
Promedio PostgreSQL	0.613	0.499	0.358	0.447	0.341	0.339	0.309	0.272	0.272	0.267
Promedio MySQL	1.921	1.677	0.933	1.703	0.972	1.006	0.924	0.91	0.916	0.946
Promedio Oracle	0.288	0.209	0.1	0.202	0.101	0.225	0.094	0.085	0.085	0.088

Gráfica 5-3 Tiempos promedio de 1701 a 1800.



	de	se	y	la	en	con	que	un	el	a
Promedio PostgreSQL	0.393	0.632	0.393	0.367	0.249	0.231	0.288	0.225	0.212	0.214
Promedio MySQL	1.355	0.898	1.365	1.81	0.869	0.865	1.336	0.862	0.86	0.861
Promedio Oracle	0.188	0.088	0.143	0.127	0.07	0.068	0.128	0.061	0.063	0.058

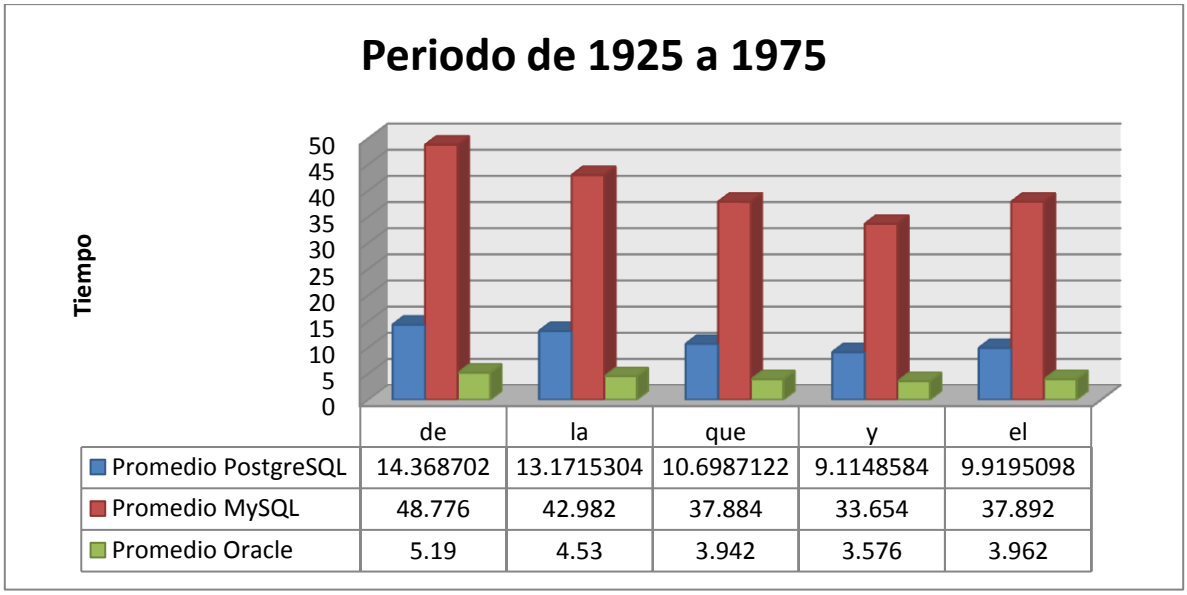
Gráfica 5-4 Tiempos promedio de 1801 a 1900

Finalmente, así como se puede ver que MySQL es el manejador que cuenta con los promedios de tiempo más altos, también resalta el hecho de que Oracle cuenta con los promedios más bajos para todas las pruebas por periodos de años. De esta forma podemos decir que el tiempo actual que toma la base de datos del CHEM para obtener las concordancias puede ser menor, mejorando de esta forma el tiempo de respuesta en la interfaz gráfica.

Por otro lado, las pruebas en el servidor arrojaron los siguientes promedios para los tres RDBMS. En la Gráfica 5-5 se puede ver la comparación de los tiempos promedio de cada uno.

Tabla 5-12 Promedios de los tres RDBMS en el servidor.

Palabra	*Promedio PostgreSQL*	*Promedio MySQL*	*Promedio Oracle*
de	14.368702	48.776	5.19
la	13.1715304	42.982	4.53
que	10.6987122	37.884	3.942
y	9.1148584	33.654	3.576
el	9.9195098	37.892	3.962



Gráfica 5-5 Tiempos promedio de los tres RDBMS en el servidor.

Semejante a las pruebas de la interfaz del CHEM, el comportamiento de MySQL tiene tiempos promedio sumamente altos, llegando a los 48 segundos con 776 milisegundos para la primera palabra de búsqueda, mientras que Oracle tiene solo 5 segundos con 19 milisegundos.

Si la base de datos del CHEM llegara a tener alrededor de doce millones de palabras, el tiempo promedio de ejecución de las consultas seria aproximadamente como se muestra en la Gráfica 5-5. Para la palabra de búsqueda “de” se obtendrían las concordancias en poco más de 14 segundos.

6. Administración de rendimiento en la base de datos del CHEM

Posterior a comparar el tiempo de respuesta de los RDBMS, se implementaron tareas de rendimiento de bases de datos para cada RDBMS. Como ya mencioné en el capítulo 4, existen diferentes técnicas de optimización que permiten mejorar el rendimiento de las bases de datos. Es importante decir que las pruebas de optimización fueron realizadas sólo en el servidor donde se realizaron las pruebas del capítulo anterior. Esto se decidió ya que las capacidades de hardware de este son mayores a las de una PC.

De las diferentes técnicas de optimización descritas en el capítulo 4, sólo se aplicaron algunas de ellas. Esto se debe a que no todos lo RDBMS soportan todas ellas. Las tareas de rendimiento que serán explicadas a continuación son la ubicación de archivos y el particionamiento, ya que los tres RDBMS soportan ambas técnicas de optimización.

