RAD PACK PARA DELPHI ▼ OLE 2.0

RPP

REVISTA PROFESIONAL PARA PROGRAMADORES

Año III/Nº 3-0696-16 ▼ 950 PTAS.

SECCIONES:

- **DELPHI:** *Cómo programar DLLs*
- **FORUM C:** *Especificaciones de requerimientos*
- **TRUCOS DEL LECTOR:** *Cómo detectar si Windows está activo*

HERRAMIENTAS OS/2

VISUAL AGE C++
VISUAL AGE SMALLTALK
VISPRO/REXX
VX-REXX
DB2/2 2.1

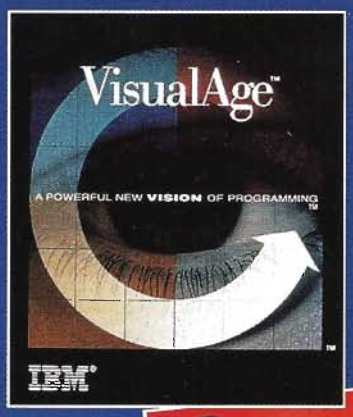
¿QUÉ ES JAVA?

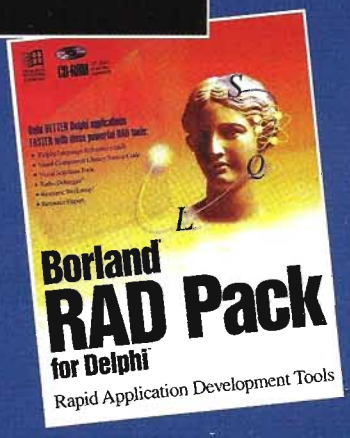
PROGRAMAS RESIDENTES: USO DE LA INTERRUPCIÓN DE TECLADO

(Canarias, Ceuta, Melilla 975 ptas.)
00016
8 424094 405039

EN ESTE NUMERO ...

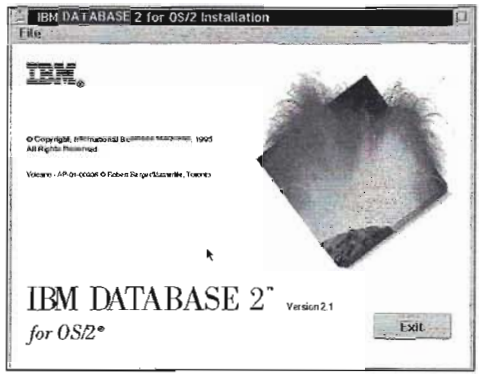
Marzo 1996 Número 16





Bases de datos relacionales

Novedades en DB2/2 versión 2.1



Hace relativamente poco que ha aparecido en el mercado la versión 2.1 de DB2/2. DB2/2 es la versión para OS/2 de DB2, la conocidísima base de datos relacional de IBM, líder absoluto en el mercado mundial de bases de datos relacionales. En este artículo, basado en nuestro contacto con la beta de junio de este producto y con la versión final, examinaremos algunas de las novedades que la nueva versión aporta.

DB2 (DataBase 2) es la implementación SQL de IBM. Hay versiones para OS/2, para sistemas Unix (actualmente AIX, HP-UX y Solaris), para OS/400, y para mainframes en VSE, VM y MVS. Las distintas versiones están convergiendo, de modo que se habla de un *common server* para OS/2 y las plataformas Unix (la mayoría del código es el mismo). El hecho de que el mismo producto esté disponible en varias plataformas permite un nivel de escalabilidad casi ilimitado: es posible escribir una aplicación en un ordenador Intel de prestaciones relativamente limitadas, convertirla automáticamente en cliente-servidor sin más que añadir nodos de LAN, migrar a un sistema Unix cuando las posibilidades de un servidor Intel empiecen a agotarse, e ir aumentando así sucesivamente la capacidad del servidor hasta un super-mainframe con sistema MVS/ESA, que admite decenas de miles conexiones simultáneas, sin necesidad de tocar ni una sola línea de la aplicación ni de nuestra base de datos.

