XIV Conferencia Latinoamericana de Informática
17avas. Jornadas Argentinas de Informática
e Investigación Operativa.
Buenos Aires, Setiembre 1988.

LOS SGBD EXTENSIBLES

H.STEFFEN*, A. GUTIERREZ, R. RUGGIA

* Laboratoire M.A.S.I. - Univ. Paris VI -
4 place Jussieu, 75252 Paris Cedex 05.
&
PE.DE.CI.BA. Casilla de Correos 1207
Montevideo - Uruguay

IN.CO. Facultad de Ingeniería
J. Herrera y Reissig 565
Montevideo - Uruguay

Resumen : *La multiplicación de nuevas necesidades y aplicaciones, sobre todo en las áreas de CAD, geografía, asistencia a la ingeniería, etc. ha puesto en evidencia las limitaciones de los SGBD basados en el modelo relacional para implantar ese tipo de aplicaciones en forma adecuada. Como respuesta a esta inadecuación una nueva clase de SGBD está apareciendo, proponiendo extensiones al modelo relacional o permitiendo la definición de nuevos modelos de datos. Esos manejadores, denominados SGBD Extensibles, permiten una representación más rica y adecuada de los datos, así como mecanismos de implantación adaptados a cada especificidad. Varios centros de investigación trabajan sobre proyectos de SGBD Extensibles y algunos prototipos están disponibles. Esta nueva área de trabajo es un punto de confluencia entre disciplinas tradicionales de las Bases de Datos (Ingeniería de sistemas) y otras como Lenguajes (Orientación Objeto y Funcionales), Inteligencia Artificial (Base de Datos Deductivas) y Concepción de Sistemas de Información (propuestas de nuevos modelos de datos).*

I - INTRODUCCION

El área Base de Datos participa actualmente a un conjunto de trabajos sobre una nueva generación de SGBD. Esta actividad está en gran parte orientada a dar respuesta a los numerosos problemas planteados por nuevas aplicaciones de tipo no tradicional, como la CAD, la gestión de documentos textuales, la concepción de VLSI, etc.

Este tipo de aplicaciones no tradicionales poseen características suficientemente diferentes de las aplicaciones clásicas de gestión como para motivar el estudio ya sea de extensiones al modelo relacional, ya sea de SGBDs de nuevo tipo.

Estas extensiones están motivadas por el hecho de que los SGBD relacionales no son capaces de resolver en forma eficiente y adecuada a la aplicación los nuevos problemas que se presentan.

El modelo relacional en sí tiene como gran ventaja su simplicidad, lo cual facilita el diseño y la manipulación. Pero en esta simplicidad tiene como desventaja el hecho de que, para la resolución de problemas cuyos elementos tienen una estructura que se aparta de las del modelo se muestra pobre en herramientas de modelización y manipulación.

Por ejemplo, supongamos un caso bastante simple y clásico para resolver en modelo relacional: Se tiene una empresa que desea llevar el registro de los empleados con datos como nombre, dirección y los nombres de sus hijos.
En modelo relacional, esto se resolvería por medio de dos esquemas de relación:

EMPLEADOS(Nombre_empleado,Dirección)
HIJOS(Nombre_empleado,Nombre_hijo)

cuando una representación más natural sería:

EMPLEADOS(Nombre_empleado,Dirección, Conjunto_hijos)

donde Conjunto_hijos es un atributo de tipo conjunto; y sus elementos son accedidos por medio de operadores sobre conjuntos.

Este problema, si bien es resuelto en forma más natural utilizando "extensiones" al modelo relacional, puede ser resuelto utilizando las herramientas "puras" del mismo. Sin embargo existen casos en que los objetos a modelizar poseen una estructura tal que se hace poco clara su representacion con modelo relacional, y difícil su manipulación por medio de los operadores relacionales.

Por ejemplo consideremos un caso en el cual deben manipularse objetos geométricos (rectas, círculos, polígonos) en el plano.
Trabajando en modelo relacional se tendría:

RECTAS(id_r,info_r)
CIRCULO(id_c,info_c)
POLIGONO(id_p,info_p)

donde cada id sería un conjunto de atributos atómicos que caracterizan el objeto.

Y se desean realizar consultas como:

- Conjunto de rectas de coeficiente angular determinado.
- Conjunto de polígonos con igual área.
- Conjunto de polígonos que se intersectan.
- etc.

Para realizar este tipo de consultas, trabajando en un modelo relacional puro, se deberán realizar programas en algún lenguaje anfitrión, ya que las condiciones que se piden no son expresables en lenguajes de consulta relacionales. En cambio si existieran definidos los tipos Recta, Círculo, Polígono, con un conjunto de operaciones asociadas (área, coeficiente angular, pertenencia de un punto a una recta o a un círculo o polígono, etc) las consultas anteriores podrían ser resueltas directamente por el lenguaje de consulta.

Por ejemplo, podría definirse:

```
CREATE TABLE RECTAS
    id_r   : Recta;
    info_r : String;
END.
```

(los esquemas CIRCULOS Y POLIGONOS se definen de forma análoga).

y expresar las consultas de la forma:

```
SELECT info_r
FROM RECTAS
WHERE coef_ang(id_r) = 30.
```

```
SELECT info_r
 FROM RECTAS 1, RECTAS 2
 WHERE coef_ang(1.id_r) =
          coef_ang(2.id_r).
```

I. 1 Limitaciones del modelo relacional.

En general, pueden considerarse un conjunto de aspectos que no son resueltos por los SGBD relacionales actuales. Dentro de este conjunto se encuentran: manipulación de grandes objetos, de versiones, definición de nuevos tipos de datos y operadores por parte del usuario, operadores más poderosos sobre conjuntos, transacciones largas y anidadas.

¿ Qué problemas se plantean con estos puntos ?

Si consideramos el aspecto de *manipulacion de grandes objetos*, vemos que existen ciertas aplicaciones que necesitan manipular objetos de gran tamaño, como documentos, imagenes, voz, programas [BERN87]; pero los SGBD relacionales permiten el almacenamiento (y manipulación) de objetos de tamaño chico (artículos, tuplas).

Actualmente los SGBD relacionales resuelven este problema descomponiendo los grandes objetos en partes más chicas. Como consecuencia de esto, su manipulación exige un trabajo de reconstrucción que significa una pérdida importante de performance, y dificulta la expresión de las consultas.

Por otro lado los SGBD relacionales ofrecen una cantidad muy limitada de alternativas de almacenamiento y métodos de acceso, lo cual impide la optimización del espacio de almacenamiento y del tiempo utilizado en manipulación mediante el uso de estructuras y métodos de acceso más adaptados a las características de los objetos.

Considerando el aspecto de *mantenimiento de versiones* de un mismo objeto, se puede observar que esta facilidad es fundamental para ciertas aplicaciones como CAD, gestión de documentos, de programas,etc. [ADIB86] [DAYA85] [KATZ86]. Sin embargo las características del manejo de versiones cambia según la aplicación y parece interesante el ofrecer mecanismos parametrables.

Otro tema clave es el del *manejo de tipos*. En general, casi todas las aplicaciones - tradicionales o no - necesitan manejar objetos de tipos no atómicos (conjuntos, tipos estructurados, tipos recursivos, grafos), o incluso necesitan que sea el propio usuario quien se defina sus propios tipos. En ciertos casos puede ser interesante contar con la posibilidad de representar datos por medio de las consultas (SQL por ejemplo) que permiten calcularlos [STON84a] [ZANI83].
Por el contrario, los tipos de datos en los SGBD relacionales se limitan en general a artículos (no recursivos), formados por campos de tipos atómicos.