6.1.Ubicación de archivos de los RDBMS

Esta tarea de rendimiento consiste en cambiar de ubicación los archivos donde se almacenan los índices de la base de datos. Para realizar esta prueba fue necesario crear una ubicación en el servidor de base de datos distinta a donde se almacenaron los archivos de datos de los diferentes RDBMS. Esta ubicación se creó en raíz bajo el nombre de BasedeDatos (/BasedeDatos).

Para realizar y evaluar esta tarea de rendimiento se tomó como punto de partida el tiempo actual de cada RDBMS en generar el resultado de la consulta. La Tabla 6-1 muestra los tiempos que tardan los RDBMS en ejecutar la consulta sin cambiar de ubicación los índices. Para realizar las pruebas se tomaron las 5 palabras más frecuentes y se realizaron 5 pruebas por cada palabra para obtener el tiempo de respuesta de los RDBMS.

Posteriormente se tomó el tiempo de respuesta con los índices en la nueva ubicación. Finalmente se crearon índices adicionales los cuales fueron creados con la intención de mejorar el tiempo de respuesta.

Tabla 6-1 Tiempo promedio de los RDBMS sin cambiar de ubicación los índices.

Palabra	*Repeticiones*	*PostgreSQL*	*MySQL*	*Oracle*
de	114,753	14.368702	48.776	5.19
la	73,612	13.1715304	42.982	4.53
que	71,751	10.6987122	37.884	3.942
y	60,309	9.1148584	33.654	3.576
el	54,900	9.9195098	37.892	3.962

Para crear las carpetas y archivos necesarios se creó dentro de la nueva ubicación una carpeta para cada RDBMS. Una vez que se contó con estas carpetas, se crearon los archivos que contendrían los índices de cada manejador de bases de datos.

En el caso de PostgreSQL se creó un espacio de tablas (*tablespace*) y después se crearon los índices dentro de él. El siguiente código muestra la creación de un *tablespace* y la asignación de índices dentro del mismo.

```
-- Creación de tablespace
CREATE TABLESPACE indices
LOCATION '/BasedeDatos/postgresql';

-- Creación de índices
CREATE INDEX indx1
    ON numeros
    USING btree (numero)
    TABLESPACE indices;

CREATE INDEX indx2
    ON numeros
    USING btree(otro_numero)
    TABLESPACE indices;
```

Como se puede ver en el código, en la sintaxis de la creación del *tablespace* se especifica el lugar donde queremos que sea almacenado este. Posteriormente se observa que se crean dos índices con un parámetro que indica el *tablespace* en donde se crearan. De esta forma obtenemos índices de tablas creados en una ruta diferente a donde están almacenados los datos.

En el caso de Oracle la creación de índices en otra ubicación es similar a la de PostgreSQL. De igual manera se debe crear un espacio de tablas en Oracle para así asignar

los nuevos índices a él. Para crear un *tablespace*, en lugar de tener el parámetro *location*, Oracle tiene uno denominado *datafile* donde se especifica la ruta, nombre y extensión del archivo. A diferencia de PostgreSQL, este manejador no asigna el nombre de los archivos de índices automáticamente. Para la creación de índices en Oracle, al igual que PostgreSQL, agrega un parámetro llamado *tablespace* el cual indica donde se almacenarán los índices. El código siguiente muestra un ejemplo de esto.

```
-- Creación de tablespace
CREATE TABLESPACE indices
DATAFILE '/BasedeDatos/oracle/indices.dbf'
SIZE 1024M;

-- Creación de índices
CREATE INDEX indx1
    ON numeros(numero)
    TABLESPACE indices;

CREATE INDEX indx2
    ON numeros(otro_numero)
    TABLESPACE indices;
```

Finalmente, en el caso de MySQL la creación de índices en una ubicación distinta es completamente diferente a las mencionadas anteriormente. MySQL maneja el almacenamiento de índices y datos mediante dos parámetros en la definición de la tabla. Estos son DATA DIRECTORY e INDEX DIRECTORY. El siguiente código muestra un ejemplo del uso para el parámetro INDEX DIRECTORY.

```
-- Creación de la tabla y uso del parámetro INDEX DIRECTORY
CREATE TABLE numeros (
    numero INTEGER,
    otro_numero INTEGER,
) ENGINE = MyISAM,
  INDEX DIRECTORY='/BasedeDatos/mysql/';

-- Creación de índices
CREATE INDEX indx1
    ON numeros(numero)
    USING btree;

CREATE INDEX indx2
    ON numeros(otro_numero)
    USING btree;
```

Como se puede ver en el código anterior, después de la definición de la tabla se definen dos parámetros. El primero de ellos es para indicar la arquitectura de la tabla. MySQL maneja dos arquitecturas para el almacenamiento de datos en la base de datos, una es MyISAM y la otra es InnoDB[51]. La razón por la que se asigna MyISAM en este ejemplo se debe a que permite asignar el parámetro INDEX DIRECTORY. Este último tiene como parámetro la ruta donde se almacenarán los índices. De esta forma los datos se guardarán en la carpeta por default de MySQL y los índices en la nueva ubicación. Posterior a la creación de la tabla fue necesario cargar los datos y crear los índices.

Una vez que los índices fueron creados en una ubicación diferente para los tres manejadores, se realizaron las pruebas correspondientes. Los resultados se pueden ver en la siguiente tabla.

Tabla 6-2 Tiempo promedio de los RDBMS después de cambiar los índices de ubicación.

Palabra	*Repeticiones*	*PostgreSQL*	*MySQL*	*Oracle*
de	114,753	14.558697	55.636	5.578
la	73,612	11.1945524	47.906	4.522
que	71,751	10.5842914	37.2	4.156
y	60,309	9.189378	33.026	3.57
el	54,900	9.559952	42.202	4.106

[51] Sus diferencias, características, ventajas y desventajas no serán mencionadas ya que no son relevantes para mi investigación.

