

# Estrategias para la Optimización de Consultas en Bases de Datos Relacionales<sup>1</sup>

Rodolfo A. Pazos Rangel<sup>°</sup> y José A. Martínez Flores<sup>\*°</sup>

<sup>°</sup> Centro Nacional de Investigación y Desarrollo Tecnológico

<sup>\*</sup> Instituto Tecnológico de Cd. Madero

E-mail: {pazos, jam}@sd-cenidet.com.mx

## Resumen

Hoy en día es normal que las consultas realizadas por grandes empresas involucren una gran cantidad de información, por tal motivo el tiempo de respuesta por parte del sistema manejador de bases de datos (SMBD) a tales consultas puede ser intolerable para algunas aplicaciones. Es común que los SMBDs corran sobre un sistema operativo (SO), renunciando con esto al control del acceso a los datos, tarea del sistema de archivos del SO, el cual está desafortunadamente diseñado para propósitos generales, por lo que sus algoritmos son para tales fines. En el área de los SOs, aun cuando se ha trabajado extensamente, las técnicas usadas ignoran información muy valiosa que está disponible, lo cual le impide un desempeño óptimo en la administración de sus páginas (datos) en relación a la información que se obtendría del planificador de consultas de un SMBD. Por lo antes descrito ciertos SMBDs construyen su propio administrador de archivos relegando al del SO. En este artículo se presentan algunas estrategias para optimizar el acceso a los datos para aplicaciones de bases de datos.

Palabras clave: bases de datos relacionales, optimización de estrategias de acceso.

## 1. Introducción

Actualmente es común que las grandes empresas tengan enormes bases de datos (BDs), por tal motivo es normal que sus consultas involucren una gran cantidad de información. Debido a lo anterior es de suponer que el tiempo de respuesta del SMBD puede llegar a ser abrumador [1, 2].

De todos es conocido que los datos sólo pueden ser manipulados en la memoria principal, por consiguiente la información de la base de datos debe ser cargada en ésta antes de ser manipulada. Comúnmente el acceso a los datos es de forma reactiva, de tal manera que la carga de éstos (de disco a memoria principal) se inicia en respuesta a una solicitud [3, 4]. Se han venido desarrollando nuevos métodos (principalmente en el área de los sistemas operativos) para acceder a la información de una manera más eficaz.

Es ampliamente conocido por la comunidad de bases de datos que la administración del contenedor (buffer) juega un rol clave en proveer acceso eficiente a datos residentes en disco y en el uso óptimo de la memoria principal [5, 6]. Debido a que en el área de BDs es posible conocer en ciertas consultas el patrón de acceso a los datos, por lo tanto es posible a) solicitar la información anticipadamente al controlador del disco y b) colocarla en un contenedor para su uso posterior (ver Figura 1), para lo cual se debe determinar dónde y en dado caso, qué dato remplazar (desalojar un dato que *de momento* no sea útil para colocar otro). **En estas circunstancias hay que sincronizar cuidadosamente la precarga (solicitud de datos) y su almacenamiento en memoria principal, lo cual constituye un problema complejo [7].**

En este trabajo de investigación se propone una nueva estrategia de solución al problema de recuperación de información, en este caso para el área de bases de datos relacionales. En el ámbito de las bases de datos no se ha trabajado con técnicas combinadas de precarga y políticas de remplazo de la información de acuerdo a nuestra investigación de la literatura especializada.

---

<sup>1</sup> Esta investigación fue financiada en parte por el convenio COSNET, COTACYT y RITOS2.

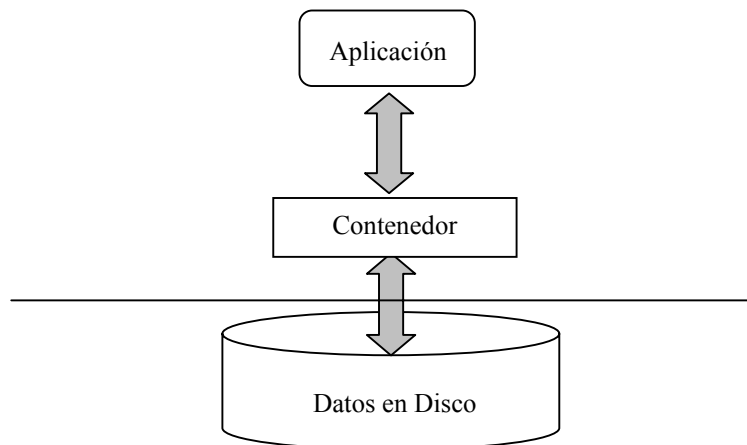


Figura 1. Situación del contenedor en una arquitectura típica.

