EL POSTGRES: UN MANEJADOR DE DOMINIO PÚBLICO DE BASES DE DATOS RELACIONALES

Juan Fernando Vélez M.*
Marta Lucía Aristizábal**
Robinson Coronado**
Wilmer Quiceno**
William Alexis Ortiz**
Mauricio Zapata Vanegas**

RESUMEN

En el presente artículo se realiza una introducción al Sistema de Gestión de Bases de Datos (SGBD) POSTGRES. Se presenta la evolución del sistema desde su comienzo como un proyecto de investigación de la Universidad de Berkeley hasta lo que es hoy en día, un proyecto de investigación a nivel mundial debido a las ventajas del Internet. Se describe en forma simple su arquitectura y su funcionamiento en lo relacionado a la gestión de páginas y al procesamiento de las consultas.

ABSTRACT

In the present article is maded a introduction to database management system to POSTGRES. The article shows the system's evolution form beginning as a investigation project of University of Berkeley until its present, a world wide investigation project due to the internet advantages. The system architecture is described in a simple way and its working in the facts of page management and query processing.

* - Ingeniero de Sistemas, Universidad de Antioquia
- Maestría en Informática, Universidad de Pierre et Marie Curie – París VI
- DEA en Bases de Datos, Universidad París 1, Pantheon – Sorbonne
- En la actualidad, estudios de Tesis Doctoral en el área de las Bases de Datos Temporales, Universidad de París 1, Pantheon – Sorbonne.

** Estudiantes Ingeniería de Sistemas.
Grupo de Investigación en Bases de Datos, Facultad de Ingeniería, Universidad de Antioquia.

1. HISTORIA

1.1 EL PROYECTO POSTGRES DE BERKELEY

El Ingres, sistema manejador de bases de datos relacionales (SGBD), fue implementado durante 1975-1977 en la Universidad de California. A partir de 1978 se crearon varias extensiones para este prototipo orientadas a soportar las bases de datos distribuidas [STON83a], relaciones ordenadas [STON83b], tipos de datos abstractos [STON83c], además de la adición de un lenguaje de consulta denominado QUEL [STON83a]. Posteriormente se propuso una nueva interfaz para la aplicación del programa [STON84Aa], pero ésta nunca fue implementada. Esta versión de Ingres a medida que se extendía fue incrementando el grado de dificultad para la inclusión de nuevas funciones y la implantación de nuevas ideas surgidas puesto que éstas no se podían integrar debido al diseño del sistema. A raíz de esto último surgió la necesidad de introducir un nuevo sistema de gestión de bases de datos, denominado POSTGRES, tomando como base del proyecto el SGBD Ingres.

La implementación del SGBD POSTGRES comenzó en 1986. Los conceptos iniciales para el sistema fueron presentados en [STON86] y la definición del modelo de los datos inicial apareció en [ROWE87]. Cabe señalar que estos dos autores, Michel Stonebraker y Lawrence A. Rowe son considerados los padres de este sistema. El diseño de las reglas del sistema fue descrito en [STON87a].

POSTGRES fue sometido a varias mejoras desde entonces, esto se reflejó en las diversas versiones del sistema. El primer "demo" del sistema en operación se realizó en 1987 y después fue presentado en la Conferencia ACM-SIGMOD en 1988. La versión 1, descrita en [STON90a], fue presentada a usuarios externos en junio de 1989. En respuesta a las críticas realizadas sobre el primer sistema de reglas ([STON89]), dicho sistema fue rediseñado ([STON90b]) y entonces la versión 2 fue presentada en junio 1990 con un nuevo sistema de reglas. La versión 3 aparecía en 1991 con la adición de soporte para múltiples administradores de almacenamiento, un ejecutor de consultas mejorado y un reescritor para reescribir el sistema de reglas. Por otra parte, las versiones fueron enfocadas desde entonces hacia la portabilidad y fiabilidad.