Después de los experimentos realizados, se puede ver que los tiempos promedio de los RDBMS no disminuyen significativamente en ninguna prueba. Para dejar en claro este comportamiento la siguiente tabla muestra los tiempos promedio de cada manejador antes de cambiar los índices de ubicación y después de cambiarlos.

Tabla 6-3 Tiempos promedio de cada RDBMS antes y después de cambiar de ubicación los índices.

	PostgreSQL		***MySQL***		***Oracle***	
Palabra	***Antes***	***Después***	***Antes***	***Después***	***Antes***	***Después***
de	14.368702	14.558697	48.776	55.636	5.19	5.578
la	13.1715304	11.1945524	42.982	47.906	4.53	4.522
que	10.6987122	10.5842914	37.884	37.2	3.942	4.156
y	9.1148584	9.189378	33.654	33.026	3.576	3.57
el	9.9195098	9.559952	37.892	42.202	3.962	4.106

Aunque el resultado que se esperaba era una mejora en el tiempo que tardaban los RDBMS en ejecutar la consulta, esto no fue así ya que no se obtuvieron cambios significativos. Incluso los tiempos promedio en algunas pruebas llegan a tardar algunos milisegundos de más después de cambiar los índices de ubicación.

Por otro lado, con el fin de mejorar la velocidad de respuesta de los RDBMS se crearon índices adicionales en la base de datos. Estos fueron creados en los campos que utiliza la clausula WHERE de la consulta que genera las concordancias con la finalidad de mejorar el tiempo de respuesta de ésta. Después de agregar los índices se obtuvieron los resultados de la Tabla 6-4. Esta muestra los tiempos sin agregar índices adicionales y los tiempos después de agregar los índices.

Tabla 6-4 Tiempos promedio de los RDBMS después de crear nuevos índices.

	PostgreSQL		***MySQL***		***Oracle***	
Palabra	***Antes***	***Después***	***Antes***	***Después***	***Antes***	***Después***
de	14.368702	14.6669408	48.776	56.57	5.19	5.86
la	13.1715304	11.4569336	42.982	48.64	4.53	4.512
que	10.6987122	10.5180712	37.884	37.822	3.942	3.924
y	9.1148584	9.430787	33.654	32.766	3.576	3.506
el	9.9195098	9.7158334	37.892	41.908	3.962	4.176

Aunque la creación de índices mejora la velocidad de respuesta de los RDBMS esto no quiere decir que a mayor número de índices, mayor es la velocidad de los RDBMS. Los resultados anteriores nos demuestran que la base de datos en los tres RDBMS tampoco mejora su rendimiento en ejecutar la consulta con los índices adicionales.

6.2.Particionamiento

El particionamiento como ya expliqué en la sección 4.2.1.1 es otra técnica de optimización de bases de datos y consiste en dividir una tabla en varias piezas, permitiéndonos así tener una mayor distribución de los datos. El beneficio de hacer lo anterior es que el rendimiento de las consultas es mejor cuando los datos más consultados se encuentran en particiones específicas. También cuando las consultas o actualizaciones de datos acceden a una gran cantidad de registros en una partición, el rendimiento puede mejorar al tomar ventaja de escaneos secuenciales en lugar de utilizar un índice o lecturas de acceso aleatorio a lo largo de toda la tabla.

El particionamiento que soporta PostgreSQL es a través de herencia de tablas[52]. Cada partición debe ser creada como una tabla hija única de una tabla padre. Por lo regular, las particiones se crean vacías y pueden ser una representación de otra tabla o en ocasiones tienen atributos adicionales.

PostgreSQL soporta particionamiento por rango y por lista. El particionamiento por rango se refiere a que las particiones están definidas por un campo o un conjunto de estos sin superponer los valores de cada partición. Por ejemplo, si suponemos que un campo debe almacenar valores del uno al cien, podemos especificar que se almacenen todos los valores menores a cincuenta en una partición y los mayores en otra.

El particionamiento por lista se refiere a que se pueden crear particiones listando específicamente que valores se encontraran en cada una de ellas. Por ejemplo podemos especificar que se almacenen los valores tres, seis y nueve en una y los demás en otra.

[52] Para ver la información oficial sobre el particionamiento en PostgreSQL se puede visitar el siguiente enlace: http://www.postgresql.org/docs/8.4/static/ddl-partitioning.html

El procedimiento para implementar particionamiento en PostgreSQL es el siguiente. Primero se debe crear la tabla padre o tabla maestra de donde las particiones serán creadas. Esta tabla no debe contener datos y se recomienda no crear CONSTRAINTS de CHECK en ella. Esto se debe a que la definición de las particiones está basada en CONSTRAINT de CHECK, lo que quiere decir que si se crea uno de estos en la tabla padre se deberá asumir que existirá una partición para este valor.

Después se deben crear las particiones. Generalmente estas no tienen campos adicionales a la tabla padre y se deben declarar las restricciones para definir que valores almacenará cada una de ellas. Una vez creadas las particiones es posible agregarles índices. El código siguiente muestra un ejemplo de este procedimiento.

```
-- Definición de la tabla padre
CREATE TABLE numeros(
    numero INTEGER,
    otro_numero INTEGER
);

-- Definición de una partición por lista
CREATE TABLE particion_1(
    CHECK (numero IN (3,6,9))
) INHERITS (numeros);

-- Definición de una partición por rango
CREATE TABLE particion_2(
    CHECK (numero > 10)
) INHERITS (numeros);
```

El código anterior muestra la creación de una tabla llamada “numeros” y dos particiones. La primera partición es de tipo lista, donde se indica que se almacenarán los valores del campo “numero” que sean tres, seis y nueve. La partición siguiente está definida por rango y ésta almacenará todos los valores del campo “numero” que sean mayores a diez. Cabe mencionar que la cláusula INHERITS indica que hereda los campos de la tabla “numeros”.

