
# UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ACATLÁN

## ANÁLISIS DE DESEMPEÑO EN MANEJADORES DE BASE DE DATOS RELACIONALES: METODOLOGÍA

TESIS

PARA OBTENER EL TITULO DE

LICENCIADO EN MATEMÁTICAS APLICADAS Y COMPUTACIÓN

PRESENTA

ISMAEL MEDINA MUÑOZ

Asesor: M. en C. Javier Rosas Hernández

Fecha: Septiembre de 2012

***DEDICATORIA***

*A Jesucristo, mi señor y salvador.*

*A mi esposa y a mis hijos.*

*A mis padres y a mi hermano.*

*A mis padrinos.*

***AGRADECIMIENTOS***

*A mis padres Ismael y Velina, a ella por ser ejemplo de fortaleza y entrega, a él por ser mi sabio en la cima de la montaña.*

*A mi amada esposa Eugenia, por su amorosa compañía al mando de nuestro barco. Por regalarme la bendición de ser padre.*

*A mis hijos Getsemaní, Ismael y Mateo, por ser mi motivo todos los días. Su existencia es mi mayor logro.*

*A mi hermano Edgar, por todo, pero particularmente por compartir la vigilia cuando mamá partió.*

*A mis padrinos, por sus consejos e incondicional apoyo.*

*A mi familia, por sus valiosos apoyos y lecciones de vida.*

*A mis amigos, por nuestras andanzas.*

*A mi asesor Javier, por sus enseñanzas y paciencia.*

*A mi entrañable Universidad y a la familia universitaria, por permitirme ser parte de esta gran institución.*

*A mí amada FES Acatlán, por alojarme en sus aulas, pasillos y espacios.*

# Contenido

# Introducción

Durante la próxima década se estima que la cantidad de datos digitales representará 44 veces los datos existentes el día de hoy, según un reporte emitido por la International Data Corporation en mayo del 2010[1]. Es evidente entonces que el manejo de volúmenes crecientes de datos se hará cada vez más complejo para las organizaciones. La criticidad de los datos digitales para las organizaciones y su tendencia al crecimiento son el principal motivo para cuidar el desempeño del almacenamiento y la recuperación de dichos datos como un factor integrante en la medición de la disponibilidad. La disponibilidad se traduce en la oferta ininterrumpida de los servicios que el negocio provee, por ello, la presente investigación se centra en establecer una metodología de análisis de desempeño que sirva para aplicarse en los manejadores de bases de datos relacionales, o RDBMS por sus siglas en inglés, más populares en el mercado[2].

De los líderes del mercado de manejadores de bases de datos relacionales se escogieron 3: IBM DB2, Oracle y Microsoft SQL Server. Para complementar la investigación se escogió el RDBMS de software libre MySQL. Esta investigación sólo menciona el motor InnoDB de MySQL, esto en razón de que es el motor más difundido de MySQL que soporta completamente las propiedades ACID de una transacción[3], y parte de la metodología que se propondrá tratará de como determinar lentitudes por concurrencia y control transaccional.

Vale la pena destacar que es en los RDBMS donde se conjuntan diferentes áreas del conocimiento en matemáticas, teorías de la computación y del desarrollo de sistemas de forma muy evidente. Podemos encontrar la convergencia en áreas tales como el Álgebra, los Métodos Numéricos, la Estadística, la Probabilidad, las Estructuras de Datos, los Algoritmos, los Lenguajes de Programación y los Autómatas por mencionar sólo algunos.

Para una mejor comprensión de la investigación el lector ocupa tener noción de los conceptos del álgebra de conjuntos y conceptos del modelo relacional de bases de datos, de la normalización propuesta por Edgar Codd y del lenguaje SQL. También es deseable contar con fundamentos de programación estructurada y programación orientada a objetos. Otro conocimiento que resulta útil para la compresión de la investigación es la teoría de autómatas. Respecto a los sistemas

---

[1] IDC, *IDC Digital Universe Study, sponsored by EMC*. May 2010, http://www.emc.com/collateral/demos/microsites/emc-digital-universe-2011/index.htm. [consulta: Noviembre de 2011]

[2] Yuhanna, N., Gilpin, M., D'Silva, D., *The Forrester Wave: Enterprise Database Management Systems, Q2 2009*. http://www.microsoft.com/presspass/itanalyst/docs/06-30-09EnterpriseDatabaseManagementSystems.PDF. [consulta: Noviembre de 2011]

[3] Tuuri, H., Sun, C., *InnoDB Internals: InnoDB File Formats and Source Code Structure*. MySQL Conference, April 2009. http://www.innodb.com/wp/wp-content/uploads/2009/05/innodb-file-formats-and-source-code-structure.pdf. [consulta: Noviembre de 2011]

operativos y al hardware es deseable que el lector tenga conocimientos básicos en los temas mencionados.

## Hipótesis

Se afirma que los 4 RDBMS escogidos para la investigación comparten similitudes en su arquitectura, las estructuras de datos, los mecanismos de aislamiento de transacciones y los mecanismos de ejecución. Estas similitudes en su conjunto permiten generar una metodología común del análisis de desempeño de RDBMS.

## Alcance

La investigación estará acotada a encontrar una metodología del análisis de desempeño utilizando una definición unificada del término a fin de establecer deficiencias en la construcción y mantenimientos de estructuras de datos, código del lenguaje SQL, manejo de concurrencia y en los planes de ejecución. Para encontrar estas deficiencias se propone medir el consumo y la espera por recursos de los diferentes subsistemas así como la duración de solicitudes al RDBMS.

No se analizarán factores como la normalización de base de datos debido a que este es un concepto aplicado al modelado de relaciones partiendo de la definición establecida por el modelo relacional, y el modelo relacional no contempla al desempeño como parte de su definición. Tampoco se analizará el impacto al desempeño de los RDBMS derivado de las capacidades de hardware ni de las características de los sistemas operativos debido a que la investigación está acotada al funcionamiento interno de los RDBMS de forma general, ya que es en esta generalidad que plantear una metodología es posible.

## Aportación

Resulta útil exponer los fundamentos matemáticos y las similitudes que los RDBMS mantienen. Clarificar estos aspectos de los RDBMS en esta investigación aporta una guía para todo aquel interesado en entender mejor los factores del desempeño de los RDBMS y proveer una metodología común para su análisis.

## Objetivo del trabajo

Demostrar que existe una metodología común para el análisis de desempeño en los RDBMS IBM DB2, Oracle, Microsoft SQL Server y MySQL basada en el fundamento matemático y las similitudes que la arquitectura de los RDBMS comparten.

Un objetivo particular es mostrar que la implementación del lenguaje SQL que cada RDBMS posee no da los mejores resultados en desempeño si se utiliza como una extensión de los lenguajes multipropósito. El lenguaje SQL es un lenguaje de tratamiento de datos desde un enfoque de conjuntos algebraicos y de las operaciones del álgebra relacional.

## Resumen de capítulos

La investigación establece las similitudes que guardan los RDBMS, la metodología de análisis de desempeño común a todos ellos y un ejemplo de aplicación de la metodología. Aquí un breve resumen de los capítulos que componen la investigación.

1. Marco Teórico
   En este capítulo se establece el concepto de desempeño, los factores de desempeño que el modelo relacional implica y los factores de desempeño en la arquitectura general de los RDBMS.
2. Metodología del análisis de desempeño
   En este capítulo se establece la metodología común para los RDBMS IBM DB2, Oracle, SQL Server y MySQL.
3. Implementación de la metodología
   En este capítulo se muestra un ejemplo de implementación de la metodología para el RDBMS SQL Server.

El anexo contiene información técnica y detallada de conceptos del desempeño asociados a hardware, estructuras de datos y arquitectura de los RDBMS.

# 1 Marco teórico

En esta parte de la investigación se encontrarán las similitudes entre los sistemas manejadores de bases de datos relacionales, o RDBMS por sus siglas en inglés, que se escogieron para establecer la metodología. En el marco teórico se sentarán los argumentos necesarios para mostrar una metodología unificada para el análisis de desempeño en los manejadores de base de datos relacionales seleccionados.

La base de este trabajo es que el modelo relacional es matemático. Otra base importante es el hecho de que los RDBMS comparten similitudes en sus componentes, además de utilizar las mismas estructuras de datos.

## 1.1 Concepto de desempeño

Un concepto fundamental para el desarrollo de la metodología es el de desempeño. Se espera que las aplicaciones que emplean los RDBMS tengan un tiempo de respuesta de apenas pocos segundos con un soporte a múltiples operaciones concurrentes.

El término desempeño aplicado a los RDBMS describe el tiempo que dura una solicitud de usuario en ser procesada. Un mejor desempeño para las solicitudes de usuario se alcanza cuando al modificar alguno de los factores que influyen en la ejecución se logra la reducción del tiempo de respuesta que la solicitud registra de forma regular.

El desempeño óptimo representa forzosamente el tiempo de respuesta mínimo necesario para atender una solicitud de usuario, aprovechando las características que los RDBMS implementan.

También es útil definir el término “buenas prácticas” para el desempeño. Bajo la premisa de hacer el menor trabajo para satisfacer las solicitudes del usuario habrá prácticas que ayudarán al procesamiento de solicitudes de la forma más óptima. A estas prácticas les denominamos buenas prácticas.

Existe entonces una metodología del análisis de desempeño basada en la evaluación de mejores prácticas, y no sólo eso, la metodología existe y es la misma para todos los RDBMS; en gran medida gracias a los fundamentos matemáticos, a los módulos y los algoritmos que son comunes en ellos.

Este capítulo aborda a los RDBMS, sus conceptos y sus similitudes desde el punto de vista del desempeño.

## 1.2 Los factores de desempeño en el modelo relacional

Las bases de datos son fundamentales para los sistemas de información, son el corazón de las aplicaciones. La estructura de una base de datos bajo cualquiera de los nombres que se le conoce (esquema, diseño de base de datos, modelo de datos) especifica una base de datos. Uno de los más importantes modelos utilizados para bases de datos modernas es el modelo relacional.

Aunque no es el modelo de datos único, probablemente es el más importante. El modelo relacional se utiliza principalmente para bases de datos transaccionales.

En comparación con otros modelos de datos más recientes, el modelo relacional es particularmente útil para bases de datos transaccionales porque la integridad de los datos es declarada y forzada por el modelo cuando se tiene un buen diseño. Para una mayor referencia del modelo relacional refiérase al anexo "VII. Introducción al modelo relacional"

La integridad de los datos es la conformidad de los datos a las reglas del negocio. Si los datos están mal la primera vez que entran en la empresa entonces estos tienen un impacto negativo sobre el negocio.

Otra ventaja para el modelo relacional es que se define matemáticamente. Por lo tanto, cuando se define un modelado este no es guiado solamente por las mejores prácticas; sino que en la evaluación del diseño se establece firmemente si este es bueno o malo[4].

Aunque la implementación física de los manejadores de bases de datos relacionales varía según el fabricante, el modelo relacional proporciona una percepción coherente de los datos en cualquiera de ellos.

### 1.2.1 Índices

A pesar que los índices no son parte del modelo relacional estos juegan un rol importante en la implementación de la integridad de entidad y en la implementación de la integridad referencial.

Los índices, como estructuras de datos que permiten la organización de registros de una tabla en un orden específico, ayudan a los RDBMS a ubicar rápidamente un registro o un grupo de registros para satisfacer solicitudes internas o de usuario.

En la integridad de entidad encontramos que una vez que se ha definido una llave primaria, cada inserción de un nuevo registro en la tabla cumplirá con la restricción de ser único, y para ello se verificará que el nuevo registro no contenga los mismos valores de llave primaria que algún otro registro existente en la tabla.

Si la tabla no cuenta con algún índice sobre todas las columnas de la llave primaria se observará que dada la inserción de un nuevo registro el RDBMS iniciará una búsqueda en todos los registros de la tabla para encontrar uno que demuestre la violación a la integridad de entidad, lo cual tomará mucho tiempo. Estos tiempos de inserción se verán mayormente afectados cuando la validación se haga en tablas con gran cantidad de registros hasta definir que no hay duplicidad.

En la integridad referencial encontramos que una vez que se ha definido una o algunas llaves foráneas hacia una tabla o un conjunto de tablas, cada inserción de un nuevo registro en la tabla

[4] Chen, Peter (1976). *The Entity-Relationship Model – Toward a Unified View of Data*. United States of America: MIT Press. pp. 10 – 19

será forzada a cumplir con la restricción de tener una contraparte en la tabla o conjunto de tablas referenciadas.

Si las tablas no cuentan con algún índice sobre todas las columnas de las llaves foráneas se observará que la búsqueda de un registro que demuestre la violación a la integridad referencial tomará mucho tiempo. Esto ocurre cuando en la validación se requiere de la lectura de gran cantidad de registros de la tabla o cuando la integridad referencial está declarada con un conjunto amplio de tablas, lo que provocará lecturas de información en múltiples tablas con un consumo de tiempo importante, y esto sólo para la inserción de un registro. En diseños muy intrincados con integridad referencial es posible que los tiempos de validación sean excesivamente altos, y esta es una de las razones que conduce a desnormalizar un diseño de base de datos.

### 1.2.2 Transacciones

Entender el modelo relacional es esencial para un buen diseño que base de datos y una buena escritura de código; además, ayuda a mantener la integridad de datos como una funcionalidad que es crucial en los sistemas transaccionales.

Una única sentencia de manipulación de datos es tratada en el RDBMS como una operación atómica incluso si con ella se modificó un conjunto de múltiples registros; por esa razón, el RDBMS es capaz de identificar uno o varios registros que violan las restricciones y no tiene que esperar a que la sentencia termine para indicarlo.

Nótese que las restricciones de aplicación inmediata simplemente no garantizan que la base de datos reflejará un estado valido de los asuntos del ambiente en el mundo real todo el tiempo. Por ejemplo, aunque la transferencia de dinero de una cuenta a otra está diseñada para ser una operación atómica esta involucra 2 actualizaciones a la base de datos. Ambas actualizaciones están forzadas a terminar exitosamente o a deshacerse por completo.

Por lo tanto, es evidente la necesidad de algún otro medio para hacer bases de datos consistentes con el mundo real en cualquier momento. Esta es la razón de ser de las transacciones. Una transacción es una unidad lógica de trabajo que extiende la noción de atomicidad más allá del nivel de una sola sentencia.

A pesar de que las transacciones juegan un rol importante en los RDBMS, las transacciones permiten definir una unidad de actividad que se considerará atómica bajo el concepto de que se harán todas las sentencias dentro de la transacción completamente y en el caso contrario cada registro afectado por las sentencias de la transacción será regresado al estado que guardaba al inicio de la transacción. Los datos serán considerados consistentes al principio de una transacción y al final, aún en el caso de que la transacción no se complete y se deshagan todas las sentencias.

En los RDBMS se utiliza un sistema de candados, también denominados locks, para asegurar y aislar recursos utilizados en la transacción, previniendo que otros procesos realicen actividades incompatibles sobre esos recursos.

La bitácora de las transacciones garantiza que una transacción almacenada es durable, esto significa que una vez que se ha efectuado el cambio, invariablemente, la modificación ya es parte de la base de datos. Los aspectos mencionados anteriormente son aspectos de las transacciones que se conocen como ACID. Cada aspecto es descrito con mayor detalle en el anexo "IV. Concurrencia y niveles de aislamiento".

## 1.3 Los factores del desempeño en los RDBMS

En este sub capítulo se abordarán, en una visión general, los factores asociados a las estructuras de datos, al aislamiento de transacciones y a los mecanismos de ejecución de sentencias que darán una mejor percepción de como se hace el procesamiento de transacciones y su repercusión en el desempeño.

Para una comprensión detallada de la arquitectura que los RDBMS comparten y los subsistemas que ocupan refiérase el anexo "II. Características y arquitectura general de los RDBMS" donde se profundiza en los módulos de los que se habla en el resto del capítulo.

### 1.3.1 Estructuras de datos en los RDBMS

El módulo de métodos de acceso en los RDBMS es el componente que contiene las rutinas de acceso a las estructuras de datos. En la definición del estándar ANSI SQL-99, un esquema se define como la colección de objetos de base de datos poseídos por un único usuario y todos ellos conforman un único espacio de nombres, también conocido como namespace en inglés. Un esquema es concretamente un contenedor de objetos de base de datos. Los consumidores primarios de almacenamiento en una base de datos son los objetos pertenecientes a los esquemas.

En este sub capítulo se describirán las estructuras físicas y las estructuras de datos utilizadas para almacenar tablas y registros, que son principalmente estructuras heap o B+-tree[5].

Cada estructura de datos tiene un propósito además del almacenamiento de información, en algunos casos estas estructuras mejoran los tiempos de acceso a la información, en algunos otros permiten la preparación de la información para ser operada como parte de la resolución de las sentencias de usuario.

Conocer estas estructuras dará una visión más clara de la importancia de los métodos de acceso, de las estructuras de datos, así como sus características, bondades e influencia en el desempeño.

#### *1.3.1.1 Pages*

Los manejadores de base de datos utilizan unidades básicas de almacenamiento de información llamadas páginas o bloques (en inglés "pages" o "blocks"), dependiendo del manejador del que se trate[6].

---

[5] Hellerstein, Joseph M., Stonebraker, M., (2005). *Readings in Database Systems (4th Edition)*. United States of America: MIT Press. pp. 71

En general los manejadores de base de datos mantienen una definición estándar de una página. Dependiendo del manejador de base de datos y del subsistema de IO, la definición de páginas consiste de asignaciones continuas de bytes que tienen un límite máximo, para algunos manejadores este límite es variable y para otros es fijo en todo el producto.

Los límites máximos serán 2KB, 4KB, 8KB, 16KB, 32KB o incluso otros múltiplos de 2KB. La ilustración 1-1 muestra la definición de páginas y bloques para SQL Server, Oracle, MySQL con el motor InnoDB y DB2.

Estás páginas permiten el almacenamiento de registros de datos pertenecientes a una tabla[7], registros del tipo de datos Large-Object (LOB), o registros de otros tipos de datos especiales, como los registros de índices.

Todos los tipos de páginas comparten las siguientes estructuras:

- Un encabezado
- Una tabla o arreglo de punteros hacia la ubicación de los registros
- Los registros en sí
- Espacio libre en la página

Algunas de las ventajas que los manejadores encuentran en organizar su información en páginas son:

1. Descripción del tipo de información contenida en la página
2. Descripción de las páginas pertenecientes a un objeto
3. Estadísticas del espacio libre en páginas pertenecientes a un objeto
4. Consistencia de datos
5. Capacidad de generar estructuras de datos más óptimas para la organización de información
6. Organización de registros con un esfuerzo mínimo

[6] Oracle es el único manejador de los que se abordan que utiliza el término “block” por lo que se utilizará mayormente el término “page” o página en esta investigación.
[7] Se usará el término “tabla” también para hacer referencia a vistas materializadas en el subsistema de IO

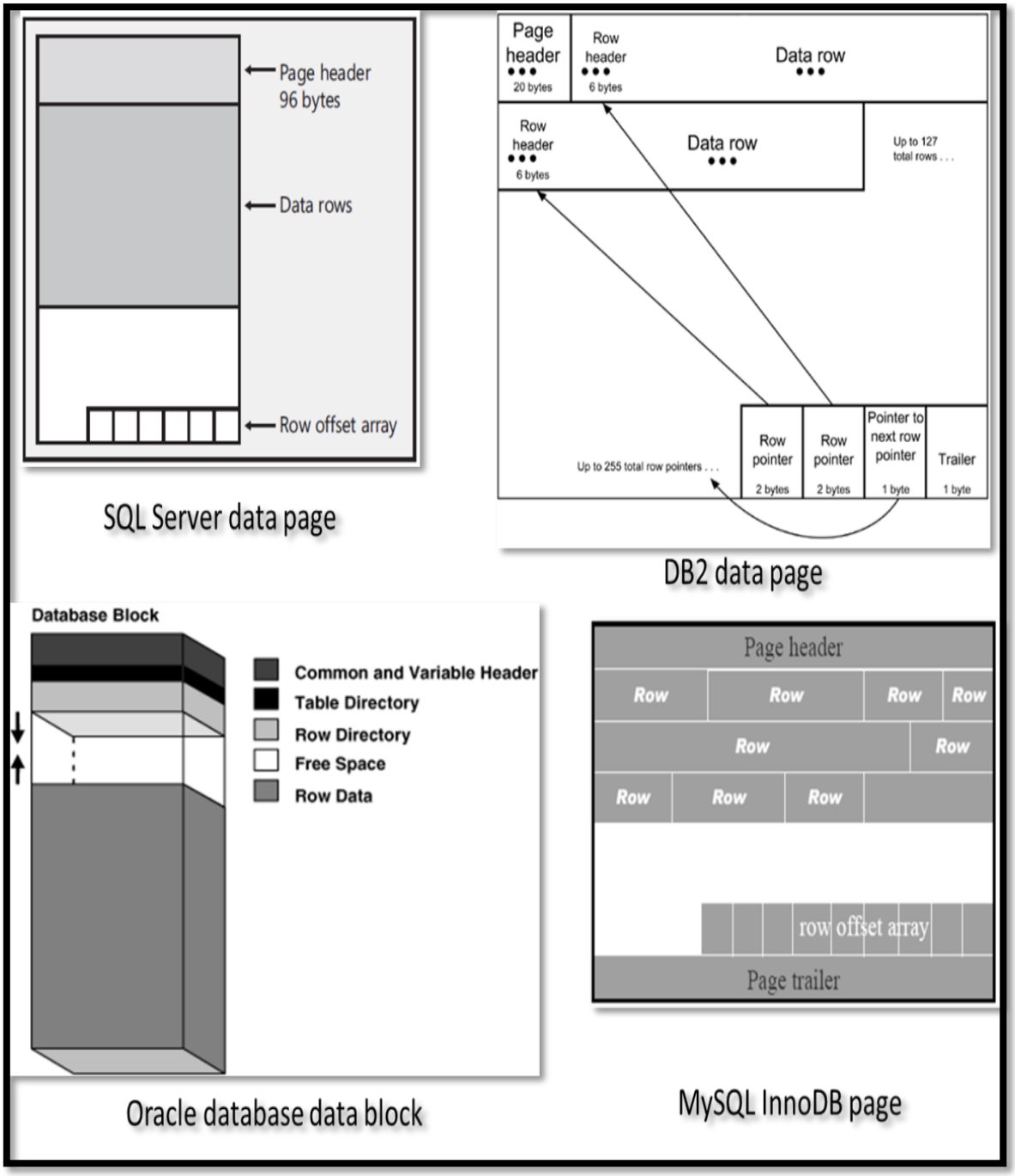



**Ilustración 1-1. La estructura de específicas de páginas de los manejadores de bases de datos**

Todas las bases de datos se conforman de archivos de datos que permiten el almacenamiento y cada archivo de datos está organizado en páginas. Esta es la estructura de datos predominante en dichos archivos. El módulo de métodos de acceso mantiene los algoritmos de las operaciones que se llevan a cabo en las páginas.

### 1.3.1.2 Extents

A pesar de que los registros de información son almacenados en páginas, una página es demasiado pequeña para ser una unidad óptima de alojamiento de objetos de base de datos. Una unidad más grande llamada extent, también conocida como extensión, corresponde a un número de páginas contiguas y es usada como medida en la asignación de espacio y como unidad de lectura de datos desde el subsistema de IO.

El hecho de que los RDBMS no soliciten espacio de almacenamiento en el subsistema de IO por páginas únicas sino por extents promueve que se haga la solicitud de un grupo de páginas libres inclusive antes que se requiera de alojamiento de información en ellas. Los RDBMS utilizan frecuentemente la totalidad de las páginas alojadas en un extent para un mismo objeto. Este alojamiento adelantado de páginas también ayuda a la recuperación más rápida, incluso anticipada, de información que se encuentra de forma contigua, donde con unas cuantas lecturas al subsistema de IO se recuperarán múltiples páginas pertenecientes a una tabla o a un índice. En tablas con una cantidad de páginas menor o igual a las que un solo extent almacena y cuyas páginas están todas en el mismo extent no se requiere más que de una sola lectura para cargar dicha tabla al subsistema de memoria completamente.

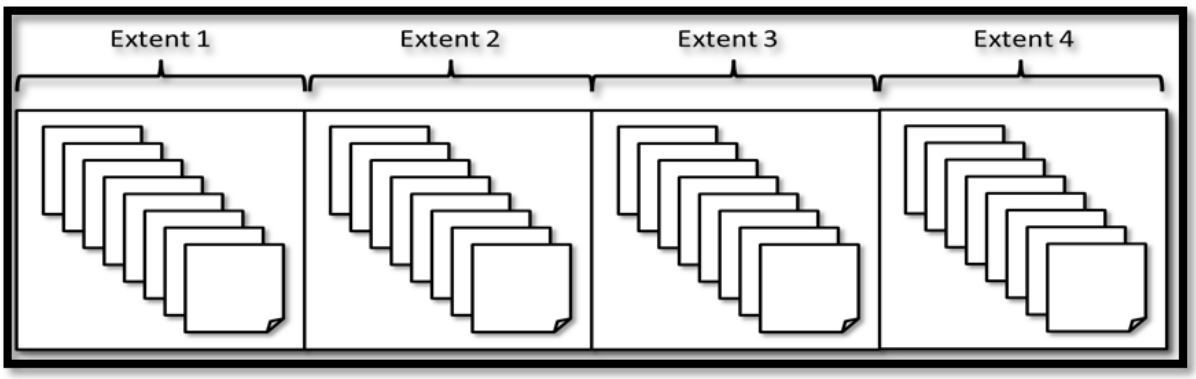



Ilustración 1-2. Relación entre extents y páginas

La ilustración 1-2 muestra la relación entre páginas y extents, donde el extent es una unidad de alojamiento de cierta cantidad de bytes, y este es dividido para formar páginas de datos donde cada una de ellas guarda la estructura descrita en el sub capítulo "1.3.1.1 Pages".

El subsistema de IO cuenta con sus propios mecanismos para el alojamiento y recuperación de datos y tradicionalmente estas operaciones se hacen en bloques que van desde los 2 KB hasta los 64 KB en una sola instrucción y dependen del diseño interno en los discos del subsistema. Contar con una buena alineación de los parámetros del subsistema de IO con los de los RDBMS es importante para mejorar los tiempos de respuesta en las operaciones de IO.

### 1.3.1.3 Heaps

Para la inserción de registros en una tabla es requerido que exista la tabla de forma lógica en el diccionario de datos y que un extent inicial se haya alojado y asignado a dicha tabla en el

subsistema de IO. Una tabla a nivel de estructuras de datos es un conjunto de extents establecidos en un archivo. Básicamente, en una tabla sin orden, los registros son simplemente insertados uno después de otro en la primera página del primer extent hasta que el espacio en la página se termina, así entonces el RDBMS utilizará una nueva página.

Cuando todas las páginas del extent inicial se llenan con registros entonces el RDBMS solicita al subsistema de IO un nuevo extent para la tabla y el proceso continúa hasta que se logra el máximo número posible de extents permitido por el RDBMS o por el subsistema de IO.

Por default, una tabla almacena registros en una forma desordenada utilizando una estructura de datos llamada montículo, también conocida como heap en inglés. En un heap los registros se almacenaran en cualquier página de cualquier extent perteneciente al objeto, sea una tabla o índice, que tenga espacio libre suficiente para alojar el registro entrante. Cuando se recupera información mediante la sentencia SELECT aplicada a una tabla de tipo heap se observará que los registros no son devueltos en el orden que se insertaron en la tabla sino en la manera que el manejador recupere los extents pertenecientes a dicha tabla.

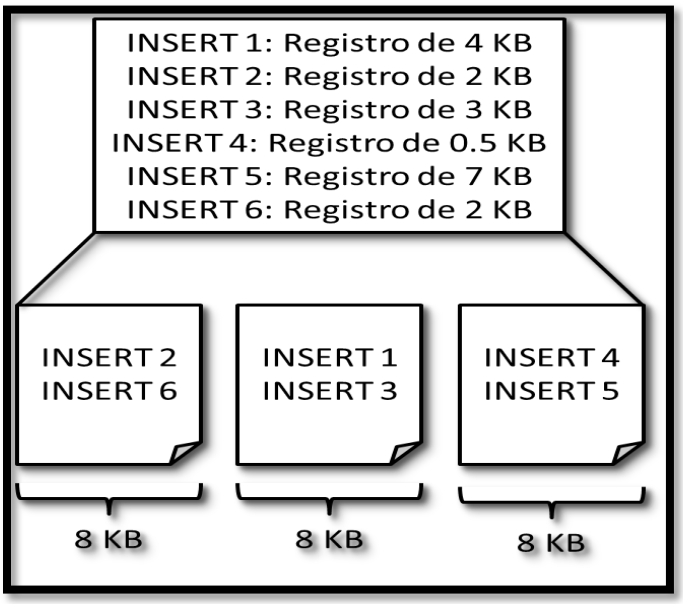



Ilustración 1-3 Inserción de registros en un heap

No todos los RDBMS tienen la capacidad de alojar tablas en forma de heaps, un caso es el de MySQL con el motor InnoDB, en donde todas las tablas tienen un índice que les da orden, inclusive para las tablas en las que no se tiene un índice definido explícitamente durante la creación, en este caso el manejador crea una columna interna que sirve de identificador único del registro.

La búsqueda de un simple registro en un heap resulta muy costosa en tablas con gran cantidad de extents, ya que para encontrar los registros coincidentes es necesario leer todos los registros de todas las páginas que conforman la tabla.

El detalle de las páginas perteneciente a un objeto en forma de heap es descrito con mapas de extents.

En la figura 1-3 se muestran 3 páginas pertenecientes a una tabla en forma de heap y la inserción de 6 registros. Cada registro ocupa una posición en cualquiera de las páginas sin que exista un orden. El RDBMS sólo toma en cuenta la longitud del registro en bytes para determinar la página donde se alojará.

El módulo de métodos de acceso implementa algoritmos para la lectura de información en heaps.

#### *1.3.1.4 Linked list*

Una lista enlazada, también conocida como linked list en inglés, es una colección o secuencia de elementos dispuestos uno detrás de otro, en la que cada elemento se conecta al elemento siguiente por un apuntador. La idea básica consiste en construir una lista cuyos elementos, llamados nodos, se componen de dos partes: la primera parte contiene la información y es, por consiguiente, un valor de un tipo genérico y la segunda parte es un puntero que apunta al siguiente elemento de la lista.

Las listas se dividen en varias categorías, sin embargo aquí sólo hablaremos de las que se utilizan en los RDBMS para el alojamiento de información:

- Listas simplemente ligadas
  Cada nodo contiene un único enlace que conecta ese nodo al nodo siguiente o nodo sucesor. La lista es eficiente en recorridos directos (hacia adelante)
- Listas doblemente ligadas
  Cada nodo contiene dos enlaces, uno a su nodo predecesor y el otro a su nodo sucesor. La lista es eficiente tanto en recorrido directo (hacia adelante) como en recorrido inverso (hacia atrás)

Existen numerosas aplicaciones en las que es conveniente acceder a los elementos o nodos de una lista en cualquier orden.

Las listas ligadas no tienen como propósito primario el ordenamiento de los elementos de la lista sino un almacenamiento que mantenga nociones de la ubicación de los elementos anteriores y posteriores a un nodo dentro de la lista. El ordenamiento es un algoritmo aplicable a la lista para mantener un orden.

En el caso de los RDBMS, son las listas doblemente ligadas las más ampliamente utilizadas y cada nodo es concretamente una página. Con la lista doblemente ligada se mantiene el orden de las páginas pertenecientes a un objeto y esta característica de orden es la diferencia con un heap.

Desde otro punto de vista, el almacenamiento ordenado es el objetivo primario de los índices, es por ello que son una parte integrante de otra estructura de datos aún mayor denominada B+-tree.

La ilustración 1-4 describe una lista doblemente ligada que se forma con las páginas del nivel de hoja de un índice ordenadas del número menor al mayor.

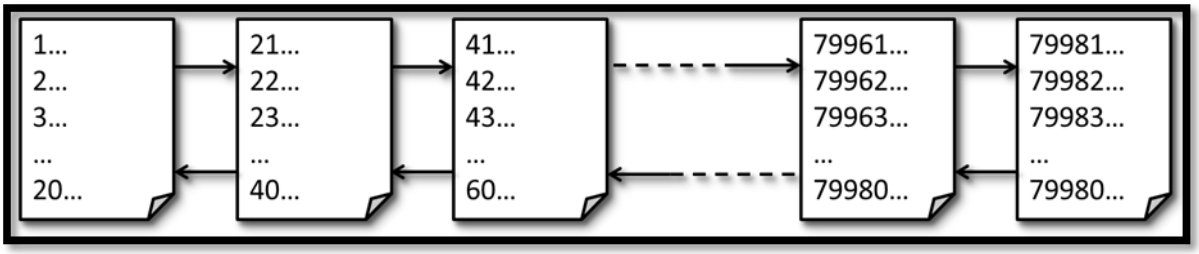



Ilustración 1-4 Lista ligada de páginas en el nivel de hojas de un índice

### 1.3.1.5 B+-trees

Los arboles balanceados, también conocidos como arboles+-B o B+-trees, son estructuras de datos poderosas que tienen amplia implementación en los manejadores de base de datos comerciales. Dentro de los RDBMS la estructura de datos del B+-tree tiene múltiples aplicaciones y la razón principal para que esta estructura de datos sea tan utilizada es que permite la organización de datos en una jerarquía de tipo árbol.

La aplicación práctica más recurrente de un B+-tree es en forma de índices de tablas. Un índice mantiene el orden físico o lógico de los registros de una tabla según se requiera. Un índice se materializa y se almacena de forma fija en el subsistema de IO, por lo que la implementación de un índice es una tarea que requiere de una planeación que evalúe el costo-beneficio. A lo largo de esta investigación se usará el término “índice” para referirnos a la estructura de datos B+-tree.

En el índice que organiza una tabla de forma física se encontrará que el nivel más inferior del árbol estará conformado por las páginas que contienen los registros de datos en sí, todos ordenados en la secuencia indicada en la definición del índice. Por otro lado, un índice que no organiza la tabla físicamente se materializa como una estructura de datos independiente a la estructura de datos con la que está organizada la tabla. Es decir, un índice de orden lógico genera un B+-tree diferente al heap o B+-tree que organiza a la tabla físicamente, con un consumo de espacio y con un orden independiente.

En un índice de orden lógico se tiene que en el nivel más inferior del árbol estará conformado por las páginas que contienen los registros de referencias, también conocidos como registros de índice. Los registros de índice contienen los apuntadores que hacen referencia a las páginas de datos pertenecientes al heap o al B+-tree que dan orden físico a la tabla.

En las ilustraciones 1-5, 1-6, 1-7 y 1-8 se muestran ejemplos de las implementaciones de B+-trees en Oracle, DB2, MySQL y SQL Server para índices. Aunque existe una diferencia visual entre las ilustraciones, el concepto del B+-tree se mantiene en todas ellas, es decir, existe una raíz, niveles intermedios, las páginas del nivel de hoja y un orden de los registros almacenados en la estructura.

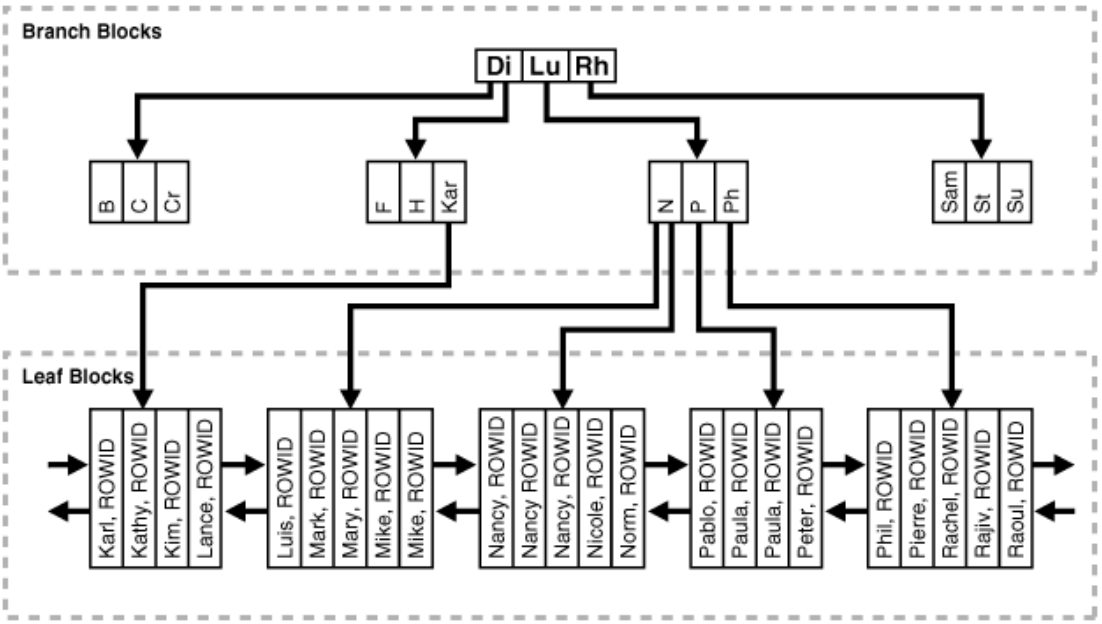



Ilustración 1-5. Implementación en Oracle

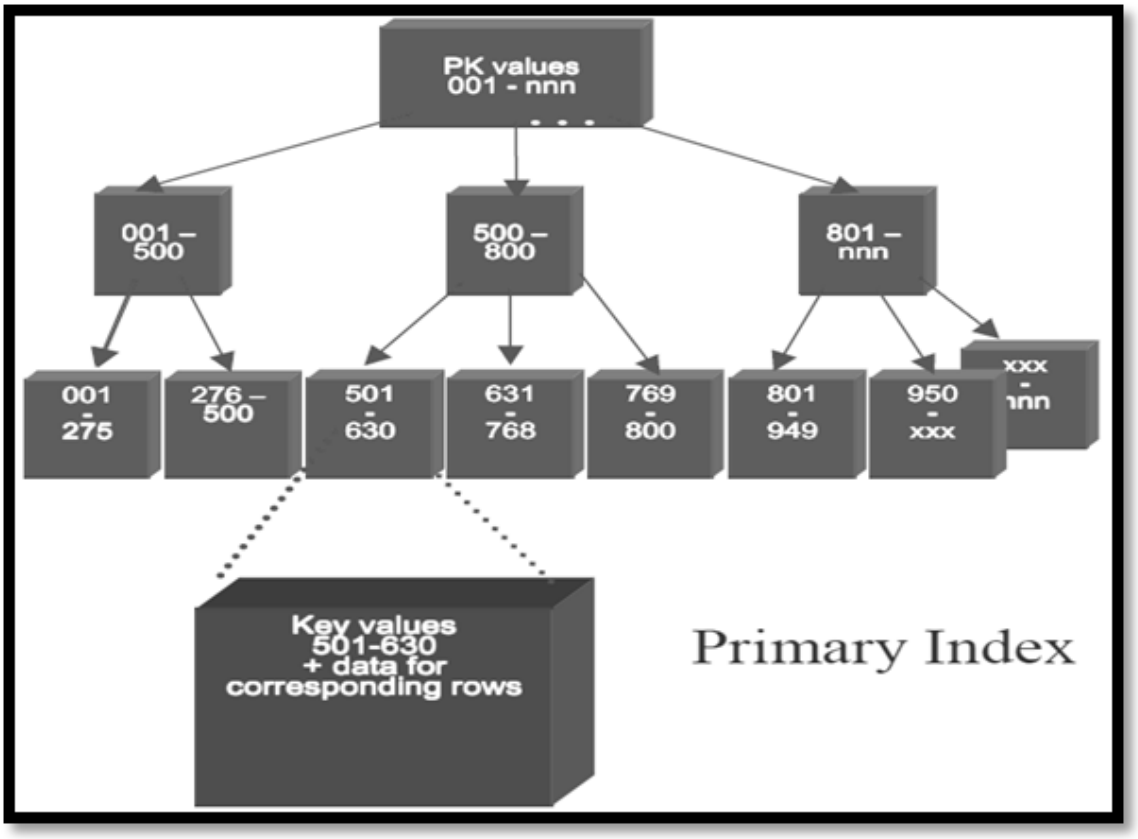



Ilustración 1-6. Implementación en MySQL

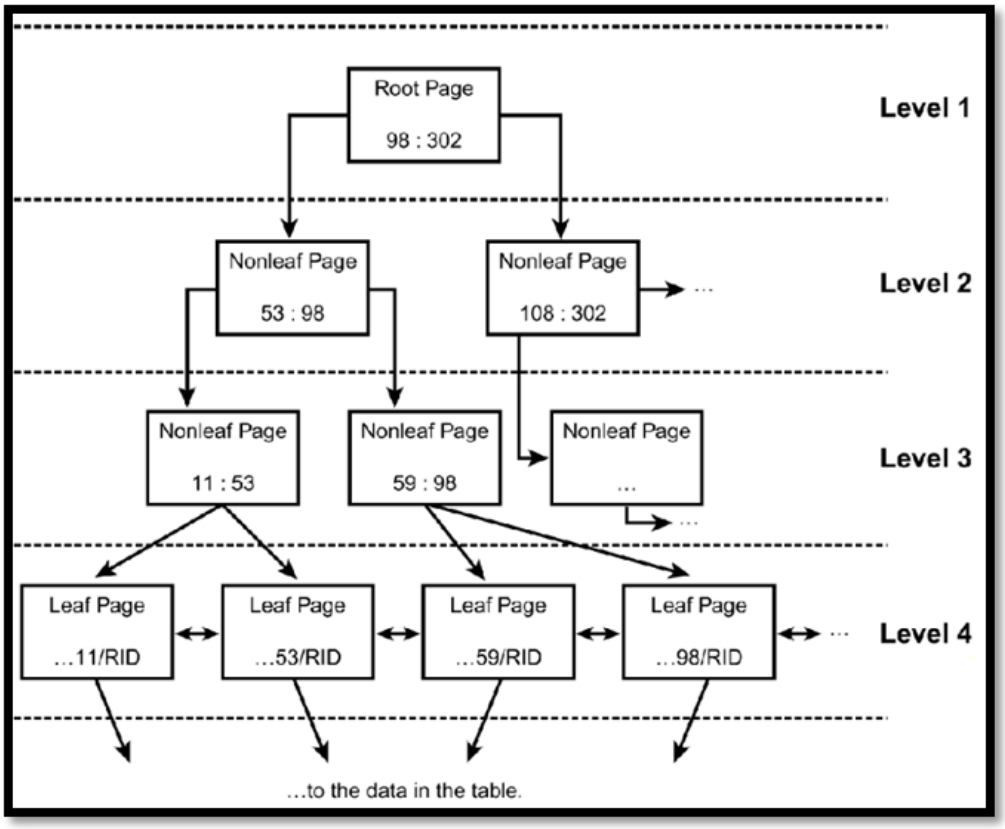



Ilustración 1-7. Implementación en DB2

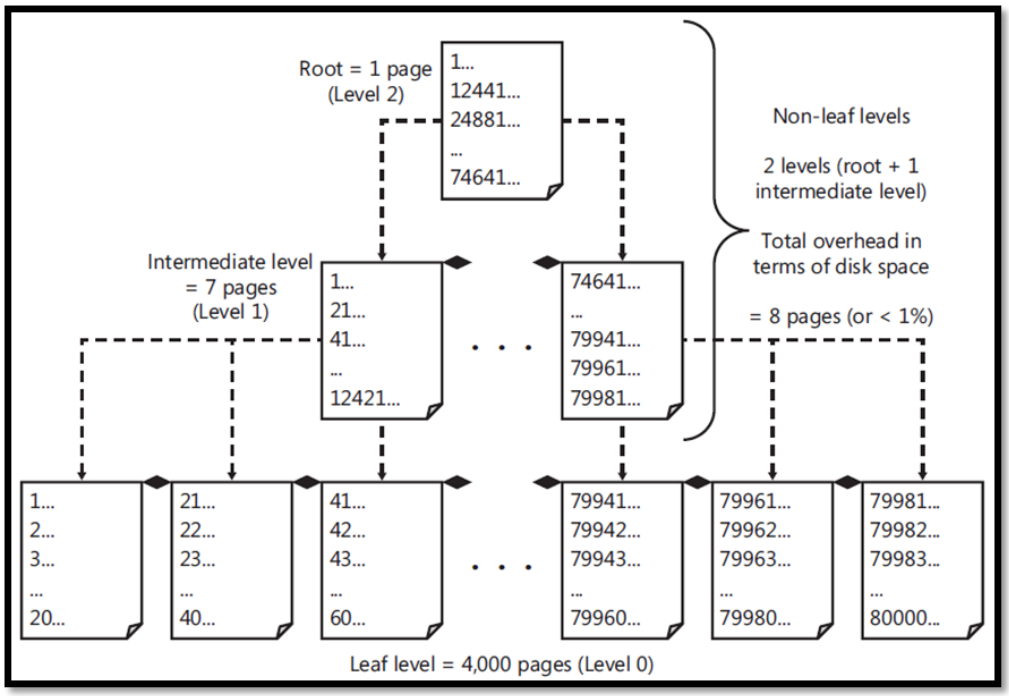



Ilustración 1-8. Implementación en SQL Server

Aunque la ilustración 1-6 del B+-tree de MySQL no lo muestra, entre las páginas de cada nivel existe una segunda estructura de datos que las une, una lista doblemente ligada, que permite la lectura hacia adelante o hacia atrás de páginas en el mismo nivel.

