COLEGIO DE POSTGRADUADOS

INSTITUCIÓN DE ENSEÑANZA E INVESTIGACIÓN EN CIENCIAS AGRÍCOLAS
CAMPUS MONTECILLO

POSTGRADO EN SOCIOECONOMÍA, ESTADÍSTICA E INFORMÁTICA
CÓMPUTO APLICADO

ELABORACIÓN DE UNA BASE DE DATOS HÍBRIDA ESTRUCTURADA POR PALABRAS CLAVE

FRANCISCO JAVIER HERNÁNDEZ RIVAS

T E S I S

PRESENTADA COMO REQUISITO PARCIAL

PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS

MONTECILLO, TEXCOCO, ESTADO DE MÉXICO

2018

COLEGIO DE POSTGRADUADOS

CARTA DE CONSENTIMIENTO DE USO DE LOS DERECHOS DE AUTOR Y DE LAS REGALÍAS COMERCIALES DE PRODUCTOS DE INVESTIGACIÓN

En adición al beneficio ético, moral y académico que he obtenido durante mis estudios en el Colegio de Postgraduados, el que suscribe, "**Francisco Javier Hernández Rivas**", alumno de esta Institución, estoy de acuerdo en ser partícipe de las regalías económicas y/o académicas, de procedencia nacional e internacional, que se deriven del trabajo de investigación que realicé en esta institución, bajo la dirección del Profesor "**Dr. Juan Ricardo Bauer Mengelberg**", por lo que otorgo los derechos de autor de mi tesis "**ELABORACIÓN DE UNA BASE DE DATOS HÍBRIDA ESTRUCTURADA POR PALABRAS CLAVE**", y de los productos de dicha investigación al Colegio de Postgraduados. Las patentes y secretos industriales que se puedan derivar serán registrados a nombre del Colegio de Postgraduados y las regalías económicas que se deriven serán distribuidas entre la Institución, el Consejero o Director de Tesis y el que suscribe, de acuerdo a las negociaciones entre las tres partes, por ello me comprometo a no realizar ninguna acción que dañe el proceso de explotación comercial de dichos productos a favor de esta Institución.

Montecillo, Texcoco, Edo. de México a 26 de noviembre de 2018

Francisco Javier Hernández Rivas

Dr. Juan Ricardo Bauer Mengelberg

La presente tesis titulada: ***"*ELABORACIÓN DE UNA BASE DE DATOS HÍBRIDA ESTRUCTURADA POR PALABRAS CLAVE**" realizada por el alumno: "**Francisco Javier Hernández Rivas**" bajo la dirección del Consejo Particular indicado, ha sido aprobada por el mismo y aceptada como requisito parcial para obtener el grado de:

MAESTRO EN CIENCIAS
SOCIOECONOMÍA, ESTADÍSTICA E INFORMÁTICA
CÓMPUTO APLICADO

CONSEJO PARTICULAR

CONSEJERO _____
Dr. Juan Ricardo Bauer Mengelberg

ASESOR _____
Dr. David Hebert Del Valle Paniagua

ASESOR _____
M.en C Edgar Ramírez Galeano

Montecillo, Texcoco, Estado de México, diciembre de 2018

ELABORACIÓN DE UNA BASE DE DATOS HÍBRIDA ESTRUCTURADA POR PALABRAS CLAVE

Francisco Javier Hernández Rivas, M.C.

Colegio de Postgraduados, 2018

RESUMEN

El proyecto de investigación que se describe consiste en la construcción de un manejador de una base de datos orientada a consultas, y no a un uso transaccional. La operación fundamental que se contempló en el diseño del paquete de SW es el armado de un subconjunto de los datos que satisfagan ciertos criterios de búsqueda. El mecanismo que dotaría estas operaciones de flexibilidad y celeridad consiste en el uso de palabras clave, o descriptores agregados a los elementos de datos, que se indican provistas de un contexto que proporciona su significado. Las búsquedas se implementan vía operaciones booleanas entre subconjuntos de datos, cada uno de los cuales satisface un criterio formulado a partir de las palabras clave. El diseño de esta base de datos no es nuevo; se han elaborado versiones anteriores, que utilizaban diversas estructuras para sus datos. Los tiempos de respuesta de búsquedas complejas en conjuntos muy numerosos de datos, del orden de los cientos de millones de ellos, fueron considerablemente menores que las resultantes del uso de bases de datos relacionales. Sin embargo, aparentemente el uso de otras estructuras podría proporcionar tiempos de respuesta más breves. En la versión nueva del SW, el énfasis recae precisamente en el aspecto de encontrar las mejores estructuras para cada una de las entidades de datos.

Palabras clave: bases de datos, llave-valor, estructuras de datos, bitmaps

DEVELOPMENT OF A HYBRID DATABASE STRUCTURED BY KEYWORDS

Francisco Javier Hernández Rivas, M.C.

Colegio de Postgraduados, 2018

ABSTRACT

The research Project described consists in the construction of a database management system to implement queries as opposed to transitional use. Thus, the main operations addressed were searches, that is obtaining subsets of the data that satisfy certain search criteria. In order for these operations to be both flexible and fast, keywords by context are added to the data elements, where the context indicates the meaning of the word. Searches are implemented through Boolean operations between subsets of the data, each one of which satisfies a criterion based on keywords. The design of this database is not new; previous versions were developed using several data structures to store the data, especially the keywords. Response times for complex searches in large datasets, containing hundreds of millions of elements, were significantly shorter than those resulting from the use of a relational database. However, apparently the use of other data structures could result in even shorter response times. The emphasis of the new version of the software is precisely finding the best data structures for each of the data entities involved.

Keywords: data bases, key-value, data structures, bitmaps

AGRADECIMIENTOS

Dedico esta tesis a todos los que creyeron en mí, a toda la gente que me apoyo, a mis amigos y familiares y ha esta institución que me ha formado, pero en especial se la dedico a mis padres y esposa, a quienes agradezco de todo corazón por su cariño, comprensión y consejos. En todo momento los llevo conmigo.

DEDICATORIA

PROFESORES DEL PROGRAMA

Por haberme ayudado con su conocimiento en esta etapa de mi formación profesional.

AL CONSEJO NACIONAL DE CIENCIA Y TECNOLOGÍA (CONACYT)

Por el apoyo económico que me otorgó para poder llevar a cabo mis estudios de Maestría.

AL COLEGIO DE POSTGRADUADOS

En especial al Programa de Cómputo Aplicado, por todas las enseñanzas y las experiencias obtenidas y por la amistad recibida de todos los que en él laboran.

Un agradecimiento muy especial a mi consejero Dr. Juan R. Bauer Mengelberg por compartir conmigo sus conocimientos, por brindarme sabios consejos, por su tiempo y dedicación y sobre todo por su paciencia le estoy muy agradecido.

CONTENIDO

LISTA DE FIGURAS

LISTA DE CUADROS

CAPÍTULO 1. INTRODUCCIÓN

1.1 Descripción general

El proyecto de investigación en el cual se enmarca el descrito en esta tesis es la construcción de un manejador de una base de datos denominada DBB orientada a consultas, y no a un uso transaccional (Silberschatz et al., 2014). De ese modo asemeja una bodega de datos (Inmon, 2005) en los aspectos contemplados en cuanto al uso de la información contenida en acervo de datos. En particular, la operación fundamental que se contempló en el diseño del paquete de Software es la búsqueda de información, es decir, el armado de un subconjunto de los datos que satisfagan ciertos criterios de inclusión (también llamados criterios de búsqueda). El mecanismo que permitiría estas operaciones con flexibilidad y celeridad consiste en el uso de palabras clave, o descriptores, de los materiales. Estas palabras clave se indican provistas de un contexto que proporciona su significado o interpretación.

Cabe señalar que el diseño de esta base de datos no es nuevo. Se han elaborado versiones anteriores, que utilizaban diversas estructuras para sus datos. A pesar de haber mostrado la eficiencia de búsquedas basadas en las palabras clave por contexto, siempre se pensó que el uso de otras estructuras podría proporcionar tiempos de respuesta más breves. En particular, a partir de trabajo en el cual se compararon tiempos de respuesta de una consulta tipo inteligencia de negocios, y que mostró las ventajas del modelo propuesto, (Gonzalez Espinosa, 2011) se iniciaron esfuerzos por disminuir aún más los tiempos de respuesta de las consultas formuladas.

Por lo tanto el énfasis de la elaboración de una nueva versión recayó precisamente en el aspecto de encontrar las mejores estructuras para cada una de las entidades de datos.

Se dividió el proyecto en varios sub-proyectos, siguiendo el método de impartido en los cursos de Sistemas de Información del Dr. Bauer.

Fase 1: Estudio de la oportunidad de crear una base de datos enfocada al uso de palabras clave para formular consultas tipo “búsquedas”.

Fase 2: La determinación de la funcionalidad esperada; la facilidad o eficacia en cuanto a las búsquedas; formular el modelo conceptual de datos, la lista de funciones incluyendo la flexibilidad proporcionada en relación con cada una de las funciones; y los procedimientos que sean necesarios.

Fase 3: Formular el modelo técnico con las entidades para almacenar las diversas entidades de datos; definir los procesos, programas y un plan de pruebas del producto.

Fase 4: El desarrollo del producto.

Fase 5: Formulación de actividades y materiales para el uso del paquete.

El tema de esta tesis corresponde a la Fase 3: seleccionar las estructuras en las cuales se almacenarían los datos del acervo, en base al modelo conceptual y a las funciones que se implementan vía el modelo técnico. Es importante señalar que cuando comienza esta fase, se recibe el modelo de datos conceptual (las entidades a almacenar y las relaciones entre ellas), la lista de funciones e información adicional necesaria para elaborar el sistema en las siguientes fases. De ese modo, en esta tesis no se presentan elementos de las dos fases anteriores, excepto sus resultados, es decir el modelo de datos conceptual, ya validado contra la lista de funciones.

Para elaborar el modelo técnico, es decir, las estructuras que se usan para almacenar los datos, se consideran diversas estructuras que podrían resultar aplicables, y se comparan con varios criterios incluyendo considerandos de eficiencia computacional – típicamente tiempos de respuesta que proporcionarían en diversas funciones, y el espacio en disco y en memoria que ocuparían. Es

importante señalar que en ningún caso se incluirían criterios de comparación relacionados con la programación, ni complejidad ni extensión de los programas.

La base de datos resultante debería satisfacer los siguientes requisitos:

- Que permita introducir datos de diversos tipos de objetos en cuanto al tipo de información que requiere su descripción. Por ejemplo se puede incluir en el mismo acervo información sobre universidades, edificios, personal, cursos ofrecidos, alumnos, libros, etc. Lo que serían registros de una base de datos en el DBB se denominan UBI (Unidad Básica de Información). Esto condujo a que las UBIs se definieran como objetos de una clase, pudiendo haber varias de éstas.
- Que permita incluir descriptores adicionales de los objetos, además de los campos indicados por la clase a la que pertenecen, sin límite del número de tales descriptores.
- Los descriptores tendrán roles parecidos a las palabras clave, es decir, podrán ser incluidos en criterios de búsqueda.
- El diseño debe contemplar que el DBB es multi-base. Esto significa, como ilustra la Figura 1, que puede haber, en un mismo SITE, varias bases de datos independientes entre sí.

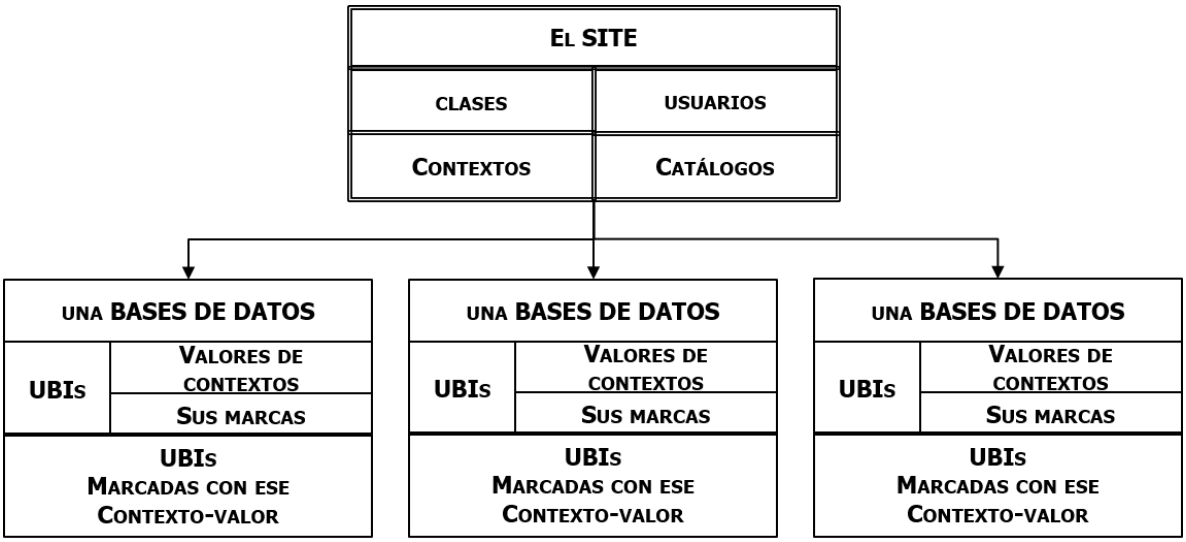


Figura 1 En el SITE puede haber más de una base de datos, cada una contiene sus propios elementos independientes, elaboración propia.

Para permitir el uso eficiente de estas palabras clave, se las asocia a un “contexto”, mismo que indica la interpretación de la palabra. Por ejemplo la palabra clave “DULCE” puede ser un sabor, un nombre, una palabra de un título de libro, etc. Debido a eso se formula el concepto de una “MARCA” de una UBI: es un triplo (Contexto, valor, número de UBI). De ese modo se puede solicitar el conjunto de UBIs que están marcados con ese par (contexto, valor). A su vez, se pueden hacer operaciones booleanas entre tales subconjuntos para incorporar varios criterios de búsqueda. Esta es la funcionalidad fundamental del DBB: en base a ciertos criterios formulados en términos de pares de (contexto, valor) encontrar el subconjunto de UBIs que satisfacen todas las condiciones impuestas vía los criterios de búsqueda. Para definir una consulta se elabora una fórmula que consiste de operandos y operaciones. Estas últimas serán uniones (OR), intersecciones (AND), unión exclusiva (XOR) y negaciones (NOT).

1.1.1 El modelo conceptual del DBB

El proyecto inicia con el modelo conceptual, es decir, las entidades de datos y las relaciones entre ellas. Este modelo permitió definir y limitar la naturaleza y extensión

del proyecto (la determinación de las estructuras de datos que se emplearían). En la Figura 2 Se presenta un diagrama con las entidades y más adelante se explican algunos elementos y sus relaciones.

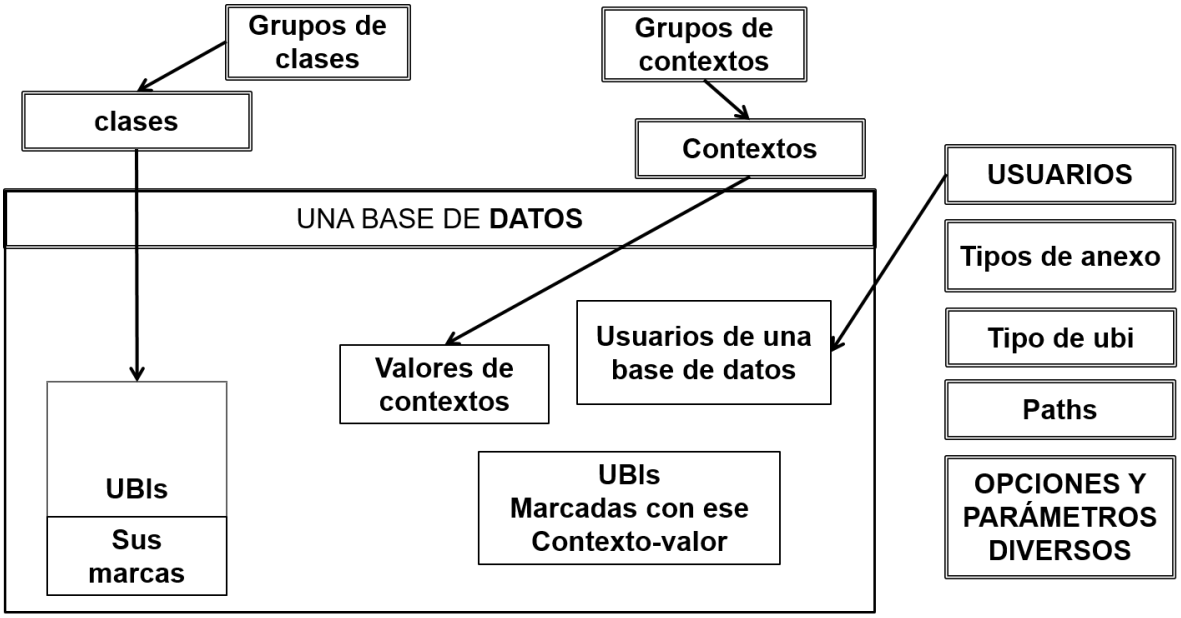


Figura 2 El modelo de datos conceptual del DBB, indica que elementos pertenecen a una base de datos y cuales al SITE, elaboración propia.

Las entidades señaladas con un doble marco son comunes a todas las bases de datos. Las que tienen un marco sencillo se almacenan por separado para cada base de datos. Observe que son las mismas que corresponden a una base de datos en especial.

Las entidades del modelo conceptual

Se describen brevemente las entidades de datos fundamentales del modelo de datos.

Se definen contextos y sus valores: los contextos se pueden agregar cuando surja la necesidad de hacerlo: los valores se actualizan cuando haya una marca de un contexto cuyo valor no estaba anteriormente. Mientras que los contextos se definen

para el SITE (es decir para todas las bases de UBIs que se utilicen), los valores se incluyen a partir de las UBIs de cada base de datos.

Se definen CLASES. Las UBIs son instancias de una clase. La clase indica cómo se interpretan los campos variables de las UBIs (esto se detalla más adelante) además de ciertos aspectos en cuanto al marcado de las UBIs de la clase (el término “marcado” significa agregar palabras clave a la UBI).

Se introducen las UBIs (los datos incluidos en la base). Se agregan descriptores a las UBIs (son pares contexto-valor).

Se preparan las marcas de las UBIs para consultas (las marcas de cada contexto-valor).

Como apoyo a los datos sustantivos, se formulan catálogos (todos ellos se explican más abajo). Por ejemplo, el tipo de anexo describe el archivo o liga descrito por una UBI, mientras que el tipo de UBI permite encontrar las UBIs de cierto tipo.

Como se dijo anteriormente, la versión que se desarrolla del DBB es multi-base. Esto significa que hay un SITE común, pero se pueden definir diversas bases de datos para un mismo SITE. Cada uno de estas bases tendrá sus propias UBIs, Valores de contextos, sus marcas y un catálogo propio de permisos de los usuarios de esa base.

Todo el sistema está protegido por un control de acceso a funciones tipo RBAC (por roles). Para ello se almacenan los usuarios autorizados para el uso del sistema. Para cada usuario se indican las bases de datos a las cuales tiene acceso (esta relación no está reflejada en el diagrama).

1.2 Objetivo del proyecto descrito en esta tesis

Como ya se dijo, el objetivo del proyecto que se describe es determinar la estructura que emplea el DBB para cada una de las entidades señaladas. En particular, cómo se almacenan las UBIs y sus descriptores, los contextos, los valores de los contextos y para cada valor, los triplos (contexto, valor, UBI) que son los que usan los criterios de búsqueda; pero también los elementos complementarios señalados en el modelo conceptual presentado anteriormente.

1.3 Organización de esta tesis

En la sección de Materiales, se presentan primero algunos elementos teóricos que se utilizaron en la investigación. En particular se describen y comparan algunas estructuras de datos; además de las comúnmente utilizadas, se presentan algunas estructuras especiales que se incluyeron como posibilidades para almacenar ciertos datos.

A continuación se describe el método utilizado para comparar los efectos del uso de ciertas estructuras con alternativas contempladas. Se describen los procesos creados para poder cuantificar algunos tiempos de respuesta y actividades de actualización de datos.

Se formularon diversos procesos (simulaciones) para determinar ciertos comportamientos de algunas estructuras para los diversos usos que se necesitarían por las entidades. Estos procesos de describen en un capítulo por separado.

Se dedicó un capítulo a cada una de las entidades que se deben almacenar. Comenzando por los contextos y sus valores, y las clases, se estudiaron las estructuras posibles para las UBIs mismas, sus descriptores (contexto-valor) y estas mismas marcas para el uso por las consultas. Además se seleccionaron estructuras para almacenar diversos catálogos, los usuarios autorizados para usar el sistema y algunos parámetros y opciones activas en una instancia de uso del paquete.

En otro capítulo, se resumen las estructuras seleccionadas en conjunto. El trabajo concluye con algunas conclusiones y las referencias bibliográficas.

En un anexo (presentado en un disco adjunto a la tesis) se incluyeron los programas utilizados para la comparación de las estructuras contempladas para las entidades principales. Además, se incluye un directorio típico de una base de datos con las UBIs, y otro que contiene las entidades adscritas al SITE mismo.

Se señala que no se incluyeron los programas fuente (por su extensión) pero sí los programas como aplicaciones y las librerías creadas que utilizan.

ANEXOS

- Las simulaciones utilizadas: los programas con las cuales se realizaron (en modo fuente).
- Un directorio típico de un SITE.
- Los programas ejecutables (aplicaciones).
- Un directorio típico de una base de datos.

CAPÍTULO 2. MATERIALES Y METODOS

2.1 Materiales

2.1.1 Base de datos relacional

Una base de datos relacional consiste en un conjunto de tablas, a cada una de las cuales se le asigna un nombre exclusivo, cada fila de la tabla representa una relación entre un conjunto de valores (Silberschatz et al., 2014).

Los componentes básicos de una base de datos definidos por (Herranz Gómez et al, 2014)son los siguientes:

- **Tablas**. Una base de datos relacional puede contener una o varias tablas donde se estructuran los datos. Las tablas no pueden repetirse dentro de la misma base de datos.
- **Columnas**. Subdivisiones lógicas que conforman una tabla.
- Registros. Una tabla está formada por el conjunto de registros.
- **Relaciones entre tablas.** Las relaciones entre tablas se establecen mediante claves primarias y claves foráneas.
- **Clave prima**ria. Cada registro posee una única clave primaria, y son claves que identifican unívocamente a un registro.
- **Clave foránea**. son referencias colocadas en las tablas hijas de las relaciones entre tablas. No relacionales.

2.1.2 Estructuras típicas

Archivos en disco

Un archivo es un conjunto de bytes almacenados en un dispositivo magnético. Ejemplos son programas textos, canciones, videos, archivos producidos por paquetes de Software. Físicamente no tienen estructura ni significado: consisten en un conjunto de bits, es decir, cero y unos.

En cambio, tienen estructura y significado en el aspecto lógico, es decir, si se leen con un software apropiado (típicamente pero no exclusivamente con los que fueron creados) se recupera la información que se grabó en el archivo.

Como se almacenan muchos archivos en un mismo dispositivo, se los provee de un nombre y una extensión, que indica el tipo de datos que contienen y por ende, cómo se interpretan.

La única manera de almacenar datos en una computadora es vía los archivos en disco. Para usar un archivo en un caso particular, se dispone de tres modos de hacerlo (y recuperar la información). Se usa la terminología “Access Mode”, es decir, como se recuperan diversos sectores de las cadenas de bytes.

- Modo texto
- Modo binario
- Acceso random (aleatorio)

Modo binario

Los datos se recuperan por conjuntos de bytes. Se indica una variable (que puede ser de un tipo definido por el usuario, por ejemplo un arreglo de números) o simplemente un conjunto de bytes - un número, una cadena de bytes que contiene una palabra, nombre, etc.

Para recuperar información de un archivo en modo binario, se especifica lo que se desea leer (obtener): esto siempre resulta en un número determinado de bytes. Se puede indicar la posición (relativa al principio del archivo) del byte donde comienza el dato que se desea recuperar. Naturalmente se debe conocer esta posición para recuperar el dato deseado. También se pueden leer varios datos uno tras otro; si están uno tras otro en el archivo, se indica que se desea leer los siguientes bytes: esto en ocasiones se hace en forma implícita, es decir, no se indica el byte inicial y el sistema supone que se trata del siguiente al último procesado.

Para grabar información en un archivo, una vez más se usan variables (que pueden ser de cualquier tipo, incluyendo tipos definidos por el usuario, es decir, estructuras).

Se envían al archivo los bytes correspondientes a la variable y se indica la posición en la cual se desea grabarlos (el número de byte relativo al inicio del archivo).

Como se dijo, para recuperar información de un disco es necesario conocer lo que se almacenó, y cómo (es decir, para cada elemento de datos, la posición y el número de bytes). Hay situaciones en las cuales se conocen los tamaños de todos los datos grabados; en otras, especialmente cuando el tamaño puede variar de un uso a otro de la misma variable, se tiene que registrar en algún lado lo que se grabó.

Modo Random (aleatorio)

Se trata de un archivo en disco, excepto que se decide dividirlo en “registros”. Es importante señalar que se trata de registros lógicos; no hay en el archivo alguna indicación del fin de un registro y el comienzo del siguiente.

Cuando se crea el archivo se indica la longitud de cada uno de estos registros (en bytes). Ahora se pueden grabar y leer los registros en una única lectura, es decir, se pueden definir variables que representen estos registros en memoria y leer o escribir esos registros.

Para entender cómo funciona el acceso aleatorio, se puede imaginar el archivo como un conjunto ordenado de segmentos (que son los registros) escritos al disco uno tras otro. El modo aleatorio permite usar estos segmentos con la referencia a su posición en el archivo (es decir, los “Numera” lógicamente). En el archivo mismo no hay tal numeración ni división en segmentos. Todo es “lógico”, es decir, se hace por programa.

Otros aspectos de los archivos

Para usar un archivo, se lo “abre”: esto significa que se establece una conexión entre el archivo en disco y el programa mediante el cual se leerán datos o se le graba información.

Se indica cuales operaciones hace el programa con ese archivo:

- READONLY: El programa usa el archivo exclusivamente para leerlo. Ejemplos son los reproductores de archivos de audio o video.
- OUTPUT: Otros, en cambio, desean solamente grabar información (por ejemplo una grabadora, y muchos otros programas).
- INPUT-OUTPUT: Lo usan programas que desean hacer ambas operaciones.
- APPEND: sólo permite agregar información al final del archivo. No se pueden leer datos, y no se indica la posición de lo que se graba puesto que siempre será el byte siguiente al último byte de los que ya están grabados. Las bitácoras frecuentemente se implementan con este tipo de uso del archivo.

Estructuras de datos

En todas las aplicaciones que se pueden ejecutar en una computadora intervienen como elementos principales LOS DATOS. Estos se usan vía VARIABLES; éste es el término fundamental de la computación. Hay variables simples, que consisten de un único elemento (por ejemplo, un número entero, uno decimal, una cadena de caracteres que se interpretan como un texto, etc.).

Cuando se trata de almacenar un conjunto de datos – típicamente varios con la misma estructura o relacionados de algún modo entre sí, por ejemplo, una lista de los alumnos de una clase, se usa lo que se ha denominado una estructura de datos. Por ejemplo se puede definir un tipo de variable FAMILIA, que está compuesta por: Apellido, nombre del padre, nombre de la madre, nombres de los hijos, etc.

Una definición sencilla sería que una estructura de datos es un modo de organizar la información en memoria que permite un uso eficiente de los datos, y que preferiblemente ocupe el menor recurso de espacio en memoria.

El objetivo de esta sección es dar una descripción de algunas de las estructuras que pueden ser utilizadas. Como se verá, no hay una estructura para una situación dada, sino se pueden utilizar varias para los mismos datos. La decisión de cuál de estas utilizar debe basarse en su eficiencia comparada con las restantes. Es importante señalar que no hay una estructura óptima o buena: todas lo son cuando el tipo de información que se almacena en ellas aprovecha sus ventajas comparadas con las demás.

Para comparar las estructuras para algún uso, se toman en cuenta:

- Eficiencia en cuanto a espacio en disco.
- Velocidad de grabación y recuperación.
- Eficiencia de las operaciones que se hacen con los datos.

Hay otros factores que pueden determinar que conviene usar una determinada estructura, pero no deberían intervenir en la decisión:

- Dificultades de implementación (programación).
- Disponibilidad de programas necesarios.
- Complejidad de los algoritmos en situaciones en las cuales es probable que terceros deban hacer cambios.
- Gusto del programador o diseñador (prefiere esa estructura).
- Y finalmente, aspectos de costos cuando se usan productos de Software comprados o alquilados.

Una estructura de datos es una colección de datos del mismo tipo que presentan una organización y operaciones que las definen. Cada estructura de datos puede ser expresada mediante un conjunto de reglas y relaciones.

Arreglo

Es un conjunto de datos del mismo tipo almacenados de manera continua en memoria; cada uno de sus elementos se distingue entre ellos por uno o varios índices, y este determina la posición en la que se encuentra el elemento en el arreglo como se muestra en la Figura 3. Un arreglo tiene una dimensión, que puede ser 1 -

llamado unidimensional o vector; 2 - llamado bidimensional (matriz), o más de 2; en este caso se dice que el arreglo es multidimensional.

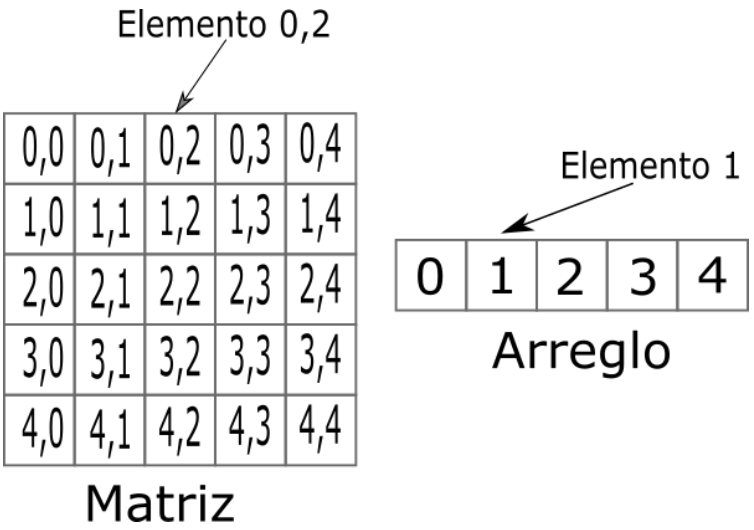


Figura 3 Posición de un elemento en un vector y una matriz dados por cada uno de sus índices, elaboración propia.

Arreglo unidimensional o vector.

La declaración de un vector en vb.net se realiza de la siguiente manera:

```
Public vector (10) As Int32 donde INT32 indica un entero de 32 bits
```

Esto supone que se reservan en memoria 11 segmentos de 4 bytes cada uno de manera contigua; los lenguajes de programación en general consideran el espacio 0 como un adicional a los declarados y sus índices van del 0 hasta el 10.

La manera de recorrer un vector puede ser de manera secuencial mediante ciclos como es "**for"** en donde el contador se convierte en el índice para recorrer el vector y de esta manera realizar la operación necesaria sobre este (asignación, recuperación, etc.).

```
For i as Integer = 0 To 10
      Vector (i) = i + 10
Next
```

Arreglo bidimensional o matriz

Es un arreglo bidimensional: al igual que el vector es un conjunto de elementos del mismo tipo y sus componentes están identificados por dos índices; en Visual Basic, el primero indica el número de fila y el segundo la columna.

La manera en que son almacenados los datos en memoria no es de la forma rectangular en la que se representa tradicionalmente una matriz, sino como un conjunto de variables enteras. Para usar un elemento en particular, el sistema calcula su “posición” (dirección en memoria) como:

Sea la matriz M (8, 20) as Integer. Para accesar el elemento (6, 5) el sistema calcula su posición como

```
Pos de M (0,0) + 4 * [(6 * 20) + 5]
```

Búsqueda en un vector

Esta operación consiste en encontrar un valor dentro del vector y retornar la posición en la que se encuentra en el mismo; si no está el valor buscado, devuelve un valor inválido (por ejemplo -1) se o genera un mensaje de error.

La búsqueda de un valor en un vector ordenado puede realizarse de 2 maneras: secuencial y con búsqueda binaria:

Búsqueda secuencial

Consiste en recorrer cada uno de los elementos del vector de inicio a fin, comparándolos con el elemento que se desea encontrar. Cuando los elementos están ordenados, se presentan 3 tipos de condiciones de paro para la búsqueda:

- Se encontró el elemento, en este caso se retorna el índice en el que se encuentra alojado el elemento.

- Si la lista está ordenada en forma ascendentes, la operación termina cuando ha encontrado el primer elemento mayor al que se busca, de tal forma en que no es necesario seguir recorriendo el vector ya que todos los posteriores serán mayores; esto implica que el valor buscado no está en el arreglo.
- Se llegó el final del arreglo sin encontrar el valor buscado, se indica esta condición.

Búsqueda binaria

Esta búsqueda se realiza en vectores ordenados. Este método es el que se utiliza posteriormente para insertar un elemento en un vector ordenado, tema que se aborda en la sección dedicada a ese tema.

Para la implementación de este algoritmo de usan básicamente 3 índices que llamaremos: izquierda (posición del elemento menor), derecha (posición del elemento mayor) y centro (posición obtenida como promedio del izquierdo y el derecho).

El algoritmo de la búsqueda binaria se basa en los siguientes pasos:

1. Examinar el elemento contenido en el índice Centro, si el elemento es el buscado, la búsqueda termina.
2. En caso de no encontrarlo, de determina si el elemento buscado se encuentra a la izquierda o derecha del elemento central y se asignan los nuevos índices de la siguiente manera.
    a. El elemento buscado es menor al centro: se mantiene el índice izquierda y el centro pasa a ser la nueva derecha.
    b. El elemento buscado es mayor al centro: se mantiene el índice derecha y el centro pasa a ser la nueva izquierda.

    En ambos casos se calcula nuevamente el centro con el promedio de izquierda y derecha.