## 2. Marco teórico

La memoria principal disponible para un programa más sus datos puede ser insuficiente, en este caso se deberá emplear una práctica conocida como *intercambio* (swapping). En ésta el programa y sus datos son organizados de tal manera que varios módulos pueden ser colocados en la misma región de la memoria. En casi todos los sistemas de multiprogramación modernos, esto involucra un sofisticado mecanismo conocido como *memoria virtual* [8].

En un ambiente de sistema operativo (SO) actual, que utilice una estrategia de memoria virtual, ejecutar un SMBD que sea tratado como un programa de aplicación normal, puede tener resultados desfavorables para el administrador del contenedor (espacio para almacenamiento temporal de datos) del SMBD. Esto se debe a que tanto su código como su contenedor serán paginados por el sistema de archivos (módulos encargados de la administración de los archivos) del SO, el cual está diseñado para propósitos generales por lo que sus algoritmos son para tales fines [9, 10].

Actualmente los sistemas de archivos tradicionales esperan hasta que una aplicación requiera datos para solicitarlos al subsistema de entrada y salida (la Figura 2 muestra la organización jerárquica de la memoria [8]). Si éstos no están presentes en algún contenedor, será necesario recuperarlos (leer los datos) del almacenamiento secundario, con su correspondiente pérdida de tiempo, además de realizar alguna política de remplazo (seleccionar un dato víctima cuando el contenedor esté lleno para colocar el que se necesita) si fuera necesario [8, 11].

Si el optimizador de consultas y el administrador del contenedor trabajaran en conjunto y no de manera independiente, se podría transmitir información relevante ya que en ciertas consultas el SMBD puede conocer el patrón de acceso (cuáles y cuándo serán leídos los datos) para satisfacer la consulta; por lo tanto, el SMBD puede precargar los datos y, cuando éstos sean requeridos ya estarán en el contenedor con lo que se evitará leerlos de disco, lo cual resulta mucho más rápido [12, 13].

Hay dos tipos de precarga: la informada y la predictiva. En la precarga informada la aplicación divulga los datos que requerirá; es decir, indica los datos a usar, de tal forma que se sabe con certeza cuáles serán. La implementación más común de precarga predictiva es la lectura adelantada secuencial; otra opción es que el sistema haga predicciones sobre los accesos futuros (basadas generalmente en accesos pasados) para precargar los datos en consecuencia. Algunos investigadores han llegado al extremo de precargar archivos enteros antes de ser referidos [13].

Debido a que un acceso a una página de la BD en disco es mucho más costoso que un acceso a una página de la BD en el contenedor, la meta principal de un administrador de contenedor de una BD es minimizar

las entradas y salidas (E/S) de las páginas. Por tal motivo la optimización del algoritmo de remplazo de páginas es muy importante para el desempeño total del SMBD [14].

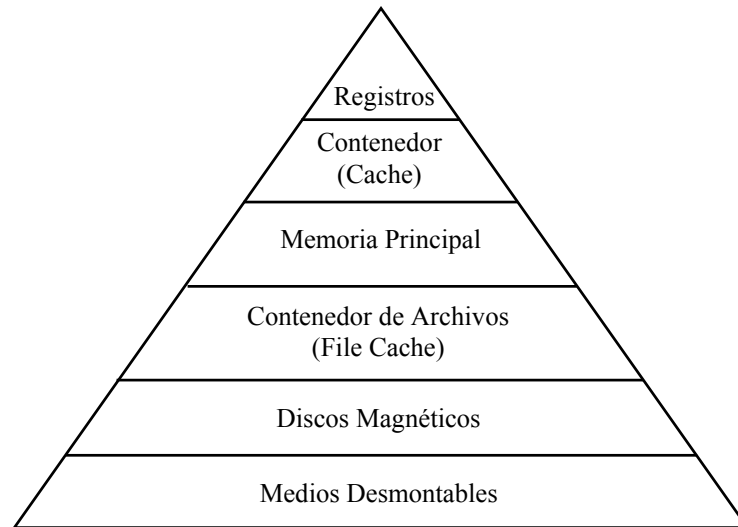


Figura 2. Jerarquía de la memoria.

Tanto en los SOs como en los SMBDs se necesita elegir a la página víctima cuando el contenedor se encuentra lleno. La respuesta a qué página quitar es sencilla: elegir a la página que ya no se va a usar. El algoritmo para implementar tal regla es muy complicado (óptimo), porque requiere que el SO tenga perfecto conocimiento de eventos futuros [8, 15], para lo cual habría que ejecutar el programa completo. En el área de los SOs se han inventado e implementado técnicas variantes a la regla óptima, es decir, reemplazan la página que tenga menor probabilidad de usarse.