Además de la característica de listas doblemente ligadas en cada nivel de un B+-tree, existe otra razón para utilizar los B+-trees ampliamente. La razón es que la lectura y alojamiento de información desde el subsistema de IO se hace en extents. La lectura de extents con páginas contiguas de un objeto permite "adelantar" lecturas que tienen una probabilidad alta de satisfacer consultas de usuarios. La figura 1-9 muestra una porción de B+-tree en el subsistema de IO organizado en extents.

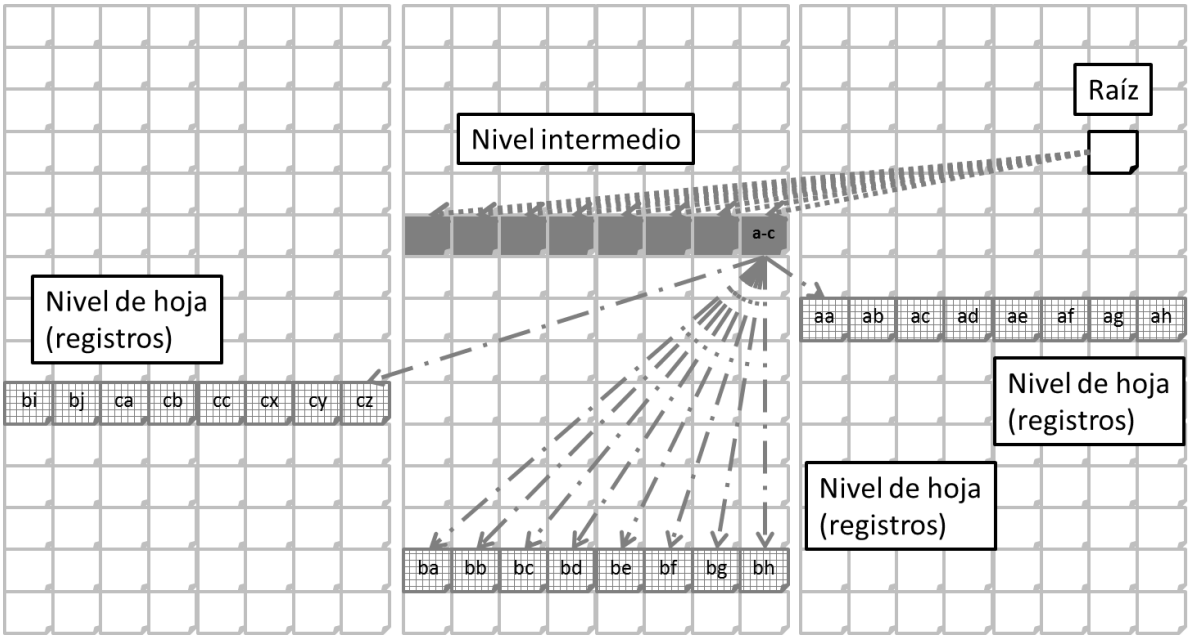



**Ilustración 1-9. Organización de un B+-tree en extents del subsistema de IO**

La consideración a tomarse en cuenta al generar índices es el costo en espacio. En términos de desempeño existe también un costo en el tiempo de mantenimiento que cada índice de la tabla requiere. Cada modificación a la tabla requerirá de la reorganización de una porción de las páginas y los registros que conforman todos los índices que dan orden a la tabla. Todas estas operaciones se llevan a cabo en el subsistema de IO, lo que por definición resulta ser una operación lenta.

Una tabla que cuenta con varios índices, ya sea de orden lógico o físico, requerirá del mantenimiento del orden para todos sus índices, así, todos los registros serán dispuestos en cada índice según el orden indicado por las columnas que componen la llave y las páginas en el nivel de hoja estarán en una lista doblemente ligada con el orden que los registros deban guardar según el índice.

La mejor forma de usar el subsistema de memoria es crear y usar buenos índices buscando mantener el mejor balance de costo y beneficio entre el espacio y el desempeño.

Los puntos a destacarse de esta estructura de datos son los siguientes:

1. El orden y su mantenimiento en todo momento
2. El espacio requerido por el índice
3. El orden permite la concreta búsqueda de información
4. La búsqueda concreta de registros reduce la cantidad de lecturas al subsistema de IO

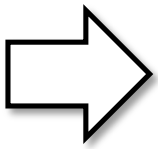
Todas estas características hacen que los B+-trees sean tan ampliamente usados en los manejadores de base de datos. El anexo "III. Ejemplo de implementación de un B+-tree en SQL Server" describe con más detalle una implementación de un B+-tree y sus bondades.

#### *1.3.1.6 Bitmaps*

Los bitmaps son estructuras de datos que permiten definir a nivel booleano alguna o algunas propiedades de un conjunto de datos en los RDBMS. Los manejadores utilizan bitmaps para definir índices materializados en el subsistema de IO o para la construcción de los mismos al momento de la ejecución de una sentencia para aplicar sobre ellos álgebra relacional o calculo relacional y así satisfacer sentencias del usuario.

Un ejemplo del uso de los bitmaps es la implementación de índices bitmap, el cual consta de un vector formado por todos los valores distintos que almacenan las columnas que conforman la llave del índice. Para cada elemento en el vector se almacena un bitmap donde cada bit representa un registro en la tabla. El primer bit representa el primer registro en la tabla ordenada físicamente mediante un heap o un B+-tree, el segundo bit representa al segundo registro en la tabla ordenada físicamente y así sucesivamente. Si el bit está establecido en 0, entonces ese registro no contiene el valor de esa llave específica. El soporte de esta estructura de datos permite a los RDBMS localizar rápidamente todos los registros que contienen una llave. La figura 1-10 muestra una implementación de un índice del tipo bitmap.

| Empleado | | | |
|---|---|---|---|
| IdEmpleado | Nombre | IdDepartamento | Sueldo |
| 1 | John | 2 | 1000.00 |
| 2 | Scott | 2 | 5000.00 |
| 3 | Andrew | 1 | 4500.00 |
| 4 | James | 3 | 1200.00 |
| 5 | Robert | 2 | 5000.00 |
| 6 | Carl | 4 | 4000.00 |
| 7 | Xavier | 4 | 2500.00 |
| 8 | Peter | 1 | 3800.00 |



| Índice Bitmap (IdDepartamento) | |
|---|---|
| Llave | Bitmap |
| 1 | 00100001 |
| 2 | 11001000 |
| 3 | 00010000 |
| 4 | 00000110 |

**Ilustración 1-10 Implementación de un índice mediante un bitmap**

Esta estructura de datos es ampliamente utilizada en la definición de tablas, de los diccionarios de datos, así como para la resolución de sentencias.

#### *1.3.1.7 Hash tables*

Las tablas de resumen, o hash tables por su nombre en inglés, son el apoyo para algunos de los tipos de búsqueda más eficientes. Fundamentalmente, una tabla hash se compone de una matriz en la que los datos se acceden a través de un índice especial llamado clave. La idea principal detrás de una tabla hash es establecer una correspondencia entre los registros de un conjunto origen y las claves resumidas mediante una función hash.

Una función hash acepta una llave y devuelve su código hash, o el valor hash. Las llaves varían en tipo, pero los códigos hash son siempre enteros.

Por lo general, el número de entradas en una tabla hash es pequeño en relación con la cantidad de registros del conjunto que se ha resumido, esto se debe a que para muchas de las llaves que se evalúan en la función hash se obtendrá el mismo código, es decir, muchos de las llaves se agruparan bajo el mismo código hash, de ahí que la tabla de resumen toma su nombre. La representación de conjuntos de datos con tablas hash reduce la cantidad de registros que se operarán para satisfacer la solicitud del usuario.

### 1.3.2 Niveles de aislamiento de transacciones y sus implicaciones de desempeño

Cada RDBMS cuenta con niveles de aislamiento que tienen su origen en el estándar ANSI de SQL y que tienen como propósito evitar los comportamientos de las transacciones dependientes. La elección de algún nivel ANSI de aislamiento para alguna transacción tiene repercusión no sólo en el proceso A que lanza la transacción, sino en otros procesos que requieren hacer uso de los mismos recursos que están siendo asegurados con algún candado desde la transacción del proceso A. El anexo “IV. Concurrencia y niveles de aislamiento” describe con mayor detalle la dependencia de transacciones y los problemas de consistencia

La elección del nivel de aislamiento determina cuál de los siguientes tipos de problemas relacionados con la consistencia ocurrirá:

- **Lost updates.** Un proceso lee un conjunto de datos, hace algunos cálculos basados en este conjunto y posteriormente actualiza el conjunto basado en los cálculos. Si un segundo proceso ha leído los mismos datos que el primero para modificarlos sin tomar en cuenta que el primero está actualizando los mismos datos en ese momento, entonces el segundo proceso sobrescribirá la actualización del primero.
- **Dirty reads.** Permite una lectura de datos no confirmados. Ocurre cuando un proceso cambia los datos pero no confirma la transacción y en ese lapso un segundo proceso lee los mismos datos. El segundo proceso habrá leído datos en un estado de inconsistencia.
- **Non repeatable reads.** Las lecturas no repetibles ocurren cuando en dos lecturas distintas durante la misma transacción se obtienen valores diferentes al leer el mismo recurso. Esto sucede si un segundo proceso de actualización de datos ocurre entre las lecturas realizadas por el primer proceso.

- **Phantom reads.** En dos lecturas separadas en la misma transacción utilizando el mismo filtro de rango para la consulta, la segunda lectura devuelve filas que no formaban parte de la primera lectura. Esto sucede si un segundo proceso inserta nuevas filas que se encuentran dentro del rango especificado en el filtro de las 2 sentencias de lecturas dentro de la primera transacción. Los nuevos registros se conocen como phantom reads.

En este capítulo veremos sólo los niveles definidos en el ANSI 99 del SQL para el aislamiento de transacciones por ser el estándar del mercado de RDBMS. No se entrará en el detalle de cómo cada RDBMS maneja estás implementaciones por lo heterogéneo de ellas. Sin embargo vale la pena indicar que la tendencia de los RDBMS es mejorar la concurrencia mediante la granularización de los niveles donde es posible poner un candado, así como permitir la lectura de información concurrente con el modelo de concurrencia optimista, a fin de reducir los tiempos de espera por candados puestos en transacciones.

#### *1.3.2.1 Locks*

Los candados, conocidos también como locks, son el mecanismo que usan los RDBMS para asegurar recursos de datos en distintos niveles, ya sea a nivel de las filas de datos, de las páginas, de los extents, de otras estructuras de datos superiores a los niveles antes mencionados o inclusive para asegurar tablas y bases de datos completas.

Los candados impuestos desde una transacción previenen que otras transacción realicen ciertas operaciones sobre los recursos de datos que la transacción inicial ha impuesto, así pues, hay candados que restringen operaciones de modificación de recursos de datos desde otras transacciones que concurren, mientras que existen otros candados que incluso previenen que los recursos de datos sean leídos desde otras transacciones en concurrencia.

Los mecanismos de imposición de candados y de su manejo dependen de la implementación que los RDBMS hagan de ellos, sin embargo las mejoras en el diseño de los candados han ido orientadas a mantener la consistencia de los datos mientras que se ofrece la mejor concurrencia posible.

#### *1.3.2.2 Read uncommited*

En el nivel de aislamiento read uncommited, cuyo significado en español es lecturas no comprometidas y que se puede asumir como la lectura de datos no confirmados, indica que una transacción que se lleva a cabo con dicho nivel de aislamiento podrá tener acceso de lectura a recursos de datos que potencialmente se encuentran en el medio de una modificación efectuada desde una segunda transacción concurrente.

En este nivel de aislamiento de transacciones están permitidos todos los problemas relacionados con la consistencia, con excepción de los lost updates, es decir, el comportamiento transaccional permite las dirty reads, las non repeatable reads y las phantom reads. El aislamiento read uncommited de una transacción permite que las operaciones de lectura dentro de dicha transacción no se detengan al encontrar una segunda transacción que ya ha impuestos candados y

que está efectuando modificaciones sobre los mismos datos. La funcionalidad y las desventajas del nivel de aislamiento read uncommited se analizarán desde el punto de vista de desempeño, concurrencia y consistencia con el siguiente ejemplo.

Supongamos que se tiene un proceso A que está ejecutando una transacción donde se modifican datos y para mantener la consistencia y el aislamiento, el RDBMS coloca candados a los datos modificados por el proceso A. Ahora supongamos hay un proceso B que ejecuta una transacción bajo el nivel de aislamiento read uncommited y este proceso B solicita un conjunto de datos que incluyen los que están siendo modificados por el proceso A. El proceso A aún no ha comprometido las modificaciones que ha realizado. En este caso la transacción en el proceso B recuperaría datos, ya sea valores o registros, con las modificaciones tal como los ha dejado la transacción en el proceso A pero sin que aún sean datos comprometidos. Ahora supongamos el proceso B efectivamente ha recuperado y ha mostrado la información que se ha solicitado inclusive con las modificaciones plasmadas por el proceso A que aún mantiene la transacción abierta, inmediatamente después la transacción en el proceso A se aborta y los datos se vuelven al estado anterior al inicio de la transacción. En este punto el proceso B ya desplegó los datos con las modificaciones hechas en el proceso A que para este momento nunca existieron. La funcionalidad de este nivel de aislamiento es la siguiente:

- **Mejores tiempos de respuesta en consultas.** El nivel de aislamiento read uncommited permitió que el proceso B no se detuviera a esperar que los candados impuestos por el proceso A fueran liberados, y por ello su tiempo de ejecución no se vio incrementado por este tipo de espera.
- **Los lectores no tienen que preocuparse por la actividad de los escritores.** En este escenario es posible encontrar procesos escritores que no bloquean a los lectores, por lo que consultas y reportes tendrían un tiempo de respuesta menor comparado con el tiempo que tardaría por esperar a la liberación de algún candado para continuar.

La desventaja de este nivel de aislamiento es:

- **Incoherencia de datos.** La información sucia leída por el proceso B generará problemas en los balances financieros de los movimientos si es que la información es usada para este propósito, lo que generaría descuadres e incoherencia en los datos.

Es evidente que se no desea utilizar este nivel de aislamiento para las transacciones financieras en las que cada número tiene una contraparte. Sin embargo, para proyectar las tendencias de ventas entonces una precisión completa no es necesaria y favorecer una mayor concurrencia hace que valga la pena utilizar este nivel de aislamiento.

#### *1.3.2.3 Read commited*

El nivel de aislamiento read commited, que significa “lecturas comprometidas” y que se puede interpretar como lecturas de datos confirmados, asegura que una transacción que se ejecuta bajo este nivel jamás leerá datos que están siendo modificados desde otra transacción. Una transacción

bajo este nivel de aislamiento sólo recuperará información si los datos no están en medio de alguna modificación.

Poniendo el ejemplo anterior de los procesos A y B se mostrará mejor este nivel de aislamiento. Supongamos que se tiene un proceso A que está ejecutando una transacción donde se modifican datos y, para mantener la consistencia y el aislamiento, el RDBMS coloca candados sólo a los datos modificados con el proceso A. Ahora supongamos que hay un proceso B que abre una transacción que tiene un nivel de aislamiento read commited que solicita un conjunto de datos que incluyen los que están siendo modificados por el proceso A. La transacción en el proceso A en este momento aún no ha comprometido las modificaciones realizadas. En este caso la transacción en el proceso B no recuperará los datos solicitados ya que los candados impuestos por la transacción del proceso A prevendrían la lectura de datos que no son consistentes. Ahora supongamos que el proceso A ha terminado las modificaciones y el proceso B ya contabilizaba 5 segundos esperando a la liberación de los candados. En este punto el proceso A termina la transacción y la compromete, inmediatamente después el proceso B continuará recuperando los datos requeridos, incluso los datos que habían sido asegurados por la transacción del proceso A, y los despliega mostrando información consistente. La funcionalidad de este nivel de aislamiento es la siguiente:

- **Coherencia.** El nivel de aislamiento read commited permitió que el proceso B tuviera una lectura consistente de los datos en todo momento.

La desventaja de este nivel de aislamiento es:

- **Mayor tiempo para la resolución de transacciones.** Todos los procesos esperando por la liberación de un candado agregarían el tiempo de espera al tiempo total de la ejecución del proceso, en este caso son 5 segundos más de ejecución que el proceso B contabilizará al tiempo total de ejecución.
- **Concurrencia.** Se permitiría una menor concurrencia de varios procesos para operar los mismos datos.
- **Es posible encontrar phantom reads.** Al ponerse candados sólo sobre los datos ocupados por primera vez todavía hay posibilidad de que se presenten phantom reads, ya que si el proceso A agrega un registro a la tabla, y el proceso B inmediatamente después hace una segunda lectura dentro de la misma transacción, entonces en esta segunda lectura recuperará este nuevo registro. Los candados sólo se ponen sobre registros existentes y no sobre registros que tienen una posibilidad de existir.
- **Es posible encontrar non repeatable reads.** Al ponerse candados sólo sobre los datos ocupados por primera vez todavía hay posibilidad de que se presenten non repeatable reads, ya que si la transacción desde el proceso B hubiera leído primero los registros, luego el proceso A hubiera modificado el mismo conjunto de registros en la tabla, y el proceso B inmediatamente después hace una segunda lectura dentro de la misma transacción, entonces en esta segunda lectura se recuperarán los mismos registros pero con diferentes valores. Los candados puestos durante la primera lectura del proceso B sólo

durarían mientras se leen los registros la primera vez, aun estando en una transacción, lo que permite que otros procesos modifiquen los datos después de la primera lectura.

- **Posibles conflictos de actualización en el modelo de concurrencia optimista.** En la implementación optimista de este nivel de aislamiento encontraríamos que el proceso B leería información mediante la lectura de versiones o fotos de los datos antes de ser modificados por el proceso A, por lo que la lectura seguiría siendo consistente y se permitiría una mayor concurrencia, pero los conflictos de actualización aún ocurrirán si la transacción B y la transacción A hicieran modificaciones de los mismos datos por lo que alguna o ambas transacciones serían desechadas.

#### *1.3.2.4 Repeatable reads*

El nivel de aislamiento repeatable read significa "lecturas repetidas" y se puede interpretar como la capacidad de una transacción de asegurar que 2 lecturas de datos dentro de la misma transacción darán el mismo resultado cada vez. Dicho nivel de aislamiento agrega algunas propiedades al nivel de aislamiento read commited que aseguran que una transacción bajo este nivel de aislamiento impondrá candados sobre los datos que ocupe hasta que la transacción se comprometa, evitando que otra transacción concurrente modifique los datos asegurados.

Poniendo el ejemplo anterior de los procesos A y B se mostrará mejor este nivel de aislamiento. Supongamos que se tiene un proceso A donde se va a ejecutar una transacción que va a operar datos y se desea asegurar que mientras la transacción calcula los nuevos valores de los datos ningún otro proceso los modificará y para ello se usa el nivel de aislamiento repeatable read. En este punto el proceso A ya leyó los datos para comenzar a operarlos. Ahora supongamos que hay un proceso B que inicia una transacción que intenta modificar los mismos datos que el proceso A ya aseguró. En este caso el proceso B no recuperará los datos solicitados para modificarlos ya que el candado puesto por el proceso A prevendría que el proceso B lea datos inconsistentes hasta que el proceso A cierre la transacción. Ahora supongamos que el proceso A requiere que dentro de la transacción se haga una segunda lectura de los mismos datos. Debido a que el proceso B no ha podido modificar aún los datos a causa del candado impuesto por la transacción en el proceso A, el proceso A recuperará los mismos datos que en la primera lectura. Durante toda esta interacción el proceso A mantuvo los candados desde la primera lectura y hasta que la transacción finalizó.

La funcionalidad de este nivel de aislamiento es la misma que la del nivel read commited con las siguientes adiciones:

- **Coherencia.** El nivel de aislamiento repeatable read permitió que el proceso A tuviera acceso a los mismos valores del conjunto de datos que se operaron durante toda la transacción. Lo que no quiere decir que el conjunto de datos no hubiera crecido en cantidad de registros

Las desventajas de este nivel de aislamiento son las mismas que las del nivel read commited con las siguientes consideraciones:

- **Es posible encontrar phantom reads.** Al ponerse candados sólo sobre los datos ocupados por primera vez todavía existe la posibilidad de que se presenten inserciones de registros dentro del rango de los registros asegurados. Los candados bajo este nivel de aislamiento sólo se ponen sobre registros existentes y no sobre registros que tienen una posibilidad de existir.
- **Mayor tiempo para la resolución de transacciones.** Debido a que los candados impuestos sobre datos ocupados durarán por el tiempo que dure la transacción se observará que el tiempo de espera por la liberación de candados es aún mayor que en el nivel read commited.

#### *1.3.2.5 Serializable*

El nivel de aislamiento serializable, que significa "en serie", forma las transacciones en una serie donde todas las transacciones están completamente aisladas y se encuentran una detrás de otra según el orden en que han llegado.

El nivel de aislamiento serializable asegura que los registros leídos o modificados en una transacción bajo este nivel de aislamiento no serán modificados desde otra transacción bajo ninguna circunstancia, y no sería posible agregar registros en el rango que comprende el primer y el último registro de los conjuntos de datos operados por la transacción. En otras palabras, los registros fantasmas no aparecen si la misma consulta se ejecuta dos veces dentro de una transacción. Serializable es por lo tanto, el más fuerte de los niveles de aislamiento, ya que evita todos los posibles comportamientos indeseables discutidos anteriormente.

La aplicación efectiva del nivel de aislamiento serializable requiere que no sólo se impongan candados sobre los datos que se ha leído o modificado dentro de la transacción, sino que también se impondrán candados en los datos que aún no existen. Por ejemplo, supongamos que en una transacción iniciada por un proceso A se ejecuta una instrucción SELECT para leer todos los clientes cuyo código postal está entre los valores 98000 y 98100. Como resultado de la primera ejecución se recuperarán 2 registros coincidentes con el rango. Para hacer cumplir el nivel de aislamiento serializable el RDBMS impone candados en ese rango de filas, es decir, se asegura que no habrá la inserción de registros entre los códigos postales 98000 y 98100, de manera que otros procesos tendrán que esperar hasta que el proceso A cierre la transacción para continuar sus propias transacciones. Si una transacción desde un proceso B intentara agregar un nuevo cliente con el código postal 98099 tendría que esperar a que el proceso A finalice su transacción para que se libere el candado impuesto en el rango, para así poder continuar con la inserción.

La funcionalidad de este nivel de aislamiento es la misma que la del nivel repeatable reads con las siguientes adiciones:

- **Coherencia.** El nivel de aislamiento serializable permitió que el proceso A siempre tuviera acceso al mismo conjunto de registros durante toda la transacción.

Las desventajas de este nivel de aislamiento son las mismas que las del nivel repeatable reads con la adición de ser el nivel de aislamiento que soporta la menor cantidad concurrente de usuarios sobre el mismo conjunto de datos.

El nivel serializable recibe su nombre del hecho de que la ejecución de múltiples transacciones en este nivel de aislamiento al mismo tiempo es el equivalente a correr a una a la vez, es decir, en serie. Por ejemplo, supongamos que las transacciones A, B y C se ejecutan simultáneamente en el nivel serializable y cada una trata de actualizar el mismo rango de datos. Si el orden en que las transacciones imponen candados en el rango de datos es B, C, y A, el resultado obtenido de la ejecución de los tres al mismo tiempo es el mismo que si se tratara de ejecutar secuencialmente las transacciones en el orden B, C, y luego A.

El nivel de aislamiento serializable no implica que el orden de las transacciones se conozca de antemano. El orden se considera un hecho fortuito. Incluso en un sistema de usuario único, el orden de las transacciones en la cola sería esencialmente al azar. El nivel de aislamiento serializable sólo significa que habrá una forma de ejecutar las transacciones en serie para obtener el mismo resultado que se obtiene cuando se ejecutan simultáneamente.

#### *1.3.2.6 Deadlocks*

Un deadlock se presenta cuando dos o más transacciones están manteniendo candados y solicitando candados sobre los mismos recursos, creando una dependencia cíclica. Los deadlocks ocurren cuando las transacciones tratan de imponer candados en los mismos recursos pero en diferente orden. Por ejemplo, consideremos estas 2 transacciones ejecutándose sobre la tabla StockPrice:

- Transacción 1

```
BEGIN TRANSACTION;
UPDATE StockPrice SET close = 45.50 WHERE stock_id = 4 and
date = '2002-05-01';
UPDATE StockPrice SET close = 19.80 WHERE stock_id = 3 and
date = '2002-05-02';
COMMIT;
```

- Transacción 2

```
BEGIN TRANSACTION;
UPDATE StockPrice SET high = 20.12 WHERE stock_id = 3 and
date = '2002-05-02';
UPDATE StockPrice SET high = 47.20 WHERE stock_id = 4 and
date = '2002-05-01';
COMMIT;
```

En un escenario desafortunado cada transacción ejecutará la primera actualización de un registro. Después cada transacción tratará entonces de actualizar el segundo registro y se encontrará que este registro ya tiene un candado impuesto por el otro proceso para asegurarlo. Ambas transacciones esperarán indefinidamente a que se complete el proceso que mantiene los candados sobre los recursos que este primero solicita, a menos que de alguna manera el RDBMS determine esta condición y rompa el deadlock.

Para combatir este problema los RDBMS implementan varias formas de detección de deadlocks. Un ejemplo es el RDBMS MySQL con el motor de almacenamiento InnoDB que es capaz de detectar estas dependencias circulares para así prevenir esta condición. Sin embargo todos optan por deshacer alguna de las transacciones aplicándole un rollback para permitir que la otra transacción termine. Vale la pena mencionar que los deadlocks no sólo se presentan entre 2 procesos, sino que incluso se presentan dependencias cíclicas entre 3 o más procesos, por lo cual alguno de los n procesos será forzado a deshacerse. En ese contexto, un deadlock representa un problema de desempeño debido a la afectación en la concurrencia, el cual es deseable evitar o reducir.

### 1.3.3 Mecanismos de ejecución de sentencias en el RDBMS

A medida que crece la cantidad de información que se genera en todos los sistemas a nivel mundial, los fabricantes de manejadores de base de datos han notado la necesidad de diseñar sus productos RDBMS de tal forma que atiendan cada vez más rápido a las transacciones del cliente con un mejor manejo de cantidades cada vez más grandes de datos. Mientras que la capacidad de almacenamiento de los subsistemas de IO es cada vez mayor, los RDBMS han especializado sus módulos de métodos de acceso para la recuperación y tratamiento de esos volúmenes de datos, logrando productos que permiten un acceso más rápido a los datos con el mejor uso de los subsistemas.

En este sub capítulo se dará una visión general de la importancia del módulo de optimización de queries que todos los RDBMS implementan para la resolución de transacciones de usuario. Para encontrar mayor detalle de los métodos de acceso comunes en los RDBMS refiérase al anexo "V. Métodos de acceso comunes en los RDBMS"

#### *1.3.3.1 Planificación de sentencias y optimización basada en costos*

Antes que cualquiera de los RDBMS ejecute una sentencia del usuario, el módulo de optimización evalúa entre diversos métodos de acceso para obtener el mejor flujo de acceso y operación de los datos en términos de desempeño. Este módulo de optimización, también llamado optimizer, es parte de la arquitectura del RDBMS y en él que están implementados todos los algoritmos de operaciones del álgebra y del cálculo relacional en forma de funciones de lectura, escritura e iteración de datos. El módulo de optimización es una máquina que toma las sentencias SQL que son operables con el álgebra relacional o con cálculo relacional con la premisa de que todas las tablas son conjuntos de datos y que la operación de dichos conjuntos debe llevarse a cabo en el menor tiempo posible.

Todas las operaciones de conjuntos tienen múltiples formas de solución y el optimizer de hecho es una máquina capaz de generar múltiples maneras de resolver la misma operación de conjuntos utilizando la información de las estructuras de datos (heaps, B+-trees, etc.) con las que cuentan las tablas de las que se extraerán los conjuntos requeridos para las operaciones. El optimizer también toma en cuenta factores como el control concurrente, niveles de aislamiento y las estadísticas para armar flujos y operaciones óptimas.

El optimizer del RDBMS evalúa cada posible forma o camino de resolución pero sólo escoge el que resulta más rápido y a ese camino se le denomina plan de ejecución o ruta de acceso[8]. El optimizer es de hecho el componente principal de los RDBMS y por mucho el componente más complejo en ellos.

El plan de ejecución o ruta de acceso generado se almacena en estructuras del RDBMS como los buffer en el subsistema de memoria o directamente en el subsistema de IO ya que el tiempo invertido en la creación del plan es valioso por ser el resultado de serie de operaciones complejas y costosas en tiempo. Una vez que se ha generado un plan es mejor reutilizarlo para ejecuciones subsecuentes de la misma sentencia, de otra forma se tendría que destinar tiempo del subsistema de procesamiento y de otros subsistemas en generar de nuevo el plan, lo cual se traduce en una sub utilización del RDBMS. Refiérase al anexo "I. Subsistemas manejados por el sistema operativo" para encontrar el detalle de los subsistemas que los RDBMS utilizan para la resolución de sentencias.

Todos los manejadores de base de datos modernos han adoptado la utilización de un optimizador de sentencias basado en el costo de ejecución, donde el costo es una medida del tiempo requerido para satisfacer una sentencia, y así elegir el mejor plan de ejecución. Vale la pena decir que los RDBMS han implementado optimizadores que no hacen la búsqueda exhaustiva del mejor plan de ejecución, y en su lugar toman el mejor dentro de un grupo finito. El revisar todos los planes de ejecución posibles para ejecutar alguna sentencia resulta tan costoso y tan lento que sería preferible ejecutar cualquiera de los planes sub óptimos al azar sin tomarse el tiempo en validar los costos de cada uno.

El caso de la más reciente adopción del modelo basado en costos para la optimización es el de Oracle, que anteriormente trabajaba con un optimizador basado en reglas de algebra relacional y en operaciones de cálculo relacional donde era más importante el conjunto de reglas y no los tiempos de ejecución[9]. Sin embargo el modelo basado en costos ofreció a Oracle diversas ventajas, las cuales se detallan en el anexo "X. Historia de la adopción del Optimizador Basado en Costos en Oracle"

[8] Para esta investigación se estará usando el término "plan de ejecución", ya que el término es usado para describir lo mismo en Oracle, MySQL y SQL Server, sólo DB2 utiliza el término "ruta de acceso"

[9] El optimizador basado en reglas dejó de ser soportado en Oracle 10g, siendo sustituido por completo por el optimizador basado en costos para la versión 11g.

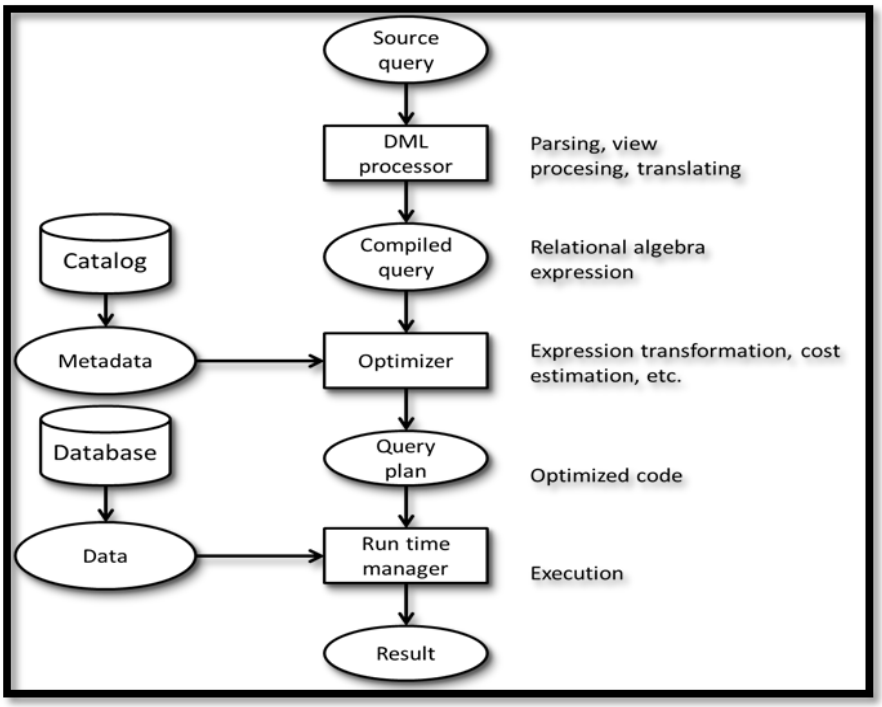



Ilustración 1-11 Mecanismo general del procesamiento de queries

La importancia de este componente es tal que ninguna sentencia de usuario se ejecuta sin antes ser optimizada por el optimizer. Las sentencias que no requieren de operaciones de conjuntos simplemente son ejecutadas sin que el optimizer les destine tiempo valioso de procesamiento.

El optimizer arma cada plan de ejecución utilizando operaciones para la lectura, escritura e iteración de los mismo para satisfacer las sentencias, además el optimizer en RDBMS como DB2 tiene la funcionalidad de considerar los detalles de los subsistemas de los que hace uso, lo que le permite una mejor utilización de las bondades y particularidades del hardware y software detrás de los subsistemas y así generar planes de ejecución que se ayuden de estas funcionalidades con lo que logran calcular mejores planes.

El entendimiento de las operaciones que el optimizer coloca en los planes de ejecución resulta muy útil en la identificación de planes sub óptimos que deriven en altos tiempos de respuesta. También resulta útil el conocimiento de las operaciones descritas en los planes de ejecución para determinar las deficiencias en codificación desde el punto de vista de las operaciones de conjuntos y de las estructuras de datos existentes para cada tabla solicitada en el plan.

Cada RDBMS expone mecanismos para indagar el plan de ejecución de una única sentencia o de un bloque completo de sentencias, denominadas batch. La utilidad de entender los planes de ejecución para el análisis de los tiempos de respuesta es otro de los puntos que resulta de interés para la investigación.

#### *1.3.3.2 El flujo de los operadores en un plan de ejecución.*

Una vez que el RDBMS recibe una solicitud de procesamiento, el optimizerw genera y elige un plan óptimo conformado de operaciones. Cada algoritmo de operación recibe el nombre de operador. En el plan de ejecución estos operadores le indican al módulo de ejecución del RDBMS, o executor, que procesos deben realizarse sobre algún conjunto de datos. Cada operador ejecutado da como resultado un nuevo conjunto de datos que es el insumo para el siguiente operador en el plan. Así, el plan de ejecución es una serie de operadores interconectados entre ellos para hacer las operaciones necesarias sobre los conjuntos de datos a fin de completar la solicitud de un usuario.

Esto se puede ejemplificar con una consulta de datos sobre una tabla llamada "Empleados" que está organizada como un heap, donde se requiere que la consulta muestre los registros ordenados por el campo "idEmpleado" de forma descendente.

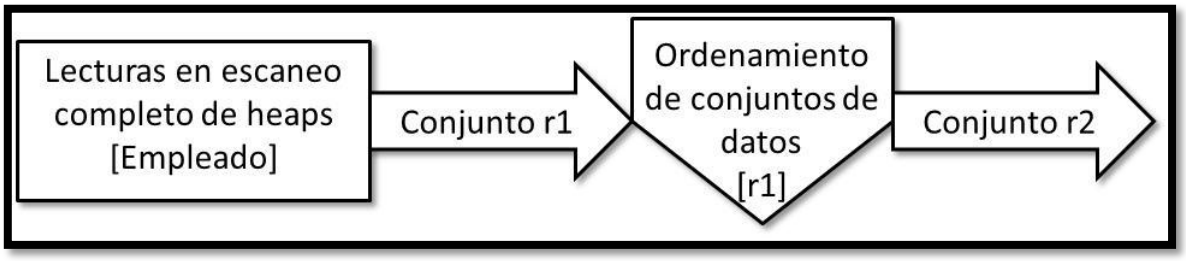



Ilustración 1-12 Plan de ejecución de lectura en heap y ordenamiento

Los operadores descritos en la figura 1-12 corresponden a los operadores descritos en el anexo "V. Métodos de acceso comunes en los RDBMS".

### 1.3.4 Codificación de solicitudes de procesamiento de conjuntos de datos

La codificación de sentencias SQL juega también un papel muy importante en el desempeño que el RDBMS alcanzará para la resolución de la solicitud.

Por ejemplo, ciertas codificaciones intrincadas obligan al optimizer a escoger un plan de ejecución sub óptimo debido a que interpretar y generar varias optimizaciones de la sentencia le llevará mucho tiempo, elevando el tiempo de ejecución. Bajo esta circunstancia el optimizer no tendrá tiempo suficiente para evaluar varios planes de ejecución, y en su lugar generará un grupo pequeño de planes y escogerá cualquiera que le permita comenzar la ejecución, aun si el plan no es el más óptimo.

Otra razón por la que la codificación es importante es que para ciertos patrones de codificación se ignorarán estructuras de datos útiles. Esta situación se traduce en una mayor cantidad de lecturas e iteraciones.

Las solicitudes concebidas como operaciones de conjuntos resultan más óptimas bajo la premisa de que sólo se debe operar la información que las aplicaciones realmente necesitan para reducir el

consumo en los subsistemas al mínimo, reduciendo por consiguiente los tiempos de respuesta del manejador.

El que la codificación se evalúe como ineficiente en algunos casos corresponde al hecho de que la construcción de la base de datos fue deficiente, es decir, no hubo un trabajo suficiente de normalización o del diseño de la base de datos que permita una mejor codificación, por lo que el diseño también es determinante para la buena escritura de código.

Otra consideración de la codificación es la que plantea Itzik Ben-Gan en su libro "*Inside Microsoft SQL Server 2008: T-SQL Programming*"[10]. En este libro menciona al modelo relacional y su propósito en el manejo de conjuntos. También menciona que la escritura de código SQL es menor cuando se utiliza una lógica regida por conjuntos respecto a otro tipo de lógica.

Un ejemplo de problemas en la codificación se presenta cuando en una consulta se selecciona información de una tabla de la cual se filtrarán registros a través de una columna que almacena "cadenas de caracteres", sin embargo las cadenas encontradas en dicha columna para todos los registros en realidad describen fechas. Si la consulta requiere sólo el conjunto de los registros menores a una fecha entonces el desarrollador elegirá primero convertir los valores de la columna de tipo cadena de caracteres por sus equivalentes en un tipo de dato "fecha". Dicha conversión de datos consumirá tiempo, reduciendo el desempeño para obtener los datos requeridos.

[10] Ben-Gan, I., Sarka, D., Katibah, E., Low, G., Wolter, R., Kunen, I. (2010). *Inside Microsoft SQL Server 2008: T-SQL Programming*. United States of America: Microsoft Press. pp. 285 - 286

# 2 Metodología del análisis de desempeño

Con las definiciones y similitudes vistas en el marco teórico se sientan los detalles de los componentes que adicionan tiempo a la ejecución de solicitudes de usuario en el RDBMS. Los factores que se medirán en la metodología del análisis de desempeño serán:

- La utilización de los diferentes subsistemas
- Los tiempos de espera en los diferentes subsistemas
- El tiempo de respuesta derivado de los candados impuestos para asegurar la consistencia
- Prácticas de codificación SQL para procesamiento de conjuntos de datos
- Estructuras de datos
- Planes de ejecución

El que esta metodología se enfoque en el análisis de las solicitudes de usuario y el impacto que estas tienen sobre el comportamiento del RDBMS tiene como propósito encontrar áreas de oportunidad en codificación y diseño de base de datos que provoquen un pobre desempeño y por consiguiente la utilización desmesurada de los subsistemas. Esta aproximación es apropiada ya que el crecimiento del hardware implica un costo monetario difícil de cubrir para las organizaciones y aún con el crecimiento en la infraestructura las áreas de oportunidad en codificación y diseño de hecho no se subsanan y por ello es importante este tipo de análisis.

La metodología de análisis está diseñada para ser ejercida de forma reactiva y proactiva, es decir, con ella se identifican eventos presentes y posibles eventos futuros que provocarán que los tiempos de respuesta sean mayores a los umbrales establecidos, lo que denotará un pobre desempeño.

La metodología consta de las siguientes operaciones, las cuales se detallarán en el resto del capítulo:

1. Definición de las métricas y los umbrales de desempeño
2. Análisis del consumo y espera por recursos en los subsistemas
3. Análisis de la calidad de código
4. Análisis de planes de ejecución
5. Análisis del mantenimiento, uso y diseño de estructuras de datos
6. Análisis de concurrencia

## 2.1 Las métricas y los umbrales de desempeño

En la evolución de los RDBMS y los sistemas operativos, los equipos de desarrollo de estos productos han encontrado que resulta muy útil exponer métricas que indiquen el comportamiento y el desempeño que el producto guarda. En el caso de MySQL existe el esquema de objetos PERFORMANCE_SCHEMA que permite contar con métricas de desempeño del producto, para el

caso de Oracle son las Dynamic Performance Views, para DB2 es el trace y para SQL Server son las Dynamic Management Views por citar fuentes de métricas en cada producto.

La definición de métricas útiles para medir el desempeño de las sentencias que se ejecutan en los RDBMS es el primer paso que permitirá ejecutar el resto de los análisis estipulados por la metodología. Las métricas tendrán que describir los siguientes comportamientos de cada sentencia que se ejecute en el RDBMS para ser elegibles como insumo del análisis:

- Tiempo de espera en la atención del subsistema de IO
- Tiempo de espera en la atención del subsistema de memoria
- Tiempo de espera en la atención del subsistema de CPUs
- Tiempo de espera en la atención del subsistema de red
- Tiempo de ejecución total
- Cantidad de lecturas
- Cantidad de escrituras
- Uso de procesador
- Consumo de memoria
- Candados adquiridos
- Ocurrencia de bloqueos
- Ocurrencia de deadlocks

Otro evento que se considerará como métrica no a nivel de sentencia del RDBMS sino a nivel del servidor es:

- Espacio libre en el subsistema de memoria

Todos los RDBMS exponen métricas similares sin importar que para cada implementación tengan incluso nombres diferentes, además que cada nueva liberación de producto contiene más y mejores mecanismos de monitoreo de desempeño.

Los umbrales máximos y mínimos dentro de los cuales se considerará una operación normal serán dictados por las necesidades del negocio con base en los tiempos de respuesta que se espera para la atención de las sentencias de usuario, es decir, sólo el conjunto de los usuarios, desarrolladores, encargados de la infraestructura, de los administradores de bases de datos y a quien le resulte de interés acordarán cuales son los umbrales ideales. Los umbrales y las desviaciones serán establecidos y medidos en cada una de las métricas elegidas.

Las métricas tienen orígenes distintos dependiendo del RDBMS y del sistema operativo que se desee analizar, en RDBMS como Oracle, MySQL y SQL Server existen tablas que almacenan información de las métricas de desempeño, estas se van poblando de acuerdo a la actividad que se registra. Estas estadísticas e información histórica también son un insumo importante para el análisis. La obtención de la información de desempeño resulta en la utilización de los subsistemas, es por ello que sólo se obtendrá la información importante. Mientras que la información que sea

posible reproducir u obtener posteriormente no se capturará durante la ocurrencia del evento. Capturar toda la información del evento acrecentaría los problemas de desempeño, siendo esto contrario a lo que se pretende de este ejercicio.