La búsqueda binaria consiste en realizar los 2 pasos anteriores tantas veces como sea necesario, pero igual presenta condiciones de paro:

- El elemento buscado es menor al primer índice izquierda o mayor al primer índice derecha, de esta manera sabemos que el elemento buscado se encuentra fuera del rango de los elementos contenidos en el vector.
- Los índices izquierdos y derechos ya no cambian en cada una de las iteraciones.

Comparativa búsqueda binara con búsqueda secuencial

Para mostrar la diferencia entre ambos tipos de búsqueda se utilizara el vector de la Figura 4 como ejemplo, y se buscara el número **55.**

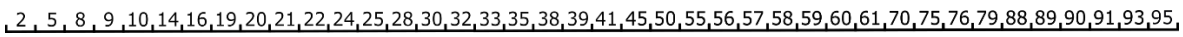


Figura 4 Vector con elementos que se utilizaran para ejemplicar los tipos de búsqueda, elaboración propia.

En el caso de la búsqueda secuencial, como se mencionó anteriormente se realiza la búsqueda elemento por elemento. En este caso fue necesario realizar 24 iteraciones para encontrar el número deseado. La búsqueda binara se muestra en la Figura 5

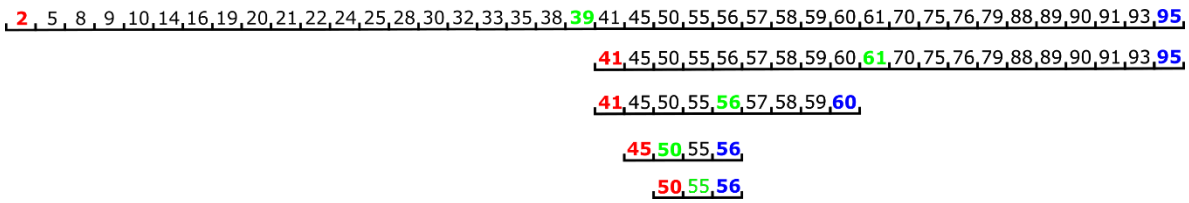


Figura 5 Demostración de cómo se realza una búsqueda binaria, en cada iteración se reduce a la mitad el número de elementos en el que se busca, elaboración propia.

Como se puede observar con este algoritmo se encontró el número en tan solo 5 iteraciones. Esto ejemplifica las ventajas que tiene la búsqueda binaria comparada con una búsqueda secuencial; la reducción del número de operaciones es muy significativa.

Operaciones con listas ordenadas

Si la estructura es un Vector cuyos elementos están ordenados (ya sea en forma ascendente o descendente) las operaciones se basan en búsquedas binarias, explicadas anteriormente.

Para insertar un valor nuevo, se ubica su posición buscando el primer elemento del vector mayor que el valor a insertar. Se crea un espacio entre este valor (su índice) y el anterior, y se pone el valor nuevo en la posición creada. Para eliminar un valor, se busca el valor y se comprime el arreglo.

Observación importante. Se puede ampliar el arreglo agregando una posición al final y recorriendo uno por uno los elementos mayores al nuevo a la posición siguiente a la que ocupaban. Esto hacía que el uso de estas listas resultara “caro”, por el número de operaciones que implicaba. El número de operaciones se reduce al efectuar esta operación mediante un copiado de memoria (se copian los elementos a desplazar a la siguiente posición, tras redimensionar el arreglo si fuera necesario).La misma técnica se usa para eliminar un valor del arreglo.

Árboles

Un árbol es una estructura que implica una jerarquía, en la que cada elemento está unido a otros por debajo de él. Tiene la particularidad de que cada elemento puede tener más de un “siguiente”, aunque un solo antecedente o padre. Solo hay un único nodo raíz. Un árbol es un nodo raíz del cual cuelgan N (o ningún) nodos hijos, cada uno de los cuales es a su vez un árbol, (Weiss, 2014).

Árbol binario

(Weiss, 2014) define un árbol binario como un árbol en el que ningún nodo puede tener más de dos hijos. Figura 6 muestra que un árbol binario consta de una raíz y dos subárboles, nodo izquierdo y nodo derecho, ambos posiblemente vacíos. El

nodo izquierdo debe contener un valor menor que la raíz mientras que el nodo derecho siempre será mayor. Un nodo que no tiene ningún hijo (ambos apuntadores son nulos) frecuentemente se denomina “hoja”.

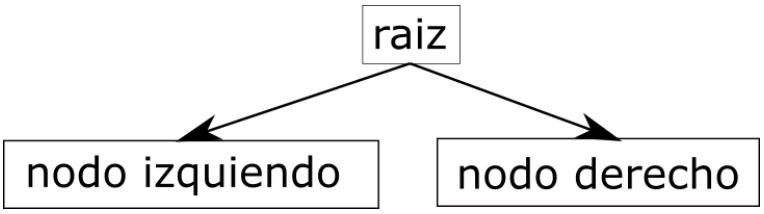


Figura 6 Árbol binario con solo 2 hijos ambos son hojas, elaboración propia.

Árboles B (B- trees)

El árbol-B se definió por primera vez en (E. McCreight, 1971) aunque hay una referencia precedente del mismo autor en 1970, en la cual establece: Sea $h >= 0$, un número entero, y k un número natural. Un árbol dirigido T pertenece a la clase $T(k, h)$, de los árboles B, si T está vacío ($h = 0$) o tiene las siguientes propiedades:

i) Cada camino desde la raíz hasta cualquier hoja tiene la misma longitud h, también llamada altura de T, es decir, h = número de nodos en el camino.

ii) Cada nodo, excepto la raíz y las hojas tiene por lo menos $k + 1$ hijos. La raíz es una hoja que tiene al menos dos hijos.

iii) Cada nodo tiene a lo más $2k$ hijos.

Cabe señalar que la condición ii) se agrega para le eficacia en cuanto al uso de memoria, y no es parte de la definición misma.

(Comer, 1979) no fue el primero en definir un árbol B+, pero lo incluye como una descripción de las variaciones de los árboles-B utilizada especialmente para almacenar índices. En un árbol B+, todas las claves residen en las hojas. Los niveles superiores, los cuales están organizados como un árbol B, sólo consisten en un conjunto de índices (nodos), que permiten localizar las claves de forma más rápida. La Figura 7 muestra la separación lógica de los índices y las claves en un árbol B

+. Naturalmente, los nodos y las hojas pueden tener diferentes formatos o incluso diferentes tamaños. Las hojas por lo general están vinculadas entre sí de izquierda a derecha, como se muestra, para permitir un proceso secuencial de las hojas. La lista enlazada de las hojas se conoce como sequence set.

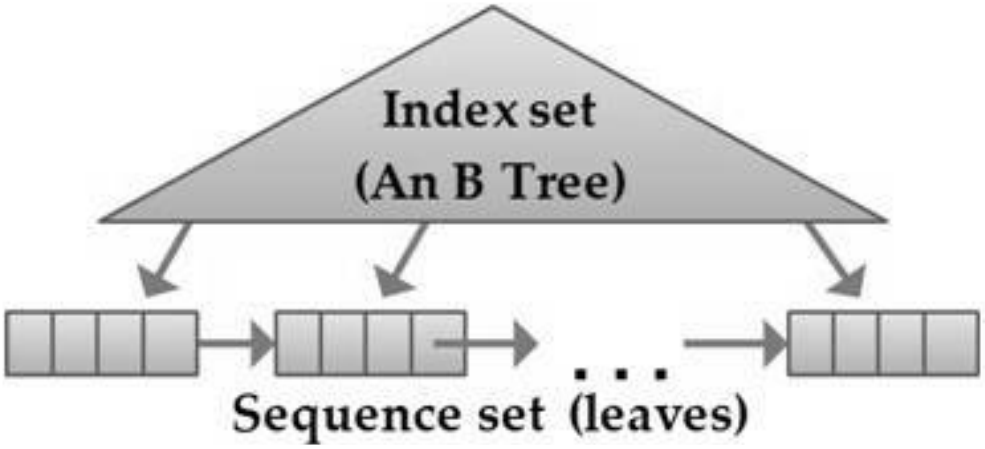


Figura 7 Un árbol B+, (Hernández Negrete, 2010).

En este trabajo, como en la mayoría de la literatura relevante sobre índices, incluyendo libros de texto, los árboles B+ se denominan árboles B. En particular, para el diseño que nos ocupa no se incluyó el uso de árboles B, en los cuales no todos las claves están en sus hojas, como es el caso en los B+.

Bitmaps

Para almacenar un conjunto de números sin repetición, se construye un intervalo de números (mediante un arreglo de bytes). Se numeran (lógica, no físicamente) los bits del arreglo.

Ejemplo: deseamos guardar espacios para acomodar números en un rango del 1 a 20000 (es decir, hasta 20000 números distintos). Se construye un arreglo de 20000 / 8 bytes (cada uno tiene 8 bits) Usando la notación de VB, A (2250) as byte. Consideramos el ultimo bit del primer número entero como la posición 0 (también

se puede hacer esto al revés, es decir comenzar por “la derecha”, pero involucra una operación adicional para “voltear” el índice (ind = 8 – ind).

Se presenta los métodos y sus sendas explicaciones mediante el código de Visual Basic (no importa la versión).

```
‘USAREMOS LA EXPRESIÓN DE BIT ENCENDIDO SI VALE 1
‘ES IMPORTANTE SEÑALAR QUE LOS BITS SE NUMERAN DE DERECHA A ‘IZQUIERDA ES DECIR
EL BIT 0 ES EL MENOS SIGNIFICATIVO
‘EN ESTE EJEMPLO GUARDAMOS NÚMEROS DEL 20001 AL 30000
‘V.GR. REGISTROS NUMERADOS DE UNA TABLA, POR EJEMPLO
‘PARA ALMACENAR ESTE RANGO, NECESITAMOS 10000 / 8 = 1250 BYTES
Dim a (1249) As Byte, VAL Orbit (7) As Byte, NN as Integer
‘en este ejemplo el índice de A varía de 0 a 1249
‘CUIDADO usamos el arreglo "al revés" es decir en forma ascendente: los números
del 1 al 8 estarán en A (0) etc.
'inicializa arreglo bits con las potencias de 2
‘1, 2, 4, 8, 16, 32, 64,128
NN = 1
For DD = 0 To 7; VAL Orbit (DD) = NN; NN = NN * 2; Next
Dim Num_a_usar as Integer, byte_numero As Integer
Dim bit_numero As Byte, num_reg As Integer
num_reg = 24001
'Supongamos que tenemos que hacer alguna operación con el registro No.24001
‘le restamos un número para que caiga dentro del rango (1, 10000)
Num_a_usar = 24001 - 20000
Num_a_usar = 5000
‘vale 4001 ' es decir usaremos este número en lugar del 24001
‘1º determinamos el byte del arreglo en el cual "figura" el 4001
byte_numero = Num_a_usar \ 8 ' división entera
‘2º determinamos el bit (dentro de A (byte_numero)
'byte numero en este caso vale 500
‘para determinar el bit dentro del byte
bit_numero = Num_a_usar Mod 8
‘ENCENDER EL BIT correspondiente al registro 24001
‘usamos la disyunción (unión) mediante la operación OR
A (byte_numero) = A (byte_numero) Ir VALORbit (bit_numero)
‘DETERMINAR SI está encendido el bit correspondiente al
‘registro 24001 Usamos la instrucción
‘AND (conjunción, intersección)
Dim bb as Byte
Bb = A (byte_numero) And VALORbit (bit_numero)
‘El bit está encendido si bb > 0
‘Quitar (APAGAR) el bit correspondiente al registro 24001
‘si no está encendido, lo deja en 0
‘usamos la operación AND NOT (diferencia)
A (byte_numero) = A (byte_numero) And Not VALORbit (bit_numero)
```

Memoria (o espacio en disco)

Supongamos que tenemos que guardar números entre 1 y 1,000,000. El arreglo correspondiente medirá 125,000 bytes. Usamos el arreglo para denotar presencia o ausencia de los números de registro en una tabla que tienen algún atributo. Supongamos que los campos incluyen:

- Sexo (M o F)
- Grupos de edades del 1 al 5
    - Por ejemplo 1= (21 a 30) 2= 1= (31 a 40) etc.
- Ingreso familiar: se agrupan en 15 categorías
- Colonia: hay registros de personas de 60 colonias

Supongamos también que hay 700.000 registros numerados del 1 a 800,000. El bitmap ocuparía 100,000 bytes. Usamos un bitmap para indicar si un registro es de "MASCULINO". Supongamos que la mitad de los registros caen en esta categoría. Si usáramos una lista de los números de registros, ésta tendría 350,000 elementos de 4 bytes cada uno, es decir, 1,400.000 bytes. En cambio si hiciéramos lo propio con la Colonia Moctezuma, quizá habría 20,000 habitantes de modo que la lista ocuparía 80.000 bytes.

Esto parece indicar una regla para determinar la conveniencia de usar Bitmaps: la "cardinalidad", es decir, el número de valores que puede tomar la variable. Se determina una proporción de los elementos totales a guardar en relación con la capacidad del bitmap. Si esta proporción es considerablemente inferior a 1/8, quizá se debería usar otro tipo de estructura.

En cambio si la mitad, o aún un cuarto, de los elementos estarían ocupados, definitivamente – en cuanto a espacio utilizado – se puede pensar en utilizar un bitmap.

Estos números pueden no resultar significativos, porque son “Kb” o aún “megas”. Pero ¿qué sucede cuando se trata de gigabytes de diferencia? Esto sucede con frecuencia en aplicaciones modernas, que tienen acervos muy numerosos de registros.

Los arreglos de una dimensión en Visual Basic casi no tienen límite de tamaño: se creó un arreglo de 1,000,000,000 de elementos y no hubo problemas, puesto que el requisito en memoria es que ésta tenga más de 4 gigas disponibles para el arreglo.

La otra manera de hacerlo es manejar varios arreglos de bytes (de las mismas dimensiones), en lugar de uno solo. En este caso se agrega una operación adicional para determinar en cuál de los arreglos está el bit deseado.

El uso de Bitmaps

Se reproduce aquí la descripción del uso de Bitmaps como índices de una base de datos (QaysAbdulhadi, Zuping, & Ibrahim Housien, 2013), con la aclaración de que se corrigieron los numerosos errores de sintaxis de la cita, de modo que ésta no es textual: “Bitmap indexes are efficient for ad hoc ranges of queries because they perform fast Boolean operations. Bitmap indexes are dynamic and effective tools to get optimal performance of large database This index data structure is mostly used for OnLine Analytical Processing (OLAP) and data warehouse applications”. La traducción libre de esta cita es: “Los índices de bitmap son eficientes par rangos específicos de consultas puesto que ejecutan operaciones booleanas con rapidez. Los índices de bitmap son herramientas dinámicas y efectivas para obtener desempeño óptimo en bases de datos grandes. Este estructura de datos de índices se usa especialmente en sistemas de Proceso Analítico On-line (OLAP) y en aplicaciones de bodegas de datos”. Es importante señalar que los bitmaps tienen muchos usos además de fungir como índices de una base de datos.

Como se indicó, los Bitmaps son cada vez comunes en aplicaciones relacionadas con el tema de índices. Diferentes productos pueden utilizar sus propias maneras de almacenar y utilizar un bitmap. Los Bitmaps están representados por arreglos de bits, o de enteros que agrupan al número indicado de bits. Como se muestra en el Cuadro 1, los bits se numeran en cada entero: el bit 0 es el bit menos significativo del entero, es decir, el que representa a 2^0, y el bit 31 es el más significativo.

Cuadro 1 Representación de un bitmap.

Bit	**31**	**30**	**.**	**.**	**.**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Valor del bit	1	0	.	.	.	0	0	1	1	1	0	1	0
+ significativo – significativo													

Esto indica que cada entero en realidad está representando 32 posibles números de instancias, en donde si la instancia está presente en el valor correspondiente a dicho bitmap, el bit está encendido (tiene valor 1), de lo contrario está apagado. Para un bitmap se especifica el número de instancia representado por el primer bit, ya que éste determina las instancias presentes en el índice como se ve en el Cuadro 2, en la que se muestran dos bitmaps, uno correspondiente a las instancias 33 a 64 y el otro a las instancias 97 a 128.

Cuadro 2 Ejemplo de un bitmap.

Bit	**31**	**30**	**.**	**.**	**.**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Valor del bit	1	0	.	.	.	0	0	1	1	1	0	1	0
Instancias presentes si el bitmap inicia en **33**	44							38	37	36		34	
Instancias presentes si el bitmap inicia en **97**	128							102	101	100		98	

Los algoritmos utilizados para encender o apagar los bits o para determinar si está encendido, usan un arreglo de 32 constantes. Sus valores son los valores de los números enteros en los que sólo el bit indicado por la posición del arreglo está

encendido. Llamamos al arreglo Only_bit_set (0-31) y se inicializa la siguiente manera:

Only_bit_set (k) = 2^{k} desde k = 0 hasta 30
Only_bit_set (31) = -2^{31}

Ejemplos del uso de este valor son:

- Si 2235 AND Only_bit_set (17) IS NOT = 0 determina si el valor del bit 17 (dentro del valor 2235) está encendido.
- 2235 OR Only_bit_set(17) encenderá el bit 17 (o permanecerá encendido)

Un arreglo similar de constantes con los complementos, es el que maneja los números en donde todos los bits están encendidos excepto el bit correspondiente al índice dentro del arreglo. Este arreglo se inicializa con las siguientes expresiones:

Only_bit_off (I) = - Only_bit_set (17) – 1 desde I = 0 hasta 30
Only_bit_off (31) = $2^{31} - 1$

De esta manera la operación 2235 AND Only_bit_off (17) apaga el bit 17 del número. Una operación usada frecuentemente para determinar si todos los bits de un numero están encendidos, es comparándolo NUM=-1**.** Del mismo modo si NUM=0 quiere decir que todos los bits están apagados. Se pueden usar bytes (es decir, números del 0 al 255) en lugar de enteros de 4 bytes.

2.1.3 Las estructuras especiales

Además de las estructuras más comunes descritas previamente, se contempló el uso de algunas estructuras no tan conocidas; típicamente no se las encuentra en textos sobe el tema. Sin embargo es importante señalar que no se pretende originalidad: muchos sistemas las han usado hace muchos años.

- Mapa de caracteres dígitos (los bitmaps ya se describieron anteriormente.

- Lista apuntada por otra lista.
- Lista de listas.
- Apuntadores a elementos de una lista a parir de un mapa de números.
- Archivo compuesto por varios segmentos.

MAPAS de caracteres dígitos

A semejanza de lo que sucede en BITMAPS (conjuntos contiguos de bits agrupados en bytes), un Mapa de caracteres es un conjunto de caracteres ASCII. Un mapa de caracteres es un conjunto de dígitos (almacenados como caracteres ASCII). Pueden ser números de 1 dígito o de varios, con la condición de que todos los elementos del mapa se representen con el mismo número de dígitos.

El mapa de caracteres

Consiste en una variable tipo STRING (cadena) de longitud fija o variable. Se interpreta de acuerdo al uso que se le fijó cuando se creó el mapa.

Ejemplo Para un conjunto numerado (del 1 al 40) de 40 alumnos de un curso, se almacenan como mapa de dígitos con valor 0/1 en una cadena de longitud 40 caracteres.

El maestro pasa lista, y el programa introduce un 1 en la posición correspondiente al número de alumno como se muestra en la Figura 8 (esto es similar a lo que sucede en un BITMAP). De ese modo almacena la asistencia de cada día como cadena de 40 bytes.

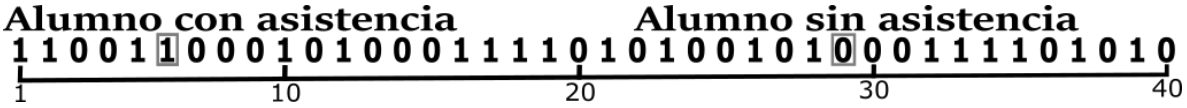


Figura 8 Ejemplo 1 Mapa de caracteres los 1 representan asistencia mientras que los 0 son ausencia, elaboración propia.

Otro uso frecuente de mapas de números es que los datos no corresponden a una numeración, sino que se agregan uno tras otro. Supongamos que el maestro

decidiera anotar los alumnos que faltaron. Construye su cadena agregando a una cadena vacía (nula) los números de esos alumnos, usando 2 dígitos para cada alumno. Resulta una cadena de longitud variable. Para recuperar los alumnos, lee consecutivamente (hasta que llegue al fin de la cadena) los números de alumnos de a 2 caracteres cada uno (naturalmente los debe grabar con esos 2 dígitos, aunque se trate del alumno 9) de esta forma se obtiene una cadena como se muestra en la Figura 9, cabe mencionar que el punto mostrado entre los números solo son de carácter ilustrativo ya que no son almacenados en la cadena.

Figura 9 Ejemplo 2 Mapa de caracteres cada elemento está representado por 2 caracteres continuos, elaboración propia.

Ejemplo de uso: en una aplicación, se especifican cuales elementos de un conjunto de éstos usa un determinado registro. Sean los colores en los cuales se vende un determinado artículo. Hay un catálogo de colores (supongamos que están numerados del 1 al 405). Un producto se puede vender en varios colores, ya sea que haya o no un máximo de colores por producto.

En lugar de crear un arreglo de los colores, se usa una cadena en la cual se concatenan los colores (con 3 dígitos cada uno) que usa el producto. Se recuperan los colores usando, unos tras otros, números de color mediante los 3 dígitos siguientes (Figura 10).

Figura 10 Ejemplo 3 Mapa de caracteres los elementos están representados por 3 caracteres cada uno, elaboración propia.

Arreglos o listas de listas

Esta estructura es conveniente cuando se tiene un conjunto de datos numerados por un número único y consecutivo. Supongamos que se trata de personas. Hay un número N relativamente grande de personas, pero no “demasiado grande”.

Para cada uno de los datos se almacena una estructura de datos por ejemplo:

```
Public Structure unaPersona
      <VBFixedString (10)> Public nombre As String
      <VBFixedString (10)> Public apellido As String
      <VBFixedString (10)> Public numeroDeTelefono As String
      <VBFixedString (10)> Public fechaNacimiento As String
End Structure
```

Se trata de armar una estructura que contenga algunas de estas personas.

Alternativa 1: se crea un arreglo de Número, y un arreglo de variables que usan la estructura contemplada.

```
Public losNumeros () As Integer
Public lasPersonas () As unaPersona
```

En el arreglo de números, de dimensión N; en cada posición se indica la posición que usa la persona de ese número en la lista de variables con sus datos.

Esto hace que si se tiene un conjunto de 40 personas, para el fin que sea, se ocupa un arreglo de 40 posiciones, cada una correspondiente a la variable con estructura. Para encontrar los datos de la persona guardada en la posición 6 del arreglo de números en un conjunto de 1000 personas totales, se usa la posición del arreglo de “números” con subíndice el valor que tiene el elemento 6 del arreglo de números, y se obtiene la posición de la persona en el arreglo de “personas” como se puede observar en la Figura 11.

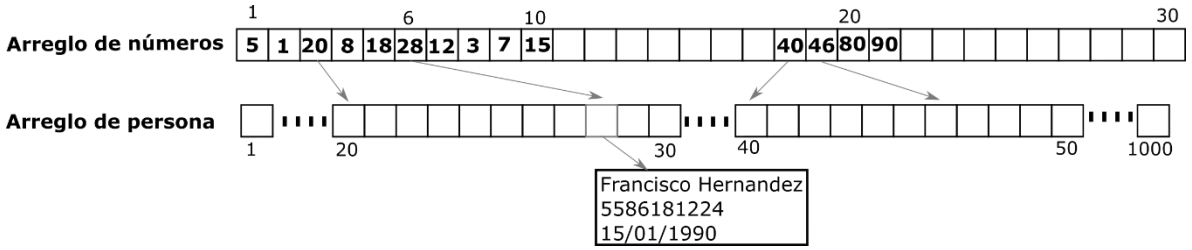


Figura 11 Ejemplo Lista de listas, cada elemento de una lista hace referencia a otra lista donde está contenida la información buscada, elaboración propia.

Esto tiene la ventaja de que el arreglo de datos no tiene que estar ordenado, es decir, cuando se agrega una persona se la agrega al final de la lista. El concepto es el de un índice: se ubica la posición en el índice y con dicho valor accedemos al dato buscado. Como se verá, en las estructuras seleccionadas para las distintas entidades del DBB, frecuentemente se usan estas estructuras especiales.

Archivo compuesto por varios segmentos

En ocasiones hay que crear varios archivos que almacenan datos totalmente distintos. Cada uno de éstos es a su vez un archivo de acceso aleatorio o binario, según convenga en cada caso.

Vale la pena decir que un archivo de acceso aleatorio es en realidad un archivo con acceso binario, excepto que se usa una rutina proporcionada (mediante la especificación de aleatorio) que convierte un número de registro en una posición dentro del archivo. Sea L la longitud de cada registro; para ubicar donde comienza el registro número N, se usa **Pos = (N-1) * L.**

De ese modo, abrir un archivo con acceso aleatorio es una conveniencia, pero puede resultar en una restricción: sólo se puede LEER (Get) o GRABAR (Put) una variable que tenga precisamente la longitud **L** bytes.

La *superestructura* que describimos aquí consta de M archivos random con distintas longitudes de registro, en un mismo archivo físico. Cada uno de estos archivos

puede tener estructuras diferentes; inclusive algunos de ellos pueden usar más de una estructura como se muestra en la Figura 12.

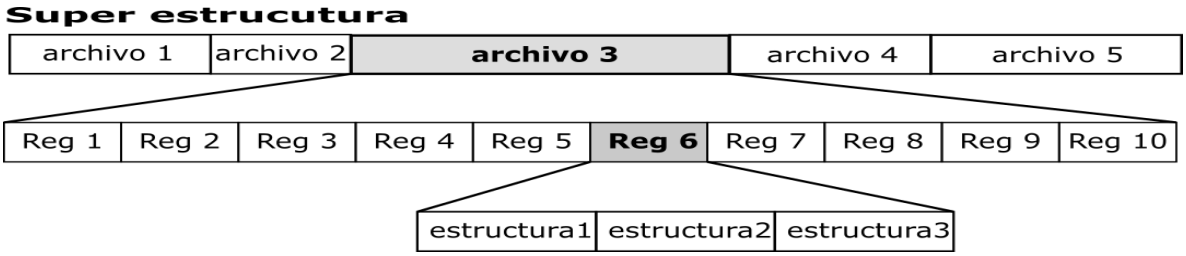


Figura 12 Super estructura cada segmento del archivo pertenece a una estrucutura diferente, elaboración propia.

Para ello se usa el archivo con acceso binario, y los parámetros para acceder a “cada archivo” se especifican ya sea en un programa que usa el archivo, o como un segmento inicial del mismo archivo. Como se verá, esta estructura es la que se utiliza en el DBB para varias entidades del modelo.

Supongamos que el archivo tiene S segmentos (archivos Random lógicos). Se anota, para cada uno de ellos, donde comienza, la longitud de cada uno de sus registros y su longitud total. Esto permite ampliar algunos de los segmentos a medida que se necesitan registros adicionales; para ello hay que hacerles lugar, recorriendo los segmentos siguientes en el número de bytes que se necesitan para acomodar los nuevos registros.

Observe que este diseño se usa en especial para satisfacer uno de los criterios de calidad de un producto de Software: el número de archivos que usa un sistema conspira contra el funcionamiento tipo SQF, que funcione siempre.

2.2 Métodos

Para establecer criterios diversos de comparación entre alternativas de estructuras y el uso de las mismas, se programaron y ejecutaron una serie de simulaciones que se describen en el Capítulo siguiente.

CAPÍTULO 3. SIMULACIONES EFECTUADAS PARA DETERMINAR CRITERIOS DE COMPARACION

3.1 Introducción

Como preparación para la toma de decisiones en cuanto a las estructuras más convenientes para almacenar y usar los datos del acervo, se ejecutaron procesos que pudieran proporcionar algunos criterios de comparación.

Esta etapa de la investigación se diseñó tomando en cuenta las diversas estructuras contempladas, especialmente en cuanto a la duración de los diversos procesos involucrados. En el caso del almacenamiento de las marcas - tanto las de cada UBI como las de un par (contexto-Valor)- también se compararon el espacio en disco utilizado por cada una de las estructuras contempladas.

De ese modo, se diseñaron programas y rutinas para generar datos y comparar las alternativas de almacenamiento, así como los algoritmos usados para efectuar operaciones entre listas de resultados. En algunos casos, se incluyó la posibilidad de ejecutar el mismo proceso varias veces para determinar su duración cuando ésta es muy breve o para obtener una mejor estimación.

3.2 Las simulaciones efectuadas

Se describen en sendas secciones los procesos y se muestran los resultados y criterios obtenidos a partir de las mismas. Se programaron y ejecutaron procesos para determinar:

- Genera simulación para comparar cómo se almacenan las UBIs, sus marcas y los triplos (contexto, valor, UBI). Aquí se crearon listas de resultados que se usaron en algunos de los otros procesos.
- Comparación de algoritmos iterativos y recursivos para las operaciones de intersección y unión

- El uso de bitmaps y arreglos para las operaciones booleanas.
- La interpretación de una lista de resultados (proporcionar datos de cada UBI de la lista)
- Lectura de arreglos de estructuras y de números de disco

3.2.1 Genera simulación para comparar cómo se almacenan las UBIs, sus marcas y los triplos (contexto, valor, UBI). Aquí se crearon listas de resultados que se usaron en algunos de los otros procesos.

Objetivo

Debido a que el objetivo principal del DBB es almacenar UBIs y agregarles marcas es importante definir correctamente la manera en que serán almacenados cada uno de estos componentes. Para realizar la siguiente simulación se definen los siguientes casos:

Alt 1. Las UBIs se almacenan en una tabla en base de datos y las marcas de cada UBI son almacenadas en un campo memo dentro de la tabla UBIs.

Alt 2. Las UBIs se almacenan igual en una tabla pero sin sus marcas como parte de la tabla; las marcas se almacenan en una tabla con los campos contexto, valor, UBI.

Aspectos a considerar:

- Espacio en disco
- Velocidad de operaciones

Realizar una simulación

Para la simulación de utiliza la interfaz mostrada en la Figura 13 que se separa en segmentos como se puede notar, cada sección se explicara por separado.

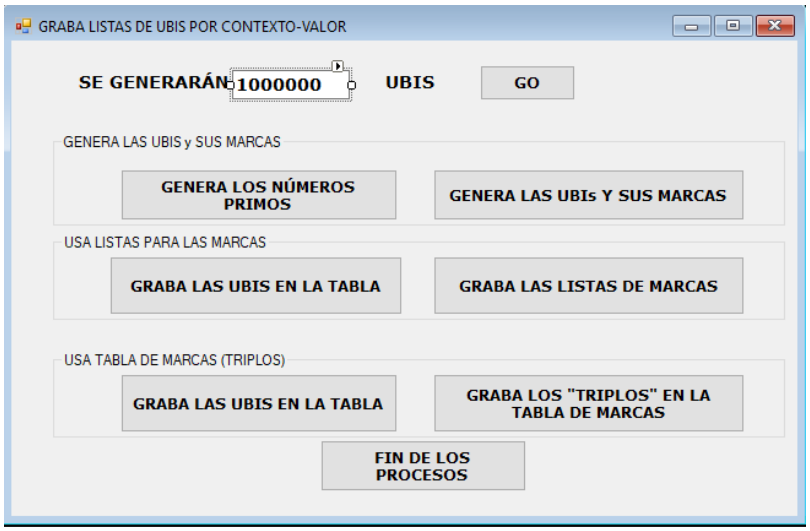


Figura 13 Interfaz simulación para comparar cómo se almacenan las UBIs, sus marcas y los triplos, captura de ejecución.

Generar las UBIs y sus marcas

Para realizar las simulaciones de generaron 1,000,000 de UBIs, y las marcas que se le asignan dependen de su número según los siguientes casos:

Contexto 1 UBI Mod 5

- Valor 0 (5, 10,15, 20…..)
- Valor 1 (1, 6, 11, 16…..)
- Valor 2 (2, 7, 12,17…..)
- Valor 3 (3, 8, 13,18…..)
- Valor 4 (4, 9, 14,19…..)

Contexto 2 Es primo Valor es único: Sí o NO

De ese modo sólo se generan las marcas SI

Contexto 3 Producto de 2 primos Valor es único: Sí o NO

De ese modo sólo se generan las marcas SI

Contexto 4 Cuadrado perfecto Valor es único: Sí o NO

De ese modo sólo se generan las marcas SI

Contexto 5 Cubo perfecto Valor es único: Sí o NO

De ese modo sólo se generan las marcas SI

Contexto 6 Múltiplo de

Valor 4 se marcan los múltiplos de 4 con el contexto (5,”4”)

Valor 11 se marcan los múltiplos de 11 con el contexto (5,”11”)

Valor 14 se marcan los múltiplos de 14 con el contexto (5,”14”)

Valor 27 se marcan los múltiplos de 27 con el contexto (5,”27”)

Valor 101 se marcan los múltiplos de 101 con el contexto (5,”101”)

Valor 803 se marcan los múltiplos de 803 con el contexto (5,”803”)

Valor 1127 se marcan los múltiplos de 1127 con el contexto (5,”1127”)

Por ejemplo la UBI **64** tendría las siguientes marcas: (1, “4”), (4, “SI”), (5, “SI”), (6,”4”).

Las marcas se generan en memoria antes de ser guardadas, ya sea en base de datos o en archivos planos.

Se agregó una funcionalidad para permitir ejecutar el proceso de guardar los tripos en varias etapas. Se crea un archivo que contendrá los triplos generados; este archivo será utilizado posteriormente para almacenar las marcas en la base de datos considerada en la Alt 2.

Grabar las UBIs en la tabla

Al igual que la alternativa anterior se guardaran las UBIs en una tabla de bases de datos pero en este caso no contendrá el campo **“sus_marcas”;** el procedimiento es exactamente el mismo, se genera un DataTable vacío con los campos correspondientes a la UBI y se crean y agregan rows uno a uno con sus campos correspondientes.

Grabar los triplos en la tabla de marcas

Como ya fue mencionado anteriormente en la generación de las UBIS y sus marcas del primer paso, se generó un archivo que contiene los triplos (UBI, contexto, valor).

En esta sección se hace uso del contenido de ese archivo para generar los registros que contendrá la tabla **“marcas”.**

De igual forma que en los procedimientos anteriores se genera un DataTable vacío y se agregan una a una las filas correspondientes. Los campos de la tabla con: contexto, valor, UBI.

