Breve introducción a la gestión de Datos

Las compañías que hace décadas fueron incorporando tecnología a sus procesos, fueron de a poco recopilando, procesando y almacenando datos. Esto al principio se ha hecho en forma libre (ad hoc), cómo mejor podían los técnicos de ese momento. De este simple hecho se deduce el concepto de almacén de datos, de base de datos.

Pero yendo a la definición, ¿qué podemos considerar *"dato"*? Según Wikipedia, un dato es una expresión mínima de contenido sobre un tema. Vale decir, es una expresión atómica (indivisible) acerca de una propiedad de *"algo"*, por ejemplo un dato es que Carlos mide 1.90m o mejor dicho, la propiedad *"altura"* del sujeto *"Carlos"* es 1.90m. Esta medida, pensada cómo una instancia de una medición, es un dato.

Desde el punto de vista informático, *"el dato es una representación simbólica (numérica, alfabética, etc.), atributo o característica de una entidad. El dato no tiene valor semántico (sentido) en sí mismo, pero convenientemente tratado (procesado) se puede utilizar en la realización de cálculos o toma de decisiones"* (Wikipedia).

Desde el punto de vista del programador, *"un dato es la expresión general que describe las características de las entidades sobre las cuales opera un algoritmo"*.

Ahora, pensándolo desde un punto de vista de procesamiento, este dato, ¿nos dice algo acerca de si Carlos es más alto que nosotros? ¿O nos dice algo sobre sus hábitos de consumo? Hay que tener en cuenta, que un dato por sí sólo es irrelevante (no tiene semántica). Quizás convendría decir, que a partir del procesamiento de datos, podemos obtener información.

La información, según wikipedia, *representa un conjunto de datos relacionados que constituyen una estructura de mayor complejidad (por ejemplo, un capítulo de un libro de ciencias).* Este conjunto de datos que ha sido procesado tiene significado y relevancia (semántica), también se puede situar la información en un contexto, lo cuál vuelve a la información más interesante. Y quizás la podemos ir corrigiendo, lo cuál la hace más valiosa. También es posible categorizar la información, lo cuál la hace mucho más interpretativa.

Volvamos a la situación de alguna empresa que hace décadas invirtió en tecnología y logró que muchos de sus archivos (de papeles) desaparezcan, y estén de alguna u otra manera, digitalizados. Más allá del ahorro en costos de almacenamiento, y en cuánto al riesgo, ¿qué valor real agregado obtuvo la empresa, si pasó de tener un archivo ordenado por orden alfabético de los clientes (por poner un ejemplo), a tener un almacenamiento digital ad hoc que implementa la misma funcionalidad de orden alfabético únicamente? Seguramente el acceso al dato por el nombre del cliente será más rápido, pero parece ser que esta empresa deja de ganar muchos de los beneficios que le brindaría convertir (procesar) sus datos en información y explotarla en consecuencia. Es entonces cuándo la disciplina de la gestión de datos se convirtió en un desafío y se encararon investigaciones que hoy nos permiten manejar conceptos relativos a datos e información que hace décadas eran impensables.

También tengamos en cuenta que dicha empresa seguramente ha necesitado disponer de los datos en forma rápida; mucha gente consultando la misma información al mismo tiempo (concurrencia), ha necesitado también clasificar dicha información, ya sea por necesidades funcionales o por necesidades de seguridad dado que no sería óptimo que todo el mundo consulte la misma información; ha necesitado disponer de un mecanismo para que dos personas no modifiquen la misma información al mismo tiempo, o más bien para que uno de los dos sea consiente de la modificación del otro; ha necesitado asegurar la disponibilidad de la información a todo momento; disponer de respaldos ante accidentes, mal funcionamientos o desastres; ingresar los datos una sola vez (redundancia mínima), mantener el conjunto de datos consistente e íntegro constantemente, ... Como vemos, estas fueron las causas que condujeron al desarrollos de sistemas administradores de bases de datos que contaran con estas características.