Posterior a la definición de la tabla y sus particiones opcionalmente se puede crear una función y un *trigger* con el fin de que por cada inserción en la tabla padre se hagan las

inserciones adecuadas en la partición correspondiente[53]. El siguiente código muestra un ejemplo de la función y del *trigger* que insertarían en la tabla "numeros".

```
-- Función para insertar números
CREATE OR REPLACE FUNCTION fc_inserta_numeros()
RETURNS TRIGGER AS $$
BEGIN
    IF (NEW.numero IN (3,6,9)) THEN
        INSERT INTO particion_1 VALUES (NEW.*);
    ELSIF (NEW.numero > 10) THEN
        INSERT INTO particion_2 VALUES (NEW.*);
    END IF;
    RETURN NULL;
END;
$$
LANGUAGE plpgsql;

-- Trigger para insertar números
CREATE TRIGGER tg_inserta_numeros
    BEFORE INSERT ON numeros
    FOR EACH ROW EXECUTE PROCEDURE fc_inserta_numeros();
```

La función anterior evalúa si el valor del campo "numero" es tres, seis o nueve y si es así inserta los datos en la partición uno. Por otro lado, si los valores son mayores a diez los valores se insertan en la partición dos. El *trigger* se ejecuta cuando hay inserciones en la tabla "numeros" y ejecuta la función antes de insertar en dicha tabla, es decir, primero inserta los valores en las particiones y después en la tabla padre aunque la herencia de tablas en PostgreSQL haga lo contrario[54].

Cuando se desea utilizar particionamiento en PostgreSQL se puede indicar a éste que evalué las consultas para que pueda utilizar las particiones. Para lograr lo anterior el parámetro constraint_exclusion en el archivo de configuración *postgresql.conf* debe estar

[53] La herencia de tablas en PostgreSQL permite que las inserciones en las tablas hijas se hagan en la tabla padre. Definir una función y un *trigger* para el particionamiento es con la finalidad de que las inserciones se realicen en la tabla padre y no en las hijas como lo maneja PostgreSQL. Lo anterior no duplica los registros en la tabla padre después de insertar en las tablas hijas.

[54] Para ver herencia de tablas en PostgreSQL vistar http://www.postgresql.org/docs/8.4/static/ddl-inherit.html

habilitado con el fin de permitir que las consultas sean optimizadas por el DBMS. En caso de que éste parámetro no esté habilitado, una consulta a la tabla padre podría escanear todas sus particiones. Por el otro lado, si está habilitado, el DBMS examinará solo las particiones donde los valores de la cláusula *WHERE* coincidan.

El particionamiento en MySQL trabaja y se implementa de forma distinta a PostgreSQL. MySQL también soporta particionamiento por rango y por lista y adicionalmente tiene dos métodos más: por *HASH* y por llave (*HASH PARTITIONING* y *KEY PARTITIONING*). La característica principal de estos métodos es que MySQL asigna la distribución de los datos a lo largo de las particiones. La diferencia entre uno y otro es que en HASH se asigna un campo a partir del cual se crearán las particiones y en KEY no se define ningún campo para particionar, ya que éste particiona por llave primaria o por campos con valores únicos (restricción de valores únicos).

Otra característica del particionamiento en MySQL es que se puede implementar subparticionamiento[55], es decir, particionamiento a las particiones de una tabla padre. Si se desea implementar esta característica MySQL indica que cada partición debe tener el mismo número de subparticiones[56]. Además sólo es posible crear subparticionamiento por *HASH* o por *KEY*.

Para indicar a MySQL el particionamiento de las tablas se debe definir en su creación toda la definición de las particiones que sean necesarias. El código siguiente muestra un ejemplo de particionamiento por rango.

[55] PostgreSQL también soporta sub particionamiento. Para lograr esto se deben crear tablas que hagan referencia a una partición de la tabla padre. También se debe crear la función y el *trigger* para que funcione.
[56] Para mas información sobre el particionamiento en MySQL visitar http://dev.mysql.com/doc/refman/5.5/en/partitioning.html

```
-- Particionamiento por rango
CREATE TABLE numeros (
    numero INTEGER,
    otro_numero INTEGER
) PARTITION BY RANGE(numero) (
    PARTITION particion_1 VALUES LESS THAN (10),
    PARTITION particion_2 VALUES LESS THAN MAXVALUE
);
```

El código anterior crea la tabla "numeros" con dos campos: "numero" y "otro_numero". Después se indica que estará particionada por rango y se definen dos particiones (particion_1 y particion_2) a partir del campo "numero" donde la primera de ellas almacenara los valores menores a 10 y la segunda todos los valores mayores a diez. Lo anterior se logra con la cláusula *MAXVALUE* y esta sirve como comodín para evaluar valores fuera de los rangos que se especifiquen.

Para crear particiones por lista se deben asignar los valores individuales que almacenarán las particiones. Cabe resaltar que todos los valores a ser almacenados deben ser listados ya que no es posible usar la cláusula *MAXVALUE* para almacenar los valores que no se listen. El código siguiente muestra un ejemplo de particionamiento por lista y subparticionamiento por *HASH*.

```
-- Particionamiento por lista y por hash
CREATE TABLE numeros (
    numero INTEGER,
    otro_numero INTEGER
) PARTITION BY LIST(numero)
    SUBPARTITION BY HASH(otro_numero)
    SUBPARTITIONS 2(
    PARTITION particion_1 VALUES IN (1,2,3,4,5)
        (SUBPARTITION sub_1,SUBPARTITION sub_2),
    PARTITION particion_2 VALUES IN (6,7,8,9,10)
        (SUBPARTITION sub_3,SUBPARTITION sub_4),
    PARTITION particion_3 VALUES IN (11,12,13,14,15)
        (SUBPARTITION sub_5,SUBPARTITION sub_6),
    PARTITION particion_4 VALUES IN (16,17,18,19,20)
        (SUBPARTITION sub_7,SUBPARTITION sub_8)
);
```