La precarga y el contenedor son dos métodos muy conocidos para mejorar el desempeño del sistema de archivos [16]. A pesar de que éstos han sido estudiados extensivamente, se han realizado varios estudios de precarga con la ausencia del contenedor o para estrategias de contenedor fijo. Según [17] la interacción entre la precarga y el contenedor no está bien entendida.

### 3. Estrategias de acceso

La investigación que se está realizando tiene como objetivo demostrar que la integración de la precarga y del contenedor en el área de BDs permite mejorar el desempeño de éstas, tal como ya se ha demostrado en el área de los sistemas operativos [7, 18]. Este proyecto consiste de tres partes básicas: 1) el desarrollo de estrategias de precarga para BDs, 2) la evaluación de la(s) política(s) de remplazo óptima(s), y 3) la integración e implementación de ambas.

Las reglas que hay que tomar en cuenta para una precarga y estrategia de remplazo óptimas son las siguientes:

- ◆ Precarga óptima: cada precarga deberá traer al contenedor el siguiente bloque en el flujo de referencia que no esté en el contenedor.
- ◆ Remplazo óptimo: cada precarga deberá descartar el bloque cuya siguiente referencia sea la menos necesaria en el futuro.

Para que la precarga y la política del contenedor estén correctamente integradas hay que tomar las decisiones siguientes:

- ◆ ¿cuándo cargar un bloque de disco?,
- ◆ ¿qué bloque cargar?, y
- ◆ ¿qué bloque remplazar cuando se realiza la carga?,

para que el tiempo total a transcurrir sea mínimo.

Hay que hacer notar que lo anterior corresponde a un sistema con un disco o un servidor de archivos, y éste sólo considera referencias de lectura.

## 4. Análisis de la instrucción SELECT

### 4.1. Un ejemplo que evidencia la utilidad de precargar datos

Para ejemplificar que es factible predecir el uso de renglones en un SMBD, se describirá a continuación el caso más sencillo. Primeramente cabe hacer mención que el SMBD, cuando recibe una instrucción en SQL, la analiza léxica, sintáctica y semánticamente [8, 19] de tal manera que puede predecir el patrón de acceso a los datos.

Por ejemplo, al solicitársele un determinado renglón de cierta tabla (que no tenga índice) **el SMBD sabe que, para proporcionar el resultado a tal consulta, deberá leer todos los renglones de la tabla**, para lo cual se deberá evaluar el tamaño de ésta para determinar si es factible precargar toda la tabla o sólo algunas páginas de datos (renglones). Como se muestra en la Figura 3, el SMBD, una vez que haya leído el renglón  $i$ , sabe que el renglón  $i - 1$  (o alguno anterior a éste) ya no le servirá, de tal forma que éste puede ser el dato víctima para colocar el renglón  $i + 1$  si fuera necesario.

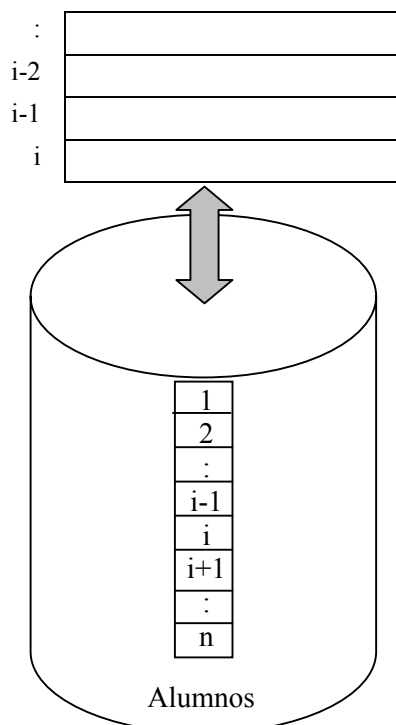


Figura 3. Interacción entre precarga y contenedor.

## 4.2. Taxonomía de la instrucción SELECT

Uno de los objetivos principales de esta investigación es identificar de acuerdo a la instrucción SELECT de SQL qué datos precargar, a partir de la información del planificador de consultas, y si es posible cargar toda la tabla. A continuación se describen los diferentes tipos de casos que pueden presentarse con la instrucción SELECT de SQL.