Si consideramos el tema de las *formas de acceso*, observamos que una posibilidad que tienen los SGBD actuales es la de acceder a los datos basandose en su contenido (acceso asociativo). Esta facilidad es muy útil en gran cantidad de aplicaciones; y en los últimos años se ha trabajado mucho en el desarrollo de nuevas estructuras y nuevos métodos más eficientes para el acceso asociativo. [FAG79, LARS80, LITW80, LOME83, LITW87, LOME87, BENT75, BENT79, NIEV84, ROBI81, SCHE82]. Sin embargo, pocos de estos mecanismos se encuentran incorporados en SGBD comerciales.[BERN87]

Otra posibilidad es la de poder acceder a los datos, no según su valor sino por su identidad. Por ejemplo, sean dos empleados que tienen un hijo con igual nombre. ¿ Se trata del mismo individuo o de individuos distintos pero iguales en su representación en la base de datos ? Manejar el concepto de identidad entre objetos permite distinguir estos casos.

Una caracteristica de los operadores relacionales es su calidad de operadores sobre conjuntos de tuplas. Podría ser interesante también tener operadores que actúen sobre conjuntos de otros tipos de objetos (por ejemplo que actuen sobre grafos). Es más, ser ía deseable que fuera el usuario quien agregara sus propios operadores a su medida. Por ejemplo, por medio de la definición de nuevas funciones agregadas del tipo PROMEDIO, SUMA, MEDIA, SEGUNDO_MAYOR, etc. [STON88]

Estas limitaciones de los SGBD Relacionales frente a loas actuales y sobre todo a las nuevas aplicaciones a impulsado el desarrollo de numerosas propuestas de mejora y de extensión de las funcionalidades de los SGBD, dando nacimiento así a los SGBD Extensibles.

II - CARACTERISTICAS DE LOS SGBD EXTENSIBLES

La denominación SGBD Extensible es utilizada para designar sistemas con objetivos y características diferentes, abarcando un espectro muy amplio.
En la primer parte de este capítulo estudiaremos los principales aspectos sobre los que puede aplicarse la extensibilidad de los sistemas, y en la segunda parte veremos los distintos enfoques arquitectónicos que pueden ser propuestos.

II.1 - EXTENSIBILIDAD

La extensibilidad de los sistemas puede ser vista como dando respuesta en los siguientes aspectos :

1) Enriquecer la representación externa y la manipulación de los datos de los usuarios.
2) Permitir la definición de métodos específicos de almacenamiento y de acceso a los datos.
3) Ofrecer nuevas funcionalidades para facilitar la implantación de nuevos operadores .

II.1.1 - Enriquecer la representación

Actualmente el diseño de una aplicación se realiza en (por lo menos) dos niveles diferentes : el nivel abstracto (utilizando la especificación y modelos semánticos) y un nivel conceptual del esquema de la Base de Datos (utilizando el modelo relacional enriquecido con restricciones de integridad y dependencias funcionales).

Tal como hemos visto, una primer dificultad de este enfoque es que el modelo relacional es pobre para representar directamente objetos complejos. Una alternativa es dar más poder de expresión a los modelos de datos a través de tipos primitivos más potentes.

Otro inconveniente es que la representación estructural de los datos está separada de su manipulación. Mientras que la estructura de los datos es conocida por el SGBD a partir del esquema de la BD, los programas de aplicación son externos al mismo.

Una solución interesante es de autorizar la definición de Tipos Abstractos de Datos (ADT) a nivel mismo del SGBD. De esta forma la representación de los datos se aproxima o se confunde incluso con la modelización semántica.

La definición de ADT permite la descripción de la estructura de los datos a un nivel arbitrario de complejidad, así como del conjunto de rutinas de manipulación de esos datos (rutinas que en el enfoque orientado objeto llamamos métodos).
Los ADT soportan la noción de encapsulación, de forma que el único acceso autorizado a los datos se realiza a partir de las rutinas definidas en el ADT. En este caso la descripción estructural y comportamental de los datos es hecha de forma integral y al mismo tiempo.

Relacionada con los tipos de datos, se encuentra la idea de **integridad**. Referente a la misma el SGBD debe chequear que los objetos se ajusten a sus definiciones de tipo, o cumplan ciertas propiedades.

Por otro lado, los objetos de la base de datos pueden tener restricciones de integridad que son más complejas que las que pueden ser expresadas en el lenguaje de definición de tipos. Por ejemplo, chequear que un documento sea consistente con el estándar de una organización. [BERN87].

Otra noción importante para enriquecer la representación y la manipulación de los datos es la de **identidad de los objetos**.
Los sistemas relacionales están basados sobre el valor de los objetos. Los tratamientos son de tipo de conjunto sobre el contenido de los objetos (tuplas). Este tipo de orientación no es suficiente para modelizar situaciones más complejas y poder realizar tratamientos adecuados.
Así es introducida la noción de identidad de los objetos, lo que nos permite :

- crear la noción de **referencia** sobre un objeto, o identificador de objeto (OID).
  Un OID es un valor externo al objeto que lo desgna en forma única e invariable.
- poder comparar dos referencias a objetos y saber si se trata del mismo o de diferentes objetos.
- poder construir objetos estructurados a partir de otros objetos, utilizando los identifiacdores (OID) de éstos en lugar de copiar su contenido. Esto nos evita las copias redundantes y las puestas al día múltiples.
- Realizar asociaciones explícitas entre objetos.

Con la noción de identidad aparece un nuevo operador: IDENTICO(x,y) que toma un valor verdadero si x e y referencian al mismo objeto.

Implementar los ADT en un SGBD

Ofrecer la definición de ADT presenta un conjunto de problemas a resolver por los SGBD extensibles :

- integrar un lenguaje de definición de ADT interactivo
- permitir la programación de los métodos asociados
- estudiar mecanismos de almacenaje adecuados
- implementar las nociones de identidad y de igualdad de objetos

Una primera solución de conjunto consiste en dividir la gestión de los ADT en dos partes : una a cargo del SGBD y otra a cargo del usuario. EL usuario define nuevos tipos, caracterizándolos únicamente por su nombre y su almacenaje físico (largo en octetos, fijo o variable). El SGBD memoriza esta definición y se encarga del almacenaje y del acceso a los objetos de dicho tipo.
En cuanto a la definición de métodos, el usuario indica al SGBD el nombre y los tipos de los parámetros formales, así como el archivo donde el programa (fuente o código) de dichos métodos han sido almacenados.
El código del SGBD no es modificado ya que los nuevos métodos son externos.

La invocación a un método identificable hace que el SGBD lo cargue a partir del archivo y los ejecute sobre los objetos designados.
En este enfoque el control de la estructura interna del nuevo tipo está enteramente a cargo del usuario.

Esta solución es la más simple y puede implementarse a partir de un SGBD clásico, al que se le agregan las funcionalidades descriptas.

Una segunda solución es de proponer un enfoque integrado, en el cual los nuevos tipos estén completamente bajo control del SGBD. En este caso los nuevos métodos pasan a formar parte del SGBD. Esta última solución permite una gestión uniforme de las funciones - sistema y usuario-, un mejor control de coherencia así como permite tener en cuenta el costo de ejecución de las funciones para poder realizar la optimización de operaciones y consultas.

La implementación de esta solución presenta dos grandes problemas :
1) estudiar métodos de almacenaje capaces de soportar objetos complejos
2) estudiar un mecanismo de integración del código de los métodos.

1) almacenar objetos complejos :
Si vemos el almacenaje en el primer enfoque (no integrado), la solución es simple, puesto que los objetos son vistos por el SGBD como una simple cadena de caracteres (cuyo largo es limitado por parámetros del sistema). La estructura interna de los objetos es conocida únicamente por el usuario.
En este segundo enfoque, el SGBD debe soportar el almacenaje de estructuras complejas de largo variable y que pueden alcanzar dimensiones muy importantes. Una simple gestión de los objetos por contigüidad no es siempre posible ni deseable (costo de almacenaje, de modificación).
El SGBD debe conocer la estructura interna de esos objetos y ser capaz de asegurar la correspondencia entre representación externa (contigua) --> implantación física (paginada).

2) integrar nuevos métodos
- fabricar un nuevo código del SGBD a partir del viejo y de los métodos agregados. Esto se hace por edición de lazos estática.
Este mecanismo permite una integración total de las rutinas pero es pesado y costoso, poco adaptado a un trabajo interactivo con el usuario.