OS/2 dispone de bases de datos relacionales desde la versión 1, donde se podía adquirir OS/2 como *Standard Edition*, que constaba sólo del sistema operativo base, o como *Extended Edition*, que incluía además soporte para comunicaciones SNA (Communications Manager) y bases de datos SQL (Database Manager). La versión 2 separó los componentes propios de la Extended Edition de la versión 1 en un paquete llamado *Extended Services*; a continuación, estos componentes empezaron a venderse por separado; es allí donde aparece por primera vez DB2/2 como producto independiente. La última versión ampliamente distribuida antes de la que nos ocupa es la versión 1.2.

José María Blasco

José María Blasco, nacido en 1960, es Licenciado en Matemáticas por la Universidad de Barcelona. Ha sido profesor de Programación en la Facultad de Informática de Barcelona, y Coordinador Nacional de la red EARN en Alemania. Sus temas de interés más recientes son las redes, la programación orientada a objeto, las bases de datos, y todo lo relacionado con OS/2. Su dirección Internet es jmblasco@eim.ub.es.

Francesc Rosés

Francesc Rosés, nacido en Barcelona en 1958, es licenciado en Filología Catalana por la Universidad de Barcelona. Desde 1985 desarrolla en el Centre dInformàtica de la Universitat de Barcelona (CIUB) la coordinación informática de diversos proyectos de lexicografía computacional. Profesor del Máster de Lingüistica Aplicada de la Universidad de Barcelona. Desde 1994 es el responsable de la sección de Microinformática del CIUB. Su dirección Internet es froses@fenix.ird.ub.es.

DB2 2.1 no es sólo un nuevo *release* de DB2, sino una nueva versión, y se nota. El procedimiento de instalación utiliza el *software installer* hacia el que están convergiendo todos los productos IBM. Una instalación completa del servidor coloca 44 *megas* en el directorio SQLLIB, contra 9 MB para la versión 1.2. La nueva carpeta de información de DB2 contiene 10 libros (*books* en formato .INF) en vez de los 3 de la 1.2; los nuevos libros son absolutamente exhaustivos. Se incluye también una carpeta con todas las utilidades necesarias para acceder a DB2 desde Windows (directamente, o mediante aplicaciones que utilicen ODBC). También es posible gestionar el servidor mediante SNMP, para lo que se incluyen iconos para activar y desactivar el agente SNMP para DB2.

Novedades en la administración

Los usuarios de DB2/2 1.2 se encontrarán con una sorpresa: Query Manager ha desaparecido. Y es una lástima: aunque Query Manager estaba bastante anticuado, y no soportaba las adiciones más recientes a DB2/2, era bastante práctico para tener una panorámica de la estructura de la base de datos, o para crear y manipular tablas. Con la nueva versión se hace necesario guardar cuidadosamente las definiciones SQL de nuestra base de datos (probablemente en ficheros ASCII), para poderlas revisar y cambiar cuando lo deseemos. Otra alternativa es utilizar el producto de IBM Visualizer Query, que no revisamos aquí, y del que sólo diremos que ofrece un nivel de integración con el Workplace Shell de OS/2 y una capacidad de administración de DB2 que alcanza niveles simplemente impresionantes, o la versión más reducida, Visualizer Flight, que está actualmente en *beta* gratuita disponible en Internet. Otra posibilidad es administrar nuestras bases de datos con la ayuda de las herramientas especializadas que se incluyen en las versiones avanzadas de Vispro/REXX o VX-REXX.

Las diferentes utilidades de administración presentes en la versión 1.2 (CONFIG.EXE, RECOVERY.EXE y DIRECT.EXE) han sido substituidas por una nueva herramienta unificada, llamada *Database Director*, que es mucho más potente y general, aunque la encontramos bastante lenta en esta versión.