1. **Consulta a sólo una tabla, sin cláusula WHERE sin índice y con índice.** Al tener que mostrar la información de todos los renglones de la tabla, lo más conveniente es utilizar la precarga, para cargar en memoria principal los renglones (en páginas) que se usarán en un futuro próximo, con el consiguiente ahorro en el tiempo de acceso a los datos, debido a que con esto se adelantará a la solicitud de los datos de tal manera que ya estarán en memoria principal cuando éstos sean requeridos. Para este caso es conveniente usar el algoritmo de remplazo APU (Any Previously Used), debido a que los renglones ya leídos no se volverán a utilizar y por lo tanto es indistinto qué renglón se reemplazará.
2. **Consulta a sólo una tabla, con cláusula WHERE sencilla sin índice.** Al no tener índice la columna que se utiliza en la cláusula WHERE, es necesario evaluar cada uno de los renglones de la tabla para saber si cumplen con la condición de búsqueda, por lo tanto se tratará igual que el caso número 1.
3. **Consulta a sólo una tabla, con cláusula WHERE sencilla con índice.** Primeramente evaluar si es conveniente usar el índice, si es así cargar los nodos del árbol (del índice) de acuerdo a como se vayan necesitando; en este caso no se pueden aprovechar las bondades de la precarga, debido a que para poder precargar algún nodo del árbol se necesitará conocer su dirección, en todo caso, es mejor saber qué nodo conduce a la hoja (dato) que se busca. Lo que se podría precargar sería el nodo raíz, ya que este nodo es indispensable para iniciar la búsqueda. Una vez encontrado el nodo hoja, se podrá acceder al renglón de la tabla (página). Se continuarán leyendo los nodos hoja adyacentes (a la izquierda o derecha, de acuerdo al análisis previo de la instrucción) siempre y cuando lo indique la cláusula WHERE (por ejemplo, columna > valor), y/o la información proporcionada por la tabla, mientras se cumpla la condición, lo cual puede arrojar como resultado uno, varios o ningún renglón; es conveniente usar el algoritmo de remplazo LRU (Least Recently Used) para los nodos del árbol, permaneciendo con esto los primeros nodos accedidos (los más próximos a la raíz) que son los que tienen mayor probabilidad de volverse a utilizar. Se podrá utilizar el algoritmo de remplazo APU para la administración de las páginas que contienen los renglones de la tabla, ya que una vez leído un determinado renglón pudiera ser que exista otro en la misma página a ser utilizado posteriormente; de tal manera que al no saber con certeza absoluta qué páginas contienen renglones a ser leídos en un futuro cercano, la página a reemplazar será hasta cierto punto indistinta.
4. **Consulta a sólo una tabla, con cláusula WHERE compleja, sin índice.** Al no contar con índice se deberán evaluar todos los renglones de la tabla, para saber cuáles cumplen la condición de búsqueda, por lo tanto se trata igual que el caso número 1.
5. **Consulta a sólo una tabla, con cláusula WHERE compleja, con índice.** Se deberá descomponer en subconsultas y si alguna de éstas tiene índice utilizarlo, siempre y cuando sea de utilidad; si hay más de un índice, hay que realizar un análisis para seleccionar el que resulte más beneficioso. En caso de existir un índice y sea conveniente utilizarlo, se continuará trabajando igual que en el caso número 3. Cabe hacer notar que además de evaluar que los renglones cumplan la condición de selección en el índice, deberán cumplir con las demás condiciones (subconsultas), para lo cual una vez leído el renglón, éstas podrán ser verificadas. Si no se puede utilizar algún índice, se tratará igual al caso número 1.
6. **Consulta a dos tablas, sin cláusula WHERE sin índice y con índice.** Debido a que se tiene que realizar el producto cartesiano, el índice no es útil para esto, lo conveniente es utilizar la precarga para agilizar las operaciones. En el proceso antes referido (producto cartesiano) una de las tablas será leída de manera secuencial, sólo una vez, de tal forma que los renglones ya leídos no se volverán a utilizar,

por lo tanto es conveniente utilizar el algoritmo de remplazo APU en éstos; en tanto en la otra tabla, sus renglones también serán leídos de manera secuencial, con la diferencia de que éstos serán accedidos en más de una ocasión, en proporción al número de renglones de la otra tabla, de tal manera que con estos renglones se podrá utilizar el algoritmo de remplazo MRU, no sin antes determinar si es posible colocar toda la tabla en memoria principal, para mejorar la eficiencia en el tiempo de respuesta.