Podemos ensayar una definición de base de datos como *"un conjunto de datos relacionados entre sí de alguna manera"*. Hay algo que no cierra en esta definición y es lo que vimos anteriormente, implícitamente sabemos que algo caótico no es una base de datos, por lo que podríamos agregarle a nuestra definición que una base de datos es un conjunto de datos que son *"lógicamente coherentes"*. Hay cosas que faltan evidentemente en nuestra definición y creo tiene que ver con algo que también mencionamos, cuándo hablábamos de la base de datos de clientes de tal empresa unos párrafos arriba, podríamos estar infiriendo que una base de datos se construye para un fin específico, no existe la *"base de datos de la vida misma"*... Veamos la definición de Wikipedia:

> *"Una base de datos es un conjunto de datos que pertenecen al mismo contexto almacenados sistemáticamente para su posterior uso. En este sentido, una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados para su consulta. En la actualidad, y debido al desarrollo tecnológico de campos como la informática y la electrónica, la mayoría de las bases de datos tienen formato electrónico, que ofrece un amplio rango de soluciones al problema de almacenar datos."*

Ahora bien, existen sistemas que gestionan las bases de datos y nos permiten un acceso mejor a los datos, que son los Sistemas Gestores de Bases de Datos, o **DBMS** (Database Management Systems) que no son más que el software que le permite a los usuarios de las bases de datos, crear, administrar, consultar y destruir los datos. En el contenido de este texto, podemos llamara indistintamente DBMS, base de datos o motor de base de datos.

Los **DBMS** más difundidos implementan el modelo relacional para bases de datos. Que una base de datos sea relacional, implica varias cosas: que respeta un modelo matemático (pero simple) bien definido; y que ha sido sometida a un nivel de normalización (tercera forma normal por ejemplo).

Ya mencionamos que una base de datos es una colección de datos, y hemos dicho que no están dispuestos en forma caótica, ya que de esta manera no tiene objeto el gestionar una base de datos. Por tanto podemos deducir, que un DBMS gestiona tanto datos sobre una problemática en particular (un **dominio**, una base de datos de clientes), como datos de la manera en que se estructuran los datos de dicho **dominio**. Por ejemplo, sabemos que vamos a almacenar en nuestra base de datos las fechas en que los clientes compraron por primera vez, pero también almacenaremos que existe un campo llamado "fecha de alta" en determinada tabla que tiene tal dominio y tales restricciones. Este conjunto de datos acerca de la estructuración de los datos es conocido cómo *"Metadata"*. Entonces, podremos hablar de *"data"* y de *"metadata"* de una base de datos.

Características de un DBMS

De lo descrito anteriormente, podemos deducir la existencia de un conjunto de características inherentes a un DBMS, entre las cuáles se pueden citar las siguientes:

- *Abstracción de la persistencia de los datos*: Los DBMS ahorran a los usuarios detalles acerca del almacenamiento físico de los datos. Da lo mismo si una base de datos utiliza uno o cientos de archivos, o qué tipo de archivos utiliza. Este hecho se hace transparente al usuario. De esa manera se definen varios niveles de abstracción.

- *Independencia de los datos respecto de las aplicaciones:* La independencia de los datos consiste en la capacidad de modificar el esquema (físico o lógico) de una base de datos sin tener que realizar cambios en las aplicaciones que se sirven de la misma.

- *Redundancia mínima:* Un buen diseño de una base de datos logrará evitar la aparición de información repetida o redundante. Al principio, lo ideal es lograr una redundancia nula; no obstante, en algunos casos la complejidad de los cálculos hace necesaria la aparición de redundancias (denormalizaciones).

- *Consistencia:* En aquellos casos en los que no se ha logrado esta redundancia nula, será necesario verificar que aquella información que aparece repetida se actualice de forma coherente, es decir, que todos los datos repetidos se actualicen de forma simultánea.

- *Seguridad:* La información almacenada en una base de datos puede llegar a tener un gran valor para los dueños de los datos. Los DBMS deben garantizar que esta información se encuentra asegurada frente a:

    - Usuarios malintencionados, que intenten leer información privilegiada
    - Frente a atacantes que deseen manipular o destruir la información
    - O simplemente ante las torpezas de algún usuario autorizado pero despistado

  Normalmente, los DBMS disponen de un complejo sistema de permisos y grupos de usuarios, que permiten otorgar diversas categorías de acceso a la información.

- *Integridad:* Se trata de adoptar las medidas necesarias para garantizar la validez de los datos almacenados. Es decir, se trata de proteger los datos ante fallos de hardware, datos introducidos por usuarios descuidados, o cualquier otra circunstancia capaz de corromper la información almacenada.

- *Respaldo y recuperación:* Los DBMS deben proporcionar una forma eficiente de realizar copias de seguridad de la información almacenada en ellos, y de restaurar a partir de estas copias los datos que se hayan podido perder.