POSTGRES ha sido utilizado en la implementación de muchas aplicaciones en los campos de la investigación y de la producción. Éstas incluyen: un sistema de análisis de datos financieros, la ejecución de un paquete de monitoreo para un motor de reacción, una base de datos para el rastreo de asteroides, una base de datos de información médica y varios sistemas de información geográfica. POSTGRES también se ha utilizado como una herramienta educativa en varias universidades. Finalmente, la compañía Illustra Information Technologies, tomó el código y lo comercializó. Posteriormente, en 1992, POSTGRES se convirtió en el principal manejador de datos para el proyecto de computación científica Sequoia 2000. Además, el tamaño de la comunidad de usuarios externos casi se duplicó durante 1993. Se hizo obvio que el mantenimiento del código del prototipo y su soporte estaba aumentando exponencialmente, lo que requería gran cantidad de tiempo y disminuía el que se debía dedicar a la investigación de la base de datos. En un esfuerzo para reducir esta carga de soporte, el proyecto concluyó oficialmente con la versión 4.2.

1.2 POSTGRES95

En 1994, Andrew Yu y Jolly Chen agregaron el interpretador del lenguaje SQL al POSTGRES y el código se puso a disposición en el Web con el objetivo de encontrar un nuevo direccionamiento en el mundo. POSTGRES95 era de dominio público junto con su código fuente, descendiente del código original de Berkeley.

POSTGRES95 es una derivación de la última versión oficial de POSTGRES (versión 4.2). El código es ahora completamente ANSI C y su tamaño ha sido disminuido en un 25%. Existe un conjunto de cambios interiores que mejoran el desempeño y el mantenimiento del código. POSTGRES95 v1.0.x se

ejecuta aproximadamente de un 30% a un 50% más rápido que la v4.2, comparación realizada con la referencia Wisconsin.

1.3 POSTGRESQL

Para 1996, se sacó en claro que el nombre "POSTGRES95" no resistiría la prueba del tiempo. Un nuevo nombre fue escogido, POSTGRESQL, para reflejar la relación entre el original POSTGRES y las más recientes versiones con soporte de SQL. Al mismo tiempo, la numeración de la versión fue reestablecida para empezar en 6.0, volviendo a colocar los números en la secuencia original implantada en el proyecto POSTGRES. El objetivo en el desarrollo para las versiones v1.0.x de POSTGRES95 se centraban en estabilizar el código backend. Con la serie v6.x de POSTGRESQL, el objetivo se centra en identificar y entender los problemas existentes en el backend para aumentar las características y capacidades, aunque el trabajo continúa en todas las áreas.

2 Arquitectura

POSTGRES utiliza un modelo cliente - servidor de "proceso por usuario". Una sesión de POSTGRES está constituida por los siguientes procesos que trabajan en cooperación:

- Un proceso demonio supervisor (*postmaster*)
- La aplicación *frontend* del usuario
- Uno o más servidores de bases de datos *backend* (este es el proceso POSTGRES en sí)

Un único proceso postmaster administra una colección dada de bases de datos en un solo host. Esta colección de bases de batos es denominada una instalación o sitio. Las aplicaciones frontend que desean acceder a una base de datos dada dentro de una instalación realizan una llamada a la librería LIBPQ. (Figura 1).

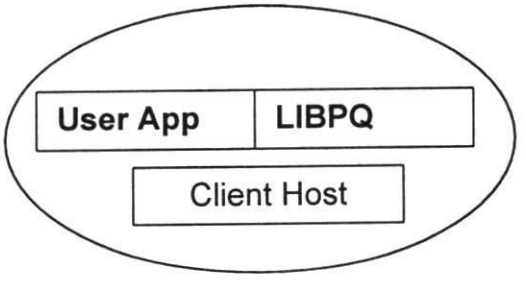

Figura 1.