Resultados

Se repiten las alternativas para facilitar la comprensión y lectura de los resultados.

Alt 1. Las UBIs se almacenan en una tabla en base de datos y las marcas de cada UBI son almacenadas en un campo memo dentro de la tabla UBIs al igual que en archivos planos (un archivo por clase).

Alt 2. Las UBIs se almacenan igual en una tabla pero sin sus marcas como parte de la tabla; las marcas se almacenan en una tabla con los campos contexto, valor, UBI.

Los resultados obtenidos fueron generando 1,000,000 UBIs y 1,751,045 marcas.En general una UBI tendrá varias marcas, pero en la simulación se decidió que un par de millones permitiría hacer las comparaciones de espacio en disco.

Comparación de espacios

Como se puede observar en el Cuadro 3 usando una tabla de marcas el espacio en disco utilizado es mayor que almacenando las marcas en un archivo plano.

Cuadro 3 Espacio en disco almacenamiento de UBIs y sus marcas.

	sin usar la tabla de marcas	Usando la tabla de marcas
base de datos	864 Kb	94.4 Mb
archivos	6.71 Mb	0 Mb
total	7.55 Mb	94.4 Mb

Esto indica que el uso de una tabla con los triplos resulta en un uso mucho mayor de disco. Cuando se trata de acervos de datos muy grandes, esta diferencia puede ser muy significativa, a pesar del tamaño casi ilimitado de espacio en disco del que se puede disponer en una computadora.

Velocidad de grabación

Los tiempos de proceso se muestran (en segundos) en el Es importante señalar que en general no se graba un millón de marcas en un solo proceso.

Cuadro 4 y se puede observar que igual en este caso la utilizar una tabla de marcas es considerablemente más lenta que utilizando archivos para almacenarlas. Es importante señalar que en general no se graba un millón de marcas en un solo proceso.

Cuadro 4 Tiempos de ejecución (en segundos) para almacenamiento de UBIs y sus marcas.

	En archivos de listas de UBIs	En la tabla de “MARCAS”
grabar las UBIs	1.4664025	1.3900795
grabar las marcas	0.4680008	1252.207622
total	1.9344033	1253.597702

3.2.2 Comparación de algoritmos iterativos y recursivos para las operaciones de intersección y unión

Objetivo

Uno de los objetivos del DBB es poder realizar operaciones entre listas de resultados. El objetivo de esta sección es mostrar las maneras contempladas para realizar las operaciones entre 2 o más listas, y así decidir cuál método se utilizara para realizar las operaciones.

Las dos maneras contempladas son realizar las operaciones de forma iterativa o de forma recursiva; las operaciones contempladas son la intersección y la unión. El requisito para hacer operaciones entre listas es que cada lista debe estar ordenada de menor a mayor.

Para preparar la simulación se utilizó la interfaz mostrada en la Figura 14 donde se indica cuantos elementos se generan para cada lista y cuantas veces se realiza la operación (es necesario realizar más de una vez las operaciones ya que los tiempos son muy cortos y no aparecerán puesto que se registran en el sistema en milisegundos). Para esta simulación se manejaron únicamente 6 listas, pero las rutinas generadas son capaces de trabajar con cualquier cantidad de número de listas.

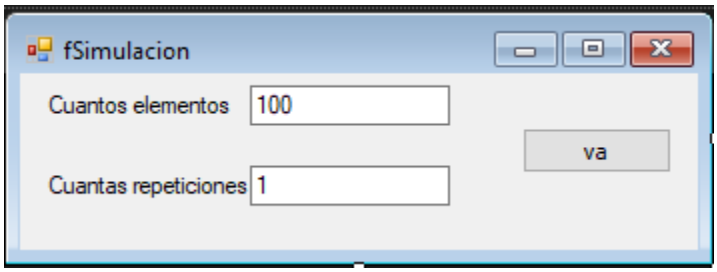


Figura 14 Preparar simulación operación entre listas se definen el número de elementos con el que se trabajaran y cuantas veces se ejecutaran los procesos, captura de ejecución.

Una vez que se han definido los parámetros anteriores se muestra la interfaz representada por la Figura 15.

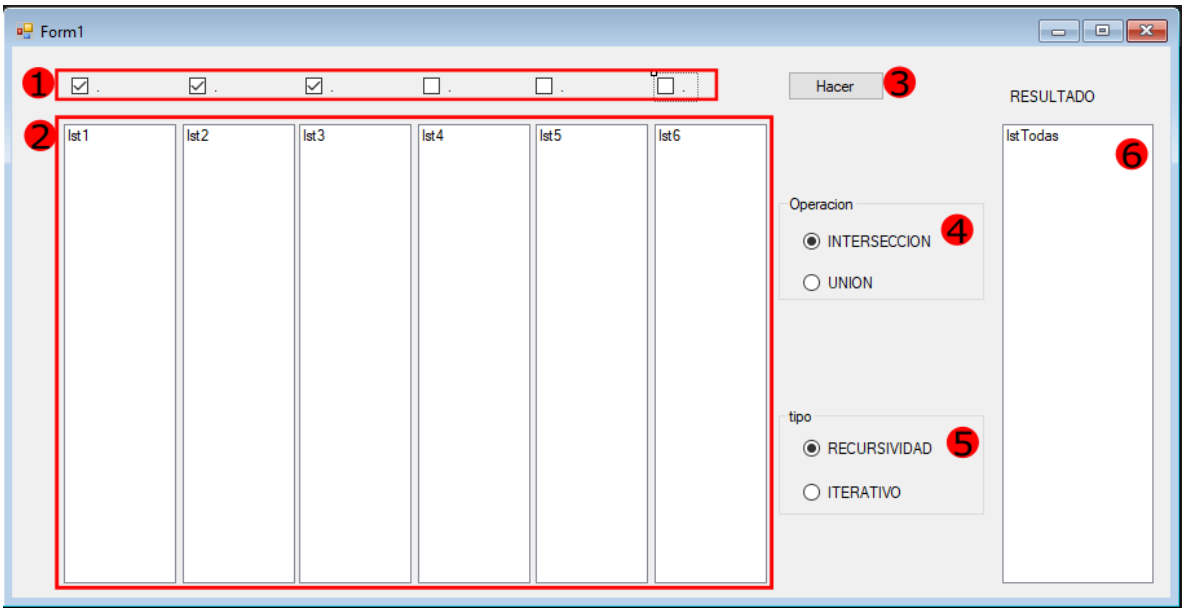


Figura 15 Interfaz operaciones entre listas permite seleccionar con que listas, que operación y como se realizara, captura de ejecución.

Donde se muestran los siguientes elementos:

1. Selectores de listas
2. Listas generadas
3. Botón “hacer” manda a llamar las rutinas
4. Selector de operación
5. Selector de tipo de operación
6. Lista de resultados

Para la simulación se generó una clase llamada “clLista” que contiene los atributos y rutinas necesarios para realizar las operaciones, a continuación se muestra la clase únicamente con sus atributos:

```
Public Class clLista
      Public laLista () As Integer, tamanioLista As Integer, queLista As
      Integer
      Public este As Integer, valor As Integer
      Private queFue As String
      Public operandos () As clLista, cuantosOperandos As Integer
      Public cualOperacion As String, elementosOcupados As Integer
      Public POS_LR As Integer
      Private entraEsteValor As Boolean, esteValor As Integer
End Class
```

Realizar una simulación

El procedimiento para realizar las operaciones consta de los siguientes pasos:

1. Preparar operandos
    a. Calcular tamaño de la lista de resultados.
2. Recorrer listas para encontrar el menor de todas.
3. Agregar a lista de resultados.

Preparar operandos

A continuación se explica la manera en que se preparan las listas de operandos: Para realizar una operación se genera una instancia de la clase clLista llama listaDeResultados; esta clase contiene un arreglo llamado **operandos**, que contendrá las listas con las que se realizan las operaciones. En el ejemplo que ilustra en Figura 16 este ejemplo se hacen operaciones con las listas **"1, 4,5"**.

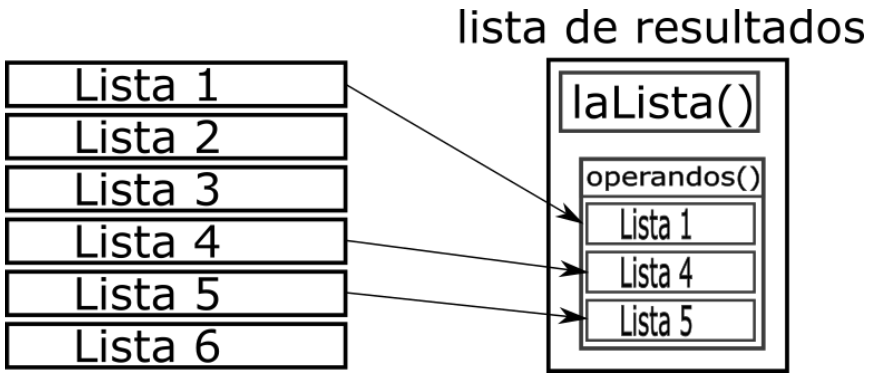


Figura 16 Cada lista de resultados tendrá como elementos las listas con las que se realizaran las operaciones, elaboración propia.

El tamaño máximo de "laLista ()" resultante se calcula sumando el tamaño de cada uno de los operandos**.**

Hacer la operación

Hacer una operación consta de dos procedimientos: **recorrer las listas y agregar a la lista de resultados;** ambos procedimientos serán descritos a continuación de forma separada:

Recorrer una lista

El funcionamiento básico es recorrer las listas hasta encontrar el menor de todos y agregarlo a una lista de resultados, se definieron 2 alternativas para hacer este recorrido: **iterativo y recursivo**.

Rutina recursiva para la UNIÓN de las listas

El procedimiento para recorrer una lista recursivamente, consta básicamente en agregar uno a uno los elementos de la lista, pero antes se deben agregar todo los números que sean menores o igual de las demás listas.

Los pasos para recorrer las listas de operandos de forma recursiva son:

1. Mientras la primera lista tenga elementos.
    a. Para cada elemento llamar la rutina recursiva con la lista siguiente y el valor de la posición actual:

        COMENTARIO: **valor** es el número que se encentra en la posición que se está comparando, **menor** es el número con el que fue llamada la función.

        i. Cuando se llega a la última lista y se intenta llamar una lista más se termina la rutina.
        ii. Mientras el valor de la lista sea menor al valor con el que fue llamada la rutina.
            1. Llamar rutina recursiva con la siguiente lista y el valor actual de la lista.
            2. Si es el final de la lita: termina la función.
            3. Avanzar a la siguiente posición de la lista (aquí se intenta agregar el valor según sea la operación que se realiza).
    b. Avanzar a la siguiente posición de la lista (aquí se intenta agregar el valor según sea la operación que se realiza).

El procedimiento descrito se muestra en la Figura 17, para este ejemplo se utilizan 3 listas.

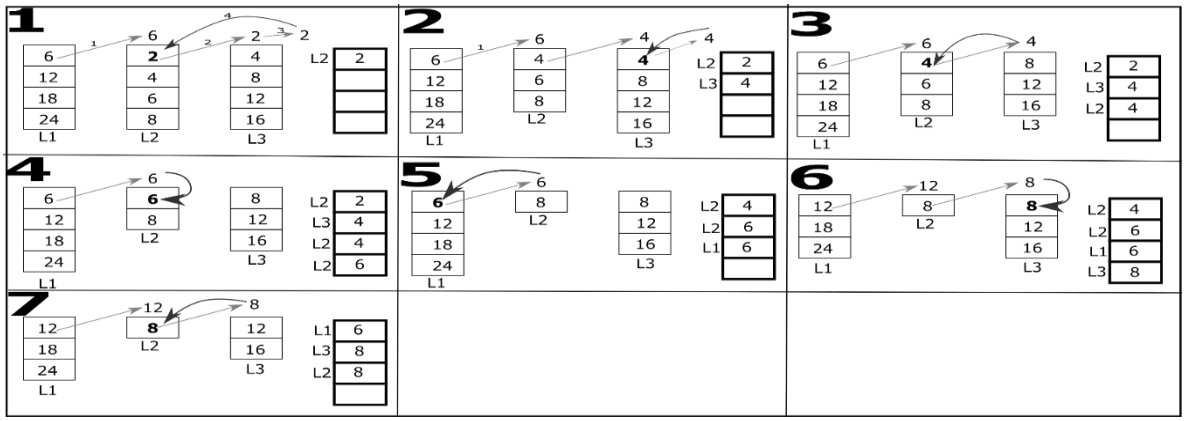


Figura 17 El recorrido consta básicamente agregar todos los elementos menores de las demás listas antes de agregar el actual, elaboración propia.

Rutina iterativa

En esta rutina el procedimiento consta de recorrer cada una de las listas, encontrar el menor número, registrar en qué lista se encuentra y agregar este número y pasar al siguiente de esta lista de la cual se obtuvo el valor.

Los pasos para recorrer las listas de operandos de forma iterativa son:

1. Mientras no se han recorrido todos los elementos de todas las listas.
    a. El menor es igual al valor de la primera lista y la posición del menor está en la primera lista.
    b. Recorrer las listas una a una.
        i. Si el valor de la lista es menor al menor actual: actualizar el menor y la lista en que esta.
    c. Agregar el número menor (se tiene guardado en que lista está el menor).

En la Figura 18 se ejemplifica el funcionamiento de esta rutina, cabe mencionar que este método hace un mayor número de comparaciones que la recursiva.

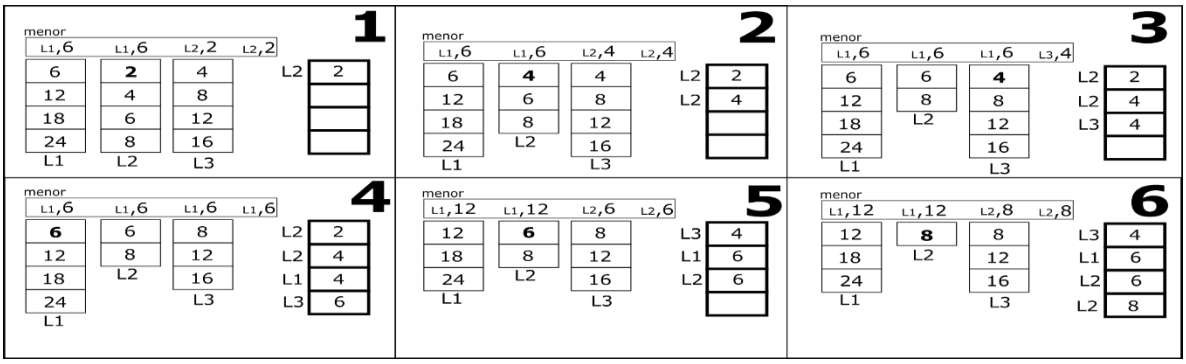


Figura 18 En el método iterativo se recorre todas las listas cada vez y se guarda el menor elemento de todas, elaboración propia.

Agregar a lista de resultados

La forma en que se agregan los valores depende de que operación se esté ejecutando (unión o intersección), de igual manera que las condiciones de paro o trucos para optimizar el recorrido.

Unión

En la unión ya sea con la rutina recursiva o iterativa, el procedimiento para agregar es el mismo, se agregan todos los valores y no se permiten números repetidos.

Las condiciones que fueron implementadas son:

1. Si el nuevo valor es igual al último agregado: no se agrega el nuevo valor.
2. Si el nuevo valor es diferente al último agregado: se agrega el nuevo valor y se actualiza el último agregado.

Artificio para optimizar la operación

En el caso que se termine una lista, se hace un cambio de posiciones de listas (los operandos): la última lista pasa a ocupar el lugar le la que se ha terminado y se actualiza el número de listas que se recorren, de esta manera el número de comparaciones se reduce en uno.

A continuación se muestra un ejemplo de cómo se implementa esta situación, considerando las listas que se muestran en la Figura 19, se puede ver que la lista número 2 se terminara primero.

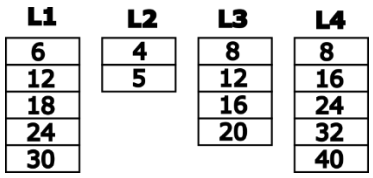


Figura 19 se muestra un conjunto de listas en el que claramente se ve que la L2 se terminara primero, elaboración propia.

Esto lleva a hacer una comparación aun sabiendo que la lista se encuentra vacía, esto se corrige haciendo un cambio de lugares, se mueve la última lista al lugar de la lista bacía.

En el ejemplo se termina primero la lista número 2, como se menciona la última lista pasa a tomar el lugar de la que quedó vacía como se puede observar en la Figura 20 y se debe actualizar el número de listas que se compararan en adelante.

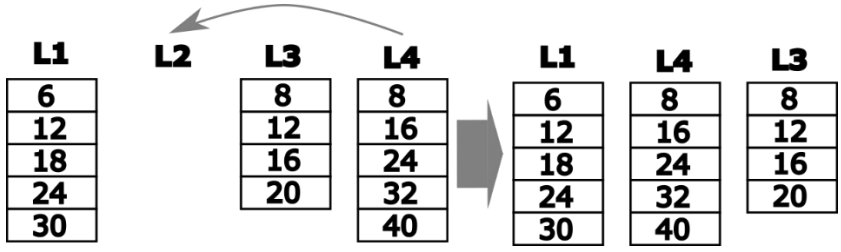


Figura 20 Cuando se termina una lista es remplazada por la ultima y de esta forma se reduce el número de veces que se llama la función, elaboración propia.

Intersección

De igual forma que la unión, el método para agregar en la intersección es el mismo tanto para la rutina recursiva como para la iterativa, el valor tiene que estar en todas las listas para poder ser agregado.

Las condiciones para agregar son:

1. El nuevo valor es diferente al último valor agregado: ultimo valor agregado es igual al nuevo valor, y se reinicia el contador en 1.
2. Si el nuevo valor es igual al último agregado: se aumenta 1 al contador.
3. Si el contador es igual al número de listas se agrega el valor a la lista de resultados.

En otras palabras, se “cuentan” las listas en las que se encontró el valor indicado de la primera lista. Si este número es igual al número de operandos, se agrega el valor a la lista de resultados.

Resultados

En este caso la simulación solo fue realizada en una computadora con las características ilustradas en el Cuadro 5.

Cuadro 5 Especificaciones de la computadora utilizada.

Marca	sistema operativo	memoria RAM Gb	procesador	velocidad procesador GHz	disco duro
Asus	Windows 10	12	AMD	2.4	SSD

Cada lista contiene 100,000 elementos y los procesos se corrieron 10 veces para ya que los tiempos de ejecución son mínimos. Los resultados obtenidos se muestran en el Cuadro 6.

Cuadro 6 Tiempos de ejecución de la simulación.

Unión		Intersección	
Recursivo	Iterativo	Recursivo	Iterativo
0.4361095	0.6550876	0.1270347	0.3321637

Estos datos llevan a la siguiente conclusión: En ambas operaciones, las rutinas recursivas son más eficientes. Sin embargo, en intersecciones la diferencia es suficiente para adoptar el proceso recursivo. Como no se obtuvieron resultados para uniones (en numerosos ejemplos) en los cuales el proceso iterativo era más rápido, también se adoptó el proceso recursivo para las uniones.

3.2.3 El uso de bitmaps y arreglos para las operaciones booleanas.

Puesto que los procesos más frecuentes en el DBB son las búsquedas (lo que el creador del DBB denomina el subconjunteo), se obtuvieron datos que permitieran comparar diversas alternativas para implementar estos procesos.

Las búsquedas serán del tipo: Todas las UBIs marcadas con un par (contexto, valor). Y la combinación de los subconjuntos obtenidos de este modo con otros similares. Por ejemplo, alguien desea las UBIs que satisfacen:

Están marcadas con (contexto 4, valor “RAUL”) Y (contexto 17, “ROJO”) donde se pueden indicar tantas operaciones como sean necesarias. Estas operaciones en general serán AND, OR o NOT. La operación XOR es mucho menor frecuente en este tipo de uso, pero también se implementa como algoritmo recursivo. De ese modo, se analizaron dos aspectos.

Cómo están almacenadas las marcas:

- En una tabla de una base de datos. En ese caso, para cada par (contexto, valor) se crea un “DataTable” con una consulta SQL que limita los resultados por la cláusula WHERE correspondiente.
- En listas de UBIS almacenadas en archivos planos. En este caso, el DBB contendrá un apuntador a la lista en cuestión.
- En “bitmaps” formados por las UBIs que están marcadas con un par (Contexto, valor).

Cómo se realizan las operaciones:

- Usando las listas (o los DataTable).
- Usando solo bitmaps.
- Usando combinaciones de listas y bitmaps cuando se trata de varios operandos.

Se elaboró un programa que usa las listas de resultados elaboradas en la simulación de comparación de espacio y proceso de las UBIs. Tiene las rutinas para crear un bitmap a partir de una lista de UBIs. Se analizan operaciones (cuando es necesario

se crea un bitmap a partir de una lista de UBIs). En la Figura 21 y Figura 22 se muestra las interfaces utilizadas para realizar la simulación, la primera corresponde a la selección de la operación y con que se realizaran, mientras que la segunda corresponde a la selección de las listas que serán utilizadas.

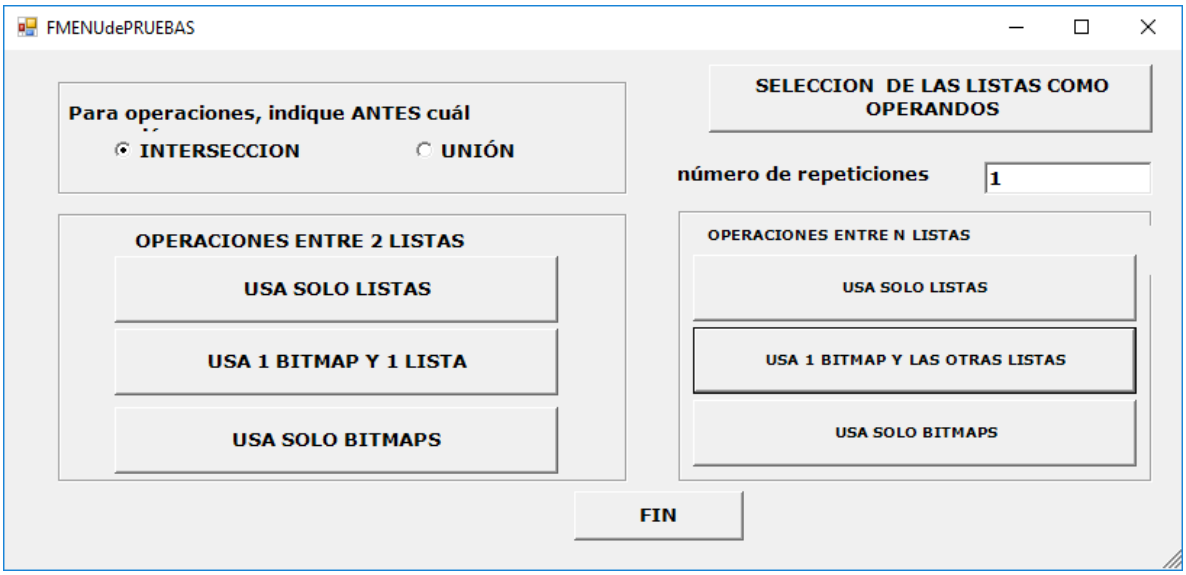
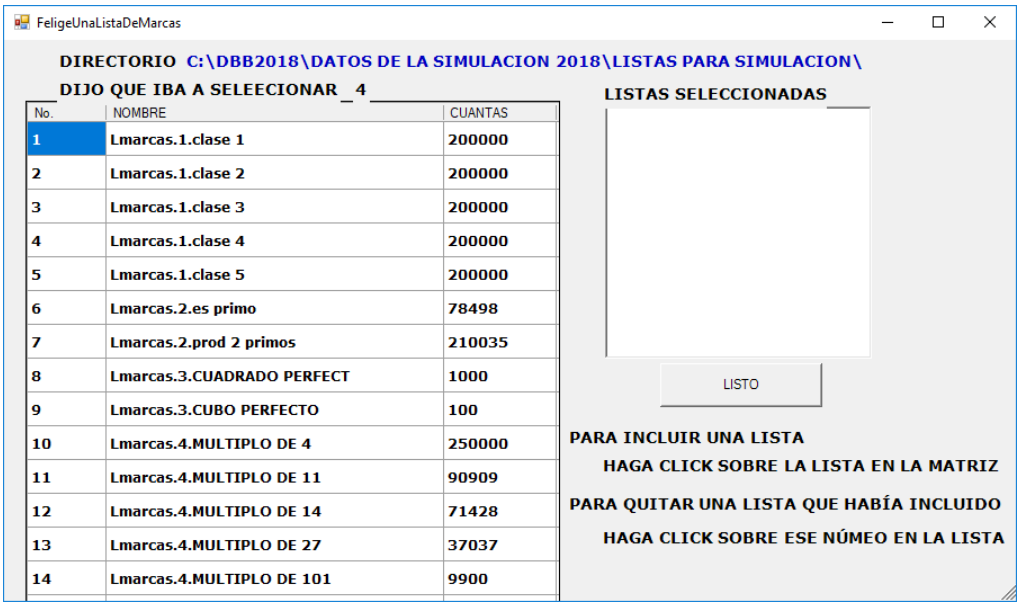


Figura 21 Interfaz utilizada para realizar las simulaciones con operaciones entre bitmaps y listas presenta las opciones para elegir listas, que operación se realizara y como.



No.	NOMBRE	CUANTAS
1	Lmarcas.1.clase 1	200000
2	Lmarcas.1.clase 2	200000
3	Lmarcas.1.clase 3	200000
4	Lmarcas.1.clase 4	200000
5	Lmarcas.1.clase 5	200000
6	Lmarcas.2.es primo	78498
7	Lmarcas.2.prod 2 primos	210035
8	Lmarcas.3.CUADRADO PERFECT	1000
9	Lmarcas.3.CUBO PERFECTO	100
10	Lmarcas.4.MULTIPLO DE 4	250000
11	Lmarcas.4.MULTIPLO DE 11	90909
12	Lmarcas.4.MULTIPLO DE 14	71428
13	Lmarcas.4.MULTIPLO DE 27	37037
14	Lmarcas.4.MULTIPLO DE 101	9900

Figura 22 Interfaz utilizada para la selección de las listas que se utilizan en las operaciones, muestra las listas que se encuentran cargadas en memoria, el nombre que las identifica y el número de elementos que contiene.

Los procesos que fueron definidos son los siguientes:

Entre 2 listas

Usando 2 listas
Usando una lista y un bitmap
Usando 2 bitmaps.

Entre N - varias – listas

Usando N listas
Usando un bitmap y los demás operandos como lista
Usando sólo bitmaps.

El Cuadro 7 presenta las duraciones de cada uno de los procesos efectuados, que se seleccionaron para contemplar todos los casos de interés. Las duraciones se redondearon (para arriba) a milisegundos. Las simulaciones que se realizaron con las siguientes características:

- 2 listas (unión e intersección):L1 múltiplos de 5 y L2 producto de dos primos.

- 4 listas (unión): L1 múltiplos de 5, L2 (1, 6, 11, 16,…), L3 primos, L4 múltiplos de 11.
- 3 listas (intersección): L1 múltiplos de 5, L2 cuadrado perfecto, L12 múltiplos de 14.

Cuadro 7 Tiempos de proceso para diversas operaciones usando bitmaps o no.

Operación	Cuantas listas	Como	Duración	Tamaños de las listas
UNION	2	2 listas	0.048	200000, 210035
UNION	2	1 lista y 1 bitmap	0.021	200000, 210035
UNION	2	2 Bitmaps	0.007	200000, 210035
INTERSECCION	2	2 listas	0.024	200000, 210035
INTERSECCION	2	1 lista y 1 bitmap	0.017	200000, 210035
INTERSECCION	2	2 Bitmaps	0.007	200000, 210035
UNION	2	1 lista y 1 bitmap	0.010	210035, 200000
UNION	2	2 Bitmaps	0.002	210035, 200000
INTERSECCION	2	1 lista y 1 Bitmap	0.01	210035, 200000
INTERSECCION	2	2 Bitmaps	0.001	210035, 200000
UNION	2	1 lista y 1 bitmap	0.005	200000, 78498
UNION	2	2 Bitmaps	0.002	200000, 78498
INTERSECCION	2	1 lista y 1 Bitmap	0.005	200000, 78498
INTERSECCION	2	2 Bitmaps	0.001	200000, 78498
UNION	4	Todos listas	0.051	200000, 200000, 78498, 90909
UNION	4	1 Bitmap y 3 listas	0.020	200000, 200000, 78498, 90909
UNION	4	todos Bitmaps	0.004	200000, 200000, 78498, 90909
INTERSECCION	3	Todos listas	0.028	200000, 1000, 71428
INTERSECCION	3	1 Bitmap y 2 listas	0.005	200000, 1000, 71428
INTERSECCION	3	todos Bitmaps	0.002	200000, 1000, 71428

Es necesario señalar que estos tiempos de ejecución no incluyen el proceso de armado de un bitmap a partir de una lista de UBIs (o de resultados previos). En el Cuadro 8 se muestran duraciones en segundos redondeadas a milisegundos del armado de bitmaps para listas con diversos números de elementos.

Cuadro 8 Duraciones del armado de bitmaps a partir de arreglos de números.

Cuantos elementos	Tiempo para crear el bitmap
250000	0.017
210035	0.015
200000	0.011
90909	0.007
78498	0.004
71428	0.004
37037	0.002
9900	0.002
1248	0.001
1000	0.001
890	0.001
100	0.000*

Algunos detalles sobre la lógica utilizada para los procesos.

Las operaciones entre bitmaps se efectúan del siguiente modo:

Para la UNIÓN:

- Se determina el tamaño del bitmap resultado. El intervalo comienza con el número “mínimo de mínimos” de los operandos, y termina el máximo de máximos. El mínimo es el menor número representado en una lista o en un bitmap.
- Se crea un nuevo bitmap resultado.
- Se copia uno de los operandos al nuevo bitmap (con un copiado de memoria).
- Se hace la operación “OR” para cada byte del otro operando. El detalle estriba en usar los bytes correctos de cada uno de los bitmaps, puesto que el segundo operando (el segundo bitmap) puede no comenzar al principio ni terminar al final del bitmap resultado que se construyó.

Para la INTERSECCIÓN

- Se determina el tamaño del bitmap resultado. El intervalo comienza con el número “máximo de los mínimos” de los operandos, y termina el “mínimo de los máximos”. El mínimo es el menor número representado en una lista o en un bitmap.
- Se crea al bitmap resultado con “ceros”.
- Se recorre la parte “común” de los operandos, byte por byte, y se usa la operación AND entre los bytes correspondientes, y el resultado se graba en el byte del bitmap resultado.

Para operaciones entre varias listas, éstas se hacen en forma consecutiva: el resultado de cada paso es un operando el siguiente paso. Se señala que esto no es óptimo: conviene hacer todas las operaciones en un mismo paso. Esto disminuirá los tiempos de ejecución. Como se trata de comparar algoritmos, esta circunstancia sólo aumenta las diferencias en tiempos de proceso que compondrán el criterio de comparación.

Estos resultados arrojaron las siguientes conclusiones:

- El uso de bitmaps para las operaciones siempre será más eficiente que usando las listas (originales).
- Cuando se trata de crear un solo bitmap, conviene usar la lista que tiene el mayor número de elementos.
- Cuando las listas son muy breves, el impacto del uso de bitmaps puede ser insignificante.

Estas conclusiones serán incorporadas a los programas del DBB, además de haber afectado la selección de estructuras a utilizar para almacenar las marcas.

3.2.4 La interpretación de una lista de resultados (proporcionar datos de cada UBI de la lista)

Objetivo

Como se vio, uno de las funciones principales del DBB consiste en poder recuperar las UBIs que cumplen con ciertos criterios de búsqueda. Naturalmente hay que interpretar los números de UBI que resultan de estas consultas; es decir, el usuario no aprovecha (en general) que se le proporcione una lista de números de UBI, sino que hay que describir cada una de ellas.

El objetivo de esta sección es mostrar los modos contemplados para interpretar y mostrar las listas de resultados, de forma que este proceso se separara en 2 secciones: **obtener una lista de resultados** y **mostrar la lista de resultados**.

Por lo tanto, se trató de este tema: dada una lista de resultados (arreglo ordenado de números de UBI resultante de una consulta) desplegar los datos de estas UBIs necesarios para que el interesado comprenda los resultados. Se trata de usar los métodos y objetos que disminuyan los tiempos de respuesta de las consultas.

Paso 1. Obtener el subconjunto de UBIs cuyos números están en la lista de resultados.
Paso 2. Mostrar estas UBIs en uno varios objetos gráficos de una forma.

Se formularon dos necesidades:

- Determinar cómo se implementaría el transformar una lista de resultados (números de UBI) en un conjunto de las UBIs correspondientes.
- Comparar el uso de listas (ListBox, ListView) con el de matrices (DataGridView) para mostrar los datos. Se agregó la comparación de que el sistema complete estas estructuras por la asignación de una fuente de datos con el proceso hecho en forma individual para cada registro (UBI).

Métodos contemplados para obtener el conjunto de UBIs (con los datos necesarios) correspondientes a una lista de resultados

Base de datos

Los siguientes métodos suponen que las UBIs se almacenaron en una tabla (llamada UBIS) de una base de datos.

Método 1. Se formula una consulta SQL de este tipo:

Select * from UBIS where UBI in (a1, a2, a3….) donde a1, etc. son los números de UBI de la lista de resultados.

Método 2. Crear una tabla temporal en la base de datos: el campo único es el número de UBI; tendrá un registro para cada elemento de la lista de resultados. Sea TTEMP esta tabla.

- Create table TTEMP (UBI integer) primary key UBI
- Se crea un DataTable (vacío) en memoria para esta tabla
- Se agregan uno tras otro los números de la lista de resultados.

Se formula una consulta SQL como:

Select * from UBI, ttemp where UBI.ubi = ttemp.UBI

Método 3. En lugar de una cláusula “in”, se determina el menor y el mayor elemento de la lista. Sean MINU y MAXU estos números.

Se ejecuta una consulta:

Select * from UBI where UBI between (MINU, MAXU)

O, equivalentemente, where UBI >= MINU and UBI <= MAXU.

