Antecedentes de las bases de datos

Daremos un repaso a la historia de las Bases de Datos, mejor dicho, de los Sistemas de Gestión de Base de Datos que existían en el momento (sobre 1984-85). Es más, comenzaremos incluso antes, por conocer cómo eran las técnicas de almacenamiento de la información antes de que existieran las Bases de Datos (de cualquier tipo), o cuando sí que existían, pero, debido a sus limitaciones, no se podían usar…

Como la serie tiene ya unos pocos artículos, en esta dirección encontraréis el enlace a todos los artículos publicados hasta ahora.

Los primeros ordenadores sólo eran capaces de procesar una clase de ficheros: los secuenciales. Es decir, un conjunto de registros de información guardados todos juntos, uno detrás de otro, y que se leían o escribían también de uno en uno, uno tras otro, hasta acabar todo el conjunto.

Los primeros ficheros *informáticos* fueron en tarjeta perforada (ya la máquina de Hollerith para el censo de 1890 funcionaba con ellas). Un fichero en tarjetas perforadas es un bloque de tarjetas que tienen perforaciones que representan los caracteres que componen la información del registro. Los bloques de tarjetas se procesaban todos completos: se comenzaba a leer desde la primera tarjeta hasta alcanzar la última. Por tanto, si por algún motivo era necesario acceder exclusivamente a un registro, se debía leer secuencialmente todo el bloque, desde el principio, hasta llegar a él.

En esta nostálgica entrada, gracias a Jaume, os mostré cómo era un programa Cobol en tarjetas perforadas. Para que os hagáis una idea, un fichero en tarjetas perforadas es la misma cosa, sólo que las tarjetas tendrían todas la misma estructura, y en ellas habría grabados datos, no programas.

Unidad de Cinta IBM 2420 (años 60)

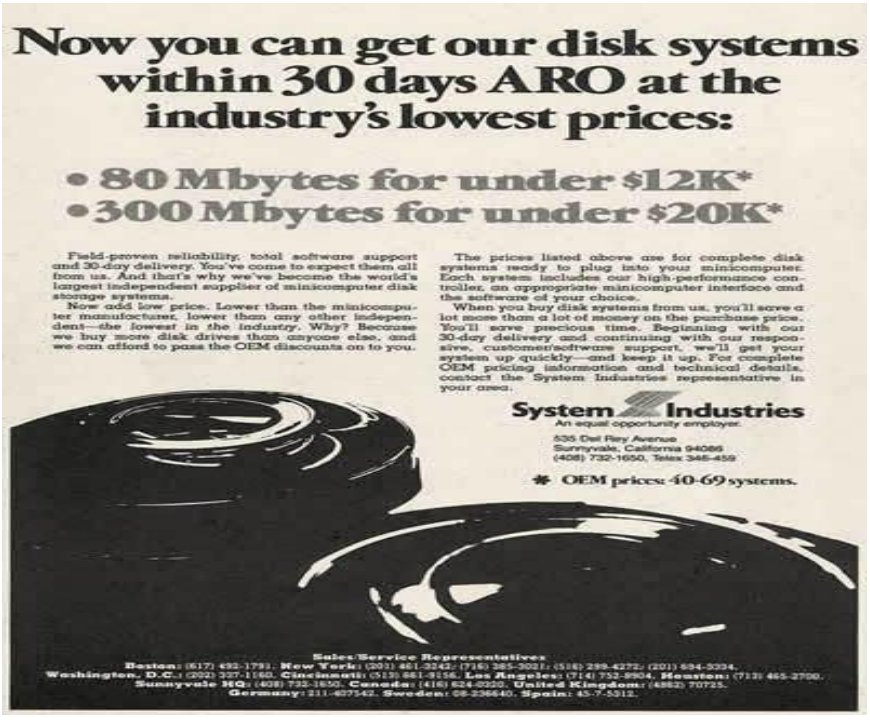
El soporte fue evolucionando. Primero, **cinta perforada**, heredada directamente del telégrafo, que es para las tarjetas lo mismo que los rollos de papiro en que los antiguos griegos escribían sus tratados, a los códices encuadernados en que los copistas medievales copiaron los escritos originales (los pocos que sobrevivieron, desde luego).

Después, a partir de principios de la década de 1950, fue la cinta magnética la que tomó el relevo… y esta vez para quedarse. Una cinta de 2.400 pies, la habitual a partir de los sesentas, grabada a 1600 ppi (*phrames per inch*, o caracteres por pulgada) podía almacenar unos 50 Mb de información… cuando los carísimos discos de la época podían almacenar quizá tres o cuatro… Las cintas magnéticas actuales almacenan Gigas y Gigas a un coste ridículo.

La aparición de la cintas magnéticas supuso un cambio sustancial en la forma de diseñar aplicaciones: ahora era posible mantener un fichero maestro (digamos, de Cuentas Corrientes) y diariamente acumular en otra cinta los movimientos del día, y entonces, leyendo simultáneamente ambos ficheros (¡el omnipresente *padre-hijo*!), escribir en una tercera cinta magnética el fichero maestro actualizado.

Este proceso igual podría hacerse en ficha perforada… pero es que, además de lento… ¡era carísimo! Las tarjetas perforadas no son reutilizables (una vez perforadas, no se pueden *des-perforar*…), mientras que las cintas sí que lo son: se pueden leer y escribir una y otra vez sin merma en su funcionamiento (al menos durante mucho tiempo).

.



Publicidad de Discos en 1977. ¡Obsérvese el precio!

Pero, con la aparición de los discos magnéticos (a finales de los cincuenta), la situación comenzó a cambiar. Porque con un disco magnético es posible algo que con una cinta no lo es: **acceder a un registro determinado de un fichero sin necesidad de leer previamente todos los registros anteriores**, simplemente dando instrucciones a la cabeza lectora de qué área del disco leer. Y lo mismo con la grabación. Esta característica abría nuevas posibilidades, que al poco se comenzaron a explotar.

Para que un programa supiera en qué dirección física del disco se encuentra la información demandada (supongamos la cuenta número 1537), **antes había que haber anotado, en algún sitio, en el momento de la escritura, la dirección física donde había ido a parar tal cuenta**.

Una solución obvia es utilizar el *propio número de cuenta como dirección* a la hora de grabar. Así, la cuenta 1537 estaría en la dirección 1537, y localizarla es inmediato.

Pero, claro, quizá la numeración de las cuentas no permita esta solución, por ejemplo, porque el propio número incluya el tipo de la cuenta, o porque tengan muchos huecos de numeración entre las cuentas (porque las cuentas adyacentes a la 1537 fueran la 1459 y la 1703, por ejemplo, con lo que se desaprovecharía muchísimo espacio). En la práctica, casi nunca puede usarse tal tipo de direccionamiento, salvo para ficheros-tabla de códigos, que tienen relativamente pocos registros y son muy estables en el tiempo.

Una forma de solucionar esto sería con la aplicación de una fórmula matemática que convirtiese el código del registro a buscar a una dirección física y unívoca. Esta técnica implica usar una **función hash**, y es muy eficiente… en los casos donde es factible usarla, que tampoco son tantos.

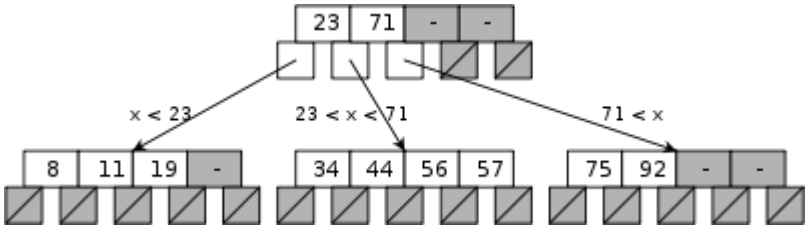
Así que rápidamente se constató que la única forma eficaz de poder conocer la dirección física de un determinado registro de un fichero era mediante la utilización de **índices**.

Un índice consiste ni más ni menos que crear, simultáneamente a la creación del fichero que se pretende indexar, otro fichero diferente al principal, que contendrá, ordenadas, las claves de acceso junto con sus direcciones. Así, tendríamos que nuestras cuentas estarían representadas, por ejemplo, con registros que podrían decir: 1459-007; 1537-008; 1703-009… Es decir, la cuenta 1459 está ubicada en el bloque 007, la 1537 en el 008, etc.

Ahora, para poder acceder a la cuenta 1537, basta con leer el fichero de índices (mucho más corto que el principal) hasta determinar la dirección de la cuenta y entonces ir al fichero principal, con un acceso directo, a dicha dirección. Es más, si

es posible, mantendremos en memoria principal el fichero de índices, agilizando muchísimo el acceso (pues evitamos siquiera leer el índice).

Pero los propios índices pueden resultar de buen tamaño, así que se crea a su vez un índice para el propio índice, y así sucesivamente hasta llegar a un tamaño de índice razonable para mantener en memoria, creando así un árbol de índices. En realidad las cosas son *un poco más complicadas* de lo que he explicado aquí, pero creo que para hacerse una idea es suficiente… y los que ya conocéis todo esto… pues eso: ya lo conocéis.



Típico árbol de índices

Ya desde el advenimiento del Cobol en 1960, se incluyó la definición y tratamiento de ficheros indexados (ORGANIZATION IS INDEXED, mediante diversas cláusulas de definición y extensiones a las sentencias de lectura y escritura), y esto permitió diseñar las primeras aplicaciones con acceso directo a la información (en inglés se llama acceso *Random*, es decir, aleatorio, pero en realidad no es aleatorio en absoluto, sino que tu programa decide a qué registro acceder en función de la información solicitada). Y las aplicaciones que típicamente necesitan acceder a información de esta forma son las **Aplicaciones Online**. Es decir, si el Online existe es ni más ni menos gracias a la existencia de los índices…

…Porque lo interesante de todo esto es que **el acceso mediante índices ha sido, y sigue siendo el método utilizado por todas las Bases de Datos de todo tipo para acceder a la información**.