- mantener separados el código del SGBD y el de los métodos, utilizando una librería de métodos y una edición de lazos dinámica. Este mecanismo es más costozao en tiempo de ejecución y exige un editor de lazos dinámico integrado al SGBD.

Las soluciones propuestas van a optar por el primer caso cuando la definición de ADTs sea realizada en función de adaptar un SGBD a una aplicación dada, y luego no se vuelve a modificar (más allá del mantenimiento del sofware). Por el contrario, si la definición de nuevos ADTs es frecuente, la segunda solución parece imprescindible.

II.1.2 - Métodos de acceso y de almacenaje

La posiblidad de definir nuevos mecanismos de acceso y de almacenaje de datos tiene como objetivo adaptar estos mecanismos a las características específicas de las aplicaciones. En efecto, los métodos de acceso (índices, hash, secuencial) y de almacenaje (relaciones en archivos Vsam, hash, secuenciales) propuestos en forma estándar por los SGBD pueden no ser una buena solución para ciertas aplicaciones. Por ejemplo el acceso a objetos espaciales - figuras geométricas etc.- es optimizado con una representación basada en R-Tree [Gutt 84] y KDB-Tree [Robi 81], o de tipo Grid-files [Niev 84].
De la misma forma, la definición de ADTs puede apoyarse en tipos de base existentes (el almacenaje es hecho por el SGBD) o puede ser completamente redefinida por el usuario. Un ejemplo de esto es el ADT Matriz. Una forma de representarla es a partir de una tabla de tablas - tabla a una dimensión sería un tipo existente en el SGBD -.
Esta definición es rápida y simple, pero ineficaz, puesto que un objeto ´matriz´ puede estar físicamente disperso entre las diferentes tablas que la componen.

Una solución a este problema es de definir un ADT a partir de macanismo de bajo nivel, sin utilizar lista o tabla, permitiéndonos reagrupar físicamente cada objeto ´matriz´ .

Implementar nuevos métodos de acceso

Para autorizar la implantación de nuevos métodos de acceso los SGBD Extensibles deben ofrecer funcionalidades de gestión de la memoria central y secundaria del tipo acceso-a-pagina. El usuario utiliza estas primitivas de bajo nivel y contruye su propia estructura de datos. Estos mecanismos de bajo nivel deben incluir funcionalidades de servicio como control de concurrencia, recuperación, buffer.

II.1.3 - Nuevas funcionalidades

La diversidad de nuevas aplicaciones exige de los SGBD Extensibles la posiblidad de implantar nuevos operadores y de emplear posibilidades del sistema .
Estas funcionalidades pueden ser estudiadas en las siguientes áreas :

1 - orientadas a aplicaciones de tipo SGBD Deductivo.
2 - gestión de versiones históricas.
3 - gestión de transacciones largas.
4 - varios tipos de gestión de la concurrencia.
5 - estrategias de buffer adaptables.

Orientadas a la deducción

Un SGBD Deductivo necesita mecanismos adaptados al almacenaje y acceso rápido a las reglas [Gard87], así como estructuras de datos que autoricen el empleo de algoritmos de cálculo deductivo.

El almacenaje de reglas necesita estructuras de datos adaptadas, del tipo listas, conjuntos, tablas dinámicas, tuplas. Sólo éstas últimas forman parte de los SGBD tradicionales.
De la misma forma es importante disponer de índices adaptados a la búsqueda de reglas cuando su cantidad comienza a ser importante .

La ejecución de una regla necesita determinar la definición de todos sus predicados, que pueden a su vez depender de otros predicados derivados. Encontrar recursivamente todos los predicados a partir de una regla se resuelve de forma fácil y eficaz por un cierre transitivo de un grafo, donde cada nodo representa un predicado - derivado o de base -.
De forma análoga, el cálculo de reglas lineales recursivas puede ser realizado eficazmente por un operador de punto fijo basado en un cierre transitivo de un grafo [Puch87][Gard 87b].
De esta forma un constructor ´grafo´se presenta como un útil de gran interés en un contexto deductivo.

Versiones

Como veíamos en el capítulo I, ciertas aplicaciones necesitan mantener versiones históricas de sus objetos. Esta gestión debe permitir una utilización simple y rápida de un objeto de una versión dada. Los dos principales parámetros a tener en cuenta son 1) copiar la cantidad mínima posible del objeto tal que obtengamos dos versiones completas del mismo, y 2) estudiar mecanismos de reconstrucción rápida de una versión a partir de todas sus partes.

Transacciones

El manejo de transacciones es un tema importante en el desarrollo de nuevas aplicaciones. En los SGBD relacionales de hoy, los mecanismos de manejo de transacciones están encarados hacia el manejo de transacciones de duración corta mientras que en nuevas aplicaciones (CAD/CAM, VLSI) es común la existencia de transacciones que duran varios días. En esta situación, no sería aceptable el hecho de que se perdiera todo el trabajo por una falla del sistema. Por esto surge la necesidad de contar con transacciones de larga duración que permitan evitar este tipo de problemas [Born 87].

Otra forma de transacciones que podría desearse manejar son transacciones anidadas. Es decir que dentro de una transacción se lanza otra, y la posible falla de la última no implica la falla de la primera.

Concurrencia y recuperación

Los SGBD clásicos proponen mecanismos de concurrencia basados en un bloqueo en dos faces - con gránulo fijo o variable -. Ciertas aplicaciones - ingeniería, texto - requieren un mecanismo de concurrencia que tenga en cuenta las versiones de los objetos [Katz 83]. Otras como las aplicaciones sobre bases de imágenes - acceso únicamente en lectura - pueden prescindir de un control de concurrencia y de la recuperación.
Otra característica de la concurrencia es su carácter inmediato o diferido. El mecanismo diferido consiste en bloquear únicamente los objetos derivados del resultado obtenido, y no todos los objetos de forma sistemática a medida que se accede a ellos - ésto sólo es posible si las consultas son ejecutadas en forma atómica -.
Si las aplicaciones clásicas utilizan un mecanismo inmediato, la implantación de un mecanismo diferido de control de la concurrencia permitiría reducir su costo de forma significativa .
Implementar este mecanismo en un SGBD implica que el manejador sea capaz de encontrar todas las tuplas que permitieron contruir el resultado, para así bloquearlas al final del tratamiento.

Estrategias de buffer adaptadas

Para obtener una buena utilización de la disponibilidad en memoria central y reducir el costo en E/S, los SGBD realizan una gestión de los datos en ante-memoria -buffer - [Chou 85].
Si tradicionalmente esta gestión es uniforme para todos los tipos de datos, parece interesante incorporar nuevas técnicas de manejo de buffer dedicadas a cada tipo de datos, debido a que las operaciones asociadas a los mismos pueden utilizar distintas formas de acceso.

II.2 - PRINCIPALES ENFOQUES DE ARQUITECTURA

Recientemente varios prototipos de sistemas extensibles vienen siendo presentados. Sus objetivos y características varían en un espectro muy amplio. En particular podemos distinguir tres grandes enfoques : los ´sistemas completos´, los ´sistemas para armar´, y los ´sistemas orientados objeto´.

Sistemas Completos

Los Sistemas Completos están dirigidos directamente a los usuarios finales. Son productos sofware completos y monolíticos, no modificables, que ofrecen numerosos mecanismos de extensibilidad.
Estos SGBDs están basados en un modelo de datos bien definido y fijo (relacional extendido, modelos funcionales , etc.) . El usuario debe adaptar su visión de los datos al modelo propuesto por el SGBD.

La extensibilidad de los sistemas completos está sobre todo orientada a autorizar la definición de nuevos ADTs por los usuarios. En algunos casos la extensibilidad está también dirigida a mejorar las performances del sistema, permitiendo la definición de nuevos métodos de almacenaje y/o de acceso a los datos.

Sistemas para armar

El enfoque de los ´sistemas para armar´ consiste en proponer un esqueleto de base de un SGBD, compuesto de módulos y de librerías de funciones de base. Estos sistemas permiten el prototipaje y la fabricación rápida de SGBD específicos, bien adaptados a las aplicaciones que deben utilizarlos.