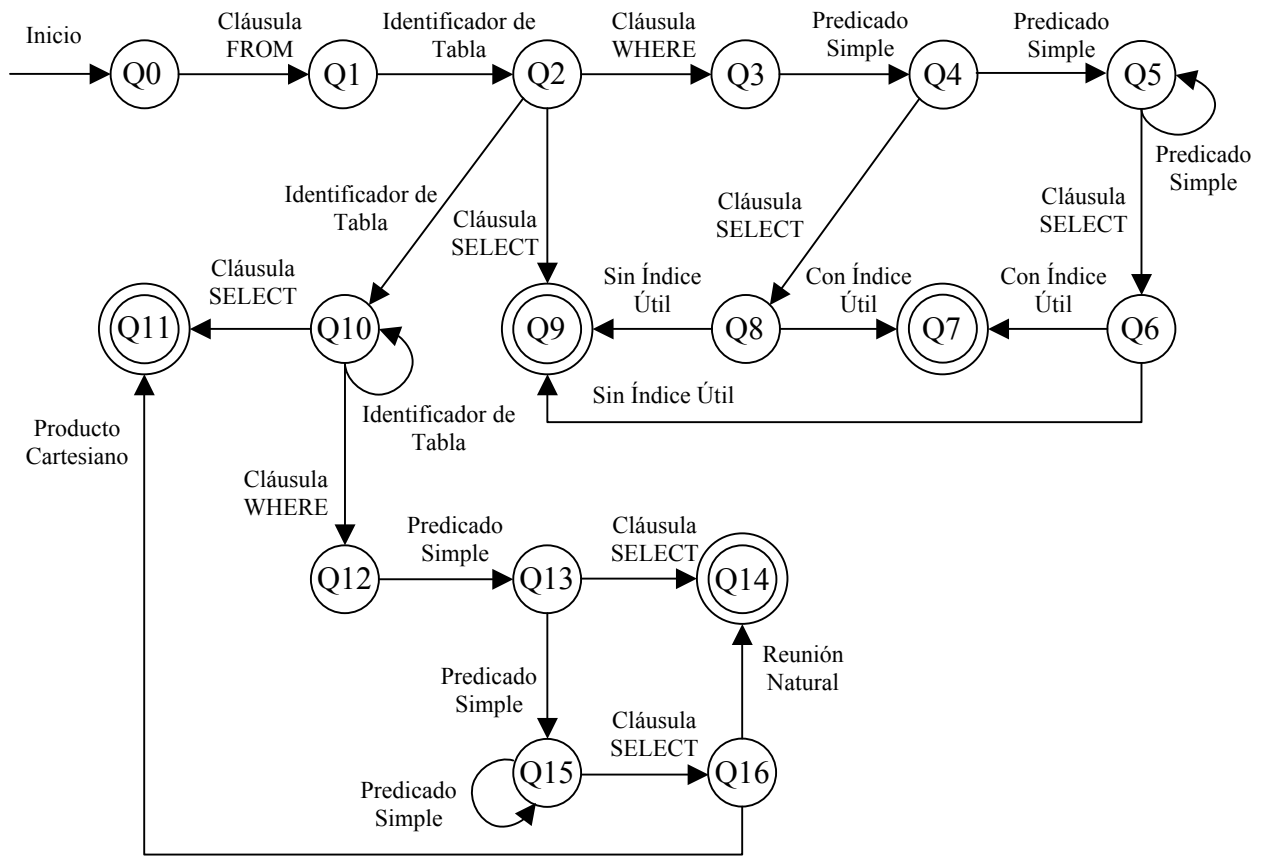
7. **Consulta a dos tablas, con cláusula WHERE sencilla, sin índice y con índice.** Conviene crearle un índice a la tabla con menor cardinalidad, en caso de que ninguna de ellas lo tenga, o si existiera alguno que no fuera útil; en forma paralela iniciar la precarga de los renglones de la otra tabla (con mayor cardinalidad); esto es debido a que con la precarga se accederán más rápido que con el índice. Si el operador entre una columna y otra es:  $>$ ,  $>=$ ,  $<$ ,  $<=$  o  $=$  se debe proceder de la siguiente forma: leer un renglón de la tabla sin índice, posteriormente obtener el dato de la columna que participa en la cláusula WHERE, a continuación buscar con el dato anterior la primer hoja en el árbol del índice que satisfaga la condición, similar al caso número 3; a partir de ahí, recorrer a la izquierda o derecha las demás hojas del árbol (si es que las hay), dependiendo del tipo de operador; terminado lo anterior, se debe continuar con otro renglón, hasta finalizar con cada uno de ellos. El algoritmo de remplazo a utilizar para las páginas que contienen los renglones de la tabla indizada será APU, debido a que tienen la misma posibilidad de que sean accedidas nuevamente y MRU para la otra tabla, que de acuerdo a la literatura es el algoritmo de remplazo más eficiente para reuniones. Si el operador fuera  $<>$ , se procedería en forma similar al caso número 6.
8. **Consulta a dos tablas, con cláusula WHERE compleja, sin índice y con índice.** También se deberá descomponer en subconsultas, tratándose igual que el caso anterior. Cabe hacer mención que puede darse una condición que involucre el producto cartesiano, para lo cual se tratará igual que el caso número 6.
9. **Consulta a tres o más tablas, sin cláusula WHERE sin índice y con índice.** Primeramente se deberán tomar dos tablas tratándolas igual que el caso número 6, con el resultado anterior (tabla resultante o intermedia) y otra tabla, nuevamente se trabajará como en el caso 6; y así sucesivamente si existieran más tablas.
10. **Consulta a tres o más tablas, con cláusula WHERE sencilla, sin índice y con índice.** Similar al caso anterior, con la diferencia de que en lugar de utilizar el caso número 6, para trabajar de dos en dos las tablas, será el caso 7.
11. **Consulta a tres o más tablas, con cláusula WHERE compleja, sin índice y con índice.** Similar al caso anterior, con la diferencia de utilizar el caso número 8, para trabajar las tablas.

