SECRETARIA DE EDUCACIÓN PÚBLICA
SEP
SUBSECRETARÍA DE EDUCACIÓN SUPERIOR

Instituto Tecnológico de Cd. Madero

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

COORDINACIÓN DE POSGRADO EN COMPUTACIÓN
U5.095/06
AUTORIZACIÓN DE IMPRESIÓN DE TESIS DE GRADO

2006-MAY-17

C. ING. MIRNA PATRICIA PONCE FLORES
P r e s e n t e.

Me es grato comunicarle que después de la revisión realizada por el Jurado designado para su examen de grado de Maestra en Ciencias en Ciencias de la Computación, se acordó autorizar la impresión de su tesis titulada:

"TÉCNICAS DE PRECARGA INFORMADA Y POLÍTICAS DE REEMPLAZO EN UN SISTEMA MANEJADOR DE BASES DE DATOS"

Es muy satisfactorio para la División de Estudios de Posgrado e Investigación compartir con Usted el logro de esta meta. Espero que continúe con éxito su desarrollo profesional y dedique su experiencia e inteligencia en beneficio de México.

A T E N T A M E N T E
"POR MI PATRIA Y POR MI BIEN"

DRA. ANA MARÍA MENDOZA MARTÍNEZ
SUBDIRECTORA ACADÉMICA

ESTADOS UNIDOS MEXICANOS
S.E.P.
DIVISION DE ESTUDIOS DE POSGRADO E INVESTIGACION
I.T.C.M.

AMMM ' NLCO ' cerc*

Av. 1° de Mayo 1501 Pte. Col. Los Mangos
Cd. Madero, Tam. C. P. 89440
Internet: www.itcm.edu.mx
Correo Electrónico: itcm@itcm.edu.mx

Teléfono: 3-57-48-20 al 29
Fax. 3-57-48-20 ext. 1002

SECRETARÍA DE EDUCACIÓN PÚBLICA | SEP

SUBSECRETARIA DE EDUCACION SUPERIOR

INSTITUTO TECNOLÓGICO DE CD. MADERO

Sistema Nacional de Educación Superior Tecnológica

Dirección General de Educación Superior Tecnológica

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

POR MI PATRIA Y POR MI BIEN

"TÉCNICAS DE PRECARGA INFORMADA Y POLÍTICAS DE REMPLAZO EN UN SISTEMA MANEJADOR DE BASES DE DATOS"

PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A:

I.S.C. MIRNA PATRICIA PONCE FLORES

A S E S O R:

M.C. JOSÉ ANTONIO MARTÍNEZ FLORES

CD. MADERO, TAM. MEXICO JUNIO DE 2006

SECRETARÍA DE EDUCACIÓN PÚBLICA | SEP

SUBSECRETARIA DE EDUCACION SUPERIOR

INSTITUTO TECNOLÓGICO DE CD. MADERO

Sistema Nacional de Educación Superior Tecnológica

Dirección General de Educación Superior Tecnológica

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

POR MI PATRIA Y POR MI BIEN

"TÉCNICAS DE PRECARGA INFORMADA Y POLÍTICAS DE REMPLAZO EN UN SISTEMA MANEJADOR DE BASES DE DATOS"

PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A:

I.S.C. MIRNA PATRICIA PONCE FLORES

A S E S O R:

M.C. JOSÉ ANTONIO MARTÍNEZ FLORES

J U R A D O:

PRESIDENTE:	M.C. JOSÉ ANTONIO MARTÍNEZ FLORES.
SECRETARIO:	M.C. CLAUDIA GÓMEZ SANTILLÁN
VOCAL:	M.C. EURI SALGADO ESCOBAR
SUPLENTE:	M.C. GUADALUPE CASTILLA VALDEZ

CD. MADERO, TAM. MEXICO JUNIO DE 2006

Dedicatorias

- *A Dios:*

  Por permitirme cerrar un capítulo mas en mi vida y enseñarme la dirección a seguir en el momento indicado, gracias por darme la paciencia necesaria y por enseñarme que en ocasiones es mejor esperar que actuar, sin duda esta maestría es una etapa que cambio el rumbo de mi vida.

- *A mi Familia:*

  Por apoyarme y alentarme en la toma de decisión de esta etapa de mi vida, gracias por su comprensión en los tiempos difíciles, a mi mamá por entender todo el tiempo que requirió el desarrollo de este proyecto de tesis y que en ocasiones me fue imposible dedicarle tiempo suficiente a múltiples tareas de la casa.

- *A mi novio:*

  Gracias primero por el amor, la comprensión y el apoyo que me brindaste en todo este tiempo, definitivamente sin ti esto hubiese sido mas difícil de lo que fue, sabes que este logro no solo es mío, sino que gran parte de el también es tuyo, no tengo palabras para agradecerte lo mucho que me has ayudado; Te amo.

- *A mis compañeros:*

  Gracias por haber hecho los momentos difíciles mas gratos y por la amistad que me brindaron durante todo este tiempo y el que viene, definitivamente su amistad es de lo mejor que me brindo esta maestría.

Resumen

En este trabajo de tesis se aborda el problema de realizar consultas a grandes bases de datos, donde debido a el tamaño de la base de datos, la obtención de resultados se vuelve lenta, este problema existe en manejadores de bases de datos comerciales, así como en los prototipos.

Este problema ha sido abordado por varios investigadores, quienes proponen diversas técnicas que prometen mejorar el rendimiento de estas consultas; entre ellas están la precarga informada y las políticas de remplazo, sin embargo, éstas no se han implementado de forma integrada por la complejidad de las mismas, y en el caso de que hayan sido integradas, su evaluación ha sido simulada.

En este trabajo de investigación se propone e implementa la integración de las técnicas de precarga informada y políticas de remplazo (tradicionales y orientadas al área de bases de datos, además de una política propuesta llamada LLU) en un prototipo de sistema manejador de base de datos. Las pruebas realizadas al término de la integración de éstas técnicas dan como resultado que, cuando se realizan consultas a una sola tabla, no es necesario el uso de la precarga informada; mientras que, en el caso de las consultas a más de una tabla, es recomendable el uso de la precarga.

Cuando no existe memoria suficiente para la precarga, se utilizan las políticas de remplazo. Las pruebas realizadas dieron como resultado que, en el caso de consultas a una sola tabla, la política de remplazo propuesta LLU es la que tiene mejor comportamiento; mientras que en el caso de consultas a dos tablas, además de la política de remplazo LLU, 2Q es la que tiene el mejor comportamiento.

Summary

In this thesis project is approached the problem of make queries to big databases, where due to the size of the database, the time of response is slow, this problem exists in commercial database managers, as in prototypes.

This problem has been approached by several investigators, they propose several techniques that suppose to improve the efficiency of this queries; some of these techniques are the informed prefetch and replacement policies, however, these ones has not been integrated due to the complexity that implies, and if so, its evaluation has been simulated.

In this investigation work is proposed and implemented the integration of informed prefetch and replacement policies (traditionals and database oriented, also a proposed replacement policy called LLU) on a database manager system prototype. The tests carried out on the end of the integration of these techniques gives as result that, when queries to one table take place, it is not necessary the use of informed prefetch; while in the case of queries to two tables, it become in handy the use of informed prefetch.

When there is not enough memory for the prefetch, replacement policies are used. The tests carried out show that, in the case of queries to one table , the proposed replacement policy LLU have the best behavior; while in the case of a query to two tables, besides the replacement policy LLU, 2Q is the other that have good behavior.

Índice general

1 Introducción **1**

1.1 Antecedentes 1

1.2 Hipótesis 2

1.3 Objetivos 3

1.3.1 Objetivo general 3

1.3.2 Objetivos específicos 3

1.4 Justificación 4

1.5 Estado del arte 4

1.5.1 Precarga informada y políticas de remplazo en algunos manejadores de bases de datos comerciales 4

1.5.2 Trabajos de investigación relacionados con la precarga y políticas de remplazo 5

1.6 Problema a resolver 10

1.7 Organización de la tesis 11

2 Marco Teórico **13**

2.1 Terminología utilizada 13

2.2 Administración de la memoria 15

2.2.1 Memoria principal 16

2.2.2 Administradores de memoria 17

2.2.3 Administración de la memoria principal 17

2.3 La memoria virtual 22

2.4 Memoria de reserva 24

2.5 Estrategias de acceso a los datos 24

2.5.1 Estrategias de búsqueda 25

2.5.2 Estrategias de colocación 29

2.5.3 Estrategias de remplazo 29

3 Solución Conceptual **34**

3.1 Arquitectura anterior del prototipo 35

3.2 Modificación a la arquitectura anterior del prototipo 36

3.3 Análisis y diseño del uso de índices 37

3.4 Directorio de páginas 42

3.4.1 Inserción de páginas en el DP 43

3.4.2 Remplazo de páginas en el DP 44

3.5 Políticas de remplazo 46

3.5.1 Políticas de remplazo tradicionales 46

3.5.2 Políticas de remplazo orientadas al área de base de datos y de propósito general 48

3.6 Precarga informada en el directorio de páginas 60

3.6.1 Especificaciones de la precarga informada en el DP 60

4 Implementación **62**

4.1 Archivo de configuración que se utiliza para ejecutar el prototipo 63

4.2 Archivos que se actualizaron para integrar el manejador de índices al prototipo 66

4.3 Integración del manejador de índices al prototipo 66

4.4 Integración del manejador de índices con precarga informada 71

4.5 Implementación e integración del directorio de páginas al prototipo . . 73

4.6 Implementación de la precarga informada en el directorio de páginas . . 79

4.6.1 Caso de una consulta a una sola tabla 79

4.6.2 Caso de una consulta a dos tablas 80

5 Pruebas **82**

5.1 Ambiente de pruebas 82

5.2 Creación de índices 84

5.2.1 Consulta a sólo una tabla 84

5.2.2 Consulta a dos tablas 86

5.2.3 Análisis de resultados de la integración del manejador de índices al prototipo 87

5.3 Directorio de páginas 88

5.3.1 Pruebas al directorio de páginas con y sin el uso de precarga informada 88

5.4 Resultados de las pruebas realizadas al DP 97

6 Conclusiones **99**

6.1 Conclusiones de la integración del manejador de índices 99

6.2 Conclusiones de la integración de las políticas de remplazo 100

6.3 Trabajos futuros 104

A Anexos **105**

A.1 Taxonomía de la instrucción SELECT de SQL 105

Índice de tablas

1.1 Resumen del estado del arte. 9

4.1 Archivo de configuración 64

4.2 Políticas de remplazo globales 65

4.3 Estructura de control del directorio de páginas. 74

5.1 Descripción de las tablas y sus índices 85

5.2 Pruebas al manejador de índices con una consulta a una sola tabla . . . 86

5.3 Pruebas al manejador de índices con una consulta a dos tablas 87

Índice de figuras

1.1 Solicitud de datos al SO 11

2.1 Patrón de acceso a los datos 14

2.2 Ejemplo de fallo de página 14

2.3 Organización jerárquica de la memoria 15

2.4 Panorama de la administración de la memoria 18

2.5 Método de traducción de la segmentación 20

2.6 Clasificación de las estrategias de acceso a los datos 25

2.7 Accesos a las tablas en un prototipo sin índice y a otro con índices . . . 28

2.8 Comparación entre las políticas de remplazo MRU y LRU 31

2.9 Esquema de la política de remplazo 2Q 32

2.10 Esquema de la política de remplazo LIRS 33

3.1 Arquitectura anterior del prototipo 35

3.2 Arquitectura propuesta del prototipo 36

3.3 Pseudocódigo del uso de índice en una consulta a una tabla. 39

3.4 Índice con una tabla 40

3.5 Pseudocódigo del uso de índice en una consulta a dos tablas. 41

3.6 Ejemplo de una página en una tabla 43

3.7 Inserción de páginas en el directorio de páginas (DP) 44

3.8 Remplazo de una página en el DP 45

3.9 Directorio de páginas ordenado de dos formas 46

3.10 Áreas de la política de remplazo 2Q 49

3.11 Pseudocódigo de la inserción y remplazo de una página en la política de remplazo 2Q. 50

3.12 Inserción y remplazo de una página en la política de remplazo 2Q . . . 50

3.13 Pseudocódigo de la actualización de página en la política de remplazo 2Q. 51

3.14 Actualización de página en la política de remplazo 2Q 51

3.15 Pseudocódigo de la inserción y remplazo de página en la política de remplazo LIRS. 53

3.16 Inserción y remplazo de página en la política de remplazo LIRS 53

3.17 Pseudocódigo de la actualización de página en la política de remplazo LIRS. 54

3.18 Actualización de página en la política de remplazo LIRS 55

3.19 Directorio de páginas usando la política de remplazo LLU 57

3.20 Precarga de datos en memoria principal 61

4.1 Estructura de la lista de expresiones de la cláusula WHERE. 67

4.2 Estructura que almacena en la memoria compartida, la información de los índices. 68

4.3 Accesos a datos al procesar una consulta en la versión anterior del prototipo. 70

4.4 Integración del manejador de índices al prototipo cuando es una tabla. 72

4.5 Integración del manejador de índices al prototipo cuando son dos tablas. 73

4.6 Definición de la estructura *Pagina*. 75

4.7 Definición de la estructura *InformacionDP*. 76

4.8 Selección del lector de renglones a utilizar. 78

5.1 Consulta a sólo una tabla, sin cláusula WHERE sencilla sin índice, sin el uso de la precarga 89

5.2 Consulta a sólo una tabla, sin cláusula WHERE sencilla sin índice, con el uso de la precarga 89

5.3 Consulta a sólo una tabla, con cláusula WHERE sencilla, con índice, sin el uso de la precarga 90

5.4 Consulta a sólo una tabla, con cláusula WHERE sencilla, con índice, con el uso de la precarga 91

5.5 Consulta a sólo una tabla, con cláusula WHERE compleja, sin índice, sin el uso de la precarga 92

5.6 Consulta a sólo una tabla, con cláusula WHERE compleja, sin índice, con el uso de la precarga 92

5.7 Consulta a sólo una tabla, con cláusula WHERE compleja, con índice, sin el uso de la precarga 93

5.8 Consulta a sólo una tabla, con cláusula WHERE compleja, con índice, con el uso de la precarga 94

5.9 Consulta a dos tablas, con cláusula WHERE sencilla, sin índice y con índice, sin el uso de la precarga 95

5.10 Consulta a dos tablas, con cláusula WHERE sencilla, sin índice y con índice, con el uso de la precarga 96

5.11 Consulta a dos tablas, con cláusula WHERE compleja, con y sin índice, sin el uso de la precarga 97

5.12 Consulta a dos tablas, con cláusula WHERE compleja, con y sin índice, con el uso de la precarga 97

5.13 Resumen de los resultados obtenidos al realizar pruebas con las diferentes políticas de remplazo sin el uso de la precarga informada 98

5.14 Resumen de los resultados obtenidos al realizar pruebas con las diferentes políticas de remplazo con el uso de la precarga informada 98

6.1 Resultados de las pruebas realizadas a los casos de taxonomía de la instrucción SELECT con un DP de 1000 páginas sin el uso de la precarga 101

6.2 Resultados de las pruebas realizadas a los casos de taxonomía de la instrucción SELECT con un DP de 1000 páginas con el uso de la precarga 102

6.3 Resultados de las pruebas realizadas a los casos de taxonomía de la instrucción SELECT con un DP de 2000 páginas sin el uso de la precarga 102

6.4 Resultados de las pruebas realizadas a los casos de taxonomía de la instrucción SELECT con un DP de 2000 páginas con el uso de la precarga 103

6.5 Resultados de las pruebas realizadas a los casos de taxonomía de la instrucción SELECT con un DP de 3000 páginas sin el uso de la precarga 103

6.6 Resultados de las pruebas realizadas a los casos de taxonomía de la instrucción SELECT con un DP de 3000 páginas con el uso de la precarga 104

CAPÍTULO 1

Introducción

1.1 Antecedentes

Uno de los problemas actuales de las empresas que manejan una gran cantidad de información, es la lentitud del tiempo de respuesta de las consultas realizadas al Sistema Manejador de Bases de Datos (SMBD) [1]. Diversos autores han realizado investigaciones de técnicas que permitan lograr un mejor tiempo de respuesta en las consultas realizadas al SMBD. Dentro de estas técnicas podemos mencionar la *precarga informada* y la administración del contenedor mediante *políticas de remplazo* cada vez más eficientes.

Se ha demostrado que la *precarga informada* y las *políticas de remplazo* contribuyen a optimizar el desempeño de una consulta [2]. Desafortunadamente dichas técnicas han sido investigadas de manera individual en la mayoría de los trabajos [3].

De acuerdo con la investigación realizada sobre el estado del arte en *precarga informada y políticas de remplazo*, la mayor parte de estas investigaciones están encaminadas para ser usadas en los Sistemas Operativos (SOs) [4, 5, 6, 7, 8], con muy buenos resultados;

lo cual ha dado la pauta y el interés de utilizarlas y mejorar el tiempo de respuesta de las consultas en los SMBDs.

El presente trabajo de investigación integra: un manejador de índices y la técnica de precarga informada e implementa un directorio de páginas que utiliza cinco políticas de remplazo; entre las cuales dos de ellas son políticas tradicionales, una de propósito general, una orientada al área de base de datos y una política propuesta.

Es importante mencionar que la implementación e integración de la precarga informada con las políticas de remplazo se realizó en un sistema manejador de base de datos experimental denominado SiMBaDD (Sistema Manejador de Bases de Datos Distribuidas), ya que se dispone del código fuente y es posible agregar módulos sin problema alguno. SiMBaDD es un prototipo académico desarrollado por el Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET), perteneciente al Sistema Nacional de Educación Superior Tecnológica (SNEST). Dicho prototipo consiste en un paquete de cómputo que permite almacenar, consultar y procesar información de bases de datos distribuidas.

En la revisión de la literatura se encontró que la mayoría de los trabajos [3, 4, 6, 7, 8, 1, 9, 10, 11, 12] han sido simulados. La ventaja de esta investigación es que se probaron de manera real los beneficios de la integración de las dos estrategias antes mencionadas.

1.2 Hipótesis

La integración de la precarga informada con políticas de remplazo orientadas al área de base de datos, disminuye el tiempo de respuesta de las consultas realizadas en un SMBD experimental

1.3 Objetivos

1.3.1 Objetivo general

Implementar un directorio de páginas que utilice diferentes políticas de remplazo e integrarlo a un prototipo de SMBD que utiliza la precarga informada para disminuir el tiempo de respuesta de las consultas.

1.3.2 Objetivos específicos

- Integrar un manejador de índices al prototipo.
- Implementar dos políticas de remplazo tradicionales LRU y MRU (Least Recently Used y Most Recently Used).
- Implementar dos políticas de remplazo, una diseñada específicamente para el área de base de datos (2Q / Two Queues) y otra de propósito general (LIRS / Low Interreference Recency Set).
- Implementar la política de remplazo propuesta (LLU / Least Likely to Use) [13].
- Integrar las políticas de remplazo implementadas con la precarga informada en el prototipo experimental.
- Evaluar y comparar el desempeño que se obtiene al realizar las consultas al prototipo experimental utilizando las diferentes políticas de remplazo en el Directorio de Páginas (DP).

1.4 Justificación

En el área de los SOs se han implementado técnicas de precarga y políticas de remplazo para aplicaciones de carácter general con buenos resultados; por tanto, se intuye que al aplicar ambas técnicas en los SMBDs se conseguirá disminuir el tiempo de respuesta de las consultas. En este trabajo de investigación se pretende integrar ambas técnicas, así como realizar una comparación entre dos políticas tradicionales, una política de bases de datos, una política de carácter general y la política propuesta.

Cabe hacer mención que hasta el momento se han desarrollado pocos trabajos de investigación en el área de bases de datos en donde evalúan la precarga con políticas de remplazo de forma conjunta [14], y gran parte de ellos son evaluados mediante simulación.

1.5 Estado del arte

A la fecha se han realizado diversos trabajos de investigación que comprenden desde los SMBDs comerciales (Sybase 5.5, Oracle 9i y SQL Server 2000), los cuales emplean técnicas de precarga y políticas de remplazo, hasta trabajos de investigación en el campo de los SOs y bases de datos que utilizan dichas técnicas. A continuación se describen los más relevantes para este trabajo.

1.5.1 Precarga informada y políticas de remplazo en algunos manejadores de bases de datos comerciales

- SyBase 5.5: Este SMBD tiene implementada la precarga y utiliza la política de remplazo LRU (Least Recently Used / Menos Recientemente Utilizada);

sin embargo, no especifica la forma en que se realiza la precarga; además, la documentación que proporcionan no menciona si las estrategias están o no integradas [15].

- Oracle 9i: Este SMBD utiliza la precarga de datos así como la política de remplazo LRU; sin embargo, no hay información disponible que describa cómo son realizadas ni tampoco si ambas están o no integradas [16].

- SQL Server 2000: Este SMBD utiliza la técnica de precarga llamada lectura adelantada (read ahead), la cual consiste en cargar en la memoria de reserva los datos contiguos a los que fueron solicitados. Utiliza tanto la política de remplazo de *reloj* como la política LRU. No hay información disponible acerca de si la precarga y las políticas de remplazo están integradas [17, 18].

1.5.2 Trabajos de investigación relacionados con la precarga y políticas de remplazo

- Pei Cao, Edward W. Felten, Anna R. Karlin y Kai Li en [3] proponen la integración de la precarga informada con las estrategias de reserva, y mencionan que la mayor parte de las investigaciones de precarga se han realizado sin reserva, o bien para una estrategia de reserva fija. Además, presentaron un estudio teórico de dos estrategias de precarga llamadas agresiva y conservadora. La conservadora ejecuta el mínimo de cargas y minimiza el tiempo de ejecución; mientras que la agresiva siempre precarga el siguiente bloque en la primera oportunidad y remplaza la página en la que su referencia es la más lejana en el futuro. Mediante simulación compararon los algoritmos de las dos estrategias propuestas contra seis algoritmos existentes para reserva y precarga. Los resultados mostraron que ambas estrategias pueden reducir el tiempo de ejecución en más de un 50 %.

- R. Hugo Patterson, Garth A. Gibson, Eka Ginting, Daniel Stodolsky y Jim Zelenka

en [9] indican cómo usar los patrones de acceso de las aplicaciones, además de ubicar dinámicamente contenedores de archivos para las siguientes tres demandas: precarga de páginas conocidas, reserva de páginas sugeridas para su reutilización y reserva de datos recientemente utilizados, para accesos desconocidos. En este trabajo se utiliza el análisis de costo-beneficio para determinar la mejor ubicación de los contenedores. Además, se implementó la precarga informada integrada con la reserva, y se observó que la precarga informada redujo el tiempo de ejecución en diversas aplicaciones. La precarga informada redujo el tiempo de ejecución en mas del 42 %, mejorando así el desempeño de un manejador de base de datos. La integración de la precarga informada y la reserva incrementó el rendimiento global. Los autores recomiendan como trabajo futuro agregar nuevos estimadores; por ejemplo, el uso de la memoria virtual con el manejo de la memoria de reserva.

- Gideon Glass y Pei Cao [10] realizaron un estudio de remplazo de páginas con base en el comportamiento de la referencia de memoria. Proponen una nueva política de remplazo de páginas la cual llaman SEQ, la cual consiste en detectar grandes secuencias de fallo de páginas y utilizar la política de remplazo Más Recientemente Utilizada (MRU). La política propuesta se prueba en un entorno simulado y los resultados obtenidos muestran que, cuando se tiene un patrón de acceso regular, la política de remplazo Menos Recientemente Utilizada (LRU) se comporta deficientemente, por lo cual aconsejan utilizar su nueva política de remplazo, la cual, como ya se mencionó, utiliza la política MRU. Además, encuentran que para sistemas multiproceso, la política propuesta es una buena política de remplazo global. Cabe señalar que las pruebas se realizaron con un número pequeño de aplicaciones y como trabajo futuro proponen incorporar la precarga en la política SEQ.

- Todd C. Mowry y Chi-Keung Luk en [8] proponen una nueva instrucción de precarga, por medio de la cual el hardware y software cooperan para ocultar la latencia de la memoria (fallos de página). El hardware ejecuta una precarga

secuencial agresiva combinada con un nuevo mecanismo de filtrado de precarga que le permite realizar una lectura anticipada sin contaminar la reserva. También proponen e implementan un nuevo compilador que automáticamente inserta instrucciones de precarga en el código ejecutable para precargar los datos suficientemente adelantados. Los resultados mostraron que este nuevo enfoque mejora el tiempo de ejecución en un promedio de 13.3 % y que ambas propuestas son esenciales y complementarias.

- Steven P. Vanerwiel y David J. Lilja en [7] proponen un controlador de precarga de datos llamado DPC, el cual combina instrucciones de bajo nivel con la precarga de un compilador directo. Los resultados muestran que el controlador de precarga puede mejorar el uso de la reserva y el tiempo de ejecución de varios benchmarks (bases de datos de prueba). En las pruebas de comparación que realizaron, el controlador de precarga ejecutó la técnica de precarga de software usando una tabla de predicción de referencias.

- Stephane Bressan, Chong Leng Goh, Beng Chin Ooi y Kian-Lee Tan en [11] presentan un marco de trabajo para afinar parámetros con el fin de modelar diversos algoritmos de remplazo y disminuir accesos a disco. Además, añaden que no existe un único algoritmo que sea mejor que todos los demás en todos los casos y mencionan que algunos manejadores sólo utilizan una política de remplazo o tal vez dos.

- Jamison Collins, Suleyman Sair y Brad Calder en [6] proponen el uso de un apuntador a la reserva, con el propósito de darle seguimiento a las transiciones para ayudar a la precarga y acelerar el proceso de carga basado en apuntadores. El apuntador permite que se ejecuten hilos de forma anticipada a la ejecución. Esta investigación está orientada hacia los SOs. Presentan cuatro diferentes configuraciones del apuntador a la reserva, y proponen el uso de la precarga predictiva en combinación con el apuntador a la reserva como una estrategia más para reducir el cuello de botella en la memoria de reserva de nivel 1 (L1).

- Stefan G. Berg en [1] asevera que al precargar en la reserva se tiene la habilidad para detectar y predecir patrones de referencia de memoria. Realiza una comparación de la manera cómo se realiza la precarga para cada uno de los patrones de referencia de memoria que fueron identificados. Expone cómo se pueden combinar técnicas de precarga para tratar con un gran número de patrones de referencia de la memoria, y propone aplicar la precarga para un sistema multimedia.

- Zhenlin Wang en [4] propone la cooperación entre software y hardware y demuestra la efectividad de esta integración en el remplazo de la reserva y la precarga. Además señala que la combinación de ambas elimina o reduce en gran medida el cuello de botella que se presenta en la memoria de reserva de nivel 2. Su propuesta consiste en un nuevo compilador y mecanismos de arquitectura que usan la predicción de accesos futuros (precarga predictiva) para mejorar la decisión de los remplazos de la reserva, para lo cual usan la política de remplazo LRU. Sus resultados garantizan minimizar el cuello de botella de la memoria con su técnica propuesta.

- Ali R. Butt, Chris Gniady y Y. Charlie Hu en [12] aseveran que el impacto de la precarga que realiza el sistema operativo puede disminuir la brecha que existe entre los buenos y malos algoritmos de remplazo. La mayoría de los SOs modernos tienen una precarga implícita, la cual no se toma en cuenta en los trabajos de investigación recientes. Debido a que sólo son evaluados los aciertos y fallos obtenidos en simuladores, implementaron un simulador que emula el comportamiento de la precarga del sistema operativo Linux y compararon una serie de algoritmos de remplazo.

La Tabla 1.1 muestra un resumen de los trabajos de investigación de las técnicas de precarga informada y políticas de remplazo:

Tabla 1.1: Resumen del estado del arte.

Investigadores	Año	Trabajo de Investigación	Área	Tipo de Trabajo
Pei Cao, Edward W. Felten, Anna R. Karlin y Kai Li	1995	Integra la precarga con la reserva.	SOs	Simulación
R. Hugo Patterson, Garth A. Gibson, Eka Ginting, Daniel Stodolsky y Jim Zelenka	1995	Integra la precarga con la reserva, recomienda el uso de la memoria virtual.	General	Implementación
Gideon Glass y Pei Cao	1997	Remplazan páginas con un nuevo algoritmo de remplazo llamado SEQ.	SOs	Simulación
Todd C. Mowry y Chi-Keung Luk	1998	Implementan un nuevo compilador que precarga datos de manera anticipada.	General	Propuesta
Steven P. Vanerwiel y David J. Lilja	1999	Controlador de precarga llamado DPC.	SOs	Propuesta
Stephane Bressan, Chong Leng Goh, Beng Chin Ooi y Kian-Lee Tan	2000	Aportan un marco de trabajo para implementar varias políticas de remplazo al mismo tiempo.	BDs	Simulación
Jamison Collins, Suleyman Sair y Brad Cader	2002	Apuntador a la reserva para dar seguimiento a las transiciones.	SOs	Propuesta
Stefan G. Berg	2004	Combinar técnicas de precarga para tratar varios patrones de acceso a la memoria.	SOs	Propuesta
Zhenlin Wang	2004	Enfoque cooperativo entre hardware y software, que usa un nuevo compilador y mecanismos de arquitectura para predecir los accesos futuros.	SOs	Propuesta
Ali R. Butt, Chris Gniady y Y. Charlie Hu	2005	Evalua el impacto de la precarga de los sistemas operativos en las políticas de remplazo.	BDs	Simulación

De acuerdo con los trabajos anteriores, tenemos la hipótesis de que las técnicas de precarga y políticas de remplazo mejoran el tiempo de ejecución en distintas

aplicaciones; sin embargo, en el área de los sistemas operativos dichas técnicas han sido generalmente simuladas y escasamente implementadas. Las escasas investigaciones realizadas en el área de las bases de datos han sido de igual manera simuladas, y en las que ya se han implementado, no se proporciona información detalla de la manera cómo son realizadas.