La comunicación entre clientes y servidores es posible utilizando cualquiera de los protocolos más comunes: APPC (que es parte de SNA), IPX/SPX (mediante la ayuda de un servidor Novell Netware que actuará como *router*), NETBIOS (nativo, o sobre IP), y TCP/IP (nuevo en esta versión). Una vez instalado el software, habrá que indicar cuáles son los protocolos mediante los cuales deseamos que nuestro servidor sea accesible. Esto se hace añadiendo una línea al CONFIG.SYS, por ejemplo:

```
SET DB2COMM=NETBIOS,TCPIP
```

Esta línea indica que entre todos los protocolos posibles, elegimos que nuestro servidor sea accesible simultáneamente mediante NETBIOS y TCP/IP (esta información está bastante escondida en los manuales del CD-ROM). La adición de TCP/IP es muy importante, porque permite, por ejemplo, publicar servidores de DB2 en Internet, o instalarlos en MANs y WANs que sólo soporten IP sin tener que añadir NETBIOS sobre IP, como era necesario hasta ahora.

La comunicación entre clientes y servidores es posible utilizando cualquiera de los protocolos más comunes

En cuanto a la ubicación física de las bases de datos, DB2 ofrece ahora dos posibilidades: podemos dejar al sistema la responsabilidad de escoger el lugar de almacenamiento de los ficheros que implementan nuestra base de datos (como en la versión 1.2, sólo necesitaremos entonces especificar la unidad lógica en la que se va a guardar la base de datos), o bien podemos definir una serie de espacios de tablas (*tablespaces*) en la misma o diferentes unidades lógicas, para decidir después dónde queremos almacenar nuestras tablas o índices.

Si optamos por esta última posibilidad, el nivel de control del almacenamiento físico es muy grande: cada *tablespace* puede consistir en un numero arbitrario de directorios, posiblemente distribuidos en diferentes unidades, y para cada directorio podemos indicar al sistema una estimación del coste de acceso (tiempo de acceso, transferencia media por segundo, etc.), que será utilizada por el optimizador de DB2 al elegir la distribución de los datos. A continuación, para cada tabla definida, podemos indicar el *tablespace* en el que se va a almacenar, indicando si lo deseamos diferentes *tablespaces* para los índices o para los objetos largos (BLOBs, CLOBs y DBCLOBs, de los que hablamos más adelante). De este modo, si contamos con un servidor con varias unidades de disco duro SCSI, para las cuales es posible la entrada y salida simultánea desde varias unidades, es posible distribuir, por ejemplo, las diferentes tablas de un *join* muy utilizado en discos distintos, de modo que los datos del *join* se carguen en paralelo; o podemos poner los índices de una tabla en un disco y los datos en otro, de modo que el acceso al índice y a los datos asociados pueda ejecutarse simultáneamente, etc. En resumen, en este aspecto el nivel de control permitido es similar al de los grandes sistemas

de bases de datos de un *mainframe*, pero sin obligar a nadie: si se desea, todo el trabajo lo realiza DB2, escogiendo por nosotros valores correctos para nuestra base de datos.

El sistema de backup ha cambiado también bastante: en vez de realizar los backups directamente a una unidad lógica, como en la versión 1.2 (lo cual obligaba muchas veces a utilizar trucos de red mediante NET USE o MAP para poder hacer backups a unidades de disco duro), ahora los backups se realizan directamente a un directorio o a un dispositivo (como una unidad de cinta). Los backups son extensivos, y ocupan mucho espacio, incluso para bases de datos pequeñas. El truco está, si deseamos transportarlos en disquete, en hacer previamente un ZIP del directorio de backup. Por ejemplo, la base de datos SAMPLE ocupa unos 8 MB, pero comprimida ocupa sólo 0.5 MB y cabe perfectamente en un disquete.

