Evaluación del desempeño de consultas en sistemas de bases de datos orientados a filas vs. Orientados por columnas: una evaluación empírica

Trabajo de Grado

Julio César Tenganán Daza

Facultad de Ingeniería
Escuela de Ingeniería de Sistemas y Computación
Programa Académico de Ingeniería de Sistemas
Cali, Junio de 2012

Evaluación del desempeño de consultas en sistemas de bases de datos orientados a filas vs. Orientados por columnas: una evaluación empírica

Trabajo de Grado

Julio César Tenganán Daza
0934723
cesartenganan@gmail.com

Directora
Martha Elena Millán González
martha.millan@correounivalle.edu.co

Facultad de Ingeniería
Escuela de Ingeniería de Sistemas y Computación
Programa Académico de Ingeniería de Sistemas
Cali, Junio de 2012

Dedicado a mis amados padres Beatriz y Julio.

AGRADECIMIENTOS

A mis padres y hermanos por brindarme su apoyo, consejo, protección y cariño incondicional, por darme tantas alegrías y por regalarme un hogar lleno de amor y respeto.
A mi novia y amigos por su agradable compañía y sus enseñanzas.
A mi directora de tesis Martha Millán, por su conocimiento y sugerencias tan importantes para la culminación de este trabajo.

Tabla de Contenido

Índice de Tablas

Índice de Figuras

RESUMEN

La competitividad de las organizaciones es un factor clave para mantener o mejorar su posición en el entorno socioeconómico. Es por esto, que muchas se han interesado en utilizar la información histórica generada del día a día de sus operaciones, para analizarla y obtener conocimientos importantes que ayude en la toma de decisiones. Uno de los mecanismos para obtener información que integra distintas áreas en la organización, es usar bodegas de datos (*Data Warehouse*), las cuales se aprovechan diseñando consultas a la medida, que satisfacen necesidades de los altos ejecutivos en las organizaciones. El problema que se presenta con estas consultas es que son, por lo general, complejas computacionalmente y disponen de datos que pueden no caber en memoria (*RAM*) para procesarlas, lo que genera demora en la entrega de resultados.

La mayoría de los *DWH* se implementan en sistemas de gestión de bases de datos (*DataBase Management System DBMS*) orientados a registros o filas, por su uso generalizado y soporte. Sin embargo, nuevas propuestas orientadas a resolver los problemas de desempeño de consultas en grandes bodegas de datos, se han desarrollado y algunas versiones de sistema se han liberado. Una de éstas es la de *DBMS* orientados a columnas, que por sus características funcionales parece que mejoran el desempeño de las consultas en los *DWH* .

El propósito de este trabajo, es comparar el desempeño de las consultas sobre un sistema de bases de datos orientadas a filas, con respecto a un sistema orientado a columnas. Las implementaciones, para el mismo dominio de aplicación, se hace respectivamente, en *PostgreSQL* y *MonetDB*.

Palabras Clave: Evaluación, desempeño de consultas, *DBMS*, orientada a columnas, bodegas de datos, *PostgreSQL*, *MonetDB*.

ABSTRACT

Organization's competitiveness is the key factor to keep or improve its position in the socioeconomic environment. In this way, many are interested in the using of the historical information generated from day to day operations, to analyze and gain important knowledge that helps in decision-making.

One of the mechanisms for to get information that integrates different areas in the organization, is to use Data Warehouses, which are exploited designing queries that satisfy the needs of the high executives in the organizations. The problem that occurs with these queries, is that they are computationally complex with data available that may not fit in memory (RAM) to process, creating delay in the delivery of results.

Most DWH are implemented in row-oriented DataBase Management System (DBMS), for its generalized use and support. However, new proposals aimed to solving, have been developed and some versions of the system have been released. One of these is the column-oriented DBMS, which, for its functional characteristics, it seems that improve the performance in the DWH.

The purpose of this study is to compare the performance of queries on a row-oriented database system with respect to column-oriented system. The implementations for the same application domain, are made, respectively, PostgreSQL and MonetDB.

Key Words: *Evaluation, Query Performance, DBMS, Column-oriented, Data Warehouses, PostgreSQL, MonetDB.*

1 INTRODUCCIÓN

La implementación de un *DWH* en una organización, debe soportar análisis complejos de grandes volúmenes de datos con la finalidad de encontrar información estratégica, que apoye la toma de decisiones. La extracción de información hecha por los usuarios, se obtiene mediante análisis de reportes suministrados por el *DWH* , usando consultas de alto nivel. Estos reportes luego de ser interpretados por los usuarios, dan lugar a búsquedas en nuevos niveles de detalle, tanto como les permita su dominio del negocio.

En el diseño del *DWH* , se utiliza el modelo multidimensional. En este modelo, los datos se organizan y se agregan en una estructura llamada cubo de datos, donde cada cubo puede contener otros cubos que ofrecen un nivel de detalle diferente. Cada lado del cubo, representa una dimensión o un punto de vista desde el cual se puede extraer información de los componentes del negocio que se hayan modelado.

La mayoría de los diseños de *DWH* en las organizaciones, se implementan en los mismos *DBMS* operacionales (orientados a filas) con los que cuentan. Sin embargo, la carga en los *DBMS* operacionales, se hace con mucha frecuencia y en pequeños volúmenes, a nivel de registro o transacción. Esta característica, hace que estos sistemas estén optimizados para la escritura orientada a filas, permitiendo el almacenamiento contiguo en disco de los campos de cada registro.

La importancia por la lectura en gran volumen, más que por la escritura de los datos en los *DWH* , ha motivado a fabricantes de *DBMS* a ofrecer alternativas que den mejor rendimiento, dejando un amplio horizonte que tendrá que ser explorado con el fin de comprobar si lo ofrecido realmente mejora el desempeño de los *DWH* .

Los *DBMS* orientados a columnas introducen un cambio en la capa de almacenamiento, separando en archivos independientes cada columna de una tabla. De esta manera se logra almacenar en disco, datos de forma consecutiva, los cuales tienen las mismas características y que posiblemente aparecen con mucha frecuencia. Estas propiedades en el almacenamiento se aprovechan para introducir mejoras en el rendimiento de las consultas, relacionadas con la disminución de accesos a disco y el procesamiento individual de cada columna, al tener que recuperar solo las columnas necesarias y no todas las relaciones involucradas en las consultas, los tiempos de entrega de resultados en grandes volúmenes de datos como en los *DWH* se disminuyen considerablemente.

En este trabajo, se evalúan las consultas bajo los dos enfoques de almacenamiento, para el mismo subconjunto de datos de un *DWH* (*Data Mart*). Se diseñan consultas que acceden, a través múltiples dimensiones y niveles de detalle, a conjuntos de datos que crecen incrementalmente en para tratar de comparar los dos enfoques de almacenamiento.

1.2 Objetivos

1.2.1 Objetivo General

- Comparar el desempeño de las consultas sobre un sistema de bases de datos orientadas a filas, con respecto a un sistema orientado a columnas en un dominio particular y en el entorno de un *DWH* .

1.2.2 Objetivos Específicos

- Identificar un dominio de aplicación, relativamente complejo, para comparar el desempeño de consultas, usando un modelo relacional clásico y un modelo basado en columnas.
- Construir un conjunto de datos sintético para cargar incrementalmente las bases de datos.
- Diseñar consultas para ambos sistemas con diferentes niveles de agregación y dimensionalidad para hallar diferencias en la respuesta de ambos sistemas.
- Calcular, empíricamente, el costo de evaluación, en términos de tiempo, de cada consulta en ambos sistemas incrementando el volumen de datos sobre el que se aplica la consulta.
- Diseñar e implementar un prototipo de herramienta que facilite la comparación de los dos modelos de almacenamiento.
- Comparar y evaluar los resultados obtenidos.

1.2.3 Resultados Esperados

A continuación en la tabla1, se muestran los resultados esperados derivados de los objetivos específicos y su ubicación dentro del documento.

Objetivo Específico	Producto(s) Esperado(s)	Sección
Identificar un dominio de aplicación, relativamente complejo, para comparar el desempeño de consultas, usando un modelo relacional clásico y un modelo basado en columnas.	Diagrama del modelo entidad relación que soporta los datos sintéticos, diagrama del modelo dimensional de datos y código *SQL* de implementación para ambos sistemas.	✓ Sección 3.2 ✓ Anexo 1 ✓ Anexo 2 ✓ Anexo 3

Construir un conjunto de datos sintético para cargar incrementalmente las bases de datos.	Documentación resultante de explorar herramientas de generación de datos sintéticos, archivos con el conjunto de datos generados, código *SQL* para inserción de datos.	✓ Sección 3.3 ✓ Anexo 4 ✓ Anexo 5
Diseñar consultas para ambos sistemas con diferentes niveles de agregación y dimensionalidad, para hallar diferencias en la respuesta de ambos sistemas.	Scripts *SQL* para las consultas, Documentación del conjunto de tipos de consultas en orden de complejidad: acceso a una sola tabla, con *join*s, con operaciones de agregación, con ordenamiento, etc.	✓ Sección 4.1
Calcular empíricamente, el costo de evaluación, en términos de tiempo, de cada consulta en ambos sistemas incrementando el volumen de datos sobre el que se aplica la consulta.	Registro del costo de ejecución en tiempo de las consultas vs. La información registrada de cantidad de filas afectadas, numero de dimensiones y niveles de agregación.	✓ Sección 3.4 ✓ Sección 4.1
Diseñar e implementar un prototipo de herramienta que facilite la comparación de los dos modelos de almacenamiento.	Definición de requerimientos, Diagrama de clases, diagrama de persistencia (Modelo E-R) Documentación. Código fuente de la herramienta.	✓ Capitulo 5 ✓ Anexo 6
Comparar y evaluar los resultados obtenidos.	Tablas, Gráficos y documento de texto con las conclusiones derivadas de los resultados.	✓ Sección 4.1 ✓ Sección 4.2

Tabla 1: Productos esperados por objetivo

1.3 Justificación

1.3.1 Justificación Económica

Identificar un enfoque de almacenamiento de datos que mejore el rendimiento de las consultas en bases de datos, constituye una ventaja competitiva para las empresas que implementan soluciones dedicadas a la inteligencia de negocio, minería de datos y/o *Data Warehousing* porque mitigarían el punto crítico relacionado con los tiempos de respuesta y con la entrega de resultados.

Implementar soluciones con un sistema de bases de datos eficiente en la tarea de extracción de información, permite disminuir la inversión que se hace en infraestructura de hardware

dedicada al suministro de recursos para la difícil tarea de procesamiento en los sistemas se bases de datos orientados a filas.

1.3.2 Justificación Social

Actualmente el debate en cuanto a los beneficios que proponen los sistemas de bases de datos orientados a columnas tiene tanto simpatizantes como detractores. La exploración de los dos enfoques a nivel práctico descartará o ratificará algunas de las afirmaciones hechas por los investigadores y los desarrolladores de sistemas de bases de datos, contribuyendo con el aprovechamiento de las características más ventajosas según sea el caso de aplicación en la práctica.

1.3.3 Justificación Académica

Académicamente, se contribuiría con el aporte documental en el tema de las bases de datos orientados a columnas que hasta la fecha en la EISC no se cuenta con un trabajo que explore estas tecnologías. La exploración y comparación estos dos enfoques en la práctica, supone la obtención de recomendaciones y criterios de selección útiles para el diseño de bases de datos que den soporte al área de descubrimiento del conocimiento en bases de datos, tema principal de una de las asignaturas electivas de la EISC.

La obtención de diferencias relevantes que propicien el uso de los sistemas de bases de datos orientados a columnas, sería un incentivo para que próximos cursos de bases de datos incluyan en sus contenidos la temática como complemento alternativo al amplio tema dedicado a las bases de datos orientadas a filas.

2 MARCO TEORICO

En este capítulo se introducen los principales conceptos relacionados con el uso o aplicación de las bases de datos, haciendo énfasis en las implementaciones que dan soporte a la toma de decisiones (bases de datos analíticas). Se representan características generales los dos *DBMS* utilizados en la comparación de desempeño y se hace un acercamiento a las características de diseño empleadas en la construcción de *DBMS* orientados a columnas, las cuales hacen posible el aprovechamiento de nuevas tendencias en el hardware, para obtener mejor desempeño en el procesamiento de consultas.

2.1 Sistema de Gestión de Bases de Datos (DataBase Management System DBMS)

Un *DBMS* es conjunto de programas especializados en la lectura y escritura de una colección de datos interrelacionados. Esta colección llamada base de datos, facilita acceso y persistencia de información relevante para las organizaciones.

2.2 Bases de Datos Operacionales

Son implementaciones de bases de datos para aplicaciones que soportan las operaciones de una organización. Con la implementación de estas bases de datos, las aplicaciones pueden reunir, almacenan y procesar ágilmente todos los datos necesarios para el éxito de las operaciones diarias. Las aplicaciones que usan estas bases de datos, son importantes para las organizaciones, en ellas procesan los pedidos, mantienen el inventario, llevan la contabilidad, etc. Desde que comenzaron a construir estas aplicaciones en la década de los 60's, las organizaciones se volvieron totalmente dependientes y en la actualidad ninguna empresa moderna puede sobrevivir sin ellas[8].