- *Control de la concurrencia:* En la mayoría de entornos (excepto quizás el doméstico), es habitual que sean muchas las personas que acceden a una base de datos, bien para recuperar

información, bien para almacenarla. Y es también frecuente que dichos accesos se realicen de forma simultánea. Así pues, un DBMS debe controlar este acceso concurrente a la información, que podría derivar en inconsistencias.

- *Tiempo de respuesta razonable:* Lógicamente, es deseable minimizar el tiempo que el DBMS tarda en traer la información solicitada y en almacenar los cambios realizados.

En conclusión, podemos decir que la utilización de in DBMS tiene que bajar necesariamente los tiempos de desarrollo de los aplicativos, dado que proveerá soporte de almacenamiento persistente a nuestros objetos y estructuras de datos. Así también en aplicativos complejos, nos provee representación de dichos vínculos complejos. Dado que los accesos a los DBMS hoy en día se hayan normalizados (o estandarizados), podemos deducir que los desarrolladores que han trabajado con un DBMS relacional, pueden tranquilamente trabajar con otro sistema relacional, eso es una ventaja para las compañías, ya que es cada vez más sencillo encontrar desarrolladores que tengan experiencia con DBMS. También al proveer de independencia con la aplicación en desarrollo podemos inferir que nos brinda un alto nivel de flexibilidad. Algo que no es menor para los sistemas que tienen cierta necesidad de operar *on-line* es que se dispone de la información actualizada al momento sin necesidad de procesos intermedios. Por el otro lado, antes de utilizar un DBMS conviene evaluar ciertas observaciones, cómo tener en cuenta que el costo de la actualización (tanto del software cómo del hardware) puede ser elevado para determinado tipo de organizaciones. El costo de un administrador de bases de datos dedicado puede ser importante. También hay que tener en cuenta la experiencia del equipo de desarrollo (o de quienes desarrollen los aplicativos), ya que un mal diseño (tanto referente a datos, cómo a la seguridad) puede traer consecuencias catastróficas.

Breve perspectiva histórica de los modelos de bases de datos

Desde hace unas décadas, el modelo relacional se ha instalado cómo un estándar de facto de la industria para la mayoría de las problemáticas de almacenamiento de datos a resolver. Pero hubo modelos que lo precedieron en la "lucha" por resolver la problemática de la gestión de datos eficiente (que parece ser una carrera sin fin). Estas soluciones previas han evolucionado hacia el modelo relacional, siendo sin dudas causa de su desarrollo.

Imaginemos que sólo tenemos un sistema operativo (un Unix por decir algo), y se nos pide implementar una base de datos. Sin dudas, si necesitamos una solución primitiva y de alguna forma eficiente, en lo primero que muchos pensaremos será en el **File System** de Unix[1] (o sistema de archivos). Cualquiera que conozca mínimamente el file system de Unix podrá corroborar que es una forma eficiente de gestionar el almacenamiento de archivos, si pensamos los archivos cómo información, podemos inferir que es una especie de base de datos. Nos da control de concurrencia, nos provee herramientas (el sistema operativo) para hacer back ups, nos garantiza la persistencia, y nos da algunas otras características básicas de las bases de datos. De todos modos, las libertades que nos da el file system tienen una implicancia negativa en el diseño de base de datos que podemos hacer, ya que por lo general no se aplicarán técnicas de modelado: podemos utilizar archivos planos, de longitud fija, separados por coma (CSV), archivos binarios, etc.... Tampoco nos abstrae de la

[1] http://es.wikipedia.org/wiki/Sistema_de_archivos

implementación física de la persistencia y de las búsquedas, todas las rutinas de búsqueda tendrán que ser programadas.

Este enfoque, fue la primera solución a la problemática de la gestión de datos y fue la solución más difundida hasta la década de 1970. El modelo cronológicamente siguiente es el *modelo jerárquico*, que consiste en la implementación de la gestión de datos mediante una estructura similar a la de un árbol, dónde una tabla "hijo" tiene un sólo "padre". Obviamente la existencia de una tabla hijo no tiene sentido sin la existencia de un padre. Cómo ventaja con respecto a la utilización de sistemas de archivos podemos decir que hay cierto proceso de modelado (para representar las jerarquías, o relaciones *one to many)*.