Figura A DB2 incluye soporte integrado para Windows y ODBC

Novedades en el lenguaje SQL

El lenguaje SQL implementado por la versión de DB2/2 que estamos examinando ha evolucionado de forma muy notable desde la versión 1.2, y lo ha hecho al menos en dos direcciones clave: por una parte, al implementar numerosas extensiones que podríamos llamar *de usabilidad*, que permiten obtener salidas más claras, realizar con mucha facilidad tareas que antes eran complejas o requerían el uso de pasos intermedios (como la creación de tablas auxiliares), o descargar la aplicación en favor del servidor, al implementar nuevas funciones incorporadas ejecutables directamente por el servidor y, por otra parte, al incluir toda una serie de *extensiones objetuales*, como los tipos definidos por usuario, funciones definidas por el usuario, *triggers*, *checks*, y objetos largos. Examinaremos estos dos tipos de extensiones separadamente.

Extensiones de usabilidad

Funciones

DB2/2 2.1 incorpora gran cantidad de funciones, además de permitir al usuario definir las suyas propias. Esto significa que gran parte del formateo necesario de los datos puede realizarlo el servidor, y que, mediante el uso de vistas escogidas cuidadosamente, podemos presentar al usuario la información de modos mucho más sofisticados que en las versiones anteriores. Además, disponemos de la construcción CAST, que permite la conversión explícita de un tipo a otro, y que ilustramos con un ejemplo:

```
CAST ( micolumna_entera AS FLOAT)
```

Identificadores entrecomillados

DB2/2 2.1 permite el uso de identificadores entrecomillados en la mayoría de los lugares en los que sintácticamente se requiere el uso de un identificador. Los identificadores entrecomillados pueden contener mayúsculas y minúsculas (que se respetan), caracteres especiales, y espacios en blanco. Esto permite asignar nombres más significativos a, por ejemplo, las columnas de una tabla o una vista, pero también nos obliga a referirnos siempre al identificador utilizando su forma entrecomillada (es decir, reproduciendo exactamente su secuencia de mayúsculas, minúsculas, espacios en blanco, etc.):

```
Create Table T ("Columna 1"
Integer, "CoLuMnA_2" Float)
```

Crea una tabla con dos columnas llamadas exactamente "Columna 1" (con blancos en medio), y "CoLuMnA_2" (con la combinación mostrada de mayúsculas y minúsculas).

Cláusula AS en la instrucción SELECT

Cuando una de las columnas de un *select* en DB2/2 1.2 era una expresión, DB2 asignaba automáticamente a la columna del resultado un nombre, normalmente COLn, donde *n* era el número de columna. Con DB2/2 2.1, podemos indicar mediante la cláusula AS de la instrucción *select* cuál es el nombre de la columna:

```
Select DIA, Max(VALOR) AS MAXIMO From...
```

es un select cuyo resultado es una tabla con dos columnas, de nombre DIA y MAXIMO. Aquí también es posible utilizar identificadores entrecomillados. Y el mecanismo es válido para definir vistas, que pueden tener así los nombres de columna que deseemos. Es especialmente interesante combinar esta posibilidad con el uso de funciones incorporadas para obtener directamente selects y vistas que con la versión anterior de DB2 requerían de una mini-aplicación.

Expresiones comunes de tabla

Las *expresiones comunes de tabla (common-table-expressions)* permiten la definición local de tablas auxiliares dentro de una instrucción *select*. La sintaxis es:

```
WITH
  expresión-común-de-tabla
SELECT ...
```

donde una expresión común de tabla tiene la forma

```
nombre-de-tabla [(columna
[,columna...])] AS (instrucción-select)
```

Supongamos, por ejemplo, que tenemos una tabla VALORES con dos columnas, DIA y VALOR; para cada DIA disponemos de varios VALORes. Deseamos construir un select que nos entregue, para cada DIA, el máximo y el mínimo VALOR. Podemos escribirlo elegantemente así:

```
WITH
  Maximos AS
    (Select Dia, Max(Valor)
As Maximo From Valores Group By Dia),
  Minimos AS
    (Select Dia, Min(Valor)
As Minimo From Valores Group By Dia)
SELECT Maximos.Dia, Maximos.Maximo,
Minimos.Minimo
FROM   Maximos, Minimos
WHERE  Maximos.Dia = Minimos.Dia
```