En el paso siguiente es necesario ignorar los registros (row) extras que se obtuvieron, de esta manera es posible mostrar solo los que se buscaron.

Método 4. Se recorre la lista de resultados, y para elemento (número de UBI) se ejecuta una consulta y se agrega el resultado al objeto gráfico. Esta opción fue descartada puesto que es obvio que el tiempo de ejecución será mucho mayor.

Archivo plano

A continuación se agregan los métodos que servirían en el caso de que las UBIs se hubieran almacenado como registros de un archivo (plano) con acceso aleatorio.

Método 5. Se recorre la lista de resultados. Para cada elemento, se calcula la posición en el archivo, se accede el registro (por número de UBI) y se agrega a una lista en memoria.

Método 6. Se determina el rango de UBIs involucrado (del MINU al MAXU). Se leen todas estas UBIS en una lectura (se crea un arreglo de UBIs del tamaño indicado por el rango). Se procede el mismo modo que en el método anterior. Se recorre la lista de resultados, se calcula el índice del arreglo correspondiente (restando MINU al número de UBI de la lista) y se agregan los campos al objeto gráfico.

En los dos casos anteriores se usa la siguiente estructura:

```
Public Structure estUnaUBI '' 219 bytes
       Public Ubi As Int32
       Public Clase As Int32
       <VBFixedString (30)> Public título As String
       <VBFixedString (10)> Public fecha As String
       Public tipo_ubi As Int32
       Public tipo_de_anexo As Int32
       Public path_del_archivo As Int32
       <VBFixedString (10)> Public nombre_del_archivo As String
       <VBFixedString (5)> Public extent As String
       Public status As Int32
       <VBFixedString (4)> Public campos_binarios_4 As String
       Public campo_num As Int32
       <VBFixedString (30)> Public campo_lit1 As String
       <VBFixedString (30)> Public campo_lit2 As String
       <VBFixedString (30)> Public campo_lit3 As String
       <VBFixedString (30)> Public campo_lit4 As String
       Public cuantas_marcas_tiene As Int32
```

```
      Public num_user_que_creo_ubi As Int32
      Public nivel_de_confidencialidad As Int32
End Structure
```

No necesariamente es la estructura final que será implementada. Se declararon las siguientes variables:

```
Public UNA_UBI As estUnaUBI, muchas_ubis () As estUnaUBI
```

El procedimiento para realizar la obtención de los archivos seria el siguiente:

a) se determina el menor y menor elemento de LR.
b) Por diferencia se calcula la dimensión de MUCHAS-UBIS (diferencia= mayor – menor).
c) Se redimensiona el arreglo muchas_ubis con la diferencia que se obtuvo en el paso anterior.
d) Se calcula dónde está la UBI número menor elemento de LR.
e) Se abre el archivo en modo binarios.
f) Se realiza la operación FileGet (1, muchas_ubis, menor).

Simulación

En este caso la simulación solo fue realizada en una maquina con las características mostradas en el Cuadro 9:

Cuadro 9 Características de los equipos utilizados para la interpretación de listas.

marca	sistema operativo	memoria RAM Gb	Procesador	velocidad procesador GHz	disco duro
Asus	Windows 10	12	AMD	2.4	SSD

Primero se comparan los tiempos para obtener la lista de UBIs, con los métodos mencionados anteriormente en una base de datos de 1,000,000 de registros y buscando 10,000 registros. Los resultados obtenidos se muestran en el Cuadro 10, cabe mencionar que el proceso se realiza 3 veces continuas para que se muestra una mayor diferencia.

Cuadro 10 Comparación de duración para obtener lista de UBIs a partir de los números de UBI contenidos en una lista de resultados.

	archivo Random	con in	tabla extra	con rango
tiempo 1	1.250082	2.2813989	74.8018201	11.0944032

Como se puede observar las mejores respuestas se obtienen con un archivo con acceso aleatorio (random) y con una consulta SQL que utiliza la cláusula IN.

Se compararon las duraciones de llenar un DataGridView (y luego, un ListView) manualmente (agregando uno tras otro las filas de la tabla de datos) con las obtenidas asignando la tabla como fuente de datos (DataSource) del objeto.

Los resultados de estas comparaciones arrojaron estos resultados:

- Es más rápido llenar un DataGridView que una lista (ListBox o ListView)
- Es más rápido hacerlo asignando la fuente de datos que llenando el objeto uno por uno.

Las duraciones resultantes de las simulaciones efectuadas se muestran en el Cuadro 11.

Cuadro 11 Comparación de duración para poblar los objetos gráficos.

	ListBox		DataGridView	
	uno a uno	DataSource	uno a uno	DataSource
tiempo 1	17.6218091	0.313082	0.7091762	0.0921482

Como resultado de estas simulaciones se decide obtener las listas mediante el uso de una consulta SQL y la cláusula IN, ya que con esta opción es posible implementar el DataSource, propiedad que no es posible utilizar con un archivo random. Para la simulación se utilizara la interfaz mostrada en la Figura 23.

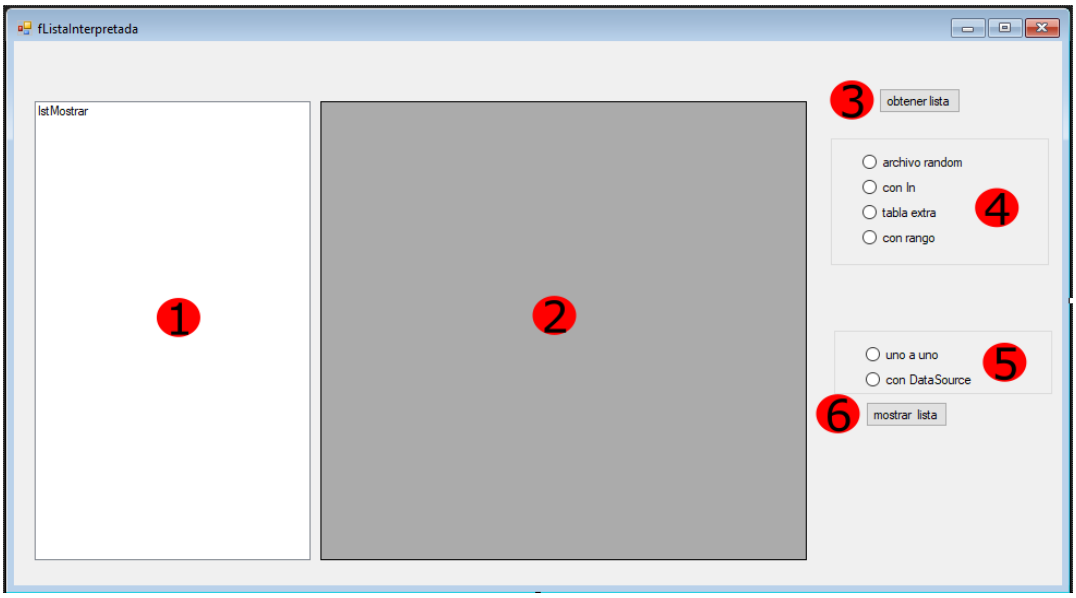


Figura 23 Se muestran las opciones que se consideraron para interpretación de listas, como obtener sus valores y como se mostraran, captura de ejecución.

1. ListBox.
2. DataGridView.
3. Botón para obtener lista.
4. Opciones para escoger la manera de recuperar la lista.
5. Opciones para mostrar una lista.
6. Botón para mostrar lista.

Obtener lista de resultados

Para obtener la lista de UBIS se utiliza la sección de la interfaz que se muestra en la Figura 24.

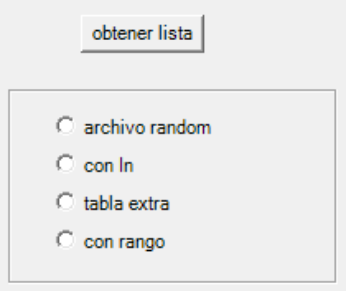


Figura 24 Sección obtener lista de resultados se muestran las opciones consideradas para la obtención de las listas, captura de ejecución.

Para realizar la simulación y obtener los tiempos de ejecución solo se necesita marcar la opción que se desea ejecutar en ese momento y presionar el botón “obtener lista” de esta manera se ejecutaran los procesos que se describen a continuación y se manda un mensaje con la duración de estos.

Debido a que los tiempos son cortos fue necesario ejecutar cada uno de los procesos 3 veces seguidas para así poder obtener mejores estimados de tiempos para realizar la comparación.

Mostrar lista de resultados

Para mostrar la lista solo se necesita la selección de la interfaz entre las que se muestran en la Figura 25.

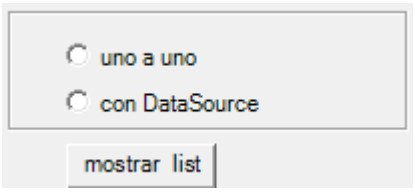


Figura 25 Sección mostrar lista, captura de ejecución.

Se selecciona la opción que se va a probar y posteriormente se presiona el botón. Igualmente en la simulación para obtener la lista los tiempos de ejecución son cortos, es por eso que también se realizan los procedimientos 3 veces seguidas para poder apreciar mejor la diferencia de tiempos.

3.2.5 Lectura de arreglos de estructuras y de números de disco

Comparar uno por uno contra “todo el arreglo”

Para leer un arreglo de un archivo en disco se tienen 2 opciones: leer 1 a 1 y por un segmento. Ambas opciones tienen ventajas o desventajas frente al otro dependiendo el caso que se presente. El primer caso que presenta diferencia es la densidad del arreglo que se lee. Para este caso se utilizara el ejemplo siguiente: en

un archivo con 30 registros, es necesario obtener los siguientes registros “4, 6, 10, 15, 18, 20, 24”. Si se lee uno a uno, solo se leerían **7 x tamaño del registro,** mientras que si se lee por rango serian **21 x tamaño del registro,** como se ve en la Figura 26.

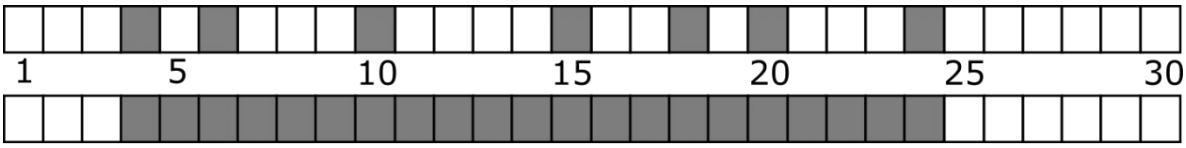


Figura 26 Leer un archivo por segmento implica traer consigo registros que no necesariamente son utilizados, elaboración propia.

Esto parecería no tener mucha importancia, pero dependiendo de la cantidad de registros que se lee, la densidad del rango y el tamaño o tipo del registro, hace que sea más rápido leer uno a uno, la única circunstancia en la que leer un rango completo siempre va a ser más rápida es que los registros que se leen sean continuos, caso que difícilmente se presenta.

Arreglo de números

Al ser un número un tipo de datos primitivo (definido por Visual Basic), el compilador tiene las rutinas necesarias para ejecutar la lectura de la manera más rápida posible. En este caso no importa la densidad del rango que se desea leer: la lectura siempre será más rápida por segmento que uno a uno, ya que si es posible leer un segmento de memoria completo de una sola petición, esto representa una gran ventaja para guardar y leer registros. El caso en el que se guardan objetos (que contienen cadenas de caracteres y no sólo números) es distinto: aunque se indique que se lea todo un arreglo, las lecturas se hacen uno por uno cada elemento del arreglo.

Arreglo de estructuras

En el caso de almacenar los registros en algún tipo de estructura (tipo de dato definido por el usuario), la lectura no importa cuál sea la intención, leer uno a uno o

por segmento, los tiempos resultantes sugieren que la lectura siempre se realiza registro a registro, y considerando el caso que se presentó en la Figura 26 el hecho de leer más de los registros deseados hace que leer por segmento siempre sea más lento.

Otro caso a considerar es la utilización de Strings dentro de una estructura, una condición necesaria es asignar los Strings una longitud fija para poder recuperar el registro posteriormente. Las pruebas efectuadas indican que agregar una cadena los tiempos de lectura aumenta considerablemente.

Programa para comparar duraciones de lecturas de datos de disco

Lógica (de ejecución)

1. Se indica el número de repeticiones (para obtener datos más precisos).
2. Se ejecuta el proceso con N elementos del arreglo.
    - Para N = 1000,10,1000, 100,000, 1,000,000 y 10,000,000.
        i. Para cada uno se calculan las duraciones.
            1. ESTRUCTURAS (10 campos enteros).
                a. 1 x 1
                b. Como segmento
            2. NÚMEROS enteros de 4 bytes.
                a. 1 x 1
                b. Como segmento

Para el desarrollo de la simulación se utilizó la interfaz mostrada en la Figura 27

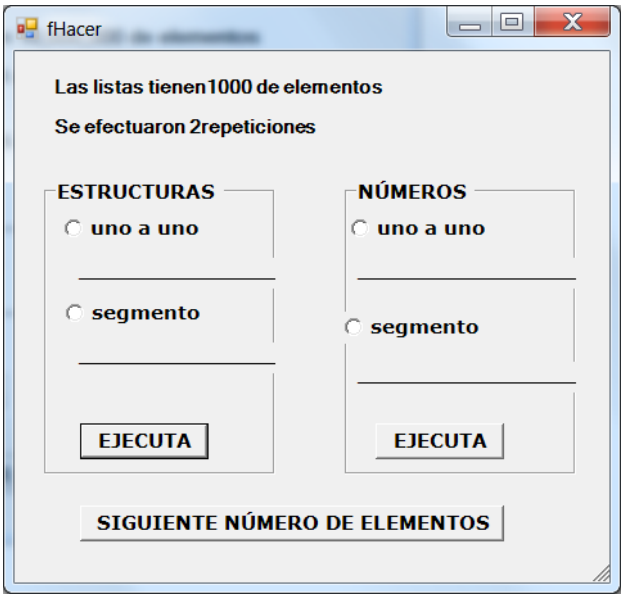


Figura 27se muestran las opciones se implementaron para la lectura de arreglos de números o estructuras, captura de ejecución.

Para cada proceso, se registran las duraciones (del número indicado de repeticiones). Se despliegan todas las duraciones en una forma como la que se ilustra en las figuras que siguen.

Resultados

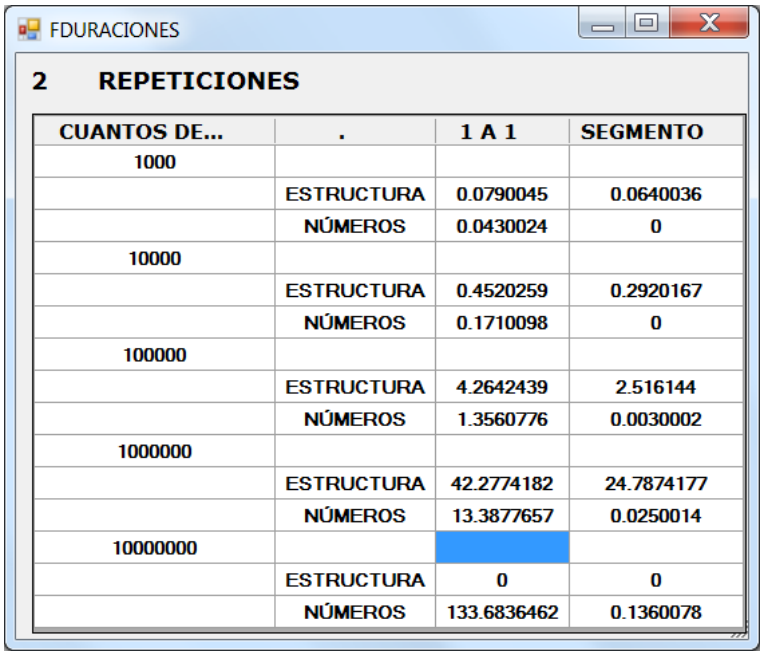
Finalmente, se ejecutarían los programas en 4 computadoras, las especificaciones de cada una se muestran en el Cuadro 12.

Cuadro 12 Computadoras utilizadas para la simulación.

	Marca	sistema operativo	memoria RAM Gb	procesador	velocidad procesador GHz	disco duro
2	Sony	Windows 8	8	Intel	1.6	SSD
4	Asus	Windows 10	12	AMD	2.4	SSD
1	Hp	Windows 7	4	AMD	2.8	SSD
3	Hp	Windows 7	4	AMD	3.8	HDD

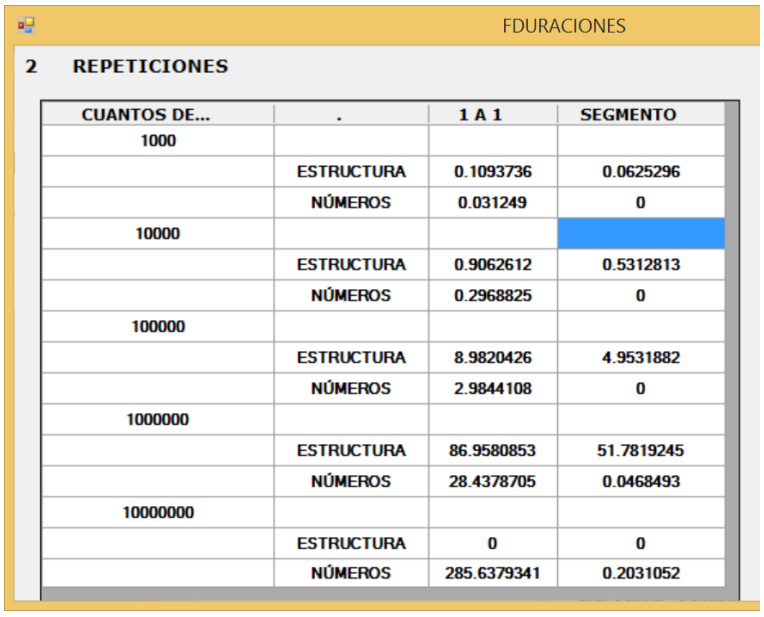
Se presentan las duraciones obtenidas en cada uno de los 4 equipos.

En la Figura 28 Tiempos en equipo hp AMD 2.8, Figura 29 Tiempos en equipo Sony Intel 1.6, Figura 30 Tiempos en equipo hp AMD 3.8 y Figura 31 Tiempos en equipo Asus AMD 2.4.



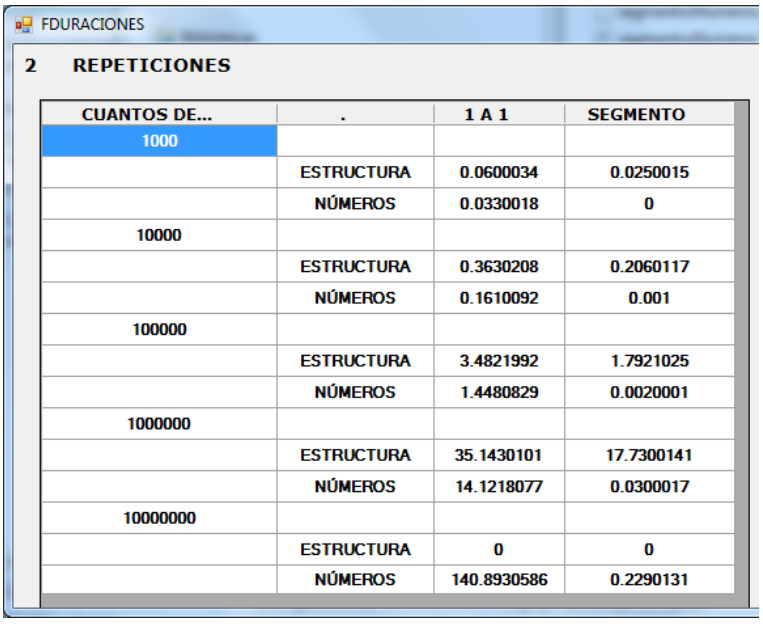
CUANTOS DE...	.	1 A 1	SEGMENTO
1000			
	ESTRUCTURA	0.0790045	0.0640036
	NÚMEROS	0.0430024	0
10000			
	ESTRUCTURA	0.4520259	0.2920167
	NÚMEROS	0.1710098	0
100000			
	ESTRUCTURA	4.2642439	2.516144
	NÚMEROS	1.3560776	0.0030002
1000000			
	ESTRUCTURA	42.2774182	24.7874177
	NÚMEROS	13.3877657	0.0250014
10000000			
	ESTRUCTURA	0	0
	NÚMEROS	133.6836462	0.1360078

Figura 28 Tiempos en equipo hp AMD 2.8, captura de ejecución.



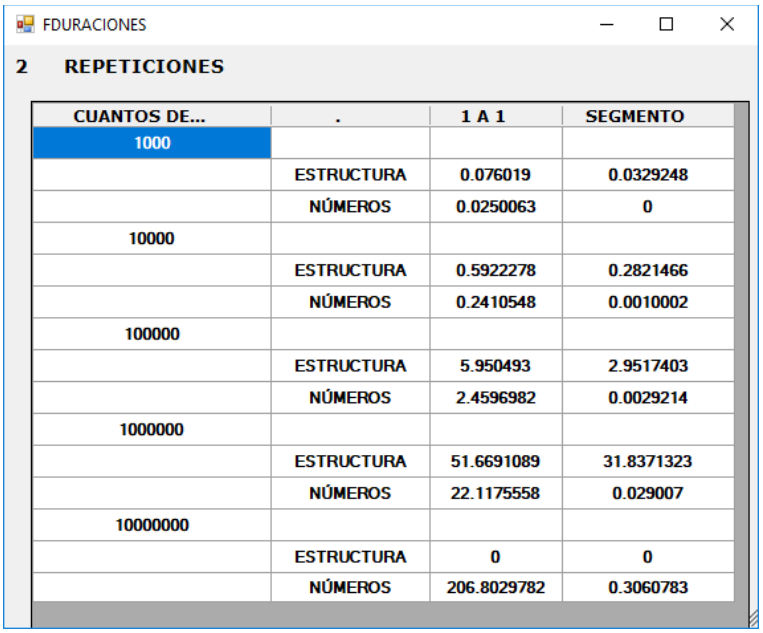
CUANTOS DE...	.	1 A 1	SEGMENTO
1000			
	ESTRUCTURA	0.1093736	0.0625296
	NÚMEROS	0.031249	0
10000			
	ESTRUCTURA	0.9062612	0.5312813
	NÚMEROS	0.2968825	0
100000			
	ESTRUCTURA	8.9820426	4.9531882
	NÚMEROS	2.9844108	0
1000000			
	ESTRUCTURA	86.9580853	51.7819245
	NÚMEROS	28.4378705	0.0468493
10000000			
	ESTRUCTURA	0	0
	NÚMEROS	285.6379341	0.2031052

Figura 29 Tiempos en equipo Sony Intel 1.6, captura de ejecución.



CUANTOS DE...	.	1 A 1	SEGMENTO
1000			
	ESTRUCTURA	0.0600034	0.0250015
	NÚMEROS	0.0330018	0
10000			
	ESTRUCTURA	0.3630208	0.2060117
	NÚMEROS	0.1610092	0.001
100000			
	ESTRUCTURA	3.4821992	1.7921025
	NÚMEROS	1.4480829	0.0020001
1000000			
	ESTRUCTURA	35.1430101	17.7300141
	NÚMEROS	14.1218077	0.0300017
10000000			
	ESTRUCTURA	0	0
	NÚMEROS	140.8930586	0.2290131

Figura 30 Tiempos en equipo hp AMD 3.8, captura de ejecución.



CUANTOS DE...	.	1 A 1	SEGMENTO
1000			
	ESTRUCTURA	0.076019	0.0329248
	NÚMEROS	0.0250063	0
10000			
	ESTRUCTURA	0.5922278	0.2821466
	NÚMEROS	0.2410548	0.0010002
100000			
	ESTRUCTURA	5.950493	2.9517403
	NÚMEROS	2.4596982	0.0029214
1000000			
	ESTRUCTURA	51.6691089	31.8371323
	NÚMEROS	22.1175558	0.029007
10000000			
	ESTRUCTURA	0	0
	NÚMEROS	206.8029782	0.3060783

Figura 31 Tiempos en equipo Asus AMD 2.4, captura de ejecución.

Se compararían los tiempos y en especial, se tomarían en cuenta las diferencias para formular los criterios de almacenamiento adoptados para el DBB-V18.

El resto de este capítulo se presenta en secciones de acuerdo al objetivo de cada una de las simulaciones.

CONCLUSIONES

1. EL uso de un disco de estado sólido no redujo los tiempos de proceso en forma significativa. Este resultado fue lo suficientemente sorprendente para que se ejecutaran cientos de procesos, que confirmaron esta afirmación.
2. La velocidad de proceso es el único factor que es determinante en la duración de los procesos.
3. No conviene el uso de estructuras (tipos de variable definidas por el usuario) grabadas en disco.
4. **La conclusión más importante es:** Siempre que se pueda, se deben leer los datos en conjunto, y no uno por uno. Es decir si se carga un arreglo de disco a memoria, se debe leer el ARREGLO completo en lugar de leer uno por uno sus componentes.

CAPÍTULO 4. CLASES

4.1 Introducción

Una UBI pertenece a una clase de las definidas para la aplicación en particular. Todas las UBIs están (en la estructura seleccionada para almacenar las UBIs, descrita en capítulo correspondiente) en una Tabla de una base de datos relacional. Esta tabla es única (lógicamente), aunque pudiera estar compuesta por varias tablas con la misma estructura si el número de unidades fuera muy grande. Las UBIs están numeradas con un número único.

De ese modo en DBB no se habla de “tablas” de la base de datos en relación con las UBIs. Como pueden ser de naturaleza muy distinta (una UBI se puede referir a un auto, otra a un artículo científico, un maestro de una escuela, o cualquier otra cosa) se especifica para cada UBI de qué tipo de registro se trata (como si fuera en cuál tabla está). Esto se hace mediante **CLASES**, de modo que una UBI es una instancia de una de las clases.

4.2 Modelo de datos de una clase

Se presentan los campos numerados en el Cuadro 13 para permitir encontrar la explicación de los campos marcados con un **“?”** en la siguiente sección. Los nombres de los campos se proporcionan para este fin: no son los que se usan en las estructuras.

Cuadro 13 Campos de una clase.

No.		NOMBRE (O SIGNIFICADO)	TIPO DE DATO
1		**DATOS GENERALES**	
1.1		número asignado a la clase	Ent-4
1.2		nombre o descripción de la clase en inglés	
1.3		nombre o descripción de la clase en español	
1.4	**?**	a cuál grupo de clases pertenece	
2	**?**	**PARA NUMERACIÓN DE UBIS (OPTATIVO)**	
2.1		inicio_rango	Ent-4
2-2		cuantos_numeros	Ent-4
2.3		fin_rango	Ent-4
2.4		mayor_ubi_esta_clase	Ent-4
3	**?**	**FECHA – CÓMO LA USA**	
3.1		nombre asignado a la fecha	Strings
3.2	**?**	que_es_fecha	Byte (código)
4	**?**	**OFRECE 4 CAMPOS BINARIOS (0/1)**	
4.1		usa alguno de los campos binarios	0/1
4.2		usa_este_bin1 ‘ son 4 campos iguales2,3,4	4 de 0/1
4.3		contexto_este_bin1 ‘ son 4 campos iguales 2,3,4	4 de byte
4.4		nom_este_bin1 ‘ son 4 campos iguales 2,3,4	4 de Strings
5	**?**	**OFRECE 4 CAMPOS LITERALES Y 1 NUMÉRICO**	
5.1		usa_algun_campo_general	0/1
5.2		usa_campo_lit1 ‘ son 4 campos iguales 2,3,4	4 de 0/1
5.3		usa_campo_num	0/1
5.4		nombre_campo_lit1 ‘ son 4 campos iguales 2,3,4	4 de Strings
5.5		nombre_campo_num	Strings
6	**?**	**PROPIEDADES PARA “MARCAR” LAS UBIS**	
6.1		Cuantos grupos de contextos ofrece	Byte
6.2		Los grupos de contextos ofrecidos	Map_digítos
6.3		cuan_contextos_ofrece	Byte
6.4		contextos_ofrecidos	Map_digítos
6.5		el_contexto_es_obligatorio	Map_digítos
7		**OPCIONES DIVERSAS**	
7.1		idioma_default	byte
7.2	**?**	ofrece_numeracion_manual	0/1
8		**PROTECCIÓN CONTRA CAMBIOS NO AUTORIZADOS**	
8.1	**?**	Audit1, audit2	2 de ent4

Ent4 es un entero de 4 bytes
Map_digítos es una cadena que se interpreta como un conjunto de dígitos

4.2.1 Qué significan los campos de una clase de UBIs

Sólo se explican en el Cuadro 14 aquéllos para los cuales el nombre no indica su significado.

Cuadro 14 Significado de campos de una clase.

1.4	a cuál grupo de clases pertenece	Para poder ofrecer sólo algunas de las (máximo 50) clases se introducen Grupos de Clases
2	PARA NUMERACIÓN DE UBIS (OPTATIVO)	Se pueden definir rangos de números de UBI de una clase
3	FECHA – CÓMO LA USA	Ofrece una fecha (se indica un código que indica qué es, y un nombre de la fecha que aparece cuando se la usa
4	OFRECE 4 CAMPOS BINARIOS (0/1)	Son 4 campos 0/1 que se pueden utilizar para usos temporales(a cada uno se le asigna un nombre)
5	OFRECE 4 CAMPOS LITERALES Y 1 NUMÉRICO	Hay 4 campos literales y 1 numérico que se pueden o no usar (a cada uno se le asigna un nombre)
6	PROPIEDADES PARA "MARCAR" LAS UBIS	Se pueden seleccionar contextos que se ofrecen para UBIs de la clase. Para permitir limitar la lista de éstos, se introdujo el concepto de grupos de contextos
7.1	Idioma default	El DBB usa 2 idiomas (1 = inglés, 2=español) Las UBIs usan en idioma indicado aquí si no hay una especificación posterior (a nivel de la UBI misma)
7.3	ofrece_numeracion_manual	La numeración es por rangos, automática o manual (el usuario asigna el número de UBI)
8.1	Audit1, audit2	Permite proteger los datos contra modificaciones no autorizadas

Observaciones sobre las estructuras seleccionadas:

1. Casi todas las funciones relacionadas con las UBIs mismas necesitan las especificaciones de la clase a la cual pertenecen. De ese modo, conviene ofrecerlas en una estructura en memoria.
2. Debido a la gran cantidad de campos, el uso de tablas relacionales es posible, pero ofrecerá algunas desventajas.
3. Las estructuras seleccionadas deben permitir adiciones (es decir, clases agregadas en cualquier momento).
4. La selección de la estructura adoptada para las clases no debe tomar en cuenta la sencillez o dificultad de la programación que involucrada; en cambio naturalmente deberá contemplar la posibilidad de implementar la estructura en la plataforma seleccionada (Visual Studio Net).
5. No son importantes las consideraciones de espacio en disco o memoria, puesto que al haber un máximo de 50 clases, el tamaño no tendrá impacto sobre el rendimiento del sistema.

4.3 Alternativas consideradas para almacenar las clases

Alt 1. Una tabla de una DBB relacional; la tabla tendría todos los campos de la clase.

Alt 2. Varias tablas relacionadas en las cuales se incluirían los diversos datos de la clase.

Alt 3. Un archivo con acceso “pseudo-aleatorio” en el que cada registro contuviera más de 1 variable de estructuras en las cuales se separarían los campos de la clase.

Alt 4. Varios archivos con acceso aleatorio; se usarían en “PARALELO” es decir el mismo número de registro, en cada archivo, contendría datos de la misma clase.

Alt 5. Una lista ordenada de una estructura, en la cual estuvieran todas las propiedades de la clase.

Alt 6. Una lista ordenada de una o más estructuras en cada elemento, donde los campos se agruparían en las diversas estructuras.

Alt 7. Dos o más listas ordenadas, que se usarían en forma paralela, es decir, los elementos de cada lista con el mismo índice corresponderían a la misma clase.

Alt 8. Un árbol, donde los nodos tenían los campos de la clase (y no sólo el elemento de búsqueda).

Se eliminaron las alternativa 5, 6, 7 y 8 por no permitir algunas de las operaciones impuestas a las clases por la funcionalidad requerida, o porque es clara que la contribución a la funcionalidad o eficiencia en prácticamente nula. En este punto no se detalla esa estructura; sólo se indica que no se tomó en cuenta en la comparación que llevaría a la selección de la estructura adoptada para el DBB.

4.4 Descripción de las alternativas

4.4.1 Alternativas 1 y 2

Alternativa 1: Una tabla de una DBB relacional; la tabla tendría todos los campos de la clase.

Alternativa 2: Varias tablas relacionadas en las cuales se incluirían los diversos datos de la clase.

Se presenta el Cuadro 15 con todos los elementos (uno por uno en el caso de varios del mismo tipo).

Cuadro 15 Elementos de una clase.

Campos “fijos”	“Arreglos”	Campos de uso variable
Número de la clase	Usa_algun_campo_binario	Fecha (Nombre)
Descripción en inglés	Usa el binario 1	Fecha qué es (código)
Descripción en español	Nombre binario 1	Usa algún campo general
Grupo de clases	Usa el binario 2	Usa campo literal 1
Inicio_rango	Nombre binario 2	Nombre campo literal 1
Cuantos_numeros	Usa el binario 3	Usa campo literal 2
Fin_rango	Nombre binario 3	Nombre campo literal 2
Mayor_ubi_esta_clase	Usa el binario 4	Usa campo literal 3
Ofrece equivalencias	Nombre binario 4	Nombre campo literal 3
Idioma default	Cuantos grupos de contextos usa	Usa campo literal 4
Ofrece numeración manual	Grupo contextos 1	Nombre campo literal 4
Cifra 1 de “auditoría”	Grupo contextos 2	Usa campo numérico
Cifra 2 de “auditoría”	Grupo contextos 3	Nombre del campo numérico
	Grupo contextos 4	
	Cuantos contextos ofrece	
	Contexto ofrecido 1	
	Contexto 1 es obligatorio	
	….. Otros 14 pares de campos	
	Contexto ofrecido 16	
	Contexto 16 es obligatorio	
Número de campos		
13	42	13

Resumen: La tabla tendría 68 campos.