Los sistemas para armar no están dirigidos a los usuarios finales sino a un especialista (DBI -Data Base Implementor-) que debe diseñar un SGBD a partir del conocimiento de la aplicación a la cual sera destinado.
A diferencia del caso anterior, estos sistemas no poseen un modelo de datos fijo al que los usuarios deban adaptarse.
Estos sistemas deben proponer un conjunto de módulos y herramientas para facilitar el trabajo de prototipage del nuevo SGBD. Dentro de esos módulos, un núcleo de base de gestión del almacenaje de objetos parece como indispensable. A éste se le pueden agregar otros como librerías (índices B-tree, operadores clásicos, etc). Otro elemento interesante son los generadores: programas que permiten obtener un módulo del SGBD a partir de ciertos parámetros. El caso más claro es el generador del módulo de optimización de consultas.
Las herramientas de ayuda al prototipage pueden ser lenguajes de alto nivel, útiles gráficos, etc.

Sistemas Orientados Objeto

Los SGBD orientados objeto presentan un intento reciente de asociar a un SGBD las características de un lenguaje orientado objeto (Smalltalk, C++, Objectif C)[Banc 88].
Estos sistemas son naturalmente extensibles a nivel de nuevos tipos usuarios, definidos en el lenguaje orientado objeto. Un SGBD OO puede comportarse como un sistema completo (basado en un modelo de datos orientado a objeto) o como un sistema para armar, permitiendo el prototipage rápido de nuevos SBBD específicos.

III - ALGUNOS EJEMPLOS DE SGBD EXTENSIBLES

Para completar el estudio sobre la extensibilidad presentaremos seis prototipos de SGBD extensibles. Estos sistemas son a nuestro entender los casos más importantes y originales. La diversidad en sus enfoques y en las soluciones propuestas es ilustrativa del estado actual del área.

III.1 POSTGRES.

POSTGRES [STON86] [STON88], es un SGBD sucesor de INGRES que esta siendo desarrollado en la Univesidad de California, Berkeley.
Es un sistema completo basado en el modelo relacional.
En este proyecto la extensibilidad esta dada en cinco áreas : tipos de datos, funciones, operadores, agregados y metodos de acceso.

Tipos de datos.

POSTGRES además de proveer tipos primitivos como enteros y cadena de caracteres, permite definir nuevos tipos de datos de base, tablas de tipos base, tipos procedimiento y tipos procedimiento parametrizado por parte del usuario. Dichas extensiones son detalladas en [ROWE87].Una caracteristica a destacar de esta facilidad es que la representación del ADT es dejada al usuario.
Para la definición de nuevos tipos de datos de base se especifica nombre del tipo,largo en octetos (puede ser variable), el procedimiento que hace la correspondencia entre la representación externa (ASCII) y la representacion interna, y el procedimiento que hace la correspondencia en forma inversa.Por ejemplo, si se desea definir el ADT BOX :

```
define type BOX is
    ( Internal lenght = 16,          /* Largo en octetos.
      Input proc = CHARTOBOX,        /* Procedimientos que hacen la correspondencia de
      Output proc = BOXTOCHAR,       /* la representación externa a la interna y viceversa.
      Filename = 'BOXCODE',          /* Nombre del archivo donde se encuentra el fuente
                                     /* de los procedimientos anteriores.
      Default = '')                  /* Valor por defecto.
```

Con el ADT BOX definido anteriormente el usuario puede entonces crear relaciones cuyos atributos sean de dicho ADT. Por ejemplo, si el usuario desea crear la tabla que relaciona los 'BOXs' con su identificación :

```
create IDENTBOX ( ID = integer, DESCRIPCION = box )
```

Funciones.

Permite extender el lenguaje de consulta por medio de nuevas funciones definidas por el usuario.

La forma de definir funciones por el usuario consiste en especificar el nombre de la misma y los parametros, el tipo del resultado, el lenguaje fuente en que está escrito el procedimiento que implementa la función, y el nombre del archivo en que se encuentra dicha función.Por ejemplo, si el usuario desea definir una función que calcule el área de una BOX :

```
define procedure BOXAREA ( BOX )
 return int4 is                          /* int4 es el tipo del resultado.
 ( Language = 'C',                       /* lenguaje en que esta escrito el procedimiento.
   Filename = 'BOXCODE')                 /* nombre del archivo donde se encuentra el
                                            procedimiento.
```

La idea en cuanto a la implementación de esta facilidad de extensión es que POSTGRES acepte el fuente , lo compile almacenandolo en una biblioteca del sistema y que a la hora de la ejecución la función sea cargada dinámicamente, de tal forma que luego quede en memoria. Además POSTGRES hace el chequeo automático de tipos de los parámetros de la función.

Operadores definidos por el usuario.

Los operadores se diferencian de las funciones en que tienen un número limitado de parametros y en que POSTGRES intenta optimizar la ejecución de las consultas que incluyen operadores, utilizando la información definida en los mismos.

Para definir un operador de debe dar el nombre, el procedimiento asociado, el nivel de precedencia (utilizable en caso de aparecer en expresiones sin paréntesis), propiedades del operador (asociatividad, conmutatividad, negación) e información para el optimizador de consultas.

En forma similar es posible definir nuevos operadores agregados.

Métodos de acceso.

No se trata de dar una forma sencilla de extensibilidad ya que no encaran la extensibilidad en este punto hacia usuarios finales, sino hacia programadores con experiencia.

Los métodos de acceso posibles a extender son solo indices secundarios, es decir conjunto de claves con puntero a artículos.
Cuando se extienden los métodos de acceso, se puede usar el manejador de buffer del POSTGRES. Si es asi, se debe llamar a las rutinas del POSTGRES para leer y escribir páginas en el pool de buffers y para agregar y retirar páginas del buffer.

El implementador debe manejar el bloqueo para lograr mecanismos de concurrencia; pudiendo usar un mecanismo de POSTGRES o uno propio. También puede usar funciones de recuperación del POSTGRES.

Estado actual del proyecto y perspectivas.

El prototipo cuenta actualmente con un conjunto de metodos de acceso y permite definir tipos de datos y operadores por parte del usuario. El parser y el optimizador de consultas se encuentran operativos. Los operadores deben ser precargados, por no encontrarse pronto el loader dinámico. Las funciones y los agredados definidos por el usuario todavia no se encuentran operativos.

III.2 STARBURST

El proyecto Starburst [McPh 87] es desarrollado en el IBM Almaden Research Center. Starburst es un sitema completo basado en el modelo relacional extendido.
Sus principales objetivos son :

- extensibilidad de tipos usuarios
- performance
- portabilidad
- datos y funciones distribuidas

La arquitectura del SGBD está organizada en dos sub-sistemas llamdos Core y Corona. Core permite el acceso tupla a tupla de las relaciones. Este subsitema se encarga de la gestión del almacenaje, de los índices, de los mecanismos de buffering, así como de la gestión de transacciones, de realizar controles de integridad y del mecanimos de triggers.

Corona asegura la interface con los usuarios. Este subsistema se ocupa del analizador de consultas, de descomponer y optimizar las consultas así como de generar un plan de acceso. También mantiene un catálogo y controla las autorizaciones.
Core y Corona contituyen una arquitectura similar a la del System R [Astr76], compuesto de los dos subsitemas RSS y RDS.

Extensibilidad

La extensibilidad en Starburst se sitúa a dos niveles : 1) definición de nuevos tipos por los usuarios bajo forma de ADTs. y 2) definir nuevos mecanimos de almacenaje y de acceso a las relaciones.

Definición de nuevos tipos

Starburst se propone autorizar la definición de nuevos tipos de datos para representar objetos complejos. Su enfoque es similar a XSQL [Hask 83] [Lori 83], donde un objeto complejo es una colección de componentes heterogéneos, represntados por tuplas. El almacenaje de esas tuplas es independiente (clustering posible) y su estructuración como objeto complejo se realiza por composición utilizando predicados y claves externas.
Este enfoque permite mantener un almacenaje en relaciones normalizadas, así como la declaración de vistas o de reorganizaciones de los objetos complejos.
Para manipular los datos se proponen realizar extensiones a SQL, así como operaciones para el recorrido de la estructura de los objetos complejos.