Como se puede observar en el código, se define particionamiento por lista y en seguida subparticionamiento por *HASH*. Adicional se especifica que se crearán dos subparticiones por cada partición y éstas dependerán de los valores que almacene el campo "otro_numero". Finalmente el código siguiente muestra un ejemplo de particionamiento por llave.

```
-- Particionamiento por llave
CREATE TABLE numeros (
    numero INTEGER PRIMARY KEY,
    otro_numero INTEGER
) PARTITION BY KEY()
PARTITIONS 2;
```

En este caso se crea la tabla "numeros" especificando que el campo "numero" es llave primaria. En seguida se indica que la tabla estará particionada por llave y que el número de particiones serán dos. Cabe resaltar que si la tabla no tuviera definida una llave primaria o una restricción de valores únicos no sería posible crearla ya que MySQL indicaría que existe un error de sintaxis.

Posterior a la creación de las tablas y sus particiones es recomendable crear índices en los campos que fueron utilizados para dividir la tabla.

Finalmente Oracle soporta particionamiento por rango, lista y por *HASH* y es posible tener subparticionamiento por cualquiera de éstos tipos[57]. El código siguiente muestra un ejemplo de particionamiento por lista.

[57] Para más información sobre el particionamiento en Oracle visitar http://docs.oracle.com/cd/E18283_01/server.112/e16541/part_admin001.htm

```
-- Particionamiento por lista
CREATE TABLE numeros(
    numero INTEGER,
    otro_numero INTEGER
) PARTITION BY LIST (numero)(
    PARTITION particion_1 VALUES (1,2,3,4,5,6,7,8,9,10),
    PARTITION particion_2 VALUES (11,12,13,14,15),
    PARTITION particion_3 VALUES (NULL),
    PARTITION particion_4 VALUES (DEFAULT)
);
```

En Oracle se pueden definir particiones para almacenar valores nulos y particiones para almacenar valores que no estén dentro de la lista usando la cláusula *DEFAULT*[58]. Por otro lado se puede definir subparticionamiento por lista, rango o *HASH*. El código siguiente muestra un ejemplo de subparticionamiento por rango.

```
-- Particionamiento por lista
-- Subparticionamiento por rango
CREATE TABLE numeros(
    numero INTEGER,
    otro_numero INTEGER
) PARTITION BY LIST(numero)
    SUBPARTITION BY RANGE(otro_numero)(
    PARTITION particion_1 VALUES (3,6,9,12)
        (SUBPARTITION sub_1 VALUES LESS THAN(100),
         SUBPARTITION sub_2 VALUES LESS THAN(MAXVALUE)),
    PARTITION particion_2 VALUES (5,10,15)
        (SUBPARTITION sub_3 VALUES LESS THAN(200),
         SUBPARTITION sub_4 VALUES LESS THAN(MAXVALUE)),
    PARTITION particion_3 VALUES (DEFAULT)
);
```

El código anterior muestra tres particiones y dos subparticiones para las primeras dos particiones. La primera partición almacenará los valores donde el campo “numero” sea 3, 6, 9 y 12 y su primera subpartición los valores donde el campo “otro_numero” sean menores a cien. La segunda subpartición almacenara los valores mayores a cien. El mismo

[58] Su uso es idéntico a la cláusula *MAXVALUE* de MySQL.

caso se repite para las subparticiones de la segunda partición pero con sus respectivos valores.

Es importante mencionar que el manejo de particionamiento en MySQL y Oracle ya está programado en ambos manejadores mientras que en PostgreSQL debe ser programado por uno mismo mediante una función y un *trigger*.

Ahora hablaré de los experimentos realizados. El particionamiento que se implementó en la base de datos del CHEM fue por lista, es decir, se crearon particiones por tipo de palabra. Posteriormente se implementó sub particionamiento y este fue por rango de años. De esta forma tendríamos particiones por tipo de palabra y para cada una de esas particiones se tendrían subparticiones por años. La tabla siguiente ejemplifica la distribución de los datos.

Tabla 6-5 Distribución de los datos del CHEM.

Partición	***Sub partición 1***	***Sub partición 2***
Tipo de palabra normalizada	*De 1925 a 1950*	*De 1951 a 1975*
Tipo de palabra ortográfica	*De 1925 a 1950*	*De 1951 a 1975*
Tipo de palabra por lema	*De 1925 a 1950*	*De 1951 a 1975*

Después de implementar el particionamiento en los tres manejadores se realizaron las pruebas de velocidad de respuesta. Primero se decidió tomar el tiempo consultando a todo el periodo de años (de 1925 a 1975) para un solo tipo de palabra (normalizada). Esto equivale a consultar solo en una partición, aunque en todas sus subparticiones. Por otro lado se hizo la consulta a la tabla sin particionamiento para poder comparar cual era más rápida.

Posteriormente se hicieron pruebas en consultar solo un rango de años, esto es, consultar en una sola de las subparticiones. Estas pruebas se realizaron con la finalidad de hacer una emulación de lo que pasaría si se buscaran palabras que sólo están presentes en un cierto periodo de años. Para este caso también se consultó la tabla sin implementar particionamiento para ver cuál de los dos casos era más rápido.

Los resultados de las primeras pruebas, consultando todo el rango de años, se muestran en la siguiente tabla.

Tabla 6-6 Tiempos promedio de los RDBMS de la consulta en todo el rango de años.