## 2.2 Análisis del consumo y esperas en los subsistemas

Una vez que en la operación del RDBMS se han rebasado los umbrales definidos, se requiere obtener los números precisos del consumo y del tiempo de esperas por recursos que se están registrando en el momento que ocurre la desviación, dicha información se obtendrá para cada sentencia que se está ejecutando en ese momento. Con esos valores se generarán los siguientes conjuntos de sentencias acotados a las primeras 10. Cada conjunto estará ordenado de mayor a menor:

- Sentencias con la mayor cantidad de lecturas
- Sentencias con la mayor cantidad de escrituras
- Sentencias con el mayor uso de procesador
- Sentencias con el mayor consumo de memoria
- Sentencias con la mayor cantidad tiempo de espera por candados de datos
- Sentencias con las mayores esperas por el subsistema de IO
- Sentencias con las mayores esperas por el subsistema de memoria
- Sentencias con las mayores esperas por el subsistema de CPUs
- Sentencias con las mayores esperas por el subsistema de red

Estos conjuntos de sentencias serán referenciados como "el conjunto de sentencias analizables". De todas estas sentencias se hará un listado que estará complementado por múltiples columnas, cada una correspondiente a un punto de evaluación de los análisis. Con esta lista se contabilizarán las ocurrencias de malas prácticas y se identificarán las sentencias que tienen mayores problemas de desempeño. Las sentencias con mayores problemas de desempeño serán las primeras que se afinarán. Un ejemplo del formato para registrar el listado del “conjunto de sentencias analizables” se encontrará en el anexo “VI. Listado del conjunto de sentencias analizables”.

No todas las sentencias tienen un valor para el análisis, por ejemplo, las sentencias de backup provocan una utilización importante del subsistema de IO sin que esto signifique un problema de desempeño mayor. Es por ello que contar con un conocimiento del negocio y de los comportamientos que el RDBMS muestra durante la ejecución de ciertas tareas administrativas es útil para evitar el análisis de sentencias no relevantes.

## 2.3 Análisis de calidad de código

Cada RDBMS cuenta con su propia implementación del lenguaje SQL y cada implementación ha nutrido el lenguaje con modificaciones que permiten al programador y al DBA hacer uso de sentencias propias de la implementación para simplificar sus tareas, entonces pues, cada implementación cuenta con un conjunto de sentencias y estructuras de programación particulares. Dependiendo de la implementación del lenguaje SQL se encontrarán sentencias y

patrones de codificación que provocan afectación al desempeño. Este análisis requiere de la definición de los patrones de codificación que afectan al desempeño. A este grupo de patrones que tienden a afectar negativamente al desempeño se le denomina también malas prácticas. La metodología estipula definir y encontrar malas prácticas de codificación de forma automatizada con herramientas como expresiones regulares, autómatas de pila, algún otro autómata o con algoritmos más específicos. No todas las prácticas son analizables con estas herramientas y en esos casos se usará la revisión visual de código, que efectuará un desarrollador o un administrador de bases de datos. El siguiente es el listado de las malas prácticas que se evaluarán:

- Solicitar al RDBMS más información de la que se requiere en una aplicación

  Las columnas que se solicitan de un conjunto de datos tendrán que estar perfectamente descritas y los registros que se requieran estarán claramente delimitados en los filtros de la cláusula WHERE y / o en los criterios de la cláusula JOIN[11].

- Examinar el mismo conjunto de datos o subconjuntos de este más de una vez dentro de una transacción o batch

  Es mejor examinar un conjunto de datos una única vez. No es óptimo recuperar el mismo subconjunto varias veces dentro de un único bloque transaccional o batch. Las operaciones de lectura de conjuntos tendrán que ser reducidas al número mínimo necesario para resolver la sentencia. Un ejemplo de esta mala práctica es el siguiente batch:

```
SELECT SUM(Sueldo) AS MontoNominaDepartamento
FROM Empleado
GROUP BY IdDepartamento

SELECT AVG(Sueldo) AS SueltoPromedioDepartamento
FROM Empleado
GROUP BY IdDepartamento
```

  Este batch ocupa la lectura del mismo conjunto 2 veces, lo cual no es óptimo. Sintetizarlas reduce a la mitad la necesidad de las lecturas como se muestra en el siguiente código:

```
SELECT SUM(Sueldo) AS MontoNominaDepartamento,
       AVG(Sueldo) AS SueltoPromedioDepartamento
FROM Empleado
GROUP BY IdDepartamento
```

- Crear objetos temporales que sean subconjuntos de datos de tablas ya existentes

[11] En particular aquí se utiliza el término “cláusula JOIN” para hacer referencia la cláusula existente en el ANSI 99 del lenguaje SQL.

Los objetos temporales se materializan en los subsistemas de IO y de memoria por lo que una utilización racional permite una mejor utilización en estos subsistemas. Las tablas temporales resultan redundantes cuando representan subconjuntos de tablas ya existentes. Un ejemplo es el siguiente[12]:

```
DECLARE LOCAL TEMPORARY TABLE [schema].[Temp]
(
   IdDepartamento INT,
   SueldoPromedioDepartamento DECIMAL (10,2)
)
INSERT INTO [schema].[Temp]
SELECT IdDepartamento, AVG(Sueldo)
FROM Empleado
GROUP BY IdDepartamento

SELECT AVG(SueldoPromedioDepartamento) AS SueldoPromedio
FROM [schema].[Temp]
```

En esta consulta se genera una tabla temporal para almacenar agregados de datos, lo cual se evitada utilizando un subquery

```
SELECT AVG(SueldoPromedioDepartamento) AS SueldoPromedio
FROM (
      SELECT IdDepartamento, AVG(Sueldo)
      FROM Empleado
      GROUP BY IdDepartamento
)
```

- Forzar comportamientos en el optimizer sin justificación de desempeño

  Existen parámetros que obligan al optimizer a utilizar operadores de forma fija sin permitir que sea el propio optimizer quien decida cuales son los mejores operadores en un cierto plan de ejecución. Un caso de este tipo de manipulación es la indicación de la compilación del código cada vez que se ejecuta, aun cuando la recompilación no sea necesaria.

- Construcción de código que no permita usar las estructuras de datos útiles existentes

  Para los RDBMS existen construcciones de código que no permiten la utilización de índices en la búsqueda de datos, tal es el caso de la filtración de datos con la cláusula WHERE sobre una columna evaluada con una función. La variabilidad en los resultados regresados por la función no permitirá que el optimizer utilice un índice existente sobre dicha

[12] La definición de la tabla temporal en el script de ejemplo utiliza la definición dada en el ANSI SQL del año 1999.

columna. Este tipo de análisis será más preciso bajo una revisión visual del código hecha por un desarrollador o un DBA

- Construcción de código que no permita la reutilización del plan de ejecución generado

  Existen construcciones de código que no permiten la reutilización de un plan de ejecución, lo que genera un mayor consumo del subsistema de CPUs y de otros subsistemas como el de memoria o el de IO, que es donde típicamente se almacenan los planes generados.

- Construcción de transacciones que no estén limitadas a la cantidad mínima de sentencias requeridas para completarse satisfactoriamente

  Existen definiciones de transacciones que contienen sentencias que en realidad no son útiles para completar satisfactoriamente dicha transacción y sin embargo consumen tiempo de resolución. Este tipo de análisis no es automatizable, se requiere de un conocimiento amplio de la aplicación a la que se proveen los datos así como del entorno asociado para una construcción concisa de transacciones.

- Sentencias inicializadoras que requieren de la ejecución de sentencias destructoras

  Existen sentencias que alojan, crean o reservan recursos que de una forma correcta irán acompañadas posteriormente de sentencias que desalojen, destruyan o eliminen la reservación de los recursos solicitados. Es importante que estas construcciones de código se completen correctamente. El ejemplo más simple es la declaración explícita de una transacción.

```
BEGIN/START TRANSACTION Ejemplo
.
.
.
COMMIT TRANSACTION Ejemplo
```

  Las transacciones son solo un ejemplo de sentencias que requieren una instrucción de inicio y una de fin, sin embargo, no se plantea este análisis en código ya que el análisis de concurrencia, propuesto en el sub capítulo "2.3 Análisis de concurrencia", sirve para detectar candados de larga duración. Los problemas en concurrencia son consecuencia directa de transacciones de larga duración o incluso transacciones explicitas sin una instrucción de COMMIT correspondiente.

- Usar la implementación de SQL como una extensión del lenguaje de programación estructurada u orientada a objetos

  Es necesario usar las capacidades de SQL y en general de los RDBMS tal como están concebidas, es decir, SQL y los RDBMS están diseñados para resolver operaciones del

álgebra y del cálculo relacional. Usar al RDBSM y a SQL como una extensión de programación estructurada u orientada a objetos significará un problema de desempeño.

La evaluación del código respecto a malas prácticas se hará sobre el "conjunto de las sentencias analizables". Este es un segundo paso en la metodología.

## 2.4 Análisis de planes de ejecución

El plan de ejecución es sujeto del tercer análisis. Para este caso se buscará la ocurrencia de cualquiera de los siguientes patrones, los cuales indicarán las causas de un pobre desempeño:

1. Operadores de lecturas que muestren la exploración de más datos de los que mínimamente se requieren para completar la ejecución del plan

   Ninguno de los operadores es sub óptimo por su definición. Los operadores de lectura sólo son óptimos si hacen la cantidad mínima necesaria de lecturas para recuperar el conjunto de datos que se requiere en el plan de ejecución, considerando que usará las estructuras de datos más útiles existentes para la tabla. Un ejemplo es el uso de un operador de lecturas en escaneo completo de heaps en una tabla compuesta de una sola página. En este caso no hay mejor operador de lectura, ya que cualquiera de los otros operadores requiere que la tabla esté organizada bajo un B+-tree, lo cual generaría un árbol con una página que fungiría como nodo raíz y una segunda página que contendría todos los datos de la tabla, por lo que leer cualquier registro requeriría de la exploración de 2 páginas.

   Otro ejemplo resulta de un plan para resolver la consulta de toda una tabla sin discriminación de registros con la cláusula WHERE, entonces el plan de ejecución mostrará alguno de los métodos de acceso de lecturas en escaneo, lo cual es conveniente ya que no hay un criterio que discrimine registros.

   Uno de los peores casos de desempeño sub óptimo se encontrará cuando la sentencia contenga una cláusula WHERE que filtre registros en base al valor de una columna que no esté considerada en algún índice. En ese caso el optimizer tendrá inclinación a decidir una lectura en escaneo completo de la tabla aun cuando un solo registro cumpla con el filtro establecido, lo que indica que se exploraron más registros de los necesarios para satisfacer la consulta.

2. Planes de ejecución con operadores join[13] que consumen gran cantidad de registros de los operadores previos y que al final tengan como salida un conjunto mucho menor

   Un plan de ejecución con este patrón estaría indicando una gran cantidad de iteraciones entre los conjuntos. Restringir los conjuntos resultantes de los operadores que proveen los

[13] Se utilizará el término “operador join” para señalar a los operadores que el optimizer utiliza para la reunión de registros de 2 conjuntos de datos.

insumos al operador join ayudará a reducir el número de iteraciones del algoritmo del operador join que el optimizer decida utilizar.

3. Planes de ejecución con operadores join que requieren un tratamiento previo de los conjuntos entrantes

   Los operadores join del tipo merge o hash requieren del tratamiento de los conjuntos entrantes para llevarse a cabo. Es importante determinar que el algoritmo actual del operador join no está influenciado por la cantidad de datos que recibe y si filtrar con anterioridad los conjuntos es una opción. La generación de estructuras de datos que ayuden a los operadores a una lectura más concreta servirá para que el optimizer decida utilizar otros operadores de join más óptimos.

   Los operadores que preparan los conjuntos previamente al operador join requieren gran cantidad de recursos de almacenamiento en el subsistema de IO o de memoria, así como el consumo del subsistema de CPUs para hacer el tratamiento, lo cual también se traduce en un mayor tiempo de procesamiento.

4. Planes de ejecución que demuestran devolver un conjunto muy pequeño de registros al cliente y que incluyan operadores sort que ordenan conjuntos de datos muy grandes

   Este patrón muestra que se están recuperando y ordenando más datos de los que realmente se regresan al usuario. En casos muy críticos, el ordenamiento de los conjuntos requerirán de gran cantidad de espacio para realizar esta operación, lo que haría un uso importante de los recursos del subsistema de IO o de memoria, en algunos momentos se observará una reducción del espacio libre en el subsistema de memoria que retrasará a otros procesos en el alojamiento de objetos además de provocar un mayor uso del subsistema de CPUs.

   Si es factible se limitará al mínimo el conjunto que se va a operar desde operadores tempranos dentro del plan de ejecución, esto se logra mediante la reescritura de código.

5. Planes de ejecución que contienen operadores de agregación de conjuntos con discriminación

   Este patrón es esencialmente el mismo que se expuso para los planes de ejecución con operadores sort, es decir, el plan de ejecución mostrará que se están recuperando y agrupando muchos datos solamente para ser filtrados por operadores posteriores. En casos muy críticos de la agregación de registros se encontrará que el conjunto resultante se materializará en el subsistema de IO, lo que retrasará a otros procesos en el acceso a los discos en general, además de provocar un mayor uso del subsistema de CPUs.

Este análisis se aplicará a los planes de ejecución del "conjunto de sentencias analizables".

## 2.5 Análisis del mantenimiento, uso y diseño de estructuras de datos

El diseño y mantenimiento de estructuras de datos es el cuarto de los análisis en la metodología de desempeño ya que la ausencia o la sobre implementación de estas estructuras provocará problemas de desempeño. Es por ello que se identificarán las siguientes malas prácticas de mantenimiento y diseño de las estructuras de datos.

1. Prácticas de mantenimiento
    a. Tablas con estructuras de datos que a lo largo del tiempo ocupen más espacio del mínimo necesario para su alojamiento

        Un ejemplo es la fragmentación de índices que incrementa la cantidad de páginas que se recuperarán para las operaciones de lectura. En una tabla con una fragmentación mayor al 90% se estarán haciendo casi 2 veces las lecturas necesarias para recuperar información

    b. Falta de la actualización de estadísticas de la distribución de llaves dentro de una estructura de datos sin una justificación de desempeño

        Un ejemplo es la falta de mantenimiento de las estadísticas de una tabla que recibe gran cantidad de consultas y actualizaciones en su contenido. Las estadísticas ayudan al optimizer a decidir por un cierto plan de ejecución. Sin estadísticas confiables el optimizer tiene mayor probabilidad de calcular planes sub óptimos

2. Prácticas de diseño
    a. Omisión de un índice de orden físico para tablas sin una justificación de desempeño

        Para tablas compuestas por muchas páginas, con consultas frecuentes y con criterios de discriminación sobre el mismo grupo de columnas será mejor generar un índice. El índice será definido en el grupo de columnas que se ha identificado a fin de explorar y discriminar registros más rápidamente y así formar sólo el subconjunto de registros solicitado

        Es mejor generar este tipo de índices sobre columnas cuyos valores nunca cambien, para así evitar el movimiento de datos por actualización o inserción de nuevos registros, así se disminuye la fragmentación

        Un B+-tree de orden físico sobre las columnas de llave primaria permite aprovechar la unicidad de los registros y la escasa o nula modificación de los valores de estas columnas.

    b. Omisión de B+-trees de orden lógico en tablas sin una justificación de desempeño

Es mejor generar este tipo de índices sobre tablas que ya cuenten con una estructura de datos de orden físico o en el caso de que las columnas exploradas almacenen valores que cambien con cierta frecuencia, para así evitar el movimiento del orden físico de la estructura de datos aunque la fragmentación se seguirá presentando

Una implementación común es la que se hace sobre las columnas utilizadas como llaves foranes en las tablas que guardan integridad referencial para agilizar las validaciones de integridad y para mejorar el desempeño en los operadores de join de conjuntos de datos

c. Índices cuyas columnas de llave duplican columnas de otros índices

Es importante también cuidar el espacio libre mínimo necesario para la creación de estructuras de datos útiles a la resolución de sentencias de usuario. Bajo esta premisa se buscarán índices que muestren duplicidad innecesaria en las columnas que los definen, esto a fin de que el optimizer decida mejores planes de ejecución y que la estructura de datos sea lo más compacta posible.

d. Tablas que mantienen grandes cantidades de registros, con gran cantidad de modificaciones y con múltiples índices

La implementación de un índice en una tabla exige que cualquier actualización realizada a los datos (INSERT, UPDATE o DELETE) se vea reflejada en el conjunto completo de los índices que posea dicha tabla.

Una operación INSERT, UPDATE o DELETE forzará al RDBMS a insertar, modificar o borrar la entrada correspondiente al registro en todos los índices a fin de mantener el orden preestablecido, además de imponer candados sobre los índices y los registros para prevenir lecturas inconsistentes. Cualquiera de estas operaciones requiere de la escritura en la bitácora de transacciones como parte de la transacción completa.

Entre más índices existan en una tabla mayor será el tiempo que dure la transacción atómica INSERT, UPDATE o DELETE. Se ha de tomar en cuenta que la escritura de la bitácora transaccional se hace en el subsistema de IO.

En casos donde se registra mucha concurrencia se verá una afectación importante a los tiempos de respuesta debido a la duración de los candados impuestos para mantener la tabla y los índices que están siendo modificados. Mantener pocos índices eficientes ayuda a lograr un buen balance entre el desempeño de escrituras y lecturas de información sobre una tabla.

Las prácticas mostradas serán evaluadas en las estructuras de datos usadas en los planes de ejecución del "conjunto de sentencias analizables". Esta no es una restricción ya que de hecho el análisis resulta útil si es aplicado sobre todas las tablas y estructuras de datos de cualquier base de datos.

## 2.6 Análisis de concurrencia

La concurrencia de procesos de usuario es una causa frecuente de lentitudes, estas se acentúan durante las fechas y horas pico de la operación de la empresa. Los patrones que se buscarán para este quinto análisis son los siguientes:

- Candados impuestos por mucho tiempo

  Se obtendrá un mejor desempeño en transacciones que sólo duren el tiempo mínimo necesario para hacer su labor.

  La duración de una transacción se alarga por factores como deficiencias de codificación, planes de ejecución o por el mantenimiento que se ha omitido para ciertas estructuras de datos. Los análisis descritos previamente ayudan a determinar y resolver problemas de la concurrencia.

- Procesos esperando por candados impuestos

  Resulta interesante encontrar los procesos que están esperando por la liberación de algún candado, esto indicará cuales son las sentencias que mayor incremento en el tiempo de respuesta reflejan a causa de este tipo de esperas y con ello se determinará si es recomendable cambiar el nivel de aislamiento para mejorar la concurrencia sin descuidar la consistencia.

- Deadlocks

  Es importante detectar los deadlocks que se generan así como definir los siguientes elementos para analizar las causas que conducen a un deadlock.

  - Los procesos participantes en el deadlock, donde entre más procesos participen en el deadlock será más complejo romper la cadena cíclica.
  - Los recursos a los que cada proceso participante ha impuesto candados y los recursos que los procesos solicitan.

## 2.7 Aplicación proactiva del análisis

El análisis proactivo corresponde a aplicar la metodología de forma preventiva sobre las sentencias que desde una fase anterior a la puesta en producción se deseen evaluar en desempeño.

La evaluación del diseño de una base de datos relacional mediante la validación de la normalización es un tema que no trata esta metodología, ya que inclusive existe literatura que

sugiere la desnormalización de un modelo para mejorar el desempeño. La metodología aplicada de forma proactiva permite evaluar el desempeño de sentencias SQL nuevas o modificaciones a las ya existentes.

Todos los análisis son aplicables de forma proactiva a excepción del análisis de concurrencia. El listado de los análisis que se efectuarán de forma proactiva son:

- Análisis de consumo de recursos en los subsistemas
- Análisis de calidad de código
- Análisis de planes de ejecución
- Análisis del mantenimiento, uso y diseño de estructuras de datos

En el caso del análisis de concurrencia se requiere justamente de contar con toda con la concurrencia normal presente en una base de datos en un ambiente productivo o al menos se ocupará una simulación muy cercana de la concurrencia en producción. Esto hace que la aplicación del análisis se haga basada en estimaciones con un grado de incertidumbre.

### 2.7.1 Metodología en ambiente simulado o basado en estadísticas

La aplicación de la metodología en ambientes simulados basados en la simulación de operaciones o basados en la simulación de estadísticas permite aplicar los análisis con un grado de certeza que dependerá directamente de las similitudes que los ambientes productivos y de pruebas conserven en sus configuraciones, en los datos y en su concurrencia. Existen las siguientes consideraciones para aplicar los análisis:

- Análisis del consumo de recursos y espera por recursos en los subsistemas

  Un servidor de pruebas indicará de forma muy precisa la utilización de los subsistemas y este análisis se hará antes de que la sentencia pase al ambiente productivo, donde tendrá un efecto negativo en el performance.

- Análisis de calidad de código

  Este análisis es ejecutable inclusive sin la intervención del manejador, y en su lugar el análisis se realizará mediante la utilización de autómatas y de la revisión del ojo crítico de un desarrollador o DBA. Se hablará de la utilización de autómatas para el análisis en el sub capítulo "2.7.2 Análisis proactivo de código con autómatas".

- Análisis de planes de ejecución

  Se usarán 2 aproximaciones para obtener el plan de ejecución, una es interrogar al optimizer sobre cuál sería el plan de ejecución que estima utilizaría dada una sentencia. Esta aproximación dará planes de ejecución estimados, es decir, no se ejecutará la sentencia, sólo se obtendrá el plan que el optimizer considere ejecutar.

La segunda aproximación contempla la ejecución de la sentencia para así obtener el plan de ejecución que se utilizó, este plan será más preciso que el plan estimado, sin embargo se requiere ejecutar la sentencia asumiendo todas las modificaciones de datos que esto implique.

- Análisis de mantenimiento, uso y diseño de estructuras de datos

  Los planes de ejecución obtenidos en el análisis de planes de ejecución son el insumo del análisis del mantenimiento, del uso y del diseño de las estructuras de datos. El plan de ejecución dictará las estructuras de datos existentes que se analizarán.

### 2.7.2 Análisis proactivo de código con autómatas

El análisis de código es un trabajo difícil y parte de él se hará utilizando autómatas que entenderán la implementación del lenguaje SQL que el RDBMS haga de dicho lenguaje. Los autómatas serán tan complejos como para entender el subconjunto específico del lenguaje SQL que se analizará. Bajo estas consideraciones se evaluarán las malas prácticas con el conjunto de los autómatas diseñados a fin de encontrar si alguno de ellos ha alcanzado estados de rechazo que indiquen que la mala práctica se cumple. Parte de esta investigación se desarrolló utilizando algunas fuentes bibliográficas sobre Teoría de Autómatas de la universidad de Murcia[14] y otros textos libres[15].

Las expresiones regulares que se utilizarán dependen directamente de como el RDBMS ha hecho la implementación del lenguaje SQL por lo mismo en este módulo no se plasmará ninguna expresión regular ya que no serviría de forma general para todos los RDBMS. Un conjunto de autómatas se describirán en el capítulo "3 Implementación de la metodología", donde se plasmarán autómatas útiles para el análisis del lenguaje SQL que implementa Microsoft SQL Server.

Para comenzar el análisis es importante efectuar algunos procesos preparatorios para adecuar el código que se va a analizar, el primer paso de este análisis será siempre omitir todo el texto que no será evaluado. Esto se consigue usando expresiones regulares para encontrar patrones de texto que no son útiles, un ejemplo de código que no será parte de un análisis son las líneas comentadas en código.

Una vez que el código sólo está conformado por el texto que va a ser analizado entonces se usarán los autómatas para encontrar los siguientes patrones:

- Solicitar al RDBMS más información de la que se requiere en una aplicación

---

[14] Navarrete Sánchez, I., Cárdenas Viedma, M., Sánchez Alvarez, D., Botía Blaya, J., Marín Morales, R., Martínez Béjar, R., *TEORÍA DE AUTÓMATAS Y LENGUAJES FORMALES*. Departamento de Ingeniería de la Información y las Comunicaciones, Universidad de Murcia. http://www.yakiboo.net/apuntes/TALF%20-%20YakiBoo.net.pdf. [consulta: Diciembre 2011]

[15] Pardo, Luis M., Gómez, D., *Teoría de Autómatas y Lenguajes Formales (para Informáticos)*. http://personales.unican.es/pardol/Docencia/TALF/TALF07.pdf . [consulta: Diciembre 2011]

- Caso del SELECT * del ANSI de SQL

  En este caso se utilizarán expresiones regulares que en sí son la implementación de autómatas. Lenguajes de programación como Java o los del .NET Framework disponen de las librerías necesarias para la evaluación de expresiones regulares

- Sentencias sin WHERE

  En este caso se usará un autómata de pila que indique la ausencia de una clausula WHERE

- Crear objetos temporales que sean subconjuntos de datos de tablas ya existentes

  En este caso se usarán expresiones regulares para encontrar sentencias que obliguen la generación de tablas temporales

- Forzar comportamientos en el optimizer sin tener justificación

  En este caso también se buscarán modificadores del comportamiento del optimizer para generar un plan de ejecución utilizando expresiones regulares

- Sentencias inicializadoras que requieren de la ejecución de sentencias destructoras

  En este caso se utilizará un autómata de pila que permita identificar si se han ejecutado las sentencias inicializadoras y destructoras en pares

## 2.8 Aplicación reactiva del análisis

La aplicación reactiva del análisis del desempeño de base de datos consiste en el monitoreo de la actividad que se genera en tiempo real en el RDBMS, la comparación de la actividad con las métricas que se han definido, la captura de la actividad relevante y la aplicación de los análisis mostrados con anterioridad a fin de encontrar las causas de un pobre desempeño.

Cada uno de los eventos relevantes estará conformado por una o más sentencias a las que se aplicarán los análisis de los que se conforma la metodología comenzando por el análisis del consumo y esperas en los subsistemas. Los análisis se aplicarán de forma reactiva a eventos que han ya ocurrido.

### 2.8.1 Mecanismo de evaluación reactiva del desempeño

La captura de información de desempeño será disparada por la ocurrencia de desviaciones en la evaluación de las métricas respecto a los umbrales establecidos. La información que precisa capturarse es la actividad que se está llevando a cabo en el RDBMS. Con dicha información se determinará la causa de la desviación.

Una vez que se presenta un evento de desviación de las métricas y que se captura la información relevante del evento el siguiente paso es almacenar la información de la captura en algún formato

que permita su rápida exploración, para efectuar los análisis descritos previamente y así determinar los orígenes de un pobre desempeño. Tanto la medición de las desviaciones como la captura de información relevante del evento se harán de forma rápida para reducir el impacto en el desempeño. Esto es prioritario ya que sin información oportuna sería muy difícil determinar el origen de los eventos y por lo tanto la remediación de los mismos sería imposible. La evaluación de las métricas se hará dependiendo de los mecanismos que el RDBMS y que el sistema operativo permitan. A continuación se detallan dos formas de evaluación que son convenientes.

Cada vez es más popular el desarrollo de aplicaciones que cuentan con una capa de publicación y notificación de eventos. Esta capa permite que otras aplicaciones se suscriban a los eventos que le resulten de interés. El mismo tipo de implementación existe en los RDBMS y los sistemas operativos. Entonces un primer mecanismo sería la implementación de un monitoreo de métricas basado en la subscripción a eventos relevantes al desempeño para así evaluar el evento contra los umbrales establecidos y decidir si es necesario o no disparar la captura de información.

Un segundo acercamiento es el monitoreo y evaluación de la métricas en intervalos definidos, así pues en cada iteración y de acuerdo a la evaluación de las métricas contra los umbrales se disparará la captura de eventos.

Será mejor evaluar las métricas mediante la suscripción de eventos ya que esto representa una menor posibilidad de pérdida de eventos importantes que se presentan entre cada iteración del monitoreo en intervalos definidos.

La metodología descrita en esta investigación busca tomar una “foto” de la actividad cuando ocurre alguna desviación de las métricas y de ninguna manera busca tomar fotos de la actividad en intervalos definidos. Con esta aproximación evitamos el almacenamiento de información no relevante al análisis reactivo y se acota la información a un conjunto más compacto que permita una fácil lectura de las condiciones en las que se presentó el evento. Sin mencionar las ventajas de almacenar sólo este conjunto compacto en términos de espacio y de la utilización de los recursos así como de los subsistemas necesarios para la instrumentación del monitoreo.

### 2.8.2 Almacenamiento y notificación de eventos relevantes

La información que se capture en la desviación de las métricas se hará considerando lo siguiente:

- Se capturará la información de desempeño de la forma más rápida posible
- La escritura de la información se hará de la forma más rápida posible
- La información una vez almacenada estará en un formato fácilmente explorable

Una buena opción para lograr esto es justamente generar y usar una base de datos de eventos en un RDBMS que permita la correlación de la información capturada en la ocurrencia del evento. Se pretende contar con una forma sencilla de exploración utilizando funciones avanzadas de consulta tal como las que provee el lenguaje SQL.

El almacenamiento de los eventos en una infraestructura diferente a la que se está monitoreando tiene sus costos y es por ello que la aproximación que se describió previamente es conveniente, es decir, usar el mismo RDBMS que se está monitoreando para capturar la información de un evento una vez que se presente una desviación ayudará a tener un mejor uso del espacio y una utilización razonable de los recursos y subsistemas, ocupándolos sólo cuando en realidad se requiera.

Una vez que se ha definido el conjunto de orígenes de la información que se colectará se diseñará un modelo de base de datos para el almacenamiento de la información con un buen desempeño, es decir, se evitará el tratamiento de la información previo a su almacenamiento, y más bien el tratamiento se hará durante el despliegue de la información para el análisis.

La notificación es una parte importante de la aplicación reactiva del análisis, ya que este será el mecanismo que alerte a las áreas que administran al RDBMS cuando ocurra una desviación. El envío de correo o de mensajes a dispositivos móviles como primera reacción a una desviación de las métricas es un buen mecanismo de notificación para las diferentes áreas involucradas en la atención de eventos.

Otra parte importante de la notificación es la generación de reportes útiles que ayuden a identificar fácilmente los patrones descritos en los análisis. Con dichos reportes se pueden entonces encontrar las sentencias que califiquen para el análisis según lo expuesto en el sub capítulo “2.2 Análisis del consumo y esperas en los subsistemas”.

# 3 Implementación de la metodología

El RDBMS escogido para ejemplificar la implementación de la metodología en este caso es SQL Server ya que resulta interesante desde un punto de vista económico, aunque es necesario enfatizar nuevamente que la metodología está diseñada para ser aplicable a cualquier RDBMS de los analizados en esta investigación. Microsoft ha ofrecido el producto SQL Server como un software "todo incluido" desde las versiones 2005 y resulta interesante explorar las funcionalidades que el producto ofrece ya que ayudan a una rápida implementación del monitoreo y del mecanismo de alertas.

En este ejercicio se escogió específicamente la utilización de Microsoft SQL Server 2008 R2 Developer Edition a 32 bits, aunque esta implementación está probada para SQL Server 2005 y posterior. En empresas que manejan grandes cantidades de datos se encuentran ejemplos de problemas de desempeño, por mostrar alguno se describe el siguiente caso:

> *La instancia SQL Server que se analizará es una instancia de misión crítica para una empresa del sector financiero que espera de forma general que el tiempo máximo de una transacción sea de 5 segundos. La instancia se llama INSTITUCION\SQL2K8R2. Hay todo un equipo conformado por diferentes áreas de la empresa que incluye personal del área de administración de bases de datos, personal del área de arquitectura de soluciones de TI, el representante del usuario y el equipo de desarrollo.*
>
> *En las últimas semanas se han registrado quejas de clientes por lentitudes en el sistema. El equipo de DBAs ha decidido implementar un análisis de desempeño que les permita determinar si las lentitudes del sistema se generan en la base de datos. También desean saber cuándo se presenta una lentitud en la instancia para atender la incidencia de forma inmediata.*
>
> *En una reunión entre las áreas se ha encontrado que la empresa tiene las especificaciones técnicas del hardware que se adquirió para soportar la solución, en dicho documento el proveedor indica que la infraestructura está preparada para recibir 2000 transacciones de usuario por segundo con un tiempo de respuesta máximo por segundo en los subsistemas referenciados. La empresa registra una cantidad de sólo 1500 transacciones en los horarios pico.*
>
> *La tabla 1 muestra los tiempos máximos de respuesta del hardware adquirido para la operación de la empresa:*

| *Subsistemas* | *Segundos* |
|---|---|
| *IO en escritura* | *0.2* |
| *IO en lectura* | *2.0* |
| *CPU* | *0.2* |
| *Memoria* | *2.0* |
| *Red* | *0.5* |

**Tabla 1 Tiempos máximos de respuesta del hardware para 2000 transacciones concurrentes**

> *Los DBAs han propuesto definir el umbral del tiempo máximo de bloqueos en 5 segundos para igualarse al tiempo máximo de una transacción. También han propuesto que los deadlocks se reduzcan al mínimo o si ocurren al menos será necesario contar con información que permita prevenir eventos futuros.*

El ejemplo muestra un detalle muy técnico, sin embargo, para poner en práctica la metodología sólo se ocupa de encontrar las sentencias cuya duración sea mayor a 5 segundos. La metodología es aplicable incluso con una noción tan básica de umbrales como este simple tiempo de respuesta.

## 3.1 Elección de métricas y umbrales de desempeño

Como primer paso en la metodología se escogieron métricas de desempeño de un amplio grupo de contadores de performance que Microsoft Windows mantiene del hardware y del software, incluido SQL Server. También se emplearon vistas dinámicas de SQL y objetos del WMI, ya que en ellos SQL expone información del desempeño que las sentencias registran. Basados en el criterio descrito en el capítulo "2.1 Las métricas y los umbrales de desempeño", se analizó documentación disponible del producto[16] con lo que se obtuvo el siguiente listado de las métricas útiles para los análisis:

- Tiempo de espera en la atención del subsistema de IO
  - Contadores de performance de SQL publicados en Windows
    - MSSQL$SQL2K8R2:Wait Statistics\Log write waits\Cumulative wait time (ms) per second
    - MSSQL$SQL2K8R2:Wait Statistics\Page IO latch waits\Cumulative wait time (ms) per second
  - Vista dinámicas de SQL Server
    - master.sys.dm_exec_requests
    - master.sys.dm_os_waiting_tasks

[16] Delaney, Kalen (2005). *Inside Microsoft SQL Server 2005: Query Tuning and Optimization*. United States of America: Microsoft Press. pp.14

- Tiempo de espera en la atención del subsistema de memoria
    - Contadores de performance de SQL publicados en Windows
        - MSSQL$SQL2K8R2:Wait Statistics\Log buffer waits\Cumulative wait time (ms) per second
        - MSSQL$SQL2K8R2:Memory Manager\Memory Grants Pending
        - MSSQL$SQL2K8R2:Wait Statistics\Page latch waits\Cumulative wait time (ms) per second
- Tiempo de espera en la atención del subsistema de CPUs
    - Contadores de performance de SQL publicados en Windows
        - MSSQL$SQL2K8R2:Wait Statistics\Wait for the worker\Cumulative wait time (ms) per second
- Tiempo de espera en la atención del subsistema de red
    - Contadores de performance de SQL publicados en Windows
        - MSSQL$SQL2K8R2:Wait Statistics\Network IO waits\Cumulative wait time (ms) per second
- Tiempo de ejecución total
    - Contadores de performance de SQL publicados en Windows
        - MSSQL$SQL2K8R2:Longest Transaction Running Time
- Cantidad de lecturas
    - Vista dinámicas de SQL Server
        - master.sys.dm_exec_requests
- Cantidad de escrituras
    - Vista dinámicas de SQL Server
        - master.sys.dm_exec_requests
- Uso de procesador
    - Vista dinámicas de SQL Server
        - master.sys.dm_exec_requests
- Consumo de memoria
    - Vista dinámicas de SQL Server
        - master.sys.dm_exec_requests
- Candados adquiridos
    - Vista dinámicas de SQL Server
        - master.sys.dm_exec_requests
    - Contadores de performance de SQL publicados en Windows
        - MSSQL$SQL2K8R2:Wait Statistics\Lock waits\Cumulative wait time (ms) per second
        - MSSQL$SQL2K8R2:Locks\Lock Timeouts/sec_Total
- Ocurrencia de bloqueos
    - Contadores de performance de SQL publicados en Windows
        - MSSQL$SQL2K8R2:General Statistics\Processes blocked

    - WMI
        - BLOCKED_PROCESS_REPORT
- Ocurrencia de deadlocks
    - Contadores de performance de SQL publicados en Windows
        - MSSQL$SQL2K8R2:Locks\Number of Deadlocks/sec_Total
    - WMI
        - DEADLOCK_GRAPH

De las métricas descritas anteriormente se identificaron sólo las más útiles para la evaluación, su utilidad fue evaluada respecto a si permiten una subscripción a los eventos, el listado de las métricas útiles es el siguiente:

- Tiempo de espera en la atención del subsistema de IO
    - Contadores de performance de SQL publicados en Windows
        - MSSQL$SQL2K8R2:Wait Statistics\Log write waits\Cumulative wait time (ms) per second
        - MSSQL$SQL2K8R2:Wait Statistics\Page IO latch waits\Cumulative wait time (ms) per second
- Tiempo de espera en la atención del subsistema de memoria
    - Contadores de performance de SQL publicados en Windows
        - MSSQL$SQL2K8R2:Wait Statistics\Log buffer waits\Cumulative wait time (ms) per second
        - MSSQL$SQL2K8R2:Memory Manager\Memory Grants Pending
        - MSSQL$SQL2K8R2:Wait Statistics\Page latch waits\Cumulative wait time (ms) per second
- Tiempo de espera en la atención del subsistema de CPUs
    - Contadores de performance de SQL publicados en Windows
        - MSSQL$SQL2K8R2:Wait Statistics\Wait for the worker\Cumulative wait time (ms) per second
- Tiempo de espera en la atención del subsistema de red
    - Contadores de performance de SQL publicados en Windows
        - MSSQL$SQL2K8R2:Wait Statistics\Network IO waits\Cumulative wait time (ms) per second
- Tiempo de ejecución total
    - Contadores de performance de SQL publicados en Windows
        - MSSQL$SQL2K8R2:Longest Transaction Running Time
- Candados adquiridos
    - Contadores de performance de SQL publicados en Windows
        - MSSQL$SQL2K8R2:Wait Statistics\Lock waits\Cumulative wait time (ms) per second
- Ocurrencia de bloqueos

    - WMI
        - BLOCKED_PROCESS_REPORT
- Ocurrencia de deadlocks
    - WMI
        - DEADLOCK_GRAPH

Los umbrales que se utilizaron son los mismos que el fabricante de la solución de hardware ofreció en la especificación técnica. Derivado de la tabla 1 se obtuvo la relación de métricas y umbrales descrita en la tabla 2.

| Métrica | Umbral |
|---|---|
| MSSQL$SQL2K8R2:Wait Statistics\Log write waits\Cumulative wait time (ms) per second | Máximo 0.2 segundos |
| MSSQL$SQL2K8R2:Wait Statistics\Page IO latch waits\Cumulative wait time (ms) per second | Máximo 2 segundos |
| MSSQL$SQL2K8R2:Wait Statistics\Log buffer waits\Cumulative wait time (ms) per second | Máximo 2 segundos |
| MSSQL$SQL2K8R2:Memory Manager\Memory Grants Pending | Máximo 2 segundos |
| MSSQL$SQL2K8R2:Wait Statistics\Page latch waits\Cumulative wait time (ms) per second | Máximo 2 segundos |
| MSSQL$SQL2K8R2:Wait Statistics\Wait for the worker\Cumulative wait time (ms) per second | Máximo 0.2 segundos |
| MSSQL$SQL2K8R2:Wait Statistics\Network IO waits\Cumulative wait time (ms) per second | Máximo 0.5 segundos |
| MSSQL$SQL2K8R2:Longest Transaction Running Time | Máximo 5 segundos |
| MSSQL$SQL2K8R2:Wait Statistics\Lock waits\Cumulative wait time (ms) per second | Máximo 2 segundos |
| \\.\root\Microsoft\SqlServer\ServerEvents\SQL2K8R2\BLOCKED_PROCESS_REPORT | Máximo 5 segundos de duración |
| \\.\root\Microsoft\SqlServer\ServerEvents\SQL2K8R2\DEADLOCK_GRAPH | Máximo 0 en cualquier momento |

**Tabla 2. Definición de los cantadores de performance y sus umbrales**

Cualquier desviación en estas métricas provoca que se capture la información relevante del evento.

## 3.2 Definición de malas prácticas de codificación para T-SQL

Del conjunto de las sentencias del lenguaje T-SQL, el cual es usado por SQL Server, se definieron las siguientes malas prácticas para el ejemplo de la implementación. El listado no está limitado y es extensible conforme se documenten otros patrones:

- Solicitar al RDBMS sólo la información se requiere en una aplicación
    - No usar SELECT *
    - Usar clausulas WHERE para limitar la extracción de información
- No examinar el mismo conjunto de datos o subconjuntos de este más de una vez dentro de una transacción o batch
- No crear objetos temporales que sean subconjuntos de datos de tablas ya existentes
    - Evitar el uso de la definición de tablas temporales
- No forzar comportamientos en el optimizer sin tener justificación
    - No forzar la elección explícita de índices para satisfacer alguna sentencia
- No construir código que no permita usar las estructuras de datos existentes para reducir el número de lecturas o iteraciones
    - No usar funciones en la cláusula WHERE para la evaluación de columnas
- No usar construcciones de código que no permita la reutilización del plan de ejecución generado
    - No usar el modificador WITH RECOMPILE para forzar la recompilación de planes de ejecución
- No construir transacciones que no estén limitadas a la cantidad mínima de sentencias requeridas para completarse satisfactoriamente
- Verificar la cerradura de sentencias inicializadoras que requieren de la ejecución de sentencias destructoras
    - Todo DECLARE CURSOR irá acompañado de la correspondiente sentencia CLOSE CURSOR y DEALLOCATE CURSOR
- No usar la implementación de SQL como una extensión de programación estructurada u orientada a objetos

- No usar ciclos iterativos como es el caso del uso de cursores

## 3.3 Aplicación proactiva

### 3.3.1 Análisis en ambiente simulado o basado en estadísticas

La aplicación preventiva de la metodología se hará en un servidor de desarrollo. El mismo equipo servirá para analizar los nuevos objetos y los desarrollos más recientes del código que se pasarán al ambiente productivo. Todos los análisis son aplicables de forma proactiva, con excepción del de concurrencia ya que un servidor de pruebas no tendrá un nivel de concurrencia que permita obtener información para el análisis, sin embargo si hay herramientas que ayudan a simular carga y permiten obtener información similar a la del ambiente productivo. Uno de esos productos es el SQLIO Stress Test Tool.

### 3.3.2 Análisis proactivo de código con autómatas

Se automatizaron las búsquedas de malas prácticas que no requieren de la revisión de un DBA o de un desarrollador. Se generaron expresiones regulares, autómatas de pila y algoritmos para la automatización. Es en este punto que se detallará la implementación de los autómatas para cada

caso. Para una fácil implementación sólo se mostrarán las expresiones regulares que se usaron, así como algunos grafos para definir las máquinas de estados y el código o pseudocódigo utilizados. En este ejercicio se utilizaron las expresiones regulares derivadas de la librería System.Text.RegularExpressions del Microsoft .NET Framework[17].

#### 3.3.2.1 Omitir texto no relevante al análisis

El primer paso en la revisión automática del código es omitir todo aquel texto que no es relevante al análisis. Para el caso de SQL Server se consideró que los comentarios no son sujetos del análisis. La implementación del lenguaje SQL reconoce 2 tipos de comentarios, los comentarios de única línea (singleline) y los comentarios multilínea (multiline).

Los comentarios del tipo singleline son los que se definen con 2 guiones medios escritos consecutivamente y que a partir de la primera ocurrencia de estos caracteres en una línea el resto se considera un comentario. Una vez que se localizó la expresión es fácil omitirla. La expresión regular que se definió es:

| Expresión regular | Ejemplo |
|---|---|
| `--.*` | `SELECT * -- REQUERIMOS--TODO (*)`<br>`FROM TABLA` |

El caso de los comentarios multiline es más complejo ya que su estructura es la de una secuencia de caracteres que marca el inicio y otra secuencia diferente la que marca el final, además entre la secuencia de inicio y la secuencia del final se encontrará cualquier combinación de caracteres o inclusive comentarios multiline anidados, es por ello que para este caso fue más práctico usar un autómata de pila.