El uso implicaría:

a) Tener muchas instrucciones para manejar las columnas de la *DataTable* correspondiente. Se puede obviar esta dificultad: en las rutinas se usarían los números de los campos, de modo que se pudieran contemplar algunos como arreglos de campos o de estructura de campos (duplas).

Una alternativa es usar más de una tabla para almacenar los campos.

Se presenta solo un diseño (hay otros diseños obtenidos combinando varios campos en una misma tabla):

Tabla 1: La UBI Misma (1 registro por UBI)
Tabla 2: los campos binarios (4) de la UBI (4 registros por UBI)
Tabla 3: los grupos de contextos que ofrece (máximo 4 registros por UBI)
Tabla 4: los contextos que ofrece (Máximo 16 registros por UBI)
Tabla 5: los campos variables que ocupa (5 registros por UBI)

4.4.2 Alternativas 3 y 4

Alternativa 3: Un archivo con acceso “pseudo-aleatorio” en el que cada registro contuviera más de 1 variable de estructuras en las cuales se separarían los campos de la clase.

Alternativa 4: Varios archivos con acceso aleatorio; se usarían en “PARALELO” es decir el mismo número de registro, en cada archivo, contendría datos de la misma clase.

En el Cuadro 16 se describe la “estructura” con todos los campos que se grabaría como registro de tamaño fijo en un archivo que se usaría en modo aleatorio. Observe que se listan variables de la estructura en 3 columnas para facilitar su lectura. Una variable seguida de (n) indica que se trata de un arreglo de n campos, pero que se dimensionan como parte de la estructura.

Structure t_campos_de_una_clase

Cuadro 16 Estrucutura para para archivo CLASES.

Campos "filos"	**"Arreglos"**	**Campos de uso variable**
Número de la clase	Usa_algun_campo_binario	Fecha (Nombre)
Descripción en inglés	Usa el binario (4)	Fecha qué es (código)
Descripción en español	Nombre binario (4)	Usa algún campo general
Grupo de clases	Cuantos grupos de contextos usa	Usa campo literal (4)
Inicio_rango	Grupo contextos (4)	Nombre campo literal(4)
Cuantos_numeros	Cuantos contextos ofrece	Usa campo numérico
Fin_rango	Contexto ofrecido (16)	Nombre del campo numérico
Mayor_ubi_esta_clase	Contexto es obligatorio (16)	
Ofrece equivalencias		
Idioma default		
Ofrece numeración manual		
Cifra 1 de "auditoría"		
Cifra 2 de "auditoría"		

Se agregan las rutinas para redimensionar los arreglos (aunque sean de longitud fija, es necesario dimensionarlos en forma individual, es decir, para cada clase de la estructura).

Se crea un arreglo de CLASES con esta estructura en memoria, o se usa el registro en disco cada vez que se necesite la especificación de una clase.

Observación importante: En VB.Net no se puede grabar en disco una variable con una estructura de tamaño variable debido a algún arreglo. Esto prácticamente elimina esta estructura como opción.

Las estrategias adoptadas para obviar esta restricción incluyen:

- usar más de una estructura y grabar una tras otra como parte del mismo registro (la Alternativa 4).
- usar algún artificio para reemplazar un arreglo por una única variable (alternativa agregada a la lista que se describirá por separado).

Formulemos un diseño correspondiente a la alternativa 4 en el Cuadro 17

Structure t_campos_unicos_de_una_clase

Aquí “únicos” indica que no se trata de un arreglo

Cuadro 17 Estrucutura para alterantiva 4 Clases.

Campos “filos”	**“Arreglos”**	**Campos de uso variable**
Número de la clase	Usa_algun_campo_binario	Fecha (Nombre)
Descripción en inglés		Fecha qué es (código)
Descripción en español		Usa algún campo general
Grupo de clases	Ofrece numeración manual	Usa campo numérico
Inicio_rango	Cuantos grupos de contextos usa	Nombre del campo numérico
Cuantos_numeros	Cuantos contextos usa	
Fin_rango	Cifra 1 de “auditoría”	
Mayor_ubi_esta_clase	Cifra 2 de “auditoría”	
Ofrece equivalencias		
Idioma default		

Structure t_campos_literales
Usa_el_campo
Nombre del campo
Se grabaría un arreglo de 4 variables de este tipo

Structure t_campos_binarios

- Usa_este_binario
- nombre de este binario

Se grabaría un arreglo de 4 variables de este tipo

Structure t_grupos_de_contextos

- Numero de grupo de contextos
- contexto

Se grabaría un arreglo de 4 variables de este tipo

Structure t_contextos_que_ofrece

- Numero de contexto
- es obligatorio

Se grabaría un arreglo de 16 variables de este tipo

Una alternativa que utiliza un artificio que resulta eficiente en muchos diseños es: Se reemplaza la estructura de los grupos de contexto por una variable de tipo String (cadena) de longitud fija (8): Hay un máximo (por diseño del paquete) de 32 grupos

de contextos. Concatenamos los 4 números de grupo (con 2 dígitos cada uno) en una variable cadena, y la agregamos a la estructura de los campos de la clase.

Grupos_que_usa (cadena de 8 bytes)
Procedemos del mismo modo con los 16 contextos.

Contextos que usa (cadena de 48 caracteres) obtenida

Concatenando el número de contexto (3 dígitos) de los 16 contextos posibles. Los espacios no utilizado se completan con ceros (hay un campo que dije cuántos contextos usa la clase).
Y luego se crea otra variable

Los contextos son obligatorios (cadena de 16 caracteres)

Un carácter 1 en una posición de esta cadena indica que el contexto correspondiente de la variable contextos que usa es obligatorio, es decir en una UBI que utiliza (ofrece) ese contexto, se deberá asignarle un valor.

4.5 Descripción de las estructuras seleccionadas para almacenar y usar las clases

Un archivo con acceso “pseudo-aleatorio” en el que cada registro contuviera más de 1 variable de estructuras en las cuales se separarían los campos de la clase.

```
Public Structure TCAMPOBINARIO ' Longitud es 18
      Public USA_ESTE_BIN As Byte ' 0/1
      Public CONTEXTO_ESTE_BIN As Byte ' 0 = NO LO MARCA
      <VBFixedString (16)> Public NOM_ESTE_BIN as String
End Structure
Public los_4_binarios_de_cada_clase (50, 4) As TCAMPOBINARIO ' no usamos el 0
Public Structure tuna_clase ' cada clase ocupa 282 bytes
      Public NUMCLASE as Byte ' 1 a 50
      <VBFixedString (24)> Public descrip_ingles As String
      <VBFixedString (24)> Public descrip_espanol As String
      Public GRUPO_CLASES_ESTA_CLASE As Byte
      '----------------------------------------
      ‘PARA NUMERACIÓN DE SUS UBI'S
      '----------------------------------------
      Public INICIO_RANGO as Int32
      Public CUANTOS_NUMEROS as Int32
      Public FIN_RANGO as Int32
      Public MAYOR_UBI_ESTA_CLASE As Int32 ' EL NÚMERO DE LA UBI
      '----------------------------------------
      ‘CÓMO USA LOS CAMPOS DE LA TABLA DE UBIS
      '----------------------------------------
      <VBFixedString (16)> Public nombre_fECHA As String
      Public que_es_Fecha As Byte ' se usa un código
      Public USA_ALGUN_CAMPO_BINARIO As Byte
```

```
        Public USA_ALGUN_CAMPO_GENERAL As Byte
        Public usa_campo_lit1 As Byte ' 0/1
        Public usa_campo_lit2 As Byte ' 0/1
        Public usa_campo_lit3 As Byte ' 0/1
        Public usa_campo_lit4 As Byte ' 0/1
        Public usa_campo_num As Byte ' 0/1
        <VBFixedString (16)> Public nombre_campo_lit1 As String
        <VBFixedString (16)> Public nombre_campo_lit2 As String
        <VBFixedString (16)> Public nombre_campo_lit3 As String
        <VBFixedString (16)> Public nombre_campo_lit4 As String
        <VBFixedString (16)> Public nombre_campo_num As String
        '----------------------------------------
        ‘que ofrece para marcar
        '----------------------------------------
        Public cuan_grupos_contextos_ofrece As Byte
        <VBFixedString (8)> Public grupos_contextos_ofrecidos As String
        ‘hasta 4 grupos; se rellena con ceros 2 dígitos cada un8
        Public cuan_contextos_ofrece As Byte
        <VBFixedString (48)> Public contextos_ofrecidos As String
        ‘hasta 16 contextos 3 dígitos cada uno SE RELLENA CON CEROS
        <VBFixedString (16)> Public el_contexto_es_obligatorio As String
        ‘un digito cada contexto paralelo al de contextos ofrecidos
        '----------------------------------------
        ‘opciones
        '----------------------------------------
        Public ofrece_equivalencias As Byte
        Public idioma_default As Byte
        Public ofrece_numeracion_manual As Byte ' 0 = no, automática
        Public ufu_byte As Byte
        Public ufu_num As Int32
        <VBFixedString (16)> Public ufu_texto As String
        Public audit1 As Int32
        Public audit2 As Int32
End Structure
```

CAPÍTULO 5. CONTEXTOS Y VALORES

5.1 Conceptos

Para marcar una UBI, es decir, agregarle un descriptor, se utiliza un par (Contexto, Valor). El contexto indica el significado o la interpretación del valor. Por ejemplo, el par (4, “JUAN”) podría indicar 4 - Nombre; otro par (14, “SEGURA”) correspondería al contexto 14 – Apellido.

Se presentan ambos tipos de datos juntos puesto que naturalmente hay relaciones entre ellos.Cada contexto utilizado debe estar en un catálogo de contextos (o conjunto de contextos declarados válidos). Además de otros datos, los elementos principales de un contexto son el número de contexto y una descripción para facilitar su uso.

Cada valor del contexto está en (o en ocasiones, será incorporado al) catálogo de VALORES de ESE CONTEXTO. En otras palabras, el valor será válido para ese contexto. Es importante señalar que el mismo término (palabra, número, frase) puede ser también válido en otros contextos. Por ejemplo el valor “AUTOMATICA” puede ser un tipo de transmisión de un auto, un modo de comunicar una solicitud, el tipo de contratación de un empleado, etc.

5.2 El modelo conceptual (las entidades)

El modelo adoptado se describe en la Figura 32 y Figura 33.

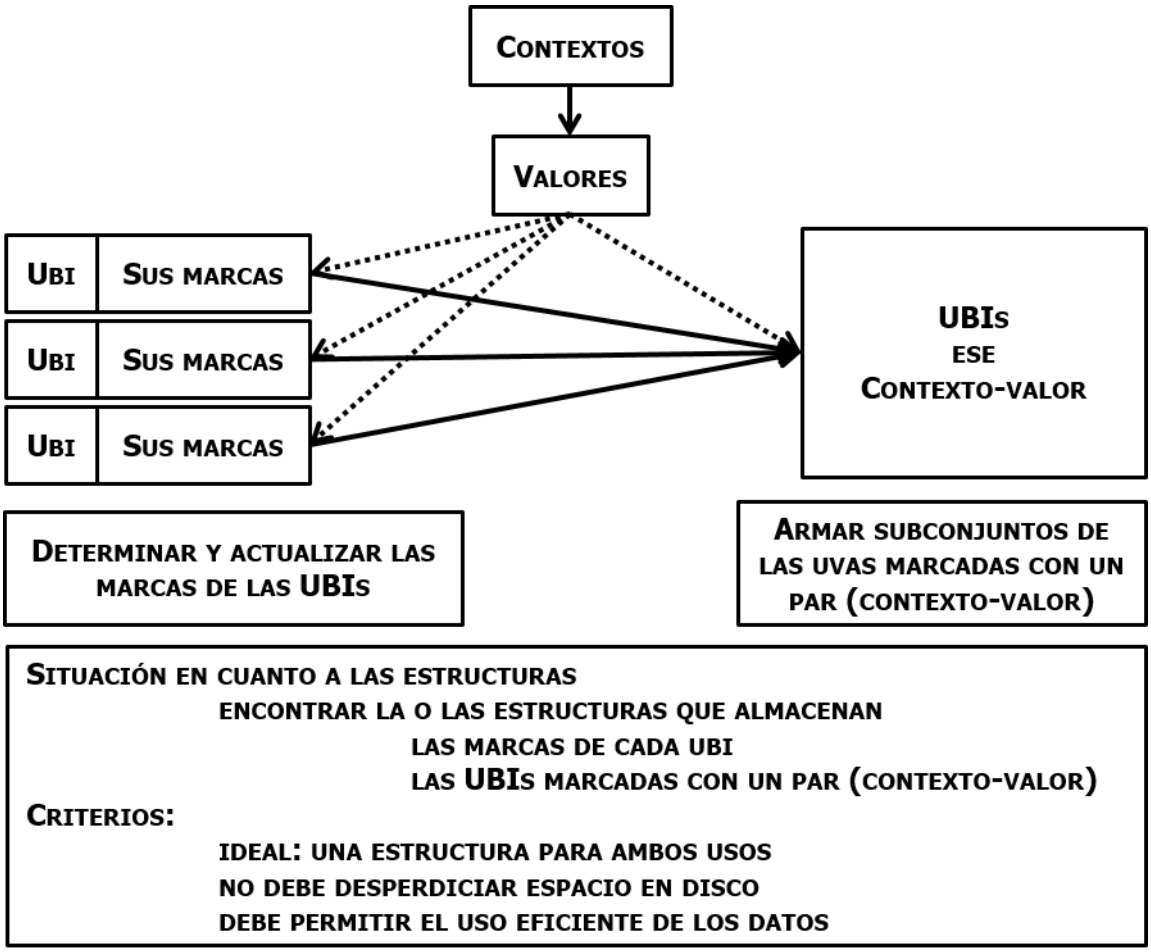


Figura 32 Modelo conseptual contextos y valores, cada UBI tiene sus propias marcas y cada marca indicara a que ubi pertenece, elaboración propia.

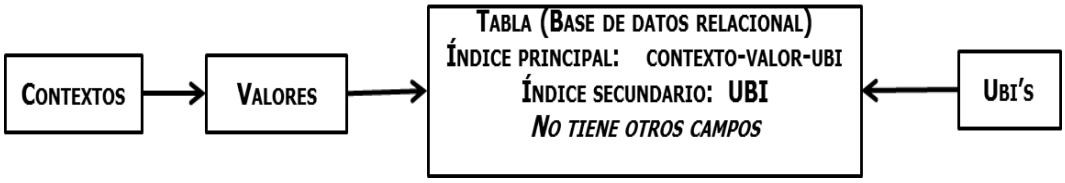
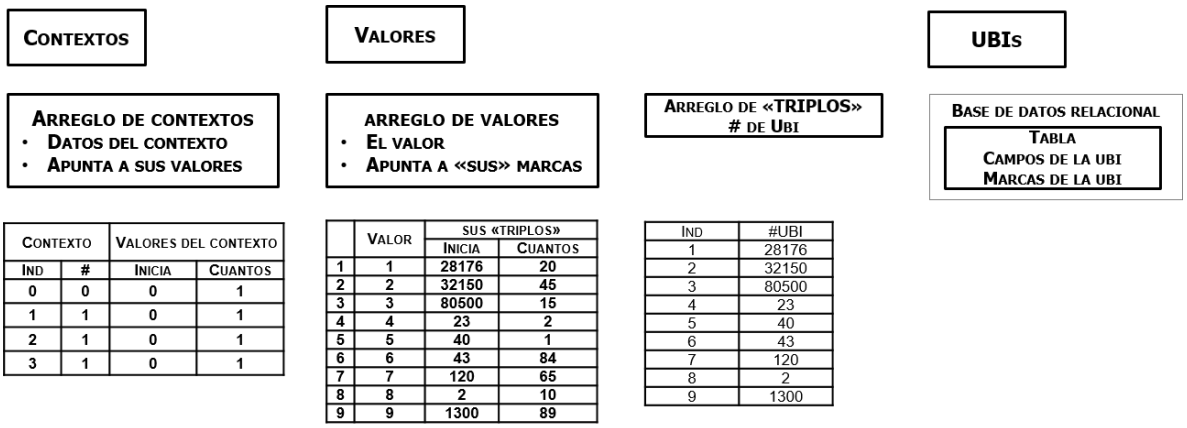


Figura 33 los contextos y sus valores seran almacenados en una tabla de base de datos, elaboración propia.

En la Figura 34 se muestran los elementos principales de las estructuras adoptadas para la presente versión del DBB.



CONTEXTO		VALORES DEL CONTEXTO	
IND	#	INICIA	CUANTOS
0	0	0	1
1	1	0	1
2	1	0	1
3	1	0	1

	VALOR	SUS «TRIPLOS»	
		INICIA	CUANTOS
1	1	28176	20
2	2	32150	45
3	3	80500	15
4	4	23	2
5	5	40	1
6	6	43	84
7	7	120	65
8	8	2	10
9	9	1300	89

IND	#UBI
1	28176
2	32150
3	80500
4	23
5	40
6	43
7	120
8	2
9	1300

Figura 34 Estrucutras adoptadas, elaboración propia.

5.3 La solución “evidente”

Las entidades del modelo conceptual indican que hay una solución que consiste en que se incluyen 3 tablas de una base de datos relacional. En el resto de este capítulo se denomina esta solución como la “evidente”. Sólo se detallan los campos principales de estas tablas. Se proporciona el índice principal y si se incluiría otro, el secundario.

TABLA CONTEXTOS

Índice principal: número de contexto

Campos: descripciones, clasificaciones, restricciones (**)

** Las restricciones del contexto se refieren a cuales bases de datos lo usan, y en cuales clases se ofrecen. Esto será motivo de descripciones y considerandos en la sección correspondiente a los contextos.

TABLA VALORES

- Índice principal: número de contexto, valor
- Campos: si está pendiente de autorización

TABLA MARCAS

- Índice principal: número de contexto, valor, UBI.
- Índice secundario: UBI.

Como se verá, estas estructuras resuelven el tema de almacenar los datos, y tiene la ventaja de que las marcas (contexto, valor, UBI) se pueden utilizar tanto para determinar las que tiene una UBI como las que corresponden a un par (contexto, valor). Sin embargo, tiene varias desventajas que se discutirán cuando se la compara con otras estructuras que se podrían utilizar para implementar el modelo conceptual, es decir, las entidades y relaciones entre ellas.

Esto es el origen de esta investigación cuyo objetivo es encontrar las estructuras más convenientes para almacenar los datos en cuestión.

En el Capítulo 3 se mostraron los resultados de una simulación en la que se compara el uso de la tabla única (contexto-valor-ubi) con las estructuras seleccionadas.

5.4 Contextos

Se han descrito ya las funciones de los contextos en cuanto a que constituyen el elemento principal de clasificación de las marcas, los triplos (contexto, valor, UBI). A continuación se detallan los aspectos de los contextos que intervienen en la funcionalidad de los mismos y que jugaron un papel principal en la selección de la estructura seleccionada para almacenarlos y usarlos en los programas.

5.4.1 Modelo de datos DE UN CONTEXTO

En el Cuadro 18 se presentan los datos (o atributos) de un contexto. Se dividieron en grupos para facilitar su lectura. A continuación se explican los campos marcados en la segunda columna con un signo de interrogación. El tipo de dato se incluyó puesto que puede ampliar la información sobre la naturaleza de los campos.

Cuadro 18 Los datos de un contexto.

No.		NOMBRE (O SIGNIFICADO)	TIPO DE DATO
1		**DATOS GENERALES**	
1.1		Num_contexto	Ent-4
1.2		Descripción en inglés	X(24)
1.3		Descripción en español	X(24)
1.4		Abreviada en inglés	X(12)
1.5		Abreviada en español	X(12
1.6	**?**	Su grupo de contextos	Byte
1.7	**?**	Tipo de contexto	byte
2		**RESPECTO A SUS VALORES**	
2.1		Valores son numéricos	0/1
2.2	**?**	El valor único	Ent 4
2.3	**?**	Cuantos valores tiene	Ent 4
2.4	**?**	Cuantos valores incluyendo UFUs	Ent 4
2.5		Posición inicial en el arreglo de valores	Ent 4
3		**RESTRICCIONES DE USO (U OFERTA)**	
3.1	**?**	Tiene restricciones en cuando a BDD	0/1
3.2	**?**	Tiene restricciones en cuando a clases	0/1
3.3	**?**	Lo usa la base	String 9
3.4	**?**	Se ofrecen en la clase	String 50
4	**?**	**PROPIEDADES PARA "MARCAR" LAS UBIS**	
4.1	**?**	Como se almacenan sus marcas	Byte
4.2	**?**	Valor no marcable 1	X(16)
4.3		Valor no marcable 2	X(16)
4.4		Ufu texto	X(16)
4.5		Ufu numérico	Ent 4

Cada contexto ocupa 206 bytes.

A continuación se describen los campos en cuanto a su significado. Se omitieron aquellos en los cuales el nombre es suficiente para entender su interpretación.

A cuál grupo de contextos pertenece: Se crean grupos de contextos (máximo 32 grupos) para permitir al usuario limitar la lista de contextos de acuerdo al grupo. Esto puede ser de utilidad cuando hay que usar un contexto y para ello elegirlo de la lista de éstos.

Valores son numéricos: Puede ser que un contexto sólo contenga valores numéricos. Por ejemplo si un dato está codificado (un grupo de edad, un país) o se trate de una referencia a algún elemento ya sea del mismo modelo (una UBI, por ejemplo) o a un registro de alguna otra base de datos. En este caso, el contexto no

admitirá que se agreguen valores que no sean numéricos; sin embargo, los valores se almacenan como si se tratara de valores literales (de 16 caracteres).

Tipo contexto: Indican si el contexto impone alguna restricción al proceso de agregarle valores. El caso más general es que no haya tal restricción: se pueden agregar valores cuando alguna UBI necesite un valor que todavía no está registrado. Son fijos, por lo tanto están codificados en los programas, no almacenados en alguna estructura. Los tipos de contextos contemplados se listan en el Cuadro 19.

Cuadro 19 Los tipos de contextos.

Tipo	Descripción
1	NORMAL SIN RESTRICCIONES: al introducir una marca (contexto-valor) para una UBI y el valor no estaba entre los del contexto, se lo agrega a estos valores
2	TIPO CATÁLOGO: permite agregar valores sin instancias. Se crea un conjunto de valores y sólo se admiten marcas con uno de ellos. Es decir, hay una tabla de “valores válidos”.
3	VALORES RESTRINGIDOS A USUARIOS AUTORIZADOS: sólo usuarios que cuentan con el permiso para hacerlo pueden agregar valores nuevos.
4	LOS VALORES SE AGREGAN PENDIENTES DE AUTORIZACIÓN: cuando se agrega una UBI con un valor “Nuevo”, se acepta el valor pero está pendiente de autorización – si no se le otorga, se quitan las marcas del valor que se hubieran generado.
5	CONTEXTO TIENE UN SOLO VALOR (valor único): se trata de casos en los cuales sólo se marcan las UBIS con ese valor. Típicamente, es el caso de los campos binarios de la UBI: se marcan si el campo tiene el valor 1. En general se usan para indicar la presencia (o sólo la ausencia) de cierto dato de la UBI.
6	VALORES DEL CONTEXTO TIENEN 1 ÚNICA UBI: Es como si el contexto es un índice “único”, es decir, un par sólo una única UBI puede estar marcada con este contexto-valor sólo. Esto sucede por ejemplo cuando se trata de especificar para una UBI una relación con otro elemento de datos (interno o externo). Típicamente se trata de relacionar la UBI con algún documento físico (numerado) o con un elemento numérico de otro acervo.

El valor único: Si el contexto es de tipo No. 5, esto significa que tiene un solo valor. En ese caso el valor se especifica aquí. El motivo es poder rechazar marcas de ese contexto cuyo valor no sea el que aquí se anota. Observe que si un contexto es de tipo 5, se especifica este valor único como campo del contexto. No debe confundirse este aspecto del contexto con los del tipo 6, que indica que sus valores (varios) sólo admitan una UBI única como marca.

El siguiente campo se describe como parte de la alternativa de almacenar los valores en archivos planos. Como se verá, se descartó esta alternativa.

Cuantos valores tiene: Indica cuántas posiciones del arreglo de valores que se graba en disco están ocupadas con un valor; las demás posiciones se deben a que se reservó espacio adicional para que al agregar nuevos valores no haya que cambiar la lista de posición en el archivo.

Cuantos valores incluyendo UFUs (para uso futuro) tiene: Cuando se graban las listas de valores, se pueden agregar algunos (adicionales a los que usa para todos sus valores). De hecho este es el dato que se usa para recuperar y grabar las listas de valores como parte de todos los valores de todos los contextos en la estructura seleccionada (como se verá más adelante).

Tiene_restricciones_en_cuanto_a_BDD: La versión del DBB que se desarrolla es "multibase". Este concepto está explicado en la 0 dedicada a este tema. Se pueden crear hasta 9 bases de datos que contendrán las UBIs y sus marcas pero todas usan la misma infraestructura (contextos, clases, etc.). Los valores de este campo son:

- 0=no tiene y no se pueden agregar restricciones.
- 1 = no tiene pero *NO* se pueden agregar.
- 2=tiene o no, pero se pueden agregar.

Lo usa (o no) la base: Si indica que tiene restricciones ese contexto: Es decir, hay bases en las que no se ofrecerá, por ejemplo, porque no lo usa ninguna UBI de esa base. Esto puede suceder si las bases se refieren a diversos objetos. En una base se incluyen fotos, y se usa un contexto "cámara utilizada". En otra base del mismo SITE se trata de personas, de modo que ese contexto no será necesario. Se indican aquí esas 9 restricciones como 0/1 de una cadena de 8 caracteres (hay un máximo de 9 bases de datos posibles por SITE).

tiene_restricciones_en_cuanto_a_CLASES: Puede haber contextos que no se usan en una determinada clase. Al incluirlos aquí el sistema no los ofrecerá para UBIs de la clase; sin embargo se pueden agregar indicando el número de contexto cuando se desee incluir una marca. El objeto de especificar un contexto en este arreglo es reducir la lista de contexto que se presentan.

Una vez más, este campo de “tiene restricciones” usa los siguientes valores:

- 0=no tiene y no se pueden agregar restricciones.
- 1 = no tiene pero *NO* se pueden agregar.
- 2=tiene o no, pero se pueden agregar.

se_ofrece_en_la_clase: en una cadena de 50 caracteres, se indica Si-no (0/1) el contexto se ofrecerá para cada una de las (máximo 50) clases definidas.

Cómo se guardan los triplos de valores (marcas) del contexto: Este dato es técnico. Las estructuras para guardar los triplos (contexto, valor, UBI) para un contexto se describen a detalle en el capítulo dedicado a ese tema En el diseño para la versión del DBB que se describe se contemplaba almacenar valores como listas de valores, pero se adoptó el uso de una tabla de VALORES de una base de datos.

Observe que los campos que se refieren a la posición inicial y cuantos valores contiene son parámetros para la estructura de listas de valores y no aplican cuando se almacenan los valores en una tabla.

valor_no_marcable1, 2: Se pueden indicar, para un contexto, hasta 2 valores que no se marcan (es decir, se puede marcar la UBI pero esto no se refleja en los triplos para búsquedas).

5.4.2 Algunas observaciones sobre contextos

El contexto 0 (cero): El contexto 0 está reservado a un uso especial: se trata de marcas (Valor, UBI) que no tienen un contexto, es decir, no se especifica al significado del valor. Son las llamadas “palabras clave” usadas en varios tipos de publicaciones. Observe que estas marcas se incluyen en listas de marcas de los valores, es decir intervienen en las búsquedas (o subconjunteo, como se denomina esta actividad en el DBB).

Las marcas en las cuales el contexto indica la interpretación del valor en el DBB se llaman KBC (keywords by context).

El contexto 1 (LA CLASE de la UBI): Este contexto viene incluido con la versión del SITE usado. Puede ser que no se desee usar este contexto; en ese caso hay que indicarlo en la UBI (hay un campo en la forma de actualización de UBIs en el cual se puede indicar esta circunstancia).

Estructuras para almacenar los contextos

A continuación se presentan las estructuras analizadas para almacenar los contextos. Se descartaron otras porque no ofrecían ninguna ventaja. También se tomó en cuenta que la elección de una estructura para este tema (los contextos) no era crítica, es decir, no afectaría significativamente la eficiencia resultante de su uso. De hecho, se cargan los contextos a memoria, independientemente de la estructura en la cual están almacenados. En las instancias no muy frecuentes en las cuales se agrega un contexto, el proceso involucrado no tiene ningún impacto sobre el resto del sistema.

Alt 1. Una tabla de una BDD relacional; la tabla tendría todos los atributos del contexto.
Alt 2. Varias tablas relacionadas en las cuales se incluirían los diversos datos del contexto.

Alt 3. Un archivo con acceso “pseudo-aleatorio” en el que cada registro contuviera más de 1 variable de estructuras en las cuales se separarían los campos del contexto.
Alt 4. Un árbol, donde los nodos tendrían los campos del contexto (y no sólo el elemento de búsqueda).
Alt 5. Una lista ordenada de una estructura, en la cual estuvieran todas las propiedades del contexto.
Alt 6. una lista “apuntada” por otra.

5.4.3 Comparación de las estructuras contempladas

Alt 1 y Alt 2. Se usan tablas de una base de datos relacional. Se describen estas mediante sus campos.

El caso “normalizado”: se usan dos tablas adicionales:

- **Contexto-número de base de datos**: presencia indica que ese contexto se puede ofrecer en esa base de datos.
- **Contexto_clase**: ut supra para ofrecerlo en esa clase.

Si se desea utilizar solamente 1 tabla, se forman cadenas para almacenar estas restricciones (es decir, se usa una de las estructuras especiales descritas, los mapas de dígitos o caracteres).

Una cadena de 9 bytes (se podrían usar bits pero no vale la pena). Cada byte representa al número de DBB de ese número, y un valor 1 indica que sí se ofrecerá el contexto en esa DBB.

Otra cadena de 50 bytes, donde cada uno representa la clase del mismo número que la posición del byte.

Esta alternativa, la de una tabla con estos mapas de caracteres, es muy superior a la de las 3 tablas. Para cada contexto, hay un único registro (en toda la base de datos).

Si hubiera 25 clases definidas, y el promedio de clases en las que se ofrecería un contexto fuera 10, esto generaría 250 registros adicionales (en la otra tabla). Lo mismo sucede con el dato: contexto se ofrece en la DBB, aunque en general no habrá más de 2 o 3 bases de datos, de modo que no serían muchos registros y no tendría mayor impacto.

Alt 3. Un archivo con “acceso random simulado” o “pseudo-aleatorio”, en el cual cada registro contuviera más de 1 variable de estructuras en las cuales se separarían los campos del contexto. Como se ve, el acceso será binary puesto que no se pueden leer varias variables usando el número de registro en el acceso random.

Aquí el número de registro correspondería al número de contexto. Cada “registro” estaría formado por 3 variables con estructura (o arreglo): datos del contexto, restricciones de clases y restricciones de bases de datos). Para leer uno o los 3 arreglos, se calcula el desplazamiento relativo al principio del archivo con:

L = L1 + L2 + L3 (la longitud de cada uno de las variables).

Posición (1er byte) del 1 es L * (número de contexto – 1) + 1, Los otros datos se leen a continuación, sin especificar la posición, es decir, usa la posición siguiente al último byte de la parte ya leída.

Alt 4. Un árbol, donde los nodos tendrían los campos del contexto (y no sólo el elemento de búsqueda).

Comentario: en una versión anterior del DBB se utilizó esta estructura.

Se usa un árbol B (lo que anteriormente se llamaba B^+). El orden no tiene que ser muy grande, al haber un máximo de 255 nodos (de hecho, habrá aproximadamente la mitad si se balancea el árbol). De hecho, un árbol binario es casi equivalente en cuanto a eficiencia.

Esto facilita la inclusión de contextos, operación que por su frecuencia no tiene impacto alguno. En la mencionada versión el máximo de contextos era de 1000; ni en ese caso se justifica el uso de un árbol de este tipo.

Alt 5. Una lista ordenada de una estructura, en la cual estuvieran todos las propiedades del contexto (los arreglos se incorporarían como cadenas de dígitos, tipo mapa de dígitos).

Se especifica una estructura con todos los campos de un contexto. Se incluyen las cadenas de 9 bytes (las bases en las que se usa el contexto, y de 50 bytes, las clases en las que se ofrece. Sea TCONTEXTO esta estructura.

Se crea un arreglo de dimensión 255 (es decir, contextos del 0 al 255). Se graba y lee este arreglo.

Alt 6. Una lista “apuntada” por otra (como se verá, esta es la estructura que se adoptó).

La estructura utilizada es la misma que en la alternativa anterior. Sin embargo, se guardan en una lista sólo los que se han definido, es decir, no habrá posiciones del arreglo que no contengan datos (puesto que se agregan fuera de orden y sus números pueden no ser consecutivos.

Se crea (y graba en disco) un arreglo de bytes de dimensión 255. El dato ARR (4) indica la posición del arreglo de contextos en las cuales está el contexto número 4. Como esta es la alternativa seleccionada, se detalla en la sección siguiente.

5.4.4 Comparación de las alternativas para almacenar contextos

Se repiten las alternativas para comodidad de lectura**.**

Alt 1. Una tabla de una BDD relacional; la tabla tendría todos los atributos del contexto.

Alt 2. Varias tablas relacionadas en las cuales se incluirían los diversos datos del contexto.
Alt 3. Un archivo con acceso “pseudo-aleatorio” en el que cada registro contuviera más de 1 variable de estructuras en las cuales se separarían los campos del contexto.
Alt 4. Un árbol, donde los nodos tendrían los campos del contexto (y no sólo el elemento de búsqueda).
Alt 5. Una lista ordenada de una estructura, en la cual estuvieran todas las propiedades del contexto.
Alt 6. una lista “apuntada” por otra (como se verá, esta es la estructura que se adoptó).

Espacio en disco:

Las alternativas 3 (el archivo pseudo-aleatorio) y 6 (lista apuntada) son las que menos espacio usa. De hecho, se puede mostrar que el espacio es el mínimo + 256 bytes, puesto que cada atributo de cada contexto se guarda 1 sola vez, es decir, no hay repeticiones. La alternativa A5 tiene espacio no aprovechado cuando no están definidos los 255 contextos posibles.