A continuación la librería envía la petición del usuario, a través de la red, hacia el postmaster. (Figura 2)

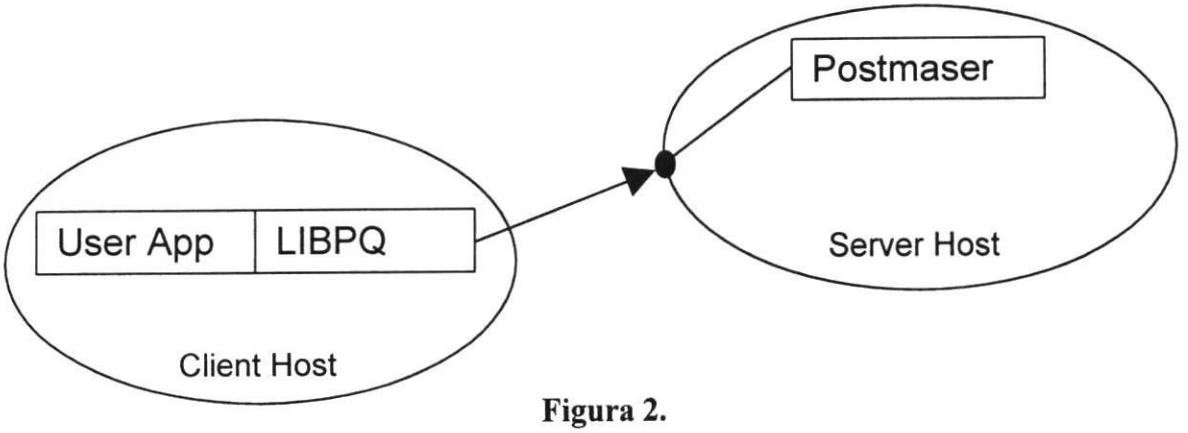

Figura 2.

El postmaster inicia un nuevo proceso servidor backend (Figura 3)

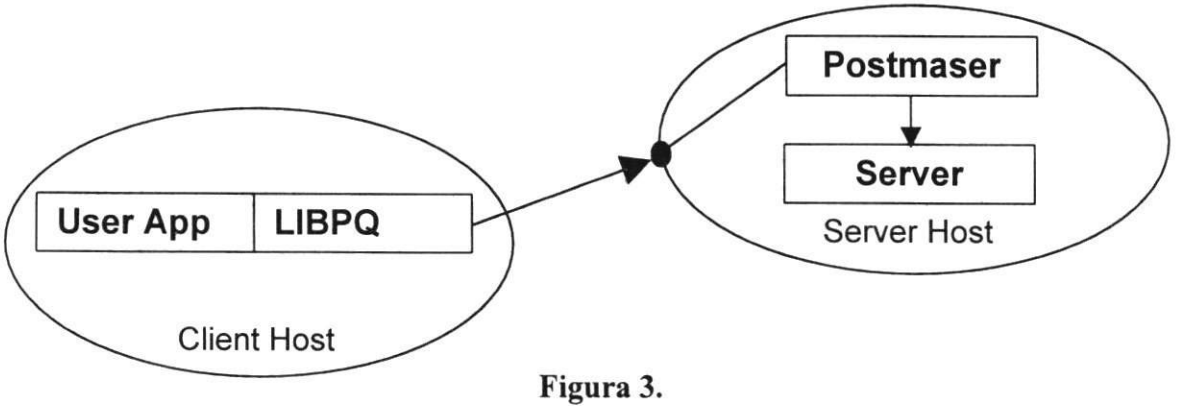

Figura 3.

Seguidamente se realiza la nueva conexión entre el proceso frontend y el nuevo proceso servidor backend. (Figura 4)