***NOTA: Es importante que primero se aplique la remoción de comentarios multiline para después aplicar la remoción de comentarios singleline.***

El autómata de pila para encontrar y remover comentarios multiline de sentencias T-SQL está definido de la forma $M = (Q, \sum, \Gamma, q_0, Z_0, A, \delta)$ donde se encuentran los siguientes elementos:

- El conjunto de los estados de la máquina
  - $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$
- El alfabeto del lenguaje que la máquina reconoce
  - $\sum = \{/, *, .\}$
  - El carácter "." significa cualquier dígito en expresiones regulares
- El alfabeto de las cadenas que son aceptables en la pila
  - $\Gamma = \{ Z_0, 1, \lambda\}$

[17] Las expresiones regulares, los autómatas y los segmentos de código para este capítulo se extrajeron de la Obra protegida por el Registro Público del Derecho de Autor, No. de Registro 03-2012-031212472500-01, otorgado a Ismael Medina Muñoz el 30 de Marzo del 2012.

- $q_0$ es el estado inicial, elemento de Q
- $Z_0$ es el símbolo inicial en la pila, elemento de Γ
- El conjunto de los estados de aceptación de la máquina
    - $A = \{q_1\}$
- Función de transición de la máquina
    - $\delta: Q\ X\ (\sum \cup \{\lambda\}) \longrightarrow 2^Q \therefore \delta:(p, a, X) = (q, \alpha)$

En ese contexto tendríamos la definición de la función de transición para cada estado como se lee en la tabla siguiente:

- $\delta:(p, a, X) = (q, \alpha)$
    - A = $\delta(q_0,\lambda,Z_0)=(q_1,Z_0)$
    - B = $\delta(q_1,.,Z_0)=(q_1,Z_0)$
    - C = $\delta(q_1,/,Z_0)=(q_2,Z_0)$
    - D = $\delta(q_2,*,Z_0)=(q_3,1Z_0)$
    - E = $\delta(q_3,.,1)=(q_3,1)$
    - F = $\delta(q_3,/,1)=(q_4,1)$
    - G = $\delta(q_4,.,1)=(q_3,1)$
    - H = $\delta(q_4,*,1)=(q_5,11)$
    - I = $\delta(q_5,\lambda,1)=(q_3,1)$
    - J = $\delta(q_3,*,1)=(q_6,1)$
    - K = $\delta(q_6,.,1)=(q_3,1)$
    - L = $\delta(q_6,/,1)=(q_7,\lambda)$
    - M = $\delta(q_7,\lambda,1)=(q_3,1)$
    - N = $\delta(q_3,\lambda,Z_0)=(q_1,Z_0)$
    - O = $\delta(q_2,.,Z_0)=(q_1,Z_0)$

El siguiente diagrama de estados muestra cómo se comportaría la máquina dada la función de transición y la lista anterior:

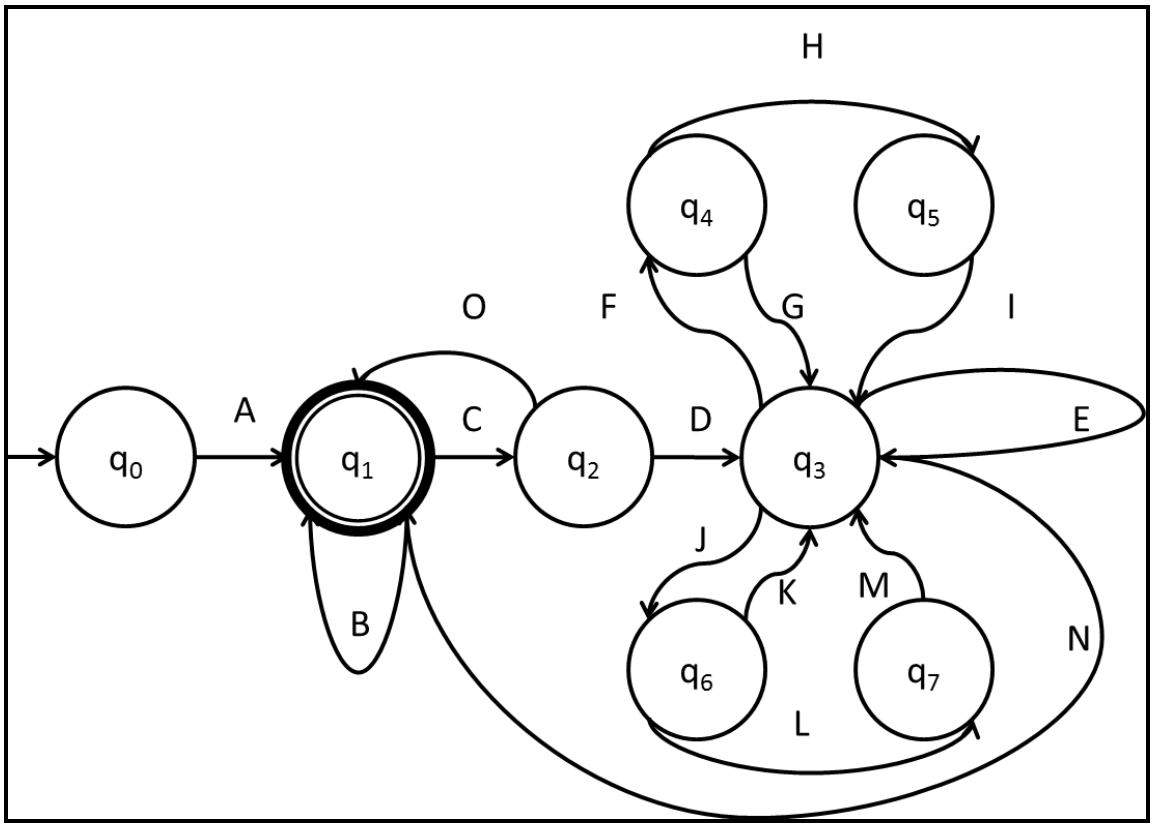



Ilustración 3-1 Autómata de pila que reconoce comentarios multiline de T-SQL

Con este autómata se implementó el siguiente pseudocódigo para eliminar los comentarios multiline en cuyo interior habrá 0, 1 o más comentarios anidados. Una vez que se determinó el inicio y el fin del comentario más externo se marcarán ambos puntos y se recortará el texto en ese rango, esta misma función se usa una única vez en el mismo texto para eliminar todos los comentarios multiline.

```
string RemoveMultilineComments(string input)
{
      Q = {q0, q1, q2, q3, q4, q5, q6, q7};
      ∑ = {'/', '*', '.', λ};
      Γ = {0, 1, λ}; //0 == Z0
      A = {q1};

      CurrentPos = 0; //posición de la cabeza lectora
      BeginPos = 0;
      EndPos = 0;

      // A = δ(q0,λ,Z0)
      ActualState = q1;
      Stack = 0;

      foreach (char i in input){
```

```
 CurrentPos = CurrentPos + 1;
if (ActualState == q1 && i=='.')
{// B = δ(q1,.,Z0)= (q1,Z0)
      ActualState = q1;}
else if (ActualState == q1 && i=='/')
{//C = δ(q1,/,Z0)= (q2,Z0)
      ActualState = q2;}
else if (ActualState == q2 && i=='*')
{//D = δ(q2,*,Z0)= (q3,1Z0)
      ActualState = q3;
      Stack = Stack + 1;
      BeginPos = CurrentPos - 2;    }
else if (ActualState == q3 && i=='.')
{//E = δ(q3,.,1)= (q3,1)
      ActualState = q3;}
else if (ActualState == q3 && i=='/')
{//F = δ(q3,/,1)= (q4,1)
      ActualState = q4;}
else if (ActualState == q4 && i=='.')
{//G = δ(q4,.,1)= (q3,1)
      ActualState = q3;}
else if (ActualState == q4 && i=='*')
{//H = δ(q4,*,1)= (q5,11)
      ActualState = q5;
      Stack = Stack + 1;
 //I = δ(q5,λ,1)= (q3,1)
      ActualState = q0;}
else if (ActualState == q3 && i=='*')
{//J = δ(q3,*,1)= (q6,1)
      ActualState = q6;}
else if (ActualState == q6 && i=='.')
{//K = δ(q6,.,1)= (q3,1)
      ActualState = q3;}
else if (ActualState == q6 && i=='/')
{//L = δ(q6,/,1)= (q7,λ)
      ActualState = q7;
      Stack = Stack - 1;
 //M = δ(q7,λ,1)= (q3,1)
      ActualState = q3;}
else if (ActualState == q3 && Stack==0)
{//N = δ(q3,λ,Z0)= (q1,Z0)
      ActualState = q1;
      EndPos = CurrentPos - 1;
      input = Trim(input, BeginPos, EndPos);
      CurrentPos = BeginPos;
      BeginPos = EndPos = 0;}
else if (ActualState == q2 && i=='.')
{//O = δ(q2,.,Z0)= (q1,Z0)
```

```
                    ActualState = q1;}
            }
            return input;
        }
```

Con este algoritmo se depura el código que es insumo del resto de los análisis.

#### 3.3.2.2 Detectar el uso de SELECT *

La detección de las sentencias SELECT * se hizo usando una expresión regular que permite encontrar incluso los patrones más intrincados. La expresión regular y los ejemplos se describen a continuación:

| Expresión regular | Ejemplos |
|---|---|
| `(\b)SELECT(\b)(\s+\|.*(,\s*)\|.*(\S+\.))\*` | `SELECT * FROM A` |
| | `SELECT a,b,*,c,d FROM (SELECT * FROM )` |
| | `SELECT a*b FROM A` |
| | `SELECT x.* FROM A` |
| | `SELECT a,b,x.*,c,d FROM A` |

#### 3.3.2.3 Detectar la creación de tablas temporales

Para la detección del uso de tablas temporales se usó una única expresión regular para el análisis. La expresión regular y sus ejemplos se muestran a continuación:

| Expresión regular | Ejemplos |
|---|---|
| `((CREATE\s+TABLE)\|INTO)\s+\s*#+\S` | `CREATE TABLE ##Pruebas` |
| | `CREATE TABLE @@Text` |
| | `CREATE TABLE @One` |
| | `CREATE TABLE #Rght` |
| | `SELECT * INTO #Este` |
| | `SELECT * INTO ##AhoraEste FROM EEEE` |

#### 3.3.2.4 Detectar el uso de cursores y su uso correcto

Referente a los cursores en SQL Server, se evalúa primero que se haga uso de cursores dentro del código y después se valida que el cursor sea destruido y desalojado de la memoria si es que el cursor en algún momento fue abierto. Para ello se usó una expresión regular que obtiene los tokens de las sentencias siguientes:

- DECLARE identificador … CURSOR
- OPEN … identificador
- CLOSE … identificador
- DEALLOCATE … @Object

Con estos tokens se forma una cinta que es procesada en este algoritmo. Dicho algoritmo permite encontrar la paridad de las sentencias para cada cursor:

```
(\bDECLARE\s+@?\w+\s+(INSENSITIVE\s+)?(SCROLL\s+)?CURSOR\b)|(\bOPEN\s+(GL
OBAL\s+)?@?\w+)|(\bCLOSE\s+(GLOBAL\s+)?@?\w+)|(\bDEALLOCATE\s+(GLOBAL\s+)
?@?\w+)
```

Las consideraciones de la paridad son:

- Un cursor es definible como una variable del tipo *@variable*
- Un cursor es definible como un identificador del tipo *identificador*
- Sólo es necesario que exista una sentencia CLOSE si se ha presentado una sentencia OPEN para el mismo cursor
- Si el cursor fue definido como una variable no es necesario que exista una sentencia DEALLOCATE
- Habrá construcciones de código con una sola sentencia OPEN y múltiples sentencias CLOSE y / o DEALLOCATE, esto como resultado de estructuras de programación como las de control de errores, sin embargo en este caso sólo se reporta la disparidad de los elementos y únicamente se hablará de indicios de problemas cuando no existan sentencias CLOSE para cursores abiertos

Los tokens generados por la expresión regular son usados para encontrar la paridad de las sentencias, para ello es necesario construir varias pilas que mantienen el control del análisis de cada uno de los cursores declarados en código. Todas estas consideraciones hacen imposible la implementación de un solo autómata de pila y es por ello que se implementó más bien un algoritmo de análisis.

Se usó para el algoritmo una clase que permite llevar una bitácora de los cursores detectados y de cuantas sentencias OPEN, CLOSE y DEALLOCATE se contabilizan en el código por cada cursor declarado. Esta es la implementación del código de la clase de mantiene la bitácora de contabilización de operaciones para un CURSOR. Está escrito en el lenguaje C# del Microsoft .NET Framework:

```
class cursorStatements
{
      string cursorName = "";
      bool isVariable = false;
      int open = 0;
      int close = 0;
      int deallocate = 0;
      public cursorStatements(string cursorName)
      {
            this.cursorName = cursorName.Trim(' ');
            if (this.cursorName.StartsWith("@"))
                  isVariable = true;
      }
      public int OPEN
      {
```

```
                get{return open;}
                set{open = value;}
            }
            public int CLOSE
            {
                get{return close;}
                set{close = value;}
            }
            public int DEALLOCATE
            {
                get{return deallocate;}
                set{deallocate = value;}
            }
            public bool IsVariable
            {
                get{return isVariable;}
            }
        }
```

Con la clase definida se implementó el siguiente algoritmo para la hacer la contabilización de las sentencias que han afectado a un cursor. Para cada cursor declarado se generará una clase Dictionary que controla las pilas de sentencias OPEN, CLOSE y DEALLOCATE. :

```
        void CursorUsageAnalysis(string input)
        {
            MatchCollection mcStream = Regex.Matches(rtxtInput.Text,
                                txtRegexPattern.Text,
                            RegexOptions.IgnoreCase);
            foreach (Match token in mcStream)
            {
                rtxtStream.Text += token.Value + "\n";
            }
                Dictionary<string, cursorStatements> cursorAnalysis =
new
                        Dictionary<string, cursorStatements>();

            foreach (Match token in mcStream)
            {
                if (token.Value.TrimStart(' ').StartsWith("DECLARE"))
                {
                    string [] tokenIdentifier = token.Value.TrimStart
                                                (' ').Split(' ');
                    cursorAnalysis.Add(tokenIdentifier[1], new
                    cursorStatements(tokenIdentifier[1])); //elemento
                                                        //identificador
                }
                if (token.Value.TrimStart(' ').StartsWith("OPEN"))
                {
                    string [] tokenIdentifier = token.Value.TrimStart
```

```
                                          (' ').Split(' ');
                        cursorAnalysis[tokenIdentifier
                              [tokenIdentifier.Length - 1]].OPEN += 1;
                  }
                  if (token.Value.TrimStart(' ').StartsWith("CLOSE"))
                  {
                        string [] tokenIdentifier = token.Value.TrimStart
                                          (' ').Split(' ');
                        cursorAnalysis[tokenIdentifier
                              [tokenIdentifier.Length - 1]].CLOSE += 1;
                  }
                  if (token.Value.TrimStart('
').StartsWith("DEALLOCATE"))
                  {
                        string [] tokenIdentifier = token.Value.TrimStart
                                          (' ').Split(' ');
                        cursorAnalysis[tokenIdentifier
                              [tokenIdentifier.Length - 1]].DEALLOCATE +=
                        1;
                  }
            }
            foreach (KeyValuePair<string, cursorStatements> element
                                    in cursorAnalysis)
            {
                  if (element.Value.IsVariable)
                        MessageBox.Show ("OPEN = " +
                              element.Value.OPEN.ToString() +
                                    "\nCLOSE = " +
                              element.Value.CLOSE.ToString(),element.Key);
                  else
                        MessageBox.Show ("OPEN = " +
                              element.Value.OPEN.ToString() +
                                    "\nCLOSE = " +
                              element.Value.CLOSE.ToString() +
                                    "\nDEALLOCATE = " +
                              element.Value.DEALLOCATE.ToString(),element.
                              Key);
            }
      }
```

#### *3.3.2.5 Detectar la indicación explícita del uso de índices*

Le detección de la indicación explícita de los índices se hizo con una expresión regular que es diferente en SQL Server 2000 respecto a la implementación en SQL Server 2005 y posterior. La expresión regular que sirve para detectar el uso de índices en SQL Server 2000 y SQL Server 2005 o posteriores es la siguiente:

| Expresión regular | Ejemplo |
|---|---|
| (((WITH)?\s*\(?\s*(?<![@_a-zA- | WITH ( A,1,2, INDEX= |

| | |
|---|---|
| Z])INDEX\>)\|((WITH)?\s*\(?\s*)(\s*\S*\s*,)+\s*(?<![@_a-zA-Z])INDEX\>)\|((?<![@_a-zA-Z])INDEX\>\s*=)\|((?<![@_a-zA-Z])INDEX\>\s*)\|((?<![@_a-zA-Z])INDEX\>\s*\(() | wITH INDEX= |
| | WITH 1, HOLDLOCK, RUN, INDEX, PASTO |
| | INDEX = , PASTO |
| | INDEX () |
| | WITH ( INDEX |
| | WITH CHAR_INDEX |
| | CHAR_INDEX () |
| | INDEX |
| | REINDEX |

### *3.3.2.6 Detectar la indicación explícita de la recompilación*

La detección de la recompilación explícita en objetos programáticos de SQL Server se hizo con la implementación de una expresión regular. La mencionada expresión regular y los ejemplos de implementación se describen en el cuadro debajo.

| Expresión regular | Ejemplo |
|---|---|
| \s+WITH\s+\S*\s?RECOMPILE | WITH CHAR_INDEX, RECOMPILE |
| | WITH CHAR_INDEX, RECOMPILE |
| | WITH RECOMPILE |

### *3.3.2.7 Detectar sentencias sin WHERE*

Para la detección de sentencias sin WHERE se creó un autómata de pila que recibe una cinta compuesta por los tokens de las sentencias del lenguaje que aceptan el uso de la cláusula WHERE. El autómata detecta sentencias SELECT anidadas.

Se usó una expresión regular para extraer los tokens del texto que se va a analizar y con ellos se armó la nueva cinta que es el insumo del siguiente autómata, la expresión regular que sirve para exponer los tokens es la siguiente:

```
(\b)SELECT(\b)|(\b)UPDATE(\b)|(\b)SET(\b)|(\b)DELETE(\b)|(\b)FROM(\b)|(\b)WHERE(\b)|\(|\)
```

Una vez que se obtienen los tokens, se alojan todos ellos en una cinta. Los elementos almacenables en la cinta son:

- SELECT
- UPDATE
- DELETE
- SET
- FROM
- WHERE
- (
- )

La nueva cinta es el insumo para el análisis. Dentro de la codificación T-SQL se permite la codificación de SELECTs anidados dentro de otros SELECTs. Otra utilidad del SELECT es que el conjunto resultante acota el conjunto al cual se aplica una operación UPDATE o DELETE, lo que da lugar al anidamiento de sentencias SELECT dentro de un UPDATE o un DELETE. La capacidad de anidamiento de las sentencias SELECT hace que el autómata sea complejo ya que es necesario que sea capaz de leer cintas con ese patrón. El autómata que se definió es un autómata de pila y en un grupo de las funciones de transición se implementó una serie de banderas para indicar si se han encontrado sentencias SELECT, UPDATE o DELETE sin WHERE. La definición matemática del autómata de la forma $M = (Q, \sum, \Gamma, q_0, Z_0, A, \delta)$ es la siguiente:

- El conjunto de los estados de la máquina
  - $Q = \{q_0, q_1, q_2, q_3, q_4\}$
- El alfabeto del lenguaje que la máquina reconoce
  - $\sum = \{SELECT, UPDATE, DELETE, SET, WHERE, (, )\}$
- El alfabeto de las cadenas que reconoce la pila
  - $\Gamma = \{ Z_0, s, S, p, u, U, d, D, 1, \lambda\}$
- $q_0$ es el estado inicial, elemento de Q
- $Z_0$ es el símbolo inicial en la pila, elemento de Γ
- El conjunto de los estados de aceptación de la máquina
  - $A = \{q_4\}$
- Función de transición de la máquina
  - $\delta: Q\ X\ (\sum \cup \{\lambda\}) \longrightarrow 2^Q \therefore \delta:(p, a, X) = (q, \alpha)$

En ese contexto tendríamos la definición de la función de transición para cada estado como se lee en la tabla siguiente:

- $\delta:(p, a, X) = (q, \alpha)$
  - $\delta(q_1, (, Z_0) = \delta(q_1 , 1)$
  - $\delta(q_1, SELECT, 1) = \delta(q_1 , s1)$
  - $\delta(q_1, SELECT, Z_0) = \delta(q_1 , sZ_0)$
  - $\delta(q_1, SELECT, s ) = \delta(q_1 , s)$
  - $\delta(q_1, SELECT, u ) = \delta(q_1 , s)$
  - $\delta(q_1, SELECT, U ) = \delta(q_1 , s)$
  - $\delta(q_1, SELECT, d ) = \delta(q_1 , s)$
  - $\delta(q_1, SELECT, D ) = \delta(q_1 , s)$
  - $\delta(q_1,(, s ) = \delta(q_1 , 1s)$
  - $\delta(q_1, FROM, s ) = \delta(q_1 , S)$
  - $\delta(q_1,(, S ) = \delta(q_1 , 1S)$
  - $\delta(q_1,(, 1 ) = \delta(q_1 , 11)$
  - $\delta(q_1, FROM, u ) = \delta(q_1 , U)$
  - $\delta(q_1,(, u ) = \delta(q_1 , 1u)$

- $\delta(q_1,(, U ) = \delta(q_1 , 1U)$
- $\delta(q_1, DELETE, Z_0 ) = \delta(q_1 , dZ_{0)}$
- $\delta(q_1, DELETE, d ) = \delta(q_1 , d)$
- $\delta(q_1, FROM, d ) = \delta(q_1 , D)$
- $\delta(q_1, FROM, D ) = \delta(q_1 , D)$
- $\delta(q_1,), 1 ) = \delta(q_1 , \lambda)$
- $\delta(q_1, WHERE, S ) = \delta(q_1 , \lambda)$
- $\delta(q_1, SELECT, S ) = \delta(q_1 , \lambda)$
- $\delta(q_1, WHERE, U ) = \delta(q_1 , \lambda)$
- $\delta(q_1, WHERE, u ) = \delta(q_1 , \lambda)$
- $\delta(q_1, DELETE, D ) = \delta(q_1 , d)$
- $\delta(q_1, DELETE, S ) = \delta(q_1 , d)$
- $\delta(q_1, DELETE, s ) = \delta(q_1 , d)$
- $\delta(q_1, DELETE, U ) = \delta(q_1 , d)$
- $\delta(q_1, DELETE, u ) = \delta(q_1 , d)$
- $\delta(q_1, WHERE, D ) = \delta(q_1 , \lambda)$
- $\delta(q_1, WHERE, d ) = \delta(q_1 , \lambda)$
- $\delta(q_1,), s ) = \delta(q_2 , \lambda)$
- $\delta(q_1,), S ) = \delta(q_2, \lambda)$
- $\delta(q_2, \lambda, 1 ) = \delta(q_1 , \lambda)$
- $\delta(q_3, SET, p ) = \delta(q_1 , u)$
- $\delta(q_1, UPDATE, Z_0 ) = \delta(q_1 , pZ_0)$
- $\delta(q_1, UPDATE, u ) = \delta(q_3 , p)$
- $\delta(q_1, UPDATE, U ) = \delta(q_3 , p)$
- $\delta(q_1, UPDATE, s ) = \delta(q_3, p)$
- $\delta(q_1, UPDATE, S ) = \delta(q_3 , p)$
- $\delta(q_1, UPDATE, d ) = \delta(q_3 , p)$
- $\delta(q_1, UPDATE, D ) = \delta(q_3 , p)$
- $\delta(q_0, \lambda, Z_0 ) = \delta(q_1 , Z_0)$
- $\delta(q_1, \lambda, Z_0) = \delta(q_4 , \lambda)$
- $\delta(q_1, \lambda, u) = \delta(q_4 , \lambda)$
- $\delta(q_1, \lambda, U) = \delta(q_4 , \lambda)$
- $\delta(q_1, \lambda, d) = \delta(q_4 , \lambda)$
- $\delta(q_1, \lambda, D) = \delta(q_4 , \lambda)$
- $\delta(q_1, \lambda, s) = \delta(q_4 , \lambda)$
- $\delta(q_1, \lambda, S) = \delta(q_4 , \lambda)$

El conjunto de las funciones donde se agregaron las banderas que definen las sentencias que no cuentan con un WHERE es el siguiente:

- $\delta(q_1, SELECT, u ) = \delta(q_1 , s)$

- $\delta(q_1, SELECT, U) = \delta(q_1, s)$
- $\delta(q_1, SELECT, d) = \delta(q_1, s)$
- $\delta(q_1, SELECT, D) = \delta(q_1, s)$
- $\delta(q_1, DELETE, d) = \delta(q_1, d)$
- $\delta(q_1, SELECT, S) = \delta(q_1, \lambda)$
- $\delta(q_1, WHERE, U) = \delta(q_1, \lambda)$
- $\delta(q_1, WHERE, u) = \delta(q_1, \lambda)$
- $\delta(q_1, DELETE, D) = \delta(q_1, d)$
- $\delta(q_1, DELETE, S) = \delta(q_1, d)$
- $\delta(q_1, DELETE, s) = \delta(q_1, d)$
- $\delta(q_1, DELETE, U) = \delta(q_1, d)$
- $\delta(q_1, DELETE, u) = \delta(q_1, d)$
- $\delta(q_1, WHERE, D) = \delta(q_1, \lambda)$
- $\delta(q_1, WHERE, d) = \delta(q_1, \lambda)$
- $\delta(q_1,), S) = \delta(q_2, \lambda)$
- $\delta(q_1, UPDATE, u) = \delta(q_3, p)$
- $\delta(q_1, UPDATE, U) = \delta(q_3, p)$
- $\delta(q_1, UPDATE, s) = \delta(q_3, p)$
- $\delta(q_1, UPDATE, S) = \delta(q_3, p)$
- $\delta(q_1, UPDATE, d) = \delta(q_3, p)$
- $\delta(q_1, UPDATE, D) = \delta(q_3, p)$
- $\delta(q_1, \lambda, u) = \delta(q_4, \lambda)$
- $\delta(q_1, \lambda, U) = \delta(q_4, \lambda)$
- $\delta(q_1, \lambda, d) = \delta(q_4, \lambda)$
- $\delta(q_1, \lambda, D) = \delta(q_4, \lambda)$
- $\delta(q_1, \lambda, s) = \delta(q_4, \lambda)$
- $\delta(q_1, \lambda, S) = \delta(q_4, \lambda)$

La figura 3-2 es un diagrama de estados que muestra cómo se comporta la máquina dada la función de transición y la lista anterior:

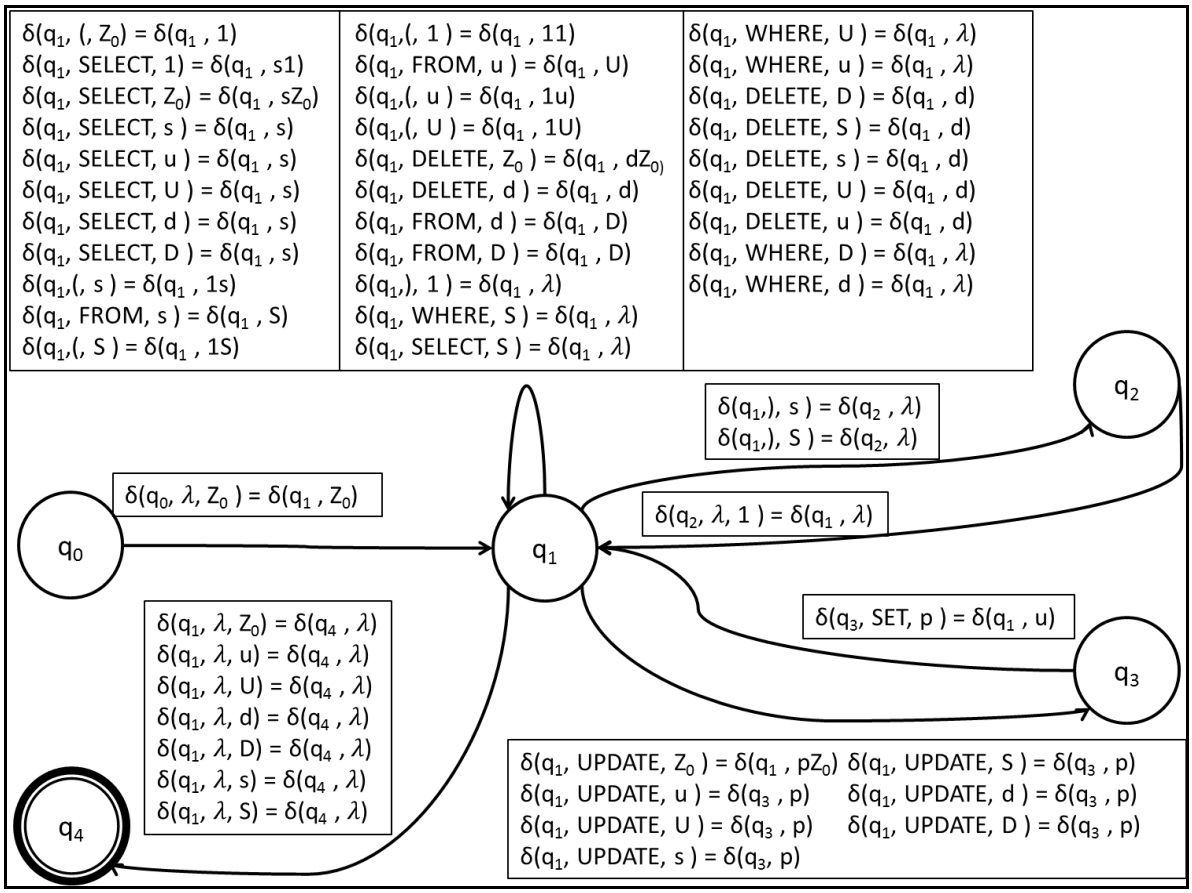



Ilustración 3-2 Autómata de pila que reconoce sentencias SELECT, UPDATE y DELETE

Con la definición del autómata se generó el siguiente código de C# en el cual se tiene la lógica que índica si existe o no alguna sentencia SELECT, UPDATE o DELETE sin WHERE:

```
void PDAStatementWithoutWhere (string input)
{
      //EXTRACCIÓN DE LOS TOKENS PARA LA NUEVA CINTA
      MatchCollection mcStream = Regex.Matches(input,
                                    @"(\b)SELECT(\b)|(\b)UPDATE(\b)|(\b)SE
                              T(\b
                                    )|(\b)DELETE(\b)|(\b)FROM(\b)|(\b)WHER
                              E(\b
                                    )|\(|\)", RegexOptions.IgnoreCase);

      //VARIABLES DE CONTROL DE LA OCURRENCIA DE SENTENCIAS SIN WHERE
      bool wasSELECTfound = false;
      bool wasUPDATEfound = false;
      bool wasDELETEfound = false;
      enum states {q0 = 0, q1, q2, q3, q4}
      // δ(q0, λ, Z0 ) = δ(q1 , Z0)
      string stack = "";
```

```
states currentState = states.q1;

foreach (Match token in mcStream)
{
      if (currentState == states.q1)
      {
            // δ(q1, (, Z0) = δ(q1 , 1)
            if (token.Value.Equals("(") && stack.Equals(""))
                  stack += "(";
            // δ(q1, SELECT, 1) = δ(q1 , s1)
            // δ(q1, SELECT, Z0) = δ(q1 , sZ0)
            else if (token.Value.Contains("SELECT") &&
                     (stack.EndsWith("(") || stack.Equals("") ))
                  stack += "s";
            ////////////////////////////////////////////////////
            /////
            // δ(q1, SELECT, s ) = δ(q1 , s) función que no requiere
            // codificación que no modifica ningún elemento de la
            // máquina
            ////////////////////////////////////////////////////
            /////
            // δ(q1, SELECT, u ) = δ(q1 , s)
            // δ(q1, SELECT, U ) = δ(q1 , s)
            else if (token.Value.Contains("SELECT") &&
                                (stack.EndsWith("u") ||
                                stack.EndsWith("U"))){
                  stack = stack.Remove(stack.Length - 1, 1);
                  wasUPDATEfound = true;
                  stack += "s";
            }
            // δ(q1, SELECT, d ) = δ(q1 , s)
            // δ(q1, SELECT, D ) = δ(q1 , s)
            else if (token.Value.Contains("SELECT") &&
                     (stack.EndsWith("d") || stack.EndsWith("D"))){
                  stack = stack.Remove(stack.Length - 1, 1);
                  wasDELETEfound = true;
                  stack += "s";
            }
            //δ(q1,(, s ) = δ(q1 , 1s)
            //δ(q1,(, S ) = δ(q1 , 1S)
            //δ(q1,(, 1 ) = δ(q1 , 11)
            //δ(q1,(, u ) = δ(q1 , 1u)
            //δ(q1,(, U ) = δ(q1 , 1U)
            else if (token.Value.Contains("(") &&
                  (stack.EndsWith("s") || stack.EndsWith("S") ||
                  stack.EndsWith("(") ||
                  stack.EndsWith("u") || stack.EndsWith("U")))
                  stack += "(";
```

```
                // δ(q1, FROM, s ) = δ(q1 , S)
                else if (token.Value.Contains("FROM") &&
stack.EndsWith("s")){
                      stack = stack.Remove(stack.Length - 1, 1);
                      stack += "S";
                }
                // δ(q1, FROM, u ) = δ(q1 , U)
                else if (token.Value.Contains("FROM") &&
stack.EndsWith("u")){
                      stack = stack.Remove(stack.Length - 1, 1);
                      stack += "U";
                }
                // δ(q1, DELETE, Z0 ) = δ(q1 , dZ0)
                else if (token.Value.Contains("DELETE") &&
stack.Equals(""))
                      stack += "d";
                // δ(q1, DELETE, d ) = δ(q1 , d)
                else if (token.Value.Contains("DELETE") &&
                stack.EndsWith("d"))
                      wasDELETEfound = true;
                // δ(q1, FROM, d ) = δ(q1 , D)
                else if (token.Value.Contains("FROM") &&
stack.EndsWith("d")){
                      stack = stack.Remove(stack.Length - 1, 1);
                      stack += "D";
                }
                ////////////////////////////////////////////////////
                /////
                // δ(q1, FROM, D ) = δ(q1 , D) función que no requiere
                // codificación que no modifica ningún elemento de la
                // máquina

                ////////////////////////////////////////////////////
                /////

                // δ(q1,), 1 ) = δ(q1 , λ)
                else if (token.Value.Contains(")") &&
stack.EndsWith("("))
                      stack = stack.Remove(stack.Length - 1, 1);
                // δ(q1, WHERE, S ) = δ(q1 , λ)
                else if (token.Value.Contains("WHERE") &&
stack.EndsWith("S"))
                      stack = stack.Remove(stack.Length - 1, 1);
                // δ(q1, SELECT, S ) = δ(q1 , λ)
                else if (token.Value.Contains("SELECT") &&
                            stack.EndsWith("S")){
                      stack = stack.Remove(stack.Length - 1, 1);
                      wasSELECTfound = true;
```

```
                }
                // δ(q1, WHERE, U ) = δ(q1 , λ)
                // δ(q1, WHERE, u ) = δ(q1 , λ)
                else if (token.Value.Contains("WHERE") &&
                         (stack.EndsWith("U") || stack.EndsWith("u"))){
                      stack = stack.Remove(stack.Length - 1, 1);
                      wasUPDATEfound = true;
                }
                // δ(q1, DELETE, D ) = δ(q1 , d)
                else if (token.Value.Contains("DELETE") &&
                            stack.EndsWith("D")){
                      stack = stack.Remove(stack.Length - 1, 1);
                      wasDELETEfound = true;
                      stack += "d";
                }
                // δ(q1, DELETE, S ) = δ(q1 , d)
                // δ(q1, DELETE, s ) = δ(q1 , d)
                else if (token.Value.Contains("DELETE") &&
                         (stack.EndsWith("S") || stack.EndsWith("s"))){
                      stack = stack.Remove(stack.Length - 1, 1);
                      wasSELECTfound = true;
                      stack += "d";
                }
                // δ(q1, DELETE, U ) = δ(q1 , d)
                // δ(q1, DELETE, u ) = δ(q1 , d)
                else if (token.Value.Contains("DELETE") &&
                         (stack.EndsWith("U") || stack.EndsWith("u"))){
                      stack = stack.Remove(stack.Length - 1, 1);
                      wasUPDATEfound = true;
                      stack += "d";
                }
                // δ(q1, WHERE, D ) = δ(q1 , λ)
                // δ(q1, WHERE, d ) = δ(q1 , λ)
                else if (token.Value.Contains("WHERE") &&
                         (stack.EndsWith("D") || stack.EndsWith("d"))){
                      stack = stack.Remove(stack.Length - 1, 1);
                      wasDELETEfound = true;
                }
                // δ(q1,), s ) = δ(q2 , λ)
                else if (token.Value.Contains(")") &&
stack.EndsWith("s"))
                {
                      stack = stack.Remove(stack.Length - 1, 1);
                      currentState = states.q2;
                }
                // δ(q1,), S ) = δ(q2, λ)
                else if (token.Value.Contains(")") &&
stack.EndsWith("S"))
```

```
                    {
                          stack = stack.Remove(stack.Length - 1, 1);
                          wasSELECTfound = true;
                          currentState = states.q2;
                    }
                    // δ(q1, UPDATE, Z0 ) = δ(q1 , pZ0)
                    else if (token.Value.Contains("UPDATE") &&
stack.Equals("")){
                          stack += "p";
                          currentState = states.q3;
                    }
                    // δ(q1, UPDATE, u ) = δ(q3 , p)
                    // δ(q1, UPDATE, U ) = δ(q3 , p)
                    else if (token.Value.Contains("UPDATE") &&
                              (stack.EndsWith("u") || stack.EndsWith("U"))){
                          stack = stack.Remove(stack.Length - 1, 1);
                          stack += "p";
                          wasUPDATEfound = true;
                          currentState = states.q3;
                    }
                    // δ(q1, UPDATE, s ) = δ(q3, p)
                    // δ(q1, UPDATE, S ) = δ(q3 , p)
                    else if (token.Value.Contains("UPDATE") &&
                              (stack.EndsWith("s") || stack.EndsWith("S"))){
                          stack = stack.Remove(stack.Length - 1, 1);
                          stack += "p";
                          wasSELECTfound = true;
                          currentState = states.q3;
                    }
                    // δ(q1, UPDATE, d ) = δ(q3 , p)
                    // δ(q1, UPDATE, D ) = δ(q3 , p)
                    else if (token.Value.Contains("UPDATE") &&
                              (stack.EndsWith("d") || stack.EndsWith("D"))){
                          stack = stack.Remove(stack.Length - 1, 1);
                          stack += "p";
                          wasDELETEfound = true;
                          currentState = states.q3;
                    }
              }
              // δ(q2, λ, 1 ) = δ(q1 , λ)
              if(currentState == states.q2 && stack.EndsWith("(")){
                    stack = stack.Remove(stack.Length - 1, 1);
                    currentState = states.q1;
              }
              // δ(q3, SET, p ) = δ(q1 , u)
              if(currentState == states.q3 && token.Value.Contains("SET")
&&
                    stack.EndsWith("p")){
```

```
                    stack = stack.Remove(stack.Length - 1, 1);
                    stack += "u";
                    currentState = states.q1;
              }
        }
        // δ(q1, λ, Z0) = δ(q4 , λ)
        if (stack.Equals("") || stack.Length == 1)
        {
              if (stack.Equals("u"))
                    wasUPDATEfound = true;
              if (stack.Equals("U"))
                    wasUPDATEfound = true;
              if (stack.Equals("S"))
                    wasSELECTfound = true;
              if (stack.Equals("d"))
                    wasDELETEfound = true;
              if (stack.Equals("D"))
                    wasDELETEfound = true;
              stack = "";
              currentState = states.q4;
        }
}
```

## 3.4 Aplicación reactiva

La implementación del monitoreo se hizo con un componente conocido como SQL Server Agent que es parte del producto y que permite la programación de tareas, la definición de alertas y la configuración de operadores para el envío de notificaciones y alertas por vías como correo SMTP, dirección de dispositivos "pager" y notificaciones con el comando NET SEND, todo como parte de las características de administración de un servidor SQL local.

El agente de SQL Server sirvió para instrumentar alertas mediante la suscripción a eventos generados en los contadores de performance de Windows, WMI y el log de eventos de SQL Server, además se utilizó la funcionalidad de la ejecución de una tarea del SQL Server Agent una vez que la alerta se presentó. También se utilizó el envío de las notificaciones ante la ocurrencia de una alerta. La notificación a múltiples operadores que tomen parte en la resolución de los eventos es otra una característica del SQL Server Agent. Utilizar el SQL Server Agent resultó la opción más natural para la implementación del monitoreo.

Se generaron las alertas definidas en la Tabla 1 del capítulo "3.1 Elección de métricas y umbrales de desempeño". Para más información sobre la generación de alertas y la definición de operadores se sugiere consultar los libros en línea de Microsoft SQL Server[18] en internet.

[18] Microsoft. *Monitor and Respond to Events*. Microsoft SQL Server Books Online. http://msdn.microsoft.com/en-us/library/ms191508.aspx. [consulta: Diciembre 2011]

### 3.4.1 Almacenamiento y notificación de eventos relevantes

Para el almacenamiento de los pormenores de un evento se utilizó la ejecución de tareas del SQL Server Agent cuyo fin es la captura de la información relevante del evento. Los orígenes de los datos son las vistas dinámicas que describen información de las solicitudes que se están entendiendo en SQL Server, además de 2 eventos que SQL Server publica en el WMI para detectar bloqueos y deadlocks, así se obtiene la información detallada de las sesiones[19] que el sistema está atendiendo. El diagrama de las tablas se muestra en la ilustración 3-3, y el mismo esquema de tablas funciona para SQL Server 2005 y cualquier versión posterior.

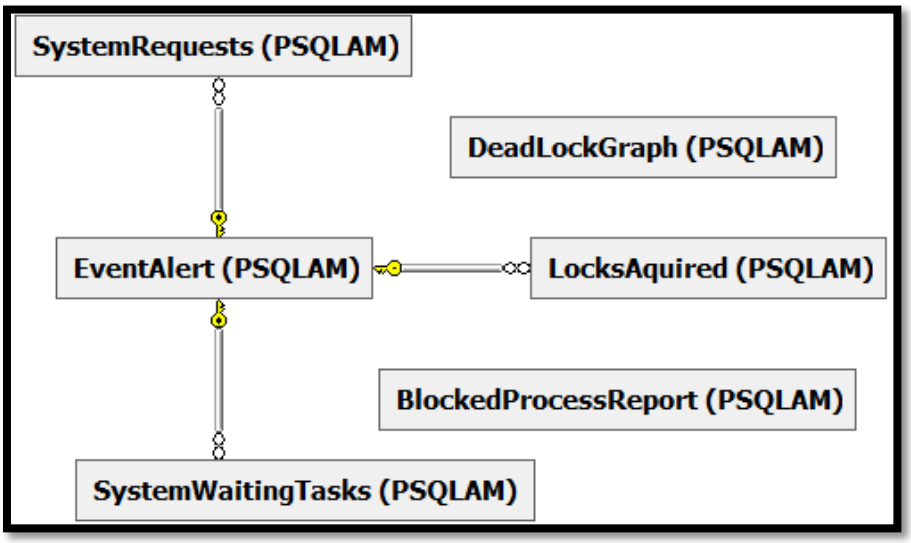



Ilustración 3-3 Modelo de la base de datos de captura de eventos PrecisoSQLAM

- Vistas dinámicas de administración
  - sys.dm_exec_requests

    Esta vista contiene la información de las solicitudes que se están ejecutando en el sistema y también expone información detallada de las lecturas y escrituras de la solicitud así como la información de la espera que dicha solicitud registra, detallando el recurso y el tiempo que ha estado esperando por cada recurso. También provee información del origen y la afectación de un bloqueo ya que esta vista mantiene una referencia hacia sí misma para indicar si la sesión que se está ejecutando está siendo bloqueada por otra sesión en el sistema.

  - sys.dm_exec_sql_text

[19] La interacción que SQL Server mantiene con clientes para la atención de solicitudes se hace a nivel de sesiones de trabajo, entonces todas las solicitudes de usuario se ejecutan desde el contexto de una sesión y cada sesión se ha generado como resultado de una conexión al RDBMS.

Mediante esta se obtiene el código de la sentencia T-SQL que se está ejecutando en la solicitud. Esta información se cruzará con la información de la vista sys.dm_exec_requests para identificar el código T-SQL que se ha ejecutado.

- sys.dm_tran_locks

  Esta vista despliega información de los candados impuestos por las solicitudes ejecutándose actualmente, los recursos asegurados y el tiempo que otras sesiones llevan esperando por algún recurso asegurado.