Utilizando **ficheros secuenciales indexados**, se pudo empezar a desarrollar las incipientes aplicaciones online, aunque hasta que no se dispuso de Gestores de Teleproceso era realmente difícil programar estos sistemas (que, además, tenían muy poca capacidad: a los exiguos tamaños de memoria y procesador existentes, se añadía la mínima velocidad de las líneas telefónicas de entonces: punto a punto, a 1200 o 2400 *baudios* –bits por segundo-).

Una curiosidad: uno de los primeros Monitores "serios" de Teleproceso (anterior en varios años al CICS de IBM), fue el **PCL**. El acrónimo PCL significa **Programa de Control de Líneas** (en español, sí), dado que fue desarrollado en el laboratorio de IBM en Barcelona, por un equipo dirigido por el holandés Rainer Berk, bajo las especificaciones del cliente, y trabajando codo a codo con ellos, y que se instaló por primera vez en La Caixa d'Estalvis i Pensions, alrededor de nada menos que 1964.

Este programa, que fue evolucionando y mejorando con los años, se usó durante los primeros tiempos en todos los bancos españoles que empezaron a hacer pinitos con su teleproceso en la década de los sesenta y primeros setenta, pues hasta al menos 1970 ó 71 no comenzó IBM a ofrecer CICS en España (e IMS/DC es posterior en cinco años o más).

No esperéis un link a algún artículo de la Wikipedia hablando de PCL, ni algún link a algún otro sitio hablando de PCL… ¡**No hay**! Es como si *nunca hubiera existido*. Fue un hito español en informática… y prácticamente nadie lo conoce, ni se encuentra documentación, aparte de en la memoria de algunos viejos rockeros. Y, aunque no tanto, lo mismo sucede, por ejemplo, con otra gran innovación española de la época (además, ésta fue completamente española): la **Red Especial de Transmisión de Datos** (RETD), que fue la primera red mundial de transmisión de paquetes, y sobre la que también ha caído el olvido, aunque afortunadamente Jesús Martín Tardío (ingeniero de Telefónica de aquellos años) ha escrito un maravilloso relato de lo que pasó esos años gloriosos.

Ignoro si será exclusivamente tradición española la de ignorar (o peor, ¡despreciar!) las cosas buenas que se han hecho y magnificar las malas, pero indudablemente eso ocurre, y hay muchísimos ejemplos. **Uno flagrante** (que no tiene nada que ver con ordenadores):

¿Sabéis quién fue el Almirante Nelson? ¿Sabéis quizá qué pasó en la batalla de Trafalgar, en 1805? Quizá os suene de algo…

Sí, ya veo que sí. Y… ¿Sabéis quién fue el Almirante Blas de Lezo? ¿Sabéis por ventura qué ocurrió en el Sitio de Cartagena de Indias sesenta años antes, en 1741? Pues cuando lo leáis, si lo leéis, igual os lleváis una sorpresa…

Volvamos a los ficheros indexados…

Efectivamente, su capacidad de recuperar o grabar la información mediante un acceso directo, sin necesidad de leer el fichero completo, permitió el advenimiento de las primeras aplicaciones online… pero si las aplicaciones en sí eran posibles, en realidad tenían muchas dificultades para su normal explotación diaria.

Una de ellas era la dificultad para organizar los accesos concurrentes, es decir, la gestión de los bloqueos. En un proceso bancario, por ejemplo un pago de cheque, es normal acceder primero a la cuenta para ver, lo primero, si existe, y después, si tiene saldo suficiente para hacer frente al cheque, luego a los talonarios de la cuenta para comprobar que el cheque existe y no está bloqueado por algún motivo, etc. Al final de todas las comprobaciones, se marca el cheque como pagado, se actualiza el saldo de la cuenta, y se graba el movimiento para su contabilidad.

Si todo va bien, no pasa nada. Pero puede ocurrir que otra transacción que se ejecuta en la región de al lado (que para algo habíamos inventado ya la

multiprogramación), pretenda pagar un recibo de la misma cuenta en el mismo momento. Si no se controla muy bien, y se obliga a la segunda a esperar a que la primera termine, se puede montar un *carajal* de mucha consideración. Con los ficheros indexados, todo este montaje hay que controlarlo a mano, complicando enormemente la programación… en unos ordenadores que, recordad, sólo tienen unas pocas Kbs de memoria…

Pero la otra es aún más seria: *Ante un error de hardware o de programa* (que, a veces, *cascan* ¿sabéis?, por muy bien escritos que estén) *el fichero indexado se queda hecho unos zorros*. Como en realidad se mantienen dos ficheros a la vez, el de índices y el de datos, es muy posible (es más: era lo normal) que el fallo deje actualizado uno de ellos, pero no el otro… dejando el fichero inservible. Entonces, la única alternativa era asumir que el contenido del fichero principal (los datos) es el correcto, y a partir de él reconstruir los índices… utilizando el fichero en exclusiva durante el tiempo que dure el "rebuild" (y rezar porque todo acabe bien y no haya que volver a la versión anterior, perdiendo toda la sesión online hasta el fallo). O sea, parando la aplicación online mientras se recupera. Bastante inaceptable, como podéis suponer.

Charles Bachman

Otro de los gurús de la informática, **Charles Bachman**, estuvo en la génesis de la solución a todos estos problemas: el concepto de "**Base de Datos**".

De sus trabajos nació la primera Base de Datos de la historia: **IDS**, de General Electric (una de las grandes compañías mundiales, también en informática; de hecho durante los años sesenta era uno de los así llamados "*Siete Enanitos*", las siguientes siete mayores compañías de informática que competían en el mercado con la *madrastra*, IBM…, hasta que decidió abandonar el negocio en 1970, vendiendo la división de informática a otro de los *enanitos*: Honeywell).

La idea fundacional de las Bases de Datos (en realidad, de los Sistemas Gestores de Bases de Datos) era solventar todos los problemas derivados del uso aislado de los diferentes ficheros, no tanto pensando en el batch (que, en realidad, estaba resuelto sin necesidad de Base de Datos alguna), sino para dar soporte a los emergentes procesos online y de tiempo compartido.

Un DBMS debía, en primer lugar, asegurar la **coherencia de los datos en todo momento**. Esta es una *conditio sine qua non* para poder desarrollar y explotar con éxito una aplicación online. Y además debe permitir interactuar con ella a desarrolladores, técnicos de sistemas, etc con *cierta* sencillez, para facilitar el desarrollo, explotación y posterior mantenimiento de las aplicaciones. Resumiendo, un DBMS (de cualquier tipo, por cierto, incluido uno Relacional) que se precie debería, al menos, de cumplir las características siguientes:

1 Ante un fallo de un programa, debe ser capaz de recuperar la información que éste se encontró, deshaciendo todos los cambios realizados antes de su fallo. Esto implica la existencia de un log de cambios donde quedan reflejados todos los cambios efectuados por el programa, y que permiten deshacerlos para volver a la situación original.

2 Debe ser, además, capaz de **recuperar un proceso completo que haya resultado erróneo**, *aunque haya terminado aparentemente bien* (o sea, que no ha cascado). Por ejemplo, puede que un determinado proceso batch haya funcionado mal, y actualizado erróneamente muchos registros, o todos; el Gestor debe permitir recuperar el proceso completo para repetirlo una vez arreglado el programa fallón. Esto exige una gestión avanzada de copias de seguridad, coordinada con la propia gestión del log.

3 Debe proporcionar un **procedimiento robusto ante fallos del propio sistema**. En el caso de un error en una Base de Datos, debe permitir recuperar la que esté afectada, sin necesidad de afectar el resto; de esta manera se independizan unas aplicaciones de otras.

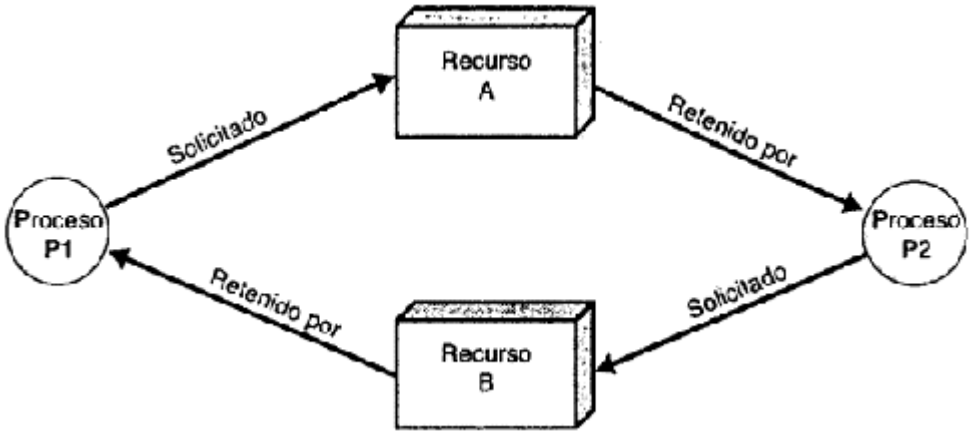
Concurrencia de Procesos

4 Debe gestionar de forma eficaz la **concurrencia entre procesos**. Esto exige un control automático y eficaz de los bloqueos: cuando un programa accede a un registro de la Base de Datos con la intención de actualizarlo, el DBMS debe

dejarlo bloqueado para cualquier otro proceso, hasta su terminación (correcta, en cuyo caso los cambios se consolidan, o errónea, en cuyo caso se deshacen).

Para lograr todo esto se requiere que el lenguaje de interfaz con la Base de Datos permita *avisarla* de que el programa tiene intención de actualizar el registro que está solicitando para lectura; que yo sepa toda Base de Datos tiene esta función (como las peticiones tipo "GHU" en IMS, o la "SELECT FOR UPDATE" en SQL).