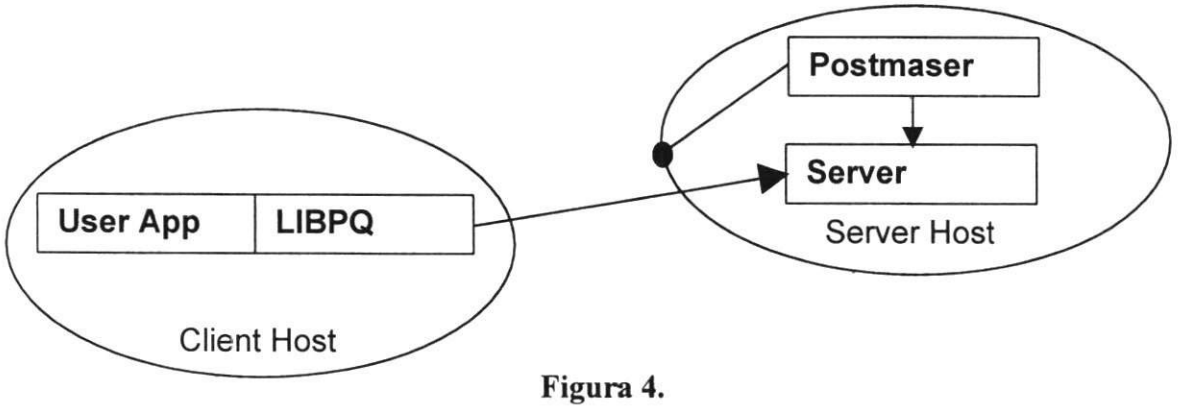

Figura 4.

En este punto, el proceso frontend y el servidor backend se comunican sin la intervención del postmaster. Sin embargo, el proceso postmaster siempre está en ejecución, esperando peticiones, considerando que los procesos frontend y backend vienen y van. La librería libpq permite que un solo proceso frontend pueda realizar múltiples conexiones con procesos del backend. Sin embargo, en la aplicación frontend solo se posee un hilo por proceso (Figura 5). No se soportan conexiones multihilo frontend/backend en la librería libpq. Una implicación de esta arquitectura es que el postmaster y el backend siempre se ejecutan en la misma máquina (el servidor de la base de datos), mientras que la aplicación frontend puede ejecutarse en cualquier parte. Esta consideración se debe tener presente, puesto que los archivos a los que se puede hacer accederse en una máquina del cliente no pueden ser acceso (o sólo se puede hacer usando un nombre de archivo diferente) en servidor que posee la base de datos.

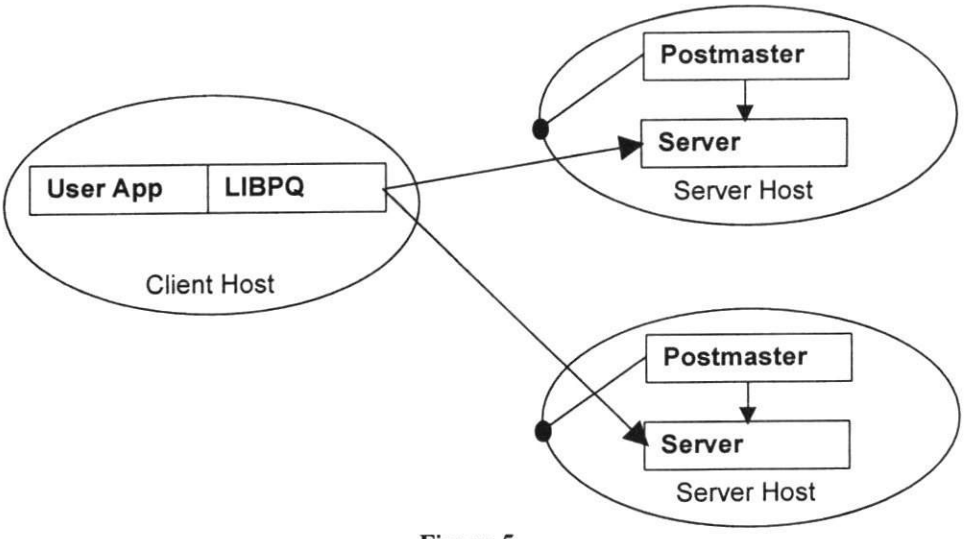

Figura 5.

También se debe tener presente que el postmaster y los servidores Postgres (backend) se ejecutan con el user-id del "superusuario" Postgres. Este superusuario POSTGRES no tiene que ser un usuario especial, como por ejemplo un usuario denominado "Postgres", aunque en muchos sistemas se instala de esta manera. Además todos los archivos que están asociados a una base de datos deben pertenecer al superusuario Postgres. Por último se recomienda que el superusuario Postgres no sea el superusuario del sistema operativo, como lo puede ser el superusuario de UNIX o de LINUX.