- sys.dm_os_waiting_tasks

  Esta vista muestra información de los recursos a nivel del sistema operativo que una sesión en curso está esperando, esta vista expone directamente las esperas por los 4 subsistemas de un servidor.

- WMI
  - BLOCKED_PROCESS_REPORT

    Este elemento del WMI se usó para la subscripción al evento a fin de detectar en tiempo real la ocurrencia de un bloqueo. El elemento WMI se consulta para obtener la información del evento de bloqueo en formato XML. Este elemento del WMI reporta bloqueos que tienen más del tiempo definido en la propiedad de la instancia SQL Server que se conoce como "Blocked Process Threshold"[20].

  - DEADLOCK_GRAPH

    Este elemento del WMI se usó para la subscripción a los eventos de deadlock en tiempo real. El elemento se consulta para obtener la información en formato XML del evento con un detalle que solamente SQL Server, ningún producto de monitoreo, proporciona.

Con la información que se obtiene de estos orígenes se diseñó una base de datos que es capaz de almacenar la información proveniente de los objetos WMI y de las DMVs. La base de datos se diseñó para ser almacenada en la misma instancia que está siendo monitoreada y así reducir la necesidad de la adquisición de hardware y del licenciamiento de productos, hecho que ayuda a que el costo de la implementación sea menor comparado con la implementación en una nueva infraestructura. El nombre de la base de datos es PrecisoSQLAM[21]. Todo el contenido de la base de

[20] Microsoft. *blocked process threshold Server Configuration Option*. Microsoft SQL Server Books Online, http://msdn.microsoft.com/es-es/library/ms181150.aspx. [consulta: Diciembre 2011]

[21] Obra protegida por el Registro Público del Derecho de Autor, No. de Registro 03-2011-090513131800-01, otorgado a Ismael Medina Muñoz el 14 de Septiembre del 2011.

datos, los códigos SQL usados para almacenar información y el código de las tareas descritas están protegidos por el Derecho de Autor.

Para la definición de las tablas basadas en DMVs sólo se escogieron las columnas que proveen la información necesaria para llevar a cabo los 5 análisis descritos anteriormente.

Para almacenar la información del objeto DEADLOCK_GRAPH se generó una tabla que contiene una columna del tipo XML para almacenar la información que regresa el objeto WMI.

Para almacenar la información del objeto BLOCKED_PROCESS_REPORT se generó también una tabla que contiene una columna del tipo XML para almacenar la información que regresa el objeto WMI.

El modelo de la base de datos está definido por el siguiente script de T-SQL[22] bajo un esquema de objetos llamado PSQLAM, a continuación se mostrará el detalle de cada tabla dentro de la base de datos y el diagrama de todas ellas en SQL Server.

- Tabla [PSQLAM].[DeadLockGraph]
  La tabla DeadLockGraph es la tabla que sirve para almacenar la información XML de cualquier evento de deadlock. Esta tabla no guarda relación con ninguna otra:

```
CREATE TABLE [PSQLAM].[DeadLockGraph](
      [AlertTime] [datetime] NULL,
      [DeadlockGraph] [xml] NULL
) ON [PRIMARY]
CREATE CLUSTERED INDEX [IX_DeadLockGraph] ON
[PSQLAM].[DeadLockGraph]
(
      [AlertTime] ASC
)
```

- Tabla [PSQLAM].[BlockedProcessReport]
  La tabla BlockedProcessReport sirve para almacenar la información XML de cualquier evento de bloqueos. Esta tabla no guarda relación con ninguna otra:

```
CREATE TABLE [PSQLAM].[BlockedProcessReport](
      [AlertTime] [datetime] NULL,
      [BlockedProcessReport] [xml] NULL
) ON [PRIMARY]
CREATE CLUSTERED INDEX [IX_BlockedProcessReport] ON
[PSQLAM].[BlockedProcessReport]
```

[22] Microsoft SQL Server ha llamado T-SQL o Transact-SQL a la implementación que ha hecho del lenguaje SQL.

```
(
      [AlertTime] ASC
)
```

- Tabla [PSQLAM].[EventAlert]
  La tabla EventAlert es la tabla maestra que almacena la información general de la hora de la ocurrencia de una alerta y la descripción de la alerta en sí. Esta tabla no se ocupa para almacenar la ocurrencia de las alertas disparadas por la captura de eventos mediante los objetos WMI de bloqueos ni deadlocks. La definición de la tabla es la siguiente:

```
CREATE TABLE [PSQLAM].[EventAlert](
      [runtime] [datetime] NOT NULL,
      [alert] [nvarchar](max) NULL,
 CONSTRAINT [PK_EventAlert] PRIMARY KEY CLUSTERED
([runtime] ASC)
) ON [PRIMARY]
```

- Tabla [PSQLAM].[SystemRequests]
  La tabla SystemRequests guarda relación con la tabla EventAlert y es la tabla que sirve para almacenar la información detallada de la actividad de las sesiones que están siendo atendidas actualmente por SQL Server. El detalle incluye el texto del batch que se está ejecutando y de la sentencia dentro del batch que actualmente se está atendiendo. También se almacenan métricas como la duración, lecturas al sistema de IO, escrituras, y si se encuentra esperado por algún recurso para completar la solicitud. La definición de la tabla es la siguiente:

```
CREATE TABLE [PSQLAM].[SystemRequests](
      [runtime] [datetime] NOT NULL,
      [batch] [nvarchar](max) NULL,
      [statement] [nvarchar](max) NULL,
      [session_id] [smallint] NOT NULL,
      [request_id] [int] NOT NULL,
      [start_time] [datetime] NOT NULL,
      [database_name] [nvarchar](128) NULL,
      [user_id] [int] NOT NULL,
      [blocking_session_id] [smallint] NULL,
      [wait_type] [nvarchar](60) NULL,
      [wait_time] [int] NOT NULL,
      [last_wait_type] [nvarchar](60) NOT NULL,
      [wait_resource] [nvarchar](256) NOT NULL,
      [cpu_time] [int] NOT NULL,
      [total_elapsed_time] [int] NOT NULL,
      [reads] [bigint] NOT NULL,
      [writes] [bigint] NOT NULL,
```

```
      [logical_reads] [bigint] NOT NULL,
CONSTRAINT [PK_SystemRequests] PRIMARY KEY CLUSTERED
(
      [runtime] ASC,
      [session_id] ASC,
      [request_id] ASC)
) ON [PRIMARY]
ALTER TABLE [PSQLAM].[SystemRequests]  WITH CHECK ADD
CONSTRAINT [FK_SystemRequests_EventAlert] FOREIGN
KEY([runtime])
REFERENCES [PSQLAM].[EventAlert] ([runtime])
ALTER TABLE [PSQLAM].[SystemRequests] CHECK CONSTRAINT
[FK_SystemRequests_EventAlert]
```

- Tabla [PSQLAM].[LocksAquired]
  La tabla LocksAquired guarda relación con la tabla EventAlert y es la tabla que almacena la información detallada de los candados impuestos actualmente desde todas las sesiones que SQL Server atiende. La definición de la tabla es la siguiente:

```
CREATE TABLE [PSQLAM].[LocksAquired](
      [runtime] [datetime] NOT NULL,
      [resource_type] [nvarchar](60) NOT NULL,
      [database] [nvarchar](128) NULL,
      [blocked_objects] [bigint] NULL,
      [request_mode] [nvarchar](60) NOT NULL,
      [request_session_id] [int] NOT NULL,
      [blocking_session_id] [smallint] NULL
) ON [PRIMARY]
ALTER TABLE [PSQLAM].[LocksAquired]  WITH CHECK ADD
CONSTRAINT [FK_LocksAquired_EventAlert] FOREIGN
KEY([runtime])
REFERENCES [PSQLAM].[EventAlert] ([runtime])
ALTER TABLE [PSQLAM].[LocksAquired] CHECK CONSTRAINT
[FK_LocksAquired_EventAlert]
```

- Tabla [PSQLAM].[SystemWaitingTasks]
  La tabla SystemWaitingTasks guarda relación con la tabla EventAlert y es la tabla que almacena la información detallada de los recursos por los que alguna sesión actual ha estado esperando. La definición de la tabla es la siguiente:

```
CREATE TABLE [PSQLAM].[SystemWaitingTasks](
      [runtime] [datetime] NOT NULL,
      [session_id] [smallint] NOT NULL,
      [wait_duration_ms] [bigint] NULL,
```

```
        [resource_description] [nvarchar](1024) NULL
) ON [PRIMARY]
ALTER TABLE [PSQLAM].[SystemWaitingTasks]  WITH CHECK
ADD  CONSTRAINT [FK_SystemWaitingTasks_EventAlert]
FOREIGN KEY([runtime])
REFERENCES [PSQLAM].[EventAlert] ([runtime])
ALTER TABLE [PSQLAM].[SystemWaitingTasks] CHECK
CONSTRAINT [FK_SystemWaitingTasks_EventAlert]
```

Se aprecia que no se está contemplando la captura del plan de ejecución de ninguna sesión, la razón de esto es que la obtención del plan de ejecución desde el procedure cache toma tiempo, y de cualquier forma el plan de ejecución real o estimado es recuperable posteriormente utilizando el texto [batch] de la sentencia que se ejecutó en la sesión.

La captura de información se hace mediante una tarea programada del agente de SQL Server que se disparará cada vez que se active alguna de las alertas de contadores de performance descritas previamente. Esta tarea no se ejecuta en la captura de eventos del WMI, para ello se implementan 2 tareas separadas. El código que contiene el primer paso de la tarea de captura de eventos de contadores de performance es el siguiente:

```
DECLARE @runtime datetime
SET @runtime = GETDATE()

INSERT INTO [PSQLAM].[EventAlert] --inserción de la alerta
SELECT TOP 1 @runtime, -- que disparó la captura
       sa.[name]
FROM [msdb].[dbo].[sysalerts] sa
INNER JOIN [msdb].[dbo].[sysjobs] sj
  ON sj.[job_id] = sa.[job_id]
WHERE sj.[name] LIKE 'Instance Activity'
ORDER BY sa.[last_response_date] DESC,
sa.[last_response_time] DESC

--System Requests, solicitudes ejecutandose en sistema
INSERT INTO [PSQLAM].[SystemRequests]
SELECT @runtime as [runtime],
        dest.[text] AS [batch],
        SUBSTRING(dest.[text],
der.[statement_start_offset]/2, (
        CASE
                WHEN der.[statement_end_offset] = -1 THEN
LEN(CONVERT(nvarchar(max), dest.[text])) * 2
                ELSE der.[statement_end_offset]
```

```
        END - der.[statement_start_offset]) / 2) AS
[statement],
        der.[session_id],
        [request_id],
        [start_time],
        DB_NAME([database_id]) AS [database_name],
        [user_id],
        [blocking_session_id],
        [wait_type],
        [wait_time],
        [last_wait_type],
        [wait_resource],
        [cpu_time],
        [total_elapsed_time],
        [scheduler_id],
        [reads],
        [writes],
        [logical_reads]
FROM   sys.dm_exec_requests der
CROSS APPLY sys.dm_exec_sql_text (der.[sql_handle]) dest
WHERE  [session_id] <> @@SPID

--Locks Aquired, candados impuestos que provocan bloqueos

INSERT INTO [PSQLAM].[LocksAquired]
SELECT @runtime as [runtime],
        t1.[resource_type],
        db_name([resource_database_id]) AS [database],
        t1.[resource_associated_entity_id] AS
[blocked_objects],
        t1.[request_mode],
        t1.[request_session_id],
        t2.[blocking_session_id]
FROM   sys.dm_tran_locks AS t1,
        sys.dm_os_waiting_tasks AS t2
WHERE  t1.[lock_owner_address] = t2.[resource_address]

--System Waiting Tasks, solicitudes esperando por recursos de
--los subsistemas

INSERT INTO [PSQLAM].[SystemWaitingTasks]
SELECT @runtime as [runtime],
        [session_id],
        [wait_duration_ms],
```

```
        [resource_description]
FROM   sys.dm_os_waiting_tasks
WHERE  [resource_description] IS NOT NULL
```

Para la captura de la información del evento de deadlocks se generó una tarea cuyo primero paso se definió como sigue:

```
INSERT INTO [PSQLAM].[DeadLockGraph]
([AlertTime], [DeadlockGraph])
VALUES (getdate(),N'$(ESCAPE_SQUOTE(WMI(TextData)))')
```

Para la captura del evento de bloqueos se generó una tarea cuyo primero paso se definió como sigue:

```
INSERT INTO [PSQLAM].[BlockedProcessReport]
([AlertTime], [BlockedProcessReport])
VALUES (getdate(), N'$(ESCAPE_SQUOTE(WMI(TextData)))')
```

Con esta base de datos se hará cualquier tipo de consultas que ayuden a identificar las sesiones a las que se aplicará la metodología que ya se expuso. Para la notificación de la ocurrencia de alertas se requiere que se definan operadores a los que se notificará vía mail. Una vez que el operador ha sido notificado del evento se contará con la información requerida para aplicar los análisis expuestos en la metodología.

### 3.4.2 Análisis del consumo y espera por recursos en los subsistemas

La tabla [PSQLAM].[SystemRequests] es la fuente primaria de información para este análisis. De esta tabla se obtiene el "conjunto de sentencias analizables" usando las columnas que se describen debajo:

- [batch]
  Esta columna indica el bloque de sentencias que se ejecutaron en la solicitud
- [statement]
  Esta columna indica la sentencia dentro del batch que se estuvo ejecutando al momento de la captura
- [total_elapsed_time]
  Esta columna indica la duración de la sentencia cuando se capturó la actividad
- [cpu_time]
  Esta columna muestra el uso que la sentencia ha hecho del CPU en milisegundos
- [reads]
  Esta columna muestra la cantidad de lecturas físicas que la sentencia ha solicitado al subsistema de IO
- [writes]
  Esta columna muestra la cantidad de escrituras que la sentencia ha solicitado al subsistema de IO

- [logical_reads]
  Esta columna muestra la cantidad de lecturas lógicas que la sentencia ha solicitado al subsistema de memoria

Por otra parte, el análisis de espera por recursos se hace mediante la visualización de las siguientes columnas de la tabla [PSQLAM].[SystemRequests]:

- [wait_time]
  Esta columna describe el tiempo que la sesión ha estado esperando por algún recurso para completar su trabajo
- [wait_type]
  Esta columna indica de forma general el tipo de recurso que la sesión está esperando
- [wait_resource]
  Esta columna indica de forma general el recurso que la sesión está esperando
- [last_wait_type]
  Esta columna indica de forma general el tipo de recurso por el cual la sesión esperó por última vez
- [blocking_session_id]
  Esta columna muestra el id de la sesión que está bloqueando a esta sesión, si el valor es 0 entonces la sesión no está siendo bloqueada por ninguna otra.

El análisis de espera de recursos en hardware se complementa con la consulta de la tabla [PSQLAM].[SystemWaitingTasks], de la cual se consultarán los registros de forma descendente y tambien estos se agregan al "conjunto de sentencias analizables". Para ello se usa la columna [wait_duration_ms] que ayuda a determinar las esperas con mayor duración. Los detalles de la espera se obtienen usando las siguientes columnas:

- [session_id]
  Indica la sesión que se encuentra esperando por algún recurso, esta tabla muestra las esperas por cualquiera de los 4 subsistemas del servidor
- [wait_duration_ms]
  Indica el tiempo en milisegundos que la sesión ha esperado por el recurso
- [resource_description]
  Detalla el recurso por el cual la sesión ha estado esperando

### 3.4.3 Análisis de calidad de código

Con las mejores prácticas definidas se hace el análisis de código del "conjunto de sentencias analizables" usando las columnas [batch] y [statement] de la tabla [PSQLAM].[SystemRequests].

### 3.4.4 Análisis de planes de ejecución

Del "conjunto de sentencias analizables" se toma cada una de ellas para obtener su plan de ejecución y sobre ellas se aplica el análisis de planes de ejecución.

Para el análisis de los planes de ejecución no se contempló el almacenamiento del plan de ejecución de las sentencias. SQL Server tiene la característica de mostrar el plan de ejecución posteriormente utilizando el texto de las columnas [batch] y [statement] de la tabla [PSQLAM].[SystemRequests] para cada sesión que se analizará.

SQL Server cuenta con 2 formas de obtener el plan de ejecución dado un código T-SQL, el primero es obtener el plan de ejecución estimado mediante el uso de la instrucción SET SHOWPLAN_XML[23], miestras que el otro es obtener el plan de ejecución real que sólo se visualizará una vez que la sentencia T-SQL se ha ejecutado utilizando la instrucción SET STATISTICS XML[24].

Cualquiera de las 2 aproximaciones tiene la bondad de mostrar los planes de ejecución de forma gráfica, tal como se observa en la ilustración 3-4, lo que permite observar cual es el flujo que siguen los operadores dentro del plan[25]. Con estos planes de ejecución se empieza el análisis.

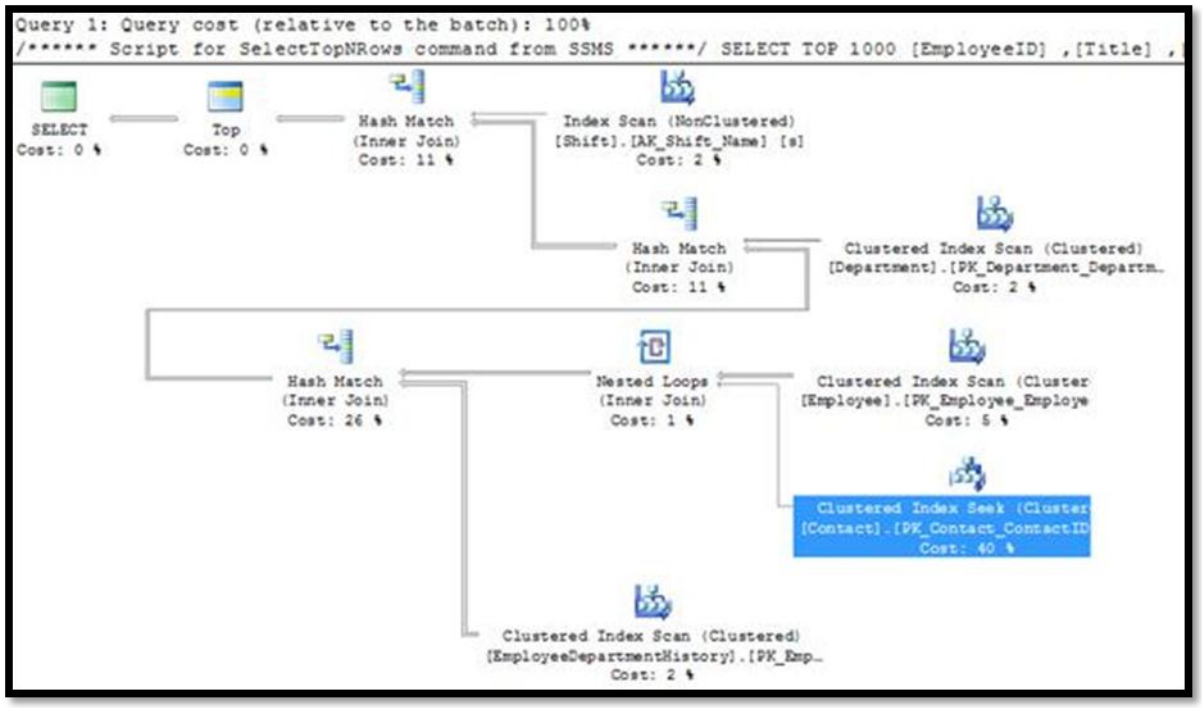



**Ilustración 3-4 Plan de ejecución gráfico que despliega SQL Server mediante la herramienta Management Studio**

[23] Microsoft. *SET SHOWPLAN_XML (Transact-SQL)*. Microsoft SQL Server Books Online. http://msdn.microsoft.com/en-us/library/ms187757.aspx. [consulta: Diciembre 2011]
[24] Microsoft. *SET STATISTICS XML (Transact-SQL)*. Microsoft SQL Server Books Online. http://msdn.microsoft.com/en-us/library/ms176107.aspx. [consulta: Diciembre 2011]
[25] Microsoft. *Displaying Execution Plans by Using the Showplan SET Options*. Microsoft SQL Server Books Online. http://msdn.microsoft.com/en-us/library/ms180765(SQL.105).aspx. [consulta: Diciembre 2011]

### 3.4.5 Análisis de mantenimiento, uso y diseño de estructuras de datos

Con el "conjunto de sentencias analizables" y sus planes de ejecución se hace este análisis para determinar la efectividad de las estructuras de datos evaluando las siguientes prácticas:

- Fragmentación de índices
- La fecha de la última actualización de estadísticas de una tabla
- Un factor de llenado de páginas de datos que no permitan el mejor uso del espacio
- Si la tabla tiene índice clustered
- Si la tabla tiene algún índice non clustered que permita ubicar rápidamente los registros solicitados en la sentencia
- Si los índices se encuentra duplicado parcial o totalmente
- Si la tabla tiene gran cantidad de índices. En este caso toda una transacción no podrá durar más de 5 segundos, entonces será esa la restricción del tiempo máximo de mantenimiento de todos los índices

### 3.4.6 Análisis de concurrencia

Para este análisis se usará el "conjunto de sentencias analizables" que sean causa de bloqueos. Los insumos serán las tablas [PSQLAM].[LocksAquired], [PSQLAM].[SystemRequests], [PSQLAM].[BlockedProcessReport] y [PSQLAM].[DeadlockGraph], con ellas se determinarán los problemas de concurrencia. De la tabla [PSQLAM].[SystemRequests] se usarán las siguientes columnas:

- [wait_type]
  Si la columna indica alguna espera del tipo LCK_% entonces la espera está siendo desencadenada por un lock impuesto sobre algún recurso de datos.
- [wait_time]
  Esta columna describe el tiempo que la sesión ha estado esperando por la liberación del candado.
- [wait_resource]
  Esta columna indica el recurso que la sesión está esperando.
- [blocking_session_id]
  Esta columna mostrará el id de la sesión que es causante del bloqueo para la sesión indicada en la columna [session_id], si el valor es 0 entonces la sesión no está siendo bloqueada por ninguna otra.

La tabla [PSQLAM].[LocksAquired] es otro insumo del análisis y de ella se determinarán más precisamente los candados y recursos por los que han esperado las sesiones que se están ejecutando actualmente, para ello usaremos las siguientes columnas:

- [resource_type]
  Describe el recurso de datos que actualmente tiene un candado impuesto.
- [database]

- Detalla la base de datos a la que pertenece el recurso de datos asegurado.
- [blocked_objects]
  Indica el objeto que está siendo asegurado por el candado.
- [request_mode]
  Indica el tipo de candado que el recurso tiene impuesto.
- [request_session_id]
  Indica la sesión que está solicitando imponer un candado en el recurso.
- [blocking_session_id]
  Indica la sesión que está generando un bloqueo por un candado impuesto en el recurso y que está previniendo que la sesión identificada con el [sesión_id] imponga el candado que ha requerido para continuar.

La tabla [PSQLAM].[BlockedProcessReport] es la tabla que sirve para analizar la ocurrencia de bloqueos y para ello se usará la siguiente columna:

- [BlockedProcessReport]
  Esta columna contiene información detallada del evento en formato XML, dicha información es consultable usando la extensión XQuery que tiene Transact-SQL para explorar este tipo de dato.

La tabla [PSQLAM].[DeadLockGraph] contiene la información detallada de la ocurrencia de un deadlock, usando las siguientes columnas se determinará el origen:

- [DeadlockGraph]
  Esta columna contiene información detallada del evento en formato XML, dicha información se consultara usando la extensión XQuery de Transact-SQL para explorar este tipo de dato.

## 3.5 Automatización de la metodología

La metodología descrita en esta investigación se encuentra parcialmente automatizada para el caso del RDBMS Microsoft SQL Server bajo 2 patentes amparadas por el Registro Público del Derecho de Autor con los números 03-2011-090513131800-01 y 03-2012-031212472500-01. Estas patentes corresponden a las obras ***Preciso SQL Server Activity Monitor*** y ***Vago.SqlServer.Assessment***, respectivamente.

El propósito de la aplicación ***Vago.SqlServer.Assessment***, desarrollada en el lenguaje C# del Microsoft .NET Framework, es aplicar de forma proactiva los análisis de seguridad, de configuración y de desempeño de una instancia de Microsoft SQL Server. Las versiones de Microsoft SQL Server que son analizables por esta aplicación son 2000, 2005, 2008, 2008 R2 y 2012.

Los análisis de seguridad, de configuración y de desempeño que el ***Vago.SqlServer.Assessment*** realiza dan como resultado archivos en formato XML que son usados para la generación de reportes para la presentación de los análisis.

Los análisis de seguridad y de configuración no serán descritos en este documento puesto que no son de interés para esta investigación, sin embargo, se destaca que la aplicación ***Vago.SqlServer.Assessment*** automatiza también estos análisis.

Los análisis que la aplicación ***Vago.SqlServer.Assessment*** automatiza son los siguientes:

- Calidad de código
    - Implementación de autómatas para el análisis
- Mantenimiento, uso y diseño de estructuras de datos

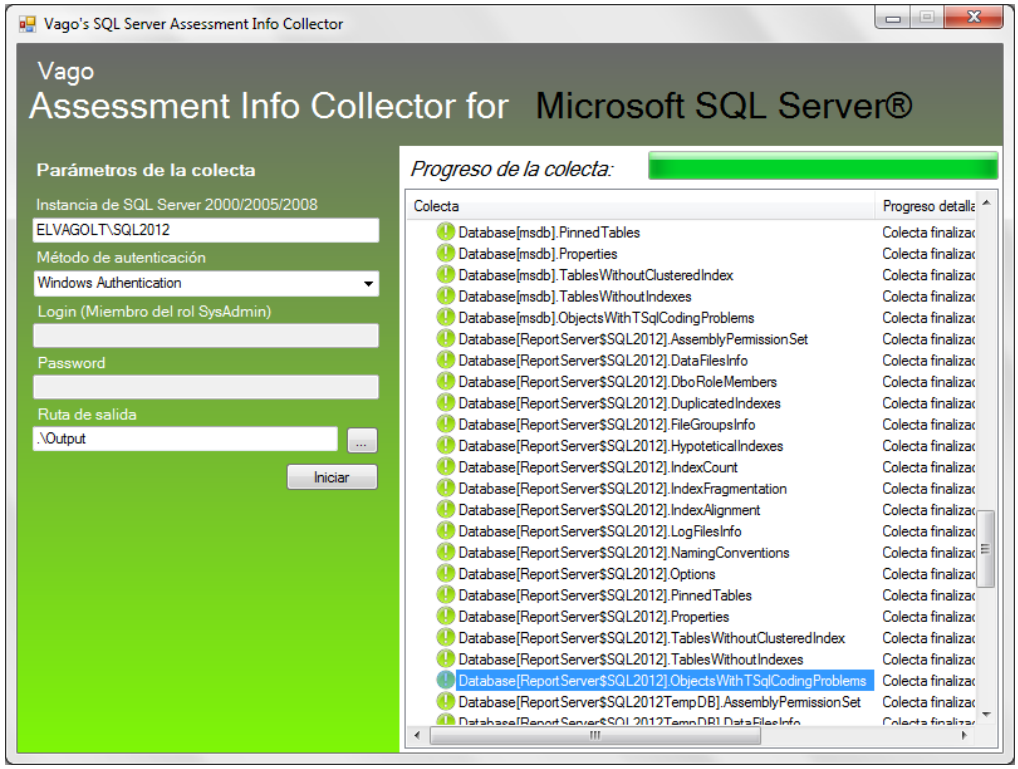



**Ilustración 3-5 Pantalla principal del Vago.SqlServer.Assessment**

La figura 3-5 muestra la pantalla principal de la ejecución de la aplicación ***Vago.SqlServer.Assessment***. La figura 3-6 muestra uno de los archivos XML con el resultado del análisis de calidad de código.

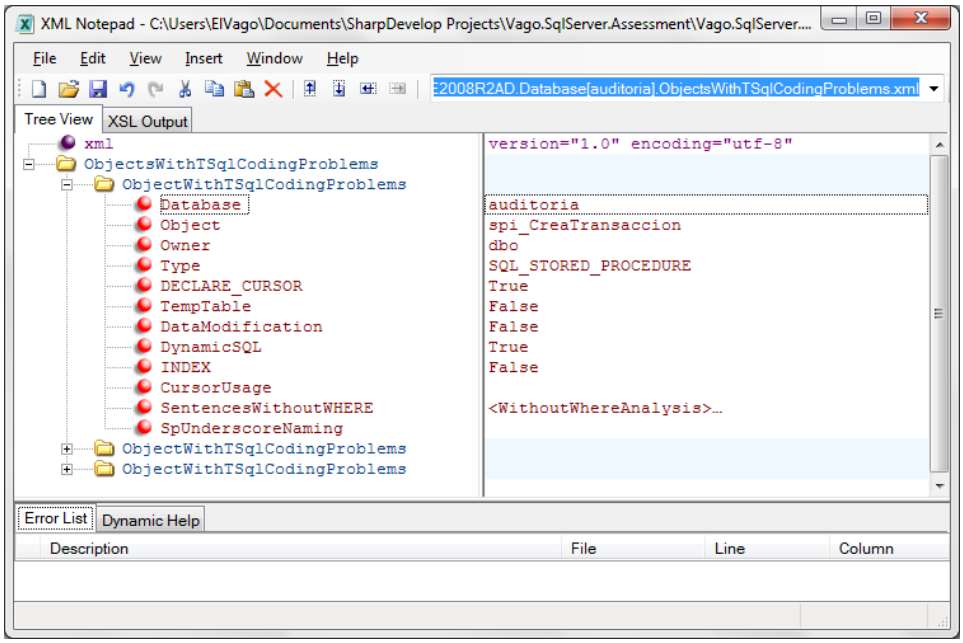



**Ilustración 3-6 Archivo XML con el resultado del análisis de calidad de código**

La aplicación de la metodología se complementa con la colecta de la actividad de la instancia de Microsoft SQL Server durante el periodo donde se presentan las lentitudes típicamente. El resultado de esta colecta se carga en una base de datos dentro de una instancia de Microsoft SQL Server para aplicar los siguientes análisis automatizados:

- Consumo y esperas en los subsistemas
- Concurrencia

El propósito del ***Preciso SQL Server Activity Monitor***, solución implementada como una serie de elementos de configuración y reportes de Microsoft SQL Server, es proporcionar la aplicación reactiva de la metodología. La solución implementa el almacenamiento y notificación automatizada de eventos relevantes, y su punto de partida es la definición de las métricas y los umbrales de desempeño.

Una vez establecidos estos puntos de la metodología, la solución permite la automatización de los siguientes análisis:

- Consumo y esperas en los subsistemas
- Análisis de concurrencia

Con la información almacenada de los eventos relevantes se realizan otros análisis de forma manual.

- Calidad de código
- Planes de ejecución

- Mantenimiento, uso y diseño de estructuras de datos

La figura 3-7 muestra un reporte de bloqueos, parte del análisis de concurrencia que lleva a cabo la solución ***Preciso SQL Server Activity Monitor***. Esta figura corresponde a un reporte obtenido de un servidor de misión crítica para las operaciones de una institución financiera en la que se implementó la solución. El evento corresponde a un problema derivado de la concurrencia y del uso, mantenimiento y diseño de estructuras de datos.

Dicha implementación permitió identificar este y otros problemas de concurrencia dentro del RDBMS. Se concluyó que el problema era causado por varias transacciones de larga duración. La duración de las transacciones era consecuencia de la falta de un índice en una tabla altamente transaccional. La falta de ese índice dificultaba recuperar puntualmente los registros requeridos por una sentencia de actualización que forma parte del código de la transacción.

**Descripción del bloqueo**

Reporte de actividades del servidor B en 1/17/2012 12:48:59 PM

| Alert Time | SQL Blocking Process ID | Client Application | Host Name | Login Name | Status | Isolation Level | Current DB ID | Input Buffer |
|---|---|---|---|---|---|---|---|---|
| 1/17/2012 12:36:57 PM | 209 | .Net SqlClient Data Provider | NBEMSMTP | uperf12 | suspended | read committed (2) | 17 | <inputbuf> Proc [Database Id = 17 Object Id = 1899205866] </inputbuf> |

| SQL Blocked Process ID | Client Application | Host Name | Login Name | Status | Isolation Level | Current DB ID | Input Buffer |
|---|---|---|---|---|---|---|---|
| 235 | .Net SqlClient Data Provider | NBEMSMTP | uperf12 | suspended | read committed (2) | 17 | <inputbuf> Proc [Database Id = 17 Object Id = 1047010811] </inputbuf> |

| Wait Resource | Wait Time | Priority | Transaction Nestling Level | Log Used | Transaction ID | Lock Mode | Scheduler ID |
|---|---|---|---|---|---|---|---|
| PAGE: 17:1:581334 | 16125 | 0 | | 0 | 13331405770 | S | 1 |

| Owner ID | Transaction Name | Transaction Description | SQL Batch ID | Execution Context ID | Process ID | Task Priority |
|---|---|---|---|---|---|---|
| 13331405770 | COND WITH QUERY | 0x3534bc9f8 | 0 | 0 | process38096d8 | 0 |

**Ilustración 3-7 Descripción de un bloqueo, parte del análisis de concurrencia**

Los candados impuestos por esas transacciones se establecían a nivel de la tabla completa y permanecían durante todo el tiempo que la instancia de SQL Server ocupaba para leer la tabla completa a fin de encontrar los registros coincidentes con el criterio de actualización. Estos

candados prevenían que otras transacciones tomaran los datos que requerían por un tiempo considerable.

El análisis permitió mejorar el desempeño de la transacción. Se agregó el índice necesario y con ello se redujo la duración de la transacción de más de 5 segundos a solo 30 milisegundos en promedio, lo que sin duda mejoró el performance y ayudó a la concurrencia.

Extender estas aplicaciones para la automatización de análisis en los RDBMS IBM DB2, MySQL y Oracle implica obtener la información equivalente para cada manejador y definir las métricas y los umbrales para cada RDBM, así como los autómatas equivalentes para cada implementación del lenguaje SQL en cada RDBMS y los reportes necesarios para mostrar los resultados de los análisis.

# Conclusión

La investigación plasmada el marco teórico muestra que es factible comparar y encontrar similitudes en los RDBMS IBM DB2, Oracle, MySQL y SQL Server.

El trabajo que Edgar F. Codd estableció, y que es el fundamento matemático de los manejadores de bases de datos relacionales, es el primero punto clave para establecer una metodología del análisis de desempeño en los RDBMS. El segundo y último punto clave en el establecimiento de la metodología es que la arquitectura de los 4 manejadores de bases de datos relacionales tiene en común algoritmos, estructuras de datos y mecanismos de ejecución de planes. Por lo tanto se concluye categóricamente que se ha alcanzado el objetivo de esta tesis al establecer una metodología del análisis de desempeño para los manejadores de base de datos IBM DB2, Oracle, MySQL y Microsoft SQL Server.

El objetivo particular también se establece como cierto, dado que la implementación del lenguaje SQL para estos 4 RDBMS no está diseñado como un lenguaje multipropósito. La existencia del módulo optimizer en la arquitectura de los RDBMS, en cuyo diseño se encuentra la codificación necesaria para transformar la sentencia SQL a una expresión del álgebra, reitera al lenguaje SQL como un lenguaje orientado a resolver expresiones del álgebra y del cálculo relacional. La adopción del lenguaje SQL como extensión de los lenguajes multipropósito resulta en problemas de desempeño.

La metodología descrita en esta investigación no es única, ya que sólo se abordaron aspectos relacionados a los mecanismos de ejecución de sentencias en los RDBMS sin tomar en cuenta factores como el hardware o los sistemas operativos. Lo que hace factible ampliar la investigación.

El definir la aplicación proactiva y reactiva de la metodología resultó un paso necesario para demostrarla, más aún, la investigación no sólo estableció la metodología como una serie de pasos a seguir, sino que se habla concretamente de una herramienta que la automatiza para el caso del RDBMS Microsoft SQL Server.

La implementación de la metodología guiada por esta investigación ayuda al lector a comprender mejor las operaciones que se llevan a cabo dentro del RDBMS. Dicho conocimiento es útil para promover una utilización más óptima de los RDBMS dentro de una organización.

La implementación de la metodología en el RDBMS Microsoft SQL Server fue una decisión que permitió una implementación rápida, ya que en el campo laboral este es el manejador con el que más me he relacionado.

Definitivamente los conocimientos adquiridos en la licenciatura de Matemáticas Aplicadas y Computación resultan útiles para la compresión de las operaciones de los RDBMS, desde los conocimientos adquiridos en materias como Álgebra Superior, Teoría de la Computación I y Teoría

de la Computación II, Programación y Lenguajes Programación, Estructura de Datos, Base de Datos y Bases de Datos Distribuidas. Todos estos campos del conocimiento facilitaron la investigación.

Otros trabajos con factibilidad de ser derivados de esta investigación son:

- Escenarios de utilización del modo de concurrencia optimista para mejorar el desempeño
- Análisis automatizado de planes de ejecución
- Análisis extenso de código SQL con autómatas
- Estudio y optimización de estructuras de datos útiles para organización de grandes volúmenes de información

# Anexo

## I. Subsistemas manejados por el sistema operativo

Manejadores de bases de datos relacionales como SQL Server, Oracle, DB2 o MySQL (con su motor de almacenamiento InnoDB)[26], están codificados para hacer uso de los recursos disponibles en el hardware y la arquitectura del servidor que los hospeda a fin de obtener un mejor desempeño de las solicitudes de usuarios mientras que mantienen un control transaccional.

Estos manejadores de base de datos relacionales son aplicaciones de cómputo que, como cualquier otra aplicación de cómputo, dependen de un sistema operativo que hace la gestión de los recursos en hardware.

Los sistemas operativos exponen a las aplicaciones, como los manejadores de bases de datos, los mecanismos de solicitud de recursos de hardware y librerías de software que permiten su correcta ejecución, mientras que de forma interna administra la distribución y el acceso a esos recursos de hardware y librerías entre todas las aplicaciones que se encuentran en ejecución y convivencia. Los comportamientos que el sistema operativo registra en la distribución del acceso a los recursos de hardware y librerías de software son ajustables mediante el uso de parámetros que favorecerán o reducirán la capacidad de respuesta de los manejadores de bases de datos relacionales.

Es posible encontrar una capa adicional entre los manejadores de base de datos y el hardware del servidor. Se trata del sistema operativo anfitrión cuando hablamos de ambientes de servidores virtualizados o de una capa de particionamiento de hardware.

Es por ello que los manejadores de bases de datos relacionales no hacen uso de los recursos del servidor de manera directa sino que en su lugar se entienden con lo que en esta investigación se definirá como subsistemas. Cada subsistema es un conjunto de recursos especializados que el sistema operativo administra, y de los cuales expone mecanismos de posicionamiento y atención de solicitudes.

Cada uno de los manejadores mencionados invariablemente hace uso de 4 subsistemas del sistema operativo para la atención de las solicitudes del usuario.

El propósito de este anexo es describir cada uno de los 4 subsistemas a fin de clarificar como los manejadores de bases de datos relacionales hacen uso de ellos.

---

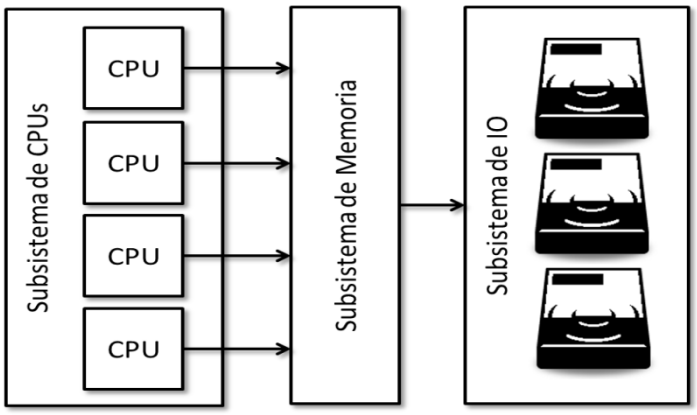
[26] Schwartz, B., Zaitsev, P., Tkachenko, V., Zawodny, Jeremy D., Lentz, A., Balling, Derek J. (2008). *High Performance MySQL (Second Edition)*. United States of America: O'Reilly Media Inc. pp. 305 - 317

## Subsistema de unidades de procesamiento central

Los sistemas operativos fungen, entre otras cosas, como encargados de administrar el poder de procesamiento y para ello ponen a disposición de los manejadores de base de datos relacionales mecanismos para el posicionamiento de solicitudes de procesamiento, ya sean solicitudes de los usuarios o solicitudes de la ejecución de los procesos internos del RDBMS.

El poder de procesamiento del hardware tiene una relación directa con la arquitectura de procesamiento con el que fue diseñado dicho hardware. En algunas de estas arquitecturas los manejadores de base de datos asumen que los CPUs, comparten la misma responsabilidad en el mantenimiento del procesamiento y que estos están físicamente juntos en el mismo hardware en lo que se denomina arquitectura en paralelo.

Las arquitecturas en paralelo representan un intento por construir computadoras centralizadas más rápidas y menos costosas. Un ejemplo de esta arquitectura es la conocida como shared-memory multiprocessor, donde una computadora tiene múltiples procesadores activos de forma simultánea que comparten un acceso único al subsistema de memoria y tienen una interface común al subsistema de IO, tal como se muestra en la ilustración siguiente.



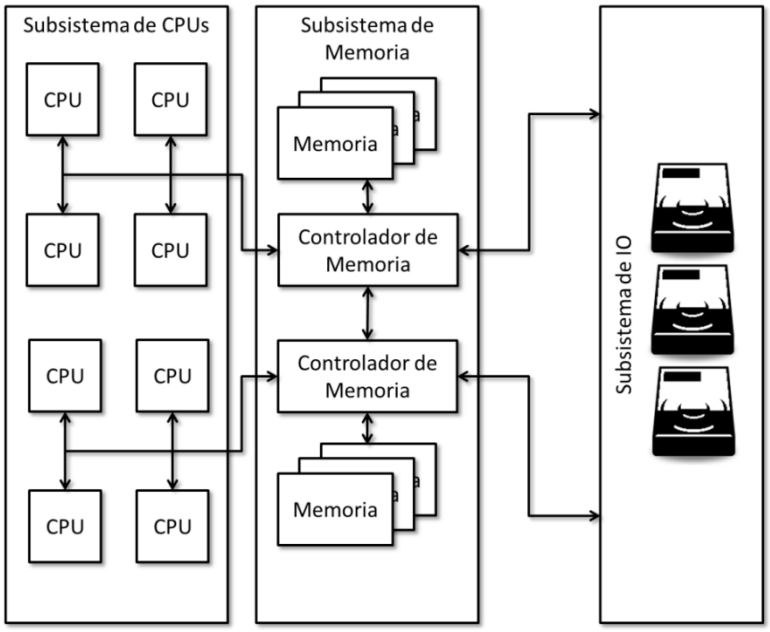

Arquitectura *shared-memory multiprocessor*

En esta arquitectura se presentan problemas con el desempeño ya que los diferentes procesadores tienen un acceso igual a un único bus de memoria que es compartido. También es difícil fraccionar el acceso al bus sin perder performance.

Un ejemplo de arquitectura en el subsistema de procesadores que es más eficiente es Non-Uniform Memory Access (NUMA). Los beneficios principales de NUMA es la escalabilidad, que tiene límites definidos cuando se utilizan arquitecturas como Symmetric Multiprocessing (SMP). Con SMP, todo el acceso a la memoria se logra a partir del mismo bus compartido de memoria. Esto trabaja bien para un número relativamente bajo de procesadores, pero los problemas aparecen cuando se tienen muchos procesadores compitiendo por el acceso al subsistema de memoria con el mismo bus. La tendencia en el hardware es tener más de un solo bus, cada uno sirviendo a un grupo pequeño de procesadores.

NUMA limita el número de procesadores en uno o más buses. Cada grupo de procesadores tiene acceso a su propia memoria en el subsistema y posiblemente sus propios canales de comunicación hacia el subsistema de IO. Sin embargo, cada CPU es capaz de tener acceso a la memoria asociada con otros grupos de una forma coherente.

Cada grupo es llamado nodo NUMA, y los nodos están ligados entre ellos con interconexiones de alta velocidad. El número de procesadores dentro de un nodo NUMA dependerá del proveedor de dicho hardware. Es más rápido el acceso a la memoria local que el acceso a la memoria asociada a otros nodos NUMA. Y esta es la razón del nombre que posee la arquitectura. La ilustración debajo muestra 2 nodos NUMA con 4 procesadores cada uno y su propio canal de acceso al subsistema de IO.