2.3 Bases de Datos Analíticas

Son implementaciones de bases de datos para recopilar datos históricos, útil para aplicaciones enfocadas en el descubrimiento y la extracción de información estratégica para las organizaciones. Con estas aplicaciones, los responsables de mantener la competitividad de las organizaciones, obtienen información para tomar decisiones apropiadas; formulando estrategias de negocio, estableciendo metas, y fijando objetivos que pueden ser monitoreados con la ayuda de estas aplicaciones[9].

2.5 Data Warehouse

Los *Data Warehouse* o bodegas de datos, son depósitos que almacenan bajo un esquema unificado la información reunida de varias fuentes y provenientes de los distintos procesos en las organizaciones. En el *Data Warehouse*, los datos se almacenan mucho tiempo, lo que permite el acceso a datos históricos. Proporcionando a los usuarios una sola interfaz consolidada con los datos de la organización, lo que facilita la escritura de consultas para el soporte a la toma de decisiones.

2.6 Data Mart

Un *Data Mart* (*DM*) es un subconjunto de datos de un *Data Warehouse*, debido a que este último puede construirse con la unión de todos los *Data Mart* de la organización. La finalidad de implementar un *Data Mart*, es la de ayudar en la toma de decisiones del área específica que haya modelado, bien sea a nivel departamental o por unidad de negocio dentro de una organización.

2.7 DBMS orientados a columnas

Los manejadores de bases de datos orientados a columnas introducen un enfoque diferente en cuanto a la forma de almacenar y procesar los datos de las tablas. A diferencia de la mayoría de los *DBMS* relacionales, en este enfoque las columnas de cada tabla en la base de datos, son almacenada por separado, permitiendo que valores almacenados contiguamente sean muy similares y logrando con ello compresiones bastante densas. Este tipo de descomposición en el almacenamiento, es ventajoso cuando se requiere lectura de datos a gran escala, porque reduce la transferencia de datos desde disco, al obtener solo los atributos deseados.

2.8 MonetDB

MonetDB es un sistema de base de datos de código abierto para aplicaciones de alto rendimiento que se caracteriza principalmente porque usa un modelo de almacenamiento basado en particionamiento vertical (orientado a columnas) [15].Su desarrollo se remonta a principios de los años 90'.

MonetDB surge con el propósito de desarrollar un *kernel* de base de datos más sencillo, que permitiese aumentar la velocidad de procesamiento de consultas. *MonetDB* no propone solo una forma diferente de almacenamiento para facilitar la lectura de atributos específicos, evitando llevar a memoria atributos irrelevantes, sino que también introduce otros enfoques de mejora en aspectos como: uso óptimo de la memoria principal, al considerarse como recuso limitado, optimizador de costos, el costoso procesamiento de *join*, la materialización

de las tablas en memoria y el bloque para las decisiones de optimización enfocada en las capas de estrategia táctica y operacional.

Inicialmente, *MonetDB* fue lanzado en 1993, con un *kernel* basado en tablas de asociación binaria *(Binary Association Tables BATs)* el cual permitía acceso a los datos almacenados mediante un lenguaje de scripts llamado MIL (*MonetDB Interface Language*). *MonetDB*, a mediados de los 90', empieza a dar soporte a proyectos de minería de datos y a bases de datos analíticas, aportando innovadoras técnicas en la mejora del rendimiento. En desarrollos más maduros incorpora el estándar *SQL99* luego *SQL2003*, interfaces para lenguajes como: *PHP, JDBC, Python, Perl, ODBC, RoR.* En 2004 lanza oficialmente la versión *open-source*. En 2011 después de un proceso de crecimiento y limpieza de código sale la versión actual de *MonetDB5*.

2.9 PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia *BSD* y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto más potente del mercado. Elegida en varias oportunidades como la mejor base de batos por: "*Linux Journal Editors*"", Producto del año por: "*Developer.com*" entre otros[17].

PostgreSQL utiliza un modelo cliente/servidor que incluye interfaces nativas para: *ODBC, JDBC, .Net, C, C++, PHP, Perl, TCL, ECPG, Python, and Ruby.*

Su última versión, *PostgreSQL*9, soporta la mayoría de las definiciones del estándar *SQL2008*, y cuenta con un número de usuarios cada vez es más grande. Es raro encontrar a una gran empresa que no esté usando *PostgreSQL* por lo menos en un departamento[18].

2.10 Tendencias en las arquitecturas

Los continuos avances en hardware de los últimos años ponen hoy a nuestra disposición, maquinas con capacidades de computo realmente impresionantes comparadas con las de hace unos 30 años, cuando recién se desarrollaban la mayoría de *DBMS* con que contamos actualmente. Estos avances han permitido que las velocidades alcanzadas por algunos dispositivos se hayan superado hasta un orden de las 10000 veces. Comparando características de los procesadores, podríamos ver que las Millones de Instrucciones por Segundo MIPS alcanzadas en los años 80's, apenas era de 1MIPS[7] versus las 27000 MIPS aprox. que puede alcanzar actualmente un procesador doble núcleo y también la capacidad de su memoria cache 1000 veces mejor, que ha pasado de medirse en KB a MB. También en cuestión de almacenamiento se ha incrementado la capacidad de disco duro y memoria *RAM*. Alcanzando velocidades de transferencia en disco 100 veces mejor y con tiempos

promedio de búsqueda por cabeza 10 veces más rápido. Pero los incrementos de velocidad de procesador y disco duro no han sido equivalentes tenemos un (10000x vs. 100x vs. 10x) que seguramente tengan efectos significativos en el desempeño de los flujos de carga de los *DBMS*[7]. También el tener actualmente procesadores con velocidades de procesamiento que superen en gran medida la velocidad de acceso a memoria *RAM*, debido a tiempos de latencia de memoria, es un cuello de botella que impide alcázar mejor rendimiento de las aplicaciones[5].

Para algunas aplicaciones de bases de datos quizá la lectura de datos de memoria principal no sería un inconveniente, sí continuaran con las cargas de trabajo de bases de datos como las transaccionales *OLTP*. Pero como el uso de las bases de datos se ha ampliado hacia otras funcionalidades más complejas, como lo es: el procesamiento analítico *OLAP* y la búsqueda de tendencias ocultas en los datos. Entonces los *DBMS* tradicionales están soportando cargas de lectura y recuperación de información en volúmenes más grandes, haciendo que el acceso a memoria sea un obstáculo para entregar resultados eficientemente.

Considerando las marcadas diferencias en las tendencias del hardware, es necesario que se evalué si los *DBMS* tradicionales que usamos desde los 80's, están en condiciones de obtener buenos resultados usando con eficiencia los recursos; o si por el contrario, los nuevos enfoques de *DBMS* que se están desarrollando, han tenido en cuenta estas características del actual hardware en su diseño, para obtener mejor rendimiento y dar como se espera, un mejor soporte a las grandes bases de datos como las que generalmente se manejan en los *Data Warehouse*. Los tradicionales *DBMS* se diseñaron cuando la principal barrera era el acceso a disco y aunque lo sigue siendo y en aumento, el acceso a datos en memoria principal también debe ser ahora un factor influyente para lograr optimizaciones que eviten los tiempos de espera para la obtención de datos (latencia de memoria).

2.11 Optimización en los DBMS orientados a columnas

A continuación se introducen algunos conceptos importantes que se han tenido en cuenta en el diseño *DBMS* orientados a columnas, estos conceptos se han ido divulgado con la aparición de motores de bases de datos como *MonetDB*, *C-Store* y múltiples trabajos de investigación publicados en revistas científicas de bases de datos[1, 3, 5, 11].

Se puede argumentar que una de las ventajas más citadas de los *DBMS* orientados a columnas es la compresión. Beneficiada por una capa de almacenamiento diferente, donde cada columna es un conjunto de datos similares almacenados contiguamente en disco. Permitiendo que el desempeño de los algoritmos de compresión funcionen mejor con datos que presentan baja entropía. Contrario a los *DBMS* orientados a filas, donde se almacena en disco una tupla seguida de otra, localizando consecutivamente, bloques o conjuntos de datos con diversos tipos, que podrían disminuir el resultado de la compresión.

La importancia de la compresión, inicialmente es, la reducción del tamaño en disco, de los archivos que conforman cada una de las columnas en las relaciones. Con ello, también reducir los accesos a disco y sus grandes tiempos de espera en la obtención de datos, que darían costosos tiempos de inactividad, más cuando se tienen unidades de procesamiento más rápidas como se mencionaba anteriormente, en las tendencias de las arquitecturas.

La implementación de esquemas de compresión tradicionales en *DBMS* orientados a columnas, ha logrado mejores resultados que los alcanzados en sistemas orientados a filas. Así mismo, la incorporación de esquemas adicionales, que no eran viables en *DBMS* orientados a filas; permiten a los sistemas orientados a columnas, tener una gama de posibilidades para elegir la compresión específica, que dará un mejor desempeño en cada columna. Como lo demuestran en[3].

Tener datos comprimidos en el *DBMS* también puede ser ventajoso, si se emplea la compresión específica de cada columna, para hacer que los operadores relacionales implementados en el sistema, sean capaces de procesar los datos sin descomprimir. Esto es útil cuando se tienen funciones de agregación, que se pueden calcular con tan solo algunas multiplicaciones que facilitan los datos en su estado comprimido y sin tener que recorrer todos los datos de la columna a la que se le está aplicando la agregación. De esta forma se elimina la desventaja de tener que invertir ciclos de *CPU* en la compleja tarea de descompresión a todos los datos[3].

También para un buen uso de la cache, se propone en [5] hacer una cuidadosa revisión del estado de la cache y de los patrones de acceso por parte de los *DBMS*. Para crear algoritmos con una especie de consciencia del entorno, que les permita trabajar sobre diversas arquitecturas de computadoras, usando modelos parametrizables en tiempo de ejecución con técnicas de calibración automática basadas en costos de acceso a memoria.

En cuanto a materialización, proceso mediante el cual se va construyendo el conjunto de atributos de salida de una consulta; se encuentra una nueva forma de obtenerle más adecuada respecto al tipo de almacenamiento orientado a columnas. En los *DBMS* tradicionales se habla del esquema usado como materialización temprana, donde tan pronto se van accediendo columnas, se agregan, pero solo si la columna es necesitada por otro operador o si es requerida en la salida. Y las columnas que no se necesiten más se eliminan en las proyecciones.

El nuevo esquema llamado materialización tardía en[1], no forma tuplas o conjunto de atributos de salida sino, hasta que parte del plan de ejecución de la consulta se haya cumplido. Entonces es posible obtener cada columna y aplicar primero filtros de selección, aun en su estado comprimido con mayor rapidez, para obtener solo las posiciones que cumplen el predicado. Expresando esta salida en forma de rangos, lista o *bit-map* y luego con operadores más eficientes como *AND* por ejemplo, hacer la intersección de los valores que formaran las tuplas de salida. Aunque en algunas ocasiones es muy eficiente, en otras se introduce mayor complejidad a la hora de unir las columnas, porque se realizan muchos

escaneos para ubicar las posiciones de cada atributo de la tupla. Por tal motivo, proponen esquemas híbridos que permitan al ejecutor de consulta elegir en su plan la mejor estrategia de materialización.

Otro factor importante introducido para la optimización se refiere a: conservar en disco otras representaciones de una misma columna llamadas proyecciones[11], que se encuentran ordenadas por ellas mismas y por múltiples atributos diferentes como llaves foráneas. Lo que permite agilidad en el procesamiento de consultas con agregados, al tener datos pre procesados y sin incurrir en excesivo aumento de espacio en disco debido a redundancia de información, ya que se cuenta con algoritmos de compresión específicos por columna que reducen el tamaño de estos archivos.

El hecho de tener todas las columnas separadas conlleva a que se deba almacenar la llave de la tupla junto con el valor de cada atributo, para poder hacer reconstrucción de tupla. Sin embargo, se pueden tener proyecciones en el mismo orden del atributo llave, haciendo innecesario almacenar las llaves, ya que se puede usar el valor de la posición como llave virtual para obtener todos los demás valores de una tupla[11].

Insertar los datos en la misma secuencia como se presentan en la entrada, facilita el proceso de escritura, pero es más costoso para la recuperación de información con cargas de trabajo que quizá requieran que los datos estén en otro orden. Por tal motivo, proponen[11] una combinación de componentes de software, como se muestra en la figura 1. Un componente optimizado para la escritura (*Writeable Store WS*), que reciba fácilmente los datos en la entrada, otro componente optimizado para la lectura *(Read-optimized Store RS*) donde se direccionarían las consultas y un tercer componente encargado hacer la conexión de los dos anteriores (*Tuple Mover*), el cual mueve las tuplas de *WS a RS.*

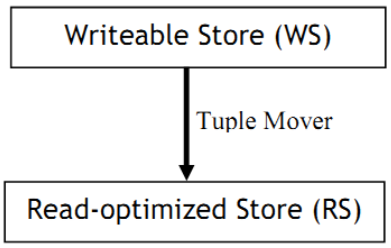


Figura 1: Arquitectura del *DBMS* orientado a columnas *C-Store.*

Se aclara que el anterior esquema funciona bien, en ambientes con consultas que implican enormes cargas de trabajo y donde los inserts se hacen pocas veces. En ambientes con inserts continuos o frecuentes update y delete, se presentarían conflictos de lectura y escritura, provocando disminuciones considerables en el desempeño del sistema.