El siguiente modelo en nuestro relevamiento tecnológico es el modelo de red, que no es más que una mejora del modelo jerárquico que incluye la posibilidad de que una tabla "hijo" tenga muchos padres, convirtiendo la jerarquía en una red. La mejora con respecto al modelo anterior consiste en la posibilidad de implementar relaciones *many to many*.

El siguiente modelo es el relacional, que estudiaremos en detalle en los siguientes capítulos. En 1970 Codd [2] sentó las bases de este modelo en los laboratorios de IBM. Las ventajas que tiene con respecto a los modelos anteriores son:

- No interesa la implementación del almacenamiento físico, el modelo no dice nada sobre cómo debe estar implementado el mismo.
- No hay que navegar una jerarquía para llegar al dato deseado.
- Hay una técnica de modelado clara basada en una definición matemática.
- Provee un marco operacional definido.

Continuando con la cronología, cabe destacar las bases de datos de objetos u orientadas a objetos que basan su implementación en el paradigma de objetos (O.O.). Las mismas implementan un concepto multidimensional que permite alcanzar un objeto muy eficientemente. Por otro lado, si queremos obtener una lista de objetos más grande, la performance se degrada. Es por eso que se ha utilizado en implementaciones específicas y hoy en día hay cierto auge de las implementaciones para dispositivos móviles (PDAs, celulares, etc.). Fue concebido para proveer de soporte de persistencia al modelo o paradigma de orientación a objetos. Más adelante veremos en detalle el funcionamiento de estas bases de datos y también veremos cómo se han concebido mapeadores objeto - relacionales para zanjar la diferencia conceptual entre los modelos de orientación a objetos (que han tenido mucho éxito dentro de las plataformas de desarrollo de software) y el modelo relacional (el más exitoso en el mundo de los DBMS).

Tipos de Bases de Datos

Evidentemente, no todas las bases de datos son concebidas para el mismo fin. De hecho, no es lo mismo pensar una base de datos con fines estadísticos (por ejemplo para predecir el

[2] http://es.wikipedia.org/wiki/Edgar_Frank_Codd

comportamiento climático, o para hacer una simulación) que una base de datos para un negocio tipo retail, cómo puede ser un hipermercado. En una base de datos confines estadísticos, quizás tengamos muy pocos usuarios que hagan consultas extremadamente complejas. En la base de datos "retail" nos encontraremos con seguridad con transacciones sencillas, pero tremendamente numerosas y concurrentes (muchos usuarios haciendo transacciones sencillas todo el tiempo).

El análisis clásico de los tipos de base de datos nos dice que existen tres variantes (aunque con seguridad se puedan generar más). Los mismos son:

- *Bases de Datos Transaccionales:*

  Es una base de datos que va a sufrir cambios relativamente pequeños, pero con un flujo que puede muy grande en términos de concurrencia. El ejemplo típico de este tipo de bases de datos son los sistemas OLTP (On-Line Transaction Processing), cómo por ejemplo una base de datos de un hipermercado, o mismo la base de datos de un sitio de internet de ventas cómo Amazon, que debe estar dando servicio las 24 horas del día los 365 días del año (7x24).

- *Bases de Datos para soporte de decisiones:*

  Son bases de datos que orientan decisiones gerenciales o ejecutivas. Para eso generalmente cuentan con años de información de origen transaccional. Pueden ser utilizadas para hacer un procesamiento analítico on-line (*OLAP*[3]). Muchas veces el modelo utilizado para generarlas puede ser muy similar al transaccional (aunque generalmente menos normalizado). Un ejemplo típico de este tipo de bases de datos, son los *Datawarehouse*[4], *Datamarts* (pequeños subconjuntos de un *Datawarehouse*) o bases de datos de *Reporting* (es cómo un *Datawarehouse*, pero sin la información histórica).

- *Modelos híbridos:*

  Son bases de datos que sirven a la vez cómo procesamiento online de transacciones, cómo reporting y cómo datawarehouse. La justificación para este tipo de bases de datos generalmente tiene que ver con el costo de implementación de todas las bases de datos necesarias.