1.6 Problema a resolver

Actualmente los SMBDs delegan la administración de sus datos al SO, el cual administra los datos con políticas de remplazo que en varios de los casos son inapropiadas [19, 20]; además de que no integra éstas con la precarga de datos por ser un trabajo muy complejo [3]. Debido a que no se utilizan estas técnicas en forma conjunta se obtiene un desempeño inadecuado en los SMBDs.

El problema radica principalmente en las operaciones de lectura/escritura que se realizan de manera reactiva; es decir, se solicitan datos al SO cuando éstos son requeridos, (ver Figura 1.1), es esta figura se observa el número de operaciones que se realizan desde que la aplicación solicita un dato, hasta que el SO provee este dato, sí la solicitud se realiza de dato en dato (uno en uno), el número de operaciones de lectura/escritura puede incrementar de manera considerable cuando se manejan bases de datos con grandes cantidades de información.

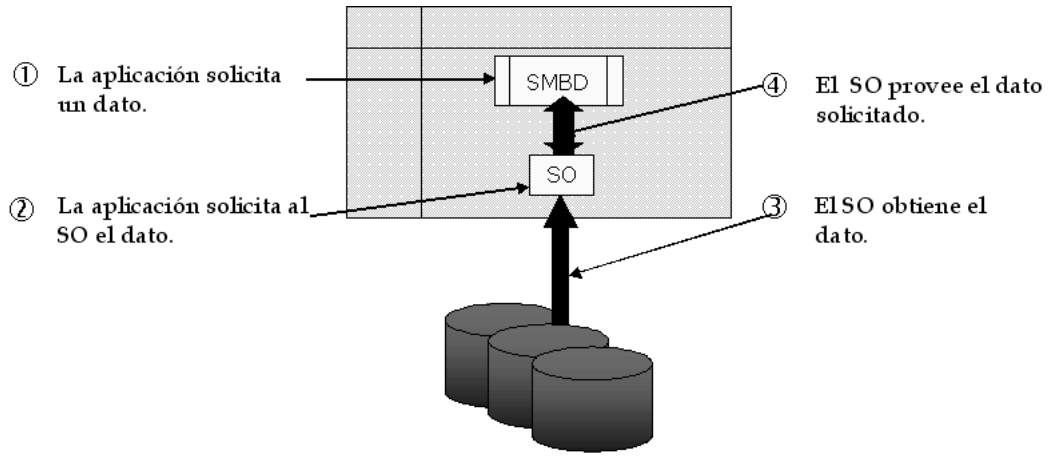

Figura 1.1: Solicitud de datos al SO

1.7 Organización de la tesis

Este trabajo de investigación se divide en 6 capítulos y 1 anexo. En el Capítulo 2 se incluyen los fundamentos teóricos que describen los conceptos utilizados en este documento de tesis tales como:

- Patrón de acceso a los datos.
- Fallo de página.
- Administrador de memoria.
- Organización de la memoria.
- Descripción del uso de un manejador de índices.
- Estrategias de acceso a los datos.
- Precarga de datos.
- Políticas de remplazo.
- Concepto del directorio de páginas.

En el Capítulo 3 se detalla el diseño del uso de índices en el prototipo, el diseño de un directorio de páginas, la inserción y remplazo de páginas en el directorio de páginas, y por último, el diseño de las políticas de remplazo que se implementaron.

En el Capítulo 4 se describen las estructuras más importantes que intervienen en la integración del manejador de índices al prototipo, se detalla la implementación del directorio de páginas y las estructuras utilizadas, así como también las funciones que permiten insertar y remplazar páginas en el directorio de páginas.

En el Capítulo 5 se describe el ambiente de las pruebas realizadas, además se muestran los resultados obtenidos al realizar consultas a una y dos tablas.

En el Capítulo 6 se describen las conclusiones a las que se llegó en este trabajo de investigación, además de proponer trabajos futuros.

CAPÍTULO 2

Marco Teórico

En este capítulo se define la terminología utilizada en el resto del documento, de la misma forma se explican de forma breve algunos temas tales como la administración de la memoria, estrategias de acceso a datos, precarga de datos, políticas de remplazo, entre otros, de tal forma que el resto del documento sea fácil de comprender.

2.1 Terminología utilizada

- Patrón de acceso a los datos: Se llama *patrón de acceso a los datos*, a la secuencia de datos solicitados desde una aplicación al SO, en la Figura 2.1 se muestra un ejemplo de patrón de acceso a los datos.

  Recientes investigaciones muestran que la mayoría de las aplicaciones siguen un patrón de acceso a los datos. Las aplicaciones científicas muestran un patrón de acceso cíclico; mientras que las bases de datos tienen un patrón de acceso regular [21].

- Fallo de página: Cuando se solicitan datos de una página, ocurre un *fallo de página* sí ésta es buscada en memoria principal y no es encontrada.

0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4

Figura 2.1: Patrón de acceso a los datos

En la Figura 2.2 se muestra un contenedor en memoria principal de tamaño 4, donde se insertarán los datos conforme se vayan solicitando, en el inciso *a* se puede observar que el dato *0* ha sido solicitado y no se encuentra almacenado en el contenedor, por lo tanto se incurre en un fallo de página; lo mismo sucede en los incisos *b*, *c* y *d* al momento de solicitar los datos *1, 2 y 3* respectivamente; sin embargo, en el inciso *e* al volverse a solicitar el dato *0*, éste ya se encuentra almacenado en el contenedor, aquí se observa que en este caso no se presenta un fallo de página.

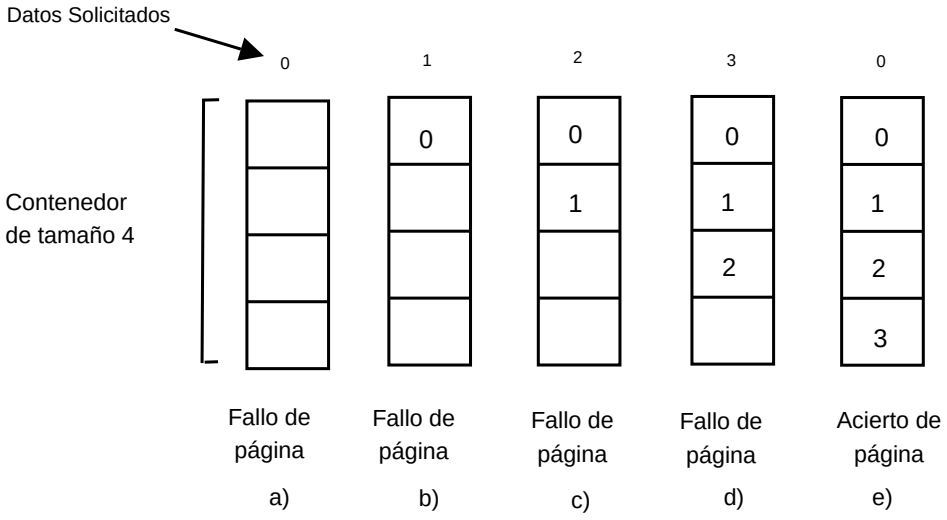

Figura 2.2: Ejemplo de fallo de página

- Administrador de memoria: Es el componente del sistema operativo que realiza la tarea de coordinar los diferentes tipos de memoria, unas de sus tareas son llevar un registro de las partes de la memoria que están en uso y cuáles no, asignar memoria a los procesos y liberarla cuando ya no se necesite, además de administrar el intercambio entre procesos (swapping) [22, 23].

2.2 Administración de la memoria

La organización y administración de la memoria principal de un sistema ha sido y es uno de los factores más importantes en el diseño de los sistemas operativos. Históricamente el almacenamiento principal se ha considerado como un recurso costoso, por lo cual su uso debería optimizarse [22].

Los sistemas con varios niveles de almacenamiento, como lo muestra la Figura 2.3, requieren destinar recursos para administrar el movimiento de programas y datos entre estos niveles [24], los cuales se dividen de acuerdo a su capacidad de almacenamiento, su velocidad y su costo. El nivel más alto lo ocupan los registros del CPU, los cuales tienen muy poca capacidad de almacenamiento, alta velocidad y alto costo. En la medida en que recorremos los niveles de arriba hacia abajo, la capacidad de almacenamiento aumenta y el costo disminuye al igual que la velocidad.

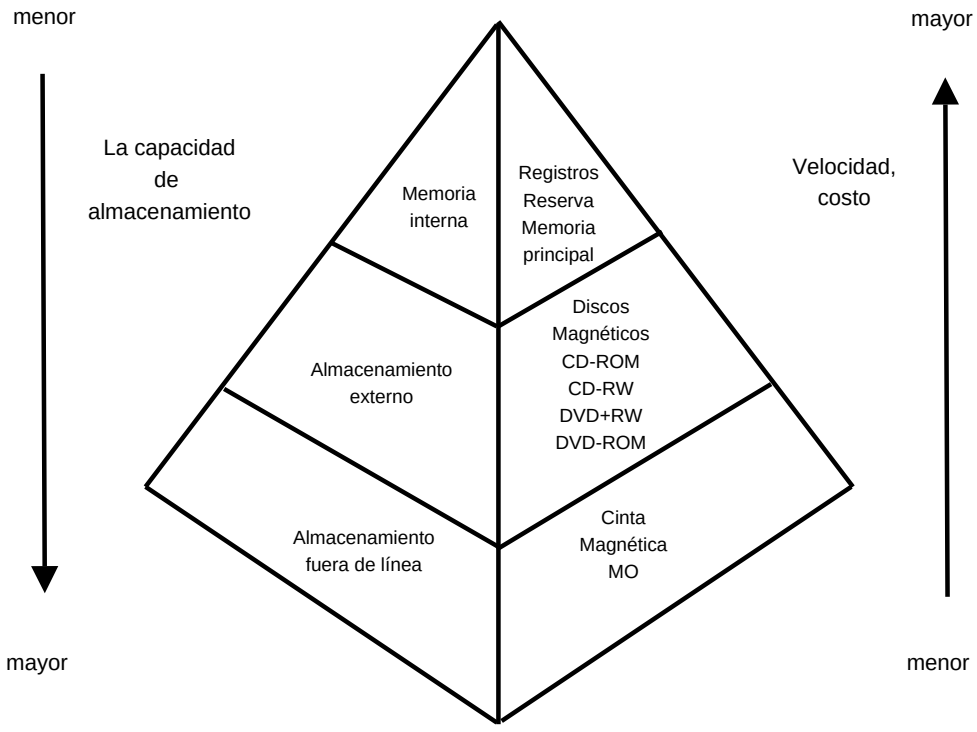

Figura 2.3: Organización jerárquica de la memoria

En el primer nivel de almacenamiento se encuentra la memoria principal entre otras, a la cual se refiere este trabajo de investigación.

2.2.1 Memoria principal

El almacenamiento primario de la computadora es la *memoria principal*, la cual se compone de un conjunto de celdas básicas con una determinada organización. Cada celda puede almacenar un bit de información. Los bits se agrupan en unidades direccionables denominadas palabras. La longitud de una palabra la determina el número de bits que la componen y constituye la resolución de la memoria (mínima cantidad de información direccionable). La longitud de palabra suele variar desde 8 bits (*byte*) hasta 64 bits [25].

Cada celda básica es un dispositivo físico con dos estados estables (o semiestables) con capacidad para cambiar el estado (escritura) y determinar su valor (lectura).

Desde el punto de vista conceptual y con independencia de la tecnología, se considera la celda básica de memoria como un bloque con tres líneas de entrada (entrada de datos, selección y lectura/escritura) y una línea de salida (salida del dato). La celda sólo opera cuando la selección está activa.

Lo ideal sería contar con un tipo de memoria infinitamente grande, rápida, no volátil y de bajo costo. Desafortunadamente la tecnología no provee ese tipo de memoria [26]; en consecuencia, la mayoría de las computadoras cuentan con la ya mencionada jerarquía de memoria, con una pequeña cantidad de memoria de reserva muy rápida, costosa y volátil, unos cuantos megabytes de memoria principal volátil, de velocidad media y precio medio y cientos de gigabytes de memoria secundaria muy lenta, barata y no volátil.

2.2.2 Administradores de memoria

El sistema operativo es encargado de coordinar los diferentes tipos de memoria. Al componente del sistema operativo que realiza esta tarea se le denomina *administrador de memoria*. Su trabajo es llevar un registro de qué partes de la memoria están en uso y cuáles no, asignar memoria a los procesos y desasignarla cuando ya no la necesiten y también administrar el intercambio (*swapping*).

El administrador de memoria del sistema operativo provee transparencia para las aplicaciones que utilizan la memoria del sistema; sin embargo, cuando un sistema computacional está dedicado a proveer un servicio específico, resulta conveniente que las aplicaciones encargadas de dicho servicio mantengan una estrecha comunicación con el administrador de memoria, de modo que sean las aplicaciones las que de cierto modo, administren la memoria [18, 22, 27, 28].

Para los SMBDs que manipulan gran cantidad de información, es de vital importancia la administración de la memoria. La mayoría de los SMBDs comerciales utilizan una porción de la memoria principal como área de reserva; por ejemplo: SQL Server de Microsoft [29], Oracle9i [16] y Sybase SQL Anywhere [15].

2.2.3 Administración de la memoria principal

La administración de la memoria es una de las actividades más importantes desde los primeros días de la computación, y desde entonces se ha necesitado más memoria de la existente físicamente en la computadora [20, 30, 31]. Para resolver este problema se desarrollaron varias estrategias que fueron evolucionando. En la Figura 2.4 se muestran gráficamente los esquemas de uso más importantes que se han desarrollado para la administración de la memoria, en el inciso *a* se observa que la memoria sólo podía ser utilizada por un proceso a la vez, mientras que en el inciso *b* se observa que se puede

seccionar la memoria de tal forma que varios procesos pueden ejecutarse al mismo tiempo.

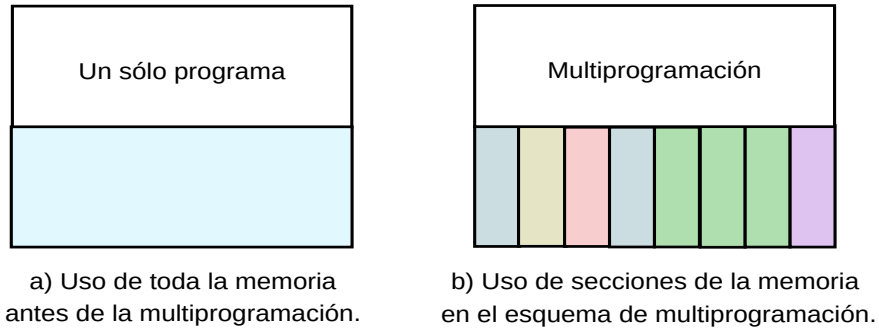

Figura 2.4: Panorama de la administración de la memoria

En un principio se comenzó con estrategias sencillas correspondientes a sistemas con un único proceso de usuario (monousuario). Se utilizaban direcciones físicas de memoria para almacenar el programa que se estuviera ejecutando en un momento dado. La desventaja era que estaba limitado por la cantidad de RAM disponible.

Las empresas e instituciones que habían invertido grandes sumas en la compra de equipo de cómputo, se dieron cuenta que los sistemas consumían una gran cantidad de tiempo de ejecución en operaciones de entrada/salida y que, durante estas operaciones, la operación de la unidad central de procesamiento (UCP) era prácticamente nula. Por este motivo comenzaron a investigar cómo hacer que la UCP se mantuviera más tiempo ocupada. Fue así como nació el concepto de *multiprogramación.*

La idea de la multiprogramación consiste en poner en la memoria principal más de un proceso al mismo tiempo, de manera que si el proceso que se está ejecutando en un determinado momento realiza una operación de entrada/salida, el sistema operativo pueda tomar otro proceso para que se ejecute.

Para lograr esto, la primera solución fue dividir la memoria en particiones fijas y utilizar

una estructura de cola para los procesos que habrán de ejecutarse. Otra solución fue que cada partición tuviera asociada una cola de tareas.

Para la administración de las particiones fijas se dispuso de un proceso planificador, el cual tomaba en cuenta los requerimientos de memoria de cada uno de los demás procesos, así como también las particiones de memoria que estaban disponibles. La mayor dificultad de esta forma de administración fue seleccionar el tamaño adecuado de las particiones.

Después se hizo evidente que, para un conjunto dinámico de procesos ejecutándose, no es posible encontrar el tamaño adecuado para las particiones, y por lo tanto, se dispuso de particiones con tamaño variable. Para poder realizar esto, se utilizan las técnicas de paginación o segmentación.

La paginación

La paginación consiste en dividir la memoria principal en unidades de tamaño fijo. De esta forma la memoria puede ser utilizada por diversos procesos y cada proceso puede utilizar más de una página sin que estas sean adyacentes; de esta manera se obtiene un mejor desempeño de la memoria.

Esta técnica es utilizada en algunos de los procesadores desarrollados por motorola e IBM.

La segmentación

La segmentación es otra forma de dividir la memoria principal, ésta es comúnmente utilizada por los procesadores intel; toma ventaja de que los programas están compuestos por datos, código y una pila, de tal forma que estas componentes se ubican

en diferentes segmentos de la memoria (al igual que la paginación estos no necesitan ser adyacentes) [32].

Uno de los aspectos más importantes de la segmentación es que utiliza direcciones virtuales de memoria las cuales deben ser convertidas a direcciones reales de memoria; la(s) conversión(es) consiste(n) en un conjunto de operaciones aritméticas. La Figura 2.5 muestra un ejemplo de la conversión de una dirección virtual a dirección real en un sistema hexadecimal.

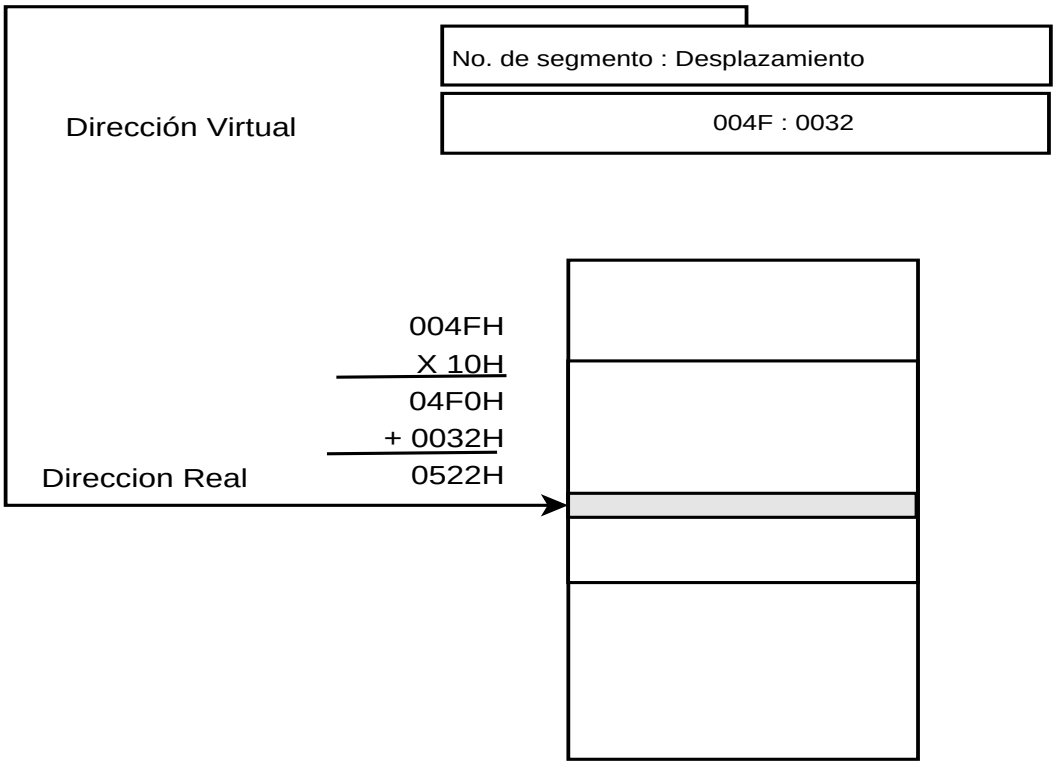

Figura 2.5: Método de traducción de la segmentación

Dentro de la técnica de multiprogramación surgieron tres problemas interesantes: la fragmentación, la protección y la relocalización.

- El problema de la fragmentación.- Ocurre debido a una mala elección en el tamaño de las particiones, ocasionando un desaprovechamiento de la memoria.

La fragmentación puede ser de dos tipos:

  - La fragmentación interna.- Ocurre cuando una parte de la memoria asignada a un proceso queda sin uso alguno.
  - La fragmentación externa.- Ocurre cuando una partición disponible no se emplea porque es muy pequeña para cualquiera de los procesos en la cola de espera.

- El problema de la relocalización.- Consiste en que los programas que necesitan cargarse a memoria principal ya están compilados y ligados a una serie de referencias de datos e instrucciones (rutinas y procedimientos), las cuales ya no son válidas en el espacio de direcciones de memoria principal de la sección en la que se carga el programa.

- El problema de la protección.- Consiste en que una vez que ha sido cargado un programa en algún segmento particular de la memoria principal, nada le impide al programador que intente direccionar (por error o deliberadamente) localidades de memoria menores al límite inferior o mayores al límite superior de su programa; es decir, referir localidades fuera de su espacio de direcciones. Obviamente, éste es un problema de protección, ya que no es legal leer o escribir en áreas de otros programas.

Una vez que surgió la multiprogramación, los usuarios comenzaron a explorar la forma de ejecutar grandes cantidades de código en áreas de memoria muy pequeñas, auxiliados por algunas llamadas al sistema operativo. Es así como nacen las capas (*overlays*).

La técnica de capas consiste en que el programador divide lógicamente un programa muy grande en secciones que puedan almacenarse en las particiones de la memoria principal. Al final de cada sección del programa (o en otros lugares necesarios) el programador inserta una o varias llamadas al sistema con el fin de descargar la sección presente de la

memoria principal y cargar otra, que en ese momento reside en disco duro u otro medio de almacenamiento secundario.

Aunque esta técnica era eficaz (porque resolvía el problema) no era eficiente (ya que no lo resolvía de la mejor manera). Esta solución requería que el programador tuviera un conocimiento muy profundo del equipo de cómputo y de las llamadas al sistema operativo. Otra desventaja era la portabilidad de un sistema a otro: las llamadas cambiaban y los tamaños de particiones también. Con esta técnica se podían ejecutar programas más grandes que las particiones de memoria principal, donde la división del código corría a cuenta del programador y el control, a cuenta del sistema operativo.

2.3 La memoria virtual

La necesidad cada vez más imperiosa de ejecutar programas más grandes y el crecimiento en poder de las UCPs, propiciaron que los diseñadores de los sistemas operativos tuvieran que implantar un mecanismo para ejecutar automáticamente programas más grandes que la memoria real disponible, dando lugar al concepto de memoria virtual.

Los objetivos de la memoria virtual son los siguientes:

- Que los procesos *crean* disponer de una cantidad de memoria principal mayor a la instalada en el sistema.
- Que la transferencia de información entre la memoria principal y la secundaria sea transparente a los procesos.
- Que puedan ejecutarse procesos más grandes que la memoria principal instalada.

- Que se aumente el grado de multiprogramación del sistema al no ser necesario que todo el mapa de memoria de un proceso esté en memoria principal, para poder ejecutarlo.
- Que la información que está usando un proceso en un determinado momento (conjunto de trabajo) esté residente en la memoria principal.

Una de las teorías más fuertes concernientes a mejorar el desempeño de las aplicaciones es la del conjunto de trabajo. Si se logra que las páginas o segmentos que contienen al conjunto de trabajo estén siempre en memoria principal, entonces el programa se desempeñará muy bien.

Prácticamente todos los sistemas operativos modernos usan la técnica de memoria virtual; sin embargo, hay que resaltar que la memoria virtual no acelera la ejecución de un programa, sino que lo hará más lento debido a la E/S de información entre la memoria principal y la secundaria.

Al proceso de transferir las páginas de memoria secundaria a memoria principal y viceversa, se le conoce como *intercambio* (*swapping*).

Comúnmente las transferencias de la memoria secundaria a la memoria principal se realizan por demanda. Cuando un proceso necesita acceder a una página que no está en memoria principal (a lo que se denomina fallo de página), el administrador de memoria se encarga de transferirla desde la memoria secundaria. Si al intentar traer la página se detecta que no hay espacio en la memoria principal (no hay marcos libres), será necesario remplazar una página. Al algoritmo para elegir qué página debe ser remplazada, se le denomina política de remplazo y se describirá más adelante en este capítulo.

Además de las políticas de remplazo, los administradores de memoria también utilizan la técnica de prepaginación (precarga). Una técnica de prepaginación común se basa en el principio de localidad de referencia espacial y consiste en que, cuando se produce un fallo

de página, no sólo se lee la página en cuestión, sino también las páginas adyacentes, ya que es posible que el proceso las necesite en un corto plazo. La efectividad de esta técnica depende de si hay acierto en esta predicción. Existen otras técnicas de prepaginación las cuales se describirán más adelante.

2.4 Memoria de reserva

Entre la memoria principal y el microprocesador existe una memoria denominada memoria de reserva, que tiene la característica de ser más rápida que la memoria principal, lo cual permite que el intercambio de información entre ambos dispositivos se realice a mayor velocidad.

Es importante señalar que el término de memoria de reserva que es utilizado en este trabajo de investigación, no es la memoria de reserva que está incrustada en los microprocesadores y que en algunos casos es un chip externo incrustado en la tarjeta madre (Nivel 2), sino al uso de una sección de la memoria principal que se reserva para realizar la precarga informada de los datos.

2.5 Estrategias de acceso a los datos

En un sistema de memoria virtual paginado hay básicamente tres estrategias que definen su funcionamiento:

- Estrategias de búsqueda: Se encargan de obtener la siguiente página de instrucciones o de datos para colocarla en la memoria principal.
- Estrategias de colocación: Se determina el lugar en la memoria en donde se colocarán las páginas, ya sea con instrucciones o datos.

- Estrategias de remplazo: Se decide qué página o páginas remplazar cuando no queda espacio libre en memoria principal para las nuevas páginas [24].

En la Figura 2.6 se observan las tres estrategias de acceso mencionadas anteriormente. A continuación se describen a detalle las estrategias que se utilizan en este trabajo de investigación.

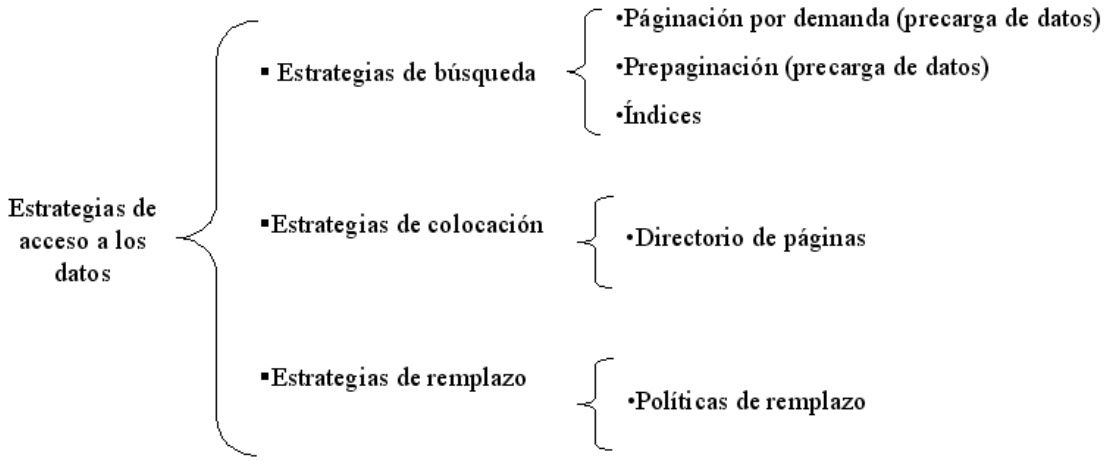

Figura 2.6: Clasificación de las estrategias de acceso a los datos

2.5.1 Estrategias de búsqueda

Precarga de datos

Los datos sólo pueden ser manipulados en memoria principal y por consiguiente toda información requerida debe ser cargada en ésta para su manipulación [33]. Es común que el sistema operativo realice la carga por demanda o reactiva [3], lo cual se debe a que las aplicaciones generan una solicitud de datos al subsistema de Entrada/Salida conforme se van requiriendo. Si los datos no se encuentran en memoria principal entonces se deben leer del disco duro [24].