3. COMPARANDO DESEMPEÑO

El propósito de este trabajo era comparar el desempeño de las consultas a un *DWH* implementado bajo dos enfoques: Uno, en un *DBMS* orientado a filas (*PostgreSQL*) y otro, en un *DBMS* orientado a columnas (*MonetDB*). Con este propósito, se presenta en la sección 3.1, el estado del arte que orienta la realización de las pruebas empíricas, En la sección 3.2 la descripción del dominio de aplicación para la implementación del *DWH*, donde se prueban las consultas. En la sección 3.3, se describe cómo se construye el conjunto de datos sintéticos para cargar incrementalmente las bases de datos. Finalmente en la sección 3.4, se ilustra el proceso usado para determinar empíricamente el costo de evaluación de cada consulta

3.1 Estado del Arte

En esta sección se describen trabajos previos, relacionados con la comparación de desempeño entre *DBMS* orientados a filas vs. *DBMS* orientados a columnas.

En [4] se compara el desempeño entre un *DBMS* orientado a filas y tres posibles aproximaciones de un *DBMS* orientado a columnas. En la experimentación se pretendía que las distintas aproximaciones o enfoques tuvieran implementado entre ellas, pequeñas variaciones para evaluar las ventajas y desventajas de estos cambios y encontrar así, una especificación de diseño para la capa de almacenamiento y el ejecutor de consultas que logre mejorar el desempeño. Posterior a la elección del mejor de los enfoques, se describen las mejoras que deberían incluirse para la completa construcción de un *DBMS* orientado a columnas. Las mejoras que describen el buen desempeño del *DBMS* denominado *C-Store*, comprende la eficiente compresión de los datos almacenados en columnas y la capacidad de procesarlos sin ser descomprimidos, modelos analíticos con heurísticas usadas para planear la construcción de tuplas y la inclusión de nuevas técnicas de *join*.

En [2] se aborda la comparación de un sistema orientado a columnas vs varios sistemas orientados a filas. Parten del supuesto de que en un sistema orientado a filas se puede simular la capa de almacenamiento físico de un sistema orientado a columnas y obtener desempeños similares.
Para lograr la simulación, se implementa un esquema de particionamiento vertical, en cual, las columnas de cada tabla se separan formando tablas separadas, acompañando al atributo de la columna con la llave que identifica la tupla, para poder, después, reconstruir toda la tabla. Esto se propone para poder leer solo la información de las columnas necesarias en las consultas. Se implementan planes de acceso solo a los índices, que se crean por columna, para mejorar el desempeño de las búsquedas y para evitar la lectura de todas las columnas

de las tablas, se implementan vistas materializadas con las columnas necesarias para responder a las consultas.
Se concluye después de la experimentación con múltiples consultas, que no es imposible simular el trabajo de un sistema orientado a columnas en un sistema orientado a filas, sin embargo, la simulación tiene desempeños pobres en los actuales sistemas orientados a filas. Si los sistemas de bases de datos orientados a filas mejoraran algunos aspectos de procesamiento al interior de su propio sistema sería posible obtener mejore resultados.

Por otra parte en [6] se afirma que muchos de los beneficios de los *DBMS* orientados a columnas se pueden simular con los sistemas tradicionales usando vistas materializadas y con datos donde no se favorezca la compresión, que es lo que mejor hace el sistema orientado a columnas. Además de que se puede hacer sin modificaciones en el código del motor de base de datos, como lo plantean los que defienden los sistemas orientados a columnas. Se da a entender que los sistemas de bases de datos orientados a filas, con décadas de investigación y desarrollo no pueden descartar tan pronto ya que son sistemas muy robustos y extensibles.

3.2 El dominio de aplicación

Para la evaluación de las consultas, se ha planteado la revisión de al menos dos escenarios o modelos de negocio, de los cuales se pueda obtener grandes cantidades de datos o registros históricos. Estos datos deben ser consistentes y estar limpios, es decir, que se puedan cargar al *DWH* , ni valores fuera de rango o valores nulos.

El propósito es comparar los escenarios en calidad y cantidad de registros. Una vez determinado cual es el apropiado se continua con el diseño del *Data Mart*, en el que se analizará el comportamiento de las consultas a ambos *DBMS*. Los escenarios que se consideraron fueron:

Primer escenario, una empresa de telefonía móvil denominada "*Colmovil*", con clientes y oficinas de atención en gran parte de territorio nacional, ofreciendo contratos a clientes para planes de voz, planes datos, servicio de *Roaming*, registro de las recargas de los clientes prepago y un flujo de operaciones es durante las 24 horas, que corresponde principalmente al registro de llamadas.

Segundo escenario, un sistema de registro académico, con información semestral, derivada de las admisiones en los diferentes programas académicos, las matriculas de los estudiantes, los profesores designados, las asignaturas programadas, las aulas disponibles, los estímulos y los bajos rendimientos académicos.

Las características de los dos escenarios se muestran en la tabla 2.

Escenario	Los datos están disponibles ?	Cantidad de Registros	Cantidad de tablas Aprox.	Origen de Datos?	Los datos están Limpios?	El Modelo Relacional se conoce?
Telefonía	Si	Millones en intervalos de tiempo cortos	12	Sintéticos	Si	Si
Registro Académico	Se requiere autorización	Miles cada semestre	10	Reales	No se sabe	No

Tabla 2: Características de los escenarios de prueba

Teniendo en cuenta estas características, se determina que el escenario apropiado para continuar con el diseño del *Data Mart*, en el que se van a ejecutar las consultas, es el de telefonía móvil. Las principales razones para su elección son el volumen de registros disponibles para la carga incremental y la posibilidad de generarlos en intervalos cortos de tiempo, además de poder cargar el *Data Mart* con datos que no requieren pre-procesamiento.

3.2.1 Definición del modelo Entidad-Relación

Una vez identificado el escenario donde se implementarán la pruebas, se construye el modelo entidad-relación de la base de datos operacional, como se muestra a continuación en la figura 2.

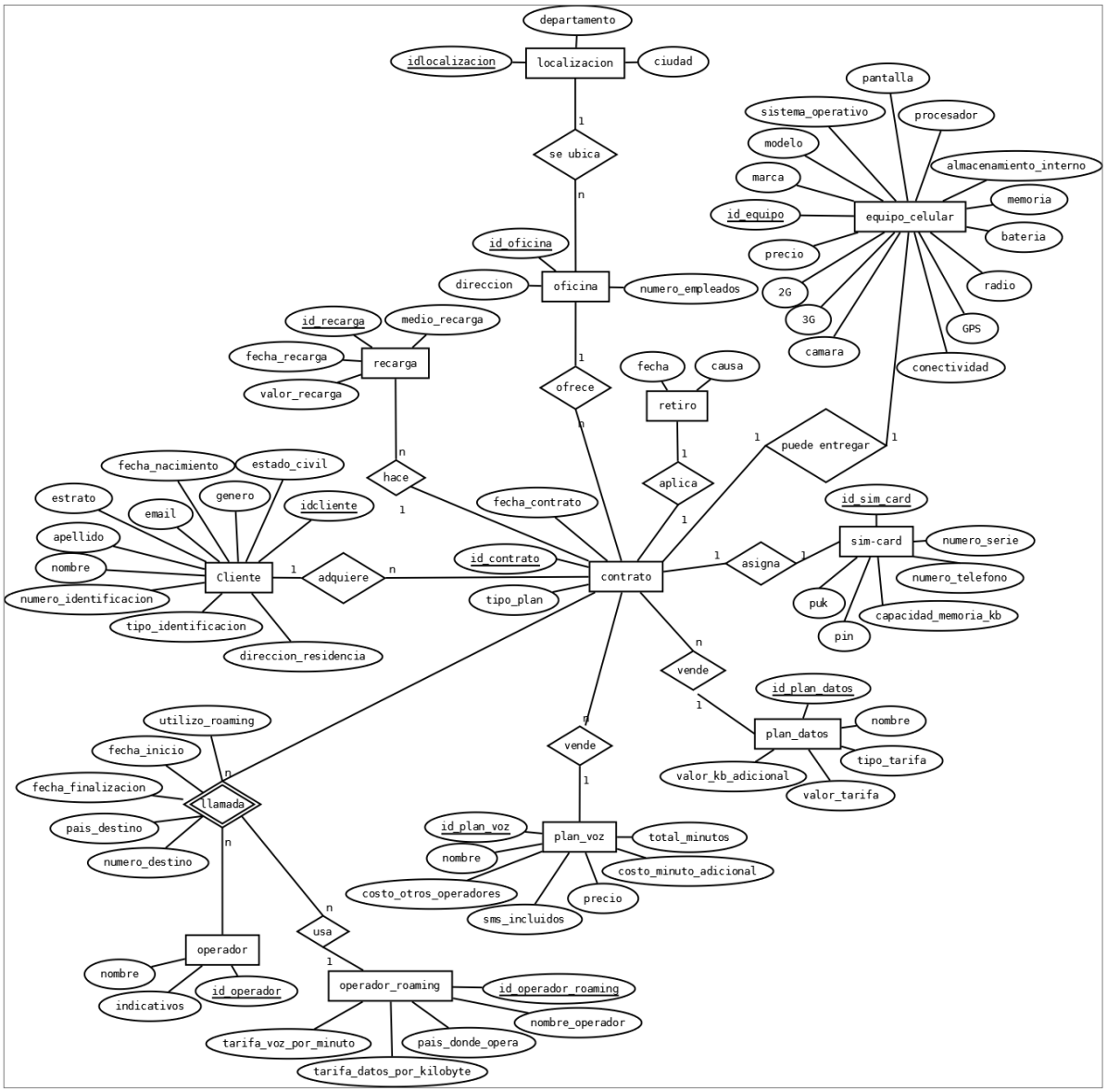


Figura 2: Modelo entidad-relación para el operador de telefonía móvil *'Colmovil'*

3.2.2 Diseño del modelo operacional.

Con base en el modelo *Entidad-Relación E-R,* se genera el esquema físico para la base de datos operacional, como se muestra en la figura 3. Este esquema se implementa en ambos sistemas manejadores de bases de datos *PostgreSQL* y *MonetDB*, para luego poblarlo de datos y realizar las pruebas de rendimiento.

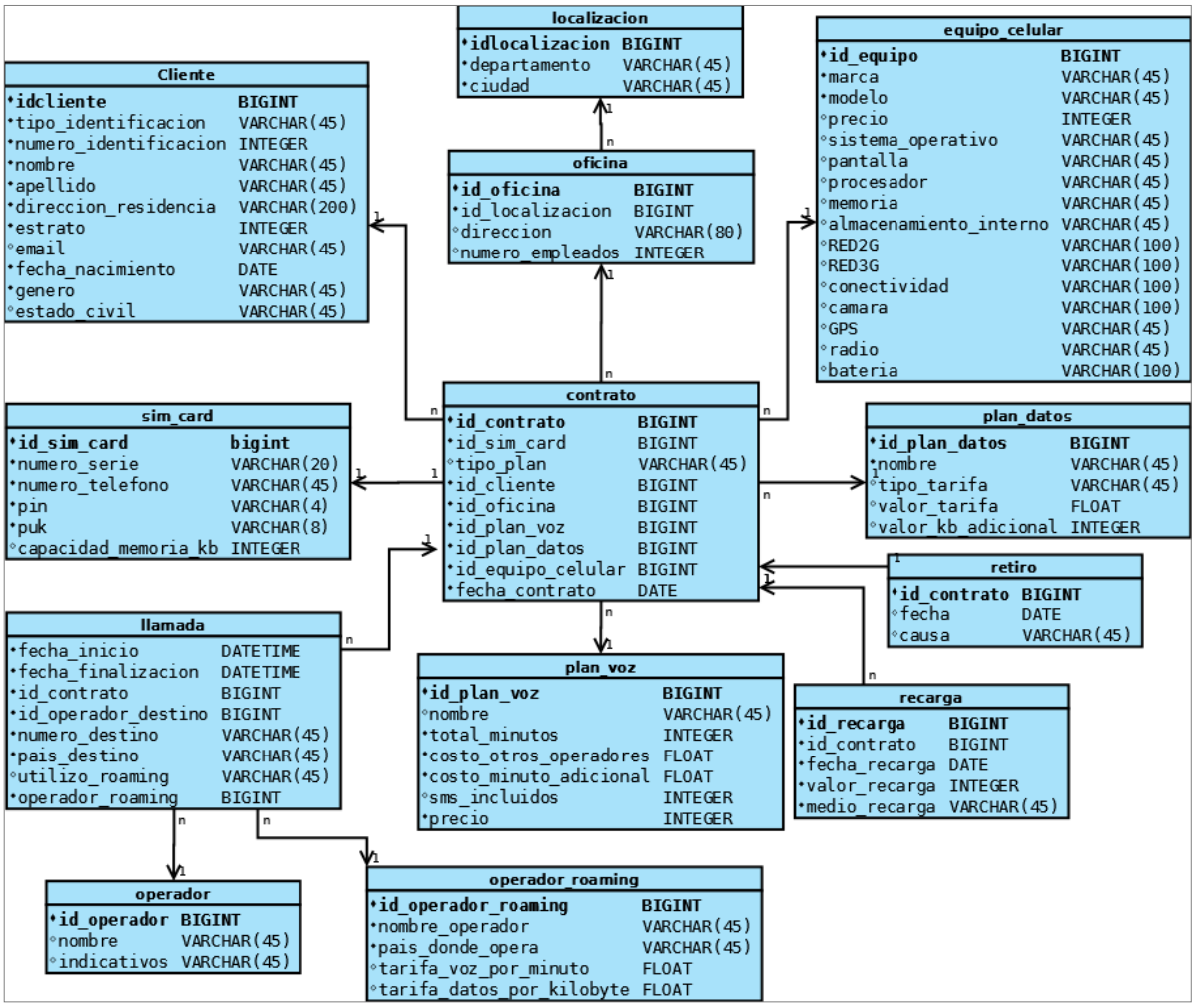


Figura 3: Modelo Físico del operador de telefonía móvil 'Colmovil'.