		PostgreSQL		***MySQL***		***Oracle***	
Palabra	***Repeticiones***	***Sin partición***	***Con partición***	***Sin partición***	***Con partición***	***Sin partición***	***Con partición***
de	114,753	14.368702	57.0313094	48.776	40.126	5.19	3.028
la	73,612	13.1715304	55.547621	42.982	33.588	4.53	2.208
que	71,751	10.6987122	54.8646824	37.884	33.532	3.942	2.15
y	60,309	9.1148584	53.1393182	33.654	31.238	3.576	1.794
el	54,900	9.9195098	54.082134	37.892	29.638	3.962	1.676

Como se puede observar el tiempo promedio de PostgreSQL en todas las pruebas es muy superior si se implementa particionamiento. Aunque se supone que es una técnica de optimización, para este RDBMS no es conveniente implementarla. Por otro lado, MySQL sí reduce sus tiempos promedio de respuesta, aunque sigue siendo mucho el tiempo que tarda en realizar la consulta, es decir, hubo mejora, pero no demasiada. Por ultimo Oracle también pudo reducir sus tiempos promedio logrando pasar, por ejemplo, de 5.19 segundos a 3.028 segundos en la primera prueba.

También se tomaron los tiempos promedio de la segunda etapa de pruebas. Estas consistieron en buscar las palabras en un rango de años de 1951 a 1975. La siguiente tabla muestra los resultados de dichas pruebas.

Tabla 6-7 Tiempos promedio de los RDBMS de 1951 a 1975.

		PostgreSQL		***MySQL***		***Oracle***	
Palabra	***Repeticiones***	***Sin partición***	***Con partición***	***Sin partición***	***Con partición***	***Sin partición***	***Con partición***
de	111,102	13.9704782	7.1586226	55.686	39.098	5.446	2.762
la	71,418	11.103154	4.8868774	49.922	32.378	4.478	2.026
que	69,781	10.0419206	4.7815232	44.04	32.604	3.976	1.99
y	58,232	8.8911308	3.9903698	40.554	29.66	3.596	1.764
el	52,902	9.6570698	3.7689436	44.222	29.278	4.044	1.69

Como se puede observar en los resultados, todos los manejadores bajaron su tiempo promedio de respuesta al consultar palabras que se encuentren en una sola subpartición. Lo anterior se puede deber a que los RDBMS no realizan un escaneo completo de todos

los datos en la tabla sino solo en aquellos que están en la cláusula WHERE, o lo que es igual, los que están en la subpartición.

7. Conclusiones

Para finalizar esta investigación, en este capítulo presentaré las conclusiones finales. Antes de llegar a lo anterior expondré un breve resumen de los capítulos de esta tesis, revisaré los objetivos e hipótesis que se plantearon, los problemas a los que me enfrenté en esta investigación así como las ventajas de implementar la experimentación que aquí se presentó y finalmente describiré el posible trabajo futuro a consecuencia de esta investigación.

En el capítulo uno se expuso la introducción de esta investigación, donde se mencionaron los antecedentes y problemas a ser resueltos. Después se expusieron los objetivos e hipótesis de esta tesis para indicar qué rumbo tendría, así como su alcance y la metodología que sería implementada para su realización. En ese capítulo se definió que se realizaría una comparación de manejadores de bases de datos y la implementación de tareas de rendimiento a las consultas de base de datos.

En el capítulo dos se presentó el Corpus Histórico del Español en México, es decir, su arquitectura, componentes, su funcionamiento y sus herramientas. Dentro de los componentes se expusieron el indexador de documentos y el generador de concordancias. En las herramientas se presentaron las formas de búsqueda que soporta y las estadísticas de asociación de palabras. También se mencionaron otros corpus del GIL.

En el capítulo tres se presentó un resumen sobre el tema de bases de datos. Este resumen contiene la definición de una base de datos y de un manejador de bases de datos. Posteriormente se expuso el modelo entidad relación, los objetos que puede contener una base de datos a nivel lógico y el lenguaje SQL para el manejo de éstos. Por último el capítulo finaliza con una explicación de consultas SQL para obtener información de las bases de datos.

El capítulo cuatro explica los aspectos más importantes (para esta tesis) de la administración de rendimiento de bases de datos. Los temas que se describen en este capítulo son el rendimiento de sistema operativo y rendimiento de bases de datos. Se explica primero el rendimiento del sistema operativo ya que las bases de datos residen en

él. Dentro del rendimiento de bases de datos se exponen las técnicas de optimización para bases de datos.

El capítulo cinco muestra la comparación de manejadores de bases de datos. Primero se presenta una breve descripción de los manejadores que fueron utilizados. Después se explica la estrategia de evaluación donde se definen los parámetros que indicarían cuál es más rápido. Luego se detallan los ajustes que fueron necesarios para cambiar la base de datos del manejador actual a MySQL y Oracle. Posterior a determinar lo anterior se obtuvo el tiempo actual de respuesta de la base de datos del CHEM para después realizar las pruebas de velocidad contra MySQL y Oracle. Finalmente, se presentó un análisis de resultados de los tiempos obtenidos por cada uno de ellos.

Por último el capítulo seis contiene la implementación de tareas de rendimiento de bases de datos en los tres RDBMS. Las tareas que se presentan en este capítulo son la ubicación de archivos y el particionamiento. Para cada tarea de rendimiento primero se expuso la forma en que se implementa en cada RDBMS para después comparar si el rendimiento de la base de datos mejora. Esto se determinó por medio del tiempo de respuesta de la consulta del CHEM en cada manejador antes y después de implementar las tareas de rendimiento.

Como ya mencioné esta tesis estuvo basada en dos rubros: el primero fue una comparación de manejadores de bases de datos mediante la comparación del tiempo promedio de respuesta que tarda la consulta que genera concordancias en la base de datos del CHEM y el segundo fue la implementación de tareas de rendimiento en los RDBMS que fueron seleccionados.

Esta consulta que genera concordancias se ejecutó a partir de una palabra de petición. Específicamente esta consulta obtiene la palabra de búsqueda junto con su identificador y su posición, el identificador del documento de donde se obtuvo la palabra, el año del documento, la posición de la décima palabra anterior a la de búsqueda y la posición de la décima palabra posterior a la palabra de búsqueda.