Nuevos mecanismos de almacenaje

En este plano, Starburst propone un conjunto de mecanismos integrados, denominado **attachment**. Estos incluyen los métodos de acceso, las restricciones de integridad y los triggers. La idea de base consiste en asociar a cada relación un conjunto de estructuras de datos o de acciones a realizar automáticamente cuando la relación es puesta al día. Por ejemplo, a un atributo de una relación podemos asociarle un índice, un índice de join, una cierta restricción de integridad, etc.

La extensibilidad prevee la posibilidad de agragar nuevos mecanismos de attachment, integrándolos a una estructura de datos sistema donde el conjunto de esas descripciones es mantenido.

Estado actual y perspectivas

El sistema está en curso de desarrollo. Han estudiado algunas modificaciones a SQL de forma de soportar las extensiones propuestas. Estos mecanismos están siendo implementados basándose en la idea de un acoplamiento fuerte de las operaciones y las facilidades de servicio ofrecidad por el sistema. También trabajan sobre la defínción de una nueva interface usuario priorizando la facilidad de utilización y de diálogo.

III.3 PROBE.

PROBE [DAYA86] [GOLD87] es un SGBD extensible de tipo completo basado en un modelo funcional, que está siendo desarrollado por la CCA (Computer Corporation of America).

El objetivo de este proyecto es desarrollar un SGBD avanzado que pueda manejar mejor los tipos de información y el conocimiento intensional de las nuevas áreas de aplicación. Para lograr esto propone aumentar los SGBD existentes con:

- Clases de objetos, como base para definir nuevos objetos y operadores.
- Conceptos dimensionales (espacio,tiempo).
- Predicados y consultas recursivas.

Al diseñar PROBE, se estableció la división entre el núcleo del SGBD (PROBE) y las clases de objetos. PROBE provee operaciones a partir del modelos de datos; por ejemplo, selección y proyección que tratan sobre concuntos de objetos de tipo genérico. Mientras que las operaciones específicas sobre objetos específicos son implementadas con las clases de objetos definidas por los usuarios.
Para incorporar nuevas clases de objetos, se hace uso de un mecanismo llamado **adapter**. Los adapter son rutinas utilizadas para compatibilizar los dos puntos de vista con que una función en PROBE es manejada, que son : desde PROBE y desde la clase del objeto. [GOLD87] Entre las tareas que conciernen al adapter se mencionan : conversión de lista de parámetros, conversión entre tuplas/relaciones y los datos de los lenguajes de programación (y viseversa), manipulación de valores nulos, ofrecer acceso a funciones agregadas.

Con respecto al problema de la **información dimensional**, se propone proveer un soporte directo del SGBD para dicha información, con el objetivo de implementar aplicaciones espaciales con una representación adecuada, y realizar operaciones eficiententemente sobre grupos u objetos espaciales individuales. PROBE propone como solución clases de objetos generales; por ejemplo la clase PTSET (conjunto de puntos), y que la implementación de los objetos sea determinada según la aplicación.

PROBE propone la utilización de la **recursividad** en la definición de vistas y de consultas, para aumentar el poder de expresión del lenguaje de manipulación. Para esto generaliza la clausura transitiva al punto de poder resolver muchas aplicaciones importantes sin sacrificar oportunidades para el procedimiento efectivo sobre grandes bases de datos. Para estos casos especiales se explota y adapta muchos algoritmos (bien conocidos) sobre grafos.

Modelo de datos de PROBE (PDM).

Los constructores de modelación del PDM son las entidades y funciones. Una entidad es usada para representar un objeto del mundo real, y puede ser miembro de uno o más tipos, estando

los tipos organizados en una jerarquía generalizada. Las funciones son usadas para representar propiedades de las entidades. Las entidades y funciones son manipuladas por el **algebra PDM**, que se diferencia del álgebra relacional en que permite la manipulación de entidades de tipos arbitrarios, soporta funciones computadas y consultas espaciales y recursivas.

Perspectivas.

Han identificado puntos de investigación concernientes a SGBD de proposito general como ser:

- Proceso de conocimiento operacional. Este tipo de conocimiento es habitualmente expresado en la forma situación-acción. (por ejemplo triggers).
- Diseño de aplicaciones. Como se distribuye el conocimiento de una aplicación en un SGBD.
- Interfaces.
- Manejo de transacciones.

III.4 EXODUS.

El proyecto EXODUS [CARE87] es desarrollado en la Universidad de Wisconsin, Madison. Es un ´sistema para armar´ donde el objetivo es permitir el desarrollo rápido de un SGBD específico a la aplicación y con alta performance.

La idea en cuanto al diseño de EXODUS es la de proveer por un lado un conjunto de facilidades de núcleo (kernel) para uso en todas las aplicaciones y por otro un conjunto de herramientas para ayudar al DBI a generar un nuevo software de sistema de base de datos específico a la aplicación a resolver.

Entre las componentes fijas que constituyen el núcleo de EXODUS se encuentran: el manejador de objetos almacenables, y el manejador de tipos.

Manejador de objetos almacenables.

El manejador de objetos almacenables es el encargado de manipular todo lo concerniente a los objetos almacenados en disco, para lo cual provee:

- Objetos para almacenar datos y archivos para agrupar logicamente y fisicamente objetos almacenables.
Los objetos almacenados como una cadena de octetos no tipados, pueden ser tan grandes como se quiera. Un objeto se identifica por un OID, que es su dirección virtual. El archivo es un conjunto no ordenado de objetos relacionados, y es útil para proveer un mecanismo de recorrido secuencial y permitir que los objetos puedan ser agrupados físicamente en el disco.

- Manejador de ´buffers´, que maneja parte de objetos de largo variable.

- Soporte para el manejo de versiones.

- Primitivas para manejo de concurrencia y servicio de recuperación.
Para resolver el problema de concurrencia utiliza un mecanismo de bloqueo en dos fases con opción de bloquear el objeto entero o parte de él. En cuanto a recuperación se utiliza un mecanismo de archivo diario (log) de imagenes de antes de la modificación.

Manejador de tipos.

El manejador de tipos es un depósito de información relacionada con los tipos persistentes. Ejemplo: existen dependencias entre los módulos que constituyen definiciones de tipos y módulos

que usan dichas definiciones, dependencias entre archivos del usuario y consultas compiladas que acceden a ellos.Para manejar estas dependencias el manejador de tipos construye un grafo de dependencias, por medio del cual mantiene actualizada dicha información.

Entre las herramientas provistas por EXODUS se encuentran el lenguaje de programación E, una biblioteca de métodos de acceso independientes del tipo y los métodos de los operadores de los modelos de datos, generadores de componentes a partir de una especificación.

Lenguaje de programación E.

El lenguaje E es una extensión del lenguaje C++. Fue diseñado para ser utilizado como herramienta para programar los métodos de acceso, operadores del modelo de datos y funciones utilitarias, permitiendo así al DBI lograr SGBD específicos a una aplicación.

El trabajar en E elimina o resuelve parcialmente los siguientes problemas:

- El DBI debe manejar el pasaje de memoria a disco debido a que todos los datos de interés están en disco, y que lo directamente almacenado no es tipado. Por otro lado, cuando se escriben los métodos de acceso se deben resolver los problemas de concurrencia y recuperación.

- Gran parte de la información de tipos es desconocida por el DBI en el momento de implementación del SGBD. Por ejemplo, con respecto a los métodos de acceso, el DBI al escribir código para manejar índices, debe escribir el mismo de tal forma que pueda ser utilizado aplicado con claves de tipos diferentes. Este problema no se plantea sólo con los métodos de acceso. En el momento de escribir el código que implemente las operaciones a nivel de modelo de datos (selección, proyección, etc, del modelo relacional), el DBI puede asumir muy poco de los tipos de datos con los que tratará.