Cuando se produce concurrencia, la estrategia correcta no es precisamente la del "*avestruz*", sino dejar en espera al segundo peticionario que llegó, hasta que el registro bloqueado quede liberado, y entonces permitirle continuar su proceso. Pero este mecanismo también tiene sus problemas…



Deadlock de procesos

5 …Porque en caso de que dos transacciones requieran registros cruzados (lo que en la literatura técnica se conoce como "deadlock", o, en *cheli*, el *abrazo del oso*), o se resuelve de alguna manera, o quedarán bloqueados ambos registros… para siempre. Un proceso P1 requiere el registro B, y lo bloquea, y después el registro A, mientras que otro proceso P2 requiere el registro A, y lo bloquea, y después el registro B (estas cosas se dan con más frecuencia de lo que podría parecer). Si hay la mala suerte de que ambos procesos, ejecutándose simultáneamente, han obtenido sin problemas sus primeros registros… tenemos un problema. Porque el proceso P1 tiene bloqueado el registro B y está a la espera del A… que está bloqueado por el proceso P2, que a su vez está esperando el registro B. Si el DBMS no reconoce el problema para ponerle solución (típicamente calzarse uno cualquiera de los dos procesos, para dejar que uno al menos, el otro, termine bien, y después relanzar el proceso abortado), podría colapsarse y requerir intervención manual, lo que no parece muy buena idea. **El Gestor de Bases de Datos debe solventar los deadlocks cuando ocurren**. Además, este caso puede darse no sólo con dos procesos, sino con tres, cuatro… lo que es bastante complicado de resolver.

6 Las bases de datos **deben proveer un interfaz** para que los programadores puedan codificar correctamente los accesos, en lectura o actualización, o la navegación entre diferentes registros relacionados; interfaz documentado y único, y a ser posible que libere al programador de todas las funciones de control, gestión de los índices, de los punteros, los backups o recoveries, etc. Deben proveer también toda una serie de programas de utilidad para todas las tareas técnicas, como reorganización, creación de índices, etc.

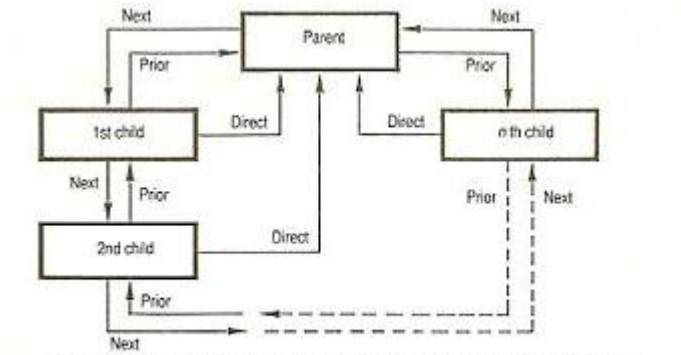
7 Y además… deben permitir al usuario (el usuario de estas Bases de Datos era el programador, no el usuario final) la **abstracción del modelo físico**: es decir, se diseña el modelo lógico de la información, cómo son las entidades de datos y cómo se relacionan entre sí, y se plasma en el propio diseño de la Base de Datos. Naturalmente, esto cambió, con el tiempo, la propia forma de diseñar las aplicaciones, dando origen al poco tiempo al nacimiento de las metodologías de Diseño Estructurado, el modelo Entidad-Relación, etc. Pero ésa es otra historia, y será contada en otro momento.

Todas estas características están resumidas (más o menos) en lo que se conoce como **ACID**: *Atomicidad, Consistencia, Aislamiento, Durabilidad*.
.

Podemos decir que **las Bases de Datos cumplieron sobradamente sus objetivos**, desde el primer momento. Doy fe. Evidentemente, las primeras de ellas no eran sencillas ni de diseñar ni de programar correctamente (bueno, si nos ponemos quisquillosos, *tampoco ahora, je, je*), pero efectivamente eran razonablemente seguras, rápidas y eficaces, cuando se rompían, se arreglaban bien y rápido, y, aunque complicadas de programar, lo eran mucho menos que si hubiera habido que hacerlo directamente con los ficheros indexados.

Programar para IMS era bastante complicado. Lo sé: yo lo hice. Lidiar con las PSBs, las PCBs, las SSAs, etc, necesita de mucha, pero mucha, atención por parte del programador (y conocimientos, desde luego), y tener siempre presente el diagrama del diseño físico de la Base de Datos.

Aún recuerdo una transacción que tenía un tiempo de respuesta normal (pequeño), y ante una (aparentemente) mínima modificación, de pronto se convirtió en la transacción que más recursos consumía de todo el Banco. Resulta que la modificación obligaba al IMS a recorrer toda una larguísima cadena de gemelos (las distintas ocurrencias de un mismo segmento) para insertar el suyo… y nadie nos dimos cuenta de ese hecho (ni el Analista -*yo*, y mira que sabía yo de IMS por entonces-, ni el programador, que sabía también lo suyo, ni el Técnico de Sistemas…). Para solucionarlo, se añadió un nuevo puntero al segmento padre que apuntara a la última ocurrencia del segmento hijo (el puntero PCL, por "Phisical Child Last")… y el programa ahora solicitaba directamente el último segmento, antes de insertar. Un éxito: la transacción volvió a ser de las más modositas de la instalación. Esta anécdota tontorrona puede ayudar a haceros una idea de lo fino que había que hilar.



A closed chain of records in a navigational database model (e.g. CODASYL), with ***next pointers***, ***prior pointers*** and ***direct pointers*** provided by keys in the various records.

Base de Datos Codasyl

A raíz de los esfuerzos de Bachman, se creó en **CODASYL** un grupo para definir las características y especificaciones que debían cumplir las Bases de Datos en Red; estas especificaciones se publicaron en 1969 y permitieron el avance de la industria de las Bases de Datos.

Aunque algunos, como IBM, habían avanzado por su cuenta, y ya tenían en el mercado la suya propia (**IMS**) desde 1968, ya que fue diseñada para poder gestionar el complicadísimo control de inventario del Programa Apolo (ése que hizo que Armstrong, Aldrin y compañía pisaran la Luna… aunque todavía hay quien lo duda).

Comenzaron a aparecer en el mercado diferentes bases de datos, que implementaron, unas más y otras menos, los estándares CODASYL: unas fueron concebidas como bases de datos en red, y otras fueron diseñadas como jerárquicas; el modelo jerárquico (que admite sólo relaciones de padre a hijo, y entre gemelos) no es más que un modelo en red restringido, y por tanto estaba también contemplado en las especificaciones CODASYL.

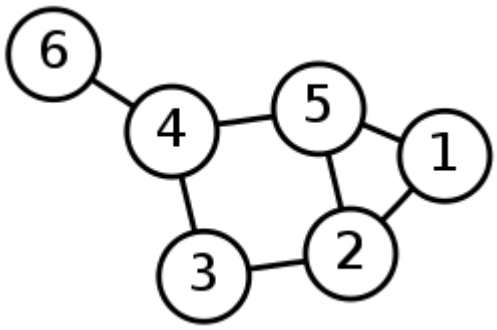


Diagrama de Base de Datos en Red

En esta página tenéis una recopilación de las Bases de Datos de todo tipo que se habían creado y/o comercializado… antes de 1986, antes de la *explosión relacional*. Hay *unas pocas*: ¡Más de 150!, y aunque en puridad algunas de ellas

no son en realidad Bases de Datos, sino más bien sistemas de acceso a ficheros, o aplicaciones escritas sobre una Base de Datos, os podéis hacer una idea del gran follón "*basedatísitico*" que había.

Casi cada fabricante importante construyó y vendió la suya, y aparecieron bastantes empresas independientes de software que construyeron y vendieron sus propias bases de datos (probablemente debieron ser de las primeras empresas que construyeron software básico sin ser fabricantes de hardware).

A la primera de todas, **IDS** (de GE, luego Honeywell), le siguieron (además de **IMS**, de IBM), **DMS1100** (de Sperry Univac), **DMS II** (Burroughs), **IDS2** (Bull), **IDMS** (Cullinet, luego Computer Associates), Total (Cincom), **Datacom** (ADR, luego Computer Associates), **Adabas** (Software AG), **System 2000** (De una tal *INTEL* -antes MRI-, que no es la misma Intel que todo el mundo conoce; yo juraría que en España la vendía Siemens, para sus ordenadores Siemens 4004, aunque no lo puedo asegurar), etc, etc, etc.

Estas tres últimas se basan en el concepto de "listas invertidas" que proporciona un excelente tiempo de acceso en lectura, aunque no precisamente en actualización, lo que las hace especialmente eficientes en entornos de sólo-consulta, con poca o nula actualización. Adabas, en concreto, sigue siendo utilizada en grandes instalaciones españolas (y de todo el mundo) después de su "*lavado de cara*" relacional de hace ya bastantes años (admite SQL como interfaz), aunque buena parte de la culpa de que siga estando vigente la tiene *Natural*, también de Software AG, el único lenguaje de Cuarta Generación que de verdad tuvo éxito, que funciona muy bien con Adabas (y con DB2, y con Oracle…), y que se mantiene funcionando en la actualidad… pero esa es otra historia, y será contada en otro momento.

Esto de las listas invertidas puede parecer antiguo, pero sigue siendo muy usado en la actualidad: por ejemplo, los buscadores que se usan en aplicaciones web, incluyendo el todopoderoso Google, utilizan variaciones de listas invertidas para realizar las búsquedas (necesitan muy buen tiempo de acceso en lectura, pero no tanto en actualización).