En la figura 4 se muestra de forma grafica cómo se determina la estrategia a usar en la instrucción SELECT mediante un autómata de estado finito, de acuerdo a como se describió en los párrafos anteriores.

## 5. Implementación

Aunque se ha trabajado bastante en el área de los sistemas operativos, sus técnicas ignoran información que está disponible y, por lo tanto, sus estimaciones de usar ciertas páginas (datos) son inciertas comparadas con la información que se obtendría del planificador de consultas de un SMBD.

En la literatura especializada, solamente se mencionan trabajos para conocer el patrón de búsqueda o en su defecto colocar 'marcas' para informar de éste al sistema operativo. En el área de BDs se puede saber con cierta certeza este patrón en varias instrucciones de SQL.



- Q7. Se utilizará la política de remplazo APU para los datos de la tabla y FIFO para los nodos del índice, utilizando la precarga únicamente en su nodo raíz.
- Q9. Se utilizará la política de remplazo FIFO y si es posible se precargará toda la tabla.
- Q11. Se precargarán los datos de las dos tablas utilizándose la política de remplazo FIFO para una de ellas, en la otra tabla se utilizará MRU
- Q14. Se utilizará la política de remplazo APU para los datos de la tabla indexada y MRU para la que no lo este. El acceso al índice será similar al del estado Q7

Figura 4. Autómata de Estado Finito para Determinar la Estrategia de Acceso a los Datos

Actualmente existe un sistema manejador de bases de datos distribuidas experimental (denominado SiMBaDD), desarrollado por el Centro Nacional de Investigación y Desarrollo Tecnológico, el cual cuenta con varias versiones, una para cada una de las siguientes funciones: a) manejo de transacciones, b) optimización de consultas, c) procesamiento de consultas globales, d) manejo de fragmentación horizontal, y e) manejo de vistas. Para propósitos de prueba se seleccionará una versión del SiMBaDD para integrarle los módulos de precarga y administración del contenedor (Figura 5), para tal efecto se implementarán los algoritmos antes propuestos, los cuales se evaluarán contra los algoritmos que tradicionalmente se han implementado en SOs y SMBDs, como por ejemplo LRU, MRU y FIFO entre otros. Se pretende trabajar con este manejador debido principalmente a que se tiene disponible el código fuente para añadirle las rutinas necesarias y poder demostrar los beneficios de la precarga y la administración del contenedor planteados en esta investigación.

Al presente ya existe el módulo *Analizador Instrucción SQL* en el prototipo, el cual verifica que la instrucción de SQL a evaluar esté correcta tanto sintáctica, como semánticamente. Posteriormente el módulo *Selector de Renglones*, que también ya existe, lee y selecciona los renglones que cumplen la condición de búsqueda; por último el módulo *Generador de Resultados* (ya implementado) formatea el resultado final de la consulta.

Con el módulo *Analizador Patrón de Acceso a los Datos* se pretende identificar los futuros renglones a leer (patrón de acceso) a partir del tipo de consulta, tal como se mostró en la sección anterior, así como la política de remplazo a utilizar. Esta información se debe transferir después al módulo *Administrador de Contenedores*, para que a su vez solicite al módulo *Precarga de Datos* los renglones antes de que éstos le sean solicitados por el módulo *Selector de Renglones*, con su consiguiente ahorro de tiempo. De tal forma que el acceso a los datos sea más rápido, además de tener una política de remplazo ad-hoc.