Para resolver dichos problemas el E provee las herramientas de generadores de clases y persistencia de clases en disco.

Clases, dbclases y generadores de clases.

Uno de los objetivos que EXODUS pretende lograr es la posibilidad de agregar nuevos ADT. El mecanismo que propone para hacer posible ésto, es hacer uso del concepto de **class** del C++ (en el sentido de clase en el enfoque objeto).

EXODUS además provee, a través del lènguaje E, una facilidad que es la de **dbclass**. Una **dbclass** es una extensión del mecanismo básico de clases diseñado para almacenamiento en memoria externa.

E divide el mundo de las clases en dos tipos: aquellos objetos que pueden existir en memoria volátil y aquellos cuyos objetos pueden existir en memoria o en disco. Las primeras son simplemente las clases del C++, y las otras son las dbclass. Las dbclass, sintácticamente, se declaran en forma similar con la restricción de que todos los datos que forman parte de la dbclass deben ser a su vez objetos de una dbclass.

A su vez el lenguaje E provee el mecanismo de **generador de clases**. Este mecanismo permite la definición de clases parametrizables en función de uno o más tipos.
Por ejemplo, para definir un nodo de un B-tree, escribimos la siguiente declaración:

dbclass BTNODE [<dbclass{} KEYTYPE; dbclass{} ENTTYPE >]

Dentro de BTNODE, KEYTYPE y ENTTYPE, son usados como cualquier otro nombre de tipo.
Dada una definición de generador de clase, se pueden crear nuevas clases dándole los parámetros adecuados. Este proceso es llamado **instanciación de una clase**.
Por ejemplo: BTNODE[<dbint, dbint >], donde dbint es una dbclass predefinida.
Además el generador de clases puede ser instanciado con funciones parametrizadas, lo cual le brinda al DBI una solución para los casos en que necesita diferentes rutinas de comparación según el tipo de la clave.

Persistencia en E.

En el lenguaje E, los archivos son introducidos como una nueva clase propia, donde, de acuerdo con el sistema de almacenamiento subyacente, un archivo es un conjunto no ordenado de objetos. Un archivo dado puede contener solo un tipo de objetos, y dicho tipo debe ser una dbclass. La forma de declarar un archivo persistente en disco es:

persistent fileof [<alguna_dbclass>] f ;

Esta declaración es sólo aplicable a objetos de clase fileof y en este ejemplo significa que f es asociado con el mismo archivo físico en toda la ejecución del programa.

Las operaciones sobre archivos están definidas simplemente como funciones sobre la clase archivo, y el acceso a los objetos en el archivo tiene la forma de operaciones sobre punteros (de dbclass). Por ejemplo, hay una operación que retorna un puntero al primer objeto del archivo, una que crea un nuevo objeto en un archivo, etc.

Métodos de acceso independientes de tipo y métodos de los operadores.

Para los métodos de acceso, EXODUS provee una biblioteca de estructuras de índice independiente del tipo, tales como B-tree, grid-files [NIEV84]y hash lineal [LITW80].
Además un DBI puede agregar nuevos métodos de acceso que considere adecuados a la aplicación específica haciendo uso del lenguaje E.

En cuanto a los métodos de los operadores, EXODUS provee los mas usuales, estando disponibles en la biblioteca de métodos independientes de tipos.
Los métodos para operadores más específicos a la aplicación deberán ser provistos por el DBI haciendo uso del lenguaje E.

Generadores de componentes a partir de una especificación del DBI.

EXODUS provee un **generador** para producir un **optimizador de consultas** para lenguajes de consultas algebraicos.[GRAE87]

Finalmente, si bien EXODUS no hace referencia a ningún modelo de datos en particular, sugiere un modelo de datos y un lenguaje de manipulación llamados EXTRA y EXCESS [CARE88].
Como resumen sus autores en [CARE 88], el modelo de datos EXTRA incluye un soporte para objetos complejos con subobjetos compartidos, una mezcla de semántica de datos orientada a valor y orientada a objeto, un soporte de ADT definido por los usuarios.Mientras que el lenguaje de manipulación EXCESS provee facilidades para consultar y actualizar objetos complejos. Además puede ser extendido agregando: funciones y operadores de los ADT.

Estado actual y perspectivas.

El estado actual[CARE87] del proyecto es el siguiente: el generador de optimizadores se encuentra operativo y bajo evaluación, la gran parte del manejador de tipos está operativo y el manejador de objetos almacenables también, el diseño del lenguaje E fue terminado y actualmente se encuentran implementando el compilador E.

III.5 GENESIS.

GENESIS [BATO 86a] [BATO 87b] es un proyecto desarrollado en la Universidad de Texas, Austin. Este proyecto puede clasificarse como ´sistema para armar´. Sus propios autores lo definen como un sistema reconfigurable.

GENESIS se apoya en una teoría sobre la construcción de SGBD. Dicha teoría identifica componentes básicos del software de SGBD, requiriendo que todos tengan la misma interface y mostrando que su composición puede ser realizada de una manera simple. A dichos componentes básicos se les llama **building blocks**. Esta teoría se describe con más detalle en [BATO 85].

La reconfiguración del SGBD es posible por la reorganización de los componentes básicos y la utilización de una biblioteca de módulos. Dicha biblioteca es extendible, aumentando las posibilidades en cuanto a la generación de SGBD.

GENESIS propone una estrutura por capas, donde las operaciones de cada capa sirven de sostén a las de la capa superior. La relación entre capa y capa está dada a través de las **interfases estandar**, las cuales son utilizadas para relacionar una operación conceptual con los algoritmos que la implementan. [BATO 87b]. Establece así una correspondencia entre el nivel abstracto y el nivel concreto tnnto en datos como en algoritmos basado en las **trasformaciones elementales**.

Para reconfigurar el sistema, el DAA(Database Architecture Administrator) tiene que escribir las correspondencias abstracto a concreto en un programa de configuración llamado **programa de arquitectura.** Luego este programa es ejecutado, creando las **tablas de arquitectura** que utilizará GENESIS para realizar las correspondencias.
La correspondencia entre las **definiciones de los datos**, es implementada mediante rutinas llamadas **transformadores de tipos** , que son las que manipulan las tablas que hacen la correspondencia entre la estructura abstracta de un archivo y su contrapartida estructura concreta. Estos transformadores de tipos, según las estructuras que se quieran utilizar, tienen que ser escritos o tomados desde una biblioteca provista por GENESIS. [BATO 86a]
A su vez la correspondencia de las **operaciones** entre el nivel conceptual y el interno es realizada a traves de la composición de **extensores de operación.** Un extensor de operación es una rutina que hace la correspondencia entre la operación sobre el archivo abstracto y una secuencia de operaciones sobre archivos concretos.
Una transacción inicia operaciones sobre archivos y links conceptuales. El **grand central** utiliza los contenidos de las tablas de arquitectura para dirigir la traducción de operaciones conceptuales en su contraparte interna. Para realizar efectivamente la correspondencia nivel a nivel son utilizados los extensores de operaciones. Un modelo de optimización de consultas basado en lo expresado es detallado en [BATO 87a}.
El nivel más bajo de estructuras y operaciones es resuelto por el **manejador de archivos** JUPITER [BATO 86a].

Extensibilidad en estructuras de almacenamiento y algoritmos.

La idea de extensibilidad en estructuras de almacenamiento no está dada sólo en poder agregar nuevas estructuras, sino también en poder construir otras a partir de la composición de las ya existentes.

La extensibilidad en algoritmos consiste en realizar de varias maneras una operación sobre una estructura dada. Por ejemplo, varias formas de recuperación en archivos invertidos o varios métodos para realizar el join de dos relaciones. Al agregar el nuevo algoritmo, se debe indicar además, la función de costo asociada al mismo (para la optimización de consultas).

La forma de poner en práctica la extensibilidad en algoritmos es similar a la de extensibilidad en estructuras de almacenamiento. La diferencia fundamental está en la forma en que se concreta la elección de un método. En el caso de estructuras de almacenamiento, la elección está dada en tiempo de compilación del SGBD, mientras que en el caso de algoritmos está dada en tiempo de ejecución de la consulta que realiza la operación.