Si se usara una tabla única de una base de datos (Alt 1), el índice duplicaría el espacio necesario para almacenar el campo “número de contexto”. Esto no es significativo. Si se usan 3 tablas (Alt 2) hay más espacio no sustantivo (es decir, que no almacenan los datos vitales).

En el árbol B, hay espacios ocupados por los apuntadores a nodos; además, por la estructura misma del árbol B, ciertos contextos (los que están en los nodos) también están en las hojas. Por último, habrá nodos con datos no aprovechados (cuando no están “llenos”). El uso de un árbol binario elimina la duplicidad de contextos en el árbol.

Acceso (usos de los contextos en las rutinas del programa)

El contexto está involucrado en prácticamente todas las operaciones que se realizan para actualizar y usar las marcas (que es el objetivo principal del DBB). De este

modo, conviene tener los contextos en memoria, en lugar de leerlos de disco cada vez que se necesitan.

Si se usara tablas de una base de datos, esto se haría mediante tablas de datos (DataTable) o en versiones anteriores, Recordset. Esto implica que cuando se quiere usar un contexto determinado, hay que hacer una búsqueda recorriendo las hileras de la tabla.

Si los contextos se almacenan en memoria como un arreglo, se pueden encontrar vía su número de contexto sin buscar (recorrer otros). Pero se tendrían que incluir los contextos no utilizados intermedios. En la alternativa 6 esto se hace sin desperdicio de memoria, puesto que sólo se incluyen los contextos utilizados. La diferencia en una búsqueda (comparada con el acceso individual) puede ser muy pequeña, pero al hacer esta operación muchas veces, se acumula proceso superfluo.

Ofrecer los contextos para que seleccionen uno: aquí no hay diferencia puesto que se usa la tabla de datos o el arreglo para armar un objeto gráfico.

5.4.5 La estructura seleccionada

Se almacenan los contextos usando 2 listas (arreglos), es decir, la Alternativa 6. Arr_apuntadores_a_contextos (255). Es un arreglo de 256 elementos (del 0 al 255). Cada posición corresponde al contexto de ese número. El valor es la posición en la lista de contextos en la cual están los datos del contexto.

Arr_contextos (dimensión variable). Es un arreglo de contextos; sus elementos son instancias de la estructura de los contextos. Se agregan éstos a medida que se definen, pero no se ordena el arreglo. Para encontrar un contexto en el arreglo, se usa el arreglo de apuntadores.

Contexto buscado NC = arr_contextos (arr_apuntadores_a_contexos (NC))

5.4.6 Observaciones sobre las estructuras seleccionadas

1. Casi todas las funciones relacionadas con las UBIs mismas usan marcas, de modo que necesitan los contextos ofrecidos. Esto implica tener las definiciones en forma rápidamente accesibles, preferiblemente en una estructura en memoria. Este debe ser el criterio prevalente en la selección de las estructuras para almacenar y usar los contextos.
2. Las estructuras seleccionadas deben permitir adiciones (es decir, contextos agregados en cualquier momento). De hecho en cualquiera de las alternativas es posible implementar esta función.
3. No son importantes las consideraciones de espacio en disco o memoria, puesto que al haber un máximo de 255, el tamaño no tendrá impacto sobre el rendimiento del sistema.
4. Como se ha anunciado, nunca se contempla el aspecto de la complejidad de la programación, que es considerablemente mayor cuando se usan archivos en disco, y aumenta si se emplean estructuras apuntadas o que ocupan espacios variables en disco.

5.5 Valores de contextos

Ya se ha dicho que un contexto puede ofrecer varios valores para su uso en marcas relativas a ese contexto. Es importante señalar que el dato “VALOR” carece de significado excepto utilizado para UN CONTEXTO en particular. EL mismo VALOR puede estar en diversos Contextos, como ya se ha explicado anteriormente.

Los valores se usan con un máximo de 16 caracteres; se contempla que para algunos contextos sus valores sean numéricos (números enteros o decimales); esto sólo tendrá impacto sobre la inclusión de nuevos valores – se rechazan los que no sean numéricos. Sin embargo se guardan los valores de este como cadenas de caracteres de 16 bytes.

Para determinar las estructuras para guardar los valores válidos, y posteriormente las marcas (triplos) que contienen el par correspondiente, se tomaron en cuenta:

La eficiencia de ofrecer los valores al usuario que desea.

- Marcar una UBI: se selecciona un contexto, y se puede invocar una lista de valores válidos para ese contexto para elegir uno de ellos;
- Usar un (contexto, valor) en una búsqueda: *ut supra.*
- El número de valores de todos los contextos, que puede ser muy numeroso.

La necesidad o conveniencia de aprovechar las estructuras utilizadas para almacenar los valores para dirigir al programa a las marcas que contienen dicho valor. En versiones anteriores del DBB, se utilizaron árboles para almacenar tanto los contextos, sus valores y las marcas correspondientes.

Para no repetir datos en los diversos nodos, se usó una estructura de “selva” (árboles de árboles). Se creó un árbol B de contextos. Cada nodo representaba un contexto, que a su vez apuntaba a otro árbol B que contenía los valores de ese contexto. Observe que esto resultó en un árbol por contexto.

Una vez más, en los nodos de este árbol de los valores del contexto, se apuntaba al conjunto de marcas (triplos) de dicho valor. Las estructuras utilizadas para almacenar y usar estas marcas se discuten a detalle en el capítulo dedicado precisamente a ese tema. En una versión se utilizó otro árbol B para dichas marcas. En otra se almacenaban las marcas como arreglos (listas); finalmente, en otra versión se usaban BITMAPS para los valores que tenían un porcentaje significativo de marcas de las UBIs con ese valor, mientras para los demás valores se usaron listas en disco.

Al contemplar la elaboración del nuevo DBB objeto del proyecto actual, se decidió revisar y si fuera posible, mejorar las estructuras en cuanto a rendimiento. En el Cuadro 20 enumeran los atributos que definen a un valor de un contexto.

Cuadro 20 Los atributos para describir un valor de un valor de un contexto.

No.		NOMBRE (O SIGNIFICADO)	TIPO DE DATO
1		**DATOS GENERALES**	
1.1		Contexto (implícito o explícito, según la estructura utilizada)	Byte
1.2		Valor	X(16)
1.3	**?**	Pendiente de confirmar	Byte
1.4	**?**	UBI-única	ENT 4
1.5	**?**	Estructura_para_sus_marcas	X(24)
2		**DONDE ESTÁN SUS MARCAS (UBIS)**	
2.1	**?**	Arch_marcas	0/1
2.2	**?**	Cuantas_marcas_usa	Ent 4
2.3	**?**	Lista_marcas_inicia_en	Ent 4
2.4	**?**	Cuantos_espacios_incluyendo_ufus	Ent 4
2.5		Menor_ubi_marcas	Ent 4
2.6		Mayor_ubi_marcas	Ent 4

Pendiente de confirmar: Un VALOR puede estar sujeto a aprobación; alguien marca una UBI con ese VALOR (que todavía no está en el contexto): Si el contexto permite este tipo de adición condicional, se incluye la UBI como marca de este valor (que se agrega a los valores del contexto). Sin embargo, al encargado de hacerlo le aparecerá un pendiente: confirmar el valor XXX del contexto NN. El valor del campo será 1.

Si lo confirma, se elimina el pendiente (Valor 0 a este campo). Si lo rechaza, se quita la marca y se asigna el valor 2 al campo. Es responsabilidad del usuario que introdujo la marca averiguar que el valor fue declarado inválido (inaceptable).

UBI única: Hay contextos para los cuales sus valores no aceptan más de 1 marca (son como “índices únicos”). En ese caso, no se usa una lista de valores, sino se indica el número de UBI en este campo. Un ejemplo es si se incluye el número del empleado (que es el número de la UBI) pero en otra base. Cabe señalar que sólo se contemplan para este tipo de situación valores numéricos.

Si este campo no vale 0, los campos del número de marcas y la posición de la lista de éstas serán 0.

Estructura_para_sus_marcas: A pesar de que en general la estructura de las marcas está dada por el contexto, se contempla la posibilidad de que para determinados valores del contexto se pudiera usar una estructura diferente.

Arch_marcas: Cuando un acervo de datos es muy numeroso, el número de marcas puede ser elevado. Para no tener archivos de marcas demasiado grandes, se contempla la posibilidad de usar varios archivos. En ese caso, este campo permite decidir en cuál de ellos está sus marcas.

Lista_marcas_inicia_en: Es la posición del arreglo de marcas (el archivo de marcas) en la cual está la primera marca de este contexto-valor.

Cuantos_espacios_incluyendo_ufus: Cuando se graba una lista de marcas en disco, que no ocupe el mismo lugar que la versión anterior (antes de agregarle marcas a la lista) se incluyen algunos espacios para “uso futuro”. Esto permite agregar algunos valores en el mismo lugar del archivo.

5.5.1 Estructuras contempladas para los valores

Es importante mencionar que además de los valores de los contextos, se contemplaron las estructuras que almacenarían las marcas (triplos) de cada valor. Este tema está descrito a detalle en el 0 sobre las marcas, pero es fundamental su inclusión parcial en la selección de las estructuras para guardar los valores.

Las estructuras que se formularon para almacenar los valores de contextos:

Alt 1. Una tabla de una BDD relacional que tendría todos los campos de los valores incluyendo apuntadores a sus marcas.
Alt 2. una tabla con los atributos del valor, y otra con sus marcas (la solución llamada “evidente”).
Alt 3. Una tabla de una BDD relacional que tendría todos los campos de los valores y arreglos indicando las UBIs que tienen esa marca.

Alt 4. Un árbol, donde los nodos tendrían los campos del valor (y no sólo el elemento de búsqueda).

Alt 5. Una lista ordenada de una estructura, en la cual estuvieran todas las propiedades de los valores.

Alt 6. Una lista ordenada de instancias de la estructura grabada en disco con los campos que describen a un valor, y además apuntaría a la lista de sus marcas

En lugar de describir a detalle cada una de estas estructuras, se optó por eliminar algunas de ellas por los motivos que se exponen. Para ello, se tomaron en cuenta los resultados de las simulaciones. En este caso, estos dos resultados:

- Si se graba un arreglo de estructuras en disco, si éstas incluyen algún campo de texto para recuperarlas, en lugar de hacerlo como arreglo, se leen una por una. Además, esto resulta en un desperdicio de espacio puesto que habrá que grabar los datos con su longitud máxima para que todos los elementos del arreglo tengan la misma longitud.
- Esto resultó en la eliminación de las alternativas Alt 5 y Alt 6.
- El uso de un árbol (o de varios, uno para cada valor) resulta en mayor espacio, puesto que hay que incluir los nodos. De hecho, el árbol se construiría ocupando posiciones de una lista (arreglo) que se grabaría como tal en el archivo. Esto hace que la recuperación sea la misma que si se tratara de una lista de UBIs del valor, sólo que complicada por la organización como árbol (apuntada de un nodo a otro) en lugar de una lista de números ordenada. Estos considerandos hicieron que se descartar la alternativa Alt 4.

5.5.2 Descripción de las estructuras contempladas

Alt 1. Una tabla de una DBB relacional que tendría todos los campos de los valores incluyendo apuntadores a sus marcas. El índice principal sería contexto-valor puesto que como ya se dijo, un mismo valor puede estar en varios contextos.

Esta estructura ofrece la facilidad de poder recuperar un valor determinado en forma eficiente, puesto que usa el índice compuesto precisamente por el contexto-valor. Sin embargo, en muchas funciones se necesitan todos los valores de un contexto (por ejemplo para ofrecerlos). Por otra parte, al leer los valores en forma individual

directamente de la base de datos, se generan muchas “lecturas” (ejecuciones de SQL) lo que aumenta el número de actividades de entrada-salida de datos y en cierto grado, el proceso.

Se almacenan todos los valores en memoria. Esto se haría por contexto, y no todos ellos. En la tabla en memoria (DataTable) se usaría una búsqueda binaria para encontrar un determinado valor (se almacenan los valores en orden alfabético).

Alt 2. Una tabla con los atributos del valor, y otra con sus marcas (la solución llamada “evidente”.

Esta es la alternativa denominada evidente, que fue descrita anteriormente. A pesar de ser la única solución que almacena una sola vez las marcas (las de la UBI y las del par contexto-valor), el número de registros y en especial, el desperdicio muy importante de espacio en disco son desventajas insuperables. Aquí se duplican los 3 datos (contexto, valor y número de UBI) de la segunda tabla, en el índice primario, y una vez más el número de UBI en el índice secundario, la UBI, que es necesario para encontrar las marcas de una UBI dada.

Alt 3. Una tabla de una DBB relacional que tendría todos los campos de los valores y arreglos indicando las UBIs que tienen esa marca.

Se le agregan campos “memo” (de longitud variable y prácticamente sin límite de tamaño). En estos campos se concatenan los números de UBI de las marcas del valor. Esto se haría convirtiendo los números de UBI (que son enteros) a su valor como dígitos, y adoptando una longitud fija o separando los número mediante algún carácter, por ejemplo, un punto y coma.

Si no fuera que puede haber valores con un número “enorme” de marcas, digamos un centenar de millones de ellas en un acervo numeroso, esta sería una buena alternativa porque evitaría el uso de apuntadores o de una estructura adicional para guardar las marcas.

Se podrían usar varios registros para valores que tuvieran demasiadas UBIs, pero de todos modos el proceso para transformar estas cadenas en arreglos de números de UBI, operación constante en DBB, sería muy significativo.

La primera implementación del DBB (Cruz Millan, 2003) usó un esquema de este tipo. Los registros de la tabla estaban organizados por valor (se usaba una tabla por contexto). Había dos casos:

- Cuando el valor no tenía muchas instancias, se indicaban las UBIs en columnas de la base (campos). Si no cabían (puesto que había 16 columnas) se agregaba otro registro para el mismo valor.
- Cuando el número de instancias era mayor que 50, se usaban los campos como apuntadores a listas que se almacenaban en disco. Estas listas tenían un tamaño fijo (100 UBIs). 8 columnas indicaban el último elemento (número de UBI) de una lista, las otras 8 (en forma paralela) indicaban la posición de la lista en el archivo.

5.5.3 La alternativa seleccionada

El tema de cómo se almacenan las marcas (los triplos) se discute en el capítulo sobre ese tema. Al eliminar la alternativa de almacenar los valores como un arreglo de ellos grabado en un disco plano por los motivos expuestos anteriormente, se adoptó la alternativa **Alt 1**:

Se usa una tabla de la base de datos para almacenar los valores de los contextos; en cada valor, se indica el tamaño y la posición del arreglo de sus marcas, grabada como tal en disco.

CAPÍTULO 6. UBIS

6.1 Introducción

Como ya se ha dicho, el DBB es un conjunto de elementos de datos llamados *UBI (*Unidad Básica de Información) que son los equivalentes a un registro de una tabla de una base de datos.

Cada UBI pertenece a una CLASE; en cierto modo es equivalente a una tabla de una base de datos, pero solamente en el aspecto de los campos que tiene, puesto que todas las UBIS se almacenan – como se verá – en una misma tabla de la base de datos relacional seleccionada para este efecto. Una UBI es una instancia de una clase, es decir, tiene los atributos derivados de la clase. En el CAPÍTULO 4 dedicado a las clases se describen a detalle estos conceptos.

El tema de la investigación relacionado con estos elementos consiste en seleccionar una estructura conveniente para almacenarlas. Para ello, además de los elementos de datos que las componen, se consideran las funciones que se realizan con estos datos:

- El acceso a una UBI debe ser lo más rápido posible. Lo mismo sucede con un conjunto de ellas.
- Se deben poder agregar “marcas” a cada UBI: este concepto se ha explicado anteriormente, pero es objeto de un capítulo especial dedicado a las estructuras contempladas para guardar estas marcas; la estructura seleccionada para almacenarlas es el tema del 0 sobre MARCAS, aunque en este se introduce el tema.
- Poder interpretar algunos campos de las UBIs según como fueron definidos por la clase a la que pertenece.
- Facilitar la interpretación de subconjuntos de UBIs obtenidos como resultado de consultas usando las marcas.

OBSERVACIÓN: en este capítulo no se tomaron en cuenta aspectos del almacenamiento de marcas.

6.1.1 MODELO DE DATOS DE LAS UBIs

El proceso comienza por definir los campos de una UBI, que se muestran en el Cuadro 21.

Cuadro 21 Campos de una UBI.

DATOS GENERALES	variables	anexo	marcas	Técnicos
1.1 UBI (Ent4)	2.1 Fecha (Fecha/hora)	3.1 Tipo de anexó(Byte)	4.1 Sus marcas	5.1 Audit1 (Ent4)
1.2 Clase(Byte)	2.2 Campos binario(Txt)s	3.2 Path del archivo(Byte)	4.2 Cuantas marcas tiene(Ent4)	5.2 Audit2(Ent4)
1.3 Tipo de UBI(Byte)	2.3 campo número(Ent4)	3.3 Nombre del archivo		
1.4 Contenido(Txt)	2.4 Campos literales(4)(Txt)	3.4 Extensión(Txt)		
1.5 Usuario que lo creo(Ent4)				
1.6 Status(Byte)				
1.7 Confidencialidad (Byte)				

Qué significan los campos de una clase de UBIs

En el Cuadro 22 Sólo se explican aquéllos para los cuales el nombre no indica su significado.

Cuadro 22 Significado de los campos de una UBI.

1.2	Clase	Si es 0, se usa default y no se
1.3	Tipo de UBI	catalogó tipo de UBIs está indizado y marcado si no vale 0
1.4	Contenido	puede ser el ANEXO o cualquier texto útil
1.6	Status	0=incompleto 1=completo 2=activo 3=a eliminar 4=eliminado
1.7	confidencialidad	0=ninguno, 1= pueden ver UBI pero no anexo, 2=pueden ver ambas pero requiere permito del USER creador 3=pueden ver ambos
2.1	Fecha	Este campo es de uso variable (según la clase)
2.2	Campos binarios	4 dígitos (0 y 1) de izquierda a derecha ejemplo 1011 el apagado es el binario 2
2.4	Campos literales ‘son 4 campos iguales	
3.2	Path del archivo	Path del archivo (del catálogo de Paths)
4.1	Sus marcas	Un número variable de pares (contexto-valor)
5.1 5.2	Audit1, audit2	Permite proteger los datos contra modificaciones no autorizadas

6.1.2 ALTERNATIVAS CONSIDERADAS PARA ALMACENAR LAS UBIs

Alt 1. Base de datos relacional con 1 tabla de UBIs; sus marcas se almacenan en estructuras independientes.

Alt 2. Base de datos Tabla UBIS; las marcas de la UBI se guardan en un campo “memo” de la misma tabla (como veremos, ésta es la alternativa seleccionada).

Alt 3. Archivo Random de UBIS

Alt 4. Arreglo de UBIS guardado en disco, pero apuntados por número de UBI (para contemplar números no utilizados)

6.1.3 Método:

Para comparar ciertas estructuras, se definieron y ejecutaron ciertos procesos que permitirían determinar duraciones. En base los resultados obtenidos, se determinó:

- El uso de un archivo RANDOM para las UBIs se descartó por dos motivos:

- Al haber variables de cadena, éstos se tendrían que guardar con longitud fija. Esto resulta en un uso mucho mayor de espacio que si los campos fueran de una tabla de una base de datos.
- Si los números asignados a las UBIs no fueran consecutivos, habría un desperdicio de espacio debido a los registros “intermedios” del archivo random.

Para obviar esta última situación, se podrían guardar las UBIs como registros de un archivo binario, pero con un esquema de apuntadores a la posición de cada una.

Los resultados de las simulaciones indicaron que el uso de un archivo binario para almacenar las UBIs no ofrecía ventajas en cuanto al uso principal: interpretar los resultados de una consulta.

Esto resultó en que sólo se compararan las alternativas 1 y 2; se descartaron las demás. La diferencia estriba en que:

Alt 1. Base de datos relacional con 1 tabla de UBIs; sus marcas se almacenan en estructuras independientes.

Alt 2. Base de datos Tabla UBIS; las marcas de la UBI se guardan en un campo “memo” de la misma tabla (como veremos, ésta es la alternativa seleccionada).

Alt 1 Base de datos relacional con 1 tabla de UBIs; sus marcas se almacenan en estructuras independientes.

Se genera una tabla con todos los campos necesarios para almacenar los datos de las UBIs. Las marcas de las UBIs se almacenan en otra tabla como se muestra en la Figura 35.

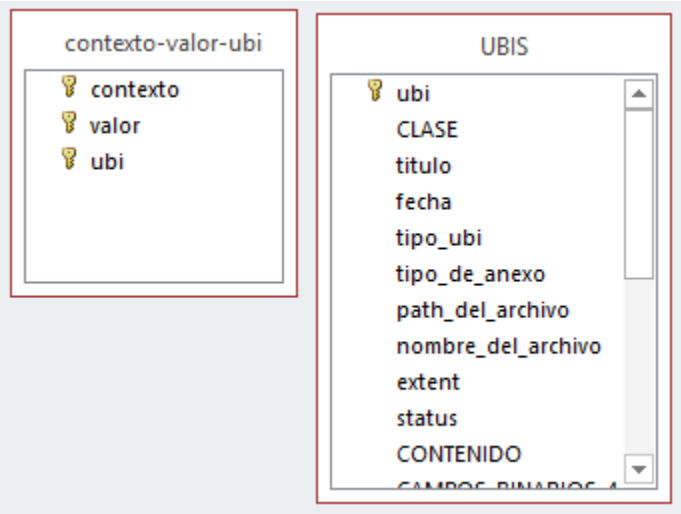


Figura 35 Alternativa 1, una tabla de UBIs y sus marcas se almacenan en estructuras independientes, camptura Access.

De esta manera el gestor de la base de datos de encarga de gestionar todo el manejo del contenido de estas tablas. Ambas tablas comparten el número de UBI y de esta manera se pueden recuperar las marcas de cada una de ellas.

Alt 2 Base de datos Tabla UBIS; las marcas de la UBI se guardan en un campo “memo” de la misma tabla

Se genera una tabla con los campos necesarios para guardar la UBIs pero se le agrega un campo “SUS MARCAS” de tipo memo y en este se almacena todas sus marcas concatenando CONTEXTO+VALOR+”;” y si de esta se puede almacenar una cadena de hasta 65535 caracteres.

6.1.4 Comparación de las estructuras

Los criterios usados para determinar cuál será la seleccionada incluyen:

- Espacio total en disco.
- Velocidad de recuperación de subconjuntos.

- Interpretación de los resultados de búsquedas (que son listas de número de UBI).
- Con estos criterios, se eliminaron algunas de las alternativas por presentar situaciones que indicaban que serían desfavorables en cuanto a uno de estos criterios.

A1. Base de datos relacional con 1 tabla de UBIs y las marcas en una tabla de triplos.

La tabla con los campos Contexto, Valor, UBI tendrían dos índices: el principal constituido por los 3 campos, y otro por el campo UBI para poder recuperar las marcas de una UBI.

El espacio en disco para almacenar esta segunda tabla es tanto mayor que el de otras alternativas que se decidió no usar esta estructura.

Velocidad de recuperación de subconjuntos: tampoco hay diferencias significativas en este aspecto. Se hicieron simulaciones para obtener conjuntos de UBIs de la tabla única y de las almacenadas como arreglo: a pesar de la diferencia muy significativa en cuanto a número de accesos a disco, no hubo diferencias en los tiempos de ejecución.

Interpretación de los resultados de búsquedas (que son listas de número de UBI).

También se hicieron simulaciones para comparar los procesos necesarios para interpretar las UBIs del subconjunto obtenido de una consulta. El resultado: no hubo muchas diferencias, fue sorprendente: Se ejecutaron procesos - descritos en el 0 dedicado a este tema - con diversos algoritmos en cuanto a la recuperación de las UBIs de la tabla de la base de datos y una vez más, no arrojaron diferencias comparadas con los tiempos de ejecución del proceso de interpretar las UBIs del subconjunto utilizando el archivo en modo binario.

6.1.5 Decisión: la estructura seleccionada para esta versión del DBB

La opción implementada para almacenar las UBIs es la **A2 Base de datos Tabla UBIS, las marcas se guardan en campos memos.**

Con esta opción es posible almacenar y recuperar cada una de las UBIs de una manera eficiente, así como sus marcas correspondientes, aunque aún falta una forma de recuperar las UBIs que satisfacen una búsqueda por **CONTEXTO-VALOR;** una vez más se posterga este punto al capítulo siguiente, dedicado a las MARCAS.

CAPÍTULO 7. MARCAS DE LAS UBIS

7.1 Introducción

El concepto fundamental del DBB es que se puedan agregar descriptores a las UBIs (además de los campos de la tabla) por medio de pares CONTEXTO-VALOR, donde el contexto indica la interpretación que se le da al valor. Llamaremos **marcas de la UBI** a estos pares.

Es importante señalar que la formulación y comparación las estructuras en las cuales se almacenarían las marcas de la UBI se basó en el modelo seleccionado para las UBIs: se guardan en una tabla de una base de datos.

Esto implica que sólo se comparan dos casos:

Las marcas de la UBI se incluyen en esa tabla. Esto hace que se tengan que usar otras estructuras para guardar los “triplos” (contexto, valor, UBI) para permitir armar los subconjuntos correspondientes a un par (contexto, valor).

Las marcas de las UBIs se almacenan en otras estructuras. Aquí se presentan dos opciones fundamentales:

- Usar una estructura para las marcas de la UBIs, y otra para los triplos.
- Usar una estructura única para ambos usos.

7.2 Modelo de datos de las marcas

Cada marca se agrega a un ARREGLO; cada elemento contiene ambos valores (contexto-valor). Los contextos deben estar incluidos en un catálogo de contextos; cada contexto tiene asignado un número, además de su descripción (el significado) y otros datos que se detallan en la sección correspondiente. Los valores se almacenan en **MAYÚSCULAS y SIN ACENTOS.**

El programa que permite actualizar las marcas de una UBI ofrece las funciones para agregar o eliminar un par (contexto, valor) del conjunto de sus marcas, de ese modo, arma un arreglo de los pares manteniendo ordenado este conjunto por contexto-valor. Lo que nos concierne en este capítulo es cómo se almacenan en disco estos conjuntos de marcas para cada UBIs.

7.3 Alternativas consideradas para almacenar las marcas

Observación importante: De ahora en adelante el vocablo MARCA indica un **triplo (CONTEXTO, VALOR, UBI)** donde este último dato es el número de la UBI en cuestión. De este modo, se tienen que almacenar estos triplos con las siguientes características:

- Se deben poder recuperar las marcas de una UBI en forma eficiente.
- El uso de las marcas implica encontrar las de un par (Contexto, Valor). Como este es la función fundamental del DBB es armar subconjuntos de UBIs que estén “marcados” con un determinado par, el diseño debe contemplar prioritariamente esta función.

Por lo tanto reiteramos el propósito de este Capítulo: encontrar las estructuras que ofrezcan **la mayor eficiencia en ambas funciones,** pero con énfasis en la segunda; y contemplar diversos aspectos computacionales que incluyen la rapidez con las que se consiguen y almacenan los datos, y el uso de espacio en disco que implica cada estructura contemplada.

La estructura “evidente”

Se crea una tabla de una base de datos relacional con los campos UBI, Contexto, Valor. El índice (único y principal) está formado por los 3 campos en el orden enumerado. Se incluye otro índice (contexto, valor) – que no será único - para las

“búsquedas”. Se contempla también la alternativa de usar los campos (Contexto, valor, UBI) como índice principal y agregar otro con el campo UBI.

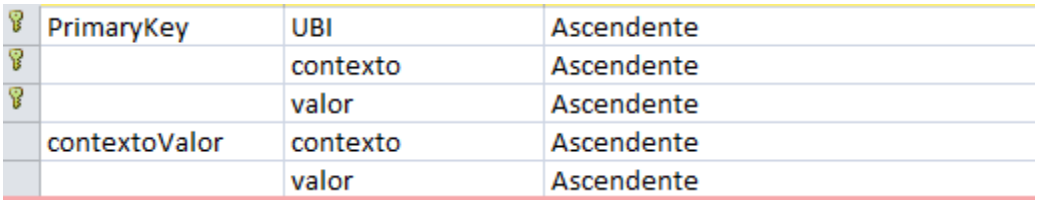
Este modelo permite, en forma eficiente:

- Recuperar las marcas de una UBI.
- Recuperar las marcas correspondientes a un par (Contexto, valor).
- Esta es la estructura “evidente”. Se crea una tabla con los campos UBI, Contexto, Valor como se muestra en la Figura 36.

Nombre del campo	Tipo de datos
UBI	Número
contexto	Número
valor	Texto corto

Figura 36 Campos de la estructura evidente, solo se tienes estos 3 campos y todos son llave, camptura Access.

- El índice (único y principal) está formado por los 3 campos en el orden enumerado. Se incluye otro índice (contexto, valor) – que no será único - para las “búsquedas”, como se indica en la Figura 37.



PrimaryKey	UBI	Ascendente
	contexto	Ascendente
	valor	Ascendente
contextoValor	contexto	Ascendente
	valor	Ascendente

Figura 37 Indices de la alternativa evidene MARCAS se le agrega un idice extra para la busqueda, captura Access.

Alternativas en cuanto a índices:

a) Principal: UBI Secundario: Contexto, valor, UBI.
b) Principal: Contexto, valor, UBI secundario: UBI.

En la alternativa b) cuando se obtengan las marcas de una UBI, éstas no estan ordenadas, pero se suple esta circunstancia con una cláusula “order by” que no afecta el desempeño por lo reducido del conjunto a ordenar. Por la frecuencia de uso de los datos para consultas, conviene utilizar el triplo mismo como índice principal.

Ésta es la única estructura que contempla simultáneamente ambos usos de las marcas: encontrar y actualizar las marcas de una UBI y armar subconjuntos de UBIs que están marcadas con un cierto par (contexto, valor).

La adopción de este modelo resuelve todos los aspectos de desempeño excepto el espacio en disco que necesita, que se aumentado por dos circunstancias:

- Los triplos se guardan con los 3 datos.
- La necesidad de utilizar índices aumenta el espacio en disco.

Si hay un gran número de marcas, este factor puede ser de gran importancia. En ese caso también se agregan las consecuencias de usar una tabla de una base de datos con *muchísimos* registros, donde este adjetivo se interpreta como que se alcance o desborde la capacidad ofrecida en ese aspecto por el manejador de base de datos utilizado.

En el resto de este capítulo se formulan y comparan alternativas a este modelo, con el objetivo de disminuir los requisitos en materia de espacio en disco. Cabe señalar que para todos ellos, las marcas se almacenan en dos versiones: ordenadas para su uso como pares agregados a las UBIs, y las UBIs marcadas con un determinado para (contexto, valor).

De este modo, se analizan primero las Estructuras para almacenar los pares (contexto, valor) agregados a cada UBI, y por separado las que se contemplaron

para almacenar las “listas de resultados”, como se denomina en DBB un conjunto de marcas de un par (contexto, valor).

Comentario: a pesar de que en el pasado se usaron árboles B para almacenar marcas, el uso de este tipo de estructura no ofrece grandes ventajas, pero sí un número de desventajas importantes, como lo son:

- Los apuntadores ocupan lugar
- La elaboración de una lista a partir del árbol es costosa en tiempo de proceso (aunque en los árboles B se realiza con una búsqueda de un nodo y a partir de ese nodo, los demás se encuentran sin necesidad de recorrer el árbol).

7.4 Estructuras para almacenar los pares (contexto, valor) agregados a cada UBI

Alt 1. Guardar las listas de las marcas de una UBI en un archivo en disco (o en uno de varios archivos). Como campo de la UBI se almacena la información que permite recuperar y cambiar esta lista como se muestra en la Figura 38.

Nombre del campo	Tipo de datos
UBI	Número
numeroDeArchivo	Número
tamañoDeLista	Número
numeroDeMarcas	Número
posicionEnElArchivo	Número

Figura 38 Campos de la Alt 1 estos deben de contener la información necesaria para recuperar las marcas contenidas en un archivo plano, captura Access.

En una base de datos se almacenan los datos necesarios para recuperar la lista de marcas, como son: número de archivo, tamaño de la lista, número de marcas y posición en el archivo. Observe que el tamaño de la lista puede contemplar algunas posiciones adicionales (además de las que ya se usan) como se muestra en la Figura 39 para poder agregar marcas a la UBI sin operaciones de “reubicación”.

Figura 39 Cuando seguarda una lista de resultados es apropiado guardar espacios estras para poder guardar nuevos elementos nuevos, elaboración propia.

En el archivo en disco se almacena un arreglo de pares, usando una estructura del tipo:

```
Public Structure marcas
      Public numContexto as byte
      <VBFixedString (16)> Public valor As String
End Structure
```

De esta forma es posible recuperar cada uno de los registros accediendo al archivo de forma binaria. Un ejemplo que pudiera aclarar este modelo. Supongamos que a la UBI número 17 se le indican las siguientes marcas –pares contexto y valor. Para entender el ejemplo no es necesario saber el significado de los contextos (sólo indicaremos el número).

(1, “AZUL”), (15, “PEREZ”), (15, “SERGIO”), (24, “LIBRO”), (32, “1956”).

Hay un archivo en disco en el que se almacenan estas listas. En el caso que nos ocupa: Se determina la posición (el byte) del archivo en el cual comienza esta lista (es decir, las marcas de la UBI en cuestión. Es importante que “quepa” la lista, es decir, que al grabarla en disco no altere alguna otra lista del mismo tipo guardada en el archivo.

Se graba el arreglo de marcas. Ocupa 5 * 17 bytes. Digamos que en lugar de apartar (y usar) 5 de estos espacios, se decide ampliarlo a 7 espacios por si alguien agrega otras marcas a esta UBI.

Para recuperar estas marcas, será necesario conocer:

- Donde está la lista de las marcas (posición donde comienza la lista).

- El número de elementos de la lista que se grabó, que puede incluir los espacios “para uso futuro”. Además, hay que conocer el número espacios de la lista que se usaron (es decir, que contienen una marca). La alternativa es determinar cuáles espacios se usaron preguntando si el contexto grabado no es nulo, cosa que tendrán los espacios no utilizados.