La precarga es una técnica que ha modificado el esquema anterior. También llamada paginación anticipada o prepaginación, es una técnica que intenta predecir cuáles páginas necesitará un proceso para cargarlas anticipadamente. Si la predicción es acertada, se puede reducir considerablemente el tiempo de ejecución del proceso [1]. Mientras el proceso se ejecuta con sus páginas actuales, el sistema carga nuevas páginas, que estarán disponibles cuando sean requeridas [22].

La precarga se divide comúnmente en precarga informada y precarga predictiva. La precarga informada utiliza el patrón de acceso de las aplicaciones, el cual proporciona información de qué datos serán requeridos en un futuro próximo. La precarga predictiva es aquélla en la que se realiza una predicción de los accesos futuros (basándose en un historial) para precargar los datos.

Para ciertas aplicaciones (como los SMBDs) se puede obtener el patrón de acceso, permitiendo conocer qué datos serán requeridos en un tiempo futuro. Esta característica puede utilizarse para cargar de manera anticipada los datos y se denomina precarga informada, y es la que se utiliza en este trabajo de investigación.

Los principales problemas al tratar de aplicar la precarga son los siguientes: decidir cuándo cargar una página del disco, qué página precargar y cuál remplazar.

Existen cuatro reglas que deben tomarse en cuenta para lograr una buena precarga [3]:

1. *Precarga óptima.-* Cada precarga deberá traer la siguiente página del flujo de referencia, verificando que ésta no se encuentre ya en la reserva. Esta regla ayuda a resolver los problemas de qué página precargar y cuándo precargarla.

2. *Remplazo óptimo.-* Cada precarga deberá descartar la página cuya siguiente referencia es la más lejana en el futuro. Esta regla ayuda a resolver el problema de cuál página remplazar.

3. *No dañar.-* Nunca se debe descartar la página A para precargar la página B cuando A será referida antes que B. Esta regla ayuda a resolver el problema de cuál página remplazar y cuándo realizar la precarga.

4. *Primera oportunidad.-* No debe precargase una página que será remplazada sin haber sido utilizada. Esta regla ayuda a resolver el problema de cuándo y qué página precargar.

Manejador de índices

Un manejador de índices es un software que es invocado por otra aplicación. En este trabajo de investigación la aplicación que lo invoca es un prototipo de SMBD experimental denominado SiMBaDD.

Los mecanismos de indexación son extremadamente importantes para el procesamiento eficiente de consultas en los SMBDs, ya que aceleran el acceso físico a los datos logrando tener un acceso más rápido a éstos [34]. Estos mecanismos ayudan a tener ordenada la información, controlar los accesos a la base de datos y realizar diferentes tipos de búsqueda en forma rápida. Por lo anterior, el uso de un manejador de índices en un SMBD es esencial para reducir el tiempo de las búsquedas y los accesos a la base de datos [35].

Técnicas de árboles aplicadas en manejadores de índices

Los árboles son estructuras jerárquicas aplicadas sobre una colección de elementos u objetos llamados nodos, los cuales son representados generalmente con estructuras no lineales y dinámicas de datos. Existe una gran variedad de técnicas de árboles, y cada una de ellas presenta características diferentes en sus propiedades y desempeño.

El árbol B es una de las estructuras más utilizadas para la organización de índices en un sistema de base de datos. Es una clase especial de árbol equilibrado de m vías que permite recuperar, eliminar e insertar registros de un archivo externo con un buen desempeño [35].

En la Figura 2.7 podemos observar los accesos que tiene un SMBD sin el uso de índices comparado con un SMBD que los utilice.

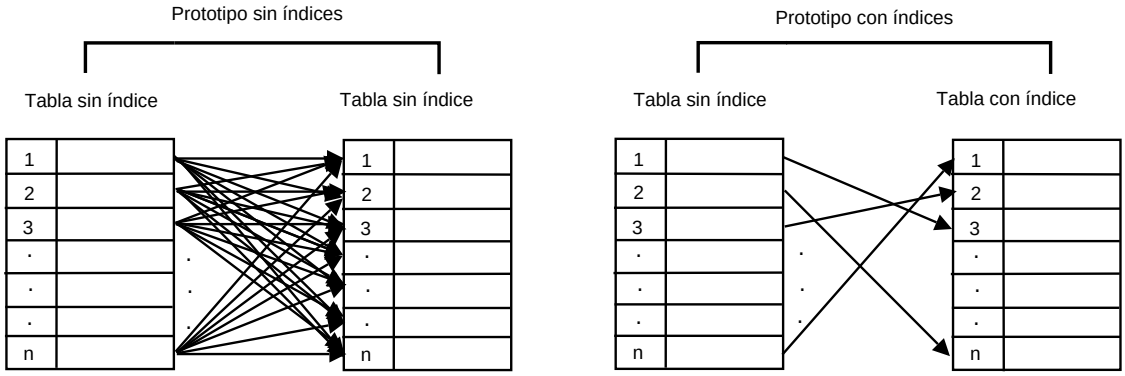

Figura 2.7: Accesos a las tablas en un prototipo sin índice y a otro con índices

Por ejemplo, si tenemos que realizar una consulta a dos tablas sin índices con n renglones, el número de accesos sería de n x n, dando como resultado n^2 accesos a los datos; sin embargo, si utilizamos las mismas dos tablas, pero al menos una de éstas tiene un índice, lo normal es que el número de accesos sea n, y en el peor de los casos n^2, suponiendo que la tabla que está indizada tenga n elementos repetidos, lo cual es poco probable que ocurra. Como se ha visto en la figura anterior, los índices reducen considerablemente el número de accesos innecesarios a la base de datos, y por consiguiente, se obtiene un menor tiempo de respuesta en las consultas realizadas.

2.5.2 Estrategias de colocación

Directorio de páginas

En este trabajo de investigación, el Directorio de Páginas (DP) es una estructura de control que administra las páginas en las cuales se encuentran almacenados los datos de la aplicación que se está ejecutando en ese momento.

La estructura de control permite al DP ordenar las páginas de acuerdo a su número de archivo y número de página. Este orden es el principal; sin embargo, estas mismas páginas almacenadas en el DP, se encuentran también ordenadas mediante una política de remplazo, la cual puede ser una política de remplazo tradicional o una orientada al área de bases de datos, lo cual se define antes de que sea ejecutada una consulta en el SMBD.

Existe un número máximo de páginas que puede contener el DP. Cuando se alcanza este máximo, es necesario remplazar una página (página víctima). Ésta se determina de acuerdo a la política de remplazo que se esté utilizando, y el espacio de memoria es usado para la nueva página solicitada.

2.5.3 Estrategias de remplazo

Políticas de remplazo

Las políticas de remplazo determinan qué página (víctima) es elegida para ser remplazada por otra que se va a cargar ante la ocurrencia de un fallo de página.

Se elige una página víctima cuando:

- No hay marcos de páginas libres.

- Hay marcos de páginas libres, pero el proceso que produjo el fallo no tiene ninguno libre.

El objetivo básico de cualquier algoritmo de remplazo es minimizar la tasa de fallos de páginas.

- **Políticas de remplazo tradicionales**

  Las políticas de remplazo tradicionales son aquéllas que se han utilizado en el área de los SOs. Las políticas de remplazo más utilizadas son:

  - Aleatoria: Remplaza de forma aleatoria.
  - MRU (Most Recently Used / Más Recientemente Utilizada): Remplaza la página más recientemente utilizada.
  - LRU (Least Recently Used / Menos Recientemente Utilizada): Remplaza la página que no ha sido utilizado durante el mayor periodo de tiempo.
  - FIFO (First In - First Out / Primero en entrar - Primero en salir): Remplaza la página que ha permanecido el mayor periodo de tiempo.

  De acuerdo al patrón de acceso a los datos que se tenga, es la política de remplazo que se debe elegir. En la Figura 2.8 podemos observar el comportamiento que tiene un contenedor de *3* páginas utilizando las políticas de remplazo MRU y LRU siguiendo el mismo patrón de acceso. Como se puede apreciar, la política de remplazo apropiada para tal patrón de acceso es la MRU, ya que sólo originó 7 fallos de página, mientras que la política de remplazo LRU presentó 10 fallos de página.

  La selección de la política de remplazo, es uno de los aspectos más importantes que se debe tomar en cuenta para implementar un administrador de memoria, ya que permite disminuir el número de Entradas/Salidas al almacenamiento secundario.

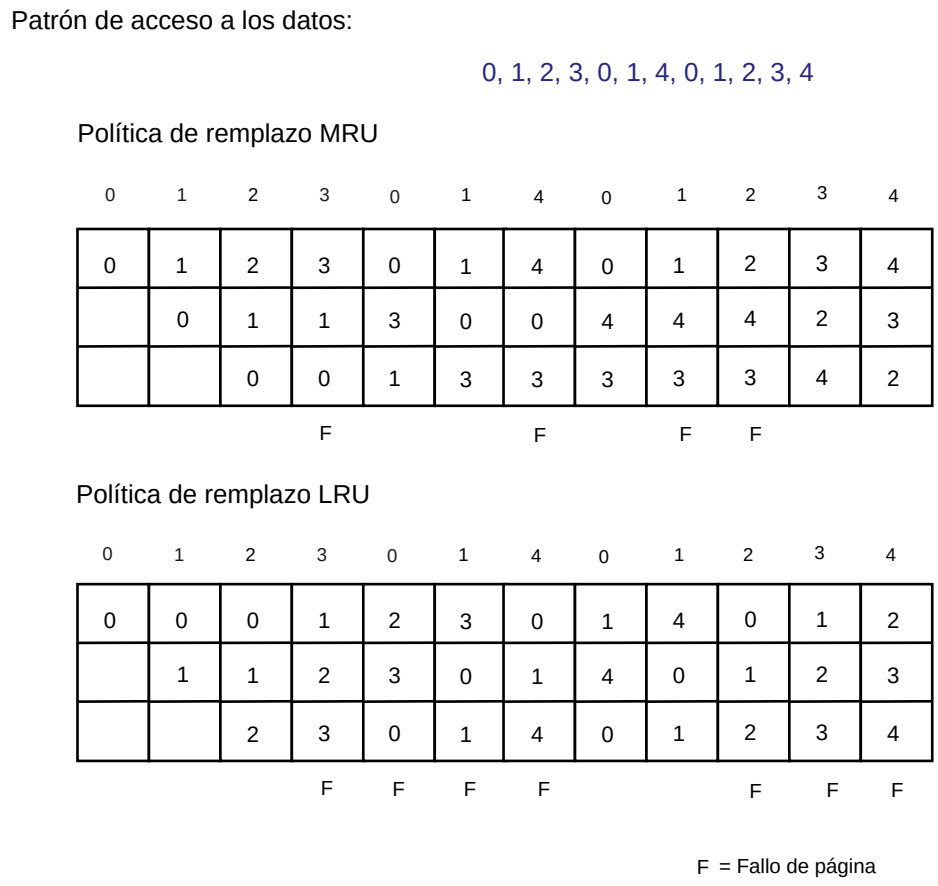

Figura 2.8: Comparación entre las políticas de remplazo MRU y LRU

- **Políticas de remplazo orientadas al área de bases de datos**

  Las políticas de remplazo orientadas al área de bases de datos, han surgido a raíz de la necesidad de políticas más eficientes, ya que en una aplicación como los SMBDs se conoce con anticipación el patrón de acceso a los datos. De esta manera es más sencillo conocer qué datos serán requeridos en un futuro cercano y por lo tanto poder mantenerlos en memoria principal, para disminuir el número de accesos a memoria secundaria.

  Existen diversas políticas orientadas al área de bases de datos (en el AnexoA en [36] se muestra a detalle una investigación de las políticas de remplazo); sin embargo, en este proyecto se selecciona la política 2Q por ser una de las más actuales en esta área.

– 2Q: Esta política de remplazo utiliza un contenedor especial tipo cola llamado área *Ain*, el cual está ordenado de acuerdo a la política de remplazo tradicional FIFO, en donde todas las páginas que incurrieron en un fallo son insertadas en este lugar. Cuando las páginas son eliminadas de este contenedor, éstas son colocadas en otro contenedor tipo cola llamado *Aout* también ordenado por la política de remplazo FIFO. Cuando una página es referida más de una vez y se encuentra en la cola *Aout*, esta página se mueve a un contenedor principal llamado área *Am*, donde se almacenan las páginas que son accedidas frecuentemente. Éste está ordenado por la política de remplazo LRU, ver esquema en la Figura 2.9 [12].

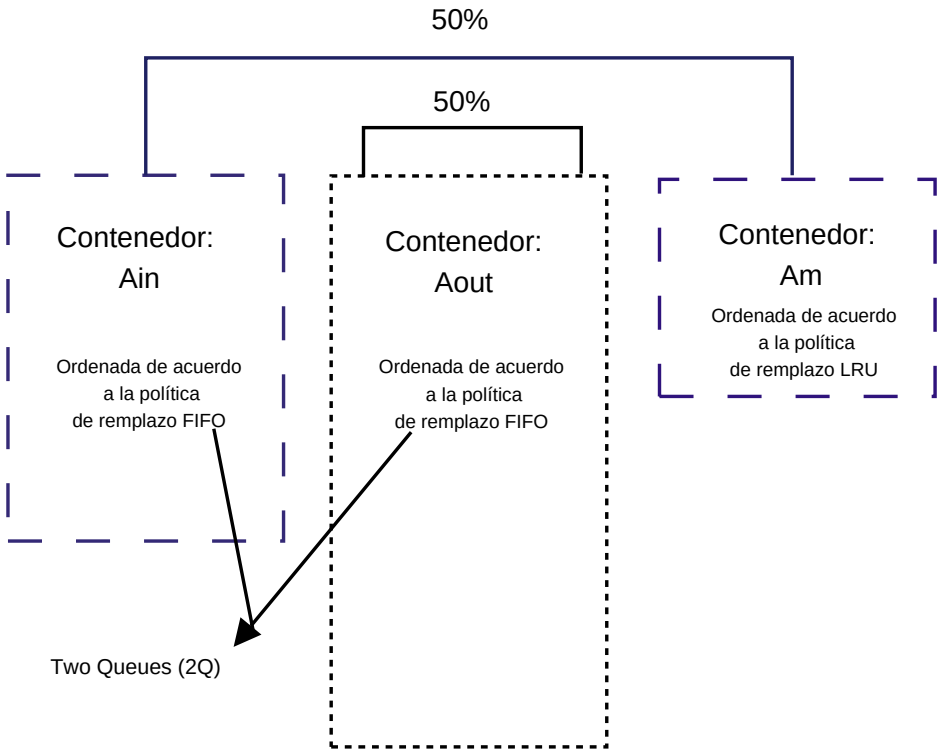

Figura 2.9: Esquema de la política de remplazo 2Q

- **Políticas de remplazo de propósito general**

  Otro tipo de política de remplazo es la de propósito general, la cual, como su nombre lo indica, no se diseñó para alguna aplicación específica. Se decidió

implementar la política LIRS (Low Interreference Recency Set), ya que es una de las más novedosas además de que promete mejorar el desempeño de la política de remplazo 2Q, la cual está orientada al área de bases de datos.

– LIRS: Es una política de remplazo recientemente propuesta, la cual consta de: una cola de tipo LRU denominada HIR, y una lista llamada LIR que es ordenada de acuerdo al número de referencia de las páginas (el número de referencia es el número de accesos que ha tenido la página mientras se encuentra cargada en memoria principal), ésta es llamada LIR. Cuando una página es referida por primera ocasiòn, ésta se inserta en la cola LRU (HIR), si esta misma página es accedida por segunda ocasión y se encuentra en la cola LRU (HIR), se elimina de esta cola y se inserta en la lista LIR acorde con el número de referencia. De manera muy similar a la política de remplazo 2Q, LIRS utiliza una pequeña porción de la memoria de reserva (lista LIR). El porcentaje sugerido por los autores es del 1 % para almacenar las páginas recientemente referidas de la cola HIR, ver esquema en la Figura 2.10 [12].

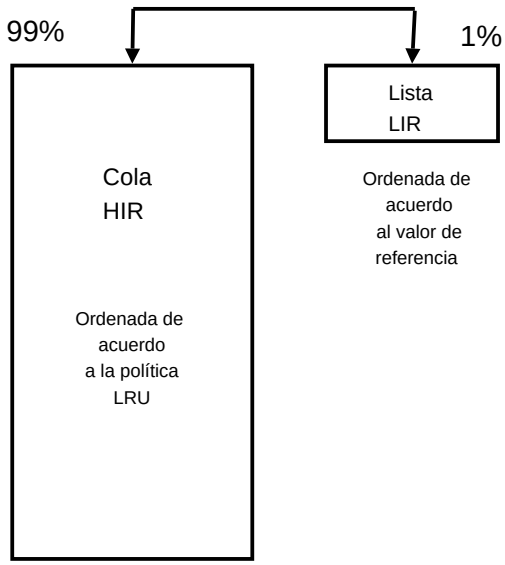

Figura 2.10: Esquema de la política de remplazo LIRS

CAPÍTULO 3

Solución Conceptual

La versión anterior del prototipo experimental de base de datos no utilizaba un manejador de índices que le permitiera acceder a los datos requeridos de una manera eficiente, sólo utilizaba la técnica de precarga informada de datos. De acuerdo a las pruebas que se le realizaron y analizando su arquitectura, se observó que todavía era posible reducir el tiempo de respuesta de las consultas realizadas al prototipo. Por otro lado, existen documentos de reconocidos investigadores [3, 1, 37], los cuales confirman que resulta muy complejo integrar la precarga con políticas de remplazo; sin embargo, esta integración da como resultado mejores tiempos de respuesta en las consultas realizadas a los SMBDs.

Para integrar las técnicas antes mencionadas se diseñó e implementó un directorio de páginas, el cual permite almacenar datos en memoria principal, donde los datos son administrados mediante políticas de remplazo tradicionales y políticas de remplazo orientadas al área de bases de datos. Además se realizaron algunas modificaciones en lo que a la administración del uso de precarga se refiere, ya que como se mencionó anteriormente, la precarga ya se encontraba implementada en la versión previa del prototipo experimental (SiMBaDD). Por otra parte, la integración del manejador de índices tuvo un papel muy importante en este trabajo de investigación, ya que con él

fue posible reducir significativamente los tiempos de respuesta de consultas realizadas a dos o más tablas de gran tamaño.

3.1 Arquitectura anterior del prototipo

La arquitectura anterior del prototipo experimental incluye la precarga informada y el administrador de contenedores (ver detalles en [36]). La Figura 3.1 muestra esta arquitectura, donde sólo la sección punteada es la que se modifica en este proyecto de investigación.

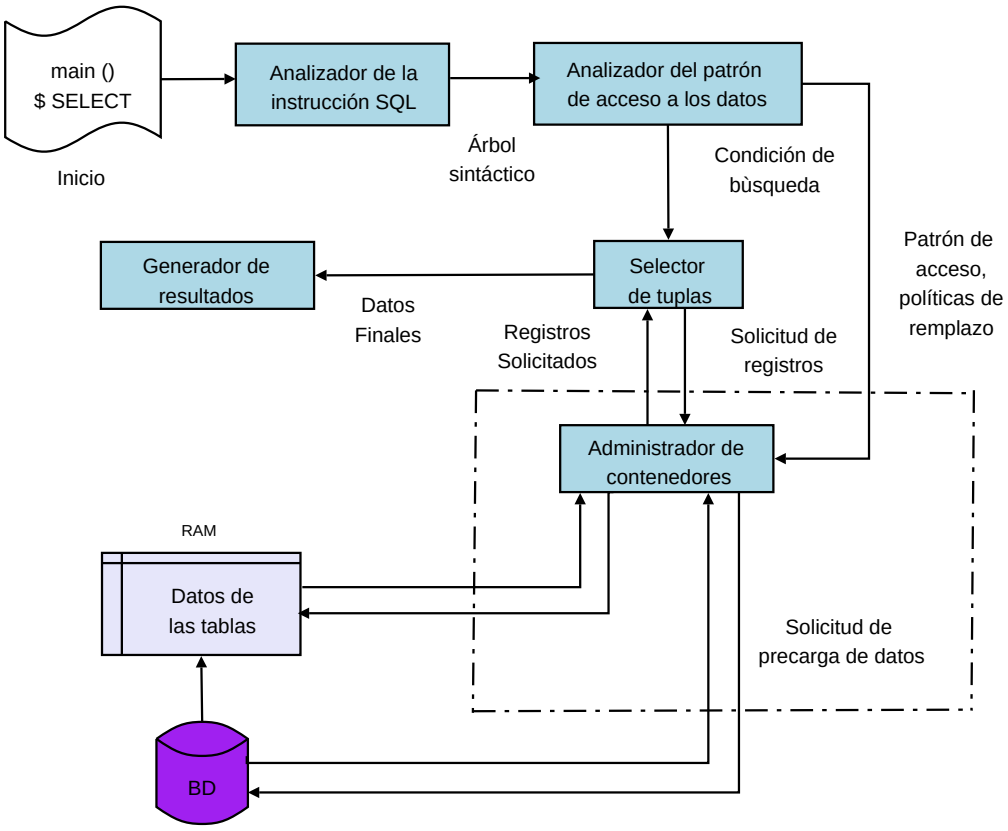

Figura 3.1: Arquitectura anterior del prototipo

3.2 Modificación a la arquitectura anterior del prototipo

La modificación consiste en un directorio de páginas, el cual utiliza diversas políticas de remplazo con el fin de que la mayor cantidad de datos solicitados se encuentren cargados en la memoria principal. La Figura 3.2 muestra la arquitectura propuesta donde se observan las técnicas de precarga informada y políticas de remplazo (directorio de páginas) ya integradas al prototipo experimental, también se observa la interacción que tienen entre la aplicación (módulo *Selector de tuplas*), memoria principal (RAM) y memoria secundaria (base de datos). En este trabajo de investigación, el directorio de páginas está cargado en memoria principal, de tal manera que cuando el *Selector de tuplas* hace una solicitud, se busca el(los) dato(s) en memoria principal, si no se encuentra(n) el(los) dato(s), se realiza una carga por demanda, la cual se pretende evitar anticipándose a la solicitud de datos y precargando estos a la memoria principal directamente o al directorio de páginas antes de ser solicitados.

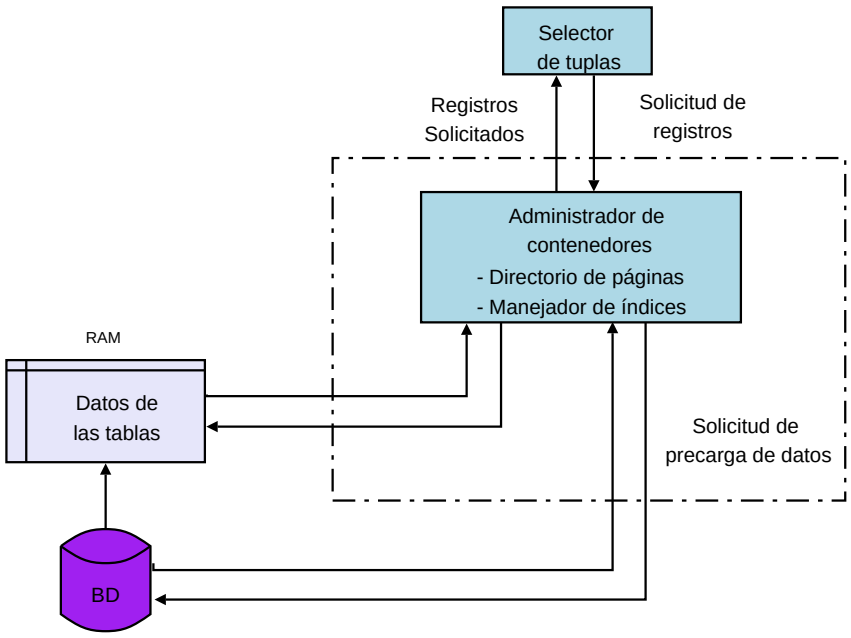

Figura 3.2: Arquitectura propuesta del prototipo

Este capítulo describe a detalle el diseño del directorio de páginas así como las políticas de remplazo utilizadas.

3.3 Análisis y diseño del uso de índices

Además de la integración de la precarga informada y las políticas de remplazo en la arquitectura propuesta, en este trabajo de investigación se integra un manejador de índices [34], donde una tabla puede tener uno o más índices, aunque sólo puede ser utilizado uno de ellos. Los índices se clasifican comúnmente en simples y compuestos, donde:

- Índice simple: es aquél que está formado por el valor de una columna de tipo entero o de tipo cadena.
- Índice compuesto: es aquél que está formado por los valores de más de una columna de tipo entero o de tipo cadena.

Dada la complejidad del uso de un manejador de índices, este proyecto de investigación limita el uso de los índices solamente a las operaciones de consulta. Los índices se usarán siempre y cuando:

1. La consulta tenga cláusula WHERE.
2. La cláusula WHERE no tenga operadores OR.

En las consultas realizadas a una sola tabla, para determinar si un índice es útil o no, se toman en cuenta los siguientes puntos:

- Si la tabla tiene índice simple, se verifica que la columna que conforma el índice sea la misma columna que participa en la cláusula WHERE.

- En el caso de que la tabla tenga un índice compuesto, se verifica que todas las columnas que conforman el índice estén participando en la cláusula WHERE, de lo contrario ese índice compuesto no es útil.

Además de los dos puntos mencionados anteriormente, en el caso de las consultas con cláusula WHERE realizadas a dos o más tablas, es necesario considerar que:

- De acuerdo al análisis realizado, cuando no existe un índice útil, es conveniente crearle un índice a una de las tablas, ya que es menor el tiempo que se consume en crear un índice y realizar la consulta, a realizar un producto cartesiano entre las tablas que conforman la consulta.

Las Figuras 3.3 y 3.4 muestran el pseudocódigo y el diagrama de flujo de una consulta a una sola tabla cuando esta hace uso del manejador de índices. El procedimiento es el siguiente: se debe verificar que la cláusula WHERE se compone solamente de operadores AND, se valida que el índice sea útil, una vez validado se abre el índice y se determina si es simple o compuesto, posteriormente se buscan los elementos que cumplen con la condición de la cláusula WHERE que utiliza el índice, en caso de que se haya encontrado algún elemento, se obtiene la posición del renglón en la tabla, se lee de forma directa el renglón y se valida que se cumpla con todas las condiciones de la cláusula WHERE, si es así, se almacena en un archivo temporal el cual será procesado en un tiempo posterior.

```
Inicio
    Si la cláusula WHERE sólo contiene operadores AND
        Si la tabla tiene índice útil
            Abrir el índice
            Si el índice es compuesto
                 Concatenar los valores de las columnas que participan en el índice
            De lo contrario
                Asignar a una variable, el valor que se buscará en el índice
            Buscar en el índice los elementos que cumplan con la condición
            Mientras se encuentre el elemento en el índice
                Obtener la posición del renglón en la tabla
                Leer el renglón de forma directa
                Si el total de las columnas de la cláusula WHERE conforman el índice
                    Guardar el renglón en un archivo temporal
                De lo contrario
                    Si el renglón cumple con la cláusula WHERE
                        Guardar el renglón en un archivo temporal
                Buscar el siguiente elemento
        De lo contrario
            No se utilizan los índices
    De lo contrario
        No se utilizan los índices
Fin
```

Figura 3.3: Pseudocódigo del uso de índice en una consulta a una tabla.