Desde luego, los nombres de la mayoría de Bases de Datos de la época parecían una *sopa de letras*. Tanto, que en aquéllos tiempos gloriosos, no podíamos evitar, entre los enteradillos de la profesión, contarnos el siguiente chiste (que entonces tenía mucha más gracia que ahora, por cierto):

En una tienda:

Cliente (señalando): *Déme ése. Y déme ése. Y de ése… y de ése, dos*.

Tendero (a la cajera): *Déle uno. Total, debe dos*.

…Ya, *ya dije que no tenía gracia*… Mejora un poco si lo ponemos en su contexto:

Cliente: *DMS. IDMS. IDS…IDS2*.

Tendero: *DL/1. Total, DB2*.

En fin. A *Adabas* no había forma de meterle en el chascarrillo…

Desde luego, la propia sobreabundancia de versiones de Bases de Datos de todos tipos y colores a finales de los setenta ya quiere decir mucho: Tuvieron un gran éxito, y cumplieron con las expectativas más halagüeñas. Efectivamente resolvieron aquellos problemas que vinieron a resolver: Las aplicaciones fueron mucho más fiables, los datos mucho más coherentes, el proceso de diseño de los datos, más estructurado y lógico, y ante un fallo de cualquier tipo, la información se recuperaba normalmente de forma rápida y completa, sin pérdida alguna.

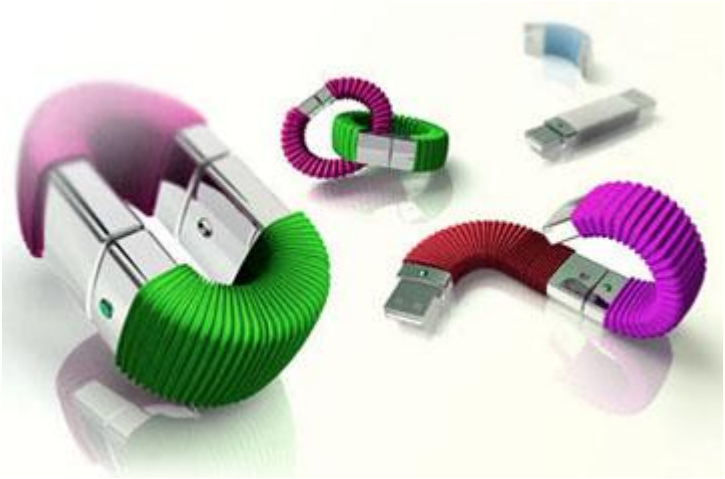
Pero tuvieron también inconvenientes, algunos de ellos bastante importantes, que abonaron el cambio de tecnología de mediados y finales de los ochenta.

En primer lugar, **cada Base de Datos era *de su padre y de su madre***. Literalmente. Cada una de ellas tenía un lenguaje de definición, un modo de ejecución, diferentes programas de utilidad, y, por supuesto, diferente interfaz para los programas de aplicación. La forma usual es resolver los accesos mediante la llamada (vía CALL, generalmente estática) a un determinado módulo del gestor (en el caso del IMS de IBM, por ejemplo, el módulo es CBLTDLI desde Cobol, ASMTDLI desde Assembler, etc), con ciertos parámetros que le indican a qué Base de Datos se quiere acceder, a qué segmento o entidad, y para hacer qué

(leer el segmento de cierta clave, o el siguiente en secuencia, o modificar el segmento con nuevo contenido, borrarlo…).

Hay que tener en cuenta que todas las Bases de Datos de la época eran Navegacionales, sin excepción, es decir, era el programador el que indicaba el orden de recuperación de la información en la Base de Datos: primero recuperar un segmento padre con una clave dada, luego recuperar la primera ocurrencia de uno de sus segmentos hijo, leer en secuencia todos estos segmentos hasta cumplir cierta condición, reescribir el segmento, insertar un nuevo segmento al final de la cadena, saltar a otro lugar distinto, y así hasta acabar el proceso, tanto en batch como en online.

Y en cada Base de Datos este proceso es distinto, pero no *un poco* distinto, sino **completamente diferente**. Migrar de una a otra Base de Datos requiere no sólo efectuar un proceso complicado de descarga de la información y carga en la nueva Base de Datos, que en realidad era lo más sencillo, sino que había que reprogramar completamente todos los accesos… para lo cual había, en primer lugar, que dar un completo Plan de Formación a todo el personal, y luego, reprogramar la Aplicación completa.



Portabilidad: brillaba por su ausencia...

En una palabra: **No existía portabilidad entre Sistemas**. Ni la más mínima. Quizá esto no fuera un problema muy serio para las instalaciones de cliente (tampoco vas a andar cambiando de Sistema a cada rato), pero era un inconveniente importantísimo para los fabricantes de Software de Gestión.

Supongamos que hemos desarrollado una Aplicación que queremos vender a muchos clientes, por ejemplo una Nómina. Inicialmente la desarrollamos en, digamos, un mainframe IBM con IMS, que para algo tiene la mayor cuota de mercado. Cuando está lista, la empezamos a vender (bueno, *antes* de que esté lista… que el marketing es el marketing).

Y aunque haya una base importante de clientes para nuestra Nómina, resulta que no todos los clientes tienen esa configuración. Los hay que tienen un mainframe de IBM, pero la Base de Datos es Total, o Datacom, por ejemplo. Otros grandes clientes tienen un Siemens 4004 con System 2000, o con Adabas. Y otros, un Bull con IDS2. Migrar la aplicación para que corra simultáneamente, con las mismas funcionalidades, en todos estos sistemas y Bases de Datos es… una locura. Migrar una aplicación de Unix a Windows o viceversa es un juego de niños, comparado con lo que era migrar de IBM/IMS a Digital/Adabas… ¡*Y eso que al menos, el Cobol sí que era el mismo*!

De hecho, la fragmentación de la tecnología, más exactamente, la diversidad de diferentes interfaces con los Sistemas, redujo muchísimo el alcance del incipiente mercado del software independiente durante las décadas de 1960, 70 y casi todos los ochenta del siglo pasado. Cosa que a los fabricantes de hardware no les preocupaba en absoluto, como es obvio: así, su mercado era cautivo. *A los fabricantes de cualquier cosa les encantan los mercados cautivos*…

Y, además de esta falta de estándares, es que **tampoco era nada sencillo diseñar y programar bien para ninguno de estos Sistemas y Bases de Datos**. En igualdad de condiciones, era mucho más sencillo que hacer lo mismo *a pedal*, directamente con ficheros indexados, desde luego que sí, pero seguía siendo difícil… mayormente porque la propia existencia de las Bases de Datos permitió la realización de Aplicaciones con una complejidad muy elevada, que hubieran sido completamente imposibles sin esas Bases de Datos.

Se requería una muy buena formación, apoyo constante en manuales (había tal cantidad de opciones y posibilidades que necesitabas consultar los formatos de las llamadas constantemente), y experiencia. Y lo primero se solventaba con cursos de formación, por caros que fueran (que lo eran), lo segundo con una buena fotocopia del original para cada uno… pero la experiencia sólo se conseguía con tiempo, y aprendiendo de las galletas, sobre todo *de las propias*, que ya sabemos que nadie escarmienta en cabeza ajena. Es decir, **los buenos técnicos (analistas, programadores, técnicos de sistemas) escaseaban**. Mucho.

Se los reconocía porque, cuando hablaban, nadie en absoluto a su alrededor que no fuera de la profesión entendía una palabra. Y a veces, ni así. Por ejemplo: ”*Hemos tenido un 0C4 en una región del VSAM, y el PTF del CAS no pudo instalarse porque la versión del HSM no estaba a nivel…*” Ya me diréis lo que entendía nadie de todo eso, y sobre todo, ya me diréis qué entendía el pobre usuario (por ejemplo, el responsable de préstamos de la sucursal de Antequera) cuando llamaba, medio llorando, para decirte que no podía abrir un préstamo hipotecario porque la máquina no le dejaba… y recibía semejante contestación.

Aquí estábamos muchos

Esta escasez de buenos profesionales trajo dos consecuencias, buenas o malas (según se mire): **Se activó el mercado** para estos profesionales (para nosotros, vaya), que comenzaron a cambiarse de empresa, explotando su conocimiento tan exclusivo y valorado, y por consiguiente **se comenzaron a mover los sueldos**, no sólo en las empresas que fichaban nuevos empleados, sino también en las que pretendían conservar los suyos… o sea, en todas. Y este movimiento salarial fue bueno para nosotros, los informáticos del momento. Ganábamos bastante dinero. Trabajábamos mucho, eso es cierto también, pero el sueldo era muy bueno, y nosotros éramos buenos, importantes, imprescindibles…

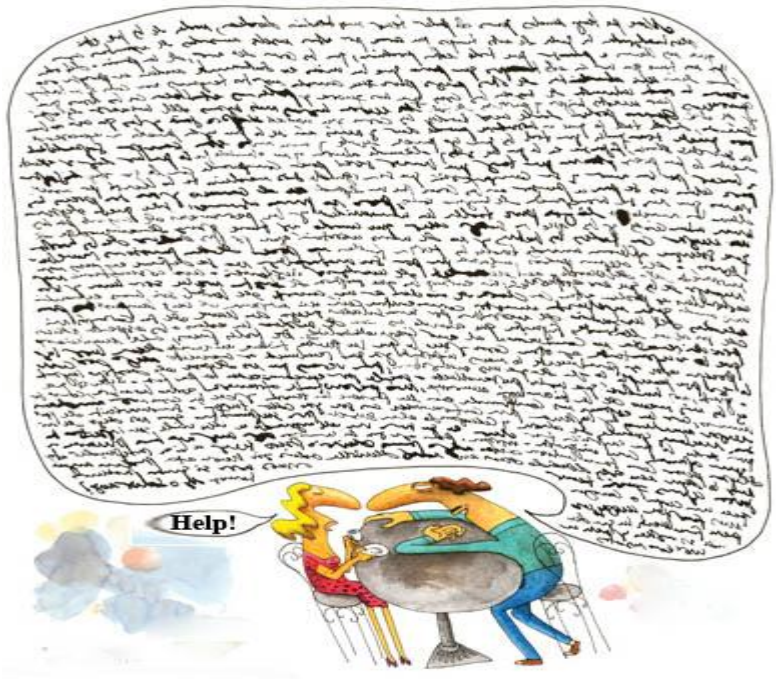
… Y nos endiosamos.