Las Bases de Datos en función de sus Usuarios

Una vez presentada una tipificación de bases de datos es conveniente reconocer quiénes son los usuarios de las mismas. En un principio los DBMS eran utilizados directamente por muchos de

[3] http://es.wikipedia.org/wiki/OLAP

[4] http://es.wikipedia.org/wiki/Almac%C3%A9n_de_datos

sus usuarios (por eso la creación de lenguajes estándar cómo SQL). Evidentemente esto tiene ciertas desventajas, entre las cuáles:

- El usuario final generalmente no está preparado para escribir consultas SQL.
- El usuario final que conozca SQL puede quizás saturar el sistema con una consulta no del todo bien escrita (por ejemplo en vez de ejecutar una junta o join, puede incurrir en el error de ejecutar un producto cartesiano)
- El usuario final puede perjudicar la performance de un sistema complejo ejecutando consultas históricas que manejen grandes volúmenes de datos sin control del administrador de la base de datos.
- Varios usuarios quizás lleguen a las mismas consultas mediante diversos caminos (no reutilización del software).
- El administrador de la base de datos termina perdiendo el control de las bases que administra.

Es por esto que la tendencia ha sido acceder a los datos o bien vía aplicaciones especialmente concebidas, o mediante otro tipo de aplicaciones cómo pueden ser las de Business Intelligence o de explotación de un datawarehouse.

Usuarios vs Actores

Un usuario (de cualquier producto) se lo puede definir cómo la persona que utiliza o trabaja con el mismo. En consecuencia, un usuario de una base de datos es alguien que utiliza o trabaja con la base de datos.

Para trabajar con una base de datos (por ejemplo para ejecutar una consulta), un usuario cualquier ingresa sus credenciales a la interfaz de usuario de la base de datos y luego ejecuta su consulta. Por tanto la implementación de un "usuario" en una base de datos no es más que la implementación de un esquema de seguridad por el cuál en la base de datos se hallan almacenados los usuarios habilitados a acceder a la misma y sus privilegios. Los mismos se hallan, mediante algún mecanismo, encriptados u ocultos a la vista de *"extraños"*.

Ya establecido lo que es un usuario de base de datos, veamos el siguiente ejemplo: Existe una aplicación de *"facturación"* digamos, del cuál el cajero de un supermercado es un usuario (usuario de la aplicación), cuándo la aplicación accede a la base de datos ¿qué sucede habitualmente? ¿Se le pide su usuario y contraseña de base de datos? ¿Hay un usuario para la aplicación y otro para la base de datos? ¿Es en realidad la aplicación en sí un usuario de la base y el cajero un usuario de la aplicación?

Esto que a priori parece ser confuso se debe al crecimiento de la disciplina del desarrollo de aplicaciones, ya que en un principio, la seguridad de autenticación y autorización podía recaer en la seguridad del DBMS. Hoy en día es una práctica que se abandonó y se relegó a servicios especializados de seguridad (servicios de autenticación y autorización cómo pueden ser los accedidos vía el protocolo LDAP). Entonces cuándo una aplicación accede a una base de datos, se le asigna

un usuario de base de datos específicos para que de esa manera se pueda configurar un pool de conexiones y otros servicios independientemente del usuario que accede a la base de datos. Cabe destacar que de esta manera los usuarios de la aplicación *"facturación"* son usuarios indirectos de la base de datos y no podrían ingresar por fuera del marco de la aplicación en cuestión.

Es aquí entonces en dónde hacemos la distinción entre usuarios y actores. Por ejemplo, el auditor es un actor (auditor, controlador). Y se le asigna un usuario en una base de datos en particular, con sus privilegios y restricciones. El usuario de la aplicación *"facturación"* es un actor (usuario final de la aplicación *"facturación"*), y no tiene creado ningún usuario en ninguna base de datos (en tanto usuario de *"facturación"*). Podríamos establecer una relación que parte del conjunto de los actores y llega al conjunto de los usuarios de bases de datos, con una cardinalidad de uno a muchos y obligatoriedad nula.

Los actores que pueden encontrarse fácilmente con relación con las bases de datos pueden ser:

- *Usuarios finales de aplicaciones*

- *Administradores de Bases de Datos:* Responsable de la continuidad operativa de la base de datos. Es por eso que es consciente de todas las operaciones críticas sobre la base de datos que administra. Tiene inferencia en el diseño, y es el técnico especializado en el DBMS que administra, por eso es consultado a diario por los actores de seguridad y desarrollo. Es también el responsable por la disponibilidad de los datos y la recuperación en caso de fallo.