El ejemplo nos permite observar que las expresiones comunes de tabla pueden ser objeto de todas las manipulaciones propias de una tabla normal. A efectos de la instrucción *select*, esas tablas existen con el mismo rango que cualquier otra tabla, pero se crean y se destruyen dinámicamente en el ámbito de su ejecución. De hecho, las tablas se crean por orden de aparición, de modo que si escribimos *WITH ect1, ect2, ect3 SELECT...*, donde *ecti* son expresiones comunes de tabla, *ect2* puede hacer referencia a *ect1*, y *ect3* a *ect1* y *ect2*.

Expresiones CASE

Las expresiones CASE permiten calcular un valor como el resultado de calcular una expresión u otra según se cumplan determinadas condiciones. La expresión

```
A/B
```

da un error de desbordamiento si B es 0. Si el problema que nos planteamos requiere substituir la expresión por 0 si B=0, en DB2/2 1.2 nos vemos forzados a escribir

```
SELECT A/B FROM... WHERE B <> 0
UNION
SELECT 0 FROM... WHERE B = 0
```

lo cual es poco claro. Con DB2/2 2.1 podemos escribir

```
SELECT
  CASE
    WHEN B <> 0 THEN A/B
    ELSE 0
  END
FROM ...
```

que está mucho mejor. Otro ejemplo es

```
CASE
  WHEN A < B THEN 'Menor'
  WHEN A > B THEN 'Mayor'
  ELSE 'Igual'
END
```

Restricciones ("constraints")

Las restricciones son condiciones asociadas a una tabla que limitan los valores de una columna o de toda la fila a cumplir determinado conjunto de condiciones. Permiten implementar hasta un cierto punto la integridad de dominio, cuando se aplican a una sola columna, y definir condiciones estructurales para las tuplas o filas de la tabla cuando se aplican a varias. Veamos unos ejemplos:

```
Create Table T (DIA Integer Check (DIA
Between 1 And 31),...)
```

define una columna DIA de tipo INTEGER que sólo podrá tener valores entre 1 y 31; si intentamos ejecutar una instrucción INSERT o UPDATE que cambie el valor de DIA en alguna fila a un valor que no esté entre 1 y 31, la instrucción fallará.

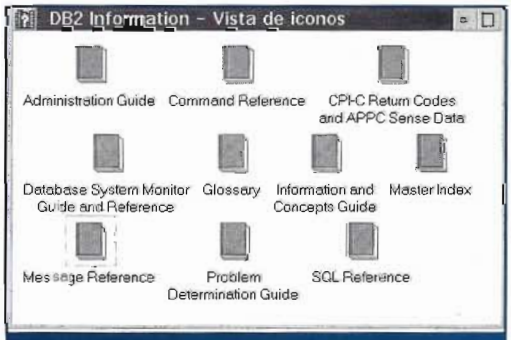


Figura B La carpeta de información de DB2 es muy completa

```
Create Table T (
  Lipidos Float
  AGS Float,
  AGM Float,
  AGP Float,
  Constraint Acidos_Grasos Check
(AGS+AGP+AGM <= Lipidos)
)
```

obliga a que cualquier fila cumpla la condición de que la suma total de los ácidos grasos no sea superior a la cantidad de lípidos. La restricción tiene un nombre ("Acidos_ Grasos"), y puede ser objeto de manipulación (p. ej. "ALTER TABLE T DROP CONSTRAINT Acidos_Grasos"). Si intentamos añadir una restricción a una tabla y existe alguna fila que no la verifica, la instrucción de añadido falla.