Sí, lo reconozco. Lo sé de buena tinta porque *yo también fui un dios*… Tan importantes nos sentimos, tan insustituibles, tan necesarios… que nos volvimos unos perfectos cretinos. Nos olvidamos de lo más importante (de *lo único* importante): que no somos ni más ni menos que un Departamento de Gasto, es decir, que el resto de la empresa nos ve como *Un Mal Necesario*. Y, encima, ¡caro!.

Nosotros, los informáticos, no generábamos un duro para el negocio: eran los sufridos comerciales, gestores y administrativos los que ganaban con sus ventas y su trabajo día a día el dinero suficiente como para pagar no sólo sus sueldos, sino los de los informáticos, y los de todos los demás, así como los dineros que se reparten a los accionistas.

Sigo reconociéndolo: Nos encastillamos en nuestra *torre de marfil*, y llegamos a pensar que **lo único importante** era que nuestros sistemas fueran como la seda (*desde nuestro punto de vista, eso sí*), que el tiempo de respuesta fuera bueno, que no fallaran las aplicaciones y, sobre todo, que *nos molestaran lo menos posible con peticiones, cambios, problemas y demás zarandajas*.

Las peticiones de cambio en alguna aplicación por parte del usuario, que hasta hacía poco habían sido atendidas con prontitud, comenzaron a ser sistemáticamente distraídas, retenidas, paradas, cortocircuitadas… Las excusas eran… bueno, eran para estar allí y oírlas, verbigracia:



Bla, Bla, Bla...

"*Mira*: para hacer esta modificación que pides, debo convertir un índice *Phisical-Child-First* en un *Phisical-Child-Last*, con lo que debo modificar cuarenta y cinco transacciones y treinta y siete programas batch, hacer un REORG y parar la Base de Datos una semana y media, y además las transacciones de consulta por LTERM tendrían un tiempo de respuesta mucho peor que ahora porque se produciría un encolamiento estocástico masivo, y afectaría al rendimiento ciclotímico del sistema transversal del VSAM, y bla bla bla…".

…Y el pobre usuario, compungido y desarbolado, te pedía perdón por habérsele ocurrido semejante proposición deshonesta, te invitaba a un café como desagravio… y se volvía a sus Cuarteles de Invierno completamente frustrado, porque en realidad él o ella sigue pensando que poder sacar los movimientos de las cuentas ordenados por fecha de valor en vez de por fecha de operación no debería ser tan complicado…

Las aplicaciones se complicaban más y más… costaban más y más… y cada vez estábamos más lejos de los usuarios. Los árboles no nos dejaban ver el bosque. *Y al final lo pagamos*. Pero ésa es otra historia, y será contada en otro momento.

Yo creo que de esta época (principios y mediados de los ochenta) fue cuando empezó a aparecer ese *sano odio* que la mayoría de usuarios de las empresas (no sólo las grandes) tenían y siguen teniendo a los informáticos. Claro que a lo mejor es completamente natural el sentimiento de unos y de otros, porque es cierto que hoy por hoy, sin su informática, casi ninguna empresa sería viable, pero *más cierto aún es que, sin su empresa, casi ningún informático sería viable*.

Yo, por mi parte, siempre intenté dar en lo posible el mejor servicio a mis atribulados usuarios, ponerme en su lugar, y mantener las aplicaciones lo más adecuadas y fáciles de utilizar posible, eso que ahora se llama *usabilidad*. Pero también he de entonar el *mea culpa*, pues soy culpable de una buena parte de los pecados descritos. Muchos años después, he de reconocer que una parte de los de la profesión sufrimos (en *pasado*) y aún sufrimos (en *presente*) esta enfermedad.

Si mis pobres palabras sirven para que algunos de vosotros, queridos lectores informáticos, reflexionéis un poco sobre esta circunstancia, habrán cumplido su misión. **Y os pido humildemente perdón, si en algo os he ofendido**…

Volvamos ya a las Bases de Datos, que esto va de Base de Datos, no de *golpes de pecho…*

.

Ya a principio de los setenta, había algunos adelantados que pensaban que **esta generación de Bases de Datos podría abocarnos, con el tiempo, a un callejón sin salida**.

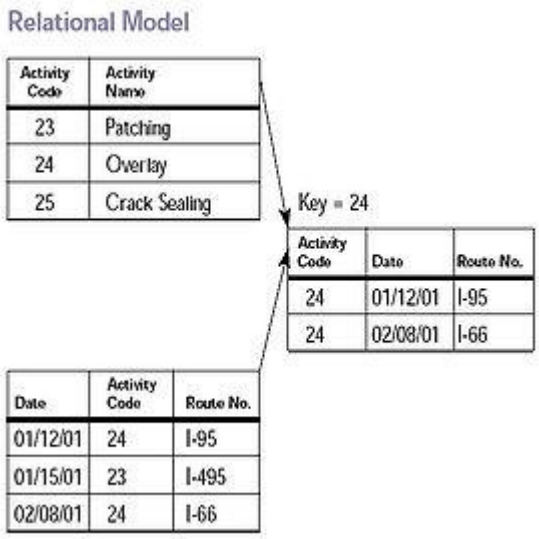
Era necesario un **método de diseño más sencillo**, pues cada vez más y más aplicaciones se escribirían en todo el mundo y no era conveniente que todas y cada una de ellas necesitaran unos técnicos especializados y muy expertos, cada uno de ellos en un producto diferente e incompatible con el resto. Era necesario **solventar la falta de compatibilidad entre aplicaciones** mediante un interfaz común. Era necesario encontrar alguna cosa, algún sistema para facilitar **la elección del mejor camino para recuperar la información pedida** sin necesidad de tener que conocer de antemano (y programar cuidadosamente) el mejor camino para hacerlo, es decir, realizar la navegación de forma automática. Era necesario que **un simple cambio en la definición de la Base de Datos no obligara a modificar todos los programas** que acceden a ella. Era necesario, por fin, **depender cada vez menos los informáticos**.

En una palabra, era necesario ***volver a acercar la tecnología al negocio***, porque en los últimos tiempos ambos se estaban separando cada vez más, y podrían llegar a perderse de vista.

Ahora miremos la historia del advenimiento de las bases de datos relacionales, cómo comenzaron y cómo lograron con los años el cuasi-monopolio que ostentan en la actualidad.

¿En qué consistían las ideas de Codd?

En primer lugar, como ya he comentado, que el diseño debe ser realizado de manera natural, utilizando la más sencilla posible de las representaciones de los datos, es decir, **los ficheros planos,** los de toda la vida. Nada de complicadas estructuras reticulares ni jerárquicas: Basta con la Relación de todos los datos necesarios para la Aplicación: un compendio de todos los campos a gestionar en una sola Lista de Datos. De aquí viene el propio nombre de "*Relacional*", puesto que se basa en Relaciones o listas de datos.



Un modelo relacional

Y además, la recuperación (toda la gestión, en realidad) de la información debería atenerse a las **normas formales del Álgebra y del Cálculo**. Así que describe el **Álgebra Relacional** aplicable a sus relaciones (que existía ya, pero no había sido muy explorada), que permite realizar operaciones con Relaciones. Estas operaciones básicas son:

Selección, que permite obtener un cierto número de filas (*tuplas* en la jerga relacional) de la Relación original;

Proyección, que permite obtener un determinado número de datos -columnas- de una determinada Relación;

Producto Cartesiano, que permite obtener todas las combinaciones resultantes de la mezcla de dos Relaciones con columnas diferentes (pero con clave idéntica);

Unión, que permite obtener una única relación que contenga todas las filas de dos o más relaciones con las mismas columnas;

Diferencia, que permite obtener las filas de una relación que no tienen correspondencia en otra relación, es decir, las que están en una, pero no en la otra; y

Renombrado, añadida posteriormente por cuestiones formales, que permite cambiar el nombre a una columna de una relación.

Si os acordáis de la Logica de Conjuntos, quizá echaréis en falta la **intersección**, pero es el resultado de aplicar dos diferencias consecutivas: si los conjuntos son A y B, la intersección de A y B se calcula como (A – (A – B)).

En definitiva, el Álgebra Relacional queda definida como un subconjunto de la Lógica de Primer Orden. Y el Teorema de Codd establece la correspondencia entre el Álgebra Relacional y el Cálculo Relacional.

¡**Menudo rollo que os he soltado**! Pero es que tenía mis razones: estando como está toda la Teoría Relacional gobernada por el Álgebra Relacional (o el Cálculo, que es lo mismo), y en definitiva, con la Lógica clásica y por tanto, con el bien conocido **Cálculo de Predicados**, *es posible establecer procedimientos puramente matemáticos para obtener el resultado de una petición a partir de la o las Relaciones Originales y la enumeración de los resultados deseados*.

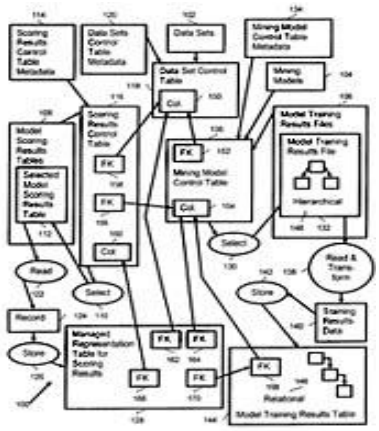
¡El programador automático!