Configuración de 2 nodos NUMA con su propio canal de IO

El subsistema de procesadores es parte importante en el procesamiento que se requiere para la ejecución de solicitudes de usuarios al RDBMS, y el procesamiento es principalmente utilizado para la generación de los planes de resolución para esas solicitudes, así como para la ejecución de los planes en sí.

Los RDBMS implementan un mecanismo denominado paralelismo, que no es más que la repartición del trabajo de ejecutar un plan entre otros CPUs con menor carga de trabajo.

El subsistema de CPUs es usado por el RDBMS de forma interna para llevar a cabo las siguientes operaciones en otros subsistemas:

1. Solicitudes de lectura y escritura en el subsistema de IO

- A mayor cantidad de lecturas o escrituras se requerirá de un mayor tiempo de procesamiento en los CPUs

2. Solicitudes de posicionamiento y lectura de información en el subsistema de memoria

   - A mayor cantidad de operaciones de posicionamiento solicitadas habrá una demanda proporcional de procesamiento en los CPUs. Más aún en subsistemas de memoria con una capacidad reducida de almacenamiento y con esperas frecuentes por la asignación de espacios libres

3. Mantenimiento de la escucha de solicitudes de usuario y para el envío de resultados a través del subsistema de red

   - A mayor cantidad de datos recibidos en el subsistema de red mayor uso del subsistema de CPUs y de memoria

   - A mayor cantidad de datos puestos en las colas de envío de resultados al cliente mayor consumo del subsistema de CPUs y de memoria

Cualquier lentitud, ya sea por diseño o por utilización, de alguno de los subsistemas da como resultado un uso de mayor tiempo de procesamiento, por lo tanto el alto consumo de tiempo de procesamiento es una consecuencia palpable de las lentitudes en alguno de los 3 subsistemas restantes.

### Subsistema de IO

Los manejadores de base de datos relaciones requieren del almacenamiento de los datos que estos manejan en medios que permitan la durabilidad. El concepto de durabilidad se explora más a detalle en el anexo "IV. Concurrencia y niveles de aislamiento" donde se abordan las propiedades ACID de una transacción.

Los RDBMS son capaces de determinar si las transacciones de usuario requieren de la escritura de datos y si estas operaciones requieren posicionarse en el subsistema de Entradas/Salidas, también conocido como Input/Output o simplemente IO por su definición en inglés. Si el caso es este entonces todas las modificaciones y escrituras que las transacciones requieran serán enviadas al subsistema de IO para que estas sean plasmadas ahí y así hacerlas durables[27]. Por otro lado, si las transacciones requieren de la lectura de datos es probable que el RDBMS haga una solicitud de lectura al subsistema de IO para que esa información sea recuperada para satisfacción de la transacción. En la realidad de los RDBMS la solicitud de la lectura al subsistema de IO es un proceso más inteligente. En este anexo se habla del subsistema de memoria y se explica más a detalle este manejo.

[27] Date, C. J. (2004). *An introduction to database systems (Eight Edition)*. United States of America: Adisson-Wesley. pp. 448

Los manejadores ponen a disposición de los administradores de bases de datos y desarrolladores estructuras de datos que permiten la organización de la información almacenada en la base de datos de acuerdo a las necesidades de los negocios y a los requerimientos de desempeño. Para grandes cantidades de datos resulta más útil mantener las estructuras de datos materializadas en el subsistema de IO a fin de evitar el tiempo y el espacio requerido en memoria para la materialización temporal cada vez que esta sea requerida.

El subsistema de IO permite que los RDBMS almacenen los datos de una base de datos en forma de archivos, y dentro de estos se almacenará el diccionario de datos y los objetos tales como tablas e índices en forma de estructuras de datos que permiten el manejo de grandes volúmenes de información, aunque no necesariamente esta organización permitirá un mejor desempeño.

Es entonces por la durabilidad transaccional, por la materialización de estructuras de datos y por el almacenamiento de las bases de datos y su contenido en sí que el subsistema de IO tiene un papel principal en el desempeño. Un subsistema de IO bien configurado, utilizado y administrado ayudará a obtener un mejor desempeño.

Cabe mencionar que este subsistema es uno de los que por mucho tiempo se ha considerado como el más lento de los 4 que se describen en este anexo[28], debido en gran parte a que su mecanismo de operación es mecánico y a las implementaciones disponibles en el mercado para la protección de los datos ya almacenados.

En los últimos años han surgido nuevas tecnologías de almacenamiento de información que ya no utilizan sistemas mecánicos, en su lugar estas tecnologías utilizan sistemas de circuitos integrados o desplazamiento de información en medios como redes de fibra, logrando mejores tiempos de respuesta en la escritura y recuperación de información.

Aun así, es importante indicar que tanto la codificación de la sentencia SQL como el plan con el que el manejador plantea recuperar o escribir la información influyen también en los tiempos de respuesta de este subsistema. Una sentencia que recupera todos los registros de una tabla, aun cuando en las aplicaciones se filtre la mayor parte de ellos, generará más lecturas en el sistema de IO que las que realmente se requieren.

Para una mayor profundización en las tecnologías actuales de almacenamiento y de las nuevas tecnologías, así como su impacto en el diseño de los RDBMS refiérase al anexo "VIII. Niveles de RAID" y al anexo "X. Nuevas tecnologías de almacenamiento"

[28] Harrison, Guy. *Best Practices for Optimizing Oracle RDBMS with Solid State Disk*. Quest Software Inc. http://questsoftware.com.mx/Quest_Site_Assets/WhitePapers/Best_Practices_for_Optimizing_Oracle_RDBMS_with_Solid_State_Disk-final.pdf. [consulta: Noviembre de 2011]

## Subsistema de memoria

El subsistema de memoria es uno de los más importantes para ayudar a reducir los accesos al subsistema de IO y por consecuencia reducir los tiempos de respuesta del RDBMS para la atención de solicitudes de usuario, esto es posible gracias a que es en el subsistema de memoria que los manejadores crean y organizan áreas destinadas para servir de buffer o caché.

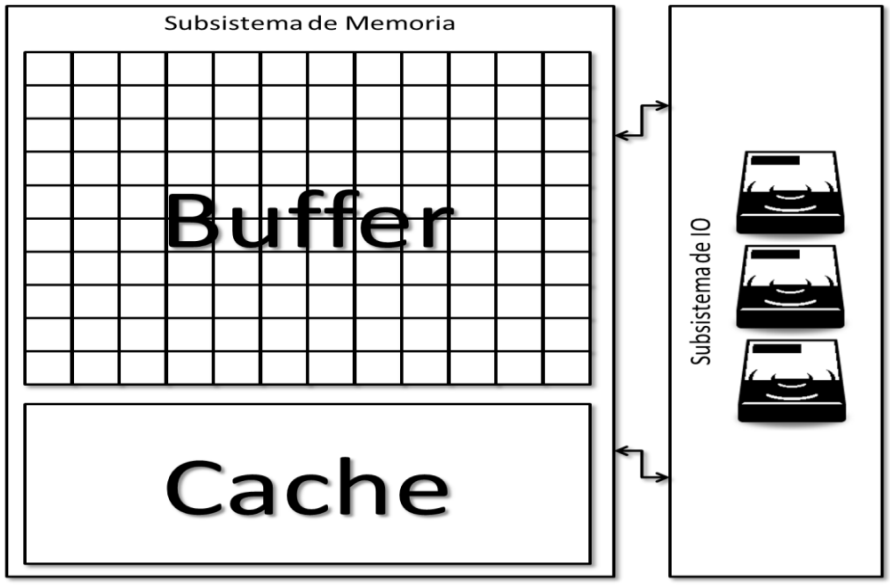
Estos buffers son utilizados por el RDBMS para almacenar diferente información de las estructuras mismas de las bases de datos, tales como el diccionario de datos, o estructuras de datos que contienen información de usuario (heaps, índices, conjuntos de datos para ordenamiento o agregación, etc.).

La forma en la que el RDBMS hace interactuar los subsistemas de IO y de memoria para la resolución de una solicitud de lectura de información se muestra en el siguiente ejemplo:

1. El usuario hace una solicitud de lectura de datos ya escritos en el subsistema de IO
2. El RDBMS toma la solicitud y busca los datos solicitados en los buffers del subsistema de memoria
3. Si la información no está en los buffers entonces el RDBMS hace una solicitud al subsistema de IO para que se recuperen los datos
4. El subsistema de IO resuelve la solicitud con una serie de operaciones de lectura y devuelve esa información al RDBMS
5. El RDBMS toma los datos devueltos por el subsistema de IO y los coloca en los buffers disponibles, así estos datos en los buffers son reutilizables para otras solicitudes de lecturas de la misma información sin necesidad de recurrir nuevamente al subsistema de IO
    a. Estos datos no permanecen en los buffer por siempre, sino que los RDBMS implementan mecanismos que determinan cuando esos datos no son útiles y los deshecha para hacer un mejor uso del espacio limitado en los buffers
6. El RDBMS procesa los datos y los envía al usuario, concluyendo el flujo

El intercambio de datos entre el subsistema de Input/Output y el subsistema de memoria ayuda a mejorar los tiempos de respuesta de los RDBMS ya que el subsistema de memoria no utiliza medios físicos para la lectura o escritura de información sino que todo dentro de ellos es circuito integrado.

Un servidor con gran capacidad en su subsistema de memoria permite al RDBMS alojar más espacio para sus buffers y para el caché, lo que reduce el acceso a disco, la ilustración siguiente muestra la organización del subsistema de memoria en buffers.




El subsistema de memoria estructurado en buffers y espacio para caché

El subsistema de memoria es de almacenamiento volátil y es vulnerable a corrupciones en bloques de memoria debido a fallas de energía o violaciones de los espacios asignados a otras aplicaciones, por lo que se requiere que para el control transaccional exista una bitácora de control del progreso de cada una de las transacciones y que dicha bitácora sea escrita en el subsistema de IO.

Una de las desventajas en el subsistema de memoria es que para ciertas arquitecturas de hardware o de procesamiento existen límites máximos de almacenamiento. Otra desventaja del subsistema de memoria es que bajo ciertas circunstancias requiere del propio subsistema de IO para emular mayor capacidad de almacenamiento, alojando datos de forma parcial en ambos subsistemas, lo que genera un menor desempeño.

La limitación de almacenamiento en el subsistema de memoria también varía entre los sistemas operativos capaces de hospedar a estos manejadores, sin embargo la utilización de este subsistema sigue siendo más eficiente que los accesos al subsistema de IO basado en movimientos mecánicos.

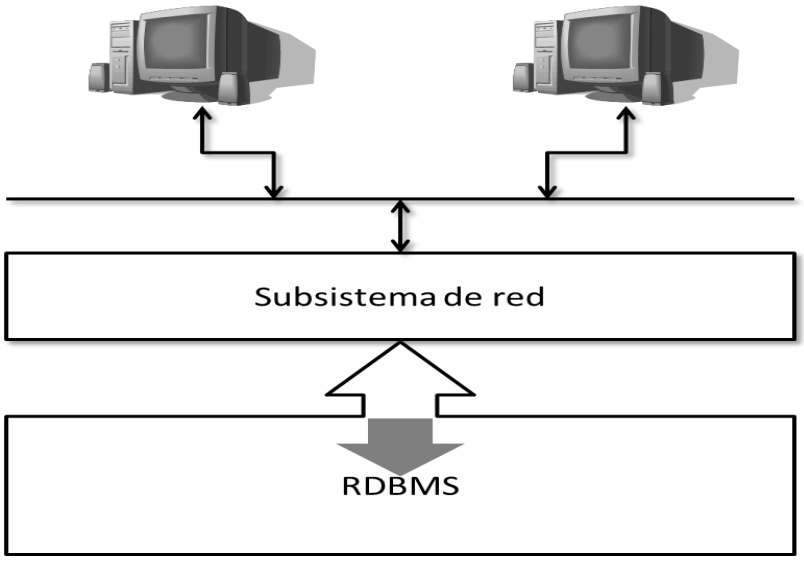
Sólo se obtendrá un mejor desempeño en el subsistema de memoria si se procura mantener la información mínima necesaria y si en las lecturas se procura sólo recuperar la información necesaria con el mínimo de operaciones.

## Subsistema de red

Los manejadores de base de datos relacionales y los clientes de dichos manejadores requieren comunicarse para intercambiar solicitudes e información. El subsistema de red es parte integral en la arquitectura de los RDBMS que atienden sistemas cliente-servidor.

Los manejadores de base de datos relacionales utilizan los mecanismos que el sistema operativo ofrece para hacer uso del subsistema de red y sobre él establecer comunicación con las aplicaciones cliente.

El subsistema de red recibe una solicitud del usuario y el RDBMS la traduce a una forma que su motor entiende y opera, también toma los resultados finales de cualquier solicitud y los traduce en formas que las aplicaciones cliente entienden. La ilustración debajo describe de mejor manera como el RDBMS utiliza el subsistema de red.




Utilización del subsistema de red en una arquitectura cliente-servidor

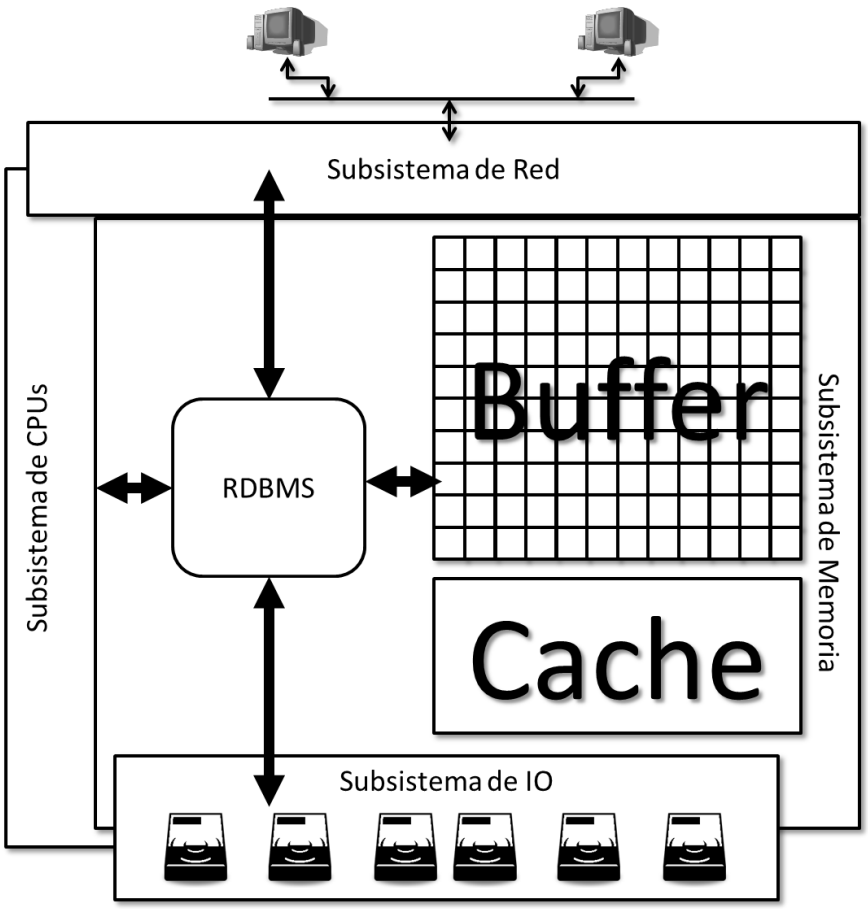
Todos los RDBMS hacen uso del protocolo TCP/IP en el subsistema de red, ya que es un protocolo ampliamente usado en internet. TCP/IP permite comunicaciones a través de redes interconectadas de computadoras con arquitecturas de hardware y sistemas operativos muy diversos. TCP/IP incluye estándares de ruteo de tráfico en la red y ofrece características de seguridad avanzada.

Por este subsistema pasa gran cantidad de datos desde o hacia el cliente, por lo que es importante aprender a transmitir o recibir sólo la información necesaria para llevar a cabo una transacción o para presentar información al usuario.

## Interacción entre los subsistemas y el motor de los RDBMS

Los subsistemas descritos anteriormente guardan una estrecha relación e interacción en la dinámica que los manejadores de base de datos requieren.

La ilustración debajo muestra la relación que existe entre los diferentes subsistemas. Todos los manejadores de base de datos mantienen la misma dinámica con los 4 subsistemas.




Interacción entre los subsistemas utilizados por los RDBMS

Para entender mejor la interacción entre los subsistemas y el manejador se mostrará un flujo simplificado que sigue la transacción de usuario dentro de los subsistemas hasta satisfacerse:

1. El usuario mediante su aplicación cliente envía una solicitud al manejador de base de datos a través de la red y esta es primero recibida por el módulo que se encarga de las comunicaciones con el subsistema de red
2. El manejador utiliza los buffers en el subsistema de memoria para determinar si es posible satisfacer la solicitud con la información en dichos buffers, es posible que utilice algún caché para poner información que ayude al manejador a atender esta u otras solicitudes
3. Si no se encuentra la información necesaria para satisfacer la solicitud en los buffers del subsistema de memoria entonces el manejador de base de datos hace una petición al subsistema de Input/Output para la lectura o escritura de información necesaria para la atención de la solicitud
4. El subsistema de Input/Output recupera la información solicitada por el manejador de base de datos
5. El manejador coloca la información retornada por el subsistema de Input/Output en los buffers del subsistema de memoria

6. El manejador de base de datos, ya con la información necesaria, satisface la solicitud del usuario. Si la solicitud requiere de escribir en el subsistema de IO lo hará a fin de completar la solicitud
7. Una vez que dentro del manejador de base de datos la solicitud ha terminado se envía la información requerida de regreso al cliente a través del subsistema de red, completándose el flujo

Todos los subsistemas y los componentes del RDBMS requiere del subsistema de CPUs para operar, sin embargo, no todas las sentencias entrantes en el manejador de base de datos requieren de acceso al subsistema de IO.

## II. Características y arquitectura general de los RDBMS

El modelo relacional de datos y la teoría relacional prescriben la definición correcta de bases de datos y como los datos serán accedidos y manipulados. Un RDBMS es el encargado del manejo de la base de datos, del procesamiento de las validaciones de la integridad que el modelo define, del manejo transaccional y del control de la concurrencia. También el RDBMS es el encargado de traducir las sentencias SQL a operaciones del álgebra y del cálculo relacional.

Todos los RDBMS implementan una serie de módulos que tienen similitud entre ellos y que permiten mantener las operaciones descritas anteriormente. De estos módulos los más determinantes en el desempeño son los siguientes:

1. Módulo de métodos de acceso

   El módulo de métodos de acceso, o access methods en inglés, contiene los algoritmos de escritura, lectura e iteración de conjuntos de datos. Es el responsable también de mantener el control de las estructuras de datos

2. Módulo de manejo del buffer

   El módulo de manejo del buffer, o buffer manager en inglés, mantiene los algoritmos de posicionamiento y recuperación de datos así como los algoritmos para administrar la reutilización del espacio en los buffers. Es también el encargado de almacenar y servir espacios de memoria para operaciones dentro de otros módulos

3. Módulo de manejo de candados

   El módulo de manejo de candados, o lock manager en inglés, contiene los algoritmos de imposición de candados. Los algoritmos establecen el tipo de candados que se impondrán sobre los datos utilizados dentro de las transacciones. Este módulo es el encargado del control de la concurrencia

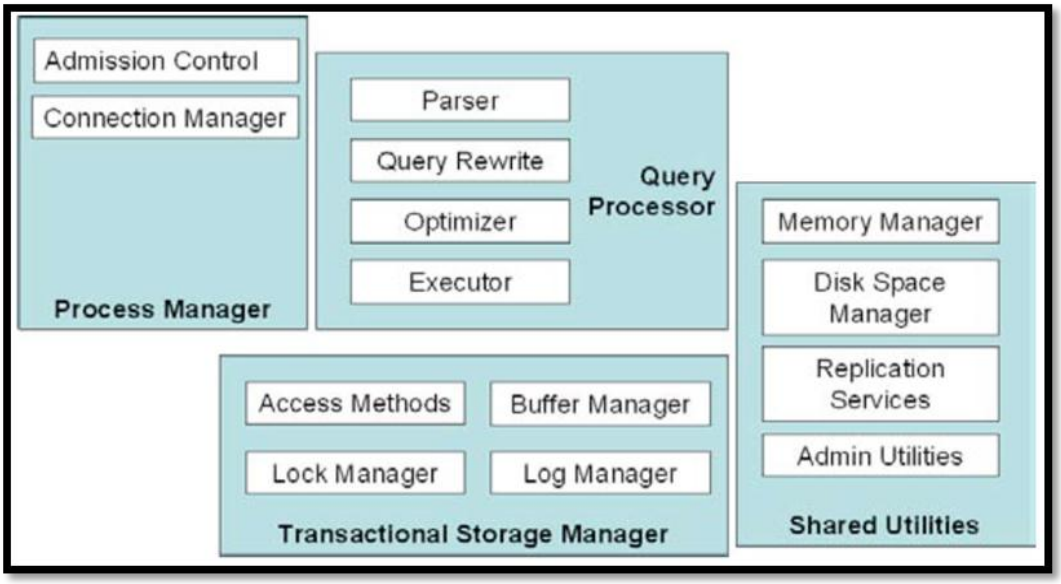
4. Módulo de administración de la bitácora de transacciones

El módulo de manejo de la bitácora de transacciones, o transaction log manager en inglés, es el encargado de mantener la propiedad de atomicidad de las transacciones mediante la escritura de los avances de cada transacción en la bitácora

5. Módulo de optimización de sentencias

   Este módulo, también conocido como optimizer en inglés, es el encargado de la generación de un plan óptimo para ejecutar las sentencias que serán procesadas con álgebra y cálculo relacional. Su función es encontrar un plan de ejecución cuyo costo de ejecución en términos de tiempo sea bajo

El diagrama general de los componentes de un RDBMS que ayudan a mantener las operaciones descritas se muestra en la siguiente figura[29].




**Componentes de un DBMS**

## III. Ejemplo de implementación de un B+-tree en SQL Server

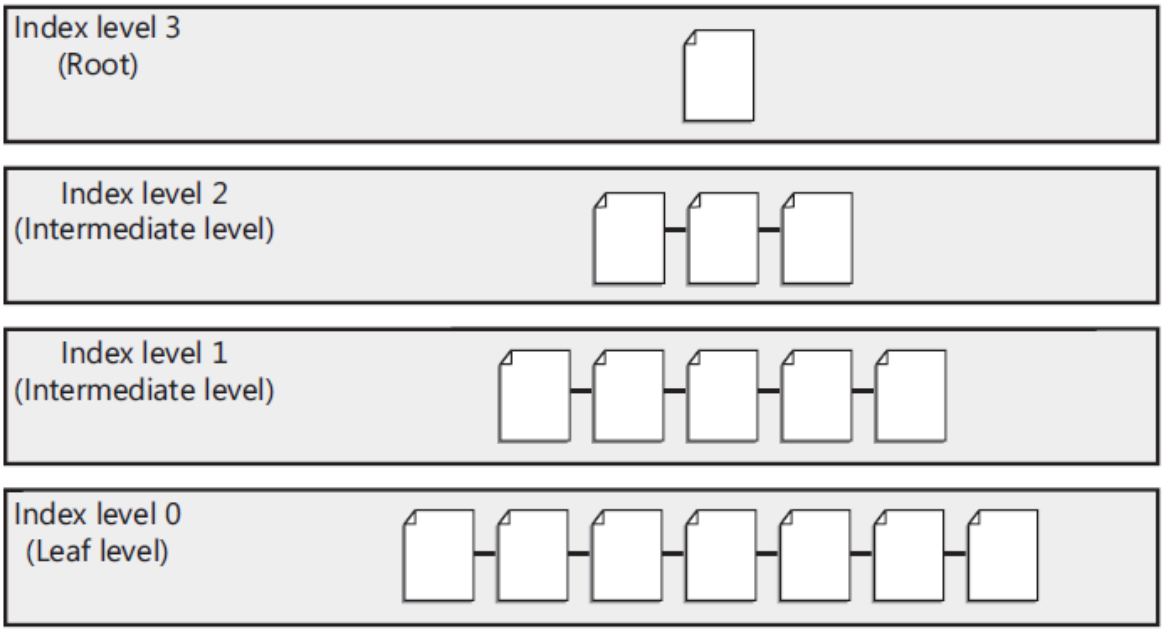
Para ejemplificar exactamente porque un B+-tree es una muy buena estructura de datos para los RDBMS se explicará el uso que hace SQL Server de la estructura para la implementación de índices. Se usarán 2 ejemplos de la implementación de índices con SQL Server ya que la bibliografía del

[29] Hellerstein, Joseph M., Stonebraker, M., (2005). *Readings in Database Systems (4th Edition)*. United States of America: MIT Press. pp. 44

producto[30] explica de forma suficientemente clara los detalles de la estructura aunque todos los manejadores utilizan los B+-trees para mantener el ordenamiento en los datos.

El nombre B+-tree, o "árbol+-B", se adoptó por el "balanceo" que mantiene entre sus niveles. SQL Server adoptó un tipo especial de árbol conocido como B+tree el cual no mantiene un balanceo estricto en toda forma todo el tiempo. A diferencia del resto de las estructuras de árbol, un B+-tree siempre se encuentra invertido, con su raíz definida por una única página en lo más alto y sus niveles de hojas están en la parte más inferior.

La figura debajo muestra la estructura típica de un B+tree utilizado en SQL Server en su forma de índice.




**B+-Tree utilizado para implementar un índice en SQL Server**

Lo interesante de los B+-Trees en SQL Server es cómo se construyen y lo que está dentro de cada nivel. Estructuralmente, los índices presentan ligeros cambios en el espacio que utiliza basado en si se utilizan o no varios CPUs para crearlos o reconstruirlos, pero en su mayor parte, el tamaño y el ancho del árbol se basan en la definición del índice y el número y tamaño de las filas de la tabla. Para demostrar los ejemplos se darán términos generales y definiciones previas. En primer lugar, los índices tienen dos componentes principales: un nivel de hoja y uno o más niveles intermedios. Es interesante comprender y debatir sobre los niveles intermedios, pero en pocas palabras, se utilizan para la navegación, en su mayoría para desplazarse hasta el nivel de hoja. Sin embargo, el

[30] Delaney, Kalen (2009). *Microsoft SQL Server 2008 Internals*. United States of America: Microsoft Press. pp. 300

primer nivel intermedio también se utiliza en el análisis de la fragmentación y para la lectura anticipada durante las exploraciones de un rango amplio del índice.

Para comprender estas estructuras, se definirá el nivel de hoja de los índices usados por SQL Server en términos genéricos, lo que significa que estos son conceptos básicos que se aplican a los índices de tipo clustered[31] y non clustered[32]. El nivel hoja de un índice contiene algo, lo que contienen varía dependiendo si la tabla está implementada como un heap o si ya cuenta con un índice clustered.

Los niveles intermedios entre el nodo raíz y los nodos hojas existen para ayudar al motor de SQL a navegar a una fila en el nivel de hojas, pero la arquitectura es bastante sencilla. Cada nivel intermedio almacena algo para cada página del nivel inferior, y los niveles son agregados de abajo hacia arriba hasta acumulase en un nodo raíz que consta de una única página.

Cada nivel superior intermedio en el índice, los más lejanos al nivel hoja, contiene menor cantidad de páginas y es mucho más pequeño que el nivel inferior debido a que cada registro en las páginas de un nivel contiene sólo la llave mínima de ordenamiento que está contenida en las páginas en el nivel inferior inmediato, además de un apuntador a esa página.

Aunque suena como que esto da lugar a un montón de niveles, es decir, un árbol muy alto, la limitación del tamaño de la clave, que tiene un máximo de 900 bytes o 16 columnas – lo que ocurra primero, ayuda a SQL Server a mantener árboles de índices relativamente pequeños.

Para el primer ejemplo se mostrarán los cálculos para determinar el espacio requerido por una tabla donde el nivel de hojas del índice contendrá un millón de “filas”. Se encierra la palabra filas entre comillas porque estas no son necesariamente las filas de datos sino que son simplemente filas en el nivel de hoja de cualquier índice. Sin embargo, en este ejemplo, se hablará de un índice abstracto haciendo hincapié sólo en los pormenores de los niveles intermedios, así como el cómo están estructurados dentro de los confines de una página de SQL Server, donde cada página es de 8 KB. En este ejemplo, las filas de nuestro nivel de hoja son de 4,000 bytes, lo que significa que sólo se almacenarán dos filas por página. Para una tabla con un millón de filas, resultaría un índice que en el nivel de hojas tendría 500,000 páginas. Relativamente hablando, esta es una estructura de fila bastante amplia; Sin embargo, no se está perdiendo gran cantidad de espacio en la página. Si la página de nivel hoja tuviera dos filas de 3,000 bytes entonces sólo cabrán dos filas por página, y habrá 2,000 bytes de espacio desperdiciado. Esto es un ejemplo de la fragmentación interna.

[31] Los índices clustered en SQL Server son B+ trees que ordenan físicamente los registros de una tabla que tuvo forma de heap. Sólo se permite hacer un índice clustered por tabla.

[32] Los índices non clustered en SQL Server se materializan como un B+tree que no organiza los registros de una tabla de forma física, sino que mantienen un orden lógico de los mismos. Se permite más de un índice non clustered

Ahora, hablar simplemente de “filas” y no específicamente las filas de datos en el nivel de hoja de un índice es útil para describir lo que almacenan los índices clustered y non clustered. El nivel hoja de un índice clustered almacena las filas de datos, es decir, filas de registro con todas sus columnas. En un índice non clustered, las filas de nivel hoja serán filas de índice, en cuya definición se permite incluir algunos campos pertenecientes al registro completo, una funcionalidad que fue añadida en SQL Server 2005, y se utiliza para agregar columnas que no son parte de la clave que define al índice pero que se almacenan justo en el nivel de hojas. Cuando se utiliza la cláusula INCLUDE en la generación del índice las páginas de nivel de hoja contendrán filas más amplias que 900 bytes o 16 columnas como máximo.

Una vez más, aunque esto actualmente no suena interesante, este diseño es beneficioso en los mecanismos de lectura de información. En este ejemplo, el nivel de hojas de este índice sería de 4 GB de espacio, es decir 500,000 páginas de 8 KB, en el momento en que se crea. Esta estructura, dependiendo de su definición, será mayor y en algún momento estará muy fragmentada si se añaden un montón de nuevos datos. Sin embargo, y nuevamente dependiendo de su definición, hay formas de controlar cómo ocurre la fragmentación en este índice cuando los datos son volátiles.

En este ejemplo, el nivel de hojas del índice es grande debido a la longitud de la “fila”. Con una longitud máxima de 900 bytes para la llave en un índice sólo se almacenarán ocho filas por página en los nivel intermedios anteriores al nivel hoja (8,096 bytes por página / 900 bytes por fila). Sin embargo, mediante este máximo, el árbol resultante sería relativamente pequeño y como resultado sólo habrá ocho niveles, como se muestra aquí. De hecho, una escalabilidad mejorada es la principal razón para que el límite de una clave de índice sea de 900 bytes o 16 columnas — lo que suceda primero:

- Página del nodo raíz (nivel 7) = 2 filas = 1 página (8 filas por página)
- Nivel intermedio (nivel 6) = 16 filas = 2 páginas (8 filas por página)
- Nivel intermedio (nivel 5) = 123 filas = 16 páginas (8 filas por página)
- Nivel intermedio (nivel 4) = 977 filas = 123 páginas (8 filas por página)
- Nivel intermedio (nivel 3) = 7,813 filas = 977 páginas (8 filas por página)
- Nivel intermedio (nivel 2) = 62,500 filas = 7,813 páginas (8 filas por página)
- Nivel intermedio (nivel 1) = 500,000 filas = 62,500 páginas (8 filas por página)
- Nivel hoja (nivel 0) = 1,000,000 filas = 500,000 páginas (2 filas por página)

Un índice con un tamaño de clave menor escalará aún más rápido.

En un segundo ejemplo suponga la existencia de las mismas páginas de nivel de hoja como se mostró anteriormente, con un millón de filas y donde caben 2 filas por página, pero con una clave de índice menor y por lo tanto un menor tamaño de fila en los niveles intermedios, incluyendo algún espacio para sobrecarga de sólo 20 bytes. Con esta definición cabrán 404 registros por página de nivel intermedio. El espacio requerido es el siguiente:

- Página del nodo raíz (nivel 3) = 4 filas = 1 página (404 filas por página)
- Nivel intermedio (nivel 2) = 1,238 filas = 4 páginas (404 filas por página)
- Nivel intermedio (nivel 1) = 500,000 filas = 1,238 páginas (404 filas por página)
- Nivel hoja (nivel 0) = 1,000,000 filas = 500,000 páginas (2 filas por página)

En este segundo ejemplo, el índice inicial consta de tan sólo cuatro niveles, y en ellos hay una capacidad de alojamiento de 130,878,528 filas adicionales antes de que se requiera otro nivel. El cálculo es sencillo ya que el máximo número posible de filas es 404 * 404 * 404 * 2 o 131,878,528 filas menos el millón de filas que ya existen. Concretando, la página raíz permite actualmente 404 entradas para apuntar a páginas en el nivel inferior. Sin embargo, sólo se están almacenando 4 apuntadores, además los niveles intermedios tienen una ocupación de página del 100 por ciento. Esto es sólo un máximo teórico sin contemplar otros factores como la fragmentación.

Un árbol de cuatro niveles en SQL Server sería útil para buscar más de 131 millones de filas en una tabla, tomando en cuenta que el tamaño de clave de índice es de 20 bytes. Esto significa que para una búsqueda en este índice, que utiliza un B+tree para desplazarse a la fila correspondiente, se requiere sólo cuatro operaciones en el subsistema de IO. Y debido a que los árboles están equilibrados, encontrar cualquier registro requiere la misma cantidad de lecturas y recursos.

## IV. Concurrencia y niveles de aislamiento

La concurrencia es definida como la habilidad de múltiples sentencias de usuario para acceder o modificar, y por lo tanto compartir, datos al mismo tiempo. Entre más grande sea el número de procesos concurrentes que estén activos sin interferir con otros se concluirá que el RDBMS es más concurrente.

La concurrencia se ve disminuida cuando un proceso que está cambiando datos inhibe a otros procesos que leen datos o cuando un proceso que está leyendo datos no permite que otros procesos cambien datos. En esta investigación los términos leer o acceder se usarán indistintamente para describir el impacto de usar sentencias SELECT sobre datos. La concurrencia también se ve afectada cuando múltiples procesos intentan cambiar los mismos datos simultáneamente y no todos ellos se completan sin sacrificar la consistencia de los datos. Los términos modificar, cambiar o escribir se usarán indistintamente en esta investigación para describir el impacto de usar sentencias INSERT, UPDATE o DELETE sobre datos.

El mecanismo con el que cuentan los RDBMS para el control de la concurrencia es la imposición de candados para asegurar los diferentes objetos participantes o sus componentes más granulares. Con la imposición de estos candados se asegura que durante la ejecución concurrente de transacciones de usuario no se permite la modificación de objetos que ya están asegurados por algún candado impuesto desde otra transacción.

En general, los sistemas manejadores de base de datos toman 2 aproximaciones para manejar los accesos concurrentes: optimista y pesimista

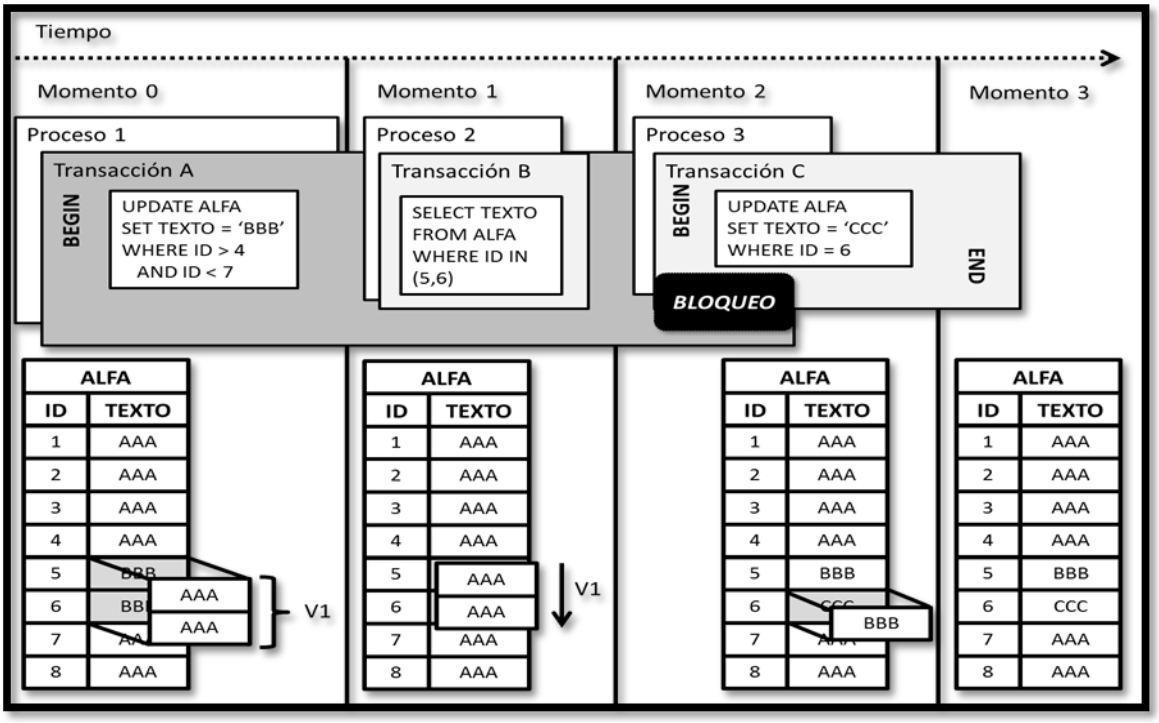
En cualquier modelo de concurrencia, un conflicto ocurre si dos procesos tratan de modificar los mismos datos al mismo tiempo. La diferencia entre los modelos optimista y pesimista radica en el hecho de evitar conflictos antes de que ocurran y en los mecanismos de tratamiento una vez que ocurren.

### La concurrencia pesimista

La concurrencia pesimista asume que en un sistema ocurren suficientes operaciones de modificación de datos que cualquier operación de lectura probablemente será afectada por una modificación de los datos realizada por otro usuario. En otras palabras, el sistema se comporta de forma pesimista y supone que un conflicto se producirá. La concurrencia pesimista evita los conflictos mediante la imposición de candados en los datos que se están leyendo, por lo que ningún otro proceso modificará los mismos datos. También se imponen bloqueos en los datos que se modifican, por lo que ningún otro proceso accederá a esos datos, ya sea para leer o modificar. En otras palabras, los lectores bloquean las acciones de los escritores y los escritores bloquean las acciones de los lectores en un ambiente de concurrencia pesimista.

### La concurrencia optimista

La concurrencia optimista asume que hay suficientemente pocas operaciones en conflicto de modificación de datos en el sistema que para cualquier transacción es poco probable que una transacción diferente modifique los datos que otra transacción está modificando también. El comportamiento predeterminado de la concurrencia optimista es el uso de múltiples versiones de filas para permitir a los lectores de datos ver el estado de los datos antes de que la modificación tuviera lugar. Las versiones anteriores de filas de datos se guardan, por lo que un proceso de lectura de datos verá los datos tal como eran cuando se inició el proceso de lectura y no será afectado por los cambios realizados a los datos. Un proceso que modifica los datos no es afectado por los procesos de lectura de los datos debido a que el lector tiene acceso a una versión guardada de las filas de datos. En otras palabras, los lectores no bloquean los escritores y los escritores no bloquean los lectores. Los escritores bloquearán a otros escritores. En la lámina debajo se ilustra un ejemplo de control de concurrencia bajo el modelo optimista.




**Manejo de 3 procesos concurrentes en el modelo optimista**

En la lámina se muestran 3 procesos que concurren en una línea de tiempo formada por 4 momentos, 3 procesos y las transacciones abiertas en cada uno. Todas las transacciones efectúan operaciones sobre la tabla llamada ALFA.

En el momento 0 se observa cómo se inicia el proceso 1 y desde este se abre una transacción A que consta de un UPDATE sobre los registros con ID 5 y 6 de la tabla ALFA para cambiar el valor de la columna TEXTO que actualmente tiene el valor 'AAA' por el nuevo valor 'BBB'. Durante ese primer UPDATE se genera la versión V1 de las filas afectadas que estará disponible para otros procesos lectores que lo requieran. La transacción A durará hasta el momento 3.

En el momento 1 se observa que desde un proceso 2 se abre la transacción B, que de hecho se ejecuta de forma concurrente con la transacción A del proceso 1. La transacción B ejecuta una sentencia SELECT que recuperara los mismos registros que la transacción A está modificando. Gracias al modelo de concurrencia optimista, el SELECT recuperará la versión V1 de los registros, lo que no ocurriría bajo el modelo de concurrencia pesimista, donde se hubiera presentado un bloqueo por la espera de que la transacción A termine para permitir al SELECT recuperar los datos en un estado consistente.

En el momento 2 se observa que desde el proceso 3 se abre una transacción C que consta de un UPDATE sobre el registro con ID 6 de la tabla ALFA para cambiar el valor de la columna TEXTO que actualmente tiene el valor 'BBB' por el nuevo valor 'CCC'. En este caso si se presenta un bloqueo con la transacción A, y sólo hasta que la transacción A termina es posible que la transacción C

también continúe y concluya. Para el momento 3 tenemos una base de datos transaccionalmente consistente.

La implementación del modelo de concurrencia optimista es la explotación de una de las múltiples aplicaciones del uso de las bitácoras transaccionales y sus algoritmos de recuperación[33], es por ello que para los RDBMS resulta fácil su implementación.

### Procesamiento de transacciones

No importa qué modelo de concurrencia se esté trabajando, una comprensión de las transacciones es crucial. Una transacción es la unidad básica de trabajo para los RDMBS. Por lo general, se compone de varios comandos SQL de lectura y actualización en base de datos, pero la actualización no se considerará definitiva hasta que se ejecute un comando COMMIT en el caso de una transacción explícita. En esta investigación se abordará el término transacción como una única unidad lógica compuesta de una o más sentencias SQL que leen o modifican datos.

El concepto de una transacción es fundamental para entender el control de concurrencia. Entrar en los detalles sobre los niveles de aislamiento de transacción que los RDBMS manejan y como estos tienen impacto en la concurrencia, así como en los tiempos de espera para la resolución de solicitudes de usuario resulta crucial para el desarrollo de esta investigación.

Una transacción implícita es cualquier sentencia SELECT, INSERT, UPDATE o DELETE individual, aunque las sentencias SELECT no se escriben en las bitácoras de transacciones cuando se procesan.

Una transacción explícita es aquella cuyo inicio está marcado con una instrucción de inicialización (BEGIN / START TRANSACTION) y una de finalización, ya sea por una instrucción de compromiso (COMMIT) o de aborto (ROLLBACK). Es dentro del bloque definido por una transacción que se encontrarán las sentencias que en secuencia definen una operación atómica.

### Propiedades ACID

El procesamiento de transacciones garantiza la consistencia y capacidad de recuperación de bases de datos[34]. El procesamiento transaccional asegura que todas las transacciones se llevan a cabo como una sola unidad de trabajo, incluso cuando ocurra un fallo de hardware o del sistema en general. Todas estas características que muestran las transacciones y otras más están dadas por las propiedades conocidas como propiedades ACID.

He aquí un ejemplo en pseudocódigo de una transacción ACID explícita:

---

[33] Bernstein, Philip A., Hadzilacos, V., Goodman, N., (1987). *Concurrency control and recovery in database systems*. United States of America: Addison-Wesley. pp 143

[34] O'Neil, P., O'Neil, E., (2000). *Database: Principles, Programming, and Performance (Second Edition)*. United States of America: Morgan Kaufmann. pp. 633 - 637

```
START TRANSACTION DEBIT_CREDIT
      Se descuenta en la cuenta de ahorros $ 1000
      Se carga en la cuenta de crédito $ 1000
COMMIT TRANSACTION DEBIT_CREDIT
```

En este anexo se describirán cada una de las propiedades ACID.