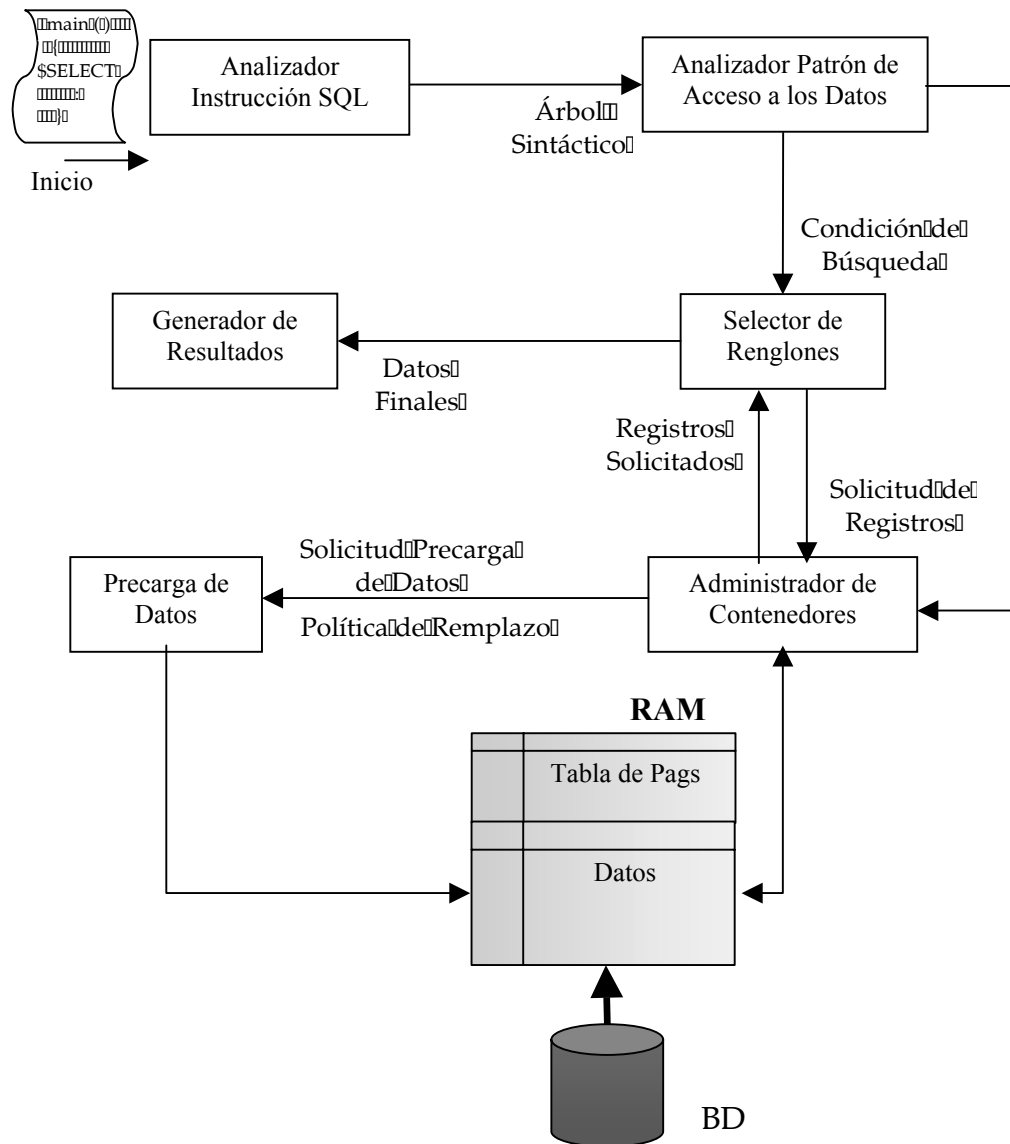


Figura 5. Arquitectura del Sistema

## 6. Conclusiones

En el área de optimización de estrategias de acceso, se han identificado problemas de investigación abiertos. Hemos encontrado que se han desarrollado muchos trabajos sobre precarga principalmente en el área de los SOs, enfocándose en tratar de obtener o predecir el patrón de acceso a los datos para aplicaciones de propósito



general. Sin embargo como se ha visto, éste es complicado de obtener comparado con el que se obtendría del planificador de consultas de un SDBD relacional, ya que éste realiza un limitado conjunto de tipos de operaciones y los patrones de referencia de páginas exhibidas por estas operaciones son muy regulares y predecibles, además de ser estos más precisos. Cabe recordar que la precarga por sí sola no basta, ya que debe estar integrada con los algoritmos de remplazo adecuados para su óptimo funcionamiento, lo cual constituye un problema complejo.