En una palabra: sería factible **construir un programa que, dados los resultados deseados (o sea, las especificaciones funcionales), y la estructura de la Base de Datos, calcule *matemáticamente* el mejor camino para obtener la información**… eliminando, con ello, uno de los puntos más espinosos de la programación de las Bases de Datos tradicionales: la necesidad de que un programador avezado escribiera el código exacto para optimizar los accesos.

En efecto, se trata de esa procelosa e insondable pieza de código que conocemos por el nombre de **Optimizador**. Pero no nos adelantemos…

Porque, incluso en 1970, era evidente que almacenar *toda la información* de una gran Aplicación en un solo fichero físico era entonces (como lo sigue siendo ahora) una auténtica locura: la gigantesca redundancia de información que se produciría haría inviable cualquier proceso posible.

Así que Codd empezó a describir Reglas para Normalizar Bases de Datos Relacionales. Ya en 1971 había formalizado la primera, y tan pronto como en 1973 estaban ya formalizadas las **Tres Primeras Formas Normales**, que es adonde todo el mundo llega normalmente en el proceso de diseño (o donde todo el mundo se queda, según lo veáis).



Proceso de Normalización

En 1974, entre Codd y Raymond Boyce definieron la así llamada *Forma Normal de Boyce-Codd*, y posteriormente se definieron la *Cuarta*, la *Quinta* y la *Sexta Formas Normales*, esta última tan tarde como en 2002. De ser sincero, no me preguntéis para qué sirven, ni si tienen mucha aplicación: en mi aburrida vida de diseñador de aplicaciones comerciales, jamás me he visto en la necesidad de llegar más allá de la famosa Tercera Forma Normal…

Aunque IBM (la *Madrastra* de los *Enanitos* de principios de los años 70) no tuvo el menor interés en abrir otro melón (bastantes tenía ya abiertos por entonces), el amigo Codd era un tipo *machacón*: con el tiempo consiguió que IBM arrancara su primer proyecto relacional, el abuelo de todos ellos: **System/R**.

Sin embargo, por algún motivo ignoto (quizá porque resultaba demasiado "*matemático*" para las entendederas de los programadores de la época, yo el primero, porque, desde luego, por tema de patentes no era), en lugar de utilizar el Lenguaje **ALPHA** que Codd había diseñado, inventaron otro diferente, de nombre **SEQUEL**, que años después, y debido a que el nombre *SEQUEL* estaba registrado ya, cambió su nombre a **SQL**, nombre que quizá os suene de algo.

Para los curiosos, Chris Date, el otro gran divulgador de la tecnología relacional, publicó en 1999, en Intelligent Enterprise, dos artículos sobre el lenguaje ALPHA: éste es el primero, y éste el segundo.

System/R terminó con cierto éxito, no porque fuera muy eficiente, que no lo era, sino porque demostró que *era factible* construir un Gestor de Bases de Datos Relacionales con un buen rendimiento en entornos transaccionales. Y muchas de las decisiones de diseño que se tomaron entonces han sido adoptadas por todas las Bases de Datos Relacionales desde entonces.

Por ejemplo, la decisión de almacenar la propia definición de las Bases de Datos (Tablas, Índices, etc, en definitiva, los metadatos) en unas tablas más (el famoso **Catálogo**), y acceder a ellas como si fueran unos datos cualesquiera, es de System/R, y todos sus seguidores siguieron la misma solución, a pesar de que desde el punto de vista del rendimiento es una solución espantosa; en IMS, por ejemplo, las definiciones de las Bases de Datos, índices, accesos, etc, se compilan (en realidad, *se ensamblan*), generando un código compacto y eficiente que es el usado por el Gestor.

Y otra decisión que tuvieron que tomar los diseñadores de System/R fue cómo realizar el interfaz entre una petición de información a la Base de Datos y los programas que usan esta información. El motivo es que la propia lógica relacional es *per se* una *Lógica de Conjuntos*, y por tanto, cada query puede devolver cero filas, o una… **o muchas**, incluso todas las filas de una tabla (o conjunto de tablas). Y, claro, los ordenadores siguen siendo unas humildes y escalares máquinas *von Neumann*, que tienen la costumbre de tratar secuencialmente los datos, y no todos a la vez, de golpe.

En una palabra, había que hacer convivir el hecho de que las queries relacionales son del tipo **Set-At-A-Time**, mientras que las máquinas eran, y siguen siendo, del tipo **Record-At-A-Time**… Y se inventaron la técnica de los **Cursores**, con todas sus sentencias asociadas (DECLARE CURSOR, FETCH, OPEN, CLOSE…), que permiten tratar un conjunto de filas (o de datos) como si se tratara de un fichero secuencial mondo y lirondo. También los Cursores están presentes en prácticamente todas las Bases de Datos Relacionales de hoy en día.

System/R incluso llegó a instalarse en algún cliente, sobre 1977-78, pero tuvo escaso éxito, debido a su no menos escaso rendimiento en las máquinas de la época.

También había *vida relacional* fuera de los esfuerzos (no demasiado esforzados todavía) de IBM. En la Universidad de Berkeley (California) habían asumido los planteamientos de Codd, y comenzaron un proyecto de investigación, que dio origen, a principios de los ochenta, a la Base de Datos Ingres. Fue un proyecto bajo licencia BSD, y por tanto, por una pequeña cantidad se podía adquirir el código fuente y reutilizarlo. Buena parte de las Bases de Datos de hoy en día

deben mucho a Ingres (Sybase, Microsoft SQL Server y NonStop SQL, entre otras).

Y mientras tanto, IBM seguía trabajando en las secuelas de System/R. En 1982 sacó al mercado la Base de Datos **SQL/DS**, exclusivamente para DOS/VSE (VSE es un Sistema Operativo de IBM "competidor" de MVS, y heredero de DOS, el primer sistema operativo basado en disco de IBM; todavía hoy en día siguen existiendo instalaciones funcionando en VSE, y tan ricamente…).

Larry Ellison, fundador de Oracle

A partir de las especificaciones públicas de SQL/DS, Larry Ellison (fundador de Relational Software) y un equipo de ingenieros (entre los que había alguno que participó en el proyecto System/R) diseñaron una Base de Datos llamada **Oracle**, que comenzó a ser vendida alrededor de 1981 u 82, en principio para Digital/VAX, aunque la primera versión "utilizable" se vendió a partir de 1983-84, los mismos año en los que IBM lanzó **DB2**, el equivalente de SQL/DS, pero para MVS (sólo en entorno mainframe, desde luego). El nombre "*DB2*" se eligió para enfatizar la idea de que IMS era *la primera* Base de Datos (DL/1 era el lenguaje de tratamiento de la información), y DB2 era *la segunda* (y mejor) de ellas…

Esta versión DB2 v1.0 teóricamente funcionaba… siempre que las tablas tuvieran algunos cientos de filas, no se hicieran muchos joins (mejor, ninguno), y no hubiera que recuperar nada ante fallos. En una palabra, era completamente inservible para Aplicaciones en Producción. Larry Ellison fue más listo: la primera versión de Oracle lanzada al mercado era la 2.0… aunque le pasaban cosas muy parecidas.

A la versión 1.0 de DB2 siguió un año después la v1.1. Esta fue la primera versión que probamos en el banco en que trabajaba por entonces. La presión comercial de IBM (y las facilidades de apoyo que ofrecía en la instalación y prueba) hizo que estas primeras versiones se instalaran en muchísimas de las instalaciones españolas de MVS en aquellos años tempranos (86-87-88).

La conclusión general es que era un producto que *prometía*, pero que no se podía usar para Producción real.

Un ejemplo que yo viví: una de las pruebas que hicimos fue pasar un proceso de actualización a una tabla de algunos miles de filas, que actualizaba todas las filas de la tabla, y a mitad del proceso, cascar deliberadamente el programa (le hacíamos dividir un número por cero, cosa que a casi ningún ordenador le gusta demasiado; en MVS obtienes un hermoso *abend S0CB*). La Base de Datos debía, entonces, de tirar de log y recuperar todos los cambios, para dejar la tabla como al principio del proceso. Y eso fue lo que *comenzó* a hacer… Un par de horas más tarde, aburridos, cancelamos también el recovery… y entonces *comenzó a recuperar la propia recuperación*… hasta que otras tres horas después, paramos todo el DB2, que de todos modos se había quedado poco menos que frito, y lo mandamos todo a hacer gárgaras. Desde luego, muy estable no era.

Chris Date

Así que nadie instaló tampoco la versión 1.1. Pero había gran ruido e interés en esta nueva tecnología. Además de los esfuerzos de los fabricantes por convencernos de las supuestas bondades de la *cosa relacional*, hay que resaltar la enorme labor evangelizadora de Chris Date, dando conferencias por todo el mundo (en España, también), escribiendo libros, artículos… Una buena parte del éxito inicial es debido a él.

Tanto interés había, que el Grupo de DB2 de GUIDE se convirtió en el más numeroso con diferencia de todos los grupos de GUIDE (en España, al menos). Para los que no sepáis qué es GUIDE (o sea, casi todos), se trata de una Asociación de Usuarios, auspiciado por IBM, en el que los usuarios participan, junto con técnicos de la propia IBM, para poner en común las experiencias que cada cuál tiene con los productos de IBM, de hardware y de software: MVS, CICS, IMS…, así como hacer peticiones sobre funcionalidades de las nuevas versiones, etc. Cuando se constituyó el Grupo de DB2, casi todo el mundo se apuntó (pues,

en la práctica, todas las grandes instalaciones sabíamos que, tarde o temprano, *acabaríamos con uno o varios DB2 en nuestra instalación*).