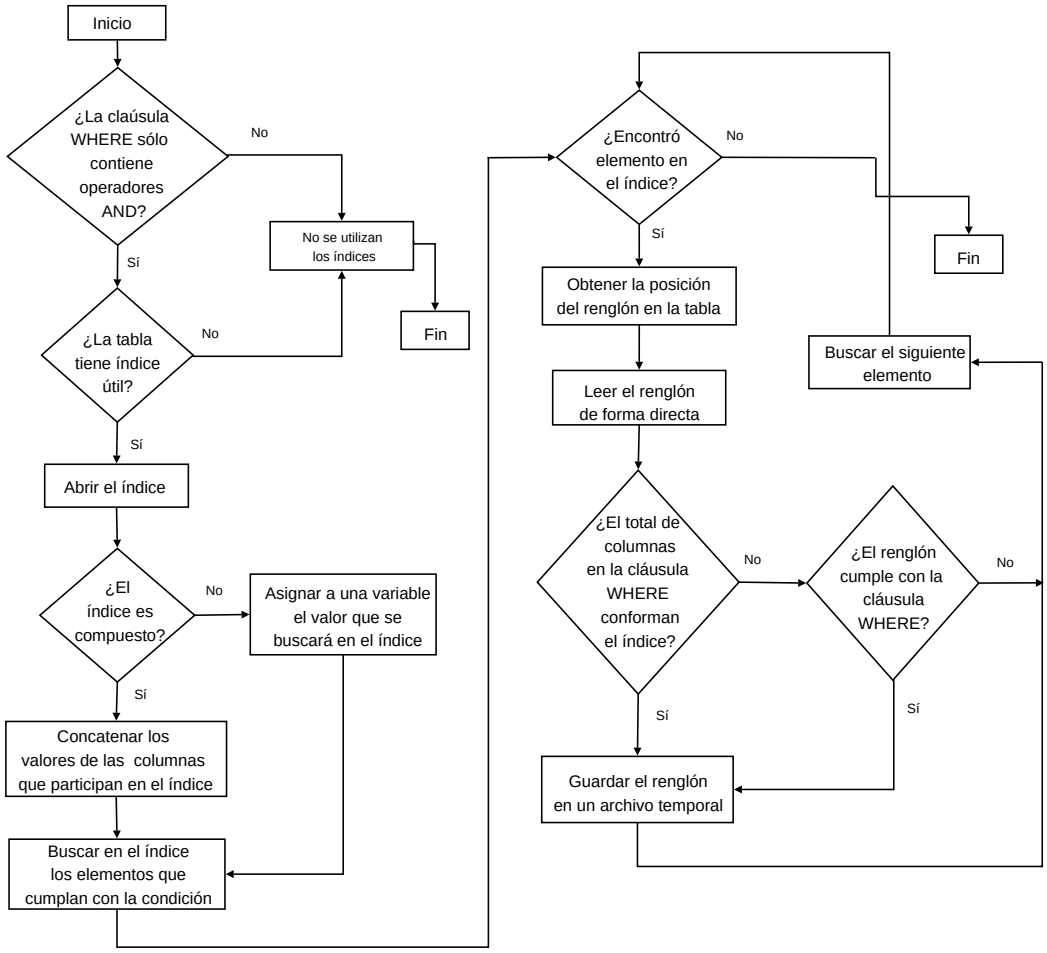

Figura 3.4: Índice con una tabla

La Figura 3.5 muestra el pseudocódigo de una consulta a dos tablas, en éste se detallan los pasos a seguir para el uso de índices. A continuación se explica de forma breve las consideraciones a tomar en cuenta para éste procedimiento: en las consultas a dos tablas es recomendable el uso de índices, de tal manera que si de las dos tablas que intervienen en la consulta ninguna tiene índice útil se creará un índice a una de ellas, una vez teniendo un índice útil, éste se abre y la tabla que no tiene índice se lee de manera secuencial, mientras que la tabla que tiene índice se lee de manera aleatoria, esto último dependerá del tipo de índice que se utilizará (simple o compuesto) y de si la condición de la cláusula WHERE que utiliza el índice es comparada con una literal o una columna.

```
Inicio
Si la cláusula WHERE solo contiene operadores AND
    Si las tablas tienen índice útil
        Seleccionar el índice a utilizar
    De lo contrario
        Seleccionar la tabla a la que se le creara el índice
        Crear el índice
    Abrir el índice
    Si la columna del índice se compara con el valor de una columna
        Mientras existan renglones sin leer de la tabla que no tiene índice
            Leer un renglón de la tabla que no tiene índice
            Si el índice es compuesto
                Extraer el renglón y concatenar los valores de las columnas que componen el índice
            De lo contrario
                Extraer del renglón el valor de la columna que compone al índice
            Asignar el(los) valor(es) a una variable temporal
            Buscar en el índice el o los elementos iguales a la variable temporal
            Si encontró el elemento en el índice
                Obtener la posición del renglón en la tabla
                Leer el renglón de forma directa
                Si el total de columnas de la cláusula WHERE conforman el índice
                    Guardar el renglón en un archivo temporal
                De lo contrario
                    Si se cumple con la cláusula WHERE
                        Guardar el renglón en un archivo temporal
    De lo contrario, (se compara con una literal)
        Si el índice es compuesto
            Concatenar los valores de la literal
        Asignar el valor de la literal a una variable temporal
        Buscar en el índice los elementos que cumplan con el valor de la variable temporal
          Si encontró el elemento en el índice
            Obtener la posición del renglón en la tabla
            Leer de forma directa el renglón de la tabla
            Mientras existan renglones sin leer de la tabla que no tiene índice
                    Leer de forma secuencial, el siguiente renglón de la tabla que no tiene índice
                    Si el total de columnas de la cláusula WHERE conforman el índice
                        Guardar el renglón en un archivo temporal
                    Si se cumple con la cláusula WHERE
                        Almacenar en archivo temporal, el producto cartesiano resultante
                    Si cumple con la(s) condicion(es) restante(s) de la cláusula WHERE
                        Almacenar en archivo temporal, el producto cartesiano resultante
De lo contrario
    No se utilizan los índices
Fin
```

Figura 3.5: Pseudocódigo del uso de índice en una consulta a dos tablas.

3.4 Directorio de páginas

Como hemos visto en el capítulo anterior, al encontrarse la información en memoria principal se disminuyen las Entradas/Salidas a disco. La finalidad de un directorio de páginas es mantener las páginas requeridas en memoria principal. Cuando es insuficiente el tamaño de la memoria principal para almacenar las páginas de la(s) tabla(s) que utiliza una consulta realizada al prototipo, es necesario realizar un remplazo eficiente de las páginas que ya no se utilizan por las páginas que son requeridas.

El DP es una estructura que permite tener en memoria principal una cantidad determinada de páginas, y se ordena de acuerdo al número de tabla y número de página.

Cuando un renglón es solicitado, se le asigna un número de tabla y un número de página. El número de tabla es el número que se le asigna a la(s) tabla(s) que participa(n) en una consulta, mientras que el número de página se obtiene dividiendo el tamaño de la página entre el tamaño del renglón que ha solicitado. El tamaño de una página es acorde al valor que se le asigne en el archivo de configuración (este archivo contiene los valores de los parámetros utilizados en la ejecución del prototipo). Esta asignación se debe realizar antes de que inicie la ejecución del prototipo.

Una página puede definirse de manera gráfica como la sección sombreada de la Figura 3.6, la cual muestra el inicio y el fin lógico de la página número uno de una tabla.

Las páginas se almacenan en el DP mediante una estructura de control, la cual permite enlazarlas de forma ordenada, en el siguiente capítulo se describe a detalle esta estructura.

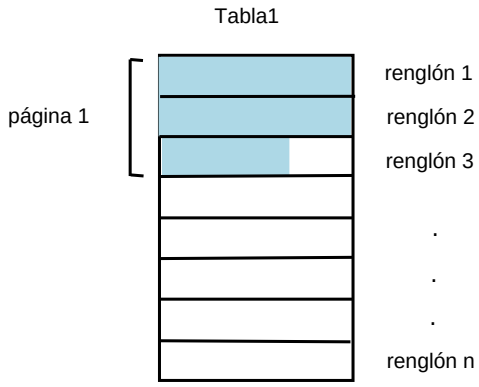

Figura 3.6: Ejemplo de una página en una tabla

3.4.1 Inserción de páginas en el DP

En el momento en que el prototipo recibe una consulta, los datos son solicitados por número de renglón, de tal forma que cuando un renglón es requerido éste es asociado al número de archivo y número de página al que pertenece. De esta manera se sabe qué página se debe buscar en el DP. Si ocurre un fallo de página (la página no se encuentra en memoria principal), ésta se debe insertar en el DP (en forma ordenada). A continuación se muestra el patrón de acceso para la solicitud de ciertas páginas. La Figura 3.7 muestra de forma gráfica la manera en cómo éstas son insertadas en el DP.

Patrón de acceso:

1,5	2,2	1,2	1,5	1,1

Como se observa en la Figura 3.7, la página *1,5*, fue requerida dos veces (incisos *a* y *d*), de tal forma que en la segunda ocasión que se solicita ya se encuentra cargada en el DP. De esta manera no hay que insertar la página *1,5*, disminuyendo así el número de Entradas/Salidas que se realizan a la memoria secundaria. Como se observa, la manera en la que se encuentran las páginas, es bastante sencilla y fácil de manipular (número de archivo y número de página). Sin embargo, también existe otro orden que nos permitirá remplazar una página que se encuentra cargada en el DP. La manera

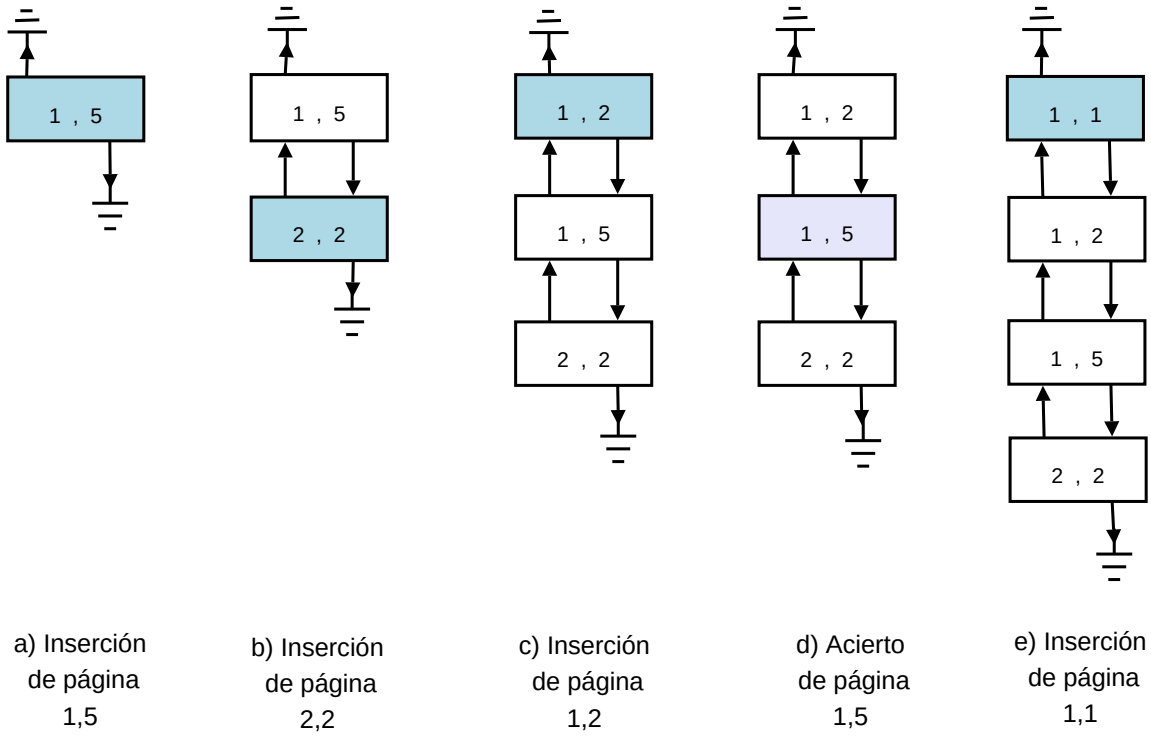

Figura 3.7: Inserción de páginas en el directorio de páginas (DP)

en cómo se determina qué página será la víctima depende de la política de remplazo asociada a la consulta.

3.4.2 Remplazo de páginas en el DP

El remplazo de páginas se presenta cuando el DP se encuentra a su máxima capacidad, y ya no es posible continuar insertando más páginas. Para saber exactamente qué página es conveniente remplazar, se hace uso de alguna política de remplazo, la cual determina qué página es la víctima; la política se determina en el archivo de configuración.

En el siguiente ejemplo se elige la página víctima y por ende la que será remplazada, de acuerdo a la política de remplazo MRU (Más Recientemente Utilizada). Esta política consiste en mantener una lista de páginas, donde se inserta al inicio la página que se esté

utilizando actualmente. Observemos que se tiene un patrón de acceso de 5 elementos y el máximo número de páginas que puede contener el DP es de 4 páginas.

En la Figura 3.8, en los incisos *a-d*, se insertan las páginas solicitadas y se colocan siempre al inicio de la lista; mientras que en el inciso *e* se ejemplifica el caso en el que el DP se encuentra a su máxima capacidad y se debe remplazar una página. La página que será remplazada es la página *2,1* y se sustituye por la página *1,1*.

Patrón de acceso:

1,5	2,2	1,2	2,1	1,1

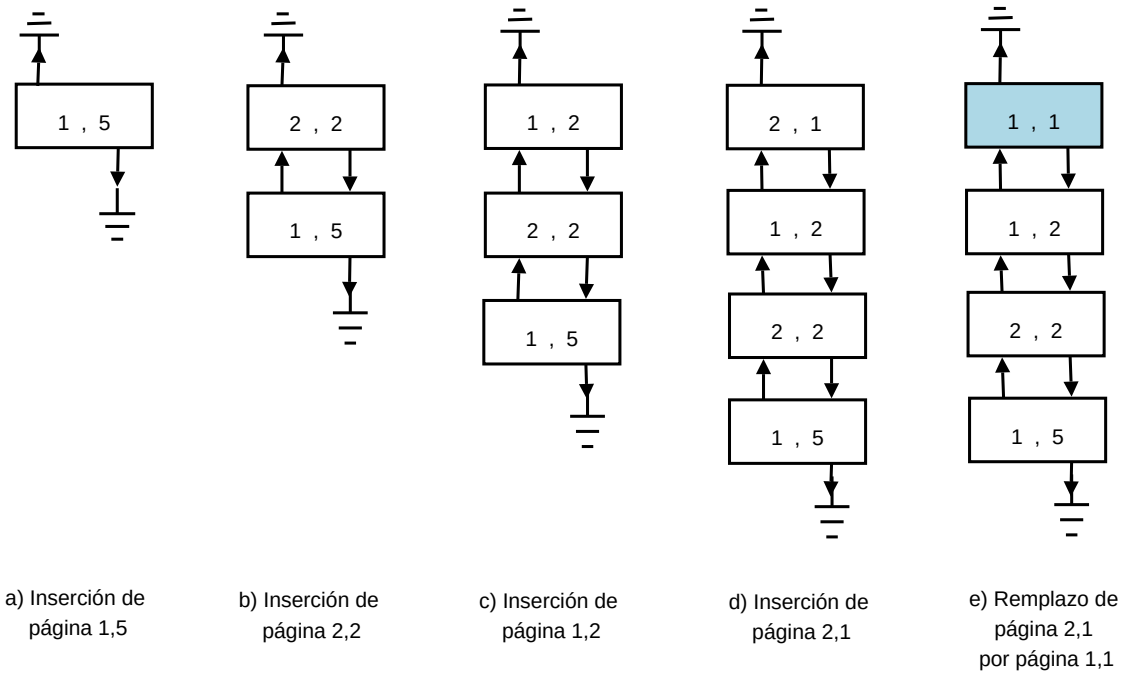

Figura 3.8: Remplazo de una página en el DP

Como hemos visto de forma separada, existen 2 maneras de ordenar el DP. En la Figura 3.7 se mostró el orden de acuerdo al número de archivo y número de página; mientras que en la Figura 3.8 se mostró el orden de acuerdo a la política de remplazo seleccionada. Sin embargo, si se observa el DP con los 2 órdenes tenemos la Figura 3.9, la cual tiene los 4 apuntadores mostrados anteriormente en la Tabla 4.3.

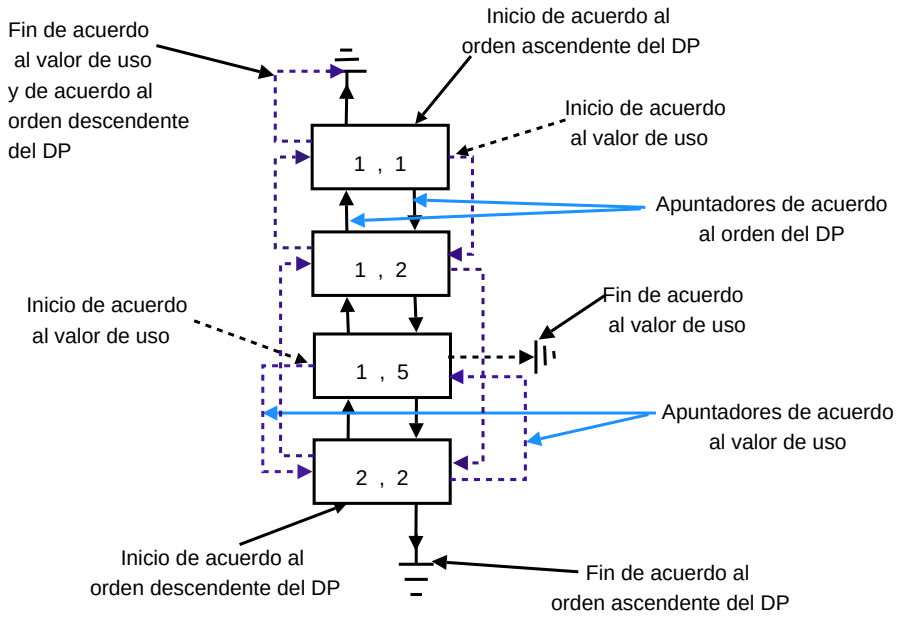

Figura 3.9: Directorio de páginas ordenado de dos formas

3.5 Políticas de remplazo

Cuando se insertan y remplazan páginas de acuerdo a las diferentes políticas de remplazo, se utilizan una o varias listas, donde los apuntadores de la estructura de control son los encargados de enlazar las páginas, ver Tabla 4.3 (apuntadores de acuerdo al valor de uso).

3.5.1 Políticas de remplazo tradicionales

Las políticas de remplazo tradicionales que se decidió implementar en este trabajo de investigación son:

- MRU (Most Recently Used/Más Recientemente Utilizada).
- LRU (Least Recently Used/Menos Recientemente Utilizada).

Cuando una página es requerida, primeramente se verifica si se encuentra en el DP. Si la página no se encuentra, se debe insertar, en caso contrario solamente se actualiza su posición de acuerdo a la política de remplazo que se esté utilizando en ese momento.

Las consideraciones que se tomaron en cuenta para trabajar con la política de remplazo MRU son:

- Buscar la página solicitada en memoria principal. Si no se encuentra, se debe insertar la página, si se encuentra, se debe actualizar su posición de la siguiente manera:
  - Obtener la posición de la página.
  - Mover la página solicitada al inicio de la lista.
- Insertar la página siempre al inicio de la lista.
- Cuando se llega al máximo número de páginas del DP y es necesario insertar una nueva página, se debe remplazar una página del DP. El criterio a seguir es remplazar la página que se encuentra al inicio de la lista por la página nueva. Los pasos que se deben seguir cuando sucede esta situación son los siguientes:
  - Eliminar la página que se encuentra al inicio de la lista.
  - Insertar al inicio de la lista la nueva página.

Las consideraciones a tomar para la política de remplazo LRU son:

1. Buscar si la página solicitada se encuentra cargada en memoria principal. Si no se encuentra, se debe insertar la página, si se encuentra, se debe actualizar su posición de la siguiente manera:
   - Buscar la página solicitada en la lista y obtener la posición.

- Mover la página solicitada al final de la lista.

2. La inserción de una página siempre es al final de la lista.

3. Cuando se llega al máximo número de páginas del DP, se le debe remplazar una. El criterio a seguir es remplazar la página que se encuentra al inicio de la lista, e insertar la nueva página al final de la lista. Los pasos que se deben seguir para el remplazo son los siguientes:

   - Eliminar la página que se encuentra al inicio de la lista.
   - Insertar al final de la lista la nueva página.

3.5.2 Políticas de remplazo orientadas al área de base de datos y de propósito general

Las políticas de remplazo orientadas al área de base de datos y de propósito general que se decidió implementar en este trabajo de investigación son:

- 2Q.
- LIRS.
- LLU (política de remplazo propuesta).

En las políticas de remplazo 2Q y LIRS, al insertar una nueva página, en ocasiones es necesario remplazar una, debido a que estas políticas en particular manejan más de una lista.

Política de remplazo 2Q

La política de remplazo 2Q está conformada por tres áreas denominadas *Ain*, *Aout* y *Am*, las cuales se administran mediante dos colas FIFO (*Ain, Aout*) y una cola LRU (*Am*), donde la suma de los porcentajes de las áreas *Ain* y *Am* debe ser del 50 % del total de páginas contenidas en el DP, sin importar cuál de las dos es más grande o más pequeña; mientras que el área *Aout* debe tener el 50 % restante. La Figura 3.10 muestra de manera gráfica las 3 áreas de esta política de remplazo.

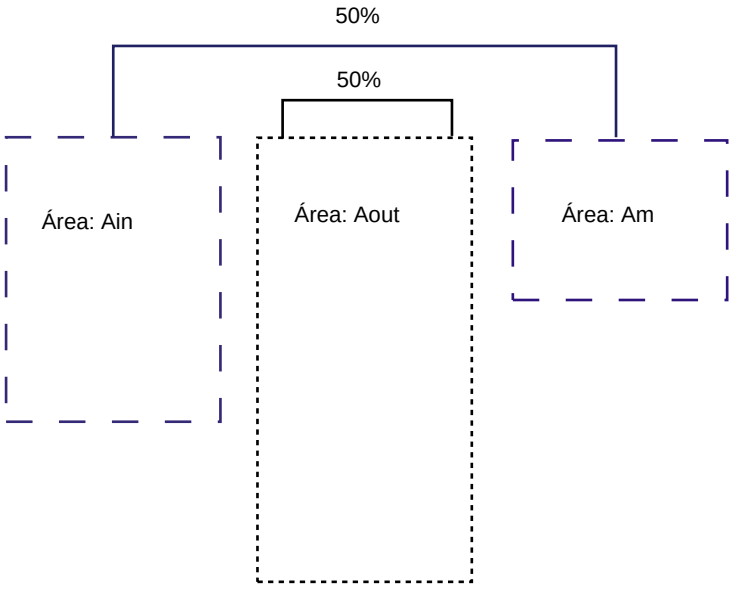

Figura 3.10: Áreas de la política de remplazo 2Q

Cuando una página es requerida, ésta es buscada inicialmente en el área *Ain*, si no se encuentra se continua la búsqueda en el área *Aout*, y si tampoco se encuentra ahí, se busca finalmente en el área *Am*. Si la página no es encontrada en ninguna de las tres áreas, se debe insertar en el área *Ain*. En las Figuras 3.11, 3.12 se muestra el pseudocódigo y el diagrama de flujo, las cuales contienen los pasos que se deben seguir para insertar y remplazar una página.

```
Inicio
Si existe espacio para insertar una página en Ain
    Insertar la nueva página en el inicio de Ain
De lo contrario
    Eliminar la última página de Ain
    Si no existe espacio para insertar en Aout
        Eliminar la última página de Aout
    Insertar la página eliminada de Ain en Aout
    Insertar la nueva página en Ain
Fin
```

Figura 3.11: Pseudocódigo de la inserción y remplazo de una página en la política de remplazo 2Q.

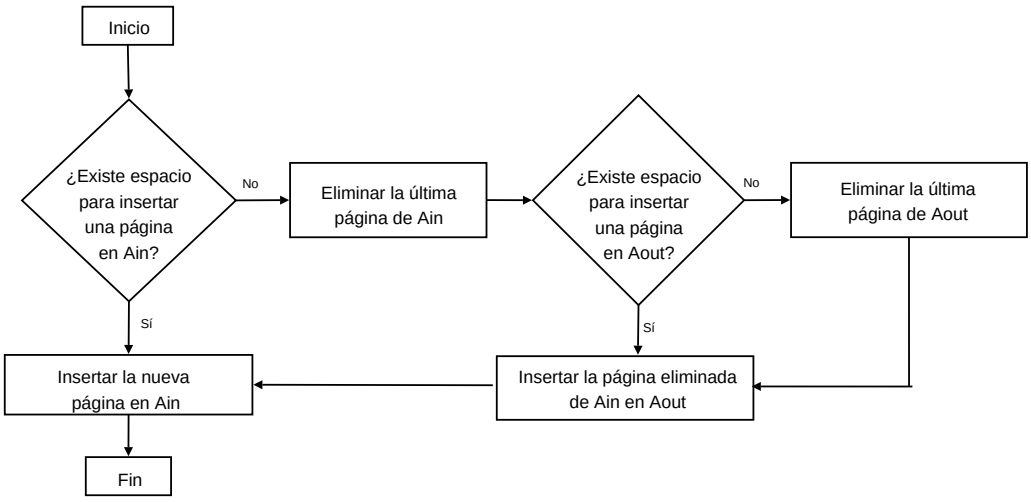

Figura 3.12: Inserción y remplazo de una página en la política de remplazo 2Q

Cuando se busca una página y ésta ya se encuentra cargada en memoria principal, se debe actualizar su posición. Las Figuras 3.13, 3.14 muestran el pseudocódigo y el diagrama de flujo donde se observa de manera sencilla cómo se actualiza la posición de la página, los pasos consisten en buscar la página en las tres áreas, primero se busca en el área *Ain*, si la página es encontrada, no se realizan modificaciones, si no se encuentra se busca en el área *Aout*, si ésta es encontrada, se elimina la página y se inserta en el área *Am*, en el caso de que la página buscada no se encuentre en el área *Aout*, ésta se busca en el área *Am*, si se encuentra la página, se actualiza su posición.

```
Inicio
Buscar la página en Ain
Si encontró la página en Ain
    No se realizan modificaciones
De lo contrario
    Si se encontró la página en Aout
        Eliminar la página de Aout
        Si existe espacio para insertar una página en Am
            Insertar la página en Am
        De lo contrario
            Eliminar la página que se encuentra al inicio de Am
            Insertar la página en la cola Am
    De lo contrario
        Buscar la página en la cola Am
        Si se encontró la página en la cola Am
            Actualizar su posición en la cola Am
        De lo  contrario
            Error
Fin
```

Figura 3.13: Pseudocódigo de la actualización de página en la política de remplazo 2Q.

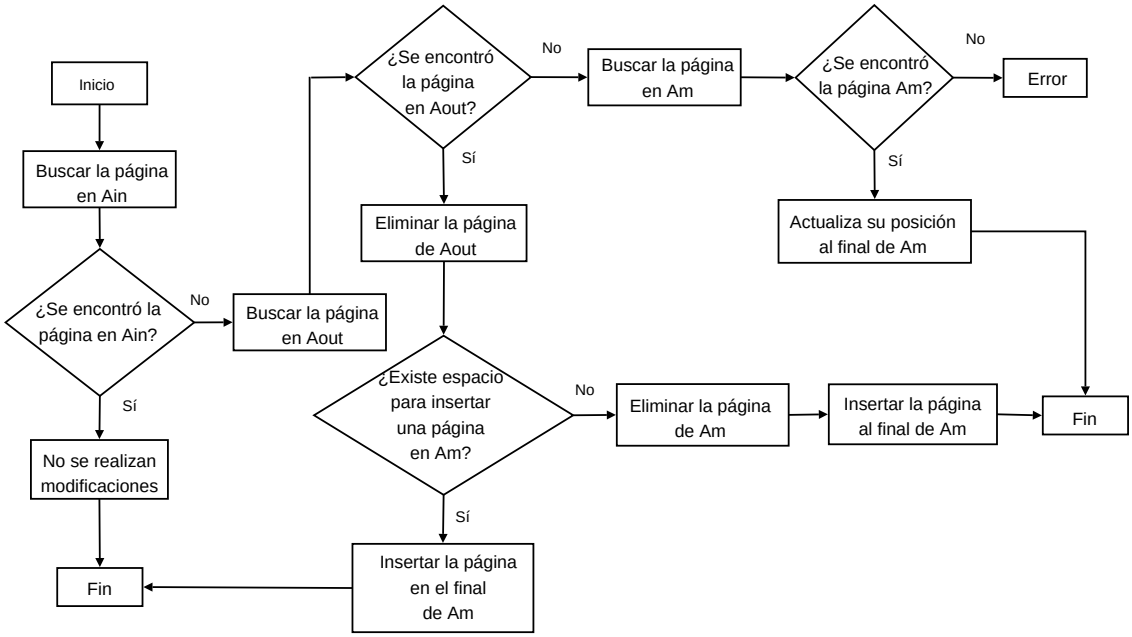

Figura 3.14: Actualización de página en la política de remplazo 2Q

Política de remplazo LIRS

En la política de remplazo LIRS se tienen dos listas, las cuales se denominan LIR y HIR, donde el tamaño de LIR es un porcentaje que se determina de acuerdo al valor que se le asigna al parámetro *TamLirs* del archivo de configuración mostrado en la Figura 4.1; mientras que el tamaño de HIR es el porcentaje restante.