La implementación de este esquema de la base de datos operacional se realizó en ambos *DBMS*, como se describe en el Anexo 1.

3.2.3 Diseño del modelo multidimensional

El diseño de la base de datos del *DWH* , se modela usando el esquema en estrella. Técnica de diseño integra una tabla de hechos consumo, relacionada con siete dimensiones que permiten identificar el hecho propio en el transcurso del tiempo. La descripción del modelo dimensional se hace con más detalle en el Anexo 2. En la Figura 4, se muestra el esquema físico de la estrella "consumo".

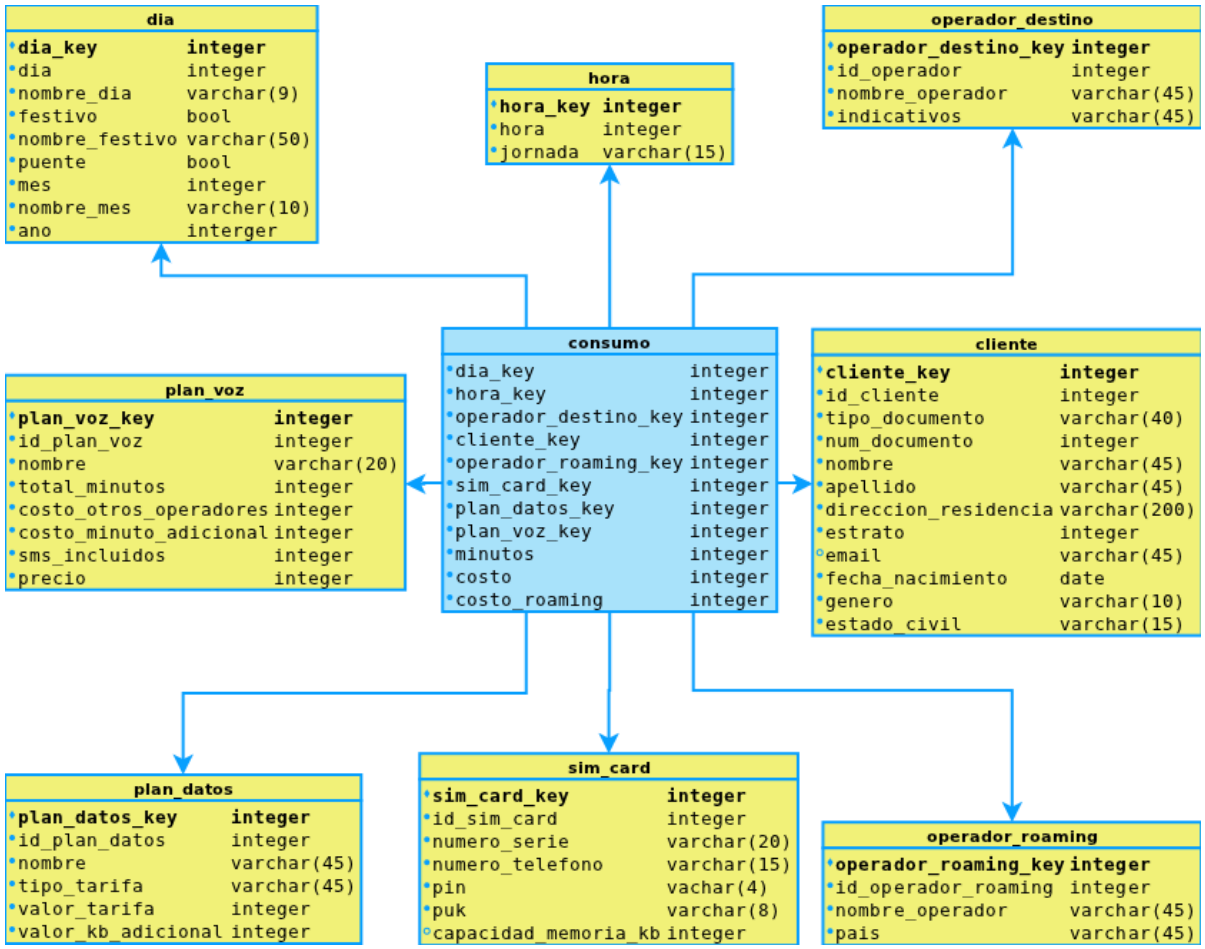


Figura 4: Modelo Físico del *Data Mart* Consumo, "estrella consumo"

El esquema dimensional se consideró, porque aparecen varios hechos: consumo de "minutos", "costo" de los minutos dependiendo de la tarifa del plan y "costo_roaming" que indica el costo en el caso de que la llamada realizada hiciera uso del servicio de Roaming a través de otro operador. La importancia de estos atributos de tipo numérico, es que posibilitan el uso de funciones de agregación y ordenamiento, que serán apropiadas para usarlas en las consultas de mayor exigencia para los *DBMS*. El código *SQL* de implementación se muestra en el Anexo 3.

3.3 Construcción y carga de los datos sintéticos

Los datos que se cargan en las bases de datos son sintéticos, es decir, son datos ficticios que se generan mediante el uso de funciones basadas en números aleatorios. Los datos que se usan en la base de datos operacional fueron tomados de un conjunto de datos sintéticos disponibles en [12]. Con base en estos datos, se cargan también las dimensiones del *Data*

Mart o estrella consumo, con excepción de las dimensiones *DIA y HORA* que no son entidades que existan en el modelo operacional.

La carga de datos de la tabla de hechos consumo, aunque hacen referencia a los registros que están en la tabla "llamada" del modelo operacional de *Colmovil*, no se tomaron de esta fuente, debido a que los datos disponibles en [12] eran pocos (500.000 aproximadamente) considerando la ventana de tiempo de dos años en el que se presentan. Para las pruebas de desempeño, se consideran incrementos de 500.000 registros en intervalos de tiempo no mayores a 1 mes. En este caso, se generaron 2 conjuntos de datos mensuales, cada uno con 500.000 registros (1.000.000 registros mensuales).

El uso de software propietario para la generación de datos aleatorios se descarta, por la necesidad de contar con licencias de uso [13], también, por un lado, se requiere que el generador tenga conexión a la base de datos, para garantizar las restricciones de llave foránea y esta característica algunos no la poseen[14] y por otro, por el nivel de detalle requerido para la generación de datos, dentro de periodos de tiempo específicos. Para generadores de libre distribución[14] las opciones de personalización ofrecidas no son suficientes y se tendría que rediseñar sus métodos para cumplir con las necesidades de los datos requeridos. Por estos motivos se decide, prescindir de generadores de datos de terceros y construir clases propias en *JAVA*, para la generación de datos sintéticos, basados en números aleatorios de distribución uniforme, respetando la integridad de los datos, la cantidad y los periodos de tiempo para los que se pretenden generar.(ver anexo 4).

3.3.1 Carga de datos en base de datos operacional

Para hacer la carga de las tablas en ambos sistemas, se utilizan datos previamente generados en archivos de texto, con valores separados por punto y coma, disponibles en [12]. La inserción de estos datos hace mediante el comando en *PostgreSQL*

copy nombre_tabla from 'archivo.csv' delimiter ';';

Igualmente en *MonetDB* se usa este mismo comando

copy into nombre_tabla from 'archivo.csv' delimiters ';','\n';

3.3.2 Carga de datos en los Data Mart

La carga de la mayoría de la dimensiones se puede hacer directamente de la base de datos operacional, excepto las dimensiones "*DIA*" y "*HORA*" las cuales no existen como entidad en la base de datos operacional y por tanto, se cargan dependiendo del periodo de tiempo que abarquen los consumos y teniendo en cuenta la granularidad o el nivel mínimo de detalle

para llevar el registro de los consumos, en este caso a nivel de horas. Como estos datos se deben generar, se diseñan clases en *JAVA* encargadas de esta tarea. (ver anexo 5)

A manera de ejemplo, se muestra como se carga la dimensión **dwh_colmovil.cliente,** mediante una consulta a la base de datos operacional en la tabla **colmovil.cliente**. Las otras dimensiones a excepción de **dwh_colmovil.dia** y **dwh_colmovil.hora** se cargan de la misma manera.

```
INSERT INTO dwh_colmovil.cliente
(ID_CLIENTE,
TIPO_DOCUMENTO,
NUM_DOCUMENTO,
NOMBRE,
APELLIDO,
DIRECCION,
ESTRATO,
EMAIL,
FECHA_NACIMIENTO,
GENERO,
ESTADO_CIVIL)
SELECT * FROM colmovil.cliente;
```

Para hacer la carga de la dimensión DIA, se generan los inserts correspondientes a la información de los días desde el año 2000 hasta el 2010. Con la ayuda de clases en *JAVA* con conexión a *PostgreSQL* y *MonetDB* se logra insertar cada tupla (anexo 5). Los inserts abarcan el lapso de tiempo necesario para dar soporte a los movimientos de los consumos que se generarán.

A diferencia de la mayoría de dimensiones, que se cargaron a partir de consultas a la base de datos operacional, la tabla de hechos se carga a partir de los datos sintéticos generados en archivos de texto con valores separados por punto y coma. Se elige este método de carga, porque es más rápido que hacer los inserts directamente desde *JAVA*.

En *PostgreSQL* : **copy nombre_tabla from 'archivo.csv' delimiter ';';**

En *MonetDB* : **copy into nombre_tabla from 'archivo.csv' delimiters ';','\n';**

3.4 Costo de las consultas

El costo de la evaluación de una consulta, se puede expresar en términos de diferentes recursos, incluyendo: los accesos a disco, el tiempo de *CPU* en ejecutar la consulta y en sistemas de bases de datos distribuidos o paralelos, el costo de la comunicación[10].
En el proceso de comparación de los dos *DBMS* empleados en el presente trabajo, se considera como medida para mostrar el costo de las consultas, el tiempo empleado por el *DBMS* para resolver la consulta, asegurándose que este tiempo, no se afecte de forma critica por otra actividad que esté ejecutándose en el sistema.

Los tiempos que tarden en resolver la consulta cada uno de los *DBMS*, se tabulan y se muestran en gráficos relacionando, los tiempos de respuesta obtenidos versus la cantidad de datos en la tabla de hechos, para permitir contrastar resultados y ver el comportamiento de los *DBMS* a medida que aumentan los datos.

Para el desarrollo de las pruebas comparativas en los dos *DBMS*, también se tiene en cuenta el uso de *CPU* durante la consulta como un segundo recurso para determinar el costo de evaluación de las consultas. Para poder conocer y después integrar esta información como evidencia comparativa, fue necesario usar el comando *pidstat* del sistema operativo Linux, que permite hacer seguimiento a los procesos que se ejecutan y a su respectivo consumo de recursos. Esta herramienta genera un archivo *LOG* con información del uso de *CPU* y memoria *RAM*, de este archivo se extrae la información que corresponde al proceso del *DBMS* y con esto es posible tabular y construir gráficos en el tiempo, que revelen el consumo de recursos de ambos *DBMS* mientras resolvían la consulta.

Para facilitar el proceso de comparación se plantea la construcción de un prototipo de aplicación que soporte esta tarea, permitiendo, la construcción y gestión de consultas, la ejecución de las consultas con el registro de los tiempos de respuesta y uso de recursos, la gestión de las pruebas y la visualización de los resultados para la comparación. (ver capitulo 5, para obtener detalles de la implementación).

4 PRUEBAS Y DISCUCIÓN DE RESULTADOS

El énfasis del trabajo se centró en el diseño y ejecución de las pruebas. Se trataba de evaluar el comportamiento de las consultas cuando aumentaba tanto el numero de dimensiones involucradas como de volumen de tuplas almacenadas. Adicionalmente en algunas consultas se implementaron funcionalidades que requerían de mayor procesamiento, esperando que alguno de los *DBMS* pudiese tomar ventaja, dadas las diferencias en su forma de almacenar y procesar los datos.

Los resultados se obtienen luego de ejecutar las consultas diseñadas para el esquema dimensional del *Data Mart*. La carga incremental se hace en la tabla de hechos, mientras, las dimensiones permanecen con una cantidad de registros fija. A continuación en la figura 5, se muestra un esquema simplificado de la estrella consumo, con las cantidades fijas de tuplas que contenían las dimensiones.

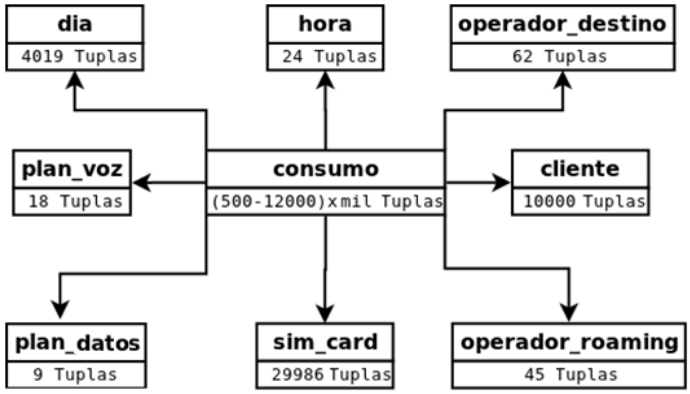


Figura 5: Cantidad de tuplas por dimensión

4.1 Diseño de las consultas y resultados

Las consultas se plantean para ser ejecutadas en la medida que la cantidad de registros en la tabla de hechos del *Data Mart* van incrementado. El objetivo de estos cambios, es que revelen gradualmente dificultad de procesamiento, ocasionando retrasos en la entrega de resultados de los *DBMS*. Las consultas pueden involucrar factores que incrementen la dificultad de procesamiento en los *DBMS*, en ese sentido, se incorporan funciones características de las consultas en *DWH* como agregación, ordenamiento y aumento en la cantidad de dimensiones involucradas.

A continuación, se muestran las consultas que prueban el desempeño de los *DBMS*, junto con los resultados obtenidos. Para las consultas, se ha querido especificar el resultado obtenido de cada una, dentro del contexto del dominio de aplicación (telefonía celular). Se describen características de los datos y se indican las dimensiones involucradas en el

procesamiento de la consulta. Las consultas se presentan en orden de menor a mayor dificultad, considerando la cantidad de dimensiones involucradas.

Los resultados de las consultas en *MonetDB* y *PostgreSQL*, se presentan en un gráfico para resaltar las diferencias en desempeño y luego en gráficos separados, para ver en detalle el comportamiento individual de cada DMBS a medida que incrementan los datos (note que en este gráfico individual, el eje que mide el tiempo de respuesta cambia de escala en cada gráfico, para poder ver en detalle el comportamiento). Todos los valores de los tiempos de respuesta usados en los gráficos, se presentan en el Anexo 8.

Consulta Q1
Resultado: Franja de mayor uso de la red.
Observaciones: Se asume que en la dimensión hora, cada hora pertenece a una jornada (mañana, tarde, noche ó madrugada); En la tabla de hechos consumo, los minutos consumidos por cada usuario se reportan según la hora de inicio de la llamada.
Dimensiones Involucradas: 1

```
select hora.jornada as franja, sum(consumo.minutos) as consumo_minutos
from dwh_colmovil.hora, dwh_colmovil.consumo
where hora.hora_key = consumo.hora_key
group by franja
order by consumo_minutos desc limit 1;
```

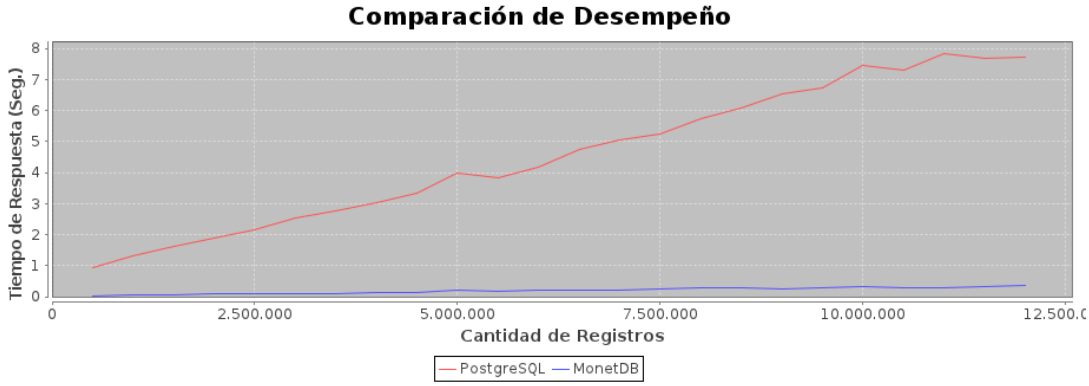
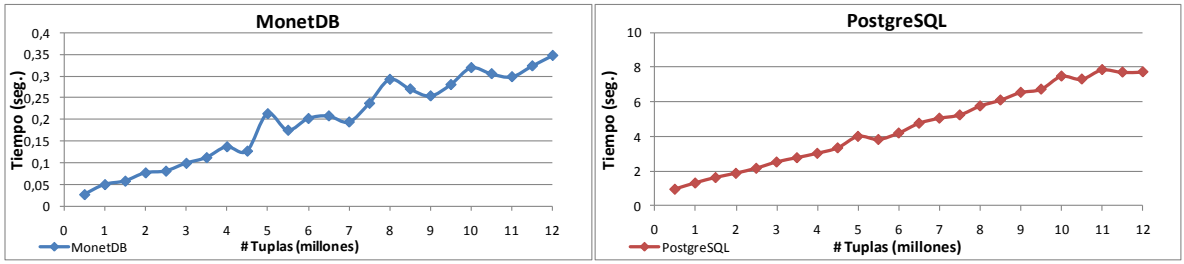


Figura 6: Tiempos de respuesta para Q1, *MonetDB* vs *PostgreSQL*



(a) (b)

Figura 7: Detalle de los tiempos de respuesta para Q1 en ambos *DBMS*
Figura (a)Tiempos de respuesta para Q1 vs. Cantidad de tuplas en *MonetDB*. Figura (b) Tiempos de respuesta para Q1 vs. Cantidad de tuplas de en *PostgreSQL*

Consulta Q2
Resultado: Consumo total de minutos por edad
Observaciones: El cálculo de la edad es aproximada; En la tabla de hechos consumo, los minutos consumidos se reportan por cada usuario.
Dimensiones Involucradas: 1

```
select ((extract (YEAR from CURRENT_DATE))-(extract(YEAR from
cliente.fecha_nacimiento))) as edad, sum(consumo.minutos) as minutos
from dwh_colmovil.cliente, dwh_colmovil.consumo
where cliente.cliente_key = consumo.cliente_key
group by edad
order by minutos desc;
```

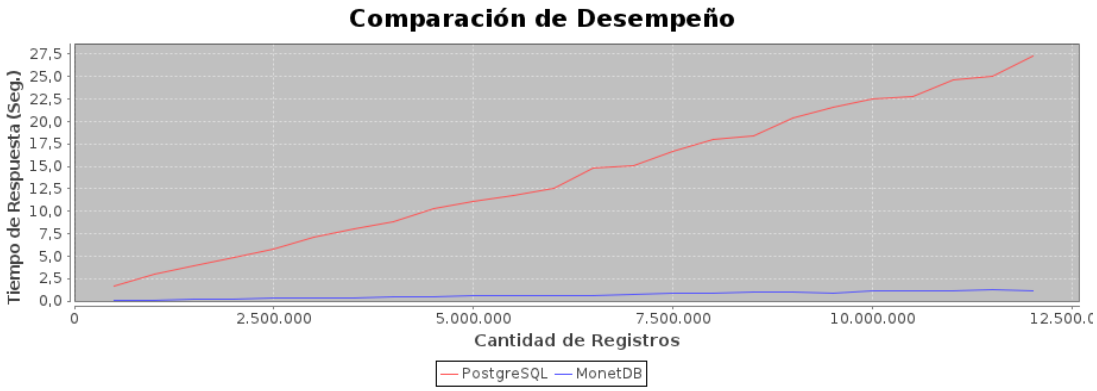
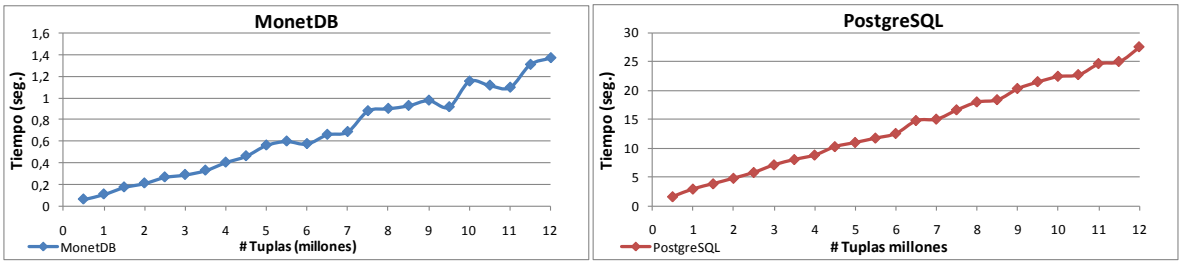


Figura 8: Tiempos de respuesta para Q2, *MonetDB* vs *PostgreSQL*



(a) (b)

Figura 9: Detalle de los tiempos de respuesta para Q2 en ambos *DBMS*
Figura (a)Tiempos de respuesta para Q2 vs. Cantidad de tuplas en *MonetDB*. Figura (b) Tiempos de respuesta para Q2 vs. Cantidad de tuplas de en *PostgreSQL*

Consulta Q3
Resultado: Operador nacional más utilizado
Observaciones: Como los consumos en la tabla de hechos están relacionados con las llamadas realizadas por los usuarios, entonces en el consumo reportado se especifica el nombre del operador destino al que se hace la llamada, sea nacional o internacional. En la tabla de hechos consumo, los minutos consumidos se reportan por cada usuario.
Dimensiones Involucradas: 1

```
select operador_destino.nombre_operador, sum(consumo.minutos) as cant_minutos
from dwh_colmovil.consumo, dwh_colmovil.operador_destino
where consumo.operador_destino_key=operador_destino.operador_destino_key and
consumo.costo_roaming=0
group by nombre_operador
order by cant_minutos desc limit 1;
```

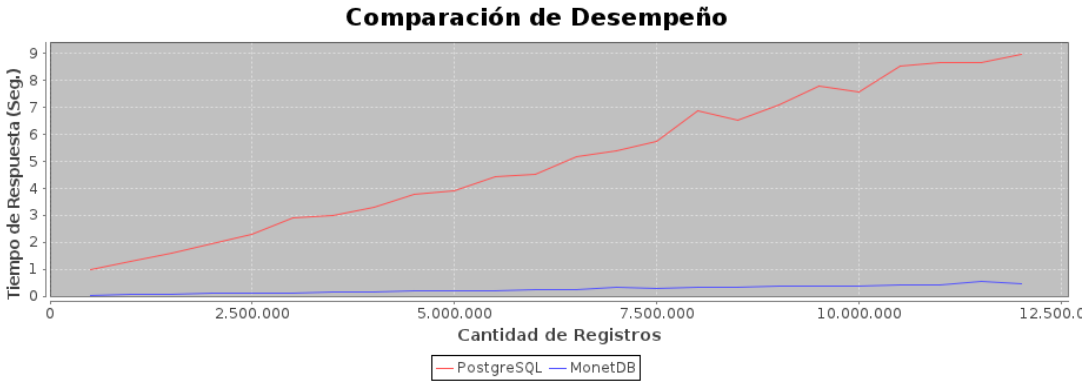
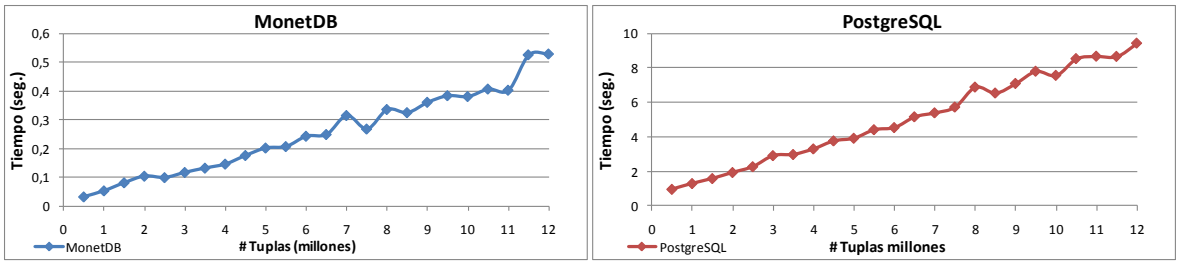


Figura 10: Tiempos de respuesta para Q3, *MonetDB* vs *PostgreSQL*



(a) (b)

Figura 11: Detalle de los tiempos de respuesta para Q3 en ambos *DBMS*
Figura (a)Tiempos de respuesta para Q3 vs. Cantidad de tuplas en *MonetDB*. Figura (b) Tiempos de respuesta para Q3 vs. Cantidad de tuplas de en *PostgreSQL*

Consulta Q4
Resultado: Tarifas de *Roaming* promedio por operador en cada país (sin ordenar)
Observaciones: Es posible que un usuario realice una llamada desde otro país, haciendo uso de una red de telefonía celular ajena a *Colmovil*. Por tal razón, se genera un cobro adicional por *roaming* en la llamada. Se asume que las tarifas *roaming* son variables, con la finalidad de forzar el cálculo de tarifas promedio. El costo *roaming* que aparece en la tabla de hechos, totaliza el costo de la llamada. En la consulta se calcula el costo por minuto.
Dimensiones Involucradas: 2

```
select operador_roaming.pais, operador_roaming.nombre_operador,
avg(consumo.costo_roaming/consumo.minutos) as costo
from dwh_colmovil.operador_roaming, dwh_colmovil.consumo, dwh_colmovil.dia
where operador_roaming.operador_roaming_key=consumo.operador_roaming_key and
dia.dia_key=consumo.dia_key and operador_roaming.operador_roaming_key<>1
group by pais, nombre_operador;
```

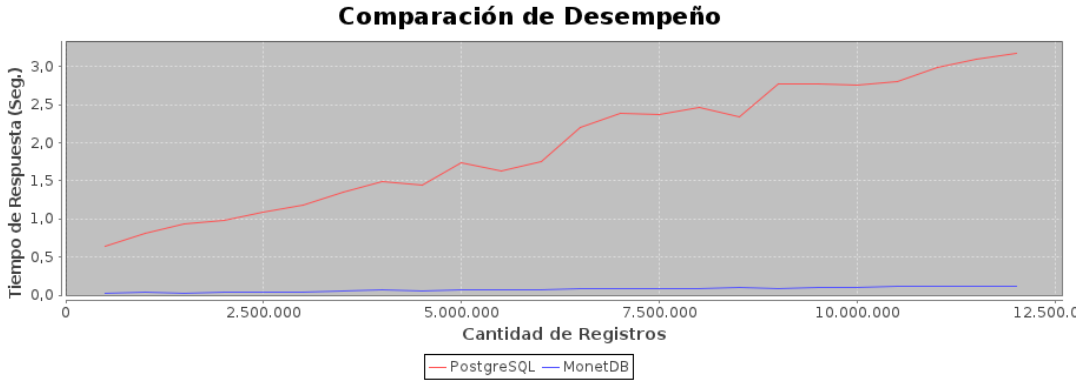
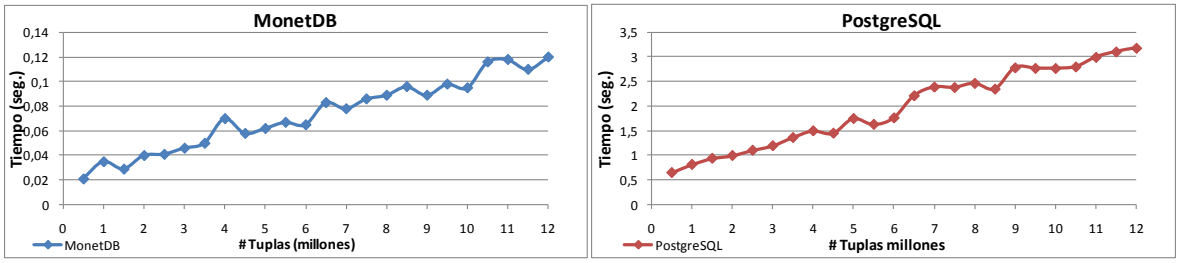


Figura 12: Tiempos de respuesta para Q4, *MonetDB* vs *PostgreSQL*



(a) (b)

Figura 13: Detalle de los tiempos de respuesta para Q4 en ambos *DBMS*
Figura (a)Tiempos de respuesta para Q4 vs. Cantidad de tuplas en *MonetDB*. Figura (b) Tiempos de respuesta para Q4 vs. Cantidad de tuplas de en *PostgreSQL*

Consulta Q5
Resultado: Países con mejores tarifas de *roaming* (ordenado)
Observaciones: Es posible que un usuario realice una llamada desde otro país, haciendo uso de una red de telefonía celular ajena a *Colmovil*, por tal razón se genera un cobro adicional por *roaming* en la llamada. Se asume que las tarifas *roaming* son variables, con la finalidad de forzar el cálculo de tarifas promedio. El costo *roaming* que aparece en la tabla de hechos totaliza el costo de la llamada. En la consulta se calcula el costo por minuto.
Dimensiones Involucradas: 2

```
select operador_roaming.pais, operador_roaming.nombre_operador,
avg(consumo.costo_roaming/consumo.minutos) as costo
from dwh_colmovil.operador_roaming, dwh_colmovil.consumo, dwh_colmovil.dia
where operador_roaming.operador_roaming_key=consumo.operador_roaming_key and
dia.dia_key=consumo.dia_key and operador_roaming.operador_roaming_key<>1
group by pais, nombre_operador
order by costo asc;
```

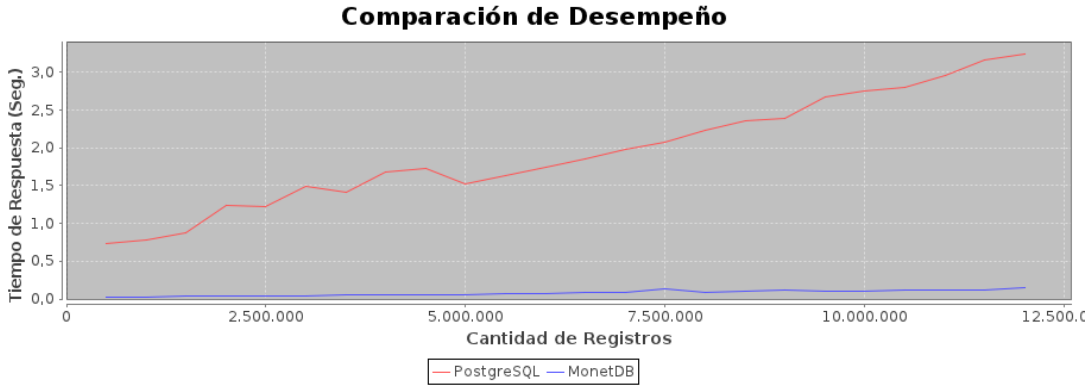
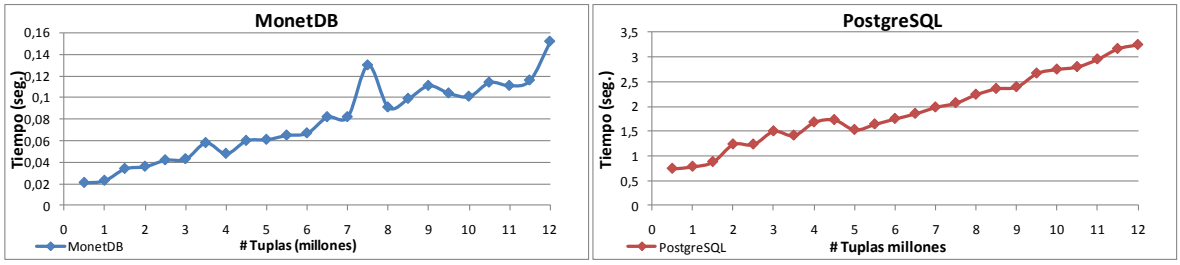


Figura 14: Tiempos de respuesta para Q5, *MonetDB* vs *PostgreSQL*



(a) (b)

Figura 15: Detalle de los tiempos de respuesta para Q5 en ambos *DBMS*
Figura (a)Tiempos de respuesta para Q5 vs. Cantidad de tuplas en *MonetDB*. Figura (b) Tiempos de respuesta para Q5 vs. Cantidad de tuplas de en *PostgreSQL*

Consulta Q6
Resultado: Días del año que más se llama, diferenciando género
Observaciones: Se totalizan las llamadas a nivel de día y se ordenan de mayor a menor cantidad de minutos consumidos.
Dimensiones Involucradas: 2

```
select dia.nombre_mes, dia.dia, cliente.genero, sum(consumo.minutos) as minutos
from dwh_colmovil.dia, dwh_colmovil.consumo , dwh_colmovil.cliente
where dia.dia_key = consumo.dia_key and cliente.cliente_key=consumo.cliente_key
group by dia.nombre_mes, dia.dia, cliente.genero
order by minutos desc, nombre_mes, dia limit 20;
```

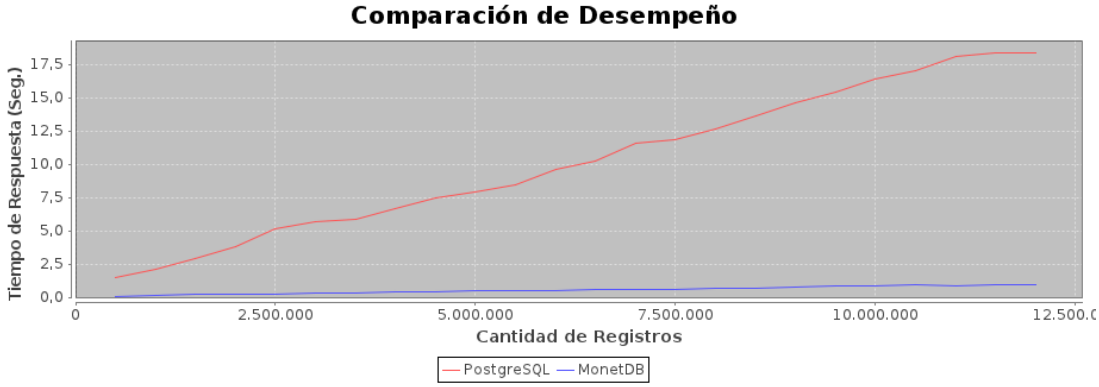
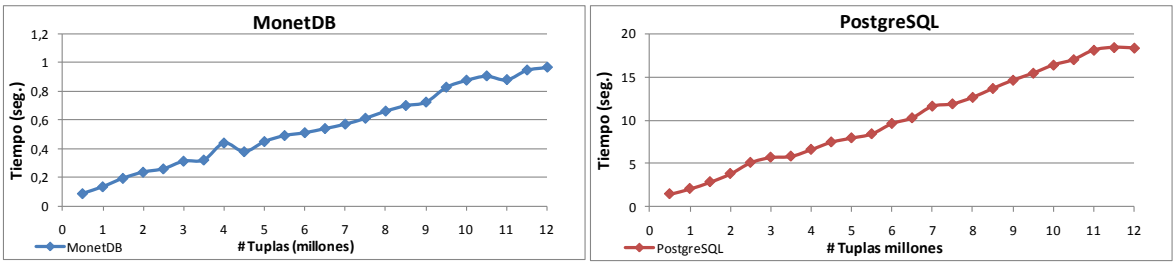


Figura 16: Tiempos de respuesta para Q6, *MonetDB* vs *PostgreSQL*



(a) (b)

Figura 17: Detalle de los tiempos de respuesta para Q6 en ambos *DBMS*
Figura (a)Tiempos de respuesta para Q6 vs. Cantidad de tuplas en *MonetDB*. Figura (b) Tiempos de respuesta para Q6 vs. Cantidad de tuplas de en *PostgreSQL*

Consulta Q7
Resultado: Consumo mensual de minutos con destino internacional diferenciando género
Observaciones: Se asume que de cada llamada se registra el destino, ya sea nacional o internacional
Dimensiones Involucradas: 3
Q7:

```
select dia.ano, dia.mes, dia.nombre_mes, cliente.genero, sum(consumo.minutos) as
minutos
from dwh_colmovil.consumo, dwh_colmovil.dia, dwh_colmovil.cliente,
dwh_colmovil.operador_destino
where consumo.cliente_key=cliente.cliente_key and consumo.dia_key=dia.dia_key
and operador_destino.operador_destino_key=consumo.operador_destino_key
and operador_destino.pais<>'Colombia'
group by ano, mes, nombre_mes, genero
order by ano asc, mes asc;
```

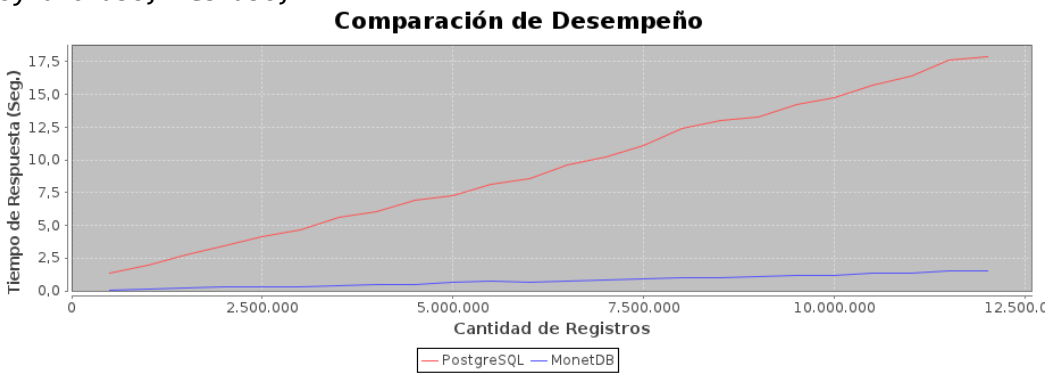
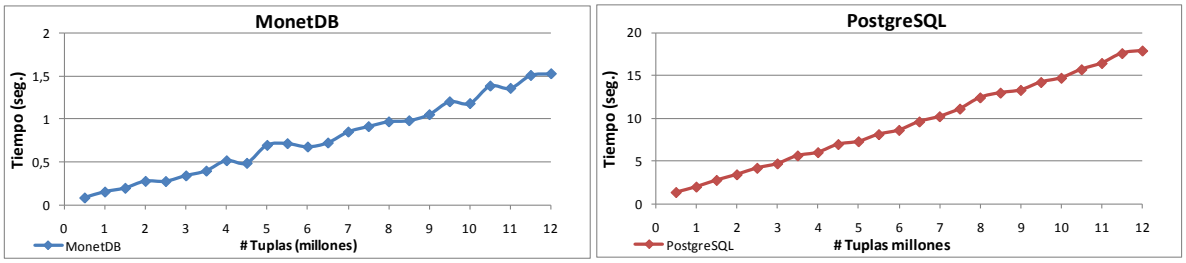


Figura 18: Tiempos de respuesta para Q7, *MonetDB* vs *PostgreSQL*



(a) (b)

Figura 19: Detalle de los tiempos de respuesta para Q7 en ambos *DBMS*
Figura (a)Tiempos de respuesta para Q7 vs. Cantidad de tuplas en *MonetDB*. Figura (b) Tiempos de respuesta para Q7 vs. Cantidad de tuplas de en *PostgreSQL*

Consulta Q8
Resultado: Consumo mensual de minutos por usuario con plan de voz
Observaciones: Se totaliza (suma) de las llamadas realizadas por cada línea telefónica en el mes.
Dimensiones Involucradas: 4

```
select sim_card.num_telefono, cliente.genero, dia.ano, dia.mes, dia.nombre_mes,
sum(consumo.minutos) as minutos
from dwh_colmovil.sim_card, dwh_colmovil.cliente, dwh_colmovil.dia,
dwh_colmovil.consumo, dwh_colmovil.plan_voz
where consumo.cliente_key=cliente.cliente_key and consumo.dia_key=dia.dia_key and
sim_card.id_sim_card=consumo.sim_card_key
and plan_voz.plan_voz_key=consumo.plan_voz_key and
plan_voz.nombre_plan<>'Sin plan de voz'
group by sim_card.num_telefono, genero, ano, mes, nombre_mes;
```

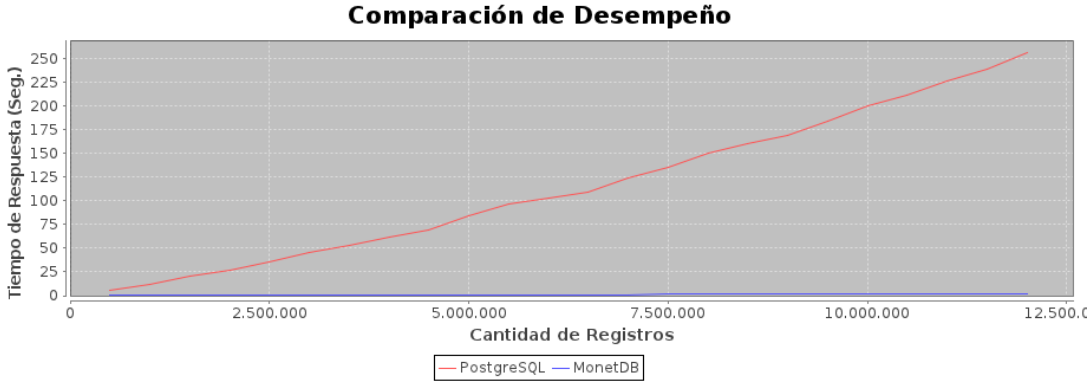
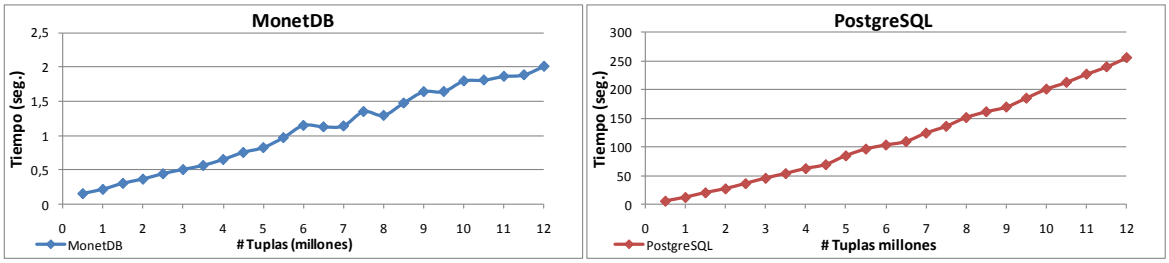


Figura 20: Tiempos de respuesta para Q8, *MonetDB* vs *PostgreSQL*



(a) (b)

Figura 21: Detalle de los tiempos de respuesta para Q8 en ambos *DBMS*
Figura (a)Tiempos de respuesta para Q8 vs. Cantidad de tuplas en *MonetDB*. Figura (b) Tiempos de respuesta para Q8 vs. Cantidad de tuplas de en *PostgreSQL*

Consulta Q9
Resultado: Consumo mensual de minutos por usuario con plan de voz
Observaciones: Se totaliza (suma) de las llamadas realizadas por cada línea telefónica en el mes, pero esta vez por medio de una vista, para observar que efectos tiene en el desempeño.
Dimensiones Involucradas: 4

```
select num_telefono, genero, ano, mes, nombre_mes, minutos
from dwh_colmovil.consumo_meses;
```

Definición de la vista usada a partir de Q8:

```
create view dwh_colmovil.consumo_meses as
select sim_card.num_telefono, cliente.genero, dia.ano, dia.mes, dia.nombre_mes,
sum(consumo.minutos) as minutos
from dwh_colmovil.sim_card, dwh_colmovil.cliente, dwh_colmovil.dia,
dwh_colmovil.consumo, dwh_colmovil.plan_voz
where consumo.cliente_key=cliente.cliente_key and consumo.dia_key=dia.dia_key and
sim_card.id_sim_card=consumo.sim_card_key
and plan_voz.plan_voz_key=consumo.plan_voz_key and
plan_voz.nombre_plan<>'Sin plan de voz'
group by sim_card.num_telefono, genero, ano, mes, nombre_mes;
```

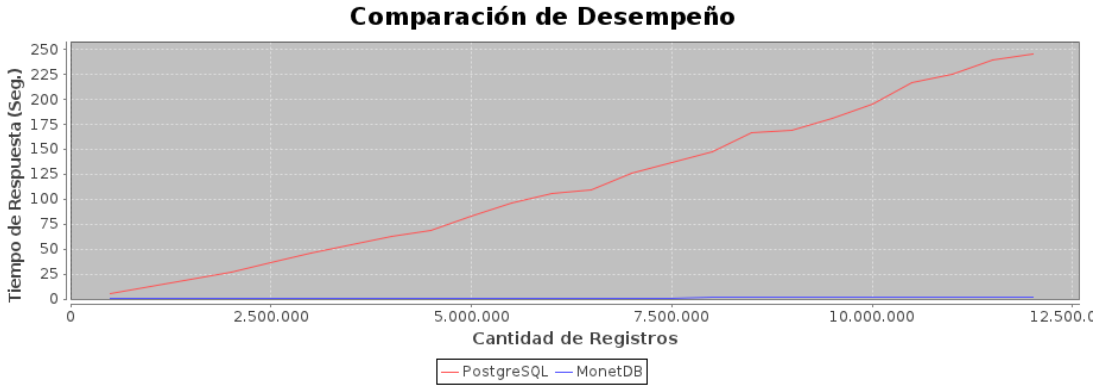
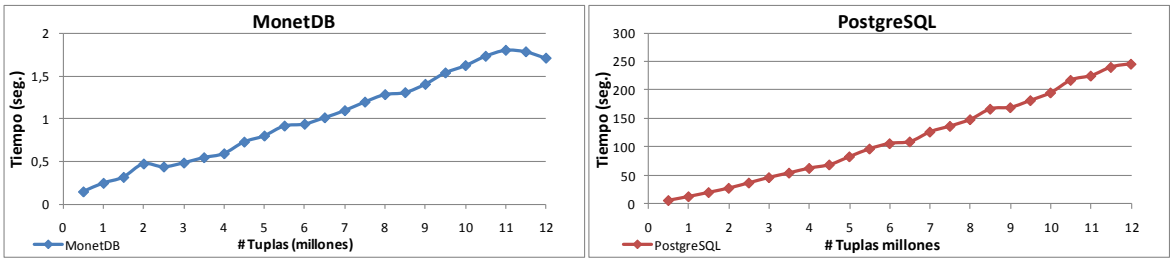


Figura 22: Tiempos de respuesta para Q9, *MonetDB* vs *PostgreSQL*



(a) (b)

Figura 23: Detalle de los tiempos de respuesta para Q9 en ambos *DBMS*
Figura (a)Tiempos de respuesta para Q9 vs. Cantidad de tuplas en *MonetDB*. Figura (b) Tiempos de respuesta para Q9 vs. Cantidad de tuplas de en *PostgreSQL*

Consultas Q10
Resultado: *Consumo promedio mensual de minutos por genero de los que tienen plan de voz.
Observaciones: Usando la vista definida en Q9, donde se totaliza (suma) de las llamadas realizadas por cada línea telefónica en el mes y calculamos a partir de la vista, el promedio de los consumos mensuales.
Dimensiones Involucradas: 4

```
(Se usa la vista definida en Q9)
select  consumo_meses.mes, consumo_meses.nombre_mes, consumo_meses.genero,
cast(avg(consumo_meses.minutos) as integer) as prom_minutos
from dwh_colmovil.consumo_meses
group by mes, nombre_mes, genero
order by mes;
```

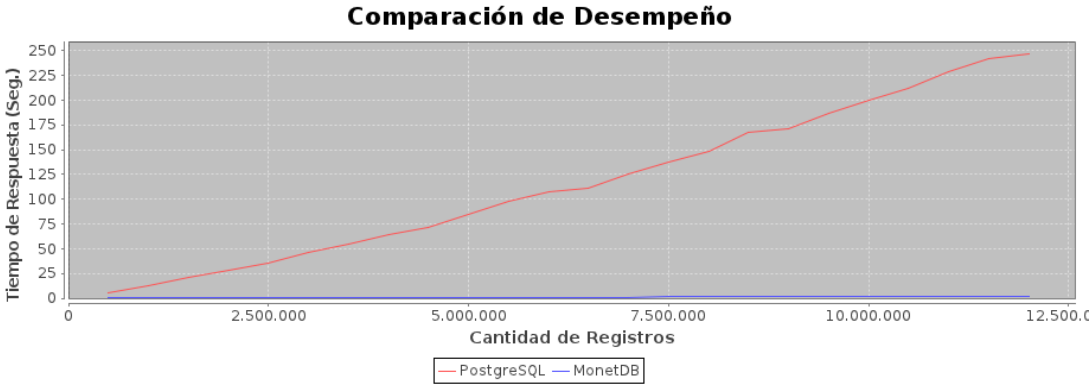
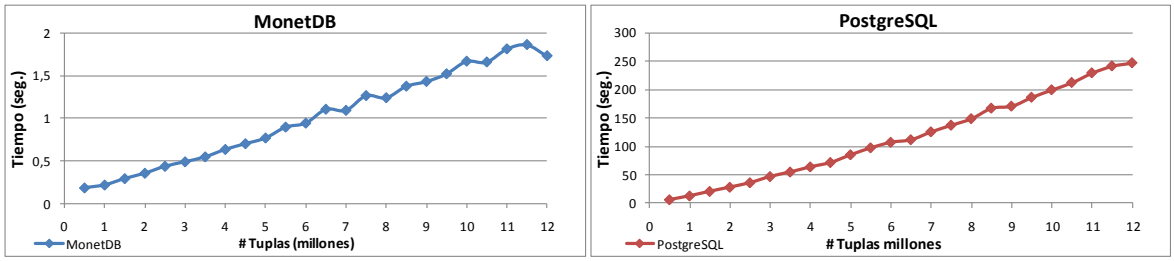


Figura 24: Tiempos de respuesta para Q10, *MonetDB* vs *PostgreSQL*



(a) (b)

Figura 25: Detalle de los tiempos de respuesta para Q10 en ambos *DBMS*
Figura (a)Tiempos de respuesta para Q10 vs. Cantidad de tuplas en *MonetDB*. Figura (b) Tiempos de respuesta para Q10 vs. Cantidad de tuplas de en *PostgreSQL*

Consulta Q11
Resultado: Consumo de minutos mensuales de los usuarios que tienen plan de voz y tienen plan datos.
Observaciones: Se totaliza (suma) por cada línea telefónica las llamadas realizadas a nivel de mes y se incluyen las dimensiones *plan_voz y plan_datos*, para poder filtrar el consumo de los que tienen estos planes.
Dimensiones Involucradas: 5

```
select sim_card.num_telefono, cliente.genero, dia.ano, dia.mes, dia.nombre_mes,
sum(consumo.minutos) as minutos
from dwh_colmovil.consumo, dwh_colmovil.sim_card, dwh_colmovil.cliente,
dwh_colmovil.dia, dwh_colmovil.plan_voz, dwh_colmovil.plan_datos
where consumo.cliente_key=cliente.cliente_key and consumo.dia_key=dia.dia_key
and sim_card.id_sim_card=consumo.sim_card_key
and plan_voz.plan_voz_key=consumo.plan_voz_key
and  plan_voz.nombre_plan<>'Sin plan de voz'
and plan_datos.plan_datos_key=consumo.plan_datos_key
and plan_datos.nombre<>'Sin plan de datos'
group by sim_card.num_telefono, genero, ano, mes, nombre_mes;
```

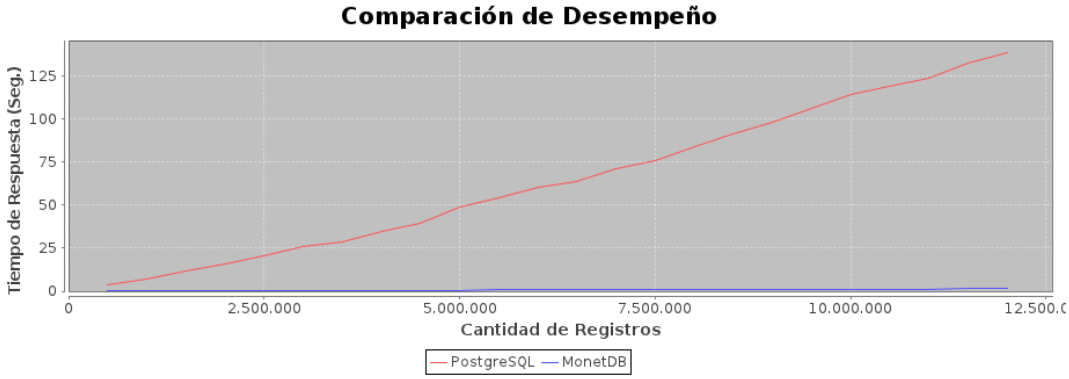
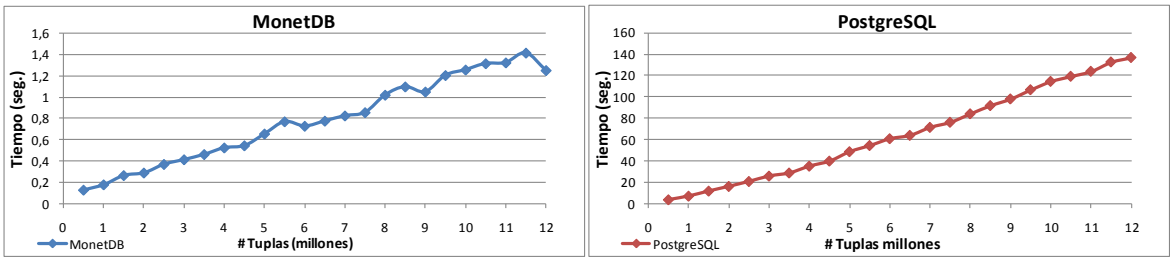


Figura 26: Tiempos de respuesta para Q11, *MonetDB* vs *PostgreSQL*



(a) (b)

Figura 27: Detalle de los tiempos de respuesta para Q11 en ambos *DBMS*
Figura (a)Tiempos de respuesta para Q11 vs. Cantidad de tuplas en *MonetDB*. Figura (b) Tiempos de respuesta para Q11 vs. Cantidad de tuplas de en *PostgreSQL*

Consulta Q12
Resultado: Consumo promedio de minutos mensuales de los usuarios que tienen plan de voz y tienen plan datos diferenciando genero.
Observaciones: Se define una vista con la consulta Q11, para aplicarle la función promedio.
Dimensiones Involucradas: 5

```
select  consumo_mes_planes.mes, consumo_mes_planes.nombre_mes,
consumo_mes_planes.genero, cast(avg(consumo_mes_planes.minutos) as bigint) as
prom_minutos
from dwh_colmovil.consumo_mes_planes
group by mes, nombre_mes, genero
order by mes;

Definición de la vista usada a partir de Q11:
create view dwh_colmovil.consumo_mes_planes as
select sim_card.num_telefono, cliente.genero, dia.ano, dia.mes, dia.nombre_mes,
sum(consumo.minutos) as minutos
from dwh_colmovil.consumo, dwh_colmovil.cliente, dwh_colmovil.dia,
dwh_colmovil.sim_card, dwh_colmovil.plan_voz, dwh_colmovil.plan_datos
where consumo.cliente_key=cliente.cliente_key and consumo.dia_key=dia.dia_key
and sim_card.id_sim_card=consumo.sim_card_key
and plan_voz.plan_voz_key=consumo.plan_voz_key
and  plan_voz.nombre_plan<>'Sin plan de voz'
and plan_datos.plan_datos_key=consumo.plan_datos_key
and plan_datos.nombre<>'Sin plan de datos'
group by sim_card.num_telefono, genero, ano, mes, nombre_mes;
```

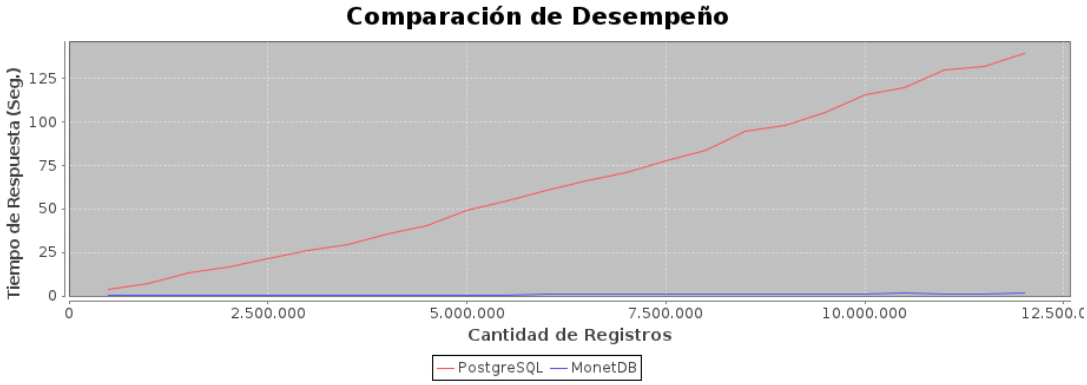
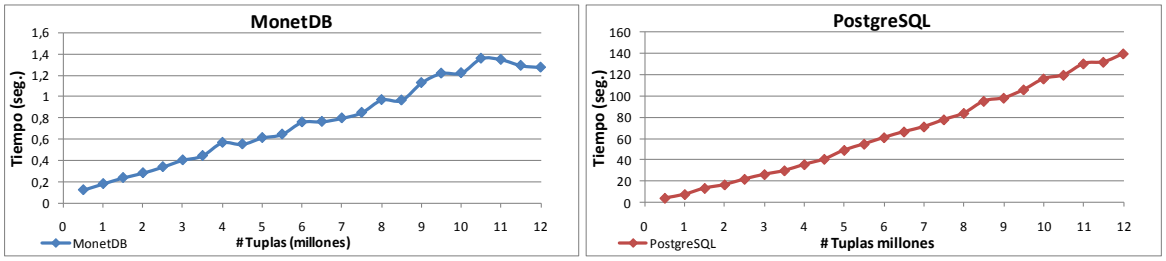


Figura 28: Tiempos de respuesta para Q12, *MonetDB* vs *PostgreSQL*



(a) (b)

Figura 29: Detalle de los tiempos de respuesta para Q12 en ambos *DBMS*
Figura (a)Tiempos de respuesta para Q12 vs. Cantidad de tuplas en *MonetDB*. Figura (b) Tiempos de respuesta para Q12 vs. Cantidad de tuplas de en *PostgreSQL*

4.2 Discusión de resultados

Los resultados obtenidos en las 12 consultas planeadas para probar el desempeño de los *DBMS*, ratifican que *MonetDB* está diseñado para dar soporte a bases de datos analíticas. Con los tiempos de respuesta de ambos *DBMS*, en ninguno de los casos se igualaron o trataron de por lo menos de acercarse. El desempeño superior de *MonetDB* no se vio afectado significativamente por el incremento de los datos como *PostgreSQL*. En el peor de los casos, en la consulta Q8 (con 12 millones de tuplas), *MonetDB* tardó 2 segundos, mientras *PostgreSQL* lo hizo en 255 segundos (4.2 min.).

Para obtener un forma de estimación de los tiempos que toma una consulta en ejecutarse se hizo un análisis de regresión lineal, basado en la herramienta[1] de análisis matemático disponible en [16].

La entrada a esta función es conjunto de puntos de la forma (x,y), donde x es el total de tuplas en millones e y el tiempo de respuesta. Estos puntos fueron obtenidos empíricamente como se muestra en la tabla del anexo 8.

[1] *http://www.wolframalpha.com/*

La formula general de la función lineal que aproxima el tiempo de respuesta es:

$$tr = aX + b$$

donde **a** es la pendiente y **b** el punto de corte con el eje y.

La función lineal aproximada para tiempos de respuesta en *MonetDB* usando la consulta Q8 es:

tr = 168.9X + 21.8 [ms], para incrementos de x en 1 por cada millón de tuplas

y la entrada para [16] fue:

linear fit {1,220},{2,368},{3,507},{4,653},{5,825},{6,1147},{7,1139},{8,1292},{9,1636},{10,1791},{11,1858}, {12,2003}

La función lineal aproximada para tiempos de respuesta en *PostgreSQL*, usando la consulta Q8 es:

tr = 22033.7X-21541.6 [ms], para incrementos de x en 1 por cada millón de tuplas

y la entrada para [16] fue:

linear fit {1,12159},{2,27113},{3,45482},{4,62258},{5,84421},{6,103237},{7,123878},{8,150883},{9,169135}, {10,200320},{11, 226289}, {12,254958}

Las funciones estimadas evidencian con sus pendientes, que el hecho de cargar incrementalmente las tablas afecta considerablemente el desempeño de *PostgreSQL* en contraste con el poco cambio que se puede presentar en *MonetDB*. De continuar así, por ejemplo al tener 160 millones de tuplas, el tiempo de respuesta en *MonetDB* será de 27.05 segundos aprox. mientras que en *PostgreSQL* será de 58.4 minutos aprox. Una diferencia que empieza a percibirse por poco en horas de retraso.

Con base en los resultados obtenidos, se puede afirmar que los avances en materia de optimización de recursos y las técnicas usadas para el almacenamiento de datos en el enfoque orientado a columnas, son el soporte para que se pueda lograr resultados superiores a los que se alcanzan en un *DBMS* tradicional. Esta afirmación se hace desde el punto de vista de la evaluación para entornos de trabajo como los *DWH* , debido a que todas las pruebas realizadas hasta este punto tenían el propósito de experimentar la extracción de datos y de encontrar cual enfoque tiene mejor soporte para las bases de datos orientadas a la lectura (bases de datos analíticas).

De los resultados obtenidos en *MonetDB* para el conjunto de datos de prueba, se podría decir que no son inesperados, si se tiene en cuenta la revisión de la literatura correspondiente a la optimización de consultas[3]. Se puede concluir que un factor influyente

para la obtención de buenos resultados en sistemas orientados a columnas, es la capacidad de comprimir mejor los datos, tarea que se facilita en esquemas de compresión que operan en presencia de datos numéricos. Esto es bastante común en las bases de datos de los *Data Mart*, donde se tiene una tabla de hechos que es el principal contenedor de los datos y cuya estructura se define a partir de datos puramente numéricos. En el caso particular de este estudio, la tabla de hechos consumo de la figura 30, sirve como ejemplo para verificar esta característica puesto que los hechos incluidos *son minutos, costo y costo_roaming*, todas de tipo *Integer*

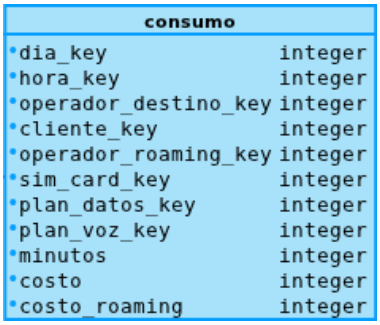


Figura 30: Estructura de la tabla de hechos consumo

Para lograr una buena compresión, también es importante tener datos que sean repetitivos en las columnas, ya que es la base de la mayoría de los algoritmos de compresión, los cuales buscan ocurrencias de un dato y lo reemplazan por una representación que ocupe menos espacio ó también buscan agrupaciones de datos con el mismo valor para representarlos, especificando el número de repeticiones presente en el grupo y la posición donde se inicia la repetición. De esta manea se evita la necesidad de almacenar todos estos mismos valores físicamente. Esta característica también es frecuente en los esquemas de los *Data Mart*. Aunque, las tabla de hechos tiene en su mayoría llaves foráneas de las dimensiones, estas dimensiones por lo general son muy pequeñas en cuanto a cantidad de datos contenidos, si se compara con la tabla de hechos que acompañan. Por tanto, los valores de las llaves foráneas se repiten con frecuencia. También en este caso, los hechos (*minutos, costo y costo_roaming*), no tienen un dominio de valores muy amplio, por lo que es posible que se presenten repeticiones.

- **Uso de recursos de Hardware durante las consultas.**

En vista de la marcada diferencia en desempeño, presentada por los dos *DBMS* propuestos para la evaluación. Se experimentó el uso de herramientas en el sistema operativo, para medir o cuantificar el uso de recursos como *CPU* y memoria *RAM* en el procesamiento de las consultas. Para esto se uso el comando *pidstat* de *Linux*. Los resultados obtenidos se muestran en las figuras 31 y 32.