### *Atomicity*

La atomicidad, o atomicity en inglés, significa que cada transacción se trata como todo o nada. Si se confirma una transacción, todos sus efectos se mantienen. Si se anula, todos sus efectos se revierten. En el ejemplo anterior de la transacción DEBIT_CREDIT, si el descargo a la cuenta de ahorro se refleja en la base de datos pero el abono a la cuenta de crédito no se hace, esencialmente los fondos desaparecen de la base de datos. Si la cuenta de crédito se abona pero la cuenta de ahorros no tiene el cargo, la cuenta de crédito tendrá un misterioso incremento sin un depósito en efectivo correspondiente al cliente. Debido a la característica de atomicidad de una transacción, tanto el movimiento en la cuenta de ahorros como el movimiento en la cuenta de crédito deben ser completados o de lo contrario ninguno de los 2 debe verse reflejado.

### *Consistency*

La propiedad de consistencia, o consistency en inglés, asegura que una transacción no permitirá que el sistema llegue a un estado lógico incorrecto. Las restricciones y reglas del modelo relacional siempre se respetarán, incluso en el caso de un fallo del sistema. En el ejemplo DEBIT_CREDIT, la regla lógica es que el dinero no se crea o se destruye: un abono será correspondido con un cargo para cada entrada. La consistencia implica en la mayoría de los casos una redundancia a las propiedades de atomicidad, aislamiento y durabilidad.

### *Isolation*

El aislamiento, también conocido como isolation en inglés, separa las transacciones simultáneas de las actualizaciones de otras transacciones en curso. En el ejemplo DEBIT_CREDIT, otra transacción no debería ver el trabajo en curso mientras que la transacción se lleva a cabo. Por ejemplo, si una transacción ajena a DEBIT_CREDIT lee el saldo de la cuenta de ahorros después de que el cargo se produce y a continuación la transacción DEBIT_CREDIT se aborta entonces la transacción ajena está trabajando a partir de un balance de cuentas que nunca existió lógicamente. El aislamiento es controlado por los RDBMS en el módulo de lock management.

Así pues, los RDBMS pondrán candados sobre los datos para evitar que se lea información inconsistente o creará versiones de registros para permitir que múltiples usuarios al mismo tiempo trabajen con datos mientras que se evitan los efectos secundarios que distorsionan los resultados, estos son mecanismos de aislamiento de transacciones. Los RDBMS soportan múltiples niveles de aislamiento donde es posible elegir el equilibrio adecuado entre la cantidad de datos para bloquear con el mecanismo de candados y el nivel de transacciones concurrentes. Los RDBMS determinan de forma inteligente a partir de los niveles de aislamiento durante cuánto tiempo se debe mantener la imposición de los candados, y si se debe permitir a los usuarios el acceso a las

versiones anteriores de filas de datos. La idea es mantener un equilibrio que se conoce como la concurrencia frente a la consistencia.

La forma de permitir mayor concurrencia en el RDBMS y ganar más desempeño es haciendo transacciones cortas que como consecuencia darán candados de menor duración, lo que permitirá a otras transacciones obtener acceso más rápidamente a los datos que requiere para completarse.

### *Durability*

Después de que una transacción se confirma, la propiedad de la durabilidad se asegura de que los efectos de la transacción persistan incluso si un fallo en el sistema ocurre. Si un fallo en el sistema ocurre mientras que una transacción está en curso entonces la transacción se deshace por completo. Por ejemplo, si un corte de energía se produce en medio de una transacción justo antes de que la transacción se confirme entonces toda la transacción se deshará cuando se reinicie el RDBMS. La durabilidad está implementada bajo la codificación del módulo del log manager de los RDBMS.

Si falla la alimentación inmediatamente después de que la transacción se ha confirmado entonces los RDBMS garantizan que la transacción existe en la base de datos. Las bitácoras de las transacciones que se escriben en el subsistema de IO son el mecanismo que usan los RDBMS para asegurar esta durabilidad.

## Dependencias de transacción

La propiedad de aislamiento de transacciones provoca que existan comportamientos específicos de concurrencia y consistencia que reciben el nombre de "problemas de dependencia" o "problemas de coherencia", pero no necesariamente representan problemas sino comportamientos posibles, con excepción del comportamiento de la pérdida de actualizaciones, también conocido lost updates, que nunca se considera conveniente. En el caso de los RDBMS existen mecanismos para establecer cuál de estos comportamientos se desea permitir y cual se desea evitar. Este control de logra mediante la manipulación del nivel de aislamiento y así es posible determinar cuál de estas conductas se permitirá para las transacciones.

### *Lost updates*

La pérdida de actualizaciones, cuyo término en inglés es lost updates, se produce cuando dos procesos leen los mismos datos y ambos efectúan una manipulación de estos. El segundo proceso sobrescribirá la primera actualización completamente. Por ejemplo, suponga que dos empleados de una sala de recepción de un almacén reciben las piezas y agregan los nuevos cargamentos a la base de datos del inventario. El encargado A y el encargado B reciben cada uno un cargamento diferente de reproductores. Ambos verifican el inventario actual y encuentran 25 reproductores actualmente en stock. Uno de los encargados tiene un cargamento con 50 aparatos, por lo que suma 25 a 50 y se actualiza el valor actual a 75. El empleado B recibe un cargamento de 20 reproductores, por lo que añade 25 al valor de 20 que originalmente había y actualiza el valor actual a 45, sobrescribiendo completamente los 50 nuevos reproductores que el encargado A ingresó. La actualización hecha por el empleado A se ha perdido.

La pérdida de actualizaciones es el único de los comportamientos descritos aquí que probablemente se quiere evitar en todos los casos. Sobre todo en aquellos donde se requiere una contabilización numérica precisa, como sería el caso de las aplicaciones que mantienen cuentas bancarias, saldos, inventario de productos o estadísticas.

### *Dirty reads*

Las lecturas sucias, o dirty reads, ocurren cuando un proceso lee datos no confirmados. Si un proceso ha cambiado datos, pero aún no confirma ese cambio, otro proceso leerá los datos en un estado incoherente. Por ejemplo, supongamos que el encargado A ha actualizado el viejo valor del inventario de 25 reproductores a 75, pero antes de comprometer este nuevo valor un agente de ventas observa que actualmente hay 75 reproductores y pacta la venta de 60 de ellos con un cliente y planea entregárselos al siguiente día. Si el encargado A se da cuenta de que los reproductores están defectuosos y los devuelve al fabricante, entonces el agente de ventas habrá hecho una lectura sucia y habrá tomado una decisión basada en información que realmente nunca existió.

Por defecto las lecturas sucias no son permitidas en los RDBMS. Debe tomarse en cuenta que el proceso que hace la actualización de los datos no tiene control sobre el hecho de que otro proceso es capaz de leer sus datos antes que este primer proceso comprometa sus operaciones. Es responsabilidad del proceso de lectura de los datos decidir si quiere leer los datos que aún no se encuentran comprometidos. De forma predeterminada los RDBMS usarán un nivel de aislamiento que permita sólo la lectura de información transaccionalmente consistente.

### *Non repeatable reads*

Una lectura es irrepetible si un proceso tiene diferentes valores en la lectura de los mismos datos en dos partes diferentes de una transacción. Esto ocurre cuando otro proceso cambia los datos después de que ocurre la primera lectura del primer proceso y antes de la ocurrencia de la segunda lectura también en el primer proceso.

Situándonos nuevamente en un almacén, supongamos que un administrador viene a hacer una comprobación rápida del inventario actual.

El administrador se acerca a cada encargado pidiendo el número total de reproductores recibidos hoy y sumando los números en su calculadora. Cuando el administrador termina este ejercicio requiere volver a verificar el resultado, por lo que se remonta a visitar nuevamente al encargado A. Sin embargo, si el encargado A recibe más reproductores entre la primera y la segunda entrevista con el administrador, el total es diferente y las lecturas son no repetibles. Las lecturas no repetibles también se conocen como non repeatable reads o análisis incoherente.

### *Phantom reads*

Las lecturas fantasmas se producen cuando la pertenencia a un conjunto de registros cambia. Esto sólo sucede cuando una consulta con un predicado del tipo conteo_de_reproductores < 10 está

involucrada. Un fantasma se produce cuando dos operaciones SELECT que utilizan el mismo predicado en la misma transacción devuelven un número diferente de filas.

Por ejemplo, digamos que nuestro administrador sigue haciendo controles del inventario en campo. Esta vez, el administrador da un rondín por el almacén y toma nota de que los encargados que tengan menos de 10 reproductores, encontrando que los encargados A y B se encuentran en esta condición. Después de que el administrador completa la lista, él cree conveniente regresar a su posición para informar que se ha alcanzado el mínimo posible de reproductores en el almacén para los encargados A y B. Sin embargo, si durante su primer recorrido el encargado C no se encontraba en las instalaciones y también contaba con menos de 10 reproductores este empleado no está en la lista del administrador a pesar de que cumple con los criterios establecidos en el predicado. Este encargado faltante es considerado como un fantasma. En un segundo rondín el administrador tomará nuevamente la lista y encontrará que 3 encargados tienen menos de 10 reproductores en su control y entonces habrá dado la primera vez una información incompleta.

## V. Métodos de acceso comunes en los RDBMS

Para esta anexo se hizo un estudio de los métodos de acceso comunes a los RDBMS DB2[35], Oracle[36], Microsoft SQL Server[37] y MySQL[38]. De este estudio resultaron los siguientes métodos de acceso, así como un nombre más genérico y la referencia al nombre particular de la implementación en cada uno de los RDBMS estudiados.

### Lecturas en escaneo completo de heaps

El método de acceso de lecturas en escaneo completo de heaps hace la lectura completa de los registros de datos de todas las páginas que componen una tabla organizada en un heap. Existen varias condiciones por las que el optimizer decidiría utilizar este método para resolver una sentencia.

Cuando el optimizer atiende una sentencia que solicita algún registro de una tabla formada como un heap y que no cuenta con ningún índice entonces tiene como única alternativa posible el escaneo de la tabla completa para satisfacer la sentencia.

En un ejemplo simple se tiene la siguiente sentencia de usuario sobre una tabla con 10 millones de registros y que se encuentra organizada en un heap. La columna IdCliente tiene una restricción de unicidad que asegura que 2 registros no tendrán el mismo valor en dicha columna.

---

[35] Mullins, Craig S. (2004). *DB2 Developer's Guide (Fourth ed.)*. United States of America: Sams Publishing. pp. 726 - 763

[36] Oracle. *The Query Optimizer*. Oracle Database Performance Tuning Guide 11g Release 1 (11.1). http://docs.oracle.com/cd/B28359_01/server.111/b28274/optimops.htm#i82107. [consulta: Diciembre 2011]

[37] Ben-Gan, Itzik (2009). *Inside Microsoft SQL Server 2008: T-SQL Querying*. United States of America: Microsoft Press. pp. 197 - 244

[38] Pachev, Sasha (2007). *Understanding MySQL Internals*. United States of America: O'Reilly Media, Inc. pp. 178 - 190

```
SELECT IdCliente
FROM Cliente
WHERE IdCliente = 5
```

Para esta sentencia el plan de ejecución utilizaría el acceso de lecturas en escaneo completo de la tabla ordenada como un heap. Lo que parecería un esfuerzo enorme para recuperar un sólo registro. Este método de acceso recibe nombres diferentes en cada RDBMS, como lo muestra la siguiente lista:

- **MySQL.** ALL record Access, system record access
- **Oracle.** Tablescan
- **SQL Server.** Table scan
- **DB2.** Table Space scan

Este operador enviará el conjunto de datos solicitado, aún si se trata de un conjunto vacío, al siguiente operador en el plan de ejecución.

### Lecturas en escaneo completo del nivel de hojas en un B+-Tree

El método de acceso de lecturas en escaneo completo del nivel de hojas de un B+-tree hace la lectura completa de los registros de datos o de los registros de índice que se almacenan en las páginas que componen el nivel hoja del B+-tree.

Esto aplica si el nivel de hojas del índice contiene todas las columnas solicitadas por la consulta, ya sea que el nivel de hojas contenga el registro completo o bien que las hojas almacenen la referencia de los registros completos de datos.

Existen varias condiciones por las que el optimizer decidiría utilizar este método para resolver una sentencia, una de ellas es la falta de un mejor índice para satisfacer la sentencia, aunque existen otros factores.

En una tabla que cuenta con la organización física de un índice con la implementación de un B+-tree es posible que se haga una consulta de datos cuyo criterio de búsqueda contemple una columna que no forma parte de la definición de la llave del índice. En una búsqueda de este tipo, y aun habiendo un índice, el optimizer lo considerará ineficiente para esta sentencia, entonces el optimizer decidirá que la única forma de recuperar los registros coincidentes es usar este método sobre el nivel de hojas del índice que da orden físico a la tabla.

En un ejemplo simple supongamos la siguiente sentencia de usuario sobre una tabla que se encuentra organizada físicamente con un índice, donde la columna IdCliente es la llave primaria que da orden a la tabla, pero la búsqueda de registros típicamente no se hace por la columna IdCliente, sino por la columna RFC.

```
SELECT IdCliente
FROM Cliente
```

```
WHERE RFC = 'ABCD000000'
```

Para esta sentencia el plan de ejecución evaluaría que el índice disponible ordena los registros en la columna IdCliente y no sobre la columna RFC, por lo que la búsqueda de una coincidencia forzosamente ocuparía la lectura de todos los registros. El optimizer utilizaría el método de acceso de lecturas en escaneo completo del nivel hoja del índice que da orden físico a la tabla. Este método de acceso recibe nombres diferentes en cada implementación del RDBMS, como lo muestra la siguiente lista:

- **MySQL.** index record access
- **Oracle.** Full Tablescan, Fast Full Index Scan
- **SQL Server.** Ordered Clustered Index Scan, Ordered Covering Nonclustered Index Scan, Unordered Clustered Index Scan, Unordered Covering Nonclustered Index Scan
- **DB2.** Table Space scan, non-matching index scan

Este operador envía el conjunto solicitado, aún si se trata de un conjunto vacío, al siguiente operador en el plan de ejecución.

### Lecturas de escaneo parcial de B+-Trees en secuencia

El método de acceso de lecturas en escaneo parcial de un B+-tree en secuencia hace la lectura de un rango de los registros de datos bajo un criterio de selección. Existen varias condiciones por las que el optimizer decidirá utilizar este método para resolver una sentencia, una de ellas es la existencia de un índice que cuenta con las columnas necesarias para que el RDBMS ubique todos los registros solicitados para satisfacer la sentencia, pero existen otros factores.

En una tabla que cuenta con la organización física de un índice es posible que se haga una consulta de datos cuyo criterio de búsqueda contemple un rango de valores posibles en una columna que forma parte de la llave del índice. En una búsqueda de este tipo el optimizer hace uso del índice para ubicar el inicio del rango de datos solicitado y a partir de ese punto se leen todos los registros hasta que se alcanza el final de dicho rango.

En un ejemplo simple se describe la siguiente sentencia de usuario sobre una tabla que se encuentra organizada físicamente con un índice, donde la columna Fecha es la llave que da orden a la tabla, y se buscan registros de un cierto periodo de tiempo en particular.

```
SELECT IdFactura
FROM Factura
WHERE FechaEmision > '2011-11-11'
  AND FechaEmision < '2011-12-11'
```

Para esta sentencia el plan de ejecución evaluaría que el índice disponible ordena los registros en la columna Fecha, por lo que la búsqueda del rango se hará fácilmente con el índice definido sobre esta columna. Cuando el operador se ejecute buscará el primer registro del rango recorriendo el

índice en profundidad hasta el primer registro que cumple la condición, a partir de ese registro se leerán los registros siguientes hasta encontrar el primero que se encuentre más allá del rango definido, lo que servirá de indicación para que la lectura parcial termine. Este método de acceso recibe nombres diferentes en cada implementación del RDBMS, como lo muestra la siguiente lista:

- **MySQL.** Range record access
- **Oracle.** Index Range Scans, Index Range Scans Descending
- **SQL Server.** Clustered index seek + ordered partial scan, Covering nonclustered index seek + ordered partial scan, Nonclustered index seek + ordered partial scan + lookups against a clustered table
- **DB2.** Matching index scan, non-matching index scan

Este operador envía el conjunto de datos solicitado, incluso si el conjunto de datos está vacío, al siguiente operador en el plan de ejecución.

### Lecturas de búsqueda concreta en B+-Trees

El método de acceso de lecturas de búsqueda concreta en un B+-tree hace la lectura a través de los niveles del índice desde el nodo raíz hasta el nivel hoja donde se encuentra el registro de datos o el registro de referencia. Este tipo de búsqueda se utilizará cuando la sentencia y la definición del diccionario de datos aseguren la recuperación de un único registro.

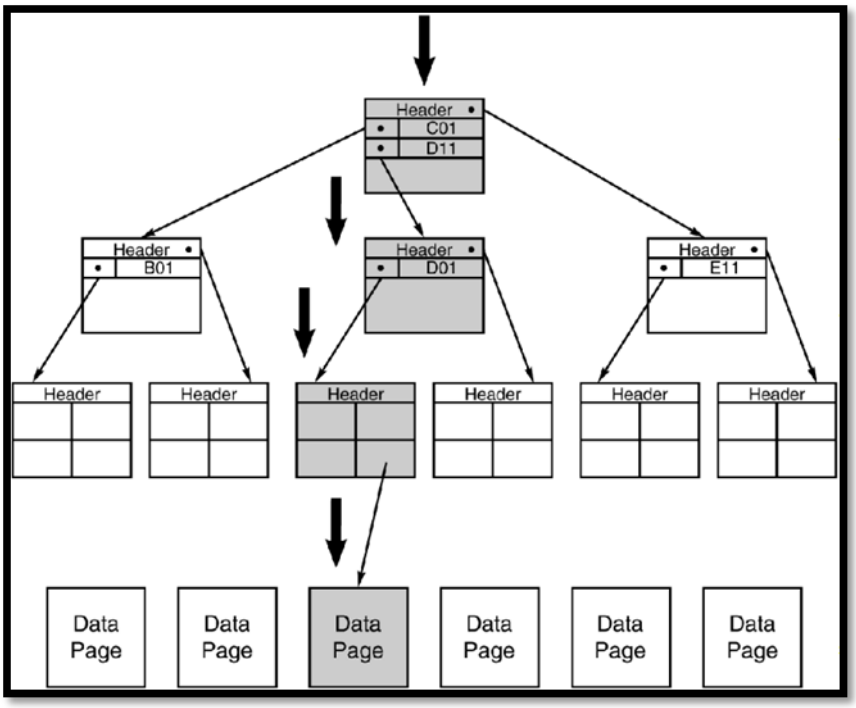
El mecanismo de ubicación del registro en el B+-tree es muy sencillo. Para observar mejor el mecanismo se pondrá un ejemplo sencillo donde se usaría este mecanismo.

Utilizando la siguiente sentencia de usuario, se tiene que existe una tabla de clientes organizada físicamente bajo un índice, donde la columna IdCliente es la llave primaria que da orden a la tabla.

```
SELECT IdCliente
FROM Cliente
WHERE IdCliente = 1000
```

Para esta sentencia el plan de ejecución evaluaría que el índice disponible ordena los registros en la columna IdCliente. El optimizer utilizaría el método de acceso de lecturas de búsqueda concreta en un B+-tree.

A continuación se ejemplifica el mecanismo de la lectura con una ilustración, este es el mecanismo usado por DB2 para recuperar registros de forma concreta, pero de hecho todos los manejadores hacen la misma operación.




El mecanismo de búsqueda concreta de DB2

Se observa que este método de acceso sólo requirió de la lectura a través de las 4 páginas que componen la profundidad del índice, esto fue mucho más óptimo que leer las 6 páginas de registros de datos que componen el nivel más bajo del índice. Se debe tomar en cuenta que la profundidad de un B+-tree es frecuentemente menor a la cantidad de páginas en el nivel de hoja, por lo cual la ubicación de cualquier registro requiere menor cantidad de lecturas en profundidad que la lectura completa del nivel de hojas. Este método de acceso recibe nombres diferentes en cada implementación del RDBMS, como lo muestra la siguiente lista:

- **MySQL.** Const record access, eq record access, eq_ref record access
- **Oracle.** Index Unique Scans
- **SQL Server.** Clustered Index Seek, Non Clustered Index Seek
- **DB2.** Direct index lookup, Index-only access

Este operador envía un conjunto de datos, aún si el conjunto es vacío, al siguiente operador en el plan de ejecución.

### Lecturas en escaneo o búsqueda concreta en B+-Trees con lookup

En el caso de que el optimizer decida usar un índice que mantiene el orden lógico de una tabla para resolver una consulta, si dicha consulta requiere más columnas que las que están definidas en el índice entonces la simple búsqueda del registro en el índice no es suficiente. Por ello, para completar la consulta, el optimizer tendrá como opción el decidir la lectura de otro índice de donde tomará el resto de las columnas necesarias para la resolución de la consulta o incluso

buscará hacer la lectura de la estructura de datos que contiene los registros de datos completos, es decir, la lectura de los registros de datos en el heap o en el B+-tree que da orden físico de la tabla.

En cada caso, la cantidad de lecturas requeridas para satisfacer la consulta se incrementará. Este tipo de lecturas son casos especiales de los mecanismos de lectura vistos previamente, con la variante de que se necesita hacer lecturas extra en otras estructuras de datos para completar la solicitud, en algunas ocasiones estos mecanismos son más rápidos que las lecturas en escaneo, sin embargo, es importante construir estructuras de datos útiles que reduzcan la necesidad de lecturas complementarias, ya que estas provocarían que se soliciten datos al subsistema de IO y/o que los datos en el subsistema de memoria sufran algún tipo de presión por tratar de alojar grandes cantidades de datos en los buffers. Este método de acceso recibe nombres diferentes en cada implementación del RDBMS, como lo muestra la siguiente lista:

- **MySQL.** index_merge record access
- **Oracle.** Operadores Index join
- **SQL Server.** Alguna rama del plan donde se use un operador lookup, alguna rama del plan de ejecución que use un operador join para juntar los registros recuperados entre 2 operadores de lectura aplicados a la misma tabla
- **DB2.** Direct index lookup

### Ordenamiento de conjuntos de datos

El operador de ordenamiento de datos es solicitado explícitamente o implícitamente, es decir, cuando una sentencia tiene una cláusula de ordenamiento de datos como la cláusula ORDER BY se habla de una solicitud explícita, por otro lado, el optimizer usa este operador implícitamente para operar conjuntos que requiere ordenar para que sean el insumo de otros operadores dentro del plan de ejecución a fin de reducir los tiempos de respuesta del manejador al resolver la sentencia.

Cuando los datos requeridos por la consulta son los mismos que definen a algún índice de la tabla entonces las operaciones de ordenamiento, también conocidas como sort, no son requeridas en el plan de ejecución, debido a que los datos se recuperaron en el orden necesario desde el método de acceso de lectura.

El ordenamiento de datos se hará en los buffers del subsistema de memoria si el espacio disponible es suficiente para hacer esta operación. Sin embargo, si el conjunto de datos es tan grande que no es ordenable en memoria entonces parte del conjunto o todo él tendrá que ordenarse en el subsistema de IO en áreas temporales dentro de los archivos de datos, ya sea de la base de datos de la que se está recuperando el conjunto desordenado o dentro de algún archivo que el RDBMS disponga para este tipo de operaciones.

Debido a que el mecanismo de ordenamiento requiere de la lectura y escritura de información se sugiere que el operador sea utilizado sólo en los casos necesarios para ayudar a reducir el costo de estas operaciones.

Estos operadores de ordenamiento también son usados para preparar los conjuntos de datos cuando existen operaciones GROUP BY que requieren que el conjunto sea operado en grupos de registros y para ello es útil un orden a fin de evitar lecturas aleatorias de los datos.

En este caso el operador de ordenamiento enviará el mismo número de registro que recibió al siguiente operador en el plan de ejecución a menos que el ordenamiento sea usado para agregar y / o discriminar registros. Este método de acceso recibe nombres diferentes en cada implementación del RDBMS, como lo muestra la siguiente lista:

- **MySQL.** filesort
- **Oracle.** SORT UNIQUE, SORT GROUP BY, SORT ORDER BY
- **SQL Server.** Sort
- **DB2.** SORT

### Agregación de conjuntos de datos

Los operadores de agregaciones de conjuntos son colocados en los planes de ejecución cuando en el código se encuentran las funciones SUM(), MAX(), MIN(), AVG(), COUNT(), GROUP BY, etc. Estas operaciones requieren consumir los registros de datos ordenados en grupos, así será más fácil para el RDBMS hacer las operaciones en los grupos definidos en la cláusula GROUP BY.

Las operaciones de agregación de datos se harán en los buffers del subsistema de memoria si el espacio disponible es suficiente para hacer esta operación. Sin embargo, si el conjunto de datos no es operable en memoria entonces parte del conjunto o todo él tendrán que escribirse y operarse en el subsistema de IO en áreas temporales dentro de los archivos de datos, ya sea de la base de datos de la que se está recuperando el conjunto de datos o dentro de algún archivo que el RDBMS disponga para este tipo de operaciones.

Debido a que el mecanismo de agregación requiere de la lectura y escritura de información se sugiere que sólo se opere la información requerida.

En este caso el operador de agregación devuelve el conjunto resultante de la agregación. Este método de acceso recibe nombres diferentes en cada implementación del RDBMS, como lo muestra la siguiente lista:

- **MySQL.** index for group-by, temporary, Distinct
- **Oracle.** SORT GROUP BY
- **SQL Server.** Stream Aggregation, Hash Aggregation
- **DB2.** SORT, GROUPBY

### Join de conjuntos de datos

El optimizer es capaz de utilizar operaciones de reunión de conjuntos, o join por su traducción al inglés, para atender solicitudes de usuario. El módulo optimizer traduce sentencias DML del lenguaje SQL hacia su forma de operaciones del álgebra relacional. El conjunto de las operaciones

del algebra relacional que el optimizer soporta son las 8 que Edgar Codd[39] definió y son las siguientes:

- Unión
- Intersección
- Diferencia
- Producto cartesiano
- Restricción
- Proyección
- Agregación
- División

Cada manejador implementa varios algoritmos para hacer el join de los conjuntos, y la elección de algún algoritmo dependerá de la lógica de costos que el RDBMS utilice para definir un plan de ejecución óptimo. De forma general se dirá que el conjunto de algoritmos de join son algoritmos de iteración de elementos de un conjunto, y dichos algoritmos comparten similitudes con el algoritmo de ordenación de burbuja, o bubble sort. Lo importante a destacar en los algoritmos de join es que la elección de alguno de ellos dependerá del costo que represente en la ejecución y de las estructuras de datos existentes que resulten útiles para resolver la sentencia. Cada algoritmo de join tiene su propia representación como operador para su identificación dentro de los planes de ejecución.

De forma general, para la ejecución de un algoritmo join se requieren de 2 conjuntos de datos, que pueden ser vistos con la representación matemática de un diagrama de Venn. Dependiendo de los datos que se requieran entonces los registros se localizarán en alguna de las regiones definidas por ambos conjuntos.

En algunas ocasiones el optimizer evalúa la necesidad de agregar operadores para el tratamiento del conjunto de datos origen para obtener otro conjunto de datos que servirá de entrada al operador de join. Un ejemplo de estos operadores se observa en planes de ejecución que requieren de un conjunto con el resumen de los registros de un conjunto origen mediante el cálculo de llaves hash (tabla hash). Otro ejemplo son los planes de ejecución que generan bitmaps a partir de un conjunto origen, ya que es más rápida la ejecución de operadores join con estructuras de datos del tipo bitmap que con registros completos.

El patrón general del plan de ejecución que usa un operador de join es la siguiente:

1. El optimizer evalúa la factibilidad de que el conjunto A y que el conjunto B sean operados por el operador de join sin ningún tratamiento previo

[39] Ben-Gan, Itzik (2009). *Inside Microsoft SQL Server 2008: T-SQL Querying*. United States of America: Microsoft Press. pp. 91

2. Si los conjuntos A y/o B deben ser operados de alguna forma entonces tendremos operadores previos al join que harán la adecuación de los conjuntos transformándolos en los nuevos conjuntos A' y B'
3. El operador de join recibe un primer registro del conjunto A o A' según sea el caso
    a. Si es necesario se hace alguna operación sobre el registro para adaptarlo a las necesidades del algoritmo
4. El registro, adaptado o no, es comparado contra un primer registro del conjunto B o B' para definir si cumple con el criterio definido en la sentencia SQL
5. Si los registros cumplen el criterio se anexan ambos como un solo registro. El registro es ingresado a un nuevo conjunto de datos denominado C, que es el conjunto resultante
6. Se iteran todos los registros mediante la lógica anterior
7. El conjunto C es el conjunto de datos que el operador de join enviará al siguiente operador dentro del plan de ejecución

Los operadores join construyen conjuntos resultantes, cada conjunto resultante se colocará en el subsistema de memoria si es que este tiene suficiente espacio libre para alojarlo, si no es así, entonces el conjunto se materializará en el subsistema de IO, lo que incrementa los tiempos requeridos para resolver una solicitud de usuario.

Por otro lado, la cantidad de iteraciones también influye en los tiempos de respuesta, por lo que siempre se debe iterar sólo los datos que realmente van a ser utilizados por el usuario. Los algoritmos de join se explicarán a continuación, contemplando sólo aquellos que comparten 2 o más RDBMS.

### *Nested loop*

El algoritmo usado en un nested loop es el más usado por los manejadores de bases de datos, de hecho para MySQL no existe ningún otro mecanismo de join. En este algoritmo se utiliza el siguiente pseudocódigo para iterar los registros de 2 conjuntos de datos.

```
for r1 in (select rows from inner_table that match filters) loop
   for r2 in (select rows from outer_table that match r1 from
inner_table) loop
      output r1 and r2 that match filters
   end loop
end loop
```

Este algoritmo es muy eficiente para grupos pequeños de datos, y como se observa, la cantidad de lecturas requeridas para conjuntos muy grandes son demasiadas, lo que eleva notablemente los tiempos de respuesta, por ello para los RDBMS comerciales no es la única opción de join. El algoritmo esta implementado en los siguientes RDBMS:

- DB2
- Oracle
- SQL Server

- MySQL

### *Merge join*

El algoritmo usado en un merge join requiere que los registros en ambos conjuntos estén ordenados, de tal forma que resulta muy eficaz la ejecución del algoritmo.

```
r1 = first row in sorted_inner_table
r2 = first row in sorted_outer_table
while not at the end of either sets
begin
  if r1 join r2
  begin
    output r1 and r2 that match filters
    r2 = next row in sorted_outer_table
  end
  else if r1 < r2
    r1 = next row in sorted_inner_table
  else
    r2 = next row in sorted_outer_table
end
```

Este algoritmo es muy eficiente para grupos de datos que tienen gran cantidad de registros en el conjunto del lado derecho, sin embargo este mismo algoritmo, en su forma de operador, es incluido en planes de ejecución donde la eficiencia de este tipo de join es mejor que cualquier otro. El algoritmo está implementado en los siguientes RDBMS:

- DB2
- Oracle
- SQL Server

### *Hash join*

El algoritmo usado en un hash join requiere del tratamiento de todos los registros de uno de los conjuntos con una función de hash para dar origen a una tabla de llaves hash. Una vez construida esta tabla el proceso iterativo toma cada uno de los registros del otro conjunto para operarlos mediante una función hash que permita la comparación de los registros pero a nivel de su representación hash. El siguiente pseudocódigo representa las operaciones descritas.

```
for r1 in (select rows from inner_table that match filters) loop
  calculate hash value on r1
  insert hash value of r1 into the hash_table
end loop
for r2 in (select rows from outer_table) loop
  for r1 in (select rows from hash_table) loop
    if r1 joins with hash value of r2
      output r1 and r2 that match filters
  end loop
```

```
end loop
```

Este algoritmo es uno de los más complicados pero para ciertos escenarios es muy eficiente, sobre todo cuando el conjunto usado para construir la tabla hash tiene muchos registros que comparten el mismo valor en las columnas usadas para el criterio del join. El algoritmo está implementado en los siguientes RDBMS:

- Oracle
- SQL Server
- MySQL[40]

### Áreas temporales y mecanismos de tratamiento de conjuntos de datos

Las operaciones de conjuntos que operan grandes cantidades de registros requieren de áreas temporales para ser llevadas a cabo, lo que implica un tiempo para posicionar, recuperar e iterar los datos del conjunto en dichas áreas.

En el caso de que el área temporal se materialice en el subsistema de memoria, se debe cuidar que el espacio requerido no comprometa ni presione a otros objetos ya alojados en el subsistema, es por ello que se debe entender el impacto de cada operador en el plan de ejecución.

Para las operaciones que no son almacenables en el subsistema de memoria por restricciones de espacio se cuenta con otro mecanismo que es el posicionamiento en el subsistema de IO. El trabajo con áreas temporales incrementa de manera notable los tiempos de respuesta, por ende debe cuidarse su buen uso.

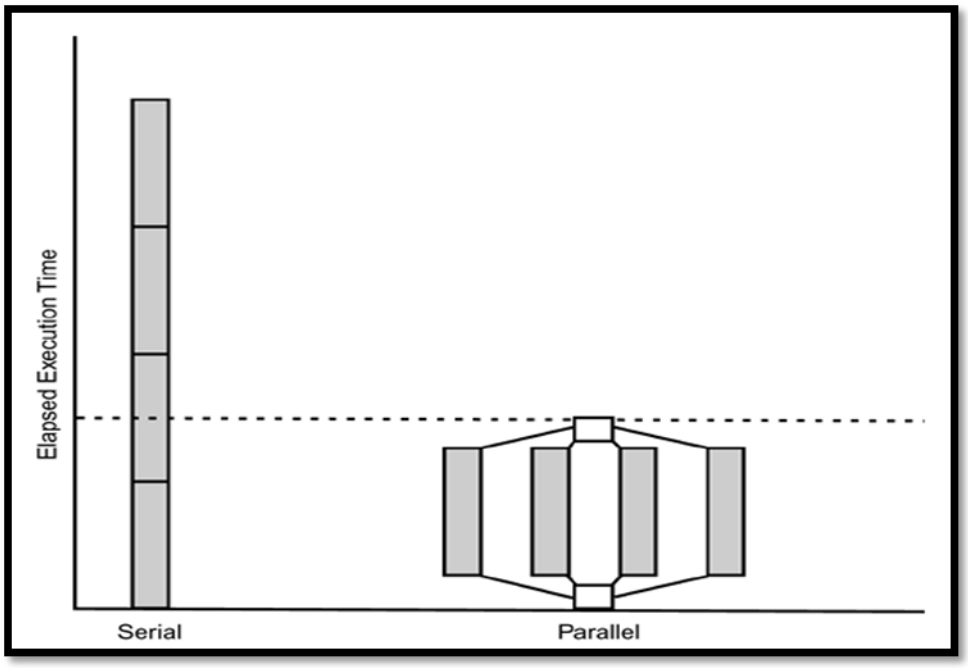
### El procesamiento en paralelo

El optimizer es capaz de definir si el costo evaluado como el tiempo necesario para satisfacer una sentencia se verá mejorado si distribuye el trabajo entre varios procesadores. Una vez que determina que el tiempo de respuesta del manejador se verá beneficiado por este tipo de procesamiento entonces las tareas de un operador se dividen entre varios procesadores del subsistema de CPUs. La siguiente figura muestra como DB2 decide y conceptualiza el ahorro en el tiempo de ejecución al utilizar paralelismo[41].

---

[40] Como tal MySQL sólo soporta Nested Loops como el único operador de join, sin embargo es posible emular un operador Hash Join utilizando Hash Indexes, los cuales sólo tienen soporte explícito para el motor de almacenamiento “Memory”.

[41] Mullins, Craig S. (2004). *DB2 Developer's Guide (Fourth ed.)*. United States of America: Sams Publishing. pp. 753 - 759




La reducción del tiempo de ejecución de una tarea en paralelo

En todo caso se desea que un RDBMS que soporta un sistema tipo OLTP no ejecute la mayoría de sus sentencias en paralelo, ya que este comportamiento indicaría que las sentencias resultan ser demasiado costosas en tiempo y por ello se decide el uso de este mecanismo. Un grupo de estas sentencias ejecutándose de forma concurrente elevará el uso de los CPUs del subsistema de procesadores, lo que hará que en general un servidor presente lentitudes en la atención de procesos o inclusive habrá un peligro latente de tener una caída del servicio por la falta de atención de los procesadores a otros procesos críticos del sistema operativo.

## VI. Listado del conjunto de sentencias analizables

Los ejemplos de los listados de control de sentencias analizables se muestran para los análisis correspondientes.

| Análisis de calidad de código | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Listado | Solicitud de más información de la que se requiere | Múltiples lecturas del mismo conjunto de datos | Objetos temporales | Forzar al optimizer | Uso de estructuras de datos útiles | Reutilizabilidad del plan de ejecución | Transacciones con mayor duración | Sentencias en pares |
| Sentencias con la mayor cantidad de lecturas | | | | | | | | |
| Sentencia 1 | | | | | | | | |
| Sentencias con la mayor cantidad de escrituras | | | | | | | | |
| Sentencia 1 | | | | | | | | |
| Sentencias con el mayor uso de procesador | | | | | | | | |
| Sentencia 1 | | | | | | | | |
| Sentencias con el mayor consumo de memoria | | | | | | | | |
| Sentencia 1 | | | | | | | | |
| Sentencias con la mayor cantidad de datos operados | | | | | | | | |
| Sentencia 1 | | | | | | | | |
| Sentencias con las mayores esperas por el subsistema de IO | | | | | | | | |
| Sentencia 1 | | | | | | | | |
| Sentencias con las mayores esperas por el subsistema de memoria | | | | | | | | |
| Sentencia 1 | | | | | | | | |
| Sentencias con las mayores esperas por el subsistema de CPUs | | | | | | | | |
| Sentencia 1 | | | | | | | | |
| Sentencias con las mayores esperas por el subsistema de red | | | | | | | | |
| Sentencia 1 | | | | | | | | |

| Análisis de planes de ejecución | | | | | |
|---|---|---|---|---|---|
| **Listado** | **Operadores con exploración innecesarias** | **Operadores JOIN que reciben muchos datos** | **Operadores JOIN con tratamiento de conjuntos** | **Operadores SORT a conjuntos de datos grandes** | **Operadores de agregación con discriminación** |
| **Sentencias con la mayor cantidad de lecturas** | | | | | |
| **Sentencia 1** | | | | | |
| **Sentencias con la mayor cantidad de escrituras** | | | | | |
| **Sentencia 1** | | | | | |
| **Sentencias con el mayor uso de procesador** | | | | | |
| **Sentencia 1** | | | | | |
| **Sentencias con el mayor consumo de memoria** | | | | | |
| **Sentencia 1** | | | | | |
| **Sentencias con la mayor cantidad de datos operados** | | | | | |
| **Sentencia 1** | | | | | |
| **Sentencias con las mayores esperas por el subsistema de IO** | | | | | |
| **Sentencia 1** | | | | | |
| **Sentencias con las mayores esperas por el subsistema de memoria** | | | | | |
| **Sentencia 1** | | | | | |
| **Sentencias con las mayores esperas por el subsistema de CPUs** | | | | | |
| **Sentencia 1** | | | | | |
| **Sentencias con las mayores esperas por el subsistema de red** | | | | | |
| **Sentencia 1** | | | | | |

| Análisis de mantenimiento y diseño de estructuras de datos | | | | | | |
|---|---|---|---|---|---|---|
| **Listado** | **Tablas con estructuras de datos que ocupen más espacio** | **Falta de la actualización de estadísticas** | **Omisión de un índice de orden físico** | **Omisión de índices de orden lógico** | **Índices duplicados** | **Tablas con múltiples índices** |
| **Sentencias con la mayor cantidad de lecturas** | | | | | | |
| **Sentencia 1** | | | | | | |
| **Sentencias con la mayor cantidad de escrituras** | | | | | | |
| **Sentencia 1** | | | | | | |
| **Sentencias con el mayor uso de procesador** | | | | | | |
| **Sentencia 1** | | | | | | |
| **Sentencias con el mayor consumo de memoria** | | | | | | |
| **Sentencia 1** | | | | | | |
| **Sentencias con la mayor cantidad de datos operados** | | | | | | |
| **Sentencia 1** | | | | | | |
| **Sentencias con las mayores esperas por el subsistema de IO** | | | | | | |
| **Sentencia 1** | | | | | | |
| **Sentencias con las mayores esperas por el subsistema de memoria** | | | | | | |
| **Sentencia 1** | | | | | | |
| **Sentencias con las mayores esperas por el subsistema de CPUs** | | | | | | |
| **Sentencia 1** | | | | | | |
| **Sentencias con las mayores esperas por el subsistema de red** | | | | | | |
| **Sentencia 1** | | | | | | |

| Análisis de concurrencia | | | |
|---|---|---|---|
| **Listado** | **Candados impuestos por mucho tiempo** | **Procesos esperando por candados impuestos** | **Deadlocks** |
| **Sentencias con la mayor cantidad de lecturas** | | | |
| **Sentencia 1** | | | |
| **Sentencias con la mayor cantidad de escrituras** | | | |
| **Sentencia 1** | | | |
| **Sentencias con el mayor uso de procesador** | | | |
| **Sentencia 1** | | | |
| **Sentencias con el mayor consumo de memoria** | | | |
| **Sentencia 1** | | | |
| **Sentencias con la mayor cantidad de datos operados** | | | |
| **Sentencia 1** | | | |
| **Sentencias con las mayores esperas por el subsistema de IO** | | | |
| **Sentencia 1** | | | |
| **Sentencias con las mayores esperas por el subsistema de memoria** | | | |
| **Sentencia 1** | | | |
| **Sentencias con las mayores esperas por el subsistema de CPUs** | | | |
| **Sentencia 1** | | | |
| **Sentencias con las mayores esperas por el subsistema de red** | | | |
| **Sentencia 1** | | | |

# VII. Introducción al modelo relacional

El modelo relacional fue concebido en la década de 1960 por Edgar F. Codd, quien trabajó para IBM. Es simplemente una forma rigurosa de cómo los usuarios perciben y trabajan con los datos.

El modelo aborda los tres aspectos principales del tratamiento de datos de la siguiente manera[42]:

- Estructural
  Los datos se perciben por el usuario como tablas y nada más que tablas.
- Manipuladora
  Los usuarios manipulan los datos con un conjunto abierto de operadores relacionales. Los operadores constituyen el álgebra relacional.
- Integridad
  Las tablas habrán de satisfacer las restricciones de integridad definida.

El aspecto estructural también se expresa por el principio de información, que afirma que toda la información en una base de datos relacional se expresa en forma de una y sólo una, y que sólo se permite su definición mediante valores explícitos en las columnas que definen las filas de una tabla.

## Las relaciones y las tuplas

Una relación es el objeto matemático que los profesionales de las bases de datos llaman tabla.

Los elementos de una relación representan instancias de alguna entidad real, como una persona, un lugar, una cosa o un evento. La relación es el conjunto de estos elementos, y estos elementos son matemáticamente denominados tuplas. El modelo relacional utiliza una noción más general de la definición de una tupla que la que se conceptualiza para la teoría de conjuntos.

Para la definición del modelo relacional las tuplas tienen partes bien definidas pero sus componentes están identificados por el nombre del atributo. Esta es la diferencia principal con la definición de la teoría de conjuntos donde cada elemento de la tupla está identificado por la posición ordinal.

Una tupla está formada por un conjunto de atributos, cada uno de los cuales está representado por tres componentes:

- El nombre del atributo
- El tipo de atributo

[42] Date, C. J (2004). *An introduction to database systems (Eight Edition)*. United States of America: Adisson-Wesley. pp. 60.

- El valor del atributo

Para el modelo relacional un conjunto desordenado de tuplas tiene el mismo encabezado.

El conjunto de los nombres de los atributos y sus tipos son el encabezado de una tupla. El encabezado de una tupla es entonces el formato en el que cada tupla estará definida. Las propiedades de las tuplas son las siguientes:

- Cada atributo nombrado de una tupla contiene exactamente un valor del tipo indicado para dicho atributo
- Los atributos de una tupla no tienen orden, por consiguiente cada atributo tendrá un nombre distinto, la razón es que no es factible referirnos a un tributo por su posición ordinal
- Un subconjunto de los atributos de una tupla también se considera una tupla, sólo que la nueva tupla tendrá menos atributos

Una relación consiste pues de un conjunto de tuplas con el mismo encabezado. Un ejemplo de una tupla y su agrupación en una relación sería el de la siguiente lámina:

Encabezado {

| Tupla | | | |
|---|---|---|---|
| Nombre del atributo | IdReproductor | Descripcion | Costo |
| Tipo del atributo | Entero | Cadena (50) | Decimal |
| Valor | 1 | MP4 Player | 5000.00 |

Encabezado de tupla {

| Reproductor (relación) | | | |
|---|---|---|---|
| Nombre del atributo | IdReproductor | Descripción | Costo |
| Tipo del atributo | Entero | Cadena (50) | Decimal |
| Tupla | 1 | MP4 Player | 5000.00 |
| Tupla | 2 | MP3 Sport | 3000.00 |
| Tupla | 3 | AVI Player | 6000.50 |
| Tupla | 4 | DivX Player | 4500.00 |

Una tupla y una relación de tuplas

Se observa que la tupla tiene un encabezado que la describe, mientras que en la relación tenemos varias tuplas descritas bajo el mismo encabezado.