Es importante mencionar que ambas listas se encuentran ordenadas mediante la política de remplazo LRU (Menos Recientemente Utilizada). Enseguida se mencionan algunas consideraciones que se deben tomar en cuenta para el uso de esta política:

- Cuando una página es insertada, se asigna su número de referencia en 1.
- La inserción de una página nueva siempre se realiza en la lista HIR.
- Cuando alguna de las dos listas llega a su máxima capacidad, se debe eliminar la página que se encuentra al inicio de la lista.
- Se debe llevar el control del mínimo número de referencia de las páginas de la lista LIR, lo cual se realiza de la siguiente manera: se busca qué página de la lista LIR tiene el número de referencia menor, y este valor es asignado a una variable.
- Cuando en la lista HIR existe una página con un número de referencia mayor que el mínimo número de referencia de la lista LIR, la página se debe eliminar de la lista HIR e insertar al inicio de la lista LIR.

En las Figuras 3.15, 3.16 se muestra el pseudocódigo y el diagrama de flujo donde se detallan los pasos a seguir cuando se inserta y se remplaza una página en la lista HIR utilizando la política de remplazo LIRS.

```
Inicio
Asignar el número uno a la referencia de la nueva página
Si no existe espacio para asignar la nueva página en la lista HIR
    Se elimina la página que se encuentra al final de la lista HIR
Insertar la nueva página al final de la lista HIR
Fin
```

Figura 3.15: Pseudocódigo de la inserción y remplazo de página en la política de remplazo LIRS.

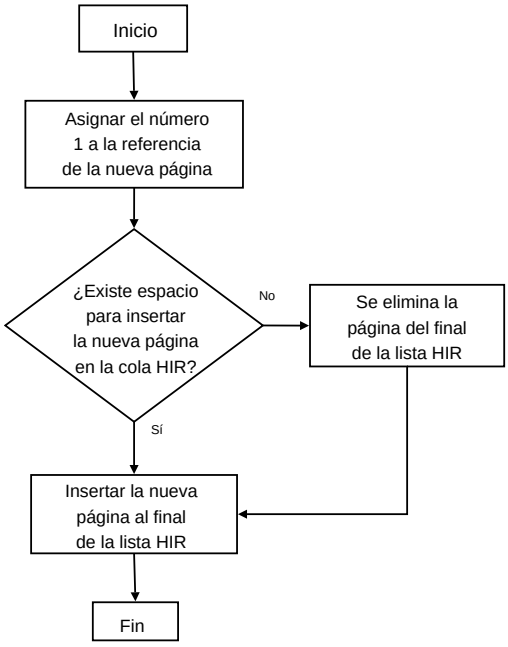

Figura 3.16: Inserción y remplazo de página en la política de remplazo LIRS

Las Figuras 3.17, 3.18 muestran de forma detallada los pasos que se deben seguir para realizar una actualización en la política de remplazo LIRS. De acuerdo al algoritmo de ésta política de ramplazo (ver 2.5.3) cuando una página es solicitada y ya se encuentra en memoria principal se debe actualizar su posición en la lista correspondiente, lo cual puede ocasionar con ello un remplazo de página, ya que una página que se encuentra en la lista HIR puede ser eliminada e insertada en la lista LIR, siempre y cuando tenga un número de referencia mayor que el mínimo número de referencia de la lista LIR.

```
Inicio
    Buscar la página en la lista LIR
    Si se encontró la página en la lista LIR
        Incrementa el número de referencia de la página y actualiza su ubicación
    De lo contrario
        Busca la página en la lista HIR
        Si se encontró la página en la lista HIR
            Actualiza el número de referencia de la página y actualiza su ubicación
            Si el número de referencia de la página es mayor que el mínimo número de referencia
            de la lista LIR
                Elimina la página de la lista HIR
                Si no existe espacio para insertar la página eliminada de HIR en la lista LIR
                    Eliminar la página del final de la lista LIR
                Insertar la página al final de la lista LIR
Fin
```

Figura 3.17: Pseudocódigo de la actualización de página en la política de remplazo LIRS.

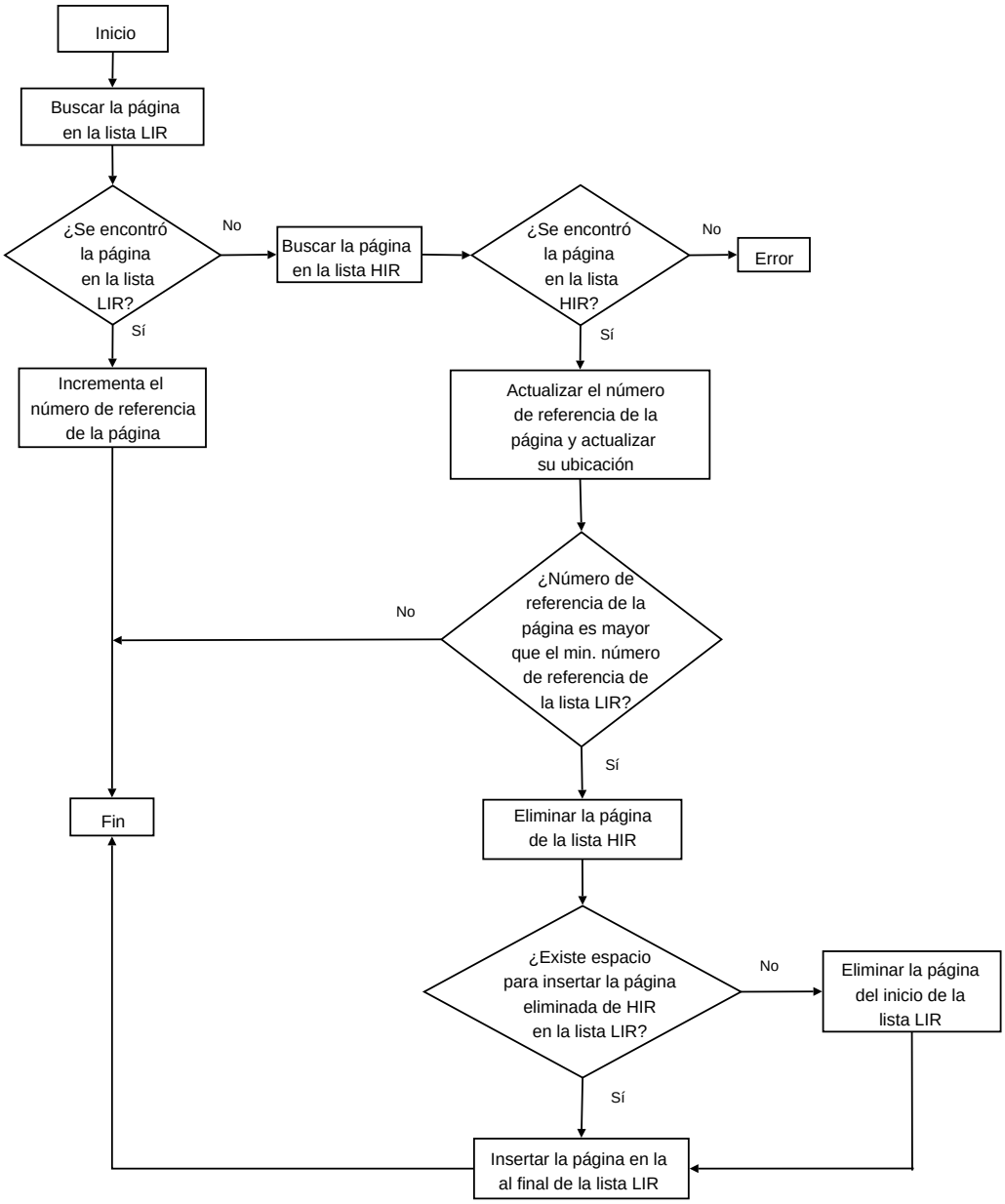

Figura 3.18: Actualización de página en la política de remplazo LIRS

Política de remplazo LLU

Esta es la política de remplazo que se propone y consiste básicamente en tener diferentes listas ordenadas acorde con el valor de uso asignado a cada página que se encuentre en el DP. Como ejemplo podemos observar la Figura 3.19, donde existen 5 páginas en el DP ordenadas por el número de archivo y número de página; sin embargo, esas mismas páginas tienen otro orden considerando el valor de uso que se les asigno. También es conveniente señalar que las listas de valores de uso están ordenadas mediante las políticas de remplazo tradicionales MRU o LRU descritas anteriormente. De esta manera definimos como política de remplazo global la LLU y política de remplazo local la MRU o LRU, originando así 2 combinaciones para la política de remplazo propuesta:

- LLU+MRU
- LLU+LRU

La manera en cómo se determina qué política de remplazo tanto global como local se debe utilizar en las listas de valor de uso, es mediante el archivo de configuración de la Figura 4.1, donde los parámetros *PolRempGlobal* y *PolRempLocal* indican los valores de las políticas a utilizar para la consulta que se esté ejecutando en ese momento.

En el inciso *a* del ejemplo de la Figura 3.19 se observa un DP con 5 páginas insertadas, mientras que los incisos *b* y *c* muestran dos listas de páginas que tienen asignados los valores de uso *0* y *5* respectivamente; además, ambas listas se encuentran ordenadas de forma local por la política LRU.

La asignación de valores de uso a las páginas se realiza tomando en cuenta los casos de la taxonomía de la instrucción SELECT, quedando de la siguiente manera:

Caso 1. Consulta a sólo una tabla, sin cláusula WHERE sin índice y con índice. La tabla base es la misma que la tabla de resultados, ya que se seleccionarán todos los

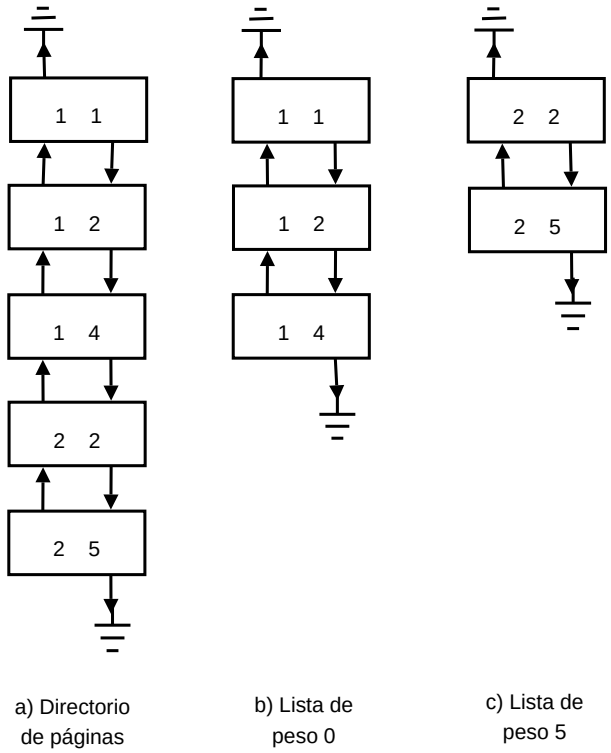

Figura 3.19: Directorio de páginas usando la política de remplazo LLU

renglones de la tabla, debido a lo cual no hay necesidad de asignar algún valor de uso a los renglones de la tabla.

Caso 2. Consulta a sólo una tabla, con cláusula WHERE sencilla sin índice. En este caso el valor de uso asignado es *0* ya que la lectura de la tabla es secuencial, y por consiguiente, una vez que un renglón es leído es posible eliminarlo de la memoria principal.

Caso 3. Consulta a sólo una tabla, con cláusula WHERE sencilla con índice. Debido a que se utiliza un índice, es posible que varios renglones de una misma página cargada en el DP, sean solicitados en tiempos relativamente diferentes, por tal razón el valor de uso asignado debe ser *5*.

Caso 4. Consulta a sólo una tabla, con cláusula WHERE compleja, sin índice. Este caso se trata de manera similar al caso 2; por lo tanto, se debe asignar el valor de uso *0*.

Caso 5. Consulta a sólo una tabla, con cláusula WHERE compleja, con índice. Este caso se trata de manera similar al caso 3; por lo tanto, se debe asignar el valor de uso *5*.

Caso 6. Consulta a dos tablas, sin cláusula WHERE sin índice y con índice. Debido a que este caso no tiene cláusula WHERE, se toman en cuenta las siguientes consideraciones:

1. Los renglones de la primer tabla se leen sólo una vez; por lo tanto, el valor de uso es *0*.
2. Los renglones de la segunda tabla se leen el número de renglones de la primer tabla; por lo tanto, el valor de uso es *10*.

Caso 7. Consulta a dos tablas, con cláusula WHERE sencilla, sin índice y con índice. Las consideraciones a tomar para este caso, son las siguientes:

1. En caso de que ambas tablas no tengan un índice útil, se debe decidir crear un índice útil a una de ellas.
2. La tabla que no tiene índice se leerá de manera secuencial; por lo tanto, se le asigna el valor de uso *0*, tal y como sucede en los casos 2 y 4.
3. Los renglones seleccionados de la tabla que tiene el índice útil, se leerán de forma aleatoria. Es posible que varios renglones de una misma página cargada en el DP sean solicitados en tiempos relativamente diferentes, por tal razón el valor de uso asignado debe ser *5*, tal y como sucede en los casos 3 y 5.

Caso 8. Consulta a dos tablas con cláusula WHERE compleja (con y sin índice). Debido a que este caso es similar al caso 7, las consideraciones a seguir son las mismas que las del caso anterior.

Caso 9. Consulta a tres o más tablas (con y sin cláusula WHERE y con y sin índice). En este caso se considera lo siguiente:

1. Se seleccionan dos de las tablas que participan en la consulta.
2. En caso de que las tablas seleccionadas no tengan un índice útil, se le crea un índice útil a una de ellas.
3. La tabla que no tiene índice se leerá de manera secuencial; por lo tanto, se le asigna el valor de uso *0*, tal y como sucede en los casos 2 y 4.
4. Los renglones seleccionados de la tabla que tiene el índice útil, se leerán de forma aleatoria. Es posible que varios renglones de una misma página cargada en el DP sean solicitados en tiempos relativamente diferentes, por tal razón el valor de uso asignado debe ser *5*, tal y como sucede en los casos 3 y 5.
5. El resultado de las dos tablas seleccionadas se toma como una sola tabla.
6. Si existe(n) tabla(s) que no haya(n) sido procesada(s), se selecciona otra tabla de las que participan en la consulta, de lo contrario se finaliza.
7. Si la tabla seleccionada no tiene índice útil, se le crea uno, y el resultado de las dos tablas que surgió del punto número 5, se toma como una sola tabla para hacer el join con la nueva tabla seleccionada, posteriormente se repiten los pasos del 3 en adelante.

Como puede observarse a lo largo de esta sección, las políticas de remplazo tradicionales requieren de un menor número de pasos que la política de remplazo orientada al área de bases de datos y la política de propósito general.

3.6 Precarga informada en el directorio de páginas

Cuando se realiza una consulta al prototipo, se hace un análisis previo donde se verifica si es posible precargar la(s) tabla(s) que interviene(n) en la consulta. En caso de que no sea posible, se verifica si es posible hacer uso del doble contenedor (ver detalles en [36]). Si no es posible hacer uso de alguna de las dos opciones mencionadas anteriormente debido a la cantidad de memoria disponible, se hará uso del directorio de páginas, donde se precargará un número determinado de páginas, las cuales se insertarán acorde con la política de remplazo que se esté utilizando en ese momento.

El anexo A.1 incluye un análisis detallado de la taxonomía de la instrucción SELECT donde se especifica cuándo utilizar la precarga informada y las políticas de remplazo.

3.6.1 Especificaciones de la precarga informada en el DP

Es importante señalar que la precarga que se utiliza para cargar toda una tabla en memoria principal, es diferente a la precarga que se usa para cargar las páginas en el DP. La Figura 3.20 muestra de manera gráfica ambas precargas. La diferencia radica en que la primera (*inciso a*) hace una carga masiva de toda la tabla como si fuese un solo bloque, y su almacenamiento no requiere el uso de una estructura de datos; mientras que la segunda *(inciso b)* se realiza en bloques pequeños, y hace uso de una estructura convencional (lista enlazada) para su almacenamiento.

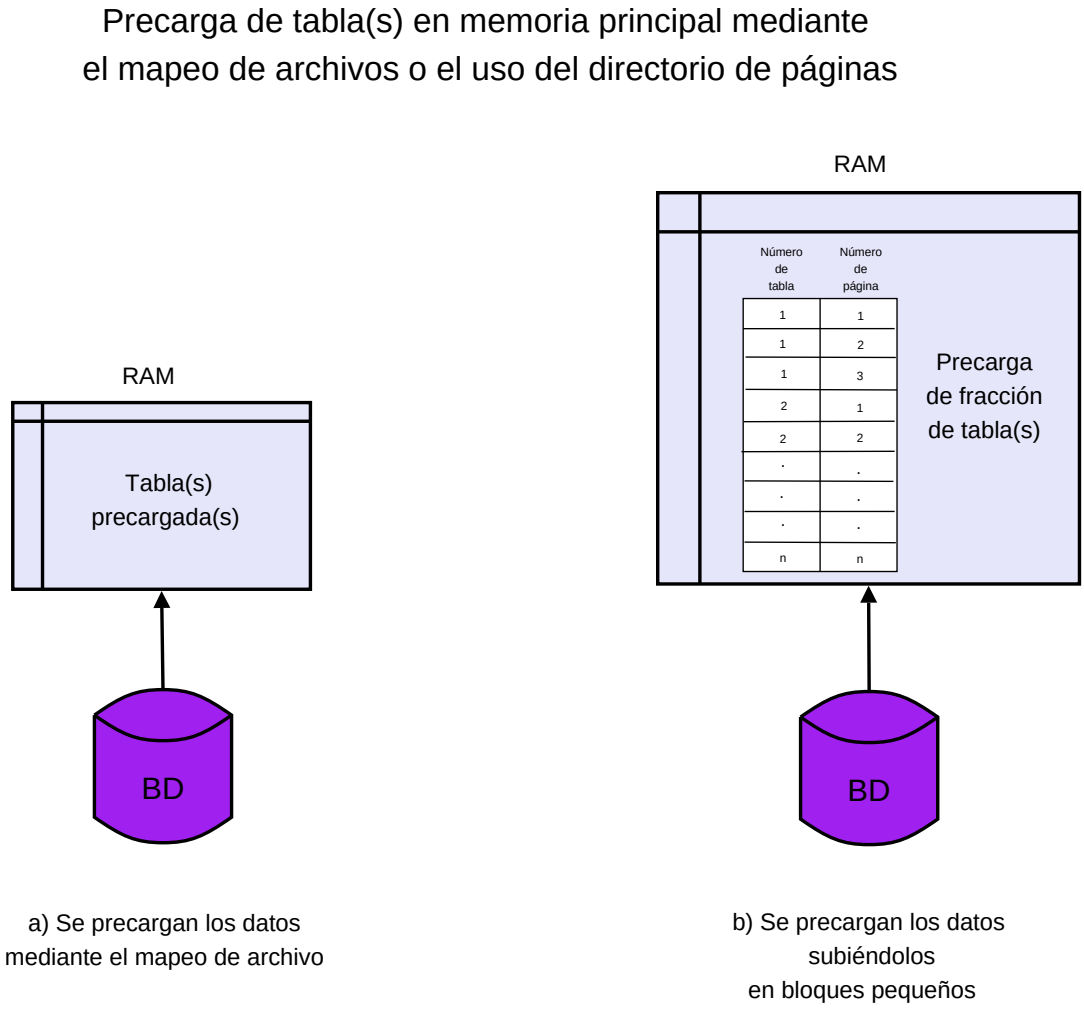

Figura 3.20: Precarga de datos en memoria principal

CAPÍTULO 4

Implementación

Al prototipo experimental, en la versión anterior se le agregó un administrador de contenedores [36]. Una de las principales tareas para lo cual fue desarrollado, es para administrar los diferentes tipos de contenedores donde se precargan los datos.

En tal versión del prototipo, además de ejecutar en paralelo el cliente y el servidor para realizar una consulta en modo cliente-servidor, se requiere que se ejecuten otros tres servicios, los cuales son: *administrador de la lista de candidatos de precarga*, *administrador de la lista de contenedores*, y *administrador del doble contenedor* donde el *administrador de la lista de candidatos de precarga* analiza la lista y decide el tipo de contenedor a utilizar:

- Supercontenedor: Utiliza un contenedor para colocar la tabla completa en memoria principal.
- Contenedor doble: Utiliza dos contenedores para colocar grandes fragmentos de la tabla en memoria principal.

El *administrador de la lista de contenedores* se encarga de administrar la lista de supercontenedores, para lo cual utiliza el mapeo de archivos; mientras que el

administrador del doble contenedor se encarga de administrar la lista de los contenedores dobles.

Los tres servicios mencionados anteriormente sirven como base para implementar el directorio de páginas e integrar el manejador de índices y las políticas de remplazo en el prototipo.

El entorno de programación que se utilizó en este proyecto de investigación fue:

- Sistema operativo Linux, distribución Fedora Core 1.
- Interfaz de desarrollo gráfica *Anjunta versión 1.2*.
- Lenguaje de programación *C*.
- Compilador *gcc versión 3.3.4*.

4.1 Archivo de configuración que se utiliza para ejecutar el prototipo

El archivo de configuración es utilizado para conocer si se aplica o no la precarga, si se utiliza(n) o no el(los) índice(s), que política de remplazo a utilizar, etc. La Tabla 4.1 muestra cada uno de los parámetros que lo integran, así como sus respectivos valores para un caso de prueba.

A continuación se detalla cada uno de los parámetros que se utilizan en este archivo de configuración:

ActivarPrecarga: Mediante este parámetro es posible determinar si se realiza o no la precarga informada de los datos. El valor de 1 indica que se realiza la precarga, mientras que el valor de 0 indica que no se utiliza la precarga.

Tabla 4.1: Archivo de configuración

Parámetro	Valor
ActivarPrecarga	0
TamanoPagina	1024
PorcentajeHolguraProc	10
OrdReunion	1
PolRempGlobal	1
PolRempLocal	4
MaxNumArchsEnDP	10
MaxNumPagsEnDP	1000
MaxNumRengPorPag	200
UsoIndices	1
CriterioSeleccionIndice	1
MinNumRenParaCrearIndice	500
TamLirs	1

TamañoPagina: Indica el tamaño de la página expresado en bits. Se recomienda utilizar múltiplos de 1024; por ejemplo, 1024, 2048, 4096, etc.

PorcentajeHolguraProc: Es un porcentaje del tamaño de la memoria principal que no será utilizado, esto es con la finalidad de que el SO utilice esta memoria en otros procesos.

OrdenReunion: Este parámetro es utilizado para determinar el orden de las tablas; es decir, cuando no hay espacio suficiente en memoria principal para precargar dos tablas. Con este valor se define si se precarga la tabla 1 o la tabla 2.

PolRemplazoGlobal: Este valor indica la política de remplazo global a utilizar. La Tabla 4.2 muestra los valores que se pueden asignar a este parámetro.

Tabla 4.2: Políticas de remplazo globales

Número	Nombre de la política
1	LLU
3	MRU
4	LRU
6	LIRS
7	2Q

PolRemplazoLocal: Este valor indica la política de remplazo local a utilizar.

MaxNumArchsEnDP: Este valor es el máximo número de archivos que se permiten en el DP.

MaxNumPagsEnDP: Este valor es el número máximo de páginas que puede tener el DP.

MaxNumRengsPorPag: Este valor es el máximo número de renglones que puede tener una página.

UsoIndices: Mediante este parámetro es posible determinar si se utilizan o no los índices. Si el valor es 1 sí se utilizan, y si el valor es 0 no se utilizan.

CriterioSeleccionIndice: Este parámetro es utilizado cuando se realiza una consulta a dos tablas y ambas tienen índice útil. Con éste se define de qué tabla se utilizará su índice.

MinNumRenParaCrearIndice: Cuando se realiza una consulta a dos tablas y ninguna de ellas tiene índice útil, es necesario determinar si es conveniente crear o no un índice. Este valor nos indica el mínimo número de renglones que debe tener una tabla para que se le pueda crear un índice.

TamLirs: Este valor determina el tamaño de la pila denominada LIR, la cual es utilizada en la política de remplazo global LIRS.

4.2 Archivos que se actualizaron para integrar el manejador de índices al prototipo

Algunos de los archivos de mayor importancia que se actualizaron para realizar este proyecto son:

- ac.c: Permite tener información relacionada con los índices en la memoria compartida.
- ac.h: Define las estructuras de datos donde se almacena información relacionada con los índices.
- index.h: Contiene rutinas relacionadas con el manejador de índices.
- AdminLCP.c: Permite administrar el uso de índices cuando se utiliza la precarga.
- ejecuta2.c: Permite obtener información relacionada con la cláusula WHERE y la cláusula FROM de la consulta realizada al prototipo.
- sql2.c: Permite integrar el manejador de índices al prototipo.

4.3 Integración del manejador de índices al prototipo

El manejador de índices que se integró al prototipo, consta de un archivo denominado index.h, el cual contiene rutinas que permiten crear, abrir, buscar, leer y eliminar

índices, entre otras operaciones. Es importante mencionar que para realizar esta integración fue necesario hacer un rediseño del mismo, ver detalles en [34].

Para determinar cuándo es útil un índice y qué índice usar (ver Sección 3.3), se utiliza la lista enlazada que se genera del análisis sintáctico de la cláusula WHERE, ver detalles en [38]. A partir de esta lista, se genera una nueva lista en memoria compartida, la cual mantiene la información accesible para diferentes procesos. La Figura 4.1 muestra la estructura de esta lista.

```
typedef struct ListaWhere LW;
struct ListaWhere {
   char operador[2]; /* 1=, 2 <>, 3 >,
                     4 <, 5 >=, 6 <= */
   Byte TipoLiteral; /* 0 Es Columna,
                     1 Char, 2 Int,
                     3 LInt, 4 Real,
                     9 Date */
   Byte TipoDato;
   int OffSetE1, OffSetE2;
   int tamanoE1, tamanoE2;
   char nomTablaE1[20], nomTablaE2[20];
      union {
      /* 1 */ str255 CharP;
      /* 2 */ short int IntP;
      /* 3 */ int LIntP;
      /* 4 */ float RealP;
      /* 9 */ long int DateP;
      } TValorLiteral;
   int Ant, Sig;

   }; LW *ListaExprWhere;
```

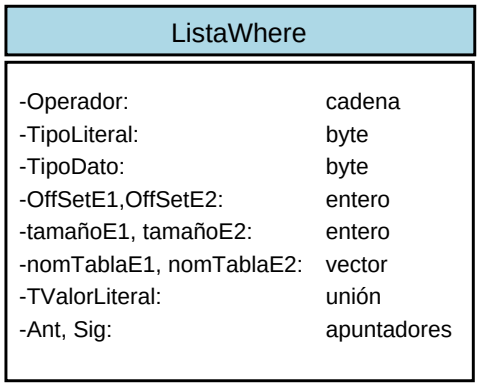

Figura 4.1: Estructura de la lista de expresiones de la cláusula WHERE.

Esta nueva lista almacena la condición de búsqueda que aparece en la cláusula WHERE. Esta condición puede tener la forma: *expresión1 operador expresion2*. También es importante almacenar qué tipo de dato corresponde a cada campo o literal (valor constante), el desplazamiento de cada uno de los elementos y el nombre de la tabla a la que se hace referencia en la(s) expresión(es) de la cláusula WHERE.

Cuando se tienen los datos mencionados, éstos se almacenan en la estructura que se muestra en la Figura 4.2. Entonces se debe verificar cada uno de los índices de la(s) tabla(s) que interviene(n) en la consulta. Primero se debe verificar si la tabla tiene un índice compuesto, si es así cada una de las columnas que lo componen deben estar en la *lista de expresiones where* (mencionada anteriormente), y los datos de este índice se almacenan en la estructura *VarIndice*. En el caso de que la(s) tabla(s) no tenga(n) un índice compuesto útil, se debe verificar si tiene(n) un índice simple útil, esto es, que la columna que lo compone debe encontrarse en la *lista de expresiones where*. Al igual que el caso del índice compuesto, si existe algún índice simple que cumpla con la condición anterior, la información de éste se almacena en la estructura *VarIndice*.

```
typedef struct VariableIndice VarIndice;
struct VariableIndice {
    Byte IndiceUtil[MaxFromTables];
    char ColIndice[MaxFromTables][MaxFromTables][20];
    char NombreIndice[MaxFromTables][20];
    Byte NumeroIndice[MaxFromTables];
    int  NumeroRenglones[MaxFromTables];
    Byte TipoIndice[MaxFromTables];
    Byte TipoColumna[MaxFromTables][MaxFromTables];
}; VarIndice *VIndice;
```

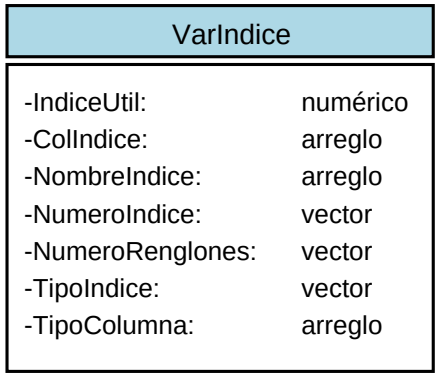

Figura 4.2: Estructura que almacena en la memoria compartida, la información de los índices.