Las restricciones sólo pueden referirse a las columnas de una fila de la tabla a la que se aplican; si se desea implementar condiciones más complicadas, deberemos recurrir a los *triggers*, explicados más adelante.

Extensiones objetuales

Las extensiones que hemos llamado *de usabilidad* son especialmente útiles para el usuario "clásico" de DB2: son un conjunto de extensiones *naturales* al DB2, que lo hacen mucho más potente, y con toda seguridad más de un usuario habrá anticipado su creación al preguntarse por qué no se implementaron antes. En cambio, las extensiones objetuales (objetos largos, y tipos y funciones definidos por el usuario) están destinadas a abordar un nuevo tipo de problema.

Por ejemplo, multimedia: interesa poder almacenar en bases de datos relacionales campos de tipo imagen, audio o vídeo, de modo que la información sobre un objeto multimedia (por ejemplo autor, tamaño, y fecha) esté almacenada junto con el propio objeto.

Los datos podrán guardarse en la base de datos si admitimos una extensión al SQL que permita el almacenamiento de objetos largos. Pero almacenar los datos no es suficiente, pues —además de las con-

sultas SQL usuales— hemos de poder consultar los valores de esos campos *en cuanto a su subestructura*. Por ejemplo, si disponemos de una base de datos con campos de autor, fecha y texto estructurado, hemos de poder preguntar ¿cuáles son los documentos del autor X escritos antes del 15 de Diciembre *que contienen la palabra "cristalografía" en la bibliografía del documento*? La segunda parte de la pregunta se refiere a la estructura interna del campo de texto estructurado. O bien, si disponemos de una base de datos de imágenes, hemos de ser capaces de buscar imágenes *parecidas* a una tomada como referencia, consistiendo el parecido en determinados criterios de color, sombreado o textura que nosotros especificaremos.

DB2/2 2.1 incorpora algunas de las extensiones del SQL3, y permite definir *tipos definidos por el usuario* basados en tipos existentes, incluyendo los *objetos largos*, y asociados a *funciones definidas por el usuario*, además de *triggers* ("gatillos" o "disparadores"). Mediante todo ello es posible definir objetos en SQL, almacenarlos en tablas, y manipularlos mediante las instrucciones usuales de SQL, además de mediante las que el propio objeto define. Estudiaremos primero cada una de estas extensiones, para comentar después cómo pueden utilizarse para implementar *extensiones relacionales*.

Objetos grandes

Los objetos grandes (*Large Objects*, o LOBs) son nuevos tipos de datos básicos que se pueden utilizar para almacenar cualquier tipo de información. Hay de tres tipos: CLOBs, o tiras de caracteres largas, asociados con una página de códigos y con una longitud máxima de 2 gigabytes; DBCLOBs, o tiras de caracteres de doble byte, con una longitud máxima de 1 GB; y BLOBs, objetos binarios largos, sin página de códigos y de hasta 2 GB.

Dado que transferir un objeto largo del servidor al cliente puede ser muy costoso, y que en muchos casos sólo interesa obtener una parte del valor, DB2/2 2.1 implementa el concepto de *localizador LOB*, que permite obtener un apuntador al valor de un objeto grande y utilizar después ese apuntador para efectuar operaciones sobre el valor, obteniendo al final sólo la parte resultante de la operación, que posiblemente será mucho más pequeña que el valor inicial, y por tanto cargará menos la máquina cliente. Por ejemplo, un BLOB que sea un vídeo puede fácilmente ocupar muchos MB de memoria, pero si pedimos sólo un determinado *frame* la memoria requerida es mucho más pequeña.

Tipos definidos por el usuario

Los tipos definidos por el usuario (*user-defined types*) permiten partir de un tipo definido por el sistema y crear un nuevo tipo, con un nombre definido por nosotros; SQL tratará a partir de entonces el nuevo tipo como un tipo más, permitiéndonos crear columnas de ese tipo, y además realizará verificaciones *strongly-typed* sobre los valores correspondientes.