- *Administradores de equipos:* Son los administradores del hardware dónde corren los DBMS y de los sistemas operativos correspondientes. Tienen que tener una interrelación estrecha con el dba, ya que el mismo para poder garantizar la disponibilidad necesaria, tendrá que disponer de ciertos mecanismos que manejan los administradores del sistema operativo.

- *Auditores:* Generalmente no existen cómo usuario de bases de datos, ya que suelen (auditores internos) pedirle la información que necesitan al dba. Sus requerimientos se basan en consultas a los tracks de auditoría (tablas o campos especialmente diseñados a tal fin). Son también los encargados de verificar con el analista de seguridad informática y con el dba de que los controles internos estén implementados.

- *Analistas de Seguridad Informática:* Responsables de definir (en cada aplicativo y en forma genérica) en conjunto con el auditor interno cuáles serán los mecanismos de control (por ejemplo para evitar SQL Injection) y los tracks de auditoría necesarios para garantizar la existencia de información auditable. También suelen definir qué usuarios acceden a qué información. Generalmente no acceden directamente a las bases de datos, sino que piden la creación de usuarios e informes al administrador de la base de datos.

- *Desarrolladores*

    - *Arquitectos:* Son los responsables de definir cómo se encararán técnicamente los desarrollos. En particular en una aplicación pueden definir estilos de diseño y quizás un modelo de poca granularidad. Estará en contacto con el dba por temas de diseño de base de datos, de optimización, de sizing y de carga transaccional.

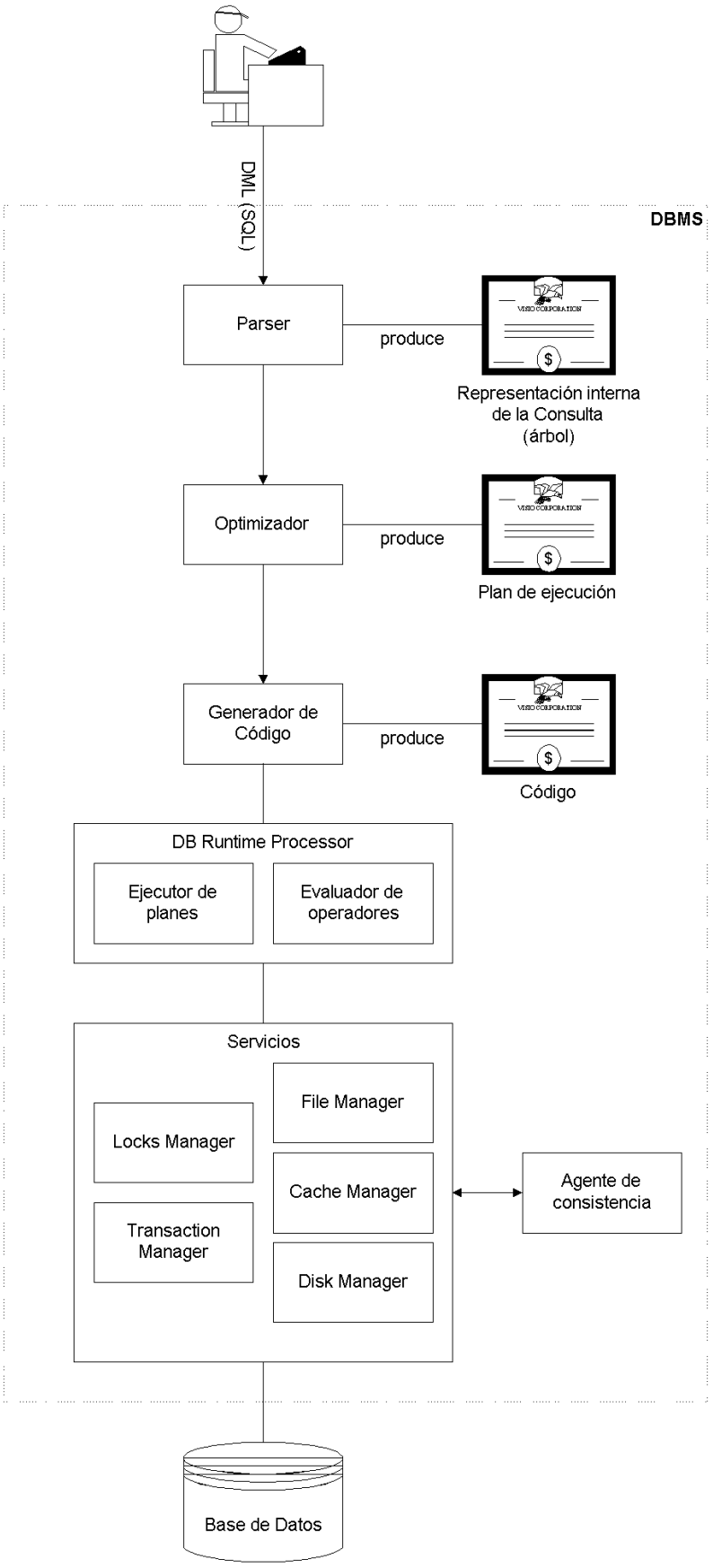
    - *Programadores:* Son los desarrolladores que programan las aplicaciones que acceden a las bases de datos. Muchas veces terminan de diseñar una base de datos. Deben tener en cuenta un contacto fluido con el dba en la etapa de diseño de la base de datos.