## El significado de las relaciones

Cada relación representa una entidad real, como una persona, un lugar, una cosa o un evento. Una entidad es una cosa que es claramente identificada y que es de interés para el negocio.

Cada representación de una entidad es identificable inequívocamente, un hecho que hace posible utilizar una relación para representar una entidad. Cada representación de una entidad desempeña un papel importante en la aplicación o el sistema en que es utilizada.

Este es el concepto abstracto:

> ***Una base de datos sólo tiene clases entidad (y los atributos de las entidades) que tienen una razón para estar allí. Cada representación de una entidad es descrita por uno o más atributos. Las relaciones son asociaciones entre entidades. Una relación es un subconjunto del producto cruz de los conjuntos de entidades involucradas en la relación. Los atributos dan alguna información sobre entidades que son de interés para la aplicación.***

El párrafo anterior define el significado de las relaciones y de las entidades, así como la relación entre ellos[43].

El enfoque de entidad-relación (ER) es también el más ampliamente utilizado para el modelado de entidades en las bases de datos relacionales, así como para el modelado de relaciones y sus atributos.

### Integridad de los datos

La integridad de los datos es fundamental para una base de datos relacional. Un RDBMS por definición cumple las reglas de integridad. Por lo que las reglas son parte integrante de una base de datos.

Las restricciones declarativas son el mecanismo utilizado para crear reglas que permitan asegurar la integridad de los datos con apego a las reglas de negocio.

Las reglas del negocio representan cualquier cosa, como reglas de cardinalidad o frecuencia (número máximo o mínimo de tuplas almacenables en una relación en cualquier momento), reglas de derivación de datos (como el calcular el estado de eventos), reglas de subconjunto (una relación es el resultado de un subconjunto de las tuplas de otra relación), reglas de inclusión (sólo se recibirán lotes de un proveedor en un período donde este tenga un contrato válido), reglas de proceso (el orden de los eventos que es necesario que ocurran previamente) y mucho más.

Es del diseñador de bases de datos y del diseñador de las aplicaciones la responsabilidad de decidir dónde aplicar las reglas del negocio. Es válido y práctico definir firmemente las restricciones declarativas en la base de datos relacional ya que estas son las características propias de todas las bases de datos relacionales.

---

[43] Definición concluida de la obra de Peter Chen. Chen, Peter (1976). *The Entity-Relationship Model – Toward a Unified View of Data*. United States of America: MIT Press. pp. 10 - 20

Las restricciones se clasifican en otras formas también. Por ejemplo, de acuerdo al tipo de objeto que restringen, ya sea un tipo de dato, un atributo, una relación o una restricción de base de datos. Otro ejemplo de clasificación surge del momento en que se hace la validación de la restricción, ya sea de forma inmediata o diferida basada en cuando la restricción se forzará ya sea inmediatamente o al final de la transacción.

Nótese que de acuerdo a la regla que dice "No debe haber ningún valor en la base de datos en ningún momento que viole un predicado de restricción"[44] sólo las restricciones inmediatas tienen cabida dentro de las bases de datos relacionales. Esto significa que una restricción se forzará necesariamente en los límites de una sentencia DML[45], no al final de la transacción o incluso después.

### Integridad de entidad

Las tablas en una base de datos son la representación física de lo que el modelo relacional define como las relaciones y cada fila de una tabla representa las tuplas de una relación. La integridad de una entidad consiste en identificar inequívocamente las filas en una tabla.

Para ello debe haber una combinación de columnas que físicamente representan atributos y que identifiquen inequívocamente a una fila. El conjunto mínimo de columnas que identifican cada fila se denomina clave. Cada tabla tendrá incluso múltiples combinaciones de columna única — en otras palabras, varias claves candidatas. Depende del diseñador de la base de datos seleccionar una de esas claves como su referencia principal para cada fila y llamarlo clave principal o llave primaria.

Como ejemplo de la integridad de entidad tenemos una tabla de empleados, donde por diseño cada empleado tiene un ID numérico único que lo identifica de entre el resto de los empleados y de hecho esta es la llave primaria, mientras que para otro sistema en otra empresa los empleados son identificados utilizando el número de seguridad social federal como la llave primaria.

### Integridad referencial

Una clave externa es un conjunto de columnas de una tabla cuyos valores coinciden con algunas claves de otra tabla, en otras palabras, una copia de una clave de una relación en otra. A las claves externas también se les conoce como llaves foráneas.

Las llaves foráneas denotan asociaciones entre las relaciones; son el pegamento que mantiene unidas las relaciones en una base de datos. La aplicación de reglas de llave foránea se expresa brevemente: en una base de datos no deben existir tuplas con llaves foráneas que no tengan

---

[44] Date, C. J (2004). *An introduction to database systems (Eight Edition).* United States of America: Adisson-Wesley. pp. 260

[45] Una sentencia de tipo DML (Data Manipulation Language) en el ANSI de SQL es aquella que permite la manipulación de tuplas, mientras que las sentencias del tipo DDL (Data Definition Language) son aquellas que permiten la modificación de definición y estructuras de los objetos en la base de datos.

contrapartes en las entidades correspondientes en ningún momento. Las llaves foráneas mantienen referencias entre las relaciones, en otras palabras, mantienen la integridad referencial.

## VIII. Niveles de RAID

Para tener un mejor entendimiento de porque este subsistema es considerado el más lento se explicará el diseño del subsistema a nivel físico así como las tecnologías que están provocando la revolución en los diseños de los RDBMS.

Disponibilidad y performance son 2 factores que se deben tomar en cuenta al momento de diseñar una estrategia de configuración del subsistema de IO. La única cosa segura al utilizar discos para almacenamiento es que, no importa el fabricante, todos los discos llegarán a un punto en que fallarán.

Para evitar que las operaciones de la empresa se vean interrumpidas o incluso condenadas a desaparecer sin una posible recuperación o que el tiempo de recuperación del subsistema sea tan alto que se pierdan grandes sumas de dinero para el negocio los fabricantes de soluciones de almacenamiento han desarrollado estrategias de almacenamiento que permitan la tolerancia a fallos del hardware asociado al subsistema. Aun así, la tolerancia a fallos impone cuotas al desempeño, lo cual hace necesario contar con un conocimiento de cada uno de los mecanismos de tolerancia a fallos para entender mejor como esto afecta o no al desempeño.

Un dispositivo de arreglo redundante de disco independientes (*redundant array of independent disk, RAID*) es una forma popular de configurar un disco lógico de gran tamaño formado de un conjuntos de discos más pequeños. La idea es simple, combinar varios discos de bajo costo en un arreglo ordenado para obtener un mejor desempeño y seguridad en los accesos a datos.

Esto permite reemplazar un gran disco caro por varios discos pequeños mucho más baratos. Los datos están divididos en pequeños trozos de igual tamaño (denominado volumen seccionado o *stripe volume*), usualmente de 32 KB o 64 KB, y un trozo es escrito en cada disco en la distribución exacta determinada por el nivel adoptado por el RAID. Cuando la información es leída el proceso es revertido, dando la apariencia de tener y hacer uso de un gran disco en lugar de muchos discos pequeños.

Los dispositivos de almacenamiento en arreglo proveen redundancia, si un disco en el RAID falla ocurrirá una reconstrucción automática e inmediata con los datos en el resto de los dispositivos en el arreglo.

Cuando se habla de desempeño del subsistema de IO dos factores son interesantes: la taza de transferencia y el número de operaciones de IO permitidas por segundo. La taza de transferencia se refiere a la eficiencia con la que los datos se desplazan en el controlador del sistema de discos.

Para las operaciones de IO, mientras más operaciones sean soportadas por el sistema de discos en un periodo de tiempo tendremos un mejor desempeño.

La disyuntiva inherente a los sistemas RAID se da entre el performance y la rentabilidad. Por este hecho existen 2 esquemas fundamentales de organización de los RAID, seccionamiento del arreglo de disco y el espejeo de los discos en el arreglo, para mejorar el desempeño e incrementar la rentabilidad.

Los esquemas de espejeo o mirroring involucran duplicación completa de datos. A pesar de que los sistemas RAID sin espejo también involucran redundancia es necesario decir que no es de tan alto nivel como en los sistemas RAID espejeados. La redundancia en sistemas RAID no espejeados se basan en el hecho de que necesitan información de "paridad" para la reconstrucción de volúmenes en caso de problemas de funcionamiento del arreglo.

La siguiente lista describe los sistemas de RAID más comúnmente utilizados. Excepto por el RAID 0 todos los otros sistemas son redundantes.

- RAID 0: Striping
  Estrictamente hablando, esto no es realmente un nivel de RAID, ya que la creación de volúmenes seccionados no proporciona ningún tipo de protección de datos. Los datos se dividen en trozos y se colocan en varios de los discos que componen el arreglo de discos. Un volumen seccionado es entonces el conjunto de todos los bloques en los discos.

  Suponga que el tamaño del fragmento o del segmento es de 8 KB. Si existen tres discos en el RAID y tenemos un bloque de datos de 24KB que se van a escribir en el sistema RAID, los primeros 8 KB se escriben en el disco primero, el segundo de 8 KB se escriben en el disco segundo, y los 8KB finales serían escritos en el último disco.

  Debido a que las entradas y salidas se distribuyen en varios discos, el rendimiento de los sistemas RAID 0 es bastante alto. Por ejemplo, escribir un archivo de 800 KB en un conjunto RAID de ocho discos con un tamaño de banda de 100 KB se hará en aproximadamente una octava parte del tiempo que se tardaría en hacer la misma operación en un sólo disco. Sin embargo, ya que no hay redundancia, la pérdida de una sola unidad resultará en la pérdida de todos los datos, los datos se almacenan de forma secuencial en los bloques. RAID 0 es sobre todo útil para el rendimiento pero no presta atención a la protección.

- RAID 1: Mirroring
  En RAID 1, todos los datos se duplican, o se espejean, en uno o más discos. El rendimiento de un sistema RAID 1 es más lento que un sistema RAID 0 porque las transacciones de entrada se realizan sólo cuando se escribe correctamente en todos los discos duplicados. La fiabilidad de arreglos en espejo es alta, aún si ocurriese el fallo de un disco en el

conjunto no se produciría ninguna pérdida de datos. El sistema seguiría funcionando en tales circunstancias, y habría forma y tiempo para regenerar el contenido de los discos perdidos copiando los datos de los otros discos. RAID 1 está orientado a la protección de los datos, con un rendimiento a un segundo plano. Sin embargo, de todos los sistemas RAID, RAID 1 sigue ofreciendo el mejor rendimiento.

Es importante señalar que para RAID 1 la adquisición de un número n de discos significa que en realidad sólo n/2 discos serán asignados para el almacenamiento en el sistema. El rendimiento de lectura mejora en un sistema RAID 1, ya que los datos se escanean en paralelo.

- RAID 2: Striping con detección y corrección de errores
  RAID 2 utiliza seccionamiento con detección de errores y cuenta con capacidades adicionales de corrección. Los segmentos garantizan un alto rendimiento, y los métodos de corrección de errores permiten garantizar la fiabilidad.

  Sin embargo, el mecanismo que se utiliza para corregir errores es voluminoso y ocupa mucho espacio en disco. Este es un sistema de almacenamiento costoso e ineficiente.

- RAID 3: Striping con paridad dedicada
  Los sistemas en RAID 3 también son sistemas en secciones, con un disco adicional de paridad que contiene la información necesaria para la corrección de errores en el stripe. La paridad implica el uso de algoritmos para calcular los valores que permiten la reconstrucción de los datos en un nuevo disco ante la pérdida de datos en uno de los discos del sistema.

  Las operaciones de Input/Output son más lentas en sistemas RAID 3 que en los sistemas de seccionamiento puro, como RAID 0, ya que la información también tiene que ser escrita en el disco de paridad. Los sistemas RAID 3 sólo procesan una petición de IO a la vez.
  Sin embargo, RAID 3 es un sistema más sofisticado que RAID 2 e implica menor consumo y carga que RAID 2. Lo único que se necesita para el arreglo es una unidad de disco adicional, además de las unidades que contienen los datos. Si un disco falla, el arreglo sigue funcionando satisfactoriamente, mientras tanto la unidad que ha fallado entrará en un proceso de reconstrucción con la ayuda de la información de paridad almacenada en el disco de corrección de errores que almacena la paridad adicional.
  RAID 5 con secciones pequeñas proporciona un mejor rendimiento que los arreglos de disco RAID 3.

- RAID 4: Striping modificado con paridad dedicada
  Las secciones de los sistemas RAID 4 se realizan en trozos mucho más grande que en los sistemas de RAID 3, lo que permite que el sistema de IO procese múltiples peticiones al

mismo tiempo. En los sistemas RAID 4, los discos individuales se acceden de forma independiente, a diferencia de los sistemas RAID 3, lo que conduce a un rendimiento muy superior al leer los datos de los discos.

La escritura es una historia diferente bajo esta configuración. Cada vez que se necesite llevar a cabo una operación de escritura, los datos de paridad para el disco correspondiente deberán ser actualizados antes de que los nuevos datos sean escritos. Por lo tanto, la escritura es una operación lenta en definición, y el disco de paridad se convertirá en un cuello de botella.

- RAID 5: Striping modificado con paridad intercalada
  Bajo esta configuración de arreglo del disco, los datos y la información de paridad se intercalan a través del arreglo de discos. Las escrituras en RAID 5 tienden a ser más lentas, pero no tan lentas como en los sistemas de RAID 4, ya que el volumen maneja múltiples solicitudes de escritura simultánea. Varios fabricantes han mejorado el rendimiento de escritura mediante el uso de técnicas especiales, como el uso de memoria no volátil para el registro de las escrituras.

  RAID 5 proporciona prácticamente todos los beneficios del seccionamiento (alta velocidad de lectura), mientras que proporciona la redundancia necesarias para la fiabilidad, que RAID 0 no ofrece.

También existen niveles más complejos de RAID que incluyen volúmenes lógicos de diferentes niveles de RAID. Los niveles múltiples de RAID permiten las siguientes configuraciones

- RAID 0 + 1
  Estos sistemas RAID ofrecen las ventajas de los discos seccionados y los espejos. Tienden a alcanzar un alto grado de rendimiento debido a la creación de secciones, mientras que ofrece una alta fiabilidad debido al hecho de que todos los discos se espejean. La única restricción es que se debe solicitar el doble de la cantidad de discos que realmente se necesitarán para los datos, porque va a reflejar todos los discos.
- RAID 3 + 0
  Esta es una implementación de RAID multinivel compuesto de varios volúmenes lógicos con seccionamiento de datos (RAID 0). Un volumen lógico con varios miembros lógicos en RAID 3 se considera un RAID (3+0) o RAID 53.

- RAID 5 + 0

  Este es un volumen lógico compuesto de varios volúmenes lógicos en RAID 5.

- RAID 5 + 1

Para este volumen multinivel se requiere de varias controladoras RAID. En un arreglo del tipo RAID 5+1 existe una primera capa de nivel 1 formada de una controladora de volúmenes en RAID 5 y en la capa de nivel 2 encontramos varios volúmenes virtuales un RAID 1 que mantienen un espejo de los datos[46].

- RAID 5 + 5

  Para este volumen multinivel se requiere de varias controladas RAID. En un arreglo del tipo RAID 5+5 existe una primera capa de nivel 1 formada de varios volúmenes lógicos en RAID 5 y en la capa de nivel 2 encontramos varios volúmenes virtuales un RAID 5.

## IX. Nuevas tecnologías de almacenamiento

Las tecnologías de almacenamiento actuales son muy superiores a las tecnologías de hace cinco años. Las unidades de disco se han vuelto más rápidas, no es difícil encontrar discos de 10,000 RPM y 15,000 RPM de velocidad en la actualidad. Estos discos tienen velocidades de búsqueda de 3.5 milisegundos.

Además, Interfaces SCSI avanzadas y el creciente uso de las interfaces de canal de fibra entre los servidores y dispositivos de almacenamiento han aumentado las tasas de transferencia de datos hasta a 100 MB por segundo o inclusive más rápido. La capacidad de los discos individuales también se ha incrementado considerablemente, con discos de 180 GB que son bastante comunes hoy en día. El tiempo promedio antes de un fallo de estos discos de nueva generación es también muy alto, a veces de más de un millón de horas.

Las nuevas arquitecturas tecnológicas para el almacenamiento de datos toman ventaja de todos los factores anteriores para proporcionar un excelente soporte de almacenamiento para los RDBMS. Existen dos arquitecturas de almacenamiento bastante rentable como son las redes de almacenamiento (SAN) y los sistemas de almacenamiento conectado a red (NAS). A continuación se describirán estas arquitecturas de almacenamiento.

### Storage Area Networks

Hoy en día, las grandes bases de datos están en todas partes, bases de datos con terabytes ya no son una rareza. Las organizaciones tienden no solamente a tener varias grandes bases de datos OLTP de trabajo, sino que también utilizan grandes almacenes de datos y data marts para apoyar la toma de decisiones gerenciales. Storage Area Networks (SANs) hacen uso de conexiones de alto desempeño y técnicas de almacenamiento en RAID para alcanzar el desempeño y la fiabilidad que demanda la información de las organizaciones hoy en día.

Los centros de datos modernos utilizan redes SAN para optimizar el rendimiento y la fiabilidad. Las SAN pueden ser muy pequeñas o muy grandes, y se adaptan fácilmente a las últimas tecnologías

[46] Oracle. Sun StorEdge™ 3000 Family RAID Firmware 4.2x User's Guide. http://docs.oracle.com/cd/E19236-01/817-3711-18/817-3711-18.pdf. [consulta: Noviembre de 2011]

de almacenamiento en disco y de redes de comunicaciones. Tradicionalmente, los dispositivos de almacenamiento se conectaban a las computadoras a través de un dispositivo SCSI. SAN puede conectarse a los servidores a través de tecnología de alta velocidad de canal de fibra con la ayuda de switches y hubs. Se puede adaptar dispositivos anteriores compatibles basados en SCSI para su uso con una SAN, o pueden utilizarse dispositivos totalmente nuevos especialmente diseñados para la SAN. Una SAN es posible gracias al uso de switches de canal de fibra llamados switches Brocade. Mediante el uso de concentradores se pueden utilizar redes SAN que están a varios kilómetros de distancia de los servidores host.

### Networked Attached Storage

En pocas palabras, el Networked Attached Storage (NAS) es una caja negra conectada a la red, y provee almacenamiento adicional. El tamaño de la NAS puede estar en un rango de un espacio tan pequeño como 2GB y hasta un tamaño en terabytes de capacidad de almacenamiento.

La principal diferencia entre una NAS y una SAN es que por lo general es más fácil ampliar el sistema de almacenamiento SAN utilizando el software proporcionado por el proveedor. Por ejemplo, se pueden combinar varios discos en un sólo volumen en una SAN. Un NAS está configurado con su propia dirección, pasando así los dispositivos de almacenamiento fuera de los servidores. El NAS se comunica y transfiere datos a los servidores cliente a través de protocolos como el sistema de archivos de red (NFS).

La arquitectura NAS realmente no es muy adecuada para grandes bases de datos OLTP. Uno de los enfoques actualmente recomendados por muchos proveedores de almacenamiento en general es tener la combinación de las tecnologías SAN y NAS para tener lo mejor de ambos mundos.

### InfiniBand

Una de las últimas tecnologías es la red InfiniBand, una alternativa basada en estándares para Ethernet, que busca superar las limitaciones de las redes basadas en TCP/IP. Uno de los impulsores de los cambios en tecnología de almacenamiento en red es reducir los cuellos de botella de IO entre la CPU y los discos. InfiniBand toma otro enfoque y trabaja entre el controlador de canal del servidor y un adaptador especial en las máquinas o dispositivos de almacenamiento, por lo que no se requiere un bus de IO. Un simple enlace puede funcionar a 2.5 GB por segundo. InfiniBand ofrece un mayor rendimiento y menor latencia y uso de CPU comparado con las soluciones normales de TCP/IP y Ethernet

Teniendo en cuenta las empresas de alto perfil que participan en el desarrollo de este concepto (Microsoft, IBM, Sun, HP, y algunos de los principales proveedores de almacenamiento), será posible ver un impulso considerable en el área de almacenamiento. InfiniBand soporta su propio protocolo, llamado Sockets Direct Protocol (SDP).

### SSD

La industria del almacenamiento ha hecho esfuerzos importantes en los últimos años para evitar el cuello de botella, cada vez más oneroso, que las unidades de disco magnético representan.

La solución más frecuente, y hasta hace poco la más práctica, ha sido la implementación de discos de "carrera corta": básicamente la instalación de más discos de los que son necesarios para el almacenamiento de datos con el fin de aumentar la capacidad de IO. Esto aumenta la capacidad general de IO del subsistema de disco pero tiene un efecto limitado en la latencia de las distintas operaciones de IO que se pueden registrar.

Las tecnologías de disco de estado sólido tienen latencias muy por debajo de las registradas por los discos magnéticos ya que están construidos únicamente por componentes electrónicos y la tendencia es la aceleración en el rendimiento y un aumento en la capacidad, según lo predicho por la ley de Moore.

Las dos categorías principales de la tecnología de disco de estado sólido son:

- DDR RAM basada en SSD, en el que la memoria utilizada como medio de almacenamiento no es muy diferente de la memoria RAM del ordenador.
  Un condensador o batería se ha incorporado para que los datos se puedan escribir en un medio de almacenamiento no volátil en caso de una falla de energía.
- Memoria basada en Flash SSD, en el que los datos se almacenan en la memoria flash NAND similar a la utilizada en las unidades USB y dispositivos de consumo. Esta memoria no es volátil, por lo que no se pierdan datos en caso de una falla de energía.
  El rendimiento de los SSD es de magnitud superior a los dispositivos de disco magnético, en especial para operaciones de lectura.

Los discos de carrera corta pueden mejorar la latencia de búsqueda de IO mediante la reducción de la distancia media que el brazo de lectura necesita moverse a través del disco y la concentración de datos en los sectores exteriores de los platos donde la velocidad de rotación general es más alta.

Paul Randal expone en su blog[47] algunas mediciones del desempeño de diferentes tecnologías de almacenamiento incluyendo a los discos SSD. De esta investigación se destaca la velocidad de almacenamiento en SSD respecto a volúmenes SCSI.

## X. Historia de la adopción del Optimizador Basado en Costos en Oracle

En Oracle 8i, el optimizador simplemente contaba el número de solicitudes que él esperaba hacer al subsistema de IO. El plan de ejecución que requiriera el número menor de solicitudes era el que se escogía de entre todos los planes generados. Esta versión no tomaba en cuenta las siguientes consideraciones:

[47] Randal, Paul. *Benchmarking: Introducing SSDs (Part 3: random inserts with wait stats details)*. http://www.sqlskills.com/BLOGS/PAUL/post/Benchmarking-Introducing-SSDs-(Part-3-random-inserts).aspx. [consulta: Noviembre del 2011]

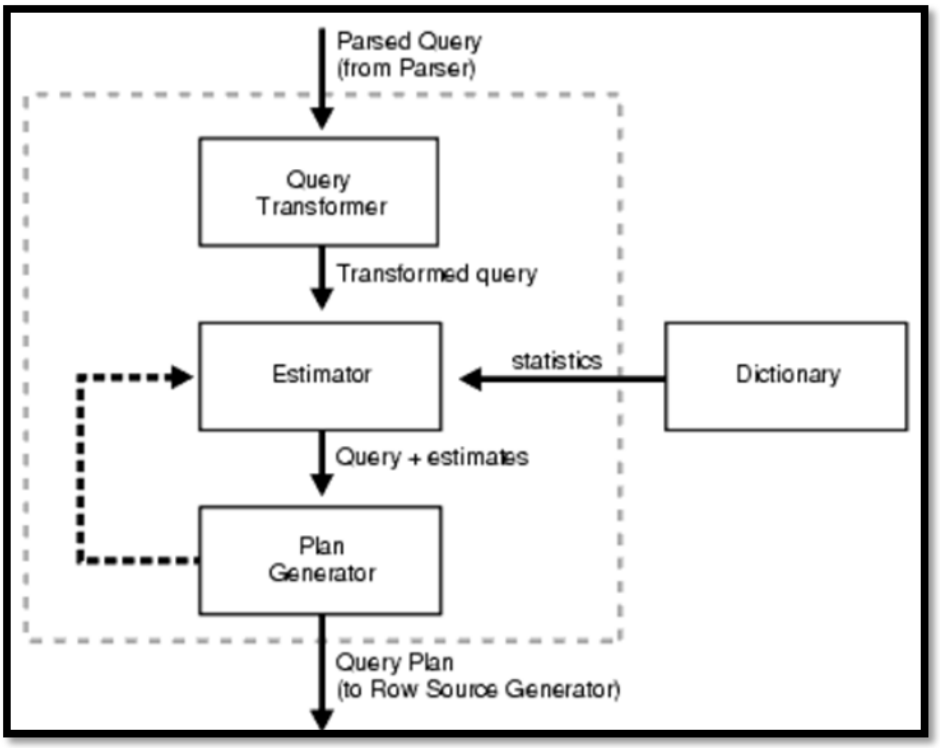
- Un escaneo completo de tabla en forma de heap podría requerir mucho más CPU que una ruta que utilizara un índice
- No tomaba en cuenta que la lectura de 128 bloques podría tomar más tiempo que la lectura de un único bloque
- No tomaba en cuenta que la lectura de 128 bloques podría hacer en varias lecturas separadas debido a la organización de la información en el subsistema de IO y en el subsistema de memoria
- No tomaba en cuenta que la solicitud podría satisfacerse desde la lectura del buffer y / o caché sin requerir acceso al subsistema de IO

En Oracle 9i el optimizador introdujo una característica referente al costeo de CPU. Se podía almacenar información de los tiempos de respuesta de solicitudes al subsistema de IO por un único bloque o múltiples bloques así como la información del tamaño típico de un bloque requerido. El optimizador tomaba este factor y lo agregaba a la ecuación para determinar mejor el costo.

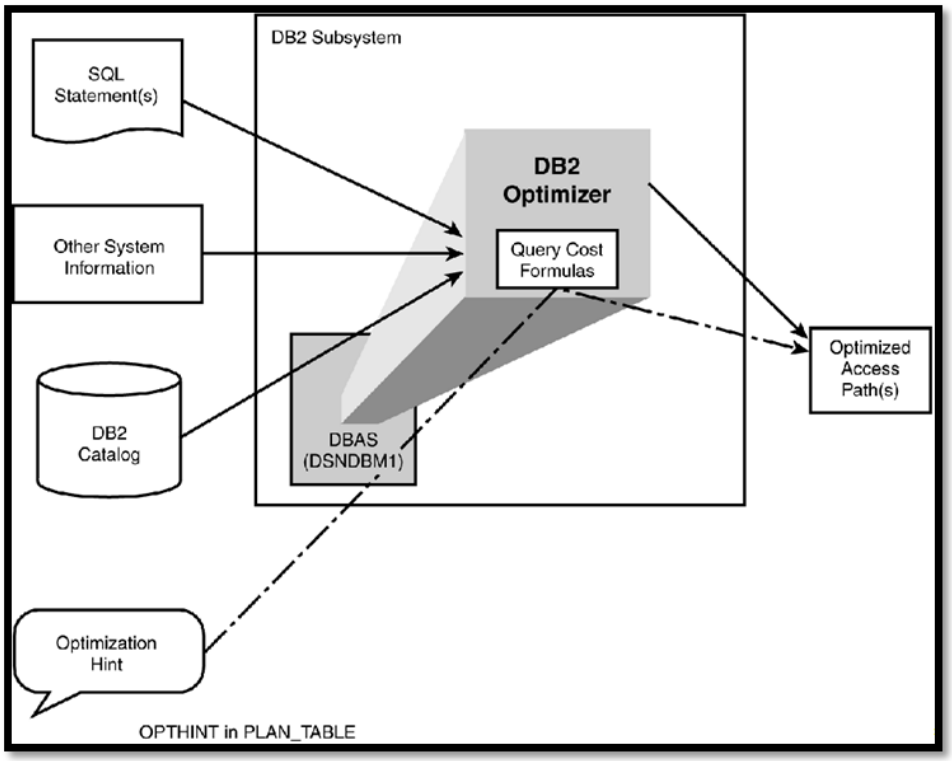
Esos refinamientos asegurarían que el optimizador tuviera un mejor estimado del costo de escaneo completo de tablas y produciría planes de ejecución más sensibles con menores anomalías.

En Oracle 10g apareció el optimizador offline. Esta característica permitía generar y almacenar información estadística crítica. Lo que ayudaba al optimizador a lidiar en línea con problemas relacionados a la distribución de datos.

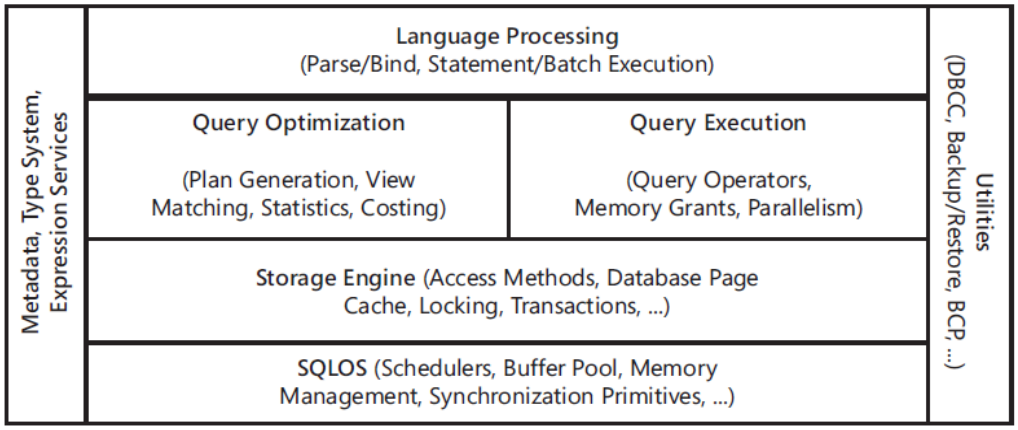
A continuación se ilustra la implementación del optimizer y algunos otros componentes de los manejadores que permitirá ver mejor como funciona por dentro cada implementación.



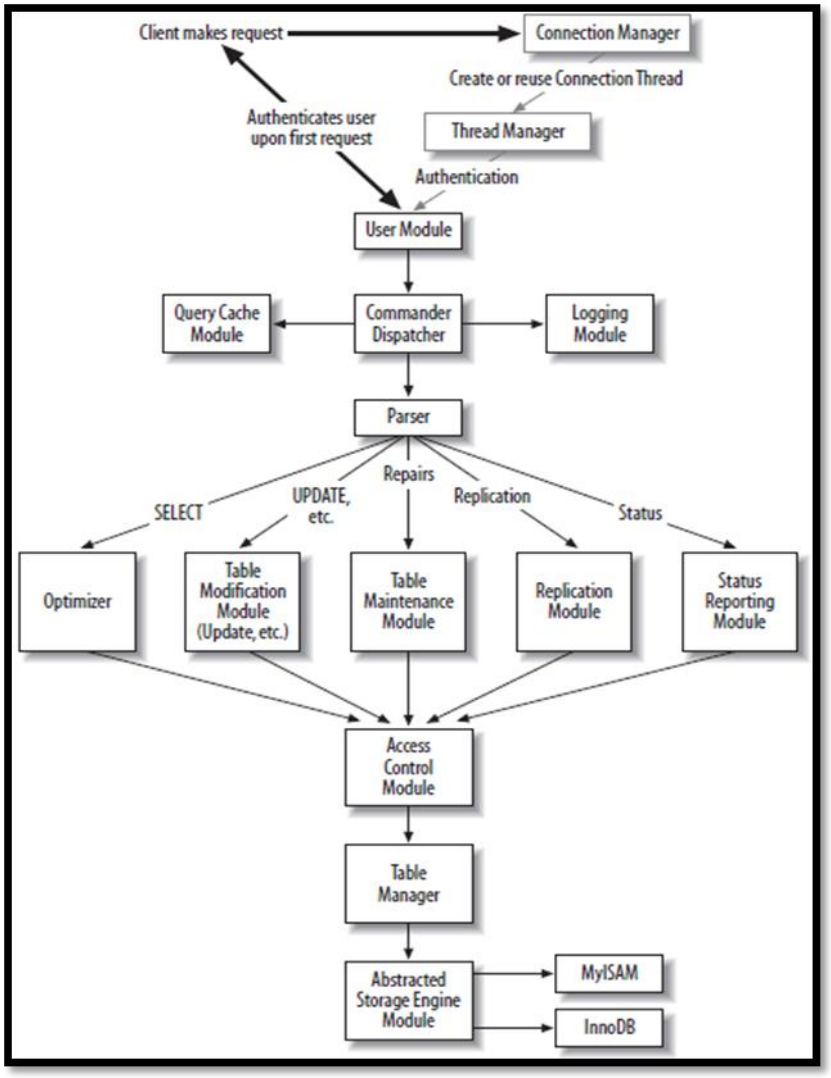

Implementación del optimizer de Oracle




Implementación del optimizer en DB2




Implementación del optimizer en SQL Server




Implementación del optimizer en MySQL

# Bibliografía


- Alapati, Sam R. (2005). *Expert Oracle Database 10g Administration*. United States of America: Apress. 1305 páginas.
- Ben-Gan, Itzik (2009). *Inside Microsoft SQL Server 2008: T-SQL Querying*. United States of America: Microsoft Press. 824 páginas.
- Ben-Gan, I., Sarka, D., Katibah, E., Low, G., Wolter, R., Kunen, I. (2010). *Inside Microsoft SQL Server 2008: T-SQL Programming*. United States of America: Microsoft Press. 832 páginas.
- Bernstein, Philip A., Hadzilacos, V., Goodman, N., (1987). *Concurrency control and recovery in database systems*. United States of America: Addison-Wesley. 370 páginas
- Delaney, Kalen (2005). *Inside Microsoft SQL Server 2005: Query Tuning and Optimization*. United States of America: Microsoft Press. 448 páginas.
- Delaney, Kalen (2009). *Microsoft SQL Server 2008 Internals*. United States of America: Microsoft Press. 783 páginas.
- Date, C. J. (2004). *An introduction to database systems (Eight Edition)*. United States of America: Adisson-Wesley. 1034 páginas
- DuBois, P., Hinz, S., Pedersen, C. (2005). *MySQL 5 Certification Study Guide*. United States of America: Sams Publishing. 672 páginas.
- Hellerstein, Joseph M., Stonebraker, M., (2005). *Readings in Database Systems (4th Edition)*. United States of America: MIT Press. 877 páginas.
- Joyanes Aguilar, L., Zahonero Martínez, I., (2005). *Programación en C, metodología, algoritmos y estructuras de datos*. España: Mc Graw Hill. 558 páginas.
- Lewis, Jonathan (2006). *Cost-Based Oracle Fundamentals*. United States of America: Apress. 537 páginas.
- Loudon, Kyle (1999). *Mastering Algorithms with C*. United States of America: O'Reilly Media Inc. 560 páginas.
- Medina Muñoz, Ismael. *Preciso SQL Server Activity Monitor*. Obra protegida por el Registro Público del Derecho de Autor, No. de Registro 03-2011-090513131800-01. México. Septiembre del 2011.
- Medina Muñoz, Ismael. *Vago.SqlServer.Assessment*. Obra protegida por el Registro Público del Derecho de Autor, No. de Registro 03-2011-090513131800-01. México. Marzo del 2012.

- Mullins, Craig S. (2004). *DB2 Developer's Guide (Fourth ed.)*. United States of America: Sams Publishing. 1512 páginas.
- O'Neil, P., O'Neil, E. (2000). *Database: Principles, Programming, and Performance (Second Edition)*. United States of America: Morgan Kaufmann. 870 páginas.
- Pachev, Sasha (2007). *Understanding MySQL Internals*. United States of America: O'Reilly Media, Inc. 256 páginas.
- Sanders, Roger E. (2007). *DB2 9 Fundamentals Certification Study Guide, First Edition (Exam 730)*. United States of America: MC Press. 658 páginas.
- Schwartz, B., Zaitsev, P., Tkachenko, V., Zawodny, Jeremy D., Lentz, A., Balling, Derek J. (2008). *High Performance MySQL (Second Edition)*. United States of America: O'Reilly Media Inc. 710 páginas.
- Watson, John (2008). *OCA Oracle Database 11g: Administration I Exam Guide (Exam 1Z0-052)*. United States of America: Oracle Press. 768 páginas.
- Cain, M., Milligan, K., *IBM DB2 for i - indexing methods and strategies*. IBM. https://www-304.ibm.com/partnerworld/wps/servlet/download/DownloadServlet?id=G_mhBFTHFjyiPCA$cnt&attachmentName=IBM+DB2+for+i+indexing+methods+and+strategies.pdf&token=MTMzNzc5NTQzMzkzNw==&locale=en_ALL_ZZ . [consulta: Diciembre 2011]
- Harrison, Guy. *Best Practices for Optimizing Oracle RDBMS with Solid State Disk*. Quest Software Inc. http://questsoftware.com.mx/Quest_Site_Assets/WhitePapers/Best_Practices_for_Optimizing_Oracle_RDBMS_with_Solid_State_Disk-final.pdf. [consulta: Noviembre de 2011]
- IBM. *DB2-Oracle terminology mapping*. http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=%2Fcom.ibm.db2.luw.apdv.porting.doc%2Fdoc%2Fc0053114.html. [consulta: Noviembre de 2011]
- IDC. *Worldwide RDBMS 2009 Vendor Analysis: Top 10 Vendor License Revenue by Operating Environment*. http://idcdocserv.com/224350E. [consulta: Noviembre de 2011]
- Microsoft. *blocked process threshold Server Configuration Option*. Microsoft SQL Server Books Online, http://msdn.microsoft.com/es-es/library/ms181150.aspx. [consulta: Diciembre 2011]
- Microsoft. *Displaying Execution Plans by Using the Showplan SET Options*. Microsoft SQL Server Books Online. http://msdn.microsoft.com/en-us/library/ms180765(SQL.105).aspx. [consulta: Diciembre 2011]

- Microsoft. *SET SHOWPLAN_XML (Transact-SQL)*. Microsoft SQL Server Books Online. http://msdn.microsoft.com/en-us/library/ms187757.aspx. [consulta: Diciembre 2011]
- Microsoft. *SET STATISTICS XML (Transact-SQL).* Microsoft SQL Server Books Online. http://msdn.microsoft.com/en-us/library/ms176107.aspx. [consulta: Diciembre 2011]
- Navarrete Sánchez, I., Cárdenas Viedma, M., Sánchez Alvarez, D., Botía Blaya, J., Marín Morales, R., Martínez Béjar, R., *TEORÍA DE AUTÓMATAS Y LENGUAJES FORMALES*. Departamento de Ingeniería de la Información y las Comunicaciones, Universidad de Murcia. http://www.yakiboo.net/apuntes/TALF%20-%20YakiBoo.net.pdf. [consulta: Diciembre 2011]
- Oracle. *MySQL. 12.2.1.1.Â Fil Header*. http://mysql.netvisao.pt/doc/internals/en/innodb-fil-header.html. [consulta: Noviembre de 2011]
- Oracle. *MySQL. 14.2.4.13. InnoDB Disk I/O and File Space Management*. http://dev.mysql.com/doc/refman/5.6/en/innodb-disk-management.html. [consulta: Noviembre de 2011]
- Oracle. *MySQL. 14.2.10.1. Clustered and Secondary Indexes*. http://dev.mysql.com/doc/refman/5.0/en/innodb-index-types.html. [consulta: Noviembre de 2011]
- Oracle. *Oracle Database Concepts 11g Release 1 (11.1). 5 Schema Objects*. http://docs.oracle.com/cd/B28359_01/server.111/b28318/logical.htm. [consulta: Noviembre de 2011]
- Oracle. *Oracle Database Concepts, 11g Release 1 (11.1). 2 Data Blocks, Extents, and Segments*. http://docs.oracle.com/cd/B28359_01/server.111/b28318/schema.htm. [consulta: Noviembre 2011]
- Oracle. Sun StorEdge™ 3000 Family RAID Firmware 4.2x User's Guide. http://docs.oracle.com/cd/E19236-01/817-3711-18/817-3711-18.pdf. [consulta: Noviembre de 2011]
- Oracle. *The Query Optimizer*. Oracle Database Performance Tuning Guide 11g Release 1 (11.1). http://docs.oracle.com/cd/B28359_01/server.111/b28274/optimops.htm#autoId0. [consulta: Diciembre 2011]
- Oracle. *The Query Optimizer*. Oracle Database Performance Tuning Guide 11g Release 1 (11.1). http://docs.oracle.com/cd/B28359_01/server.111/b28274/optimops.htm#i82107. [consulta: Diciembre 2011]

- Pineda, Luis A. *Tema 18, Autómata de pila (Pushdown automata)*. IIMAS UNAM. http://leibniz.iimas.unam.mx/~luis/cursos/ALF_old/sesiones/tema18.pdf. [consulta: Noviembre de 2011]

- Prabhakaran, M., Viswanathan, M., *CS 373: Theory of Computation*. University of Illinois, Urbana-Champaign. http://www.cs.uiuc.edu/class/fa08/cs373/Lectures/lect13.pdf. [consulta: Noviembre de 2011]

- Prabhakaran, M., Viswanathan, M., *CS 373: Theory of Computation*. University of Illinois, Urbana-Champaign. http://www.cs.uiuc.edu/class/fa08/cs373/Lectures/notes13.pdf. [consultado en: Noviembre de 2011]

- Pardo, Luis M., Gómez, D., *Teoría de Autómatas y Lenguajes Formales (para Informáticos)*. http://personales.unican.es/pardol/Docencia/TALF/TALF07.pdf . [consulta: Diciembre 2011]

- Randal, Paul. *Benchmarking: Introducing SSDs (Part 3: random inserts with wait stats details)*. http://www.sqlskills.com/BLOGS/PAUL/post/Benchmarking-Introducing-SSDs-(Part-3-random-inserts).aspx. [consulta: Noviembre del 2011]

- Tuuri, H., Sun, C., *InnoDB Internals: InnoDB File Formats and Source Code Structure*. MySQL Conference, April 2009. http://www.innodb.com/wp/wp-content/uploads/2009/05/innodb-file-formats-and-source-code-structure.pdf. [consulta: Noviembre de 2011]

- Wikipedia. *Árbol-B*. http://es.wikipedia.org/wiki/%C3%81rbol-B. [consulta: Noviembre del 2011]

- Yuhanna, N., Gilpin, M., D'Silva, D., *The Forrester Wave: Enterprise Database Management Systems, Q2 2009*. http://www.microsoft.com/presspass/itanalyst/docs/06-30-09EnterpriseDatabaseManagementSystems.PDF. [consulta: Noviembre de 2011]

# Glosario

| | |
|---|---|
| **RDBMS** | Sistema manejador de bases de datos relacionales o sistema de gestión de bases de datos relacionales, es software capaz de manejar las relaciones definidas según el modelo relacional. |
| **IO** | Input / Output, término utilizado para hablar del subsistema de almacenamiento en disco. |
| **DBA** | Database Administrator, Administrador de base de datos en español. |
| **Buffer** | Espacio de memoria utilizado para el almacenamiento general de datos. |
| **Cache** | Espacio de memoria utilizado para duplicar datos almacenados en otros subsistemas. |
| **Índice CLUSTERED** | Estructura B+-tree que ordena físicamente los registros de una tabla. |
| **Índice NON CLUSTERED** | Estructura B+-tree que no ordena físicamente los registros de una tabla, en su lugar se mantienen apuntadores a la ubicación de los registros. |
| **Función Hash** | Función de conversión de una cadena de bytes entrantes en una representación numérica. |
| **Query** | Predicado del lenguaje de consultas SQL utilizado para operar relaciones y sus atributos bajo el álgebra de relaciones y el cálculo relacional. |