Alt 2. Crear una tabla para las Marcas con índice UBI + un consecutivo. Se definen N campos (N es parámetro del diseño) para los contextos otros N para el valor correspondiente a cada contexto (se interpretan como arreglos paralelos). Se crean registros adicionales (vía el consecutivo) para las marcas necesitan más de N posiciones de estos arreglos de campos.

Crear una tabla para las Marcas con índice UBI + un consecutivo. Se definen N campos (N es parámetro del diseño) para los contextos, y otros N para el valor correspondiente a cada contexto, como se muestra en la Figura 40.

	Nombre del campo	Tipo de datos
🔑	UBI	Número
🔑	consecutivo	Número
	contexto1	Número
	valor1	Texto corto
	contexto2	Número
	valor2	Texto corto
	contexto3	Número
	valor3	Texto corto
	contexto4	Número
	valor4	Texto corto

Figura 40 Campos de la tabla MARCAS de la opción Alt 2 una tabla para las Marcas con índice UBI + un consecutivo, captura Access.

Para este ejemplo de definen un máximo de 4 marcas por registro, pero en general se usaría otro número de campos. Las marcas se guardarían como se ve en la Figura 41.

UBI	consecutivo	contexto1	valor1	contexto2	valor2	contexto3	valor3	contexto4	valor4
12	1	23	ROJO	24	AUTO	54	EDIFICIO	123	MEXICO
34	1	34	AZUL	43	ENERO	54	MEDICO	56	23434

Figura 41 Ejemplo de marcas almacenadas con la alternativa A2, captura Access.

Si se excede el número de marcas a guardar se crean registros adicionales (vía el consecutivo) quedando el registro de la siguiente manera.

UBI	consecutivo	contexto1	valor1	contexto2	valor2	contexto3	valor3	contexto4	valor4
12	1	23	ROJO	24	AUTO	54	EDIFICIO	65	PABLO
12	2	65	PEDRO	123	MEXICO	0		0	
34	1	34	AZUL	43	ENERO	54	MEDICO	56	23434

Figura 42 Asignación de más de 4 marcas, captura Access.

Alt 3 Crear una tabla para las Marcas con índice UBI + contexto + consecutivo. Se definen N campos (N es parámetro del diseño) que contendrán valores de ese contexto incluidos en las marcas de la UBI como se muestra en la Figura 43.

Nombre del campo	Tipo de datos
UBI	Número
contexto	Número
consecutivo	Número
valor1	Texto corto
valor2	Texto corto
valor3	Texto corto
valor4	Texto corto

Figura 43 Campos de la opción Alt 3 una tabla para las Marcas con índice UBI + contexto + consecutivo, captura Access.

De manera que las marcas se guardaran de la siguiente manera, como se ve en la Figura 44.

UBI	contexto	consecutivo	valor1	valor2	valor3	valor4
32	7	1	AZUL	ROJO	PUERTA	VERDE

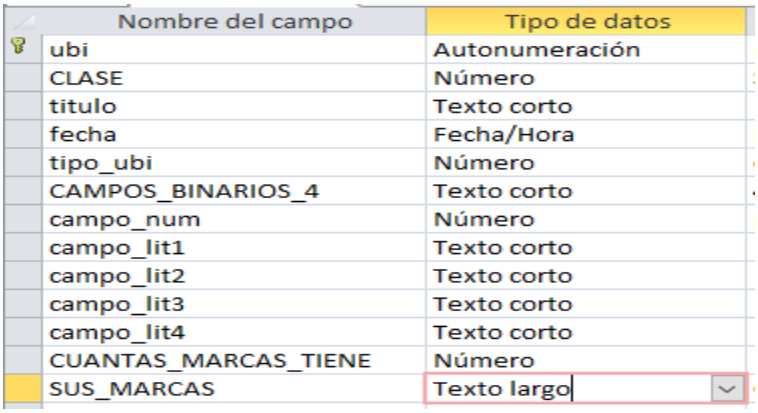
Figura 44 Ejemplo de cómo se almacenan marcas usando la alternativa Alt 3, captura Access.

Comentario: este diseño sólo sería interesante en un acervo de datos en los cuales las UBIs estarían marcadas con varios valores de un mismo contexto.

Alt 4. Almacenar las marcas concatenadas en una cadena que se guarda en un campo *ad hoc* de la tabla de las UBIs. Como se verá, esta estructura es conveniente para la gestión de las marcas, pero no para el armado de los subconjuntos, de modo que se usaría otra estructura para almacenar los triplos (contexto, valor, UBI) como

se ve en la Figura 45. Las marcas se concatenan con algún tipo de estándar por ejemplo 999 + 16 + SEPARADOR (;).

Esta estructura contempla el modelo adoptado para almacenar las UBIs. La tabla contendrá un campo tipo texto largo como se ve en la Figura 45.



Nombre del campo	Tipo de datos
ubi	Autonumeración
CLASE	Número
titulo	Texto corto
fecha	Fecha/Hora
tipo_ubi	Número
CAMPOS_BINARIOS_4	Texto corto
campo_num	Número
campo_lit1	Texto corto
campo_lit2	Texto corto
campo_lit3	Texto corto
campo_lit4	Texto corto
CUANTAS_MARCAS_TIENE	Número
SUS_MARCAS	Texto largo

Figura 45 Campos de la opción Alt 4 Almacenar las marcas concatenadas en una cadena que se guarda en un campo *ad hoc* de la tabla de las UBIs, captura Access.

Aquí se contemplan dos alternativas: Usar un campo TEXTO, y cuando no alcanza el tamaño para guardar todas las marcas se crea un registro adicional. O usar un campo “MEMO” que tendrá la capacidad de almacenar un gran número de marcas. La otra posibilidad: para evitar los tamaños “enormes” de estos campos, se podrían incluir varios de ellos para las marcas: si no se utilizaran (porque no resultaron necesarias) no ocuparían espacio en disco (al estar vacíos).

Cuando se emplea el manejador de bases de datos ACCESS, el campo MEMO es capaz de almacenar 65535 caracteres cuando se introducen datos a través de la interfaz de usuario, 1 gigabyte de almacenamiento de caracteres al introducir datos mediante programación. Mientras que el campo texto amacena 255 caracteres.

7.5 Estructuras para almacenar los triplos (contexto, valor, ubi)

Alt 1. El modelo “evidente” mencionado anteriormente: una tabla de una base de datos con los tres campos.

Alt 2. Guardar las listas de las UBIs de un par (Contexto, valor) en un arreglo grabado en disco. En el registro del “VALOR” (es decir, contexto-valor), se indica la ubicación de esta lista.

Es decir, se agregan al registro de la tabla VALORES los campos:

- Archivo_numero as Integer.
- Posicion-inicial as Integer.
- Cuantas marcas (o posiciones del arreglo) ocupa.

Esto contempla que, para no tener archivos demasiado grandes, se usen varios de ellos. Las listas de las UBIs marcadas con un contexto-valor se graban como arreglos en disco, que a su vez es parte de un arreglo de todas las marcas grabadas en el archivo. Esto plantea una situación: cuando se agregan UBIs a un tal arreglo, ya no podrá ocupar el mismo espacio en el disco, de modo que hay que grabarlo en otro lado. Esto resulta en que el espacio anterior estaría desaprovechado. Para disminuir el impacto de esta circunstancia, se controlan los espacios disponibles y se asignan a listas que hay que grabar. Este tema se detalla en el CAPÍTULO 11 donde se presenta el modelo técnico total.

Alt 3. Usar una tabla de datos con los siguientes campos:

- Contexto.
- Valor.
- Consecutivo.
- 16 campos enteros (para los números de UBI).

Se crean tantos registros como fueran necesarios para almacenar todas las UBIs marcadas con el par (contexto, valor) ordenadas en forma ascendente.

Este diseño tiene una desventaja enorme. Puesto que se desea que las UBIs se acomoden en forma ascendentes, cuando se agrega una UBI a la lista, si no cabe en un registro donde debería ser insertada, hay que crear un registro intermedio o “recorrer” todas las UBIs a un consecutivo siguiente. Por lo tanto se descartó este diseño.

Alt 4. Usar una tabla con los siguientes campos:

- Contexto.
- Valor.
- Menor_ubi_del_registro.
- 16 campos enteros (para los números de UBI).

Aquí se elimina la necesidad de numerar los registros (consecutivo).

Para insertar la UBI número NN, se ubica el registro para el cual menor_ubi_del_registro es mayor que el NN. Se usa el registro anterior a ese (si no hay uno, se crea).

Se agrega NN al arreglo de 16 campos. Si no cabe, se divide el registro en 2, y siempre se actualiza el campo menor_ubi_del_registro. Este diseño se implementó en una versión anterior, pero presentó la desventaja de usar una tabla con un gran número de registros. En comparación con la alternativa Alt 1, disminuye el espacio total en disco, pero sigue ocupando mucho más que la alternativa Alt 2.

7.6 Criterios que se usaron para comparar las estructuras

1. Eficiencia computacional, especialmente en operaciones de entrada y salida.
2. Flexibilidad para incorporar las características de las Marcas que no se pueden implementar vía “campos fijos”.
3. Oferta de diversos modos de armar subconjuntos (BUSQUEDAS) especialmente en cuanto a la ejecución de fórmulas entre pares (contexto, valor).
4. Eficiencia al mostrar los resultados de búsquedas.

5. Manejo de los datos en memoria.
6. Espacio en disco.
7. Aspectos de consistencia (especialmente pérdida de información).

7.7 Comparación de las estructuras

Se ha mencionado previamente que las marcas (TRIPLOS) deben estar accesibles para 2 usos:

- Encontrar las marcas (pares) de una determinada UBI.
- armar conjuntos de UBIs que tienen una cierta Marca.

Esto plantea dos alternativas fundamentales: se usa un modelo que permite usar sus datos para ambos fines, o se almacenan las marcas en 2 estructuras, una para cada uso.

La única alternativa que permite ambos usos en forma relativamente eficiente es la denominada Alt 1. Pero ésta a su vez tiene las desventajas que se describieron para ella. No se usan dos estructuras, pero la duplicación de información (en tabla e índices) hace que no sea una desventaja el uso de dos estructuras independientes: en una se guardan las marcas de una UBI (objeto de este capítulo) y en la otra, los triplos que servirán para las búsquedas.

7.8 Decisión: las estructuras seleccionadas para esta versión del DBB

Observación: el criterio predominante fue el espacio en disco, aunque cuando intervienen pares (contexto, valor) con un gran número de marcas (UBIs) la solución adoptada también reduce notablemente los tiempos de proceso. Se agrega que con las computadoras actuales y las velocidades de procesos, los tiempos pueden ser muy breves, de modo que un proceso que tarda “el triple que otro” puede tener una duración de milisegundos.

Para almacenar las marcas de una UBI se concatenan los pares (contexto, valor) en una cadena que se guarda en un campo *ad hoc* de la tabla de las UBIs.

Para almacenar los triplos, se adoptó la estructura Alt 2: Guardar las listas de las UBIs de un par (Contexto, valor) en un arreglo grabado en disco. En el registro del “VALOR” (es decir, contexto-valor, se indica la ubicación de esta lista.

CAPÍTULO 8. CATÁLOGOS Y OPCIONES DIVERSOS

Además de los elementos sustantivos (las UBIS, sus marcas y los catálogos de contextos y sus valores) el DBB se apoya en algunos catálogos o listas de elementos a los que se refieren algunos campos de dichos elementos.

Se describe cada uno de estas entidades complementarias, y se agrega en cada caso la descripción y el resultado del proceso de la estructura seleccionada para almacenar los datos involucrados. Sin embargo, el modo de presentación varía de los restantes capítulos. Aquí se especifica la estructura seleccionada para el DBB, y se comentan brevemente las otras estructuras contempladas y el motivo por el cual fueron desechadas.

8.1 Paths (directorios)

El DBB usa un catálogo de “directorios”. Cuando se especifica un archivo para cualquier uso, el directorio se asigna mediante un número entero. Para ello se construye un catálogo de directorios (que se denominan Paths) que tiene la particularidad de asignar un número a cada directorio individual. Esto resulta en que un Path está “anidado” en otro, y así sucesivamente. Por ejemplo, el directorio “c:\dbb\base1” podría estar indicado como se ve en el Cuadro 23.

Cuadro 23 Descripción de paths.

Número de Path	DIRECTORIO	anidado en
1	C:	0
2	Dbb	1
3	Bas 1	2

Es importante señalar que un Path puede tener varios “Hijos” (anidados en él).

El propósito principal del uso de los números de Path consiste en permitir cambiar un solo directorio, que afecta a todos sus hijos. Por ejemplo si un conjunto de datos

se puede utilizar en la computadora del cliente, el directorio tendría el valor “C:”, pero al usarlo en forma remota, se cambiaría por “Http…..”.

Los Paths se almacenan como un arreglo de “Estructuras” en el archivo del SITE. De ese modo, hay un segmento de este archivo en el que se encuentran. Se decidió imponer un máximo de 99 Paths.

Estructura: TPATH

- Número de Path (superfluo puesto que está dado por la posición en el arreglo).
- Nombre del directorio (siempre será “simple” y finaliza con una “\”.
- Número del “dad” es decir, el directorio en el cual éste está anidado.

Se usan mediante un arreglo de estas estructuras.

LOS_PATHS (99) AS TPATH (no se ocupa la posición 0 de este arreglo).

Se contemplaron otras dos estructuras.

- Una tabla de una base de datos: cada registro indicaría un Path. El índice principal sería precisamente dicho número. Los campos son los mismos que tiene la estructura TPATH.
- Usar un archivo con acceso aleatorio en el cual cada Path ocuparía un registro correspondiente a su número.

Se contemplaron 3 funciones principales:

- Cargar los Paths a memoria o crear una imagen de la tabla.
- Proporcionar el directorio “completo” de un número de Path.
- Actualizar el catálogo de Paths.

Al haber un máximo de 99 Paths, el espacio en disco no es un factor a tomar en cuenta.

La tabla de una base de datos tiene la desventaja de que, al crear una tabla en memoria, puesto que los registros no tienen por qué estar en forma consecutiva, siempre habrá que buscar en la tabla (DataTable) el registro deseado.

El archivo con acceso aleatorio tendría que ser un archivo propio, es decir, no contendría otra información. Esto está reñido con el criterio de minimizar el número de archivos que usa del paquete.

Con la estructura seleccionada, se carga la tabla a memoria (con una lectura única que lee todo el arreglo). Cuando se actualiza, se cambian los datos en memoria y se regraba el arreglo en disco.

La desventaja se presenta por aumentar ligeramente la dificultad de programación, puesto que hay que tomar en cuenta la posición de este arreglo en el archivo del SITE, que tiene diversos segmentos de los cuales el de Paths es sólo uno.

8.2 Nombres de fechas

Una UBI tiene un campo de uso “variable”: Se interpreta ý usa de acuerdo a la clase a la que pertenece la UBI. De ese modo, cuando se define una clase, se indica si se ocupa dicho campo (fecha); en caso positivo, se le asigna un “tipo de fecha” que resulta en que aparezca un nombre de ese campo que se especifica en la clase.

Para garantizar cierta homogeneidad de estos nombres, se creó un catálogo de tipos de fecha. Es importante señalar que este catálogo no se graba en disco, sino que se arma en memoria por código de programación. Los tipos de fecha que contempla el DBB en su versión actual son: Observe que el arreglo es de 2 dimensiones. La primera corresponde al idioma (1 = inglés, 2 = español).

```
Valores del arreglo los_tipos_de_fecha (2,15) – no se aprovechan las posiciones
0 en ninguna de las 2 dimensiones.
los_tipos_de_fecha (1, 1) = "UBI CREATION"
los_tipos_de_fecha (1, 2) = "LAST UPDATE"
los_tipos_de_fecha (1, 3) = "BIRTH"
los_tipos_de_fecha (1, 4) = "DEATH"
los_tipos_de_fecha (1, 5) = "ANNIVERSARY"
los_tipos_de_fecha (1, 6) = "PUBLICATION"
los_tipos_de_fecha (1, 7) = "GRADUATON"
los_tipos_de_fecha (1, 8) = "START VOYAGE"
los_tipos_de_fecha (1, 9) = "CREATION"
```

```
los_tipos_de_fecha (1, 10) = "FECHA DEL EVENTO"
los_tipos_de_fecha (1, 11) = "APOINTMENT DATE"
los_tipos_de_fecha (1, 12) = "EXPIRATION"
los_tipos_de_fecha (1, 13) = "..."
los_tipos_de_fecha (1, 14) = "" ‘no se ocupa
los_tipos_de_fecha (1, 15) = "" ‘no se ocupa
INDICA QUE LA CLASE NO USA EL CAMPO FECHA
los_tipos_de_fecha (1, 14) = "EVENT DATE"
los_tipos_de_fecha (1, 15) = "OBVIOUS"
los_tipos_de_fecha (2, 1) = "CREACION UBI"
los_tipos_de_fecha (2, 2) = "ULT ACTUALIZ"
los_tipos_de_fecha (2, 3) = "NACIMIENTO"
los_tipos_de_fecha (2, 4) = "DEFUNCIÓN"
los_tipos_de_fecha (2, 5) = "ANIVERSARIO"
los_tipos_de_fecha (2, 6) = "PUBLICACION"
los_tipos_de_fecha (2, 7) = "GRADUACION"
los_tipos_de_fecha (2, 8) = "INICIO VIAJE"
los_tipos_de_fecha (2, 9) = "CREACION"
los_tipos_de_fecha (2, 10) = "FECHA DEL EVENTO"
los_tipos_de_fecha (2, 11) = "FECHA DE CITA"
los_tipos_de_fecha (2, 12) = "VENCIMIENTO"
los_tipos_de_fecha (2, 13) = "..."
‘INDICA QUE LA CLASE NO USA EL CAMPO FECHA
 los_tipos_de_fecha (2, 14) = "” ‘nulo, no se usa
 los_tipos_de_fecha (2, 15) = "" ‘nulo, no se usa
```

8.3 Tipos de anexo

Ya se ha mencionado que uno de los elementos de datos de una UBI puede ser un archivo relacionado a la UBI., De hecho, se puede tratar también de un LINK (liga) a alguna página en la red. De ese modo, uno de los campos de una UBI es precisamente un descriptor del tipo de anexo.

Se decidió ofrecer a los usuarios un catálogo de tipo de anexos. En él se definen algunos elementos (numerados). En las UBIs se indica el tipo de anexo mediante el número de éste, y en los programas se interpretan usando este catálogo.

Para almacenarlo se decidió usar un arreglo de dimensión 50 (se contempla un máximo de 50 tipos). Se almacena en el archivo SITEFILE con la estructura:

- número (byte).
- descripción (texto tamaño fijo).

El uso de una tabla de la base de datos no ofrecía ninguna ventaja excepto que no exigía incluir un máximo del número de tipos de anexos. Del mismo modo que en los Paths, el uso de un archivo con acceso aleatorio no ofrecía ninguna ventaja.

8.4 Tipos de UBI

El uso del campo tipo de UBI debe ser flexible para los usuarios de un conjunto de UBIs. Para ello se incluyó un campo Tipo de UBI como elemento de estas unidades. Se consideró conveniente usar una tabla (catálogo) de modo que se indique en la UBI un número de tipo, y se interpreta utilizando la mencionada tabla. La estructura utilizada y las alternativas son las mismas que para los tipos de anexo, de modo que no se repite aquí.

8.5 Extensiones válidas

El DBB permite indicar ciertos archivos para usos diversos. El típico es asociar un archivo a una UBI. Cuando se usa esta facilidad, el sistema permite “ver” dicho archivo. Para ello, se basa en la “extensión”, que en general describe el contenido del archivo, pero además lo asocia con algún producto de Software que permite visualizarlo.

A pesar de que cuando se indica un tal archivo se puede usar cualquier extensión, el DBB no muestra dicho archivo si la extensión no está en el catálogo. De hecho, el DBB se basa en esta extensión para invocar el programa necesario para desplegar el archivo.

Es importante señalar que en el catálogo de extensiones NO se indica el programa correspondiente. Esto lo hace el programa por código. Este hecho limita la actualización del catálogo de extensiones, puesto que incluir una nueva extensión no resulta en que se pueda mostrar un archivo que la tenga: hay que modificar el programa correspondiente. Esta operación en general se reserva a las actualizaciones del DBB (preparación de una versión siguiente):

Las extensiones se guardan en un arreglo de tamaño 24 (además de la posición 0 que no se usa). Para cada extensión se almacena la extensión y una descripción que en general indica el Software que usa el programa para desplegar el archivo.

El uso de una tabla de una base de datos no hubiera agregado funcionalidad. Sin embargo, hubiera evitado la especificación del máximo (en este caso 24) extensiones que se podrían introducir.

8.6 Opciones del SITE (instalación)

El DBB contempla que se pueden limitar las funciones en un SITE en particular. Esto se puede deber a alguna restricción relacionada con la distribución del producto o una decisión del que crea el SITE por considerar que no necesita la funcionalidad indicada. Cabe señalar que al indicar que no desea alguna característica, el sistema nunca ofrece ni solicita un dato referente al tema. Se presentan las opciones como estructura: ésta se graba como parte del archivo SITEIL

```
Public Structure tOPCIONES
       Public usa_2_idiomas As Byte
       Public cual_idioma_usa As Byte
       Public usa_equivalencias As Byte
       Public uses_several_books As Byte
       Public offers_read_aloud As Byte
       Public offers_show_items As Byte
       Public ufu1 As Byte
       Public ufu2 As Int32
End Structure
```

El uso de una tabla de una base de datos relacional hubiera resultado en un registro único con estos campos, o con un registro para cada una de las opciones.

8.7 Parámetros del SITE

El DBB usa ciertos parámetros en los programas. Estos parámetros (a diferencia de las opciones) pueden ser actualizados en cualquier momento. El campo AUDIT

permite proteger los valores contra modificaciones no autorizadas de algún parámetro.

```
Public Structure tconstantesYPARAMETROS ' 79 bytes
       Public last_update As DateTime
       <VBFixedString (24)> Public site_name As String
       <VBFixedString (24)> Public site_owner_name As String
       Public base_de_datos_default As Byte
       ‘si está se usa ésta; se ofrecen las restantes como excepción
       Public no_pedir_pwd_user_default As Byte
       Public num_user_default As Int32
       Public idioma_user_default As Byte
       Public siteownerpwd As Int64
       Public audit_parametros As Int64
End Structure
```

Se almacena la estructura (es decir, una variable definida como tal) en el archivo SITEFILE. Se descartó una vez más la inclusión como una tabla de una base de datos relacional. El motivo es el mismo que el descrito referente a las opciones.

8.8 Grupos de contextos

Hay un máximo de 255 contextos que se pueden definir. Cuando una aplicación utiliza un número considerable, por ejemplo, 120 contextos, el proceso de seleccionar un contexto para un uso determinado consiste de seleccionarlo de una lista de éstos. Para facilitar esta operación, se pueden definir grupos de contextos y limitar las listas de éstos que se ofrecen a los de un grupo.

Cabe señalar que como en todas las funciones del DBB se trató de que fuera optativo el uso de ciertos atributos. Es decir, se pueden definir todos los contexto como parte de un grupo 0, o de un grupo 1 “único” si no se desea aprovechar esta funcionalidad.

La estructura seleccionada para almacenar estos grupos es un arreglo (con una capacidad máxima de 50 grupos de contextos). Se graba el arreglo como un segmento del archivo SITEFILE. Esto permite cargar el arreglo a memoria al inicio

de una sesión de trabajo. Cuando se modifica algún grupo, se efectúan las operaciones en el arreglo cargado en memoria y se regraba todo el arreglo.

La alternativa contemplada era una tabla de grupos de contextos, pero no ofrecía ninguna ventaja, de modo que se adoptó el mecanismo de guardar un arreglo en disco.

8.9 Grupos de clases

A pesar de que hay un máximo de 50 clases que se pueden definir para un SITE, se estimó conveniente una vez más permitir que se agruparan de algún modo. Esto permite que en una de las bases de datos se excluyan algunas clases como utilizables en dicha base. Cuando se incluye una UBI en la base, la primera operación será seleccionar la clase a la que pertenece. Esta lista de clases se puede limitar por grupo de clases (los grupos que usa la base están registrados como parte de la descripción de la base en el catálogo de éstas).

El catálogo de grupos de clases se almacena como arreglo (de tamaño fijo 8, que es el máximo número de grupo de clase permitido). Cada grupo está representado por una estructura, que en este caso consiste de los siguientes campos:

- Número de grupo (superfluo por ser igual al índice del arreglo).
- un nombre o descripción para distinguirlo de otros grupos (en cada uno de los 2 idiomas que ofrece el DBB).
- si procede, un rol que debe tener un usuario para usar las clases que corresponden a este grupo de clase.

Una vez más se desechó el uso de una tabla de la base de datos puesto que no ofrecía ninguna ventaja sobre el uso de un arreglo en un archivo de disco plano.

CAPÍTULO 9. LISTAS DE RESULTADOS

Quizá la componente más importante del DBB son las listas de resultados (o listas de marcas). Se trata de conjuntos de números de UBI que poseen cierto atributo. Hay dos tipos de listas de resultados (aunque la primera no se denomina de ese modo). Las estructuras posibles son las mismas para ambos tipos de listas.

- Las listas de los números de UBIS marcadas con un par CONTEXTO-VALOR que fueron descritas en el 0 sobre marcas.
- Las listas de números de UBIs que resultaron de operaciones entre listas de marcas (y otras listas de resultados).

Las operaciones booleanas (definidas por una fórmula que puede tener varios operandos, o solamente uno de éstos) se efectúan entre las listas de marcas y listas de resultados obtenidas anteriormente.

En los MÉTODOS se comentaron las operaciones entre conjuntos ordenados de números (en el caso del DBB, de números de UBI). De ese modo, en memoria se pueden usar diversas estructuras para construir una lista de resultados.

El programa determina la estructura utilizada para armar estas listas resultantes de operaciones. El sistema les asigna un número y crea un archivo donde graba la estructura usad; se crea un archivo para cada lista. Estos archivos son “temporales”: se indica si tienen una vigencia (se indica ésta) o el usuario los elimina cuando ya no las necesite. Se usan tres estructuras para armar y almacenar las listas de resultados:

Como arreglos de enteros. Al principio del archivo está el número de elementos que tiene el arreglo, y a continuación se graba el arreglo con una operación única de escritura. La recuperación se hace en dos pasos: se lee el (entero) que indica cuantos elementos tiene la lista. Se redimensiona un arreglo y se carga la lista.

Como bitmap: se arma el bitmap de modo que incluya los números de UBI desde el menor hasta el mayor que resulta de la operación.

Como intervalos de bitmaps: en lugar de un bitmap “total”; se arman bitmaps parciales, cada uno de los cuales representa un intervalo de números. Esta estructura se utiliza cuando en el bitmap total hay segmentos “grandes” intermedios sin usar.

Observación técnica (de programación). En el programa, una lista será una instancia de una de las 2 clases: arreglos de números y bitmaps. Esto permite crear y eliminar listas en memoria, lo que permite liberar el espacio ocupado por listas que ya no se usan.

Los archivos de listas de resultados residen en un subdirectorio del de la base de datos de la sesión. Esto permite eliminar con una sola operación todas las listas cuando ya no se necesiten en el futuro.

- El nombre de estas listas está formado por:
- Prefijo “ARCH.LR.”.
- Número de lista: siempre se agrega con 3 dígitos seguida por un “.”.
- Una fecha de vencimiento: Si no tiene fecha de vencimiento, se pone 00.00.00.00; Si tiene vencimiento se indica con formato. :MES.DD.HH.MM.
    - (2 dígitos cada uno, que pueden ser cero).
    - La hora (en minutos) de vencimiento.
    - La ausencia de un dato indica “todos”.
    - Ejemplo: 00.00.14.03 indica que después de las 2:03 pm.

El archivo será eliminado en el primer proceso que elimine Archivos de resultados.

La extensión de estos archivos de **.jbr** como todos los archivos planos del DBB. Finalmente, el *default* (cuando se crea una lista de resultados) es que vence ese día a las 00.00.23.59.

CAPÍTULO 10. USUARIOS AUTORIZADOS Y SUS ROLES

10.1 Introducción

Se determinó que los usuarios del DBB necesitan permiso para invocar y usar ciertas funciones del sistema. Se decidió adoptar un RBAC (Roll Based Access Control). Sin embargo, siguiendo el modelo de (Bauer Mengelberg, 2005), cada usuario tendrá un ROL en cada GRUPO de FUNCIONES.

El usuario inicia una sesión de trabajo. Lo hace con un “LOGON”. Indica su nombre o número de usuario (si no lo sabe, podrá buscarlo entre todos los usuarios).

Proporciona su palabra clave. El sistema valida la palabra: se inicia la sesión si es válida.

Si no sabe su palabra clave, podrá invocar el modo de “preguntas de seguridad”. El sistema le hace una pregunta y él debe proporcionar la respuesta (exacta) que había indicado como tal.

El sistema “sabe” en este momento los datos del usuario en cuanto a las funciones del

SITE. También conocerá los permisos que tiene para cada base de datos definida (usada).

Si tuviera permiso para una sola base, esta sería la que se usaría en la sesión de trabajo.

De lo contrario, se le ofrecen aquéllas a las que tiene acceso, de las cuales selecciona la que desea usar. Se verifica que la hora de inicio de la sesión no viole alguna de las restricciones de día u horarias si las tuviera. Si no puede comenzar la sesión, se le indica esta circunstancia y termina la sesión.

Finalmente, se cargan a memoria sus permisos (roles).

Por lo tanto en el DBB “Un usuario” se interpreta de dos modos:

- Como Usuario de SITE Mismo
- Como Usuario de cada una de las bases que puede utilizar.

De ese modo, tendrá roles en el SITE o en la Base de datos.

Bauer Mengelberg (Bauer Mengelberg, 2005) incluye lo que llama DVDC (Data Value Dependent Constraints) para poder limitar el uso de las funciones a ciertas variables (de entorno, valores de algún dato, otras). En DBB, puesto que no está basado en tablas y registros, sino en CLASES y “UBI” (unidad básica de información) de cada clase, es necesario incluir algún mecanismo que permita limitar lo que puede hacer un usuario de DBB con los datos de cada clase por separado.

De ese modo, se adoptó un modelo diferente. Para las funciones referentes a DATOS (siempre serán datos de la base, puesto que el SITE no tiene UBIs), tendrá permisos en cada una de las 50 (posibles) CLASES. En cuanto a los DATOS, los GRUPOS son las CLASES (de 1 a 50). El dato no se usa si la clase no está definida (o si la base de datos que contiene las UBIs no utiliza alguna clase).

10.2 Lista de funciones

Se elaboró una Lista de las funciones que necesitan alguna autorización antes de permitir el uso por el usuario de la sesión. Se agruparon las funciones en dos aspectos:

- Son funciones aplicables a datos del SITE.
- Funciones aplicables a datos de cada base de datos (incluye UBIs).

A su vez, se decidió asignar los permisos en tres grupos (Roles) L:

- DATOS (de la base de datos).
- KBC (funciones para el marcado).
- ADMINSTRACIÓN.

10.2.1 Las funciones protegidas en el DBB

La lista de funciones que fue definida para los usuarios de cada base se muestra en el Cuadro 24, mientras que en el Cuadro 25 se muestran los de los usuarios del SITE.

Cuadro 24 Permios de usuarios en cada base de datos.

	#	En inglés	En español
Funciones de DATOS (Aplican a cada clase)			
1	1	Use the UBIs	Ver las UBIs
2	2	change certain fields	Modificar algunos elementos
3	3	add new UBIs of this class	Agregar UBIs nuevas de esa clase
4	4	delete UBIs o cambiar cualquier dato	Eliminar UBIs o cambiar cualquier dato
Funciones de KBC			
5	1	update contexts	actualizar contextos
6	2	change groups of contexts	Cambiar grupos de contextos
7	3	add values of contexts	agregar valores de contextos
8	4	authorize values	Autorizar valores
Funciones de admin de cada base de datos			
9	1	add new users (of this base)	Introducir usuarios nuevos (de esta base)
10	2	update admin role of other users	cambiar rol admin de otros usuarios
11	3	change kbc and data roles of other users	cambiar roles datos y admin de otros usuarios
12	4	compress files and database	Comprimir archivos y bases de datos
13	5	restore data from backups	Bajar un respaldo de datos
14	6	clear backups	Limpiar respaldos

Cuadro 25 Permisos de usuarios del SITE.

	#	En inglés	En español
Funciones de KBC			
15	1	add or delete contexts	Agregar o quitar contextos
16	2	change groups of contexts	Cambiar grupos de contextos
17	3	add values of contexts	Agregar valores de contextos
18	4	authorize values of contexts	Autorizar valores de contextos
19	5	establish structures to be used	establecer estructuras a usar
Funciones de admin del SITE			
20	1	update catalogues	Actualizar catálogos
21	2	update PATHS	Actualizar PATHS
22	3	update site users	Actualizar usuarios del site
23	4	update users of each database	Actualizar usuarios de cada base de datos
24	5	emergencies (forgot pwd, etc)	Emergencias (olvidó pwd, etc.)
25	6	change a user's security question	cambiar pregunta de seguridad de 1 usuario
26	7	update CLASSES	cambiar pregunta de seguridad de 1 usuario
27	8	update groups of classes	Actualizar grupos de clases
28	9	compress files and databases	Comprimir archivos y bases de datos
29	10	use a data backup	Bajar un respaldo de datos
30	11	clear backups	Limpiar respaldos
31	12	reserved for future use	reservado para uso futuro
32	13	reserved for future use	reservado para uso futuro

Basado en estos conceptos de protección de funciones, se elaboró el modelo conceptual de datos de AC (Access Control) ilustrado en la Figura 46.

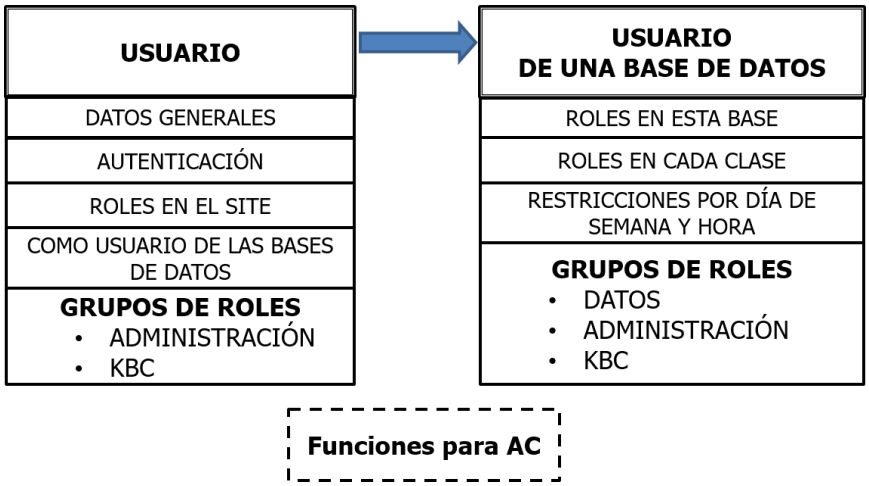


Figura 46 El modelo conceptual de datos de Access Control en DBB, elaboración propia.