El motivo de tanta expectación era, sobre todo, la esperanza de que por fin íbamos a poder disfrutar de una herramienta con un **interfaz común** (el SQL), que permitiría migrar aplicaciones de entorno con facilidad y sencillez (aunque no se hiciera casi nunca) y, sobre todo, que **permitiría reaprovechar** (¡*por fin*!) **la formación de los técnicos**: una vez sabes SQL, puedes manejarte con cualquier Base de Datos Relacional… más o menos.

Porque nunca nos creímos del todo eso de que el Optimizador sería cada vez más listo y sería capaz siempre de encontrar el camino más eficaz para resolver la query sin necesidad de *ayudarle*. Y sabéis que, al menos hasta ahora, veinte años más tarde, teníamos razón: los Optimizadores son cada vez más listos… y siguen metiendo la pata con demasiada frecuencia como para poder dejarlos siempre solos; de ahí que en ocasiones haya que darles "hints" para orientarles.

IBM comenzó también a dar formación para los incipientes "probadores", y así hasta que a principios de 1987 IBM sacó al mercado la versión v1.2. Esta versión ya funcionaba razonablemente bien, pero aún tenía un problema: ¡no hablaba con IMS/DC, sólo con CICS! Y esto era un serio inconveniente para aquéllas instalaciones (la más grandes) que teníamos IMS no sólo como Base de Datos, sino como Gestor Transaccional… porque nadie estaba dispuesto a cambiar el robustísimo IMS por un CICS que, en aquellas épocas, era mucho menos potente, más inestable y tenía un rendimiento bastante inferior.

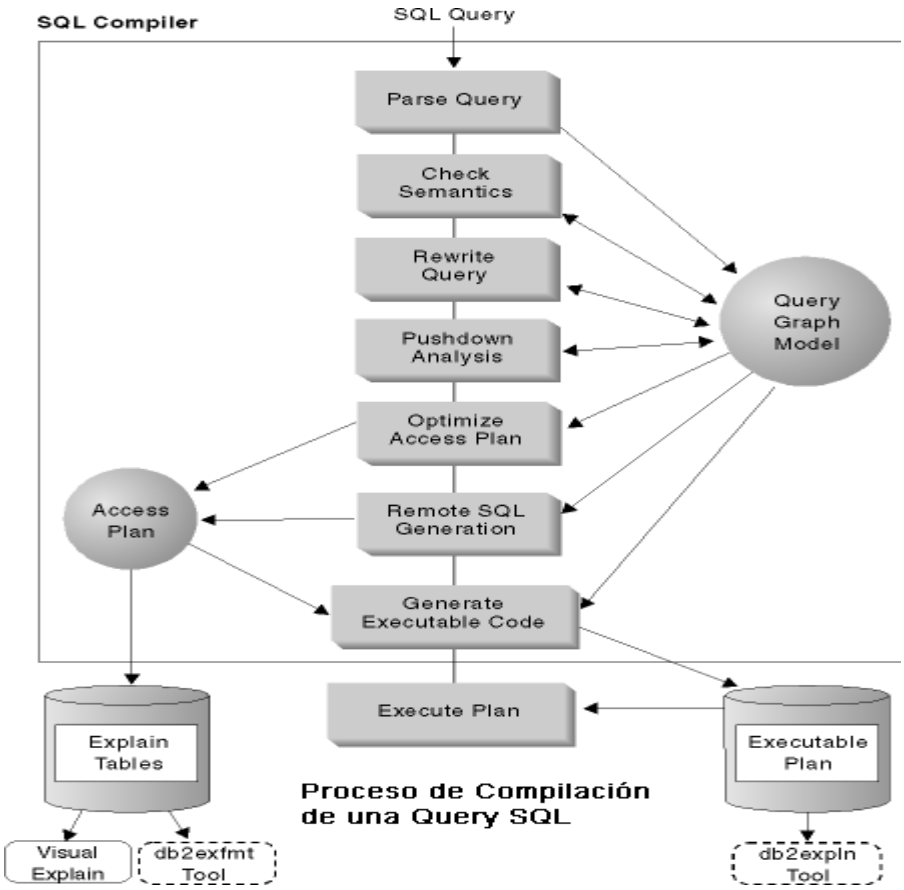
Y por fin, a fines de 1987 o principios de 1988, IBM publicó la versión v1.3, que (además de seguir mejorando su funcionalidad y su rendimiento), por fin era compatible también con IMS. Seguía teniendo importantes lagunas: por ejemplo, la compilación de las queries tardaba muchísimo y además producía espeluznantes encolamientos en el acceso al Catálogo; el rendimiento de los joins seguía siendo malo (como ahora, vaya); un recovery de un proceso batch en base al log seguía tardando mucho (aunque al menos ahora siempre acababa bien)… pero obviando estos problemas conocidos, que podían solucionarse a base de normativa (prohibiendo *por Decreto* hacer lo que se sabía que no iba a ir bien), daba un rendimiento suficientemente bueno como para poder confiar en ella.

Ese fue el momento en que, en el plazo de unos pocos meses, muchas empresas (la mía entre ellas) tomaron la decisión de **comenzar a usar DB2 "en serio"**, es decir, para las nuevas aplicaciones que se comenzaban a escribir… y para ello, hubo que preparar extensísimos Planes de Formación para Analistas, Programadores, Técnicos de Sistemas… El Departamento de Formación de IBM quedó desbordado (no había tantos buenos formadores en algo nuevo, como era DB2), y los clientes tuvimos literalmente que hacer cola, porque no había otra alternativa: ninguna empresa de formación informática tenía entonces conocimientos de DB2 (ni de ninguna otra Base de Datos Relacional, diría yo) como para poder formar a nadie. Y claro, cuando una gran empresa decidía entrar

en una tecnología básica como ésta debía dar formación igual a cuatrocientas o quinientas personas…

Y sobre 1988 comenzamos por fin a escribir las primeras aplicaciones usando DB2. Y hubo que reescribir buena parte de las normas de nomenclatura, diseño de programas, etc. Y hubo que asegurarse que todas las incipientes aplicaciones que se escribían para DB2 fueran de una mínima calidad, y que no hacían la cosas que se sabía que no iban bien… Por cierto, muchas de aquellas prohibiciones de fines de los ochenta siguen vigentes hoy en día en muchas instalaciones, a pesar de que lo que entonces iba mal ahora ya va bien… y es que ¡*mira que nos cuesta trabajo cambiar a los informáticos*!

Así que fue el terreno abonado para la tercera gran explosión tecnológica de la época: la adopción de las Arquitecturas, las Metodologías de Desarrollo y las Herramientas CASE, cosas muy interesantes de las que hablaremos en los próximos capítulos.



Proceso de Compilación SQL (DB2)

Sólo unos breves apuntes sobre la compilación de programas que accedan a DB2: es posible, desde luego, lanzar una query desde un programa Cobol (o lo que sea) y que el Optimizador la reciba, la compile, hallando el mejor camino para su

resolución, y luego la ejecute. Pero es un proceso costoso (entre otras cosas, debido a que toda la información acerca de las propias Bases de Datos está contenida en otras Bases de Datos), que provoca una fuerte concurrencia sobre el Catálogo, y por tanto no es nada aconsejable hacer "**SQL Dinámico**", ni en batch ni tampoco en online.

Así que IBM inventó para DB2 el "SQL Estático", es decir, **las queries se compilan y optimizan en batch**, mediante un proceso llamado "**BIND**", que compila cada query, calcula el mejor camino a usar, en función de la propia query, del número de filas de cada tabla, la distribución de las claves, los índices existentes y lo desorganizados que estén… y esa información del "mejor camino", denominada **Plan**, se almacena en el propio catálogo y se usa siempre… para bien o para mal. Técnica ésta, la del SQL Precompilado, que, a pesar de que según la teoría no es recomendable en absoluto, casi todas las Bases de Datos actuales utilizan… por algo será.

El proceso de "BIND", pues, se convierte en una parte más de la compilación de un programa que acceda a DB2, que queda, por tanto, con cuatro pasos (por lo menos): **Precompilación DB2** (este paso convierte las llamadas a DB2, con "EXEC SQL", en llamadas a módulos internos de DB2 que resuelven esa petición particular); **Compilación** (por ejemplo, Cobol, que genera el programa objeto); **Linkedit** (genera el programa ejecutable, uniendo todos los módulos llamados por el principal, incluidos los de DB2), y **BIND** (que obtiene el mejor camino para cada query). Si además tiene CICS, tendrá también su propia Precompilación (que en caso de IMS no es necesaria). En una palabra, la compilación comienza a ser un proceso pesado y muy consumidor de recursos…

…Pero este sistema soluciona el problema… siempre que las condiciones de las tablas no varíen, al menos en exceso (que se creen nuevos índices o se borre alguno existente, que la tabla se desorganice, debido a las inserciones y borrados reiterados, o, simplemente… que crezca o decrezca el tamaño de las tablas). Cuando algo de esto ocurre, se debe hacer "REBIND" de todos los Planes que afecten a la tabla, para permitir a DB2 calcular de nuevo el mejor camino… o no, pero bueno.

Pero también era posible acceder a las tablas DB2 directamente desde TSO, sin necesidad de programar, mediante dos productos específicos: **SPUFI** y **QMF**, que permiten a los usuarios finales lanzar sus propias queries contra las Bases de Datos, y obtener los resultados; estos dos productos, sobre todo QMF, fueron pilares básicos para la introducción, a fines de los ochenta, del concepto de **Centro de Información**, del que algo comentaremos cuando en la serie le toque el turno al artículo sobre el Data Warehouse.

IBM concentró sus esfuerzos *relacionales* de la década de los ochenta en el mundo mainframe… donde arrasó (y también en el mundo AS/400, para el que sacó al mercado DB2/400, base de datos que comparte con DB2 el nombre, casi

todo el interfaz SQL… y poco más). No creo que hoy en día subsistan muchas Bases de Datos Relacionales en entorno mainframe que no sean DB2…

Pero en los ochenta, IBM dejó de lado el *resto de los mundos* (de hecho, recordad que hasta 1990 no anunció IBM su entrada en el mundo UNIX, de la mano de la gama RS/6000).