## 7. Referencias

- [1] David Lomet, Avi Silberschatz, Mike Stonebraker, y Jeff Ulman, "Database Research: Achievements and Opportunities Into the 21<sup>st</sup> Century (Extended abstract)", *Sumario del Workshop on Future of Database Systems Research*, Mayo de 1995.
- [2] Todd C. Mowry, Angela K. Demke, y Orran Krieger, "Automatic Compiler-Inserted I/O Prefetching for Out-of-Core Applications", *Second Symposium on Operating Systems Design and Implementations*, Oct. de 1996.
- [3] R. Hugo Patterson, Garth A. Gibson, Eka Ginting, Daniel Stodolsky, y Jim Zelenka, "Informed Prefetching and Caching", *Proc. of the 15<sup>th</sup> ACM Symp. on Operating System Principles*, Dec. 1995.
- [4] Abraham Silberschatz, Peter B. Galvin y Greg Gagne, *Operating System Concepts*. 6<sup>ta</sup> ed. John Wiley & Sons, 2002. pp. 34-35.
- [5] Christos Faloutsos, Raymond Ng, y Timos Sellis, "Flexible and Adaptable Buffer Management Techniques for Database Management Systems," *IEEE Transactions on Computers*, vol. 44, No. 4, Abril 1995. pp. 546-560.
- [6] Björn T. Jonson, Michael J. Franklin y Divesh Srivastava, "Interaction of Query Evaluation and Buffer Management for Information Retrieval", *Proceeding ACM SIGMOD International Conference on Management of Data*, Junio 1998, Seattle, WS, USA. 118-129.
- [7] Pei Cao, Edward W. Felten, Anna R. Karlin, y Kai Li, "A Study of Integrated Prefetching and Caching Strategies", *Proc. ACM SIGMETRICS*, 1995.
- [8] William Stallings, *Operating Systems Internal and Design Principles*. 3<sup>rd</sup> ed. Prentice Hall, 1997.
- [9] Leonard D. Shapiro, "Join Processing in Database System with Large Main Memories", *ACM Transactions on Database Systems*, Vol. 11, No. 3, Sep. 1986. pp 239-264.
- [10] Steve Bobrowski, *Oracle8i para Windows NT*, Oracle Press., 2000, pp. 4-5.
- [11] Andrew Tomkins, *Practical and Theoretical Issues in refetching and Caching*, tesis doctoral, Universidad de Carnegie Mellon, Pittsburgh, Oct. de 1997.
- [12] Douglas W. Cornell y Philip S. Yu, "Integration of Buffer Management and Query Optimization in Relational Database Environment", *Proc. of the 15<sup>th</sup> Intern. Conference on Very Large Data Base VLDB'89*, 1989. pp 247-255.
- [13] Pei Cao, Edward W. Felten, y Kai Li, "Application-Controlled File Caching Policies", *Proc. USENIX*, conferencia técnica, Junio de 1994.
- [14] Wolfgang Effelsberg, "Principles of Database Buffer Management", *ACM Transactions on Database Systems*, Vol. 9, No. 4, Dic. 1984, pp. 560-595. 1984.
- [15] Andrew S. Tanenbaum, *Modern operating systems*. 2<sup>nd</sup> ed. Prentice Hall, 2001. pp. 214-228.
- [16] Garth A. Gibson, Jeffrey Scott Vitter y John Wilkes, "Report of the Working Group on Storage I/O for Large-Scale Computing", *ACM Workshop on Strategic Directions in Computing Research*, Dic. 1996.
- [17] Pei Cao, *Application-Controlled File Caching and Prefetching*, tesis doctoral, Universidad de Princeton, Enero de 1996.
- [18] Asit Dan, Philip S. Yu y Jen-Yao Chung, "Characterization of Database Access Pattern for Analytic Prediction of Buffer Hit Probability", *VLDB Journal*, vol. 4, No. 1. 1995. pp 127-154.
- [19] G. Elsa Juárez, *Optimización de Estrategias de Acceso para un Sistema Manejador de Bases de Datos Distribuidas*, tesis de maestría, Centro Nacional de Investigación y Desarrollo Tecnológico, Cuernavaca, Mor., México, Junio de 1995.