Para la comparación de manejadores de bases de datos se realizaron dos fases de experimentos. El primero consistió en tomar los tiempos promedio de respuesta ejecutando la consulta desde la interfaz gráfica del CHEM en una PC con PostgreSQl, MySQL y Oracle. La segunda ronda de pruebas consistió en ejecutar la consulta desde la línea de comando SQL de cada RDBMS en un servidor virtual con una cantidad de registros mayor a los de la primera prueba. Los resultados de la primera ronda de experimentos demostraron que de los tres RDBMS el más rápido fue Oracle seguido de PostgreSQL y al último MySQL. Oracle logro ejecutar la consulta en dos veces menos de tiempo de lo que tarda PostgreSQL.

Los resultados de las pruebas en el servidor virtual de igual forma a los resultados mencionados anteriormente determinaron que el más rápido de ellos fue Oracle seguido de PostgreSQL y finalmente MySQL.

Por otro lado, para la implementación de tareas de rendimiento se realizaron dos fases de experimentos en el servidor virtual. La primera de ellas, descrita en la sección 4.2.1.9, fue la ubicación de archivos y esta consistió en separar los índices a una nueva ubicación. Después de realizar esta tarea de rendimiento los resultados indicaron que para los tres manejadores el tiempo promedio de respuesta a la consulta aumento algunos milisegundos por lo que no hubo ninguna mejora.

Otro de los experimentos como parte de esta técnica de optimización fue agregar índices adicionales. Estos fueron creados en los campos de la cláusula WHERE de la consulta con la finalidad de mejorar el rendimiento de la base de datos. Los resultados de estas pruebas al igual que las pruebas anteriores mostraron que los tiempos promedio de respuesta en los tres RDBMS aumentaron en milisegundos.

Como se pudo apreciar, en ninguno de los tres manejadores bajo el tiempo de respuesta al cambiar de ubicación los índices y al agregar nuevos índices, y de hecho el tiempo promedio de respuesta de los RDBMS subió algunos milisegundos. Esto pudo haber sido causado por la arquitectura del servidor ya que cuenta con una configuración RAID-5 lo que provoca que el mismo sistema distribuya los datos en los discos.

La segunda de ellas, descrita en la sección 4.2.1.1, fue el particionamiento y consistió en distribuir los datos de las tablas en segmentos más pequeños. El criterio que se implementó para implementar esta técnica fue particionar por tipo de palabra y después cada partición se dividió por rango de años. Posteriormente se hicieron dos pruebas: la primera para consultar todos los años y la segunda para consultar la mitad del rango de años.

Los resultados de la primera ronda de pruebas indicaron que MySQL y en Oracle el tiempo de respuesta de la consulta mejoro mientras que en PostgreSQL incremento. Los resultados de la segunda ronda de pruebas indicaron que los tres manejadores redujeron su tiempo promedio de respuesta.

Uno de los objetivos que se plantearon para esta investigación fue dar una alternativa de solución que ayudara a optimizar el tiempo de respuesta de la base de datos del CHEM. Esto se lograría a través de una comparación de manejadores de bases de datos. Dicha comparación se llevó a cabo y se obtuvieron resultados positivos ya que Oracle resulto ser más rápido que MySQL y PostgreSQL.

También se intentaría demostrar que las técnicas de optimización de bases de datos mejorarían el rendimiento de la base de datos del CHEM. Con la implementación de las técnicas de rendimiento que en esta investigación se plantearon, se obtuvo que para el ambiente actual, es decir con PostgreSQL, el rendimiento no mejora mientras que con MySQL y Oracle sucede lo contrario sólo en el caso del particionamiento.

Se propusieron algunas preguntas que esta tesis resolvería al final de la experimentación. En seguida comento cada una de ellas.

- ¿Qué técnicas de rendimiento existen en bases de datos para mejorar el rendimiento? Existen diversas técnicas de rendimiento de bases de datos. Estas son particionamiento, sistema de archivos o dispositivos en crudo, indexación, desnormalización, clustering, intercalado de datos, espacio vacío, compresión,

ubicación de archivos, tamaño de paginado y reorganización. Todas ellas son descritas en el capítulo 4.

- ¿Es posible mejorar el rendimiento de la base de datos del CHEM usando *tuning* de base de datos? Los resultados de esta investigación, para el particionamiento, indican que Oracle y MySQL si mejoran su rendimiento. Por otro lado la ubicación de los archivos de índices indicaron que no[59] para los tres manejadores de bases de datos.
- ¿Será posible que haya un manejador de bases de datos que se adapte mejor a las demandas del CHEM en cuestión de velocidad? Después de comparar los tres RDBMS resultó que Oracle es más rápido que PostgreSQL, reduciendo el tiempo actual de respuesta, lo que quiere decir que la base de datos puede que tenga un mejor desempeño en este manejador.

Las hipótesis que se plantearon fueron las siguientes:

- Existe un manejador de bases de datos que responde más rápido a las consultas del CHEM.
- MySQL es el manejador de bases de datos que responde más rápido a las consultas del CHEM.
- Existe una optimización de la base de datos que mejora la velocidad de respuesta de las consultas del CHEM.

Acepto la primera hipótesis ya que efectivamente hay un manejador de bases de datos que responde más rápido a las consultas del CHEM y este es Oracle. La segunda hipótesis no se puede aceptar ya que de los tres manejadores que se compararon MySQL resulto ser el más lento. La tercera hipótesis se acepta parcialmente ya que en MySQL y Oracle el particionamiento obtuvo resultados que indicaron que esta técnica sí mejora el rendimiento de las consultas. Además, para PostgreSQL, de las fases de experimentos sólo

[59] Aunque en esta investigación los resultados para esta prueba fueron negativos, en mi experiencia laboral he comprobado que sí mejora el tiempo de respuesta de las consultas con esta técnica de optimización. La diferencia es que en otros ambientes de bases de datos he podido separa los archivos de índices de los archivos de datos en diferentes discos y aquí no fue posible.

se obtuvieron resultados positivos en la última de ellas, es decir, cuando se hizo particionamiento por un rango de años.