Es importante mencionar que la estructura *VarIndice* almacena la información de los índices para las tablas que participan en una consulta. A continuación se detalla cada uno de los elementos que la componen.

- *IndiceUtil*: Guarda el valor de *1* o *0*, el cual indica si el índice es útil o no.
- *ColIndice*: Almacena las columnas que conforman el índice que se utilizará.
- *NombreIndice*: Almacena el nombre del índice útil.

- *NumeroRenglones*: Almacena el número de renglones de la tabla. Esto es importante para saber si es conveniente crear un índice a la tabla.
- *TipoIndice*: Guarda el valor de 1 si el índice es simple y un valor diferente de 1 si el índice es compuesto. (Este valor determina el número de columnas por las que está formado el índice compuesto.)
- *TipoColumna*: Almacena el tipo de dato de la(s) columna(s) que compone(n) el índice simple (o compuesto).

Para integrar el manejador de índices al prototipo, fue necesario conocer cuál es el proceso que se realiza para acceder a los datos solicitados por una consulta. La Figura 4.3 muestra un diagrama de flujo de este proceso tal y como lo realiza la versión anterior del prototipo. En la figura aparecen dos secciones sombreadas, las cuales se requieren modificar para integrar los índices. Cada sección utiliza una función diferente donde se realiza la integración del manejador de índices al prototipo, las cuales son:

- ExclausulaWhere(): Se usa cuando se realiza una consulta a una tabla.
- _From2_Indices(): Se usa cuando se realiza una consulta a dos tablas.

Cuando se realiza una consulta a sólo una tabla y el parámetro *UsoIndices* del archivo de configuración (ver Figura 4.1) tiene el valor de 1, se debe validar lo siguiente:

- Que la tabla tenga por lo menos un índice útil.
- Que el uso de índices se haya activado en el archivo de configuración (ver Sección 3.4).
- Que la cláusula WHERE esté compuesta sólo de operadores AND.

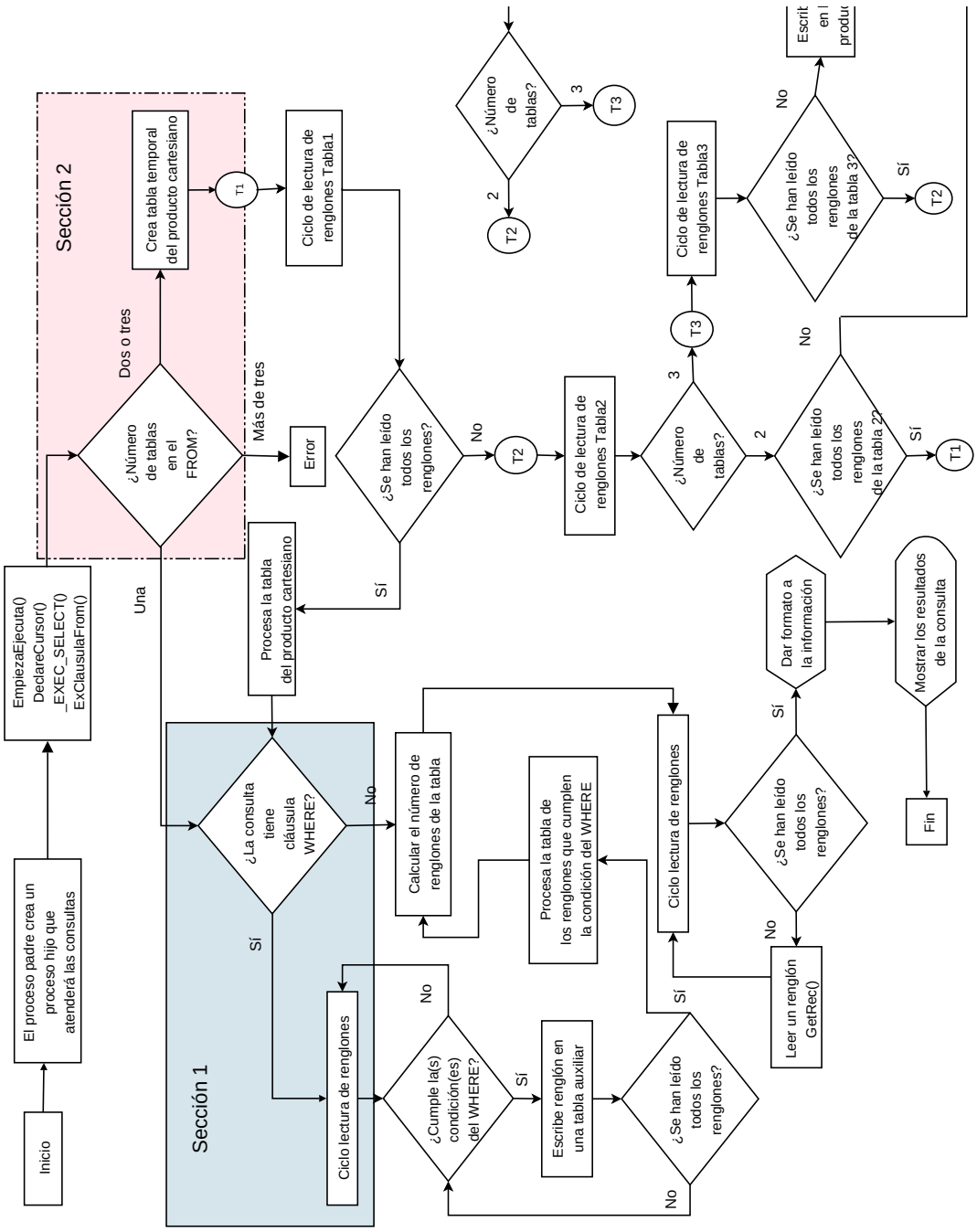

Figura 4.3: Accesos a datos al procesar una consulta en la versión anterior del prototipo.

Si alguno de los tres puntos anteriores no se cumple, el proceso se realiza tal cual lo hace la versión anterior del prototipo; en caso contrario se abre el índice y se busca la llave, si ésta es encontrada se recupera el renglón de la tabla y en caso de que haya más de una condición en la cláusula WHERE se evalúa(n) la(s) condición(es) restante(s). Si el renglón cumple con la(s) condición(es), se almacena en un archivo temporal, se solicita la siguiente llave y se repite este proceso. En la Figura 4.4 se observa la integración del uso de índices en la Sección 1 de la Figura 4.3. Es importante mencionar que el proceso *uso de índices(una tabla)* corresponde al diagrama de flujo que se presentó en el capítulo anterior, ver Figura 3.4, la cual corresponde a la integración del manejador de índices al prototipo para el caso de consultas a una sola tabla.

En el caso de que el valor del parámetro *UsoIndices* del archivo de configuración sea 0, eso indica que no deben utilizarse los índices en caso de que los haya; así que el proceso se realiza tal cual lo hace la versión anterior del prototipo.

Para el caso de una consulta a dos tablas, la Sección 2 de la Figura 4.3 se modificó para integrar el manejador de índices al prototipo. Tales cambios se muestran en la Figura 4.5, donde el proceso *uso de índices(dos tablas)* corresponde a la Figura 3.5 del capítulo anterior la cual corresponde a la integración del manejador de índices al prototipo para el caso de consultas a dos tablas.

4.4 Integración del manejador de índices con precarga informada

La versión anterior del prototipo donde se implementó la precarga, utiliza un archivo llamado *AdminLCP.c*, el cual es el encargado de revisar la lista de candidatos de precarga y colocarlas en las listas de acciones más convenientes:

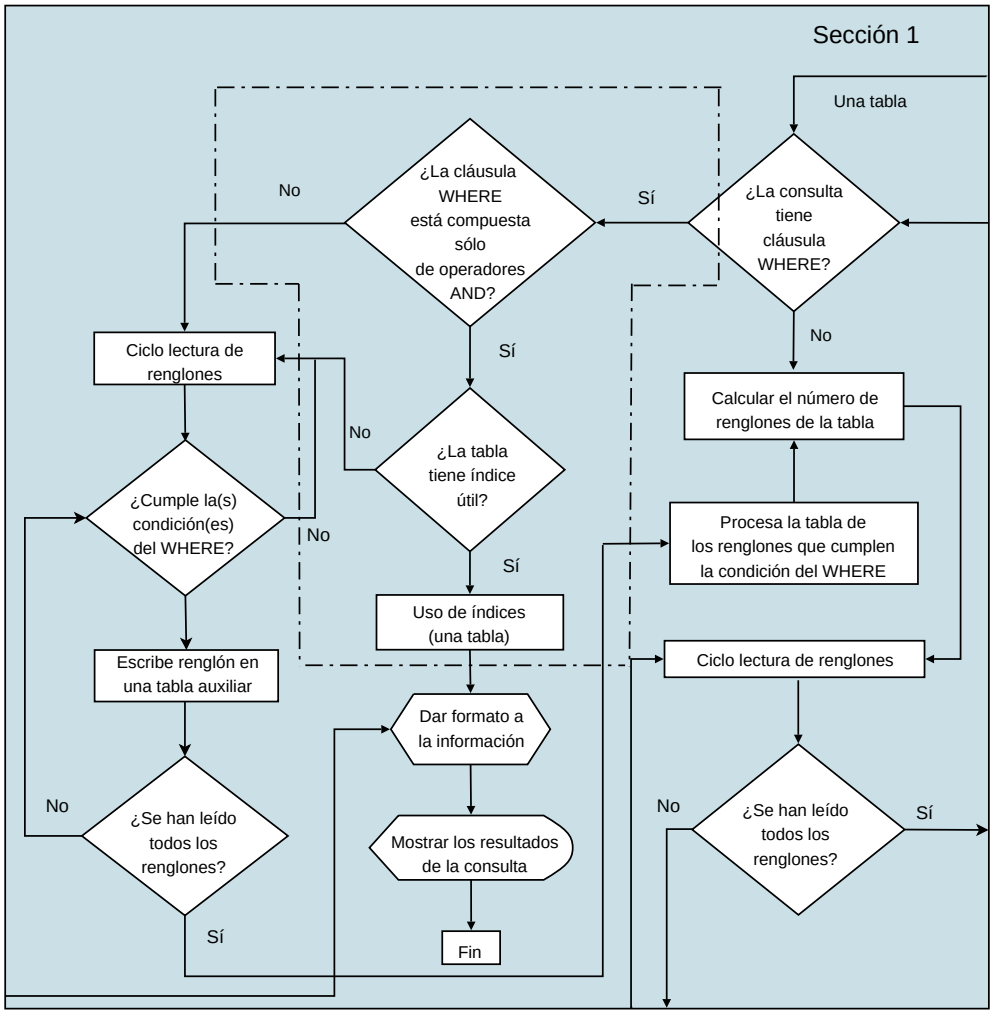

Figura 4.4: Integración del manejador de índices al prototipo cuando es una tabla.

- Doble contenedor.
- Mapeo de archivo.

La lista de candidatos de precarga tiene almacenados en memoria compartida los datos de la(s) tabla(s) que participan en la(s) consulta(s). Esto es con la finalidad de saber qué acciones se deben tomar de acuerdo al tamaño de memoria que se encuentre disponible en ese momento, ver detalles en [36]. Para que fuese posible el uso de los índices, sólo fue necesario hacer modificaciones en algunos estados donde se especifica qué índice se utilizará.

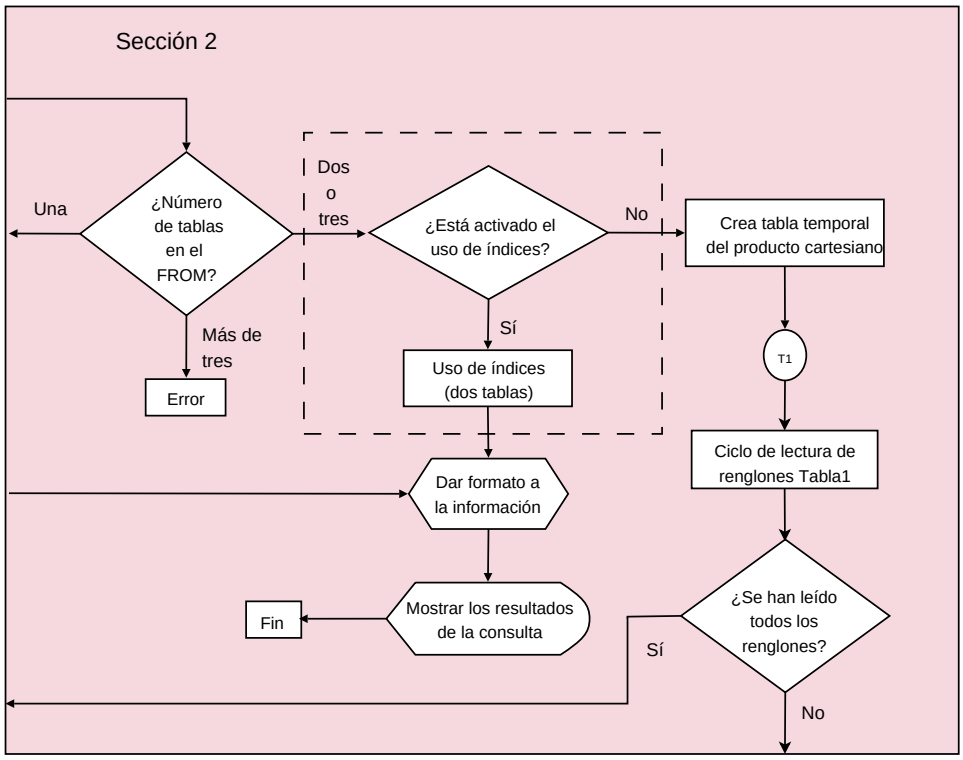

Figura 4.5: Integración del manejador de índices al prototipo cuando son dos tablas.

4.5 Implementación e integración del directorio de páginas al prototipo

Para que el prototipo procese una consulta se requieren de diferentes subprocesos que permitan obtener información necesaria. Uno de esos subprocesos realiza un análisis sintáctico de la consulta para obtener el número de tablas que participan en la consulta, así como también el número de renglones de cada una de ellas. Al finalizar este análisis se verifica qué cantidad de renglones es posible precargar de acuerdo al tamaño y espacio disponible de la memoria principal, ver detalles en [36]. Cuando no es posible mapear la(s) tabla(s) que intervienen en la consulta o utilizar el doble contenedor, se debe hacer uso del DP, el cual mantiene en memoria principal ciertas páginas que contienen algunos renglones solicitados hasta ese momento.

La Tabla 4.3 muestra la estructura de control del directorio de páginas y enseguida se describe cada uno de los elementos que la componen.

Tabla 4.3: Estructura de control del directorio de páginas.

Número de archivo
Número de página
Marco de página
Indicador de uso de página
Apuntador a la página siguiente
Apuntador a la página anterior
Apuntador a la página siguiente de acuerdo al valor de uso
Apuntador a la página anterior de acuerdo al valor de uso

Número de archivo: Guarda el número de tabla a la que pertenece esa página.

Número de página: Guarda el número de página de la tabla.

Marco de página: Contiene el fragmento de información de la tabla.

Indicador de uso de página: Valor que indica si la página está en uso o no.

Apuntador a la página siguiente: Valor que permite acceder a la página siguiente del DP.

Apuntador a la página anterior: Valor que permite acceder a la página anterior del DP.

Apuntador a la página siguiente de acuerdo al valor de uso: Valor que permite acceder a la página siguiente del DP de acuerdo a la política de remplazo que se esté utilizando en ese momento.

Apuntador a la página anterior de acuerdo al valor de uso: Valor que permite acceder

a la página anterior del DP de acuerdo a la política de remplazo que se esté utilizando en ese momento.

En la implementación del DP se utilizaron las estructuras *Pagina* e *InformacionDP*, las cuales se describen a continuación.

- *Pagina*: Esta nueva estructura es utilizada por el DP para almacenar los datos de la(s) tabla(s). La Figura 4.6 muestra su definición.

```
typedef struct Pagina_es Pagina;
struct Pagina_es {
      int           numArch;
      int           numPagina;
      int           num_ref;
      Byte          vRemplazo;
      Byte          PagEnUso;
      unsigned char *Marco;
      Pagina        *sig;
      Pagina        *ant;
      Pagina        *sigVR;
      Pagina        *antVR;
 };
```

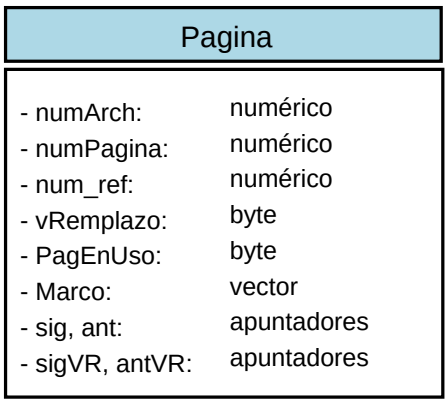

Figura 4.6: Definición de la estructura *Pagina*.

La descripción de los elementos de la estructura *Pagina* se presenta a continuación:

  - *numArch*: Identifica la tabla a la que pertenece la página.
  - *numPagina*: Identifica el número de página, ver Figura 3.6.
  - *num_ref*: Es utilizada por la política de remplazo LIRS. Contabiliza el número de veces que refiere una página.
  - *vRemplazo*: Es utilizada por la política de remplazo LLU. Almacena el valor de uso que se le asigna a una página.
  - *PagEnUso*: Es utilizada en la política de remplazo LLU. Indica si la página se encuentra en uso o no.

- *Marco*: Contiene un fragmento de información de la tabla.
- **sig*: Permite acceder a la página siguiente del DP.
- **ant*: Permite acceder a la página anterior del DP.
- **sigVR*: Permite acceder a la página siguiente del DP de acuerdo a la política de remplazo que se esté utilizando en ese momento.
- **antVR*: Permite acceder a la página anterior del DP de acuerdo a la política de remplazo que se esté utilizando en ese momento.

- *InformacionDP*: Estructura utilizada por el DP para mantener un control del inicio del DP así como también el inicio y fin de las listas utilizadas por las diferentes políticas de remplazo. La Figura 4.7 muestra su definición.

```
typedef struct Informacion_DP InformacionDP;
struct Informacion_DP
  {
          Pagina *InicioDP;
          Pagina *RaizVR[11];
          Pagina *FinalVR[11];
          Pagina *Raiz;
          Pagina *Final;
          Pagina *Head_Ain, *Tail_Ain;
          Pagina *Head_Aout, *Tail_Aout;
          Pagina *Head_Am, *Tail_Am;
          Pagina *Head_Hir, *Tail_Hir;
          Pagina *Head_Lir, *Tail_Lir;
  }; InformacionDP *InfoDP;
```

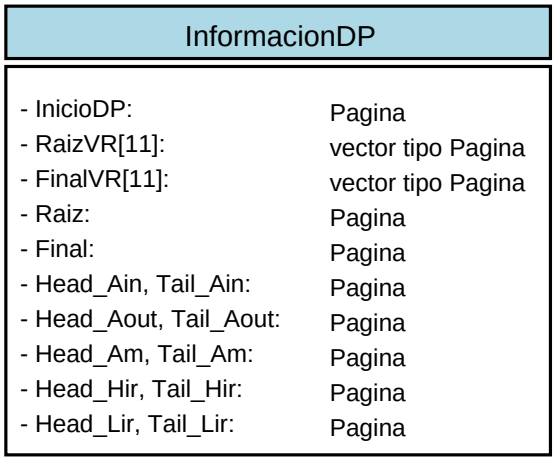

Figura 4.7: Definición de la estructura *InformacionDP*.

La descripción de los elementos de la estructura *InformacionDP* se presenta a continuación:

- **InicioDP*: Indica el inicio del DP.
- **RaizVR[11]*: Vector de apuntadores que se utiliza en la política de remplazo *LLU*, donde cada posición indica la raíz de la lista de acuerdo al valor de

uso.

- **FinalVR[11]*: Vector de apuntadores que se utiliza en la política de remplazo *LLU*, donde cada posición indica el final de la lista de acuerdo al valor de uso.
- **Final*: Apuntador que se utiliza en la política de remplazo *LRU*, el cual indica el final de la lista.
- **Head_Ain, *Tail_Ain, *Head_Aout, *Tail_Aout, *Head_Am, *Tail_Am*: Apuntadores que se utilizan en la política de remplazo *2Q*, los cuales indican el inicio y final de cada una de las 3 listas.
- **Head_Hir, *Tail_Hir, *Head_Lir, *Head_Lir*: Apuntadores que se utilizan en la política de remplazo *LIRS*, los cuales indican el inicio y final de cada una de las 2 listas.

Una vez que se ha decidido dónde almacenar el(los) renglón(es) de la(s) tabla(s) que interviene(n) en la consulta realizada al prototipo, es necesario utilizar una función que realice la lectura de la memoria secundaria y la almacene en el DP. La Figura 4.8 muestra el pseudocódigo donde se muestra la forma en que se selecciona el tipo de lectura que se debe realizar, ya sea en un archivo mapeado, doble contenedor, directorio de páginas o en memoria secundaria. En este último caso la lectura se hace al directorio de páginas, y la solicitud de el(los) renglón(es) se realiza(n) directamente al DP; si el(los) renglón(es) solicitado(s) no se encuentra en el DP, se inserta(n) la(s) página(s) que contiene(n) el(los) renglón(es) requerido(s).

La función *GetRecDP()* se agregó al prototipo y es la encargada de solicitar el(los) renglón(es) al DP. A continuación se describen las funciones principales que intervienen en esta función:

- *ExistePagEnMem()*: El objetivo de esta función es regresar un *acierto* o *fallo de página*. Si la página es encontrada (acierto), se reubica su posición en la lista

```
Inicio
    Si se utiliza el mapeo de archivos
        Se utiliza la función GetRecMap() que permite leer renglones de
        archivos mapeados
    De lo contrario
        Si se utiliza el doble contenedor
            Se utiliza la función GetRec2C() que permite leer renglones de
            el doble contenedor
        De lo contrario
            Si se utiliza el DP
                Se utiliza la función GetRecDP() que permite leer
                renglones de el directorio de páginas
            De lo contrario
                    Se utiliza la función GetRec() que permite leer los
                    renglones de la memoria secundaria
Fin
```

Figura 4.8: Selección del lector de renglones a utilizar.

correspondiente de acuerdo a la política de remplazo que se esté utilizando, si la página no se encuentra (fallo de página) y aún hay espacio suficiente para insertar en el DP, se hace un llamado a la función *InsertaPaginaEnDP*().

- *InsertaPaginaEnDP()*: Esta función es la encargada de insertar las páginas que provocan un fallo. La inserción sólo se realiza si hay espacio en el DP. En caso de que no haya espacio y la política de remplazo que se esté utilizando sea la *MRU*, *LRU* o *LLU*, se hace un llamado a la función *RemplazaPaginaEnDP()*. En el caso de que la política de remplazo utilizada sea LIRS o 2Q, se hace el llamado a funciones que realizan la inserción y el remplazo en sus respectivas estructuras.

- *RemplazaPaginaEnDP()*: Esta función es utilizada cuando ya no hay espacio para insertar nuevas páginas en el DP. Su función es eliminar la página víctima (la cual se determina mediante la política de remplazo que se esté utilizando) y realizar un enlace de la nueva página a la lista correspondiente acorde con la política de remplazo.

4.6 Implementación de la precarga informada en el directorio de páginas

La precarga informada en el directorio de páginas es la carga de datos en el DP de manera anticipada; es decir, será llenado el directorio de páginas antes de que sean solicitados los renglones necesarios para la consulta.

4.6.1 Caso de una consulta a una sola tabla

Cuando se realiza una consulta a una sola tabla, se deben tomar en cuenta las siguientes consideraciones:

- Que la precarga de datos este activa.
- Que sea necesario el uso del DP.
- Si la política de remplazo a utilizar es la LRU, MRU o propuesta(LLU), el DP será precargado en su totalidad.
- Si la política de remplazo a utilizar es la LIRS, sólo se precarga el 99 % del tamaño del DP, es decir, el tamaño de la cola HIR.
- Si la política de remplazo a utilizar es la 2Q, sólo se precarga el 50 % del tamaño del DP, es decir, el tamaño de las áreas *Ain* y *Am*.
- Si la tabla tiene un índice útil, no es necesario precargar todos los renglones, sólo se precargarán los que cumplen con la condición de la cláusula WHERE que hace uso del campo llave.
- Si la tabla no tiene índice útil, se precargarán la cantidad de renglones posible acorde con el tamaño del DP.

Una vez que se finaliza la precarga de datos en el DP, se continúa con el proceso ya establecido.

4.6.2 Caso de una consulta a dos tablas

Cuando se realiza una consulta a dos tablas, se deben tomar en cuenta las siguientes consideraciones:

- Que la precarga de datos este activa.
- Que sea necesario el uso del DP.
- Si la política de remplazo a utilizar es la LRU, MRU o propuesta(LLU), el DP será precargado en su totalidad; el 70 % se utilizará para cargar los renglones de la tabla que no tiene índice (acceso secuencial), y el 30 % restante se utilizará para la carga de los renglones de la tabla que tiene índice.
- Si la política de remplazo a utilizar es la LIRS, sólo se precarga el 99 % (cola HIR) del tamaño del DP, y de ese 99 % se asignará un 70 %/30 % tal como en las políticas de remplazo LRU, MRU o propuesta (LLU).
- Si la política de remplazo a utilizar es la 2Q, sólo se precarga el 50 % del tamaño del DP, y de ese 50 % (área Ain y Am) se asignará un 70 %/30 % tal como en las políticas de remplazo LRU, MRU o propuesta (LLU).

Es importante señalar que el proceso de precarga de datos clasifica el campo llave de la cláusula WHERE en una literal o en una columna, ya que el tratamiento de la búsqueda de una literal en el índice de una tabla es más rápido que buscar n valores de columnas en el índice de una tabla.

Una vez que se finaliza la precarga de datos en el DP, se continúa con el proceso ya establecido.

Estructura que se utiliza para precargar datos en el DP

La estructura a utilizar es la misma que se usa en el directorio de páginas (lista doblemente enlazada). Cada vez que se inserta una página en el DP, ya sea mediante la precarga de datos o vía carga por demanda, es necesario enlazar las páginas en el directorio de páginas de tal forma que sea posible su acceso en un tiempo futuro.

CAPÍTULO 5

Pruebas

Las pruebas que se le realizaron al prototipo de SMBDs (SiMBaDD) fueron para identificar qué política de remplazo es la mas eficiente de las que se le incorporaron al administrador de memoria, además de mostrar la utilidad que se tiene cuando se le integra un manejador de índices y la técnica de precarga informada. Como se mencionó en capítulos anteriores, es conveniente que todos los datos que sean requeridos se encuentren cargados en memoria principal para que su acceso sea más rápido; sin embargo, no siempre es posible que la memoria principal tenga la capacidad suficiente. Por lo tanto, otro de los objetivos de las pruebas es conocer la configuración de los valores de los parámetros con los que se comporta mejor el administrador de contenedores para cada caso de la taxonomía de la instrucción SELECT que se muestra en el Anexo A.1.

5.1 Ambiente de pruebas

Las características del equipo en el que se realizaron las pruebas tiene un impacto de manera importante en los resultados que se obtienen. El ambiente que se utilizó para la realización de las pruebas se describe a continuación:

Software:

- Sistema Operativo Linux, distribución Fedora Core 1.
- Interfaz de desarrollo gráfica Anjuta versión 1.2.
- gcc versión 3.3.4.

Hardware:

- Computadora Genérica Pentium IV a 533 Mhz.
- 384 MBs en RAM.
- 30 GBs en disco duro.

Para realizar las pruebas en los 8 diferentes casos de la taxonomía de la instrucción SELECT antes mencionada, se utilizó un benchmark (base de datos de pruebas) [39], el cual contiene 4 tablas: *Update*, *Hundred*, *Uniques* y *Tenpct*, con 100,000 renglones cada una de ellas. Para simplificar las pruebas y resultados obtenidos, se han escogido las tablas Uniques y Tenpct. La selección fue en base a que estas tablas fueron diseñadas para realizar pruebas de consultas, además de que existen columnas que relacionan ambas tablas.

La definición de las tablas es la siguiente:

- Tabla Tenpct

```
CREATE TABLE Tenpct (Num1 INTEGER, Num2 INTEGER, Num3 INTEGER,
Real1 DECIMAL, Num4 INTEGER, Num5 INTEGER, Fecha1 DATE, Cad1
CHAR(10), Cad2 CHAR(20), Cad3 CHAR (80));
```

- Tabla Uniques

```
CREATE TABLE Uniques (Num1 INTEGER, Num2 INTEGER, Num3 INTEGER,
Real1 DECIMAL, Num4 INTEGER, Num5 INTEGER, Fecha1 DATE, Cad1
CHAR(10), Cad2 CHAR(20), Cad3 CHAR (80));
```

5.2 Creación de índices

La creación de los índices a la(s) tabla(s) del prototipo se realiza de manera independiente desde una terminal de Linux.

Los pasos a seguir son los siguientes:

1. Se ejecuta la aplicación *createindex* (manejador de índices) la cual es una aplicación independiente del prototipo de SMBD experimental.
2. Se introduce la instrucción *Create Index* la cual puede estar escrita en un archivo de texto o se puede teclear directamente al momento de ser solicitada.

Una vez que ya se ha creado el índice, las consultas que se realicen al prototipo pueden hacer uso de éste.

Para efectuar las pruebas se crearon cuatro índices a las tablas Tenpct y Uniques, de los cuales dos de ellos son simples y dos son compuestos. La Tabla 5.1 especifica los detalles de éstos.

5.2.1 Consulta a sólo una tabla

Las consultas a sólo una tabla que se realizaron al prototipo para verificar el comportamiento de los índices simple y compuesto fueron:

Tabla 5.1: Descripción de las tablas y sus índices

Tablas	Num. Registros	Índice simple	Índice compuesto
Tenpct	100,000	indice2.inx(cad1), indice3.inx(cad2)	indice5.inx (cad1, cad2)
Uniques	100,000	indice1.inx(cad1), indice4.inx(cad2)	indice6.inx (cad1, cad2)

```
$DECLARE C1 CURSOR FOR
          SELECT Uniques.Cad1
          FROM Uniques
          WHERE Uniques.Cad1 = 'Qgj8pSWKhU';


$DECLARE C1 CURSOR FOR
          SELECT Uniques.Cad1
          FROM Uniques
          WHERE Uniques.Cad1 = 'Qgj8pSWKhU'
          AND Uniques.Cad2 = 'TUgTZUvcV6vyOlC183991841';
```

La Tabla 5.2 muestra los resultados obtenidos al realizar la consulta al prototipo con el manejador de índices integrado.

Tabla 5.2: Pruebas al manejador de índices con una consulta a una sola tabla

Total de Renglones	Precarga	Índice	Tipo de Índice	Tiempo (segundos)
100,000	Sí	Sí	Simple	2.988950
	Sí	No		1.254647
	No	Sí		2.849788
	No	No		**1.135860**
	Sí	Sí	Compuesto	5.138161
	Sí	No		**1.359287**
	No	Sí		3.327854
	No	No		1.816688

5.2.2 Consulta a dos tablas

La consulta a dos tablas que se realizó al prototipo fue:

```
$DECLARE C1 CURSOR FOR
        SELECT Uniques.Cad1, Tenpct.Cad1
        FROM Uniques, Tenpct
        WHERE Tenpct.Cad2=Uniques.Cad2 AND Tenpct.Cad1=Uniques.Cad1;
```

La Tabla 5.3 muestra los resultados obtenidos al realizar la consulta al prototipo con el manejador de índices integrado.

Tabla 5.3: Pruebas al manejador de índices con una consulta a dos tablas

Total de Renglones	Precarga	Índice	Tipo de Índice	Tiempo (segundos)
100,000	Sí	Sí	Simple	**11.103257**
	Sí	No		Indefinido
	No	Sí		255.499971
	No	No		Indefinido
	Sí	Sí	Compuesto	4.723194
	Sí	No		Indefinido
	No	Sí		**4.692594**
	No	No		Indefinido

5.2.3 Análisis de resultados de la integración del manejador de índices al prototipo

Como se observa en la Tabla 5.2, los mejores tiempos de ejecución de la consulta se obtienen sin el uso de los índices. Aplicar la técnica de precarga informada no es de mucha utilidad en este caso; ***sin embargo, para el caso de una consulta a dos tablas es indispensable el uso de los índices***. La Tabla 5.3 muestra que, si no es utilizado un índice para este tipo de consulta, el tiempo se vuelve indefinido. El mejor tiempo registrado corresponde al caso de un índice simple utilizando de forma combinada la precarga informada; mientras que en el caso de un índice compuesto el uso de la precarga informada no influye en el tiempo.

5.3 Directorio de páginas

Las pruebas al directorio de páginas se realizaron con el manejador de índices integrado, y dependiendo del caso de la taxonomía de la instrucción SELECT, se utilizan o no los índices.

A continuación se describen las pruebas realizadas a cada uno de los casos.

5.3.1 Pruebas al directorio de páginas con y sin el uso de precarga informada

Caso 1: Consulta a sólo una tabla, sin cláusula WHERE sin índice y con índice

Este tipo de consulta no requiere que se realicen pruebas, dado que es una consulta a una sola tabla y no existe una cláusula WHERE; por lo tanto, la tabla de resultado es la misma tabla que se especifica en la cláusula FROM.

Caso 2: Consulta a sólo una tabla, sin cláusula WHERE sencilla sin índice

La consulta que se realizó al prototipo es la siguiente:

```
$DECLARE C1 CURSOR FOR
          SELECT Uniques.Cad1
          FROM Uniques;
```

Los tiempos de ejecución obtenidos al realizar la consulta anterior se muestran en las Figuras 5.1 y 5.2. El mejor tiempo se obtuvo al utilizar la política de remplazo propuesta (LLU), sin el uso de precarga independientemente del tamaño del DP.

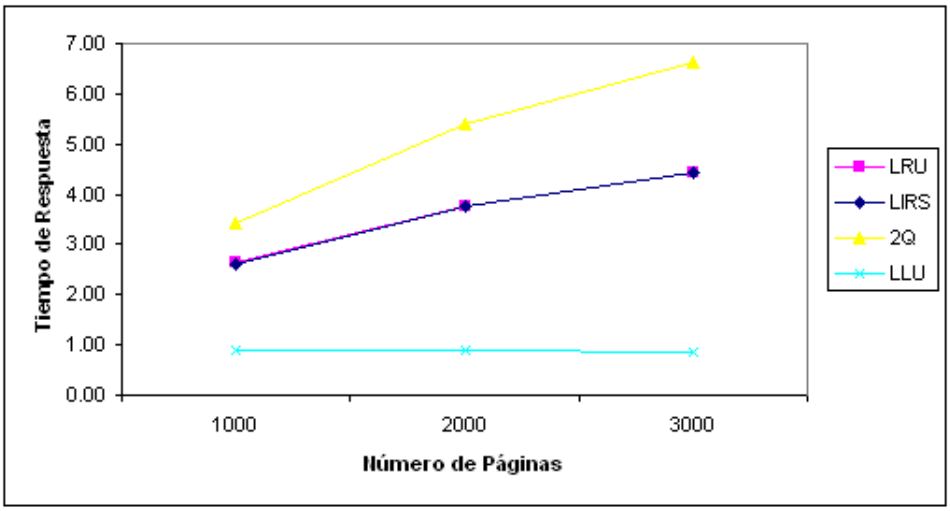

Figura 5.1: Consulta a sólo una tabla, sin cláusula WHERE sencilla sin índice, sin el uso de la precarga

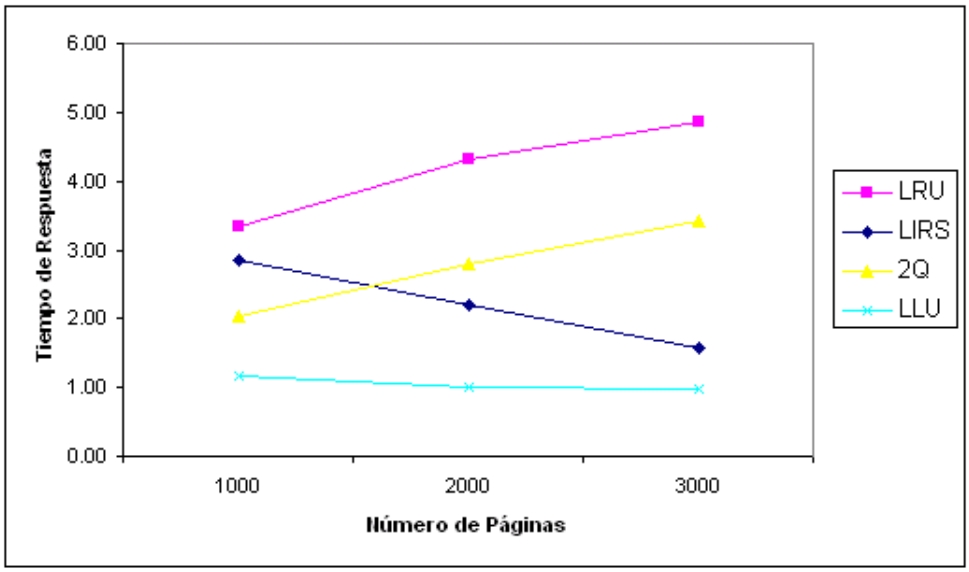

Figura 5.2: Consulta a sólo una tabla, sin cláusula WHERE sencilla sin índice, con el uso de la precarga

Caso 3: Consulta a sólo una tabla, con cláusula WHERE sencilla, con índice

La consulta que se realizó al prototipo es la siguiente:

```
$DECLARE C1 CURSOR FOR
            SELECT Uniques.Cad2
            FROM Uniques
            WHERE Uniques.Cad2 = 'dPEAB2tcrry11TQhK3VS';
```

Las Figuras 5.3 y 5.4 muestran los tiempos de ejecución de esta consulta. Sin el uso de la precarga informada, la mejor política de remplazo de manera general fue la LLU; sin embargo, el mejor tiempo de ejecución con el uso de la precarga se obtuvo al utilizar la política de remplazo LIRS utilizando un DP de 1000 y 2000 páginas, en el caso del uso del DP de 3000 páginas, la política que mejor comportamiento presentó fue la LRU. En general los mejores tiempos de respuesta se obtuvieron sin el uso de la precarga.

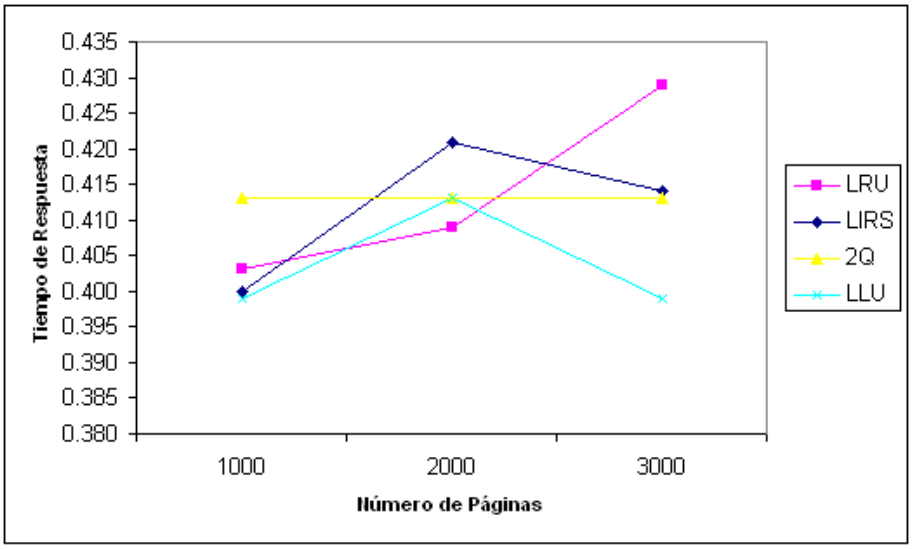

Figura 5.3: Consulta a sólo una tabla, con cláusula WHERE sencilla, con índice, sin el uso de la precarga

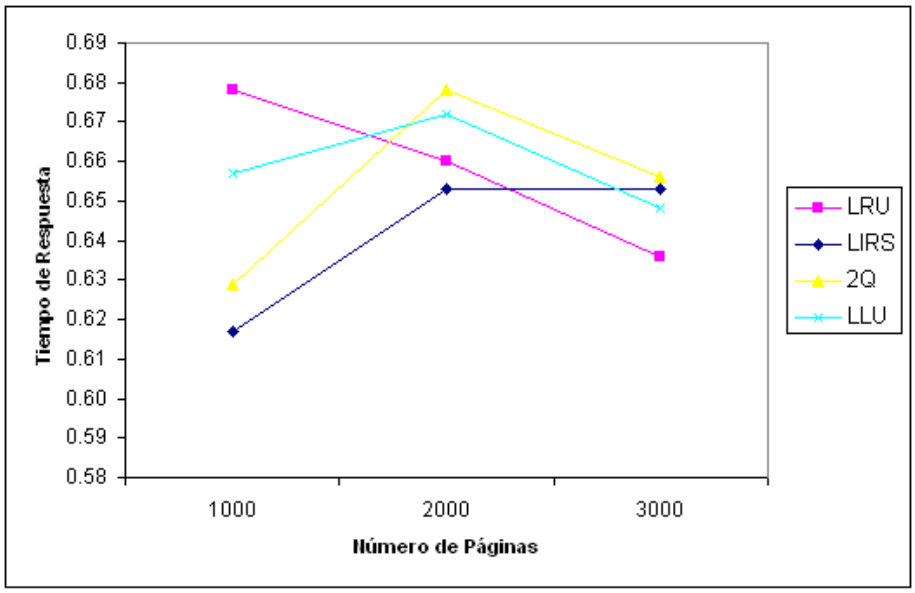

Figura 5.4: Consulta a sólo una tabla, con cláusula WHERE sencilla, con índice, con el uso de la precarga

Caso 4: Consulta a sólo una tabla, con cláusula WHERE compleja, sin índice

La consulta que se realizó al prototipo es la siguiente:

```
$DECLARE C1 CURSOR FOR
            SELECT Uniques.Cad1
            FROM Uniques
            WHERE Uniques.Cad3 = '3cCnNBBbVO'
                  AND Uniques.Num1 = 851038511;
```

En este caso, los mejores tiempos de ejecución se presentaron al utilizar la política de remplazo propuesta LLU con y sin el uso de la precarga informada e independientemente del tamaño del DP, obteniendo un ligero mejor desempeño sin el uso de la precarga. Las Figuras 5.5 y 5.6 muestran los tiempos de ejecución al realizar esta consulta.

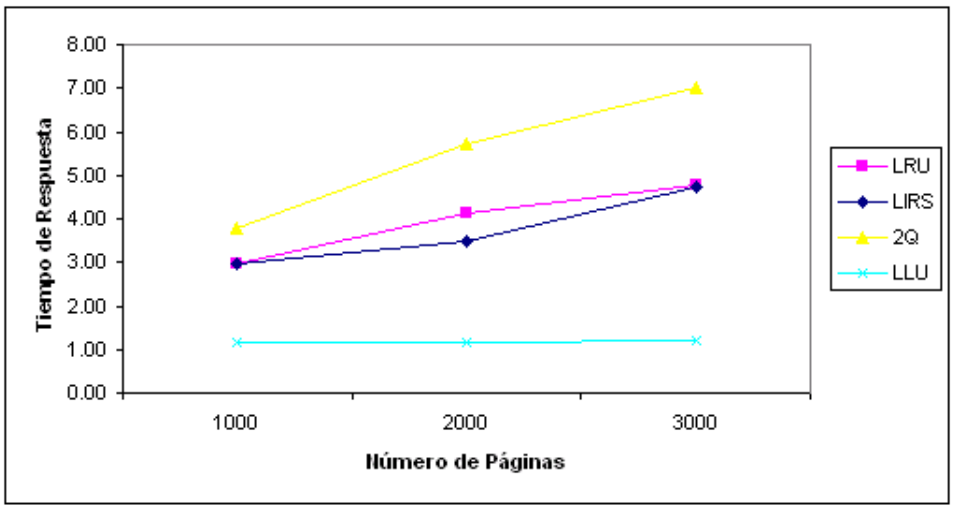

Figura 5.5: Consulta a sólo una tabla, con cláusula WHERE compleja, sin índice, sin el uso de la precarga

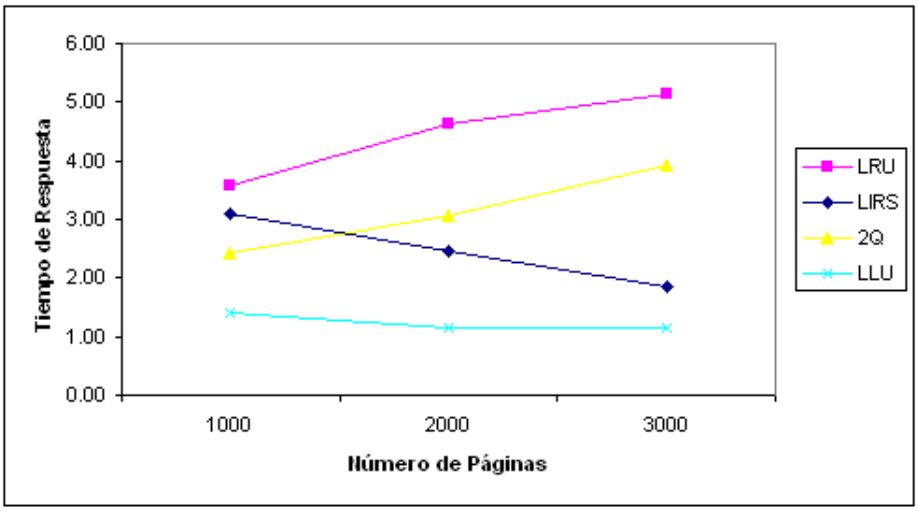

Figura 5.6: Consulta a sólo una tabla, con cláusula WHERE compleja, sin índice, con el uso de la precarga

Caso 5: Consulta a sólo una tabla, con cláusula WHERE compleja, con índice

La consulta que se realizó al prototipo en este caso es la siguiente:

```
$DECLARE C1 CURSOR FOR
            SELECT Uniques.Cad1
            FROM Uniques
            WHERE Uniques.Cad2 = 'JXqMwvAtsPVaa:bjry0T'
                  AND Uniques.Num1 = 592645927;
```

Las Figuras 5.7 y 5.8 muestran los resultados de los tiempos de ejecución que se obtuvieron al realizar la consulta al prototipo. Sin el uso de la precarga informada, la politica que mejor comportamiento presentó fue la LRU para los tres tamaños del DP, estos resultados fueron mejores que los obtenidos con el uso de la precarga informada, donde el mejor tiempo de ejecución se presentó al utilizar la política de remplazo propuesta LLU en un DP de 2000 y 3000 páginas, con un DP de 1000 páginas la mejor política fue la LRU.

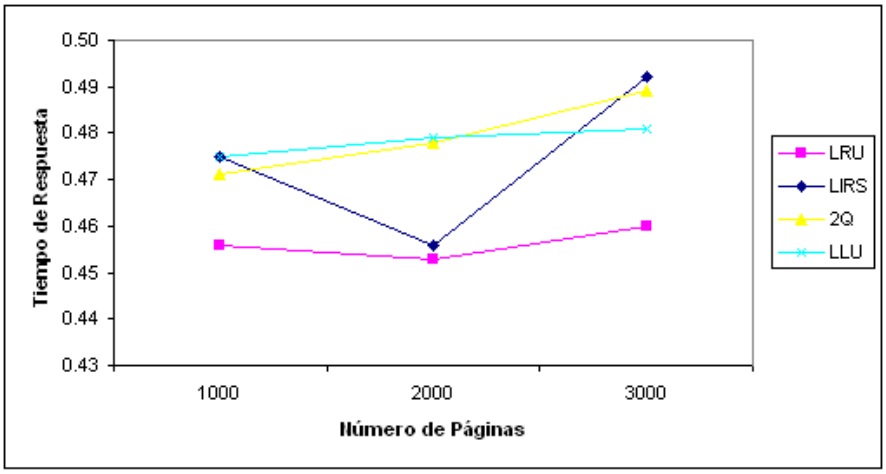

Figura 5.7: Consulta a sólo una tabla, con cláusula WHERE compleja, con índice, sin el uso de la precarga

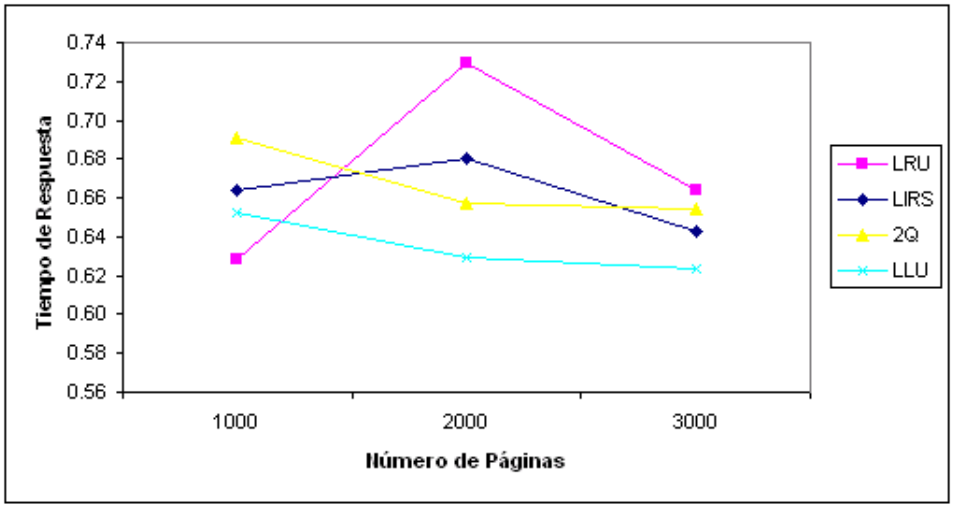

Figura 5.8: Consulta a sólo una tabla, con cláusula WHERE compleja, con índice, con el uso de la precarga

Caso 6: Consulta a dos tablas, sin cláusula WHERE, sin índice y con índice

En este caso, no se realizaron pruebas debido a que no existe una cláusula WHERE y de esta forma las pruebas pueden resultar fácilmente en un tiempo indefinido. Lo anterior se debe a que, al no tener una cláusula WHERE, se realiza un producto cartesiano; es decir, de la tabla 1 se leen todos los renglones contra todos los renglones de la tabla 2. Esto genera que los tiempos se eleven demasiado y no puedan ser obtenidos de tal forma que se declaran como indefinidos.

Caso 7: Consulta a dos tablas, con cláusula WHERE sencilla, sin índice y con índice

La consulta que se realizó al prototipo para este caso es la siguiente:

```
$DECLARE C1 CURSOR FOR
            SELECT Uniques.Cad1
            FROM Uniques, Tenpct
            WHERE Uniques.Cad1 = Tnpct.Cad1;
```

Como se mencionó en el Capítulo 3, cuando se realiza una consulta a dos tablas, es recomendable utilizar un índice; por lo tanto, aunque este caso se puede presentar con o sin índice, siempre se utilizará un índice al realizar la consulta. En las Figuras 5.9 y 5.10 se muestran los resultados obtenidos de los tiempos de ejecución, el mejor tiempo de ejecución con y sin el uso de la precarga informada se presentó con la política de remplazo propuesta LLU independientemente del tamaño del DP, mientras que los mejores tiempos en general se presentaron con el uso de la precarga.

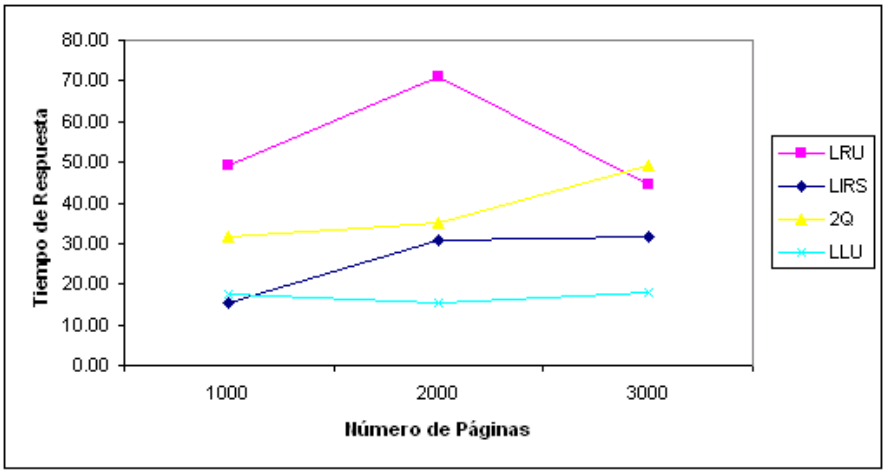

Figura 5.9: Consulta a dos tablas, con cláusula WHERE sencilla, sin índice y con índice, sin el uso de la precarga

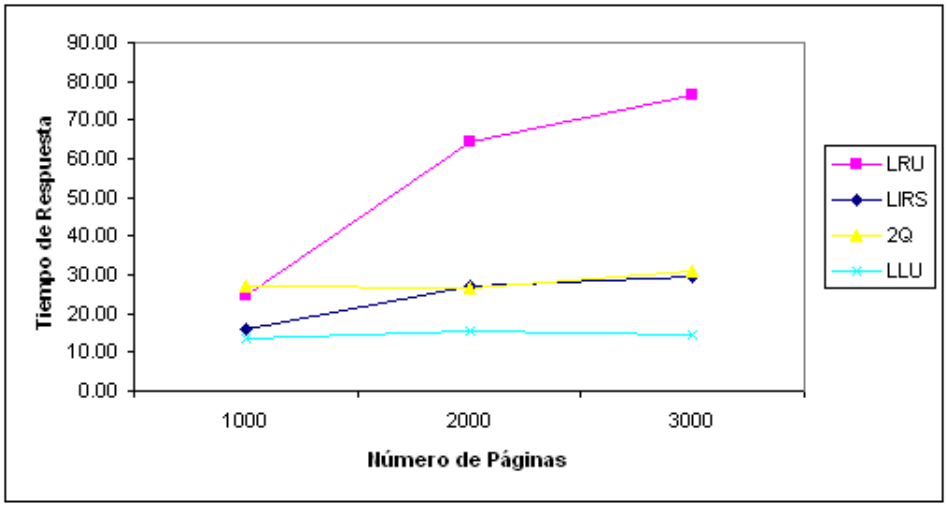

Figura 5.10: Consulta a dos tablas, con cláusula WHERE sencilla, sin índice y con índice, con el uso de la precarga

Caso 8: Consulta a dos tablas, con cláusula WHERE compleja, (con y sin índice)

La consulta que se realizó al prototipo en este caso es la siguiente:

```
$DECLARE C1 CURSOR FOR
            SELECT Uniques.Cad1
            FROM Uniques, Tenpct
            WHERE Uniques.Cad1 = Tenpct.Cad1
                  AND Uniques.Num1 = Tenpct.Num1;
```

Las Figuras 5.11 y 5.12 muestran los tiempos de ejecución que se obtuvieron al realizar esta consulta, donde se observa que sin el uso de la precarga, el mejor tiempo de ejecución se obtuvo al utilizar la política de remplazo LLU sin importar el tamaño del DP, mientras que con el uso de la precarga informada, la política de remplazo que mejor comportamiento tuvo fue la 2Q seguida de la política de remplazo LIRS, éstos tiempos fueron mejores que los obtenidos sin el uso de la precarga.

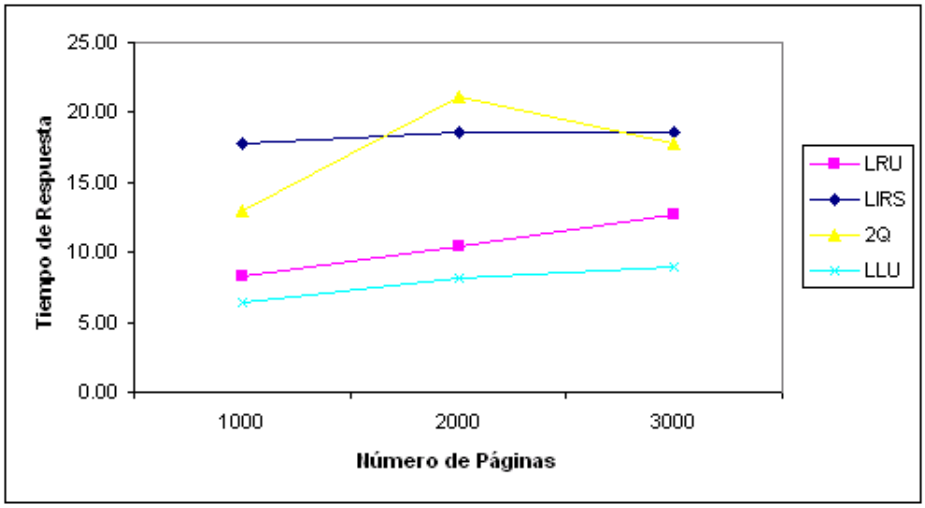

Figura 5.11: Consulta a dos tablas, con cláusula WHERE compleja, con y sin índice, sin el uso de la precarga

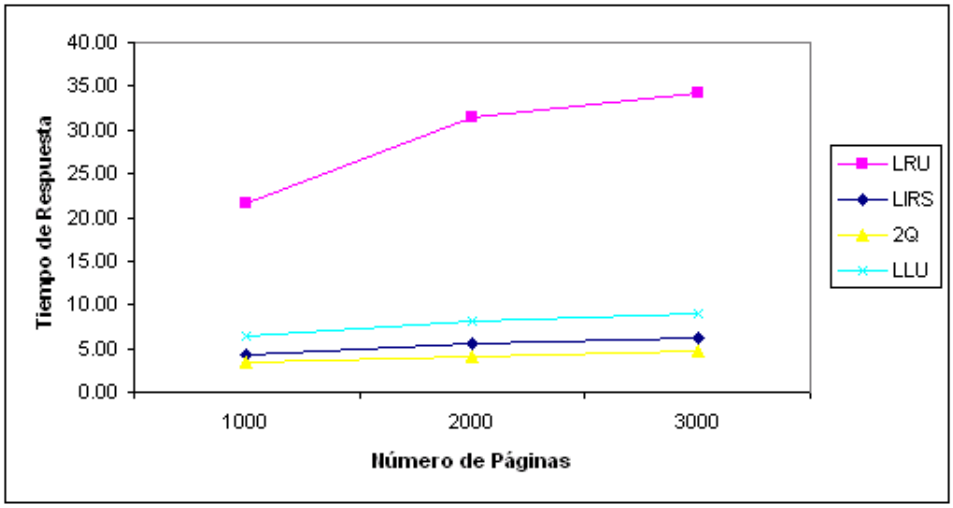

Figura 5.12: Consulta a dos tablas, con cláusula WHERE compleja, con y sin índice, con el uso de la precarga

5.4 Resultados de las pruebas realizadas al DP

Como se observa en las figuras anteriores, las pruebas se realizaron con tres tamaños diferentes de DP. El comportamiento de las políticas de remplazo es constante en los

tres diferentes tamaños; es decir, si la política de remplazo LLU se comporta mejor en un DP con 1000 páginas, es muy probable que se comporte mejor que las demás políticas en un DP con 2000 o con 3000 páginas; sin embargo, puede presentarse el caso de que otra política de remplazo se comporte mejor. En las Figuras 5.13 y 5.14 se observa un resumen de estos resultados obtenidos, a los casos 1 y 6 no se les realizaron pruebas (ver detalles en 5.3.1).

Tamaño del DP	Caso 2		Caso 3		Caso 4	
	Mejor Política	Peor Política	Mejor Política	Peor Política	Mejor Política	Peor Política
1000	LLU	2Q	LLU	2Q	LLU	2Q
2000	LLU	2Q	LRU	LIRS	LLU	2Q
3000	LLU	2Q	LLU	LRU	LLU	2Q

Tamaño del DP	Caso 5		Caso 7		Caso 8	
	Mejor Política	Peor Política	Mejor Política	Peor Política	Mejor Política	Peor Política
1000	LRU	LLU, LIRS	LIRS	LRU	LLU	LIRS
2000	LRU	LLU	LLU	LRU	LLU	2Q
3000	LRU	LIRS	LLU	2Q	LLU	2Q

Figura 5.13: Resumen de los resultados obtenidos al realizar pruebas con las diferentes políticas de remplazo sin el uso de la precarga informada

Tamaño del DP	Caso 2		Caso 3		Caso 4	
	Mejor Política	Peor Política	Mejor Política	Peor Política	Mejor Política	Peor Política
1000	LLU	LRU	LIRS	LRU	LLU	LRU
2000	LLU	LRU	LIRS	2Q	LLU	LRU
3000	LLU	LRU	LRU	2Q	LLU	LRU

Tamaño del DP	Caso 5		Caso 7		Caso 8	
	Mejor Política	Peor Política	Mejor Política	Peor Política	Mejor Política	Peor Política
1000	LRU	2Q	LLU	2Q	2Q	LRU
2000	LLU	LRU	LLU	LRU	2Q	LRU
3000	LLU	LRU	LLU	LRU	2Q	LRU

Figura 5.14: Resumen de los resultados obtenidos al realizar pruebas con las diferentes políticas de remplazo con el uso de la precarga informada

En las gráficas anteriores, el comportamiento del DP se deteriora al incrementar el tamaño del mismo. Esto tiene su origen debido a que, entre más grande sea el DP, toma más tiempo localizar la página deseada.

CAPÍTULO 6

Conclusiones

En este capítulo se mencionan las conclusiones de este trabajo de investigación, así como sus limitaciones y se proponen posibles trabajos futuros.

La hipótesis planteada al inicio de este trabajo de investigación propone que al integrar la técnica de precarga informada con políticas de remplazo orientadas al área de base de datos, se disminuirá el tiempo de respuesta de las consultas realizadas al prototipo de SMBDs experimental utilizado. Dicha hipótesis, al finalizar esta investigación, se concluye que no es verdadera para todos los casos debido a que en su mayoría la política que mejor resultados presentó fue la política propuesta LLU.

6.1 Conclusiones de la integración del manejador de índices

De acuerdo a las pruebas realizadas en el capítulo anterior, se puede concluir que:

- El integrar un manejador de índices a un SMBD es sumamente importante, ya

que el uso de éste permite disminuir el número de accesos a la memoria secundaria o primaria, lo cual depende de dónde se encuentren almacenados los datos.

- No es necesario combinar el manejador de índices con la técnica de precarga informada para generar buenos resultados.
- En el caso de consultas a una sola tabla, no es relevante el uso de índices y de precarga informada.
- En el caso de consultas a más de una tabla, el uso de estas técnicas (índices y precarga) combinadas se vuelve cada vez más importantes debido a que se obtienen mejores tiempos de respuesta.

6.2 Conclusiones de la integración de las políticas de remplazo

Como se vio en el Capítulo 1, uno de los principales objetivos de esta tesis es la implementación de un directorio de páginas que utilice diferentes políticas de remplazo orientadas a diferentes áreas. Después de las pruebas realizadas, podemos concluir lo siguiente:

- Entre más grande es el directorio de páginas (tamaño), el tiempo de ejecución de las consultas realizadas al prototipo es mayor. Sin embargo, en esta investigación no se determina cuál es el tamaño óptimo del DP.
- Cuando se realiza una consulta a una sola tabla, la política de remplazo que mejor comportamiento tiene es la política de remplazo propuesta (LLU / Least Likely to Use), las Figuras 6.1, 6.2, 6.3, 6.4, 6.5 y 6.6 muestran los resultados de las pruebas realizadas en un DP de 1000, 2000 y 3000 páginas, los casos 2, 3, 4, y

5 corresponden a una consulta a una sola tabla, donde se puede apreciar que en promedio la política que mejor comportamiento presentó fue la política propuesta LLU, en el caso del DP de 1000 páginas, en términos generales fue mejor sin el uso de la precarga informada, en el caso del uso del DP de 2000 páginas, los tiempos se comportaron de manera similar con y sin el uso de precarga, y en el caso del DP de 3000 páginas los mejores tiempos se presentaron con el uso de la precarga informada.

- Cuando se realiza una consulta a dos tablas, es variable la política de mejor comportamiento debido a que en ocasiones es mejor la política propuesta y en otras las política 2Q (orientada al área de bases de datos), ver Figuras 5.13 y 5.14. Asimismo las Figuras 6.1, 6.2 6.3, 6.4, 6.5 y 6.6 muestran los resultados de las pruebas realizadas en un DP de 1000, 2000 y 3000 páginas, los casos 7, y 8 corresponden a una consulta a dos tablas, para los tres tamaños del DP los mejores tiempos se presentaron con el uso de la precarga informada.

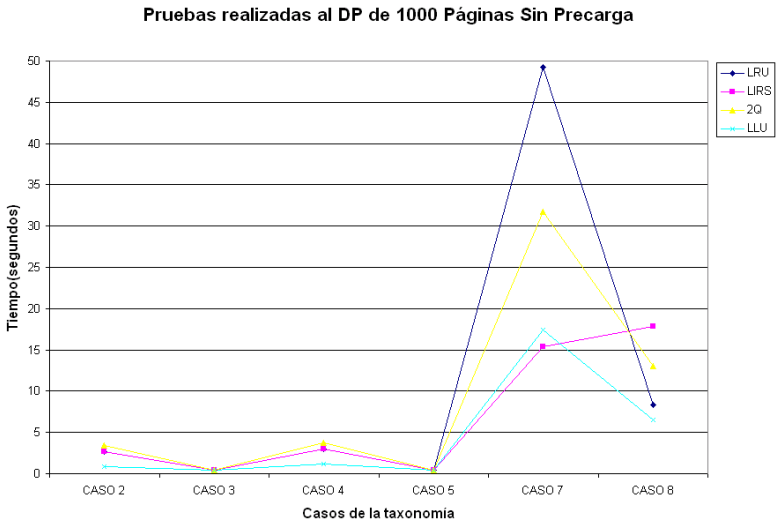

Figura 6.1: Resultados de las pruebas realizadas a los casos de taxonomía de la instrucción SELECT con un DP de 1000 páginas sin el uso de la precarga

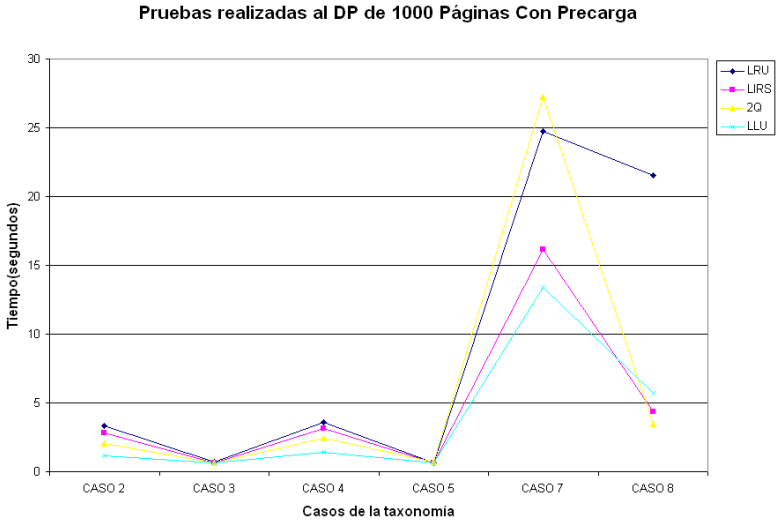

Figura 6.2: Resultados de las pruebas realizadas a los casos de taxonomía de la instrucción SELECT con un DP de 1000 páginas con el uso de la precarga

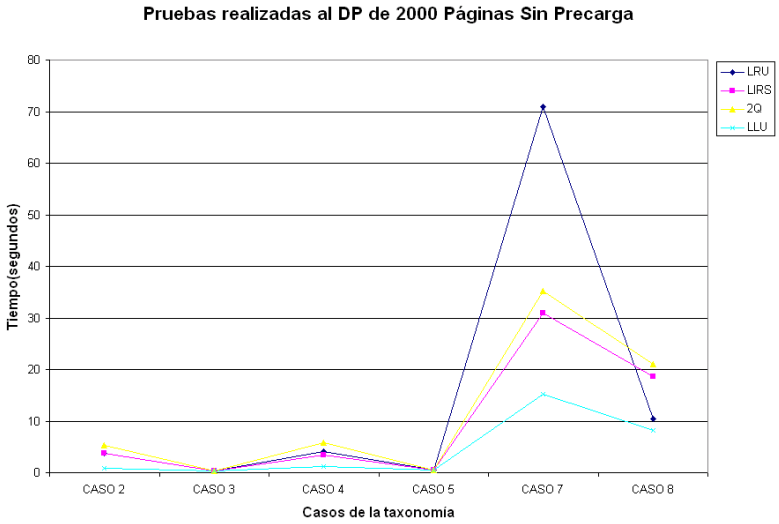

Figura 6.3: Resultados de las pruebas realizadas a los casos de taxonomía de la instrucción SELECT con un DP de 2000 páginas sin el uso de la precarga

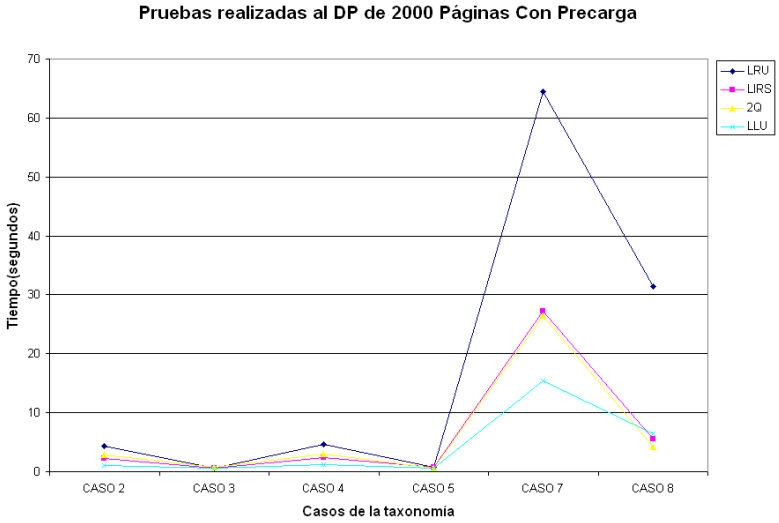

Figura 6.4: Resultados de las pruebas realizadas a los casos de taxonomía de la instrucción SELECT con un DP de 2000 páginas con el uso de la precarga

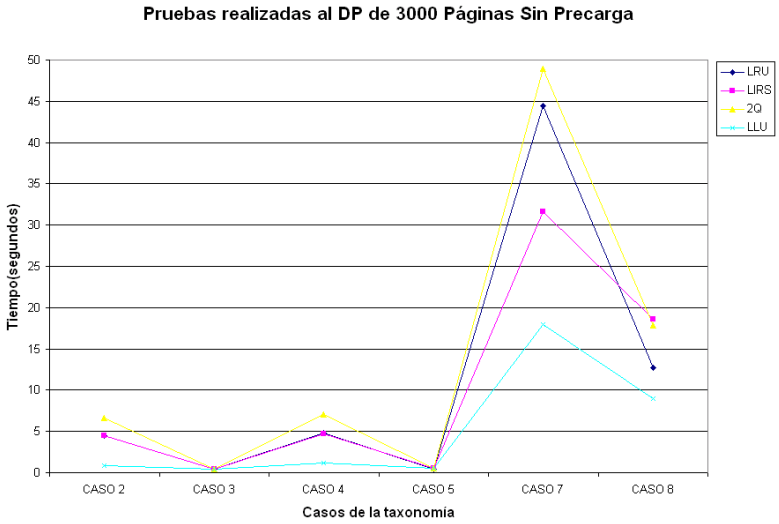

Figura 6.5: Resultados de las pruebas realizadas a los casos de taxonomía de la instrucción SELECT con un DP de 3000 páginas sin el uso de la precarga

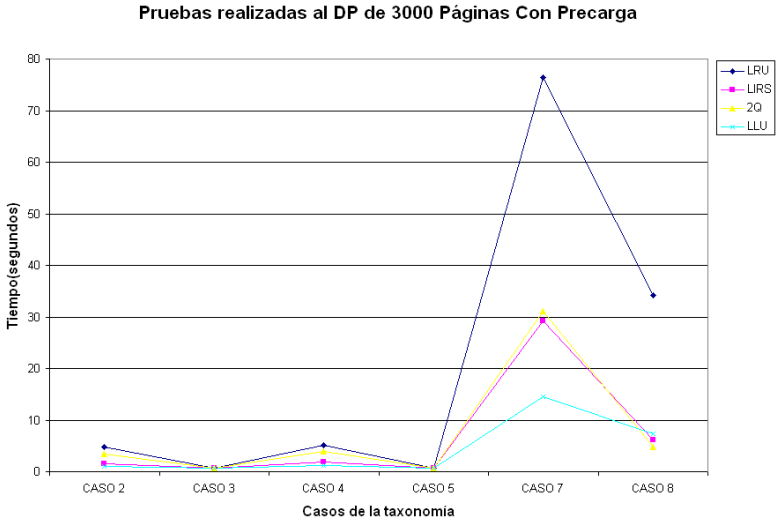

Figura 6.6: Resultados de las pruebas realizadas a los casos de taxonomía de la instrucción SELECT con un DP de 3000 páginas con el uso de la precarga

Todos los objetivos propuestos en esta tesis fueron cubiertos, lo cual nos ayuda a concluir que, de acuerdo a la consulta que se realice, es conveniente utilizar la política de remplazo propuesta en esta tesis (LLU), 2Q ó LIRS.

6.3 Trabajos futuros

- Se propone el uso de la precarga predictiva (estadística) en el directorio de páginas.
- Se propone el uso del DP en memoria compartida, ya que en este trabajo de investigación su uso se limita a sólo una sesión.
- Se propone la implementación del segundo nivel del DP, el cual consiste en utilizar un índice que mejore el acceso de las páginas almacenadas en el DP (se utilizaría solamente con la política de remplazo LLU).

Apéndice A

Anexos

A.1 Taxonomía de la instrucción SELECT de SQL

La clasificación de la instrucción SELECT se realizó mediante un análisis de los casos que se pueden presentar cuando se realiza una consulta en un SMBD. Es importante señalar que se incluye el análisis de la integración de la precarga informada y de las políticas de remplazo que se adecúan mejor para cada caso.

1. Consulta a sólo una tabla, sin cláusula WHERE sin índice y con índice. Dado que la información de todos los renglones de la tabla han de ser entregados, es conveniente usar la precarga para mover a memoria principal los renglones (almacenados en páginas) que serán utilizados en un futuro inmediato. Esto acelera el acceso a los datos debido a que el sistema anticipa la carga de los renglones de modo que ya estarán en memoria principal cuando éstos sean requeridos. Para este caso es conveniente usar el algoritmo de remplazo MRU (Most Recently Used) o LRU (Least Recently Used), debido a que los renglones ya leídos no serán utilizados nuevamente, y por lo tanto, es indistinto qué renglón remover.

2. Consulta a sólo una tabla, con cláusula WHERE sencilla sin índice. Cuando la columna que aparece en la cláusula WHERE no tiene un índice, es necesario verificar cada renglón de la tabla para determinar si satisface 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. Primero es necesario determinar si es conveniente usar el índice, si es así los nodos del árbol (del índice) son cargados conforme son necesitados; en este caso no es posible aprovechar los beneficios de la precarga, debido a que para precargar algún nodo del árbol es necesario conocer su dirección. No obstante, sería conveniente precargar el nodo raíz, ya que este nodo es indispensable para iniciar la búsqueda. Una vez que el nodo hoja es encontrado, es posible acceder al renglón (página) de la tabla. Posteriormente, los nodos adyacentes a la hoja (a la izquierda o derecha, acorde al análisis previo de la instrucción) se deben leer mientras la cláusula WHERE se satisfaga (por ejemplo, columna >valor), lo cual puede arrojar como resultado uno, varios o ningún renglón. Es conveniente usar el algoritmo de remplazo MRU (Most Recently Used) para los nodos del árbol, para intentar mantener en el contenedor los primeros nodos accedidos (los más próximos a la raíz), los cuales son los más probables a ser usados por la consulta. El algoritmo de remplazo LRU (Least Recently Used) puede ser usado para administrar las páginas que contienen los renglones de la tabla, ya que una vez usado un determinado renglón podría ocurrir que algún otro renglón en la misma página sea solicitado posteriormente. Por lo tanto, cuando es imposible saber qué páginas contienen renglones que serán solicitados en un futuro cercano, la página elegida para removerse es indistinta.

4. Consulta a sólo una tabla, con cláusula WHERE compleja, sin índice. Cuando no exista índice cada uno de los renglones de la tabla debe ser evaluado para encontrar cuáles cumplen la condición de búsqueda; por lo tanto, este caso se trata igual al caso 1.

5. Consulta a sólo una tabla, con cláusula WHERE compleja, con índice. La cláusula WHERE debe descomponerse en subconsultas, y si alguna de éstas tiene un índice, utilizarlo siempre y cuando éste sea útil; si existen varios índices, es necesario hacer un análisis para seleccionar el más beneficioso. Si existe un índice y es conveniente utilizarlo, el proceso continúa exactamente como en el caso 3. Es necesario mencionar que los renglones recuperados usando el índice tienen que ser evaluados para determinar si satisfacen las demás condiciones (subconsultas). Si ningún índice puede ser utilizado, este caso será tratado exactamente como el caso 1.

6. Consulta a dos tablas, sin cláusula WHERE sin índice y con índice. Debido a que se tiene que realizar un producto cartesiano, un índice no es útil para esto; en este caso es conveniente utilizar la precarga para acelerar la ejecución. Para realizar el producto cartesiano una de las tablas tiene que ser leída secuencialmente solamente una vez, así los renglones ya leídos no serán utilizadas otra vez, y por lo tanto, es recomendable utilizar el algoritmo de remplazo LRU (Least Recently Used). Concurrentemente, los renglones de la otra tabla se leen secuencialmente, pero estos renglones tienen que ser accedidos proporcionalmente a la cardinalidad de la primera tabla; en este caso es conveniente utilizar el algoritmo de remplazo MRU (Most Recently Used), a menos que sea posible colocar la tabla entera en memoria principal para mejorar el tiempo de respuesta.

7. Consulta a dos tablas, con cláusula WHERE sencilla, sin índice y con índice. Es conveniente crearle un índice a una de las tablas, si éstas no tienen ningún índice útil; en paralelo los renglones de la otra tabla tienen que ser precargados. Si el operador que aparece en la cláusula es $>$, $>=$, $<$, $<=$ o $=$, el proceso debe proceder de la siguiente forma: leer un renglón de la tabla sin índice, obtener posteriormente el dato de la columna que participa en la cláusula WHERE, después con los datos anteriores buscar la primera hoja en el árbol del índice que satisface la cláusula WHERE similarmente al caso 3; de esta hoja explorar a la izquierda o derecha

para encontrar las hojas del árbol que satisfacen la cláusula según el tipo de operador; este proceso se debe realizar para cada renglón de la tabla sin índice. El algoritmo de remplazo que se debe utilizar para las páginas que contienen los renglones de la tabla indizada es el LRU (Least Recently Used), debido a que tienen la misma probabilidad de ser accedidos nuevamente; mientras que para la otra tabla el algoritmo MRU (Most Recently Used) debe ser utilizado, el cual según la literatura es el más eficiente para reuniones. Para el operador <> el proceso procedería similarmente al caso 6.

8. Consulta a dos tablas con cláusula WHERE compleja (con y sin índice). La consulta debe ser descompuesta en subconsultas, tratándose este caso igual que el anterior. Es necesario mencionar que para algunas cláusulas debe efectuarse el producto cartesiano, así que este caso debe ser tratado exactamente como el caso 6.

9. Consulta a tres o más tablas (con y sin cláusula WHERE y con y sin índice). Las primeras dos tablas deben ser procesadas, posteriormente la tabla resultante y una de las tablas restantes deberán procesarse similarmente, y así para el resto de las tablas. Cada par de tablas se debe procesar según lo descrito en los casos 6, 7 u 8 dependiendo de la existencia o no de la cláusula WHERE, una cláusula simple o una cláusula compleja.

Como se puede apreciar en la clasificación anterior, dependiendo del caso que se presente serán las acciones a seguir, ya sea para precargar los datos o para utilizar un algoritmo de remplazo.

Bibliografía

[1] Berg, S., Cache Prefetching, University of Washington. (2004)

[2] Gibson, G., Vitter, J., Report of the Working Group on Storage I/O for Large-Scale Computing. ACM Workshop on Strategic Directions in Computing Research, ACM Press. (1996)

[3] Cao, P., Felten, E., Karlin, A., Li, K., A Study of Integrated Prefetching and Caching Strategies. ACM SIGMETRICS. (1995)

[4] Wang, Z., Cooperative Hardware/Software Caching for Next Generation Memory Systems. ACM SIGMETRICS, University of Massachusets, Beijing University. (2004)

[5] Wang, Z., Burger, D., McKinley, R., Steven, R., Weem, C., Guided Region Prefetching: A Cooperative Hardware/Software Approach. University of Michigan. (2003)

[6] Collins, J., Suleyman, S., Calder, B., Tullsen, D., Pointer Cache Assistant Prefetching. (2002)

[7] VanderWiel, S., Lilja, D., A Compiler-Assisted Data Prefetch Controller. University of Minnesota. (1999)

[8] Mowry, T., Luk, C., Compiler and Hardware Support for Automatic Instruction Prefetching. Carnegie Mellon University Pittsburg. (1998)

[9] Patterson, R., Gibson, G., Ginting, E., Stodolsky, D., Zelenka, J., Informed Prefetching and Caching. (1995)

[10] Glass, G., Cao, P., Adaptative Page Replacement Based on Memory Reference Behavior. University of Wisconsin-Madison. (1997)

[11] Bressan, S., Leng, C., Chin, B. A Framework for Modeling Buffer Replacement Strategies. ACM 2000. (2000)

[12] Butt, A., Gniady, C., Hu, Y. The Performance Impact of Kernel Prefetching on Buffer Cache Replacement Algorithms. ACM SIMETRICS, University of Purdue. (2005)

[13] Pazos, R., Martinez, F., González, B. Estrategias de Precarga y Administración para Sistemas Administradores de Bases de Datos. (2003)

[14] Dahlin, M., Chandra, B., Gao, L., Nayate, A., End-to-end WAN Service Availability. University of Texas at Austin. (2001)

[15] Sybase, I. SQL Anywhere 5.5 Memory Use. (2005)

[16] Dageville, B., Zait, M., SQL Memory Management in Oracle9i. Proceedings of the 28 VLDB Conference, Hong Kong, China, Oracle (2002) 12

[17] Microsoft. Memory Architecture (SQL Server Arquitecture). (2005)

[18] Schumacher, R., Smart SQL-Server Space with Embarcadero Space Analys. Proceedings of the 28 VLDB Conference, Embarcadero Technologies. (2004)

[19] Silberschatz, A., Galvin, P., Gagne, G., Operating System Concept Sexta Edición, John Wiley & Sons. (2002) 34–35

[20] Stonebraker, M. Operating System Support for Database Management. Commun. ACM **24** (1981) 412–418

[21] Choi, J., Cho, S., Noh, S., Min, S., Cho, Y. Characterization of Block Reference Pattern for Flexible and Adaptive Buffer Management Scheme. Computer Engineering Technical Report, Seoul National University. (1999)

[22] Deitel, H., M.: Sistemas Operativos Segunda Edición. Adisson Wesley Iberoamericana. (1990)

[23] Milenkovic, M.: Sistemas Operativos Conceptos y Diseño Segunda Edición. Mc Graw Hill. (1998)

[24] Stallings, W.: Operating Systems Internal and Design Principles Fourth Edition. Prentice Hall, New Jersey. (2001)

[25] Estructura de Computadores. Facultad de Informática UCP. (2003)

[26] Tanenbaum, Andrew, S.: Sistemas Operativos Modernos. Prentice Hall. (2001)

[27] Milenkovic, M.: Sistemas Operativos Conceptos y Diseño Segunda Edición. Mc Graw Hill. (1998)

[28] Maccabe, B., A. In: Sistemas Computacionales Arquitectura y Organización. Mc Graw Hill/Irwin. (1995) 580, 430, 431, 462, 463

[29] Microsoft. Sitio Web de Microsoft. (2004)

[30] Reiter. A Study of Buffer Management Policies for Data Management Systems. (1976)

[31] William, G. Tuel, J. An Analysis of Buffer Paging in Virtual Storage Systems, IBM J. Res. Dev. **20** (1976) 518–520

[32] Abel, P.: Lenguaje Ensamblador y Programación para PC IBM y Compatibles. Tercera. edn. Prentice Hall (1996)

[33] Effelsberg, W., Haerder, T. Principles of Database Buffer Management. ACM Trans. Database Syst. **9** (1984) 560–595

[34] Mancilla, M. Implementación de un Manejador de Índices en un Sistema Administrador de Bases de Datos Relacionales. Tesis de Licenciatura. Instituto Tecnólogico de Cd. Madero. (2005)

[35] Gómez, G. Manejador de Índices Para un Administrador de Bases de Datos Distribuidas Relacionales. Tesis de Maestría. Instituto Tecnólogico de León. (2000)

[36] Capistrán, E. Implementación de un Administrador de Memoria para un Manejador de Bases de Datos Relacionales. Tesis de Maestría. Instituto Tecnólogico de Cd. Madero. (2002)

[37] Sokolinsky, L. LFU-K: An Effective Buffer Management Replacement. Algorithm Database Systems for Advances Applications, 9th International Conference, Jeju Island, Korea (2004) 670–681

[38] Castro, I., Hernández, S. Complementación del Lenguaje de Manipulación del SiMBaDD. Tesis de Licenciatura. Instituto Tecnólogico de Cd. Madero. (2002)

[39] Riebs, A., Compaq, Computer, C. The Open Source Database Benchmark. (2005)