Ejemplo: supongamos que se quiere recuperar los registros de un archivo abstracto F que satisfacen la consulta Q utilizando la operacion RET.

```
RET(F,Q)
{   case( F.implemetación ) of
    {       bplus:  RET_BPLUS(F,Q);
            isam:   RET_ISAM(F,Q);
                    .....
            index: RET_INDEX(F,Q);
    }
}
```

Mediante este ejemplo observamos que, cuando se agrega un nuevo caso, lo que se está haciendo es extendiendo las estructuras de almacenamiento.

Como un ejemplo de extensibilidad de algoritmos, supongamos que queremos realizar la operacion JOIN, que realiza el join de dos archivos abstractos A1 y A2.

```
JOIN(A1,A2)
{   case (costo_menor) of
    {       alg1:   NESTED_LOOPS(A1,A2);
            .....
            algn:   MERGE_SCAN(A1,A2);
    }
}
```

Aqui se puede observar que cuando se agrega un nuevo caso, se esta extendiendo los algoritmos que podrían realizar la operación.

Extensibiblidad en tipos de datos y operadores.

Para lograr extensibilidad en tipos de datos y operadores proponen una variante en el modelo funcional, la cual simplifica el modelo computacional. El modelo de datos esta basado en **producciones** en lugar de funciones, donde las producciones son reglas de reescritura de "streams". Un "stream" es una codificación de una **secuencia**, siendo una secuencia un conjunto de objetos entre llaves. Una producción hace la correspondencia entre un "stream" de entrada y un "stream" de salida. La implementación de una producción es un trasformador de "streams".
Por ejemplo: para listar los nombre de los departamentos que tienen empleados con edad superior a 55

```
Dept.WHERE(Empt.age .gt. 55).Dname.PRINT
```

Dept genera la secuencia de todos los departamentos. WHERE elimina aquellos departamentos que no tienen empleados cuya edad sea superior a 55. Dname reeplaza los objetos departamentos por sus nombres y PRINT los imprime.
Aplicaciones de bases de datos no tradicionales son manejadas en forma similar.

Perspectivas.

En cuanto al desarrollo sobre SGBD extensibles, este grupo se encuentra desarrollando un álgebra para el diseño de SGBD moleculares [BAT87a], con la cual pretenden que los aspectos significativos del diseño de arquitecturas de SGBD pueden ser reducidas a un formalismo simple, y reglas por las cuales se componen building blocks (o átomos) pueden ser expresadas algebraicamente.

En cuanto a perspectivas de desarrollo del prototipo, están investigando en dos campos: el de las base de datos y el del software. Con respecto al primero están incorporando al prototipo: control de concurrencia, procesamiento de consultas y control de las interfaces con usuarios finales. En particular sobre este último punto están estudiando la forma de permitir modelos de datos y lenguajes de consulta alternativos, así como la posibilidad de agregar nuevos tipos de datos.

III.6 GEODE

GEODE [PUCH 88] es un proyecto en desarrollo en el INRIA (Francia) y la Universidad de Paris VI. Es un sistema que puede ser considerado a la vez como ´sistema para armar´y como un núcleo de base para un SGBD orientado objeto.

Los principales objetivos del proyecto son :
- ofrecer un sistema abierto que permita la integración de varios modelos de datos.
- permitir la extensión de los tipos de datos por los usuarios.
- autorizar el almacenaje de objetos de gran volumen.
- tener en cuenta la evolución temporal de los datos a partir de la gestión de versiones.
- ofrecer mecanismos de almacenaje de bajo nivel para autorizar el dasarrollo de nuevos métodos de acceso a los datos.
- ofrecer mecanismos de gestión de la concurrencia adaptables a los tipos de datos y parametrables por el usuario.

Principales características

GEODE se presenta como un servidor de operaciones sobre los objetos de la base de datos. La extensibilidad del sistema se sitúa a dos grandes niveles : extensión vertical, a partir de la definición de nuevos tipos y extensión horizontal, a partir de nuevos métodos de acceso de bajo nivel.

La arquitectura de GEODE está organizada en 5 capas principales : gestión de tipos, gestión de Constructores genéricos, gestión de objetos y colecciones, gestión del almacenaje, gestión de la memoria virtual.
Esta arquitectura permite el acceso al sistema a varios niveles .

Búsqueda de buenas performances

GEODE propone técnicas adecuadas para mejorar las performances del SGBD. Ellas conciernen tres aspectos principales : contexto gran memoria, mecanismos de clustering y realización de las E/S en forma diferida.
El **contexto gran memoria** tiene por objetivo tener en cuenta la evolución tecnológica en el dominio de las memorias RAM, partiendo de la idea que para un gran número de aplicaciones se podrá disponer de una capacidad memoria suficiente como para soportar en memoria central todos los datos de una sesión.
Esto supone imaginar estructuras de datos en memoria que permitan optimizar el tiempo de cálculo CPU.

Clustering

Los objetos de un mismo tipo son organizados en Colecciones. El almacenaje de objetos en esas colecciones puede ser secuencial o ser organizados por regrupamientos en clusters .

E/S en diferido

GEODE trabaja como si la base de datos estuviera permanentemente en memoria central. El mecanismo de repercusión de las modificaciones sobre la memoria estable es realizado por un proceso independiente. Esto permite de no bloquear al sistema durante la realización de una escritura sobre disco, lo que significa una ganancia cosiderable de performance.

OIDs

Como EXODUS, GEODE ofrece el tipo de base OID, que representa una referencia única e invariable a un objeto.

Almacenaje de los objetos

El principio expuesto es basado en un almacenaje por páginas. Un objeto puede estar repartido entre varias páginas no forzosamente consecutivas. El acceso a los objetos se realiza a partir de primitivas de manipulación.

Versiones

El mecanismo de versiones permite la creación dinámica de nuevas versiones de objetos a partir de la secuencia de modificaciones realizada sobre ese objeto.
El método utilizado consiste en fabricar una nueva versión por la duplicación del artículo atómico del objeto donde fue hecha la modificación.

Concurrencia

Los algoritmos de control de la concurrencia de acceso a los datos son basados sobre el blocage en dos faces. Varios gránulos son propuestos : colecciones, objetos, sub-objetos y artículo atómico. Además del control de concurrencia clásico, es posible utilizar un mecanismo diferido aplicado únicamente a los objetos que participan del resulatado de una consulta.

Estado actual y perspectivas

GEODE es un proyecto iniciado recientemente (1987). Las capas internas del sistema son operacionales, y actualmente el desarrollo se concentra en la gestión de tipos y en la definición de un lenguaje de interface con características del enfoque objeto.

IV - CONCLUSION

Existe coincidencia en que los sistemas actuales plantean limitaciones para el desarrollo de nuevas aplicaciones, y en utilizar herramientas de otras áreas (ADT, objetos, deducción, etc) para hacer posible dicho desarrollo. Sin embargo, las propuestas existentes difieren en aspectos tales como : hacia quién está dirigido el producto (usuario o DBI), si cuenta o no con un modelo de datos definido (completo o para armar) o en que aspectos de extensibilidad pone mayor énfasis.

Las perspectivas de actividad en esta área parecen muy prometedoras. Los prototipos presentados muestran la puesta en práctica de un conjunto de ideas y soluciones originales. Estos sistemas constituyen actualemente una base importante para el prototipaje y desarrollo de nuevos modelos de datos y lenguajes de interrogación de tipos declarativo. A su vez, la confluencia con otras disciplinas, en particular con la inteligencia artificial, lenguajes funcionales y el enfoque orientado a objetos, han generado un campo importante de trabajo e investigación.

AGRADECIMIENTOS.

En su pasaje por Montevideo, el profesor Isidro Ramos de la Universidad de Valencia, tuvo la gentiliza de comentar este trabajo, por lo cual le estamos sumamente agradecidos. También han contribuido Phillipe Pucheral de la Universidad Paris VI, aportándonos ideas y preciosa documentación; así como Guillermo Calderón y Oscar Sanz, cuyas observaciones nos fueron sumamente valiosas.