Supongamos, por ejemplo, que deseamos construir una tabla de alumnos y otra de cursos, ambas con claves primarias enteras. Definiremos primero dos tipos:

```
Create Distinct Type ALUMNO as
INTEGER With Comparisons
Create Distinct Type CURSO as
INTEGER With Comparisons
```

A continuación podemos definir las tablas:

```
Create Table Alumnos (ID Alumno,
...)
Create Table Cursos (ID Curso,
...)
```

en las que hemos utilizado *alumno* y *curso* como tipos, respectivamente, en vez de *integer*. De este modo, si intentamos comparar Alumnos.ID con Cursos.ID, el sistema detectará el error a nivel sintáctico. Además, las funciones normalmente utilizables sobre *integer* no están disponibles (a menos que las habilitemos manualmente mediante funciones definidas por el usuario). De este modo, no será posible calcular *Max(Alumnos.ID)*, lo cual tiene bastante sentido.

Funciones definidas por el usuario

Las funciones definidas por el usuario tienen dos usos fundamentales: permitir que un tipo de datos creado por nosotros *vea* alguna de las funciones incorporadas (por defecto no ve ninguna, como hemos explicado), y realmente definir nuevas funciones, aplicables a uno o más valores, ya sean de tipos definidos por el sistema o de tipos definidos por el usuario. Esto último se

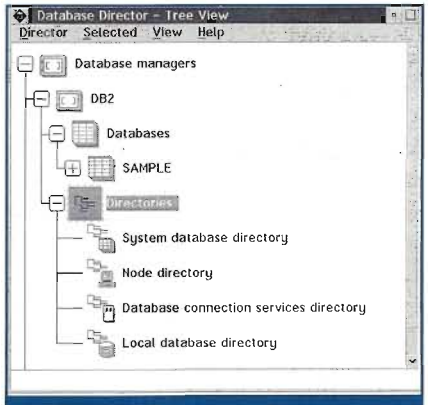


Figura C La nueva herramienta de administración, el Database Director, muy potente y completo pero algo lento

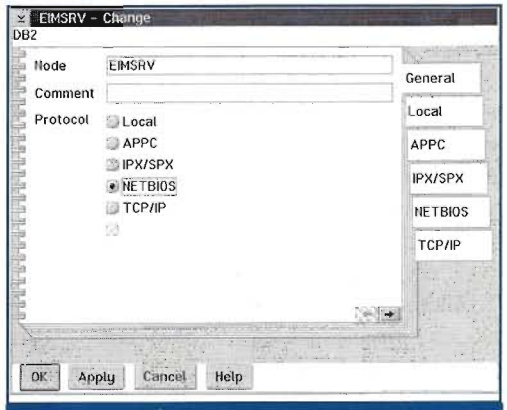


Figura D El cliente puede acceder a servidores (nodos) utilizando diferentes protocolos de comunicaciones

hace escribiendo código en lenguaje C, que, una vez compilado, se añade a la definición de la base de datos. De este modo, *es la base de datos y no la aplicación la que contiene la definición de la función*, de modo que la posibilidad de errores (accidentales o intencionados) se ve muy reducida, y el mantenimiento de múltiples aplicaciones sobre la misma base de datos muy simplificada. No reproducimos aquí nada del diagrama sintáctico para las funciones definidas por el usuario, aunque indicaremos dos cosas: que es posible definir atributos que indican cosas como si una función tiene o no efectos laterales, y que, dado que escribimos nuestras funciones en lenguaje C, es posible que, de haber efectos laterales, éstos tengan efecto sobre universos distintos al SQL: por ejemplo, podemos escribir una función que envíe correo electrónico.