    - *Testers:* Son desarrolladores que programan y ejecutan pruebas unitarias e integrales y que acceden a las bases de datos de test cómo usuarios finales y cómo desarrolladores a modo de corroborar la correctitud de una prueba mediante el análisis del estado de la base de datos.

- *Usuarios autónomos:* Son usuarios finales que tienen acceso real a la base de datos, ya sea directamente o por medio de algún aplicativo de business intelligence.

Arquitectura básica de un DBMS

Para cumplir con las características deseables que hemos enumerado, los DBMS implementan (en general) cierto diseño arquitectónico. En el siguiente diagrama se puede visualizar de forma simplificada los elementos de un diseño de un DBMS:


DML (SQL)
DBMS
Parser
produce
Representación interna de la Consulta (árbol)
Optimizador
produce
Plan de ejecución
Generador de Código
produce
Código
DB Runtime Processor
Ejecutor de planes
Evaluador de operadores
Servicios
File Manager
Locks Manager
Cache Manager
Agente de consistencia
Transaction Manager
Disk Manager
Base de Datos

Al DBMS entran consultas SQL (nos enfocamos en las DML o consultas de manipulación de datos), es cuándo son analizadas por un parser que determina la coherencia gramática del comando ingresado, analizando también la existencia de todos los artefactos citados y generando, finalmente, una representación interna de la consulta (por tanto este proceso de "*parsing*" es mucho más que un "*parsing*" formal, pero se permite el abuso de notación).

La representación interna de la consulta es entregada al optimizador, que obteniendo información estadística de la base de datos evaluará cuál es la mejor estrategia para resolver la consulta, generando un plan de ejecución, que consiste en un plan detallado para la resolución de consultas. Generalmente se representa cómo un árbol de operadores relacionales.

A partir del plan de ejecución, el generador de código se servirá de la información provista por el runtime processor para generar código que puede almacenarse para luego ejecutarse (en el caso de consultas pre compiladas), o para ser interpretado inmediatamente.

Durante la ejecución de la consulta, el ejecutor de planes pedirá gradualmente la resolución de operadores relacionales al evaluador de operadores, a su vez se servirá de los servicios internos que provee el DBMS:

- *File Manager:* Se centra en el concepto de archivo (cómo conjunto de páginas de registros) supervisando la información contenida en cada página.

- *Cache Manager:* Implementa las políticas necesarias para mantener una memoria cache lo más eficiente posible.

- *Disk Manager:* Ejecuta las operaciones de lectura y escritura encargándose del almacenamiento físico.

- *Lock Manager:* Para la gestión de la concurrencia, es el servicio que provee los locks (candados) necesarios para garantizar la no aparición de anomalías debido a la ejecución paralela de planes o schedules.

- *Transaction Manager:* Gestor de transacciones que garantiza que las transacciones pidan y liberen locks al *Lock Manager*.

El agente de consistencia consiste en un módulo que continuamente estará "pendiente" por así decirlo de dejar la base de datos en forma consistente.

Por último encontramos la base de datos en sí, representada cómo un cúmulo de datos administrados por el DBMS.

Bibliografía y referencias

- Wikipedia (http://wikipedia.org)
- "Beginning Database design" - Gavin Powell - Ed. Wrox
- "Sistemas de Bases de Datos - Conceptos fundamentales" - Elmasti/Navathe - Segunda Edición Adison Wesley Iberoamericana.
- "Sistemas de Gestión de Bases de Datos" - Ramaklrishnan/Gerke - Tercera edición Ed. McGraw Hill