A lo largo de la experimentación para la comparación de manejadores de bases de datos se presentaron diversos problemas. El primero de ellos fue la forma en la que se tomó el tiempo de respuesta desde la consola SQL de los manejadores. Mientras que PostgreSQL y MySQL pueden dar el tiempo en que tardan en hacer una consulta, Oracle toma en cuenta el tiempo que tarda en imprimir en pantalla los resultados. Esto llevó a investigar la forma en que sólo se obtuviera el tiempo de la consulta.

Otro de los problemas se presentó cuando se implementó la primera técnica de optimización en MySQL. Para cambiar de ubicación los índices fue necesario borrar la tabla y volver a crearla con sus respectivos directorios para posteriormente cargar los datos. En este caso el problema fue el retraso en volver a cargar los más de doce millones de registros.

Otro problema que más tomó tiempo en ser resuelto fue la migración de la base de datos del CHEM a MySQL y Oracle. Primero se tuvo que investigar la sintaxis adicional que requerían los objetos de la base de datos para poder ser creados. Una vez que la estructura de la base de datos se cambió a los diferentes manejadores se presentaron problemas para cargar los datos. Esto se debió al uso de acentos en las palabras almacenadas por lo que fue necesario investigar y cambiar el juego de caracteres de los manejadores.

Por otro lado, a lo largo de esta investigación se presentaron situaciones que motivan a pensar en trabajo futuro, como la realización de experimentos adicionales. Uno de ellos sería cambiar la configuración del servidor de forma que se tengan dos sistemas RAID 5 y no uno sólo. De esta forma se podrían cargar los datos en uno de ellos y en el otro cargar los índices para así asegurar que el uso de los disco es diferente.

Otro aspecto sería la implementación de más técnicas de rendimiento a la base de datos del CHEM. Por ejemplo, en PostgreSQL y en Oracle es posible implementar *clusterig* de tablas con la idea de agilizar las búsquedas (mencionada en la sección 4.2.1.5).

También sería factible realizar optimización de la consulta a nivel de código SQL. Esta tarea aunque parece sencilla requiere de un análisis amplio para hacer que sea más rápida.

Finalmente, después de realizar esta investigación y de acuerdo con los resultados de las pruebas puedo concluir lo siguiente:

En la comparación de los RDBMS que fueron utilizados en esta tesis Oracle fue el más rápido por lo que recomiendo al Grupo de Ingeniería Lingüística que adopte este manejador para la base de datos del CHEM. Cabe mencionar que aparte de ser el más rápido, logró obtener un tiempo promedio de respuesta hasta tres veces menor que el tiempo promedio actual de la base de datos.

Para la implementación de técnicas de rendimiento, en la ubicación de archivos de índices, no es conveniente crear los índices en una carpeta diferente ya que el uso del RAID-5 como configuración de discos duros no nos asegura que se estén almacenando en discos diferentes. Tampoco parece conveniente agregar más índices a la base de datos ya que los índices que probé tampoco redujeron el tiempo de respuesta de la base de datos.

Respecto al particionamiento no es conveniente implementar esta técnica para PostgreSQL. En caso de que la base de datos se cambie a Oracle, implementar esta técnica de rendimiento reduciría el tiempo de respuesta de la consulta que genera las concordancias.

Para terminar éste capítulo y ésta tesis espero que los conocimientos mostrados aquí sean de utilidad para el Grupo de Ingeniería Lingüística y para aquellas personas que se interesen por la velocidad de respuesta de los RDBMS expuestos aquí así como por la administración de rendimiento de bases de datos. En lo personal ésta tesis me permito ampliar mis conocimientos en bases de datos, me ayudo a mejorar la manera de investigar

acerca de temas que son de interés para mi vida profesional y más que nada ha servido de motivación para superarme día con día.

8. Referencias

8.1.Bibliográficas

- Elmasri, R. y Navathe S. (2007) *Fundamentos de sistemas de bases de datos*, 5ª ed., México, Pearson Educación, Addison-Wesley.
- Adoración de Miguel Castaño, Mario Piattini Vethuis y Esperanza Marcos Martínez (2000) *Diseño de bases de datos relacionales*, México, Alfa Omega.
- Adoración de Miguel Castaño y Mario Piattini Vethuis, (1999) *Fundamentos y modelos de bases de datos*, 2ª ed., México, Alfa Omega.
- Peter Rob y Carlos Coronel (2001) *Database Systems: design, implementation and management*, 8ª ed., Estados Unidos, Thomson Course Technology.
- Catherine M. Ricardo (2009), *Bases de datos*, México, Mc Graw Hill.
- Douglas Biber, Susan Conrad y Randi Reppen. (1998). *Corpus linguistics. Investigating language structure and use*, Reino Unido, Cambridge Uneversity Press.
- Michael P. Oakes. (1998). *Statisticas for corpus linguistics*, Reino Unido, Edinburgh Uneversity Press.
- Craig S. Mullins, (2002), Database administration: the complete guide to practices and procedures, Estados Unidos, Addison-Wesley Professional.

8.2.Web

- Medina Urrea, Alfonso y Méndez Cruz, Carlos Francisco. (2006). El Corpus Histórico del Español en México. revista.unam.mx, núm. 7. Recuperado el 16 de agosto de 2012, de http://www.revista.unam.mx/vol.12/num7/art64/index.html
- Oracle. (2005a). CREATE PROCEDURE [en línea]. Oracle® Database SQL Reference 10g Release 2 (10.2). Recuperado el 31 de enero de 2013 de http://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_6009.htm.