"Triggers"

Los *triggers* ("gatillos" o "disparadores") son mecanismos que son activados ("disparados") cuando se produce una determinada modificación en una tabla. Por ejemplo, podemos tener un contador de personal en una tabla de contadores que se incremente automáticamente cada vez que se añade una fila a la tabla de personal y se decremente automáticamente cuando se suprima una fila. O podemos mantener automáticamente una tabla de seguridad con los valores suprimidos o modificados en otra tabla que contenga datos altamente sensibles. La idea de los triggers es a la vez muy sencilla y extraordinariamente potente (especialmente si los combinamos con las restricciones ("constraints") y con las funciones definidas por el usuario. Como ejemplo, escribiremos los *triggers* para mantener el contador de personas.

```
Create Trigger Contratar
  After Insert On Personal
  For Each Row Mode DB2SQL
   Update Estadisticas Set Empleados =
Empleados + 1
```

y

```
Create Trigger Despedir
  After Delete On Personal
  For Each Row Mode DB2SQL
  Update Estadisticas Set Empleados =
Empleados - 1
```

Su significado es bastante evidente. Los *triggers* funcionan del modo atómico a que estamos acostumbrados con DB2: si una instrucción SQL activa un *trigger* y por alguna razón éste no se ejecuta con éxito, la operación entera se deshace por completo (*rollback*). Esto quiere decir que, siguiendo con nuestro ejemplo, *Estadisticas.Empleados* contendrá el número de filas de *Personal* exactamente, *con el mismo grado de fiabilidad a que estamos acostumbrados cuando empleamos la integridad referencial o las unidades lógicas de trabajo*. O sea que, efectivamente, estamos añadiendo integridad estructural a nuestra base de datos a unos niveles impensables antes de la aparición de los *triggers*. Naturalmente, un *trigger*, al ser activado, puede a su vez activar uno o más *triggers* suplementarios, y así sucesivamente. Además, las restricciones (*constraints*) correspondientes se verificarán. La operación que estamos realizando seguirá siendo para nosotros atómica, o se ejecuta todo bien, o no se ha ejecutado nada.

Si una instrucción SQL activa un trigger y éste no se ejecuta con éxito, la operación entera se deshace por completo (rollback)

Las extensiones relacionales

Los mecanismos que hemos descrito constituyen una arquitectura bastante completa para la implementación de *extensiones relacionales*. Una extensión relacional es un paquete de tipos y funciones definidos por el usuario, junto con algunos *triggers*, que permiten la adición de nuevos tipos de datos orientados a objeto a nuestra base de datos. Un nuevo tipo *strongly typed* accesible mediante funciones es lo más parecido que conocemos a un objeto. IBM dispone de extensiones relacionales para imágenes, audio, vídeo y texto estructurado, que se incorporan a DB2 y permiten utilizar esos nuevos tipos como si fuesen nativos, manipulando cada uno de ellos de acuerdo con sus características.

Por ejemplo, la extensión de texto permite definir campos de texto, que podrán después buscarse por contexto, por sinónimos en 17 idiomas, por proximidad, o por contenido, sin más que instalar la extensión. En realidad, lo que IBM ha hecho ha sido convertir su producto *SearchManager/2*, que es una evolución OS/2 del antiguo STAIRS/VS, en una extensión relacional que permite la extracción de información de documentos estructurados (por ejemplo, en SGML).

la extensión para imágenes que incorpora la tecnología QBIC (*Query By Image Contents*), que permite buscar imágenes por forma, color o textura (el anuncio de las botellas que se está publicando actualmente).

Conclusión

Hemos intentado cubrir las novedades de DB2/2 2.1 que nos han parecido más llamativas en los primeros meses de contacto; obviamente, no hemos podido explicarlas todas. El producto ha crecido mucho, y para bien, de modo que no tiene nada que envidiar a los sistemas del *mainframe*. Esto se ha conseguido mediante mejoras en la administración, extensiones de usabilidad del lenguaje SQL que lo hacen mucho más potente, y una primera aproximación a las bases de datos relacionales orientadas a objeto que —estamos seguros— es sólo un primer paso y abre un futuro que promete ser apasionante.