3. EL MOTOR DEL POSTGRES

3.1 GESTIÓN DE PÁGINAS

A continuación se realiza una descripción del formato de página predefinido del archivo de la base de datos.

Esta sección proporciona una descripción global del formato de la página usada por las clases de POSTGRES. Los métodos de acceso definidos por el usuario no necesitan usar este formato de página.

En la siguiente explicación, se asume que un byte contiene 8 bits. Además, el término item se refiere a datos que se almacenan en las clases de POSTGRES.

Estructura De La Página

La Figura 6 muestra cómo se estructura la pagina para las clases normales de POSTGRES y para las clases índice de POSTGRES (ej., un índice B-tree).

Esquema de la Página
itemPointerData
filler
ItemData...
Unallocated Space
ItemContinuationData
Special Space
"ItemData 2"
"ItemData 1"
ItemIdData
PageHeaderData

Figura 6.

Los primeros 8 bytes de cada página consisten en la cabecera de la página (PageHeaderData). Dentro de la cabecera, los primeros tres pares de bytes (inferior, superior y especial) representan los bytes de desplazamiento al comienzo del espacio no asignado, al fin del espacio no localizado y al comienzo del espacio especial, respectivamente. El espacio

especial es una región al final de la página que se asigna en el momento de la inicialización de la página y que contiene información específica de un método de acceso. Los últimos 2 bytes de la cabecera de la página, denominados *opaque*, codifican el tamaño de la página e información sobre su fragmentación interior. El tamaño de la página se guarda en cada una de ellas porque los marcos en el buffer pool pueden ser subdivididos en páginas de igual tamaño en un marco por base de marco dentro de una clase. La información de fragmentación interior se usa para ayudar a determinar cuándo debe hacerse la reorganización de la pagina.

A continuación del encabezado de la página están los identificadores de item (ItemIdData). Cada identificador de item nuevo es asignado en los primeros cuatro bytes de espacio sin localizar. Ya que un identificador de item nunca se remueve hasta que se libere, su índice puede usarse para indicar la posición de un item dentro de una página. De hecho, cada puntero a un item (ItemPointer) creado por POSTGRES consiste en un número de marco y de un índice de identificador de item. Un identificador de item contiene un byte de desplazamiento al inicio del item, su longitud en bytes y un conjunto de bits atributo que afectan su interpretación.

Los items se almacenan en espacio asignado a la inversa desde el extremo del espacio no asignado. Normalmente, los items no son interpretados. Sin embargo cuando el item es demasiado largo para ser almacenado en una sola página o cuando la fragmentación del item se desea, él es dividido y cada parte se maneja como uno distinto de la siguiente manera: La primera parte por medio de la próxima, hasta la última parte son colocadas en una estructura de continuación del item (itemContinuationData). Esta estructura contiene el itemPointerData que apunta a la próxima parte y a la propia parte. La última parte del item se maneja normalmente.

3.2 PROCESAMIENTO DE UNA CONSULTA

POSTGRES procesa una consulta de la siguiente forma:

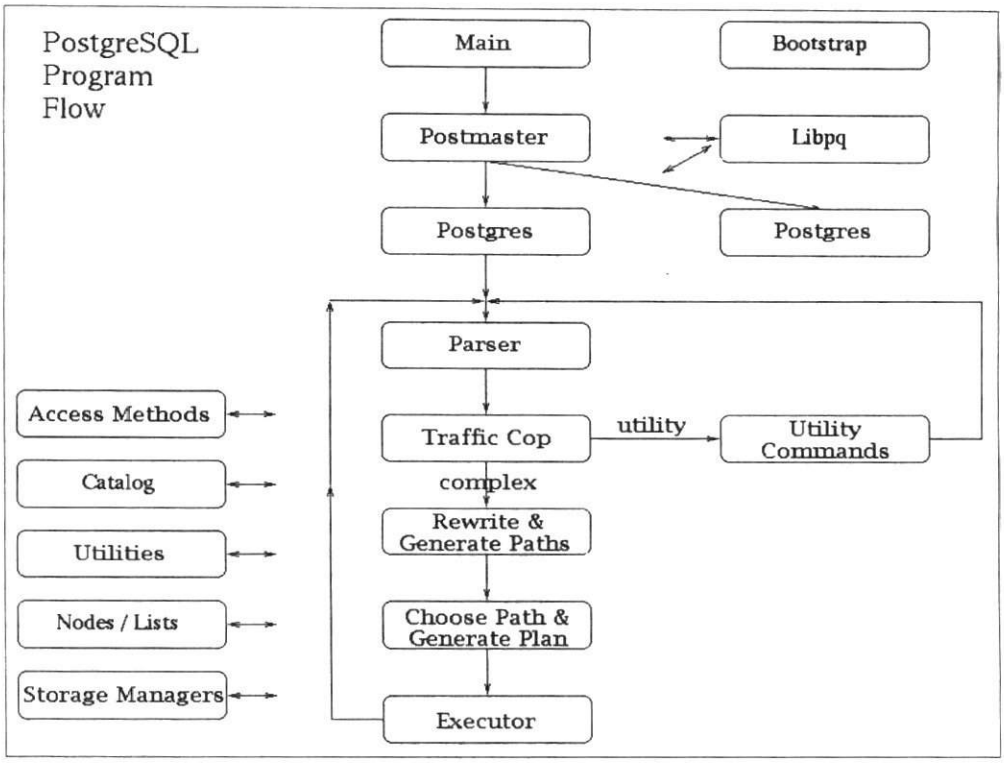

Figura 7. Diagrama de flujo para el procesamiento de una consulta.

La consulta viene al backend por medio de paquetes de datos que llegan a través de TCP/IP o sockets del dominio de Unix. Dicha consulta es cargada dentro de un string, y pasa al parser, donde se examina léxicamente, utilizando la librería scan.l, se fracciona la consulta en tokens (words). El parser utiliza la librería gram.y y los tokens para identificar el tipo de consulta, y carga la estructura para consulta-específica apropiada, como CreateStmt( ) o SelectStmt( ).

La consulta se identifica entonces como una consulta de utilidad o una consulta más compleja. Una consulta de utilidad es procesada por una función de consulta-específica mediante *Command*. Una consulta compleja, como SELECT, UPDATE, y DELETE requiere mas manejo.

El parser toma una consulta compleja, y crea una estructura de la consulta que contiene todos los elementos usados por consultas complejas. Query.qual soporta la cláusula WHERE, que es procesada mediante la función transform-WhereClause(). Cada tabla referenciada en la consulta es representada por un RangeTableEntry y estos son enlazados para formar la tabla de rangos de la consulta, la cual es generada por makeRangeTable(). Query.rtable soporta la tabla de rangos de la consulta.

Ciertas preguntas, como SELECT, retornan columnas de datos. Otras consultas, como INSERT y UPDATE, especifican las columnas modificadas por la consulta. Estas referencias de la columnas son convertidas en entradas de Resdom, las cuales son colocadas en las entradas de target list, y se enlazan para construir la target list de la consulta. La target list se guarda en Query.targetList que es generada por transformTargetList( ).

Otros elementos de la pregunta, como el aggregates(SUM ( )), GROUP BY y ORDER BY también se almacenan en sus propios campos de la consulta.

El próximo paso para la consulta es analizar las posibles modificaciones realizadas por cualquier VISTA o por las REGLAS que pueden aplicar a la consulta. Esto es realizado por el sistema de reescritura.

El optimizador toma la estructura de la consulta y genera un plan óptimo que contiene los procedimientos a realizar para ejecutar la consulta. El módulo path determina la mejor tabla para comenzar a realizar el join, además del tipo de join de cada tabla en el RangeTable, usando la cláusula de Query.qual(WHERE) para considerar el uso óptimo de index.

El plan se pasa entonces al ejecutor para la ejecución, y el resultado es retornado al cliente. El plan es realmente un conjunto de nodos, arreglados en una estructura (árbol de ejecución) con un nodo en el nivel superior y varios subnodos como hijos.