BIBLIOGRAFIA

Adib 86 Adiba M., Bui N. CCollet C. ´Aspects Temporels, Historiques dans les Bases de Données. Revue TSI 1987.

Astr 76 Astrahan M. et al. ´System R: Relational Approach to Database Management´, ACM Trans. on Database Systemas, Vol 1, No 2. Junio 1976.

Banc 88 Bancilhon F. ´Object-Oriented Database Systems´. Invited Lecture. 7th ACM. Symposium on Principles of Data Base Systems. Austin. Texas. Marso 1988.

Bato85 Batory D. S. "Modeling the Storage Architectures of Commercial Database Systems". ACM - TODS 10,4 (Diciembre 1985).

Bato86a Batory D.S. "GENESIS: A reconfigurable database management system". To apear IEEE 1988.

Bato86b Batory D.S and T.Y.Leung. "Implementation Concepts for an Extensible Data Model and Data Language". TR-86-24. University of Texas, Austin. 1986.

Bato87b Batory D.S. "Principies of Database Manajement System Extensibility". IEEE Junio 1987.

Bent 75 Bentley J.L. "Multidimensional search trees user for associative searching". Communications ACM 18,9. (Setiembre 1975).

Bent 79 Bentley J.L. "Multidimensional binary search trees in database applicatios". IEEE Transactions in Software Engineering SE-5, 4. (Julio 1979).

Bern 87 Bernstein, Lomet. "CASE Requiremest for extensible database systems" IEEE Junio 1987.

Care 86 Carey M., DeWitt D. "Extensible database systems". "On Knowledge base management systems" Springer -Verlag. 1986.

Care 87 Carey M., DeWitt D. "An overview of the EXODUS Project". Database Eng. , Junio 1987.

Care 88 Carey M., DeWitt D, Vanderberg S. "A Data Model and Query Language for EXODUS". 1988.

Chou 85 Chou H. and DeWitt D. "An evaluation of buffer management strategies for relational databases". Proc. of the 11th VLDB Conf. Stockholm, Swedem, Agosto 1985.

Daya 85 Dayal U. Smith J. ´PROBE : A Knowledge-Oriented Database Management System´, Proceeding of the Islamorada Workshop on Large Sacle Knowledge Base ans Reasoning Systems, Febrero 1985.

Fag 79 R. Fagin, J. Nievergelt, N. Pippenger and H.R. Strong. "Extendible hashing-a fast access method for dynamic files". ACM Transactions on Database Systems 4,3. (Setiembre 1979), 315-344.

Gard 87a Gardarin G., Simo E. ´Les Systemes de Gestion de Bases de Données Deductives´, Revue TSI. 1987

Gard 87b Gardarin G., Pucheral P. ´Optimization of Generalized Recursive Queries Using Graph Traversal´, Informe INRIA. 1987.

Gold 87 Goldhirsch D. Orenstein J. "Extensibility in the PROBE Database System". Proc. of Extending database technology, Venesia 1987.

Grae 87 Graefe G., and DeWitt D. "The EXODUS Optimizer Generator". Proc. of the 1987 SIGMOD Conf. San Fransisco, CA. Mayo 1987.

Gutt 84 Guttman T. ´R-Tree : A Dynamic Index Structure for Spatial Searching´, Procceding the 1984 SIGMOD Conference, BOston MA Mayo 1984.

Hask 83 Haskin R., Lorie R. "On Extending the Functions of a Relational Database System". Proc. of ACM Enineering Design Applications (1983).

Katz 83 Katz R.H., Weiss S. ´Transaction Management for Design Databases´, Computer Sciences Technical Report 496, Febrero 1983.

Katz 86 Katz, Chang, Bhateja. "Version modeling concepts for computer-aided design databases". Proc 1986 ACM-SIGMOD. International Conference on Managemente of Data. Washington DC, Mayo 1986.

McPh 87 Mc Pherson J. , Pirahesh H. ´An Overvieuw of Extensibility in Starburst´, IEEE. Vol 10 no 2. Junio 1987.

Lars 80 Larson P. "Linear hashing with partial expansions" . Proceedings 6th Conference on VLDB, Montreal, Canada , 1980, pp. 224-232.

Litw 80 Litwin W. "Linear Hashing: A new tool for file and table addressing". Proc. 6th Int´l Conf. on VLDB, Montreal, 1980, pp. 212-223.

Litw 87 Litwin W and Lomet. "A new method for fasst data searches with keys". IEEE Software 4,2 (Marzo 1987).

Lome 83 Lomet D. B.. "Bounded index exponential hashing". ACM Transactions on Database Systems 88,1. (Marzo 1983), 136-165.

Lome 87 Lomet D.B. "Partial expansions for file organizations with an index". ACM Transactions on Database Systems 12,1. (Marzo 1987).

Lori 83 Lorie R., Plouffe W. "Complex Objects and their use in design transactions". Proc of ACM Engineering Design Applicatios (1983).

Niev 84 Nievergelt J., Hintenberger H., Sevcik K.C. ´The Grid File : An Adaptable, Symmetric Multikey File Structure´, ACM Transaction on Database Systems, Vol 9 no 1, Marzo 1984.

Rich 87 Richardson J., Carey M. "Programming Constructs for Database System Implementation in EXODUS". Proc. of the 1987 SIGMOD Conf. , San Fransisco, CA. Mayo 1987.

Robi 81 Robinson J.T. ´The k-d-B-tree : A Search Structure for Large Multidementional Dynamic Index´, Proccedings of the 1981 SIGMOD Conference, Junio 1981.

Puch 87 Pucheral P. ´Traitement Efficace des Regles Récursives par un Opérateur Basé sur des Parcours de Graphe´, IIIe Journées Bases de Données Avancées, Port Camargue , 1987.

Puch 88 Pucheral P., Thevenin J.M., Steffen H. ´GEODE : Un Gestionnaire d´Objetos Extensible Orinté Grande Mémoire´, Proceeding of Base de Données de Troisieme Génération. Benodet, Junio 1988.

Lori 83 Lorie R., Plouffe W. "Complex Objects and their use in design transactions". Proc of ACM Engineering Design Applicatios (1983).

Niev 84 Nievergelt J., Hintenberger H., Sevcik K.C. ´The Grid File : An Adaptable, Symmetric Multikey File Structure´, ACM Transaction on Database Systems, Vol 9 no 1, Marzo 1984.

Rich 87 Richardson J., Carey M. "Programming Constructs for Database System Implementation in EXODUS". Proc. of the 1987 SIGMOD Conf. , San Fransisco, CA. Mayo 1987.

Robi 81 Robinson J.T. ´The k-d-B-tree : A Search Structure for Large Multidemention al Dynamic Index´, Proccedings of the 1981 SIGMOD Conference, Junio 1981.

Puch 87 Pucheral P. ´Traitement Efficace des Regles Récursives par un Opérateur Basé sur des Parcours de Graphe´, IIIe Journées Bases de Données Avancées, Port Camargue , 1987.

Puch 88 Pucheral P., Thevenin J.M., Steffen H. ´GEODE : Un Gestionnaire d´Objetos Extensible Orinté Grande Mémoire´, Proceeding of Base de Données de Troisieme Génération. Benodet, Junio 1988.

Sche 82 Scheuermann P. and M. Ouksel. "Multidimensional B-trees for associative searching in database systems". Informations Systems 7,2. (1982).

Ston 84a Stonebracker et al. "QUEL as a data type". Proc. 1984 ACM-SIGMOD.

Ston 86 Stonebraker M. and Rowe L. "The design of POSTGRES". Proc. 1986 ACM-SIGMOD Conference on Management of Data. Washington D. C. Mayo 1986.

Ston 88 Stonebraker M., Anton J., Hirohama M. "Extendability in POSTGRES". To appear.

Zani 83 Zaniolo. "The database lenguaje GEM". Proc 1983 ACM-SIGMOD.