Y esos mundos fueron aprovechados por pequeñas empresas independientes, que lanzaron rápidamente productos que funcionaban en VMS (el Sistema de los VAX de Digital), en UNIX, en OS/2, e incluso en MS/DOS (yo tuve la oportunidad de probar la versión de Oracle para MS/DOS allá por 1990 o así: os podéis imaginar que no servía para nada, aunque las queries las hacía, y las hacía bien… siempre y cuando las tablas no tuvieran más allá de quince o veinte filas).

Estas compañías fueron, sobre todo, Relational Software (con su producto **Oracle**) aunque pronto cambió su nombre a *Oracle Corp*.), **Informix** (acrónimo de Information on Unix), y **Sybase**, fundamentalmente, aparte de la propia Ingres, que sacó al mercado su versión UNIX a mediados de la década.

El gran avance de todas estas Bases de Datos lo constituyó el hecho de que todas ellas tuvieran el mismo (o casi el mismo) interfaz, que copiaron del que IBM inventó (pero *no patentó*, o quizá *no pudo* patentar: de ahí su gran éxito, como ocurrió con los PC's): **SQL**. Por una parte les ahorró costes de diseño y desarrollo; por otra parte les ayudó comercialmente a crear "masa crítica"; todas eran compañías pequeñas, que un buen día podían quebrar… pero la inversión realizada por los clientes podría ser fácilmente transportable a otra de las que sobrevivieran; este hecho facilitó la toma de decisiones a favor de esta tecnología relacional.

Yo creo que IBM se equivocó de estrategia, al dejar durante unos años desatendido este mercado (bueno, no sólo yo, también IBM lo cree… *ahora*), que creció mucho más rápido de lo que pudieron prever. Y ahí está ahora Oracle, por ejemplo, como una de las mayores compañías de tecnología de mundo (y mayor que IBM, de hecho), que además hace sólo unas pocas horas (hoy, 20 de abril de 2009) que ha anunciado que compra a Sun, entrando de lleno, por primera vez, en el negocio del hardware…

Este no fue, desde luego, el único error de estrategia que IBM cometió… pero ésa es otra historia, y será contada en otro momento.

Fueron apareciendo, además, otros nichos de mercado que IBM no atendía: en el mundo de los sistemas tolerantes a fallos (ahora casi todos los sistemas importantes tienen una gran tolerancia a fallos, pero no entonces) Tandem se hizo con una buena parte del mercado de los Sistemas Críticos que no podían dejar de funcionar por un fallo de un componente cualquiera. Por cierto, IBM tenía también, cómo no, oferta en este mercado: el IBM System/88, que en realidad eran ordenadores Stratus, vendidos por IBM como OEM… aunque fue Tandem quien lideró el mercado esos años. Y Tandem puso en el mercado NonStop SQL (a partir de la base de Ingres), Base de Datos Relacional para sus sistemas tolerantes a fallos.

Otro nicho lo ocupó otra pequeña compañía, Teradata, que comenzó a vender la primera **Máquina de Base de Datos**, es decir, un hardware diseñado específicamente para obtener el máximo rendimiento en accesos a los discos magnéticos, y una Base de Datos especializada, que permite usar eficientemente el hardware, paralelizando accesos, e incrementando en mucho su velocidad, sobre todo en lectura, y, de paso, permitiendo incrementar el espacio en disco disponible en los mainframes, ya que Teradata tiene desde su creación una conectividad excelente con estos mainframes: se conecta vía canal, como un dispositivo externo más, y para MVS se trata de una cinta magnética… por lo que instalarlo no tiene apenas trabajo que hacer en el sistema principal.

Teradata 5400H

Teradata, de todos modos, encontró su nicho de mercado (bueno, en realidad prácticamente *lo creó*) cuando a mediados de los noventa comenzó a ponerse de moda (mejor, *a necesitarse* por las compañías) el concepto de Data Warehouse. Pero ésa es otra historia, y será contada en otro momento.

Y por fin, otras bases de datos que no eran ni mucho menos relacionales comenzaron a añadirles un interfaz relacional, para adaptarse a los nuevos tiempos. Datacom y Adabas fueron dos de ellas. El rendimiento era peor, sin duda,

que utilizar su interfaz tipo CALL de toda la vida… pero así tenían más *appeal* para su venta, basándose en la portabilidad de aplicaciones, etc. No les fue mal del todo: ambas siguen existiendo en nuestros relacionales días, que es mucho decir.

En una palabra, aparecieron multitud de Bases de Datos Relacionales, pseudo-Relacionales o no-Relacionales-con-interfaz-SQL, con lo que la confusión del mercado creció y creció… Así que nuestro buen Codd, siempre velando por la *pureza relacional*, se aprestó a sacarnos de dudas: publicó sus famosas **12 Reglas**, para poder discernir las Bases de Datos Relacionales *de verdad*, de las que sólo *decían que eran* relacionales, pero no lo eran (para los que tengáis curiosidad, *ninguna* Base de Datos de principios de los noventa cumplía con las 12 reglas, ni tan siquiera DB2).

Ignoro si el cumplimiento o no de las reglas famosas se tuvo en cuenta a la hora de hacer la selección de una u otra Base de Datos para un proyecto concreto por una empresa concreta. Pero sí que se hicieron estudios sobre cuál era la que más o la que menos cumplía con las reglas.

Recuerdo que, a principios de los noventa, cierta Universidad madrileña anunció, en el entorno del **SIMO**, una charla en la que nos explicaría un estudio completísimo que habían realizado sobre cuál era la mejor Base de Datos Relacional. Aprovechando que por aquella época aún iba yo al SIMO todos los años (hace ya muchos que no, y este último año 2008 no ha ido al SIMO *ni siquiera el propio SIMO*…), fui a la charla de conclusiones. Acudimos muchos profesionales, pensando… "Vaya, por fin un estudio *serio, independiente y español* sobre Bases de Datos, ¡pardiez, qué interesante!".

Pues no.

Una encuesta...

Resulta que habían realizado una encuesta a los doce o catorce vendedores de RDBMSs que entonces había en España, mediante un cuestionario, en el que éstos contestaban cuál era el grado de cumplimiento de su Gestor sobre las doce reglas de Codd. Por ejemplo: "¿*Su Base de Datos mantiene un Catálogo Dinámico*

Online basado en el modelo relacional?" Conteste Sí, No, o *según el día*", y así con todo.

Y ya.

…Y en base a las respuestas, con un sencillo proceso de tabulación (porcentajes de cumplimiento y poco más, nada de estadísticos complejos), llegaron a la conclusión de que la mejor de todas, la *chachi piruli*, era CA/Universe (de Computer Associates, que por entonces estaba empeñado en fusionarla con la otra Base de Datos de su propiedad: Datacom, aunque ignoro si llegaron a terminar la fusión alguna vez…).

Yo (que siempre he sido un poco *preguntón*) no pude resistirme, e hice dos preguntas en el turno de Ruegos y Preguntas, sólo dos:

Una: "¿Han hecho algún tipo de prueba de funcionamiento en la práctica?" *Respuesta*: "**No, ninguna**. La pobrecita Universidad no tiene fondos suficientes, *tralarí, tralará*…"

Dos: "¿Han utilizado, al menos, los manuales técnicos de las Bases de Datos para hacer la evaluación?" *Respuesta*: "Uy, no, menudo trabajo, pues vaya, todo en inglés… **Nos hemos basado exclusivamente en las respuestas que los fabricantes han dado a las doce preguntitas del cuestionario**. Es que la Universidad tiene pocos recursos, ya sabe, *tralará, tralarí…*"

Y no hubo una pregunta *tres*: me levanté y me fui; a ver qué iba yo a hacer allí perdiendo el tiempo… Sinceramente, me dio mucha pena. Cuando lo comparo con la Universidad de Berkeley, arrancando un proyecto de investigación y creando de la nada Ingres… pues eso, que me da mucha pena. Y no creo yo que las cosas hayan mejorado mucho desde aquellos años hasta ahora, desgraciadamente…

Bueno, para acabar esta historia que se está alargando mucho (*como siempre*, pensaréis) IBM entró por fin en el mercado de Bases de Datos Relacionales para UNIX… perdón, para AIX, que era el UNIX que IBM se inventó para su gama RS/6000, y sacaron, sobre 1991 o 1992, **DB2/6000**. Que no tenía nada que ver con DB2 (el del mainframe) ni con DB2/400, menos el nombre y la mayor parte del

interfaz. La versión mainframe fue durante mucho tiempo responsabilidad del Laboratorio de Santa Teresa (California), mientras que DB2/6000 lo era del Laboratorio de Toronto (Canadá)… así que, para no perder la costumbre, los laboratorios parecían peleados, y los planes de desarrollo de una y otra Base de Datos eran no sólo distintos, sino en ocasiones, contradictorios.

Este DB2/6000 con el tiempo se convirtió en lo que es hoy: **DB2 Universal Database**, y ya funciona en todos los sabores, olores y colores de UNIX, OS/2, Linux, supongo que Windows, PDAs, etc… menos en mainframe, donde sigue funcionando una cosa llamada *DB2*, sí, pero que no tiene mucho que ver con su *Universal* primo.

… Y colorín, colorado, las Bases de Datos Relacionales se hicieron con el mercado.Y esta entrada, por cierto, se ha acabado (que *ya es hora*).

Artículo publicado por : Macluskey

Portal El cedazo : http://eltamiz.com/elcedazo/2009/04/13/historia-de-un-viejo-informatico-el-camino-hacia-las-bases-de-datos-relacionales/