La siguiente etapa fue determinar las estructuras en las cuales se almacenan los permisos (y los datos de cada usuario), pero también las constantes (funciones y preguntas de seguridad). Se decidió que los roles se calcularían en base a los permisos, en lugar de definir roles (por nombre) y asignar estos roles a los usuarios.

Estructura seleccionada para almacenar la lista de funciones.

La lista es definitiva, es decir, no podrá sufrir modificaciones. Por lo tanto se compararon 2 estructuras:

- Un arreglo almacenado en SITEFILEUSE T.
- Una tabla de la base de datos SITEBASE.

De hecho, se postergó esta decisión para tomar en cuenta la estructura utilizada. Para guardar los Usuarios del SITE. Como ésta (como se verá) fue la de incluir una tabla de la base de datos SITEBASE, se decidió hacer lo propio con las funciones a proteger, dando como resultado una tabla como se muestra en la Figura 47.

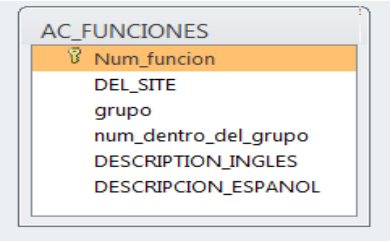


Figura 47 Tabla AC_FUNCIONES funciones de acceso de control, captura Access.

Las funciones tienen 2 números:

- Un consecutivo (de todas las funciones).
- Un número dentro de su “Grupo”.

Este es un dato técnico. El sistema usa ambos números según el que convenga a alguna rutina.

10.2.2 Las preguntas de seguridad

Un usuario se puede autenticar (cuando no recuerda su palabra clave) mediante la respuesta correcta a una pregunta de seguridad. Proporcionó dicha respuesta como parte del proceso de inclusión del usuario como tal.

Se seleccionaron dos estructuras para almacenar las preguntas de seguridad, que también son definitivas (no se pueden cambiar las preguntas).

Se compararon 2 estructuras:

- Un arreglo almacenado en SITEFILE
- Una tabla de la base de datos SITEBASE

Del mismo modo que para las funciones, se postergó esta decisión para tomar en cuenta la estructura utilizada para guardar los Usuarios del SITE. Como ésta (como se verá) fue la de incluir una Tabla de la base de datos SITEBASE, se decidió hacer lo propio con las preguntas de seguridad guardándolas en una tabla como se ve en la Figura 48, mientras que en el Cuadro 26 se enlistan las 10 preguntas que han sido consideradas para este caso.

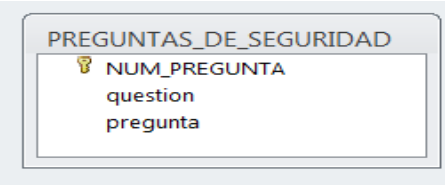


Figura 48 Campos de la tabla Preguntas de seguridad, captura Access.

Cuadro 26. Las 10 preguntas de seguridad incluidas en DBB.

No.	QUESTION (en inglés)	PREGUTA (en español)
1	Name of your favorite aunt	Nombre de su tía favorita
2	Color of your favorite car	Color de su auto favorito
3	Name of your favorite teacher	Nombre de su maestro/maestro favorito
4	Your favorite movie	Su película favorita
5	Your favorite autor	Su autor favorito
6	Your favorite song	Su canción favorita
7	A phrase you remember	Una frase que recuerda
8	A string of characters you will remember	Una cadena de caracteres que recordará
9	The number of your bank account	El número de su cuenta bancaria
10	A phone number you always remember	Un número de teléfono que siempre recuerda

Comentarios sobre el uso de las preguntas de seguridad.

Durante el registro como usuario, selecciona UNA pregunta:

- Indica una respuesta (que debe recordar): **no puede exceder de 40 caracteres.**
- Por lo tanto, se recomienda elegir una respuesta muy conocida.
- Se pueden agregar o insertar espacios y caracteres especiales, pero con la advertencia de que se toman en cuenta para su uso. No se distinguen mayúsculas y minúsculas.

Se registra la respuesta (encriptada). Es importante señalar que no se puede recuperar la respuesta. Si no logra proporcionar su palabra clave, se invoca el uso de preguntas de seguridad.

La pregunta se presenta en el idioma aplicable en ese momento en la sesión. Para usar la pregunta como autenticación, la respuesta proporcionada **debe coincidir**

exactamente con la proporcionada (una vez más, minúsculas se reemplazan por mayúsculas). Pero cuidado, cuentan los caracteres especiales y espacios.

ADVERTENCIA: se permitirán 5 intentos de reproducir la respuesta proporcionada. Si no se reproduce la correcta en estos intentos, el usuario será marcado como "inhabilitado". En ese caso, deberá recurrir al administrador del SITE para que lo reinstale. Es importante señalar que no podrá intentar iniciar una sesión (y allí seguir proporcionando respuestas).

10.2.3 Los usuarios del DBB

Del site

Cada usuario del site tendrá un registro en la tabla USERS de SITEBASE mostrada en la Figura 49.

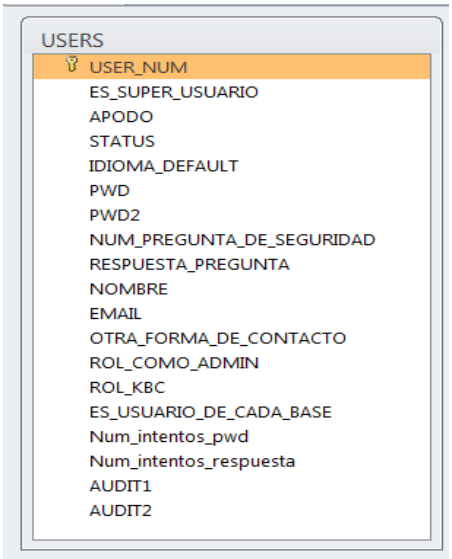


Figura 49 Campos de la tabla USERS de SITEBASE, captura Access.

Detalles sobre los campos de la tabla USERS:

- **Número de usuario:** número asignado por el sistema.
- **Es superusuario:** indica que tiene todas las funciones del site.
- **Apodo:** un nombre que puede usar para el inicio de una sesión ("LOGON").

- **Status:** dato técnico. Indica si está habilitado o no, o pendiente de algún tipo de acción por parte del administrador.
- **IDIOMA_DEFAULT:** es el que se usa para la pregunta de seguridad, y el que se establezca como idioma de la sesión de trabajo.

Datos de autenticación:

- **Palabra clave**: una cadena de (no más) de 1 caracteres.
- **Palabra clave adicional para funciones doblemente protegidas:** Lo mismo. Esta palabra se solicita para ciertas funciones delicadas.
- **Numero de Pregunta de seguridad.**

- **Respuesta a pregunta** *Observación: las palabras clave, el número de pregunta y la respuesta se almacenan encriptados.*

Para contactar al usuario:

- **NOMBRE.**
- **Email.**
- **Otra_forma_de_contacto:** es un texto que indica el modo y los datos (por ejemplo, celular 12345678).

SUS ROLES (cuáles funciones puede utilizar).

- **Rol_COMO_ADMIN**: es un entero (2 bytes) que indica si puede usar cada una de las funciones de este grupo, estos roles se enlistan en el Cuadro 27.

El algoritmo para calcular el ROL es el mismo, pero ahora el ROL es entero puesto que el valor máximo del rol será 2^{12} -1 = 4095.

Cuadro 27 Roles de un usuario.

1	ACTUALIZAR CATÁLOGOS
2	ACTUALIZAR PATHS
3	ACTUALIZAR USUARIOS DEL SITE
4	ACTUALIZAR USUARIOS DE BASES
5	EMERGENCIAS (PWDS, ETC.)
6	CAMBIAR PREGUNTAS DE SEGURIDAD
7	ACTUALIZAR CLASES
8	ACTUALIZAR GRUPOS DE CLASES
9	COMPRIMIR ARCHIVOS Y BASES
10	BAJAR UN RESPALDO DE DATOS
11	LIMPIAR RESPALDOS
12	PARA USO FUTURO 1
13	PARA USO FUTURO 2

ROL_KBC: es un byte que indica si puede usar cada una de las funciones de este grupo.

Se usa el mismo algoritmo para calcular el rol, Para cada función autorizada, se suma al rol:

2 elevado a la potencia (número de función -1).

Cuadro 28 Funciones de un grupo.

1	AGREGAR O QUITAR CONTEXTOS
2	CAMBIAR GRUPOS DE CONTEXTOS
3	AGREGAR VALORES
4	AUTORIZAR VALORES
5	DEFINIR ESTRUCTURAS A USAR

Permisos para usar las bases de datos:

Es_usuario_cada_base cadena 9 dígitos 0/1.

Datos para uso del sistema:

Num_intentos_pwd.
Num_intentos_respuesta.
Audit1.
Audit2.

Los datos técnicos se usan para implementar algunos aspectos de protección. Los AUDIT son “cifras de control” calculadas a partir de ciertos campos del registro.

De cada base de datos

Para utilizar los datos contenidos en una base de datos (del DBB) el usuario debe estar registrado en la tabla USER_BASE de la base UBIBASE, el diseño de la tabla se muestra en la Figura 50.

Figura 50 Los campos de la tabla USER_BASE, captura Access.

Los campos de la tabla USER_BASE

USER_NUM: su número como usuario registrado del SITE
STATUS: 0=INACTIVO, 2 = SUSPENDIDO 3=ACTIVO
LOS_ROLES_DATOS: es una cadena de 50 dígitos: cada digito es el rol que tiene el usuario para cada clase.

El rol es ascendente (0 a 5). Si tiene rol 3, puede usar 1, 2,3

0 = no puede usar UBIs de esa clase
1 = puede verlas
2 = puede modificar algunos elementos
3 = puede agregar UBIs nuevas de esa clase
4 = Puede eliminar UBIs o cambiar cualquier dato.

ROL_KBC: byte calculado a partir de las funciones: se especifica el rol como un campo de 1 byte, que tendrá el valor.

Para cada función, se suma la potencia de 2 elevado al (número de función-1) si el usuario tiene permiso de usar esa función.

Ejemplos Si el usuario tiene permios para la funciones 1 y 3

Su “rol” será 1 + 4 = 5

En cambio, si el usuario tiene permios para la funciones 2 y 4

Su “rol” será 2 + 8 = 10

Para conveniencia del lector, en el Cuadro 28 se repiten las funciones de este grupo de funciones.

Cuadro 28 Funciones de un grupo KBC.

1	AGREGAR O QUITAR CONTEXTOS
2	AGREGAR VALORES
3	AUTORIZAR VALORES
4	DEFINIR ESTRUCTURAS A USAR

ROL_ADMIN: byte calculado: en la base de datos también tendrá un rol en el grupo ADMINISTRACIÓN. Se calcula estel ROL del mismo modo que para el grupo anterior.

Para cada función autorizada, se suma al rol 2 elevado a (número de función – 1). Una vez más se repiten las funciones de este grupo en el Cuadro 29.

Cuadro 29 Funciones del grupo Administración.

1	Introducir usuarios nuevos (darlos de alta)
2	Eliminar usuarios o cambiar sus roles de administración
3	Cambiar el rol que tiene en esta base a otros usuarios
4	Comprimir archivos y bases
5	Bajar un respaldo de datos
6	Limpiar respaldos

IDIOMA_DEFAULT: si se indica, se usa ese idioma cuando esté activa esta base.

RESTRICCIONES_DIAS: es una cadena de 7 dígitos 0/1 correspondiendo a los días de semana.

El primero corresponde al LUNES. Un 0 significa que el usuario no podrá iniciar una sesión ese día de semana.

RESTRICCION_HORA_DESDE: se indica hora * 100 + minutos

Si está indicado este dato, no podrá iniciar sesión antes de esa hora.

RESTRICCION_HORA_HASTA: se indica hora * 100 + minutos:

Si está indicado este dato, no podrá iniciar sesión después de esa hora.

AUDIT1: cifra de control calculada a partir de ciertos campos.

AUDIT2: cifra de control calculada a partir de otros campos.

CAPÍTULO 11. EL MODELO DE DATOS TÉCNICO ADOPTADO

11.1 Las bases de datos

Las bases de datos que finalmente serán implementadas son las siguientes.

11.1.1 SiteBase

Contiene la información aplicable a cualquier base de datos seleccionada: contextos, usuarios, parámetros, opciones, catálogos diversos, funciones protegidas y preguntas de seguridad. La base de datos contiene las tablas que se muestran en la Figura 51.

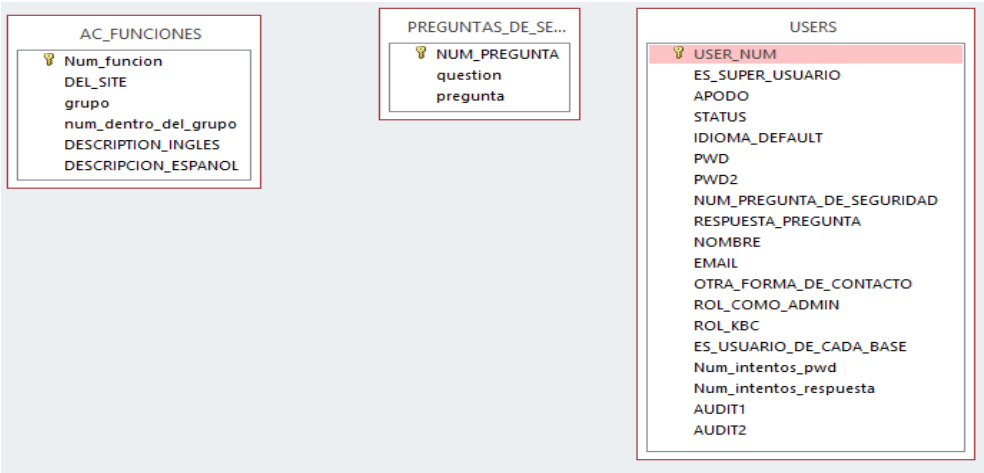


Figura 51 Las tablas que conforman el SITEBASE, captura Access.

11.1.2 UbiBase

Contiene información aplicable sólo a una de las bases (la activa). Incluye las UBIs mismas, los Valores de los contextos, las marcas de la UBIs y los roles (permisos) de cada usuario de esa base como se muestra en la Figura 52.

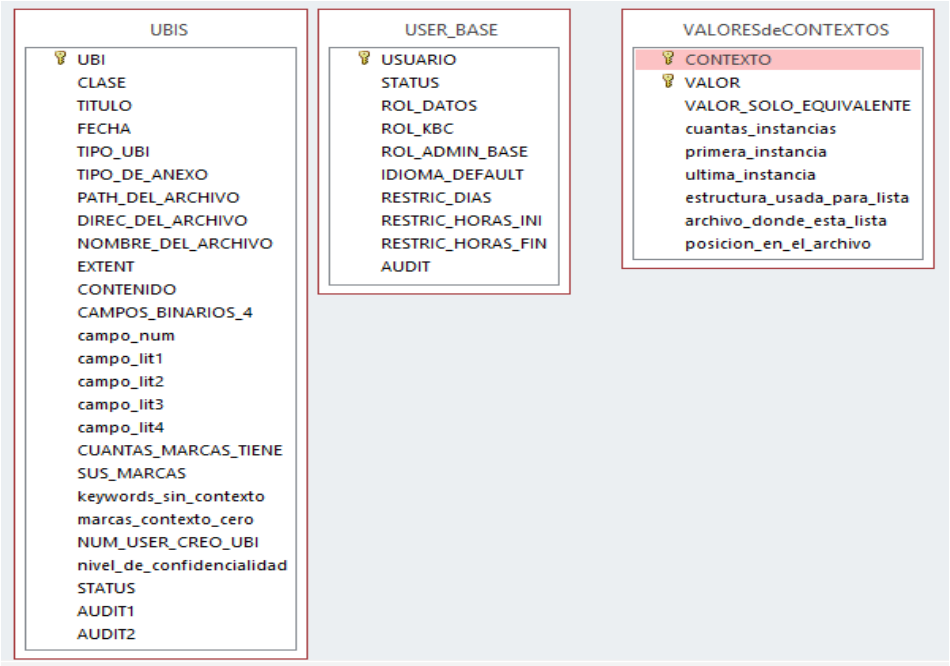


Figura 52 Las tablas que conforman UBIBASE, captura Access.

11.2 El archivo SiteFile

11.2.1 Descripción del archivo

SiteFile es un archivo plano. Está dividido (lógicamente) en SEGMENTOS, cada uno de los cuales corresponde a ciertos datos que usa el DBB.

El nombre del archivo es constante (inalterable): SiteFile. Jbr. El directorio en el cual se encuentra es – por default – es mismo del programa ejecutable. Si al inicio de una sesión no lo encuentra, pide el directorio donde está alojado.

El archivo está compuesto por 16 segmentos; cada uno contiene alguno de los datos que se almacenan. El primer segmento es el índice de segmentos; para leer uno de los segmentos, se utiliza la posición del primer dato de ese segmento. Las longitudes y posiciones iniciales (respecto al principio del archivo) se muestran en el Cuadro 30.

Cuadro 30 Posiciones y longitudes de los segmentos del archivo SITEFILE.

#	Longitud	Posición	Descripción
1	408	1	INDICE DE SEGMENTOS
2	79	409	PARAMETROS
3	11	488	opciones del site
4	867	499	Tipos de UBI
5	561	1366	Tipos de anexo
6	600	1927	Extents reconocidos
7	5000	2527	Los Paths
8	2930	7527	Las bases de datos
9	450	10457	Grupos de clases
10	4590	10907	Binarios de cada clase
11	14076	15497	Clases
12	2601	29573	Tipos de UBI de cada clase
13	1617	32174	Grupos de contextos
14	256	33791	Apuntadores a contextos
15	2	34047	Cuantos contextos hay
16	VARIABLE	34049	Arreglo de contextos

A continuación se muestran las estructuras (tipos de variable) usados para cada uno de los tipos de datos que contiene el archivo.

INDICE DE SEGMENTOS

```
Public Structure tdatosdeSEGmentos ' 24 bytes cada uno
       <VBFixedString(16)> Public NOM_ESTE_segmento As String
       Public LONGItud As Int32
       Public posicion As Int32
End Structure
Public INDICE_DE_SEGMENTOS_DEL_ARCHIVO(16) As tdatosdeSEGmentos ' 17 * 24
```

OPCIONES ACTIVAS

```
Public Structure tOPCIONES ' 11 bytes
       Public usa_2_idiomas As Byte
       Public cual_idioma_usa As Byte
       Public usa_equivalencias As Byte
       Public uses_several_books As Byte
       Public offers_read_aloud As Byte
       Public offers_show_items As Byte
       Public ufu1 As Byte
       Public ufu2 As Int32
End Structure
Public OPCIONES_ACTIVAS As tOPCIONES
```

PARAMETROS

```
Public Structure tconstantesYPARAMETROS ' 79 bytes
       Public last_update As DateTime
       <VBFixedString (24)> Public site_name As String
       <VBFixedString (24)> Public site_owner_name As String
End Structure
Public CONSTANTES_Y_PARAMETROS As tconstantesYPARAMETROS
```

TIPOS DE UBI

```
Public TIPOS_de_UBI(50) As tcatalogonumerico, MAYOR_TIPO_UBI_USADO As Short
Public Structure tcatalogonumerico ' 17 bytes
       Public clave As Byte
       <VBFixedString(16)> Public descripcion As String
End Structure
Public Structure tcatalogonumerico ' 17 bytes
       Public clave As Byte
       <VBFixedString (16)> Public descripción As String
End Structure
Public TIPOS_de_UBI (50) As tcatalogonumerico
```

TIPOS DE ANEXO

```
Public Structure tcatalogonumerico ' 17 bytes
       Public clave As Byte
       <VBFixedString(16)> Public descripcion As String
End Structure
Public TIPOS_DE_ANEXO(32) As tcatalogonumerico, MAYOR_TIPO_ANEXO_USADO As Short
```

LAS EXTENSIONES VALIDAS

```
Public Structure tcatalogoEXTENSIONES ' 24 bytes cada uno
       <VBFixedString (8)> Public la_extension As String
       <VBFixedString (16)> Public descripcion As String
End Structure
Public EXTENSIONES_VALIDAS(24) As tcatalogoEXTENSIONES, MAYOR_EXTENSION As Short
```

LOS PATHS

```
Public Structure tPATHS ' son 50 bytes
       Public num_path As Byte
       Public anidado_en As Byte
       <VBFixedString(24)> Public ruta As String
       <VBFixedString(24)> Public comentarios As String
End Structure
Public los_paths(99) As tPATHS ' nohay path 0 maximo 99 paths total 5000 bytes
```

LAS BASES DE DATOS

```
Public Structure TLASBASESDEDATOS ' 194 BYTES
       Public NUM_PATH As Byte ' si vale 100 es que no existe esa base
       <VBFixedString(80)> Public ruta_COMPLETA As String ' OPTATIVO
       <VBFixedString(40)> Public DESCRIPCION As String
       <VBFixedString(120)> Public PROVEEDOR As String
       Public usa_solo_esta_clase As Byte ' 0 indica varias
       Public desactivar_equivalencias As Byte ' 0 = no, 1=no se ofrecen
equivalencias
       <VBFixedString(50)> Public Cuales_clases_usa As String ' 0=no, X=si cada
byte
End Structure
Public LAS_Bdd_OFRECIDAS(9) As TLASBASESDEDATOS
```

GRUPOS DE CLASES

```
Public Structure tungrupoclases ' cada uno mide 34 bytes
       Public NUM_GRUPO As Byte
       Public necesita_rol As Byte
       'Roles para usar cada grupo (max 3; 0=sin restricciones)
       <VBFixedString(16)> Public nombre_grupo_clases_ingles As String
       <VBFixedString(16)> Public nombre_grupo_clases_espanol As String
       <VBFixedString(16)> Public nombre_grupo_clases_de_la_sesion As String
End Structure
Public LOS_GRUPOS_DE_CLASES(8) As tungrupoclases
```

BINARIOS DE CADA CLASE

```
Public Structure TCAMPOBINARIO ' LONGItud es 18
       Public USA_ESTE_BIN As Byte ' 0/1
       Public CONTEXTO_ESTE_BIN As Byte ' 0 = NO LO MARCA
       <VBFixedString(16)> Public NOM_ESTE_BIN As String
End Structure
Public los_4_binarios_de_cada_clase(50, 4) As TCAMPOBINARIO ' no usamos el 0
```

LAS CLASES

```
Public Structure tuna_clase ' cada clase ocupa 282 bytes
 Public NUMCLASE As Byte ' 1 A 32
 <VBFixedString(24)> Public descrip_ingles As String
 <VBFixedString(24)> Public descrip_espanol As String
 Public GRUPO_CLASES_ESTA_CLASE As Byte
 '---------------------------------------
 ' PARA NUMERACIÓN DE SUS UBI'S
 '---------------------------------------
 Public INICIO_RANGO As Int32
 Public CUANTOS_NUMEROS As Int32
 Public FIN_RANGO As Int32
 Public MAYOR_UBI_ESTA_CLASE As Int32 ' EL NÚMERO DE LA UBI
 '---------------------------------------
```

```
 ' CÓMO USA LOS CAMPOS DE LA TABLA DE UBIS
 '---------------------------------------
 <VBFixedString(16)> Public nombre_fECHA As String
 Public que_es_Fecha As Byte ' se usa un código
 Public USA_ALGUN_CAMPO_BINARIO As Byte
 Public USA_ALGUN_CAMPO_GENERAL As Byte
 Public usa_campo_lit1 As Byte ' 0/1
 Public usa_campo_lit2 As Byte ' 0/1
 Public usa_campo_lit3 As Byte ' 0/1
 Public usa_campo_lit4 As Byte ' 0/1
 Public usa_campo_num As Byte ' 0/1
 <VBFixedString(16)> Public nombre_campo_lit1 As String
 <VBFixedString(16)> Public nombre_campo_lit2 As String
 <VBFixedString(16)> Public nombre_campo_lit3 As String
 <VBFixedString(16)> Public nombre_campo_lit4 As String
 <VBFixedString(16)> Public nombre_campo_num As String
 '---------------------------------------
 ' que ofrece para marcar
 '---------------------------------------
 Public cuan_grupos_contextos_ofrece As Byte
 <VBFixedString(8)> Public grupos_contextos_ofrecidos As String
 ‘hasta 4 grupos; se rellena con ceros 2 dígitos cada un8
 Public cuan_contextos_ofrece As Byte
 <VBFixedString(48)> Public contextos_ofrecidos As String
 ‘hasta 16 contextos 3 dígitos cada uno SE RELLENA CON CEROS
 <VBFixedString(16)> Public el_contexto_es_obligatorio As String
 ' un digito cada contexto paralelo al de contextos ofrecidos
 '---------------------------------------
 ' opciones
 '---------------------------------------
 Public ofrece_equivalencias As Byte
 Public idioma_default As Byte
 Public ofrece_numeracion_manual As Byte ' 0 = no, automática
 Public ufu_byte As Byte
 Public ufu_num As Int32
 <VBFixedString(16)> Public ufu_texto As String
 Public audit1 As Int32
 Public audit2 As Int32
End Structure
Public LAS_CLASES(50) As tuna_clase ' NO USAMOS LA 0
```

TIPOS DE UBI DE CADA CLASE

```
Public Structure tcatalogonumerico ' 17 bytes
 Public clave As Byte
 <VBFixedString (16)> Public descripción As String
End Structure
Public TIPOS_de_UBI(50) As tcatalogonumerico, MAYOR_TIPO_UBI_USADO As Short
'NO SE USA EL 0
Public Const TAMANO_ARREGLO_TIPOS_DE_UBI As Short = 50
```

GRUPOS DE CONTEXTOS

```
Public Structure tGruposDEContextos ' 49 BYTES CADA UNO
 'LONGITUD TOTAL DEL ARREGLO 50 * 33 = 1584 BYTES
 Public NUM_GRUPO_CONTEXTO As Byte ' SI TIENE 0 ES QUE NO SE USA
 <VBFixedString(24)> Public NOMBRE_GRUPO_INGLES As String ' 0=ENGLISH
 <VBFixedString(24)> Public NOMBRE_GRUPO_ESPANOL As String ' 0=ENGLISH
End Structure
Public LOS_GRUPOS_DE_CONTEXTOS(32) As tGruposDEContextos'NO USAMOS LA POSICION 0
```

CUANTOS CONTEXTOS HAY

```
Public cuantos_contextos_usa As Int16,
```

APUNTADORES A CONTEXTOS

```
Public los_apuntadores_a_contextos(255) As Byte
```

ARREGLO DE CONTEXTOS

```
Public Structure tUnCONTEXTO ' TIENE 176 BYTES CADA CONTEXTO
 ' tenia 176 bytes ahora tiene 206
 Public Num_contexto As Byte ' 1 A 255
 <VBFixedString(24)> Public descrip_en_ingles As String ' EN INGLÉS
 <VBFixedString(24)> Public descrip_en_espanol As String
 <VBFixedString(12)> Public abreviada_en_ingles As String ' EN INGLÉS
 <VBFixedString(12)> Public abreviada_en_espanol As String
 Public su_grupo_de_contextos As Byte
 Public tipo_contexto As Byte ' ' tabla tipos en memoria
 ‘aquí incluye si tendrá un valor único o si sus valores tienen una UBI única
 Public valores_son_numericos As Byte
 Public el_valor_unico As Int32 ' el valor único siempre debe ser numérico
 ' donde están sus valores hay un solo archivo de valores
 Public cuantos_valores_tiene As Int32 ' esto debe estar actualizado
 Public cuantos_valores_INCLLUYENDI_UFU As Int32 ' cuidado usa la posición 0 del
arreglo de valores
 Public pos_ini_en_arreglo_de_valores As Int32 '
 Public tiene_restricciones_en_cuanto_a_BDD As Byte
 Public tiene_restricciones_en_cuanto_a_CLASES As Byte
 'valores de tiene restricciones.... 0=' ==no tiene y no se pueden agregar, 1='
==no tiene pero se pueden agregar, 2=tiene
 <VBFixedString(9)> Public LO_USA_LA_BASE As String
 <VBFixedString(50)> Public SE_OFRECE_EN_LA_CLASE As String
 Public como_se_almacenan_marcas As Byte ' código, uso futuro
 <VBFixedString(16)> Public valor_no_marcable1 As String
 <VBFixedString(16)> Public valor_no_marcable2 As String
 <VBFixedString(16)> Public ufu_texto As String
 Public ufu_numerico As Int32
End Structure
Public los_contextos() As tUnCONTEXTO
```

Detalles de cómo se almacenan los Contextos en el archivo sitefile

Se usa una lista de contextos “apuntada” por otra. Se almacenan los contextos usando 2 listas (arreglos), es decir, la Alternativa 6: Arr_apuntadores_a_contextos (255). Es un arreglo de 256 elementos (del 0 al 255). Cada posición corresponde al contexto de ese número. El valor es la posición en la lista de contextos en la cual están los datos del contexto.

Arr_contextos (dimensión variable). Es un arreglo de contextos; sus elementos son instancias de la estructura de los contextos. Se agregan éstos a medida que se definen, pero no se ordena el arreglo. Para encontrar un contexto en el arreglo, se usa el arreglo de apuntadores.

Contexto buscado

NC = arr_contextos (arr_apuntadores_a_contexos (NC))

11.3 Valores de contexto

Se usa una tabla llamada VALORES de la base de datos para almacenar los valores de los contextos. El índice de la tabla está constituido por los campos Contexto y Valor. En cada valor, se indica el tamaño y la posición del arreglo de sus marcas, grabada como tal en disco.

11.4 UBIs

Base de datos Tabla UBIS, las marcas se guardan en campos memos. **C**on esta opción es posible almacenar y recuperar cada una le das UBIs de una manera correcta, así como sus marcas correspondientes, aunque aún falta una forma de recuperar las UBIs que satisfacen una búsqueda por **CONTEXTO-VALOR.**

11.5 Marcas

Para almacenar las marcas de una UBI se concatenan los pares (contexto, valor) en una cadena que se guarda en un campo *ad hoc* de la tabla de las UBIs.

Para almacenar los triplos, se adoptó la estructura Alt 2: Guardar las listas de las UBIs de un par (Contexto, valor) en un arreglo grabado en disco. Cómo se ha señalado previamente, en el registro correspondiente de la tabla “VALORES” (es decir, contexto-valor), se indica la ubicación de esta lista.

CAPÍTULO 12. CONCLUSIONES

Se cumplieron los objetivos del proyecto: Las estructuras a usar para cada entidad se seleccionaron entre varias opciones, aplicando criterios teóricos y en ocasiones, los resultados de las simulaciones efectuadas para comparar las estructuras para diversas funciones.

Se hicieron numerosos ejemplos, con diferentes números de UBIs y de marcas, y los tiempos de respuesta a consultas – con varios operandos – fueron instantáneos. Estos ejemplos también indicaron que este tipo de base de datos podría resultar útil en muchas aplicaciones.

En cuanto a investigación futura, se mejorará la determinación automática del tipo de estructuras que se usan para almacenar las marcas (contexto, valor. UBI) de un par contexto valor, con el objeto de mejorar los procesos que las usan como operandos de las fórmulas, es decir, en operaciones con otros conjuntos similares.

CAPÍTULO 13. REFERENCIAS

Bauer Mengelberg, J. R. (2005). Teaching System Access Control. *InSITE 2005: Informing Science + IT Education Conference*, *5*, 20.

Comer, D. (1979). Ubiquitous B-Tree (Vol. 11, pp. 121-137). Presentado en Computer Sctence Department, Purdue Untverstty, West Lafayette, Indiana 47907, Indiana.

Cruz Millan, M. (2003). *Desarrollo de una interfase para establecer criterios para la recuperación de información a partir del uso de claves* (Tesis Maestría Cómputo Aplicado). Colegio de Postgraduados, Montecillo, Texcoco, Edo. México.

Gonzalez Espinosa, J. (2011). *Comparación de una consulta típica de análisis de negocios usando dos estructuras diferentes de bases de datos* (Tesis Maestría Cómputo Aplicado). Colegio de Postgraduados, Montecillo, Texcoco, Edo. México.

Hernández Negrete, N. (2010). *Un sistema de gestión de palabras claves por contexto para un acervo de datos* (Tesis Maestría Cómputo Aplicado). Colegio de Postgraduados, Montecillo, Texcoco, Edo. México.

Herranz Gómez, R., & Maqueda Iglesias, A. M. (2014). *Bases de datos NoSQL: arquitectura y ejemplos de aplicación* (Tesis Ingeniería Informática). Escuela Politécnica Superior de la Universidad Carlos III de Madrid, Leganés.

Inmon, W. H. (2005). *Building the data warehouse* (4th ed). Indianapolis, Ind: Wiley.

QaysAbdulhadi, Z., Zuping, Z., & Ibrahim Housien, H. (2013). *Bitmap Index as Effective Indexing for Low Cardinality Column in Data Warehouse.* International Journal of Computer Applications, 68(24), 38-42. https://doi.org/10.5120/11730-7386

R. Bayer, & E. McCreight. (1971). *Organization and maintenance of large ordered indexes.* (In: Broy M., Denert E. (eds) Software Pioneers. Springer, Berlin, Heidelberg), 17.

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2014). *Database system concepts* (4th ed). Boston: McGraw-Hill.

Weiss, M. A. (2014). *Data structures and algorithm analyis in C++* (Fourth edition). Boston: Pearson.

ANEXOS

El texto de la tesis en formato pdf

Programas utilizados para las simulaciones

Esquema de las bases de datos

Datos para la simulación

Un ejemplo de directorio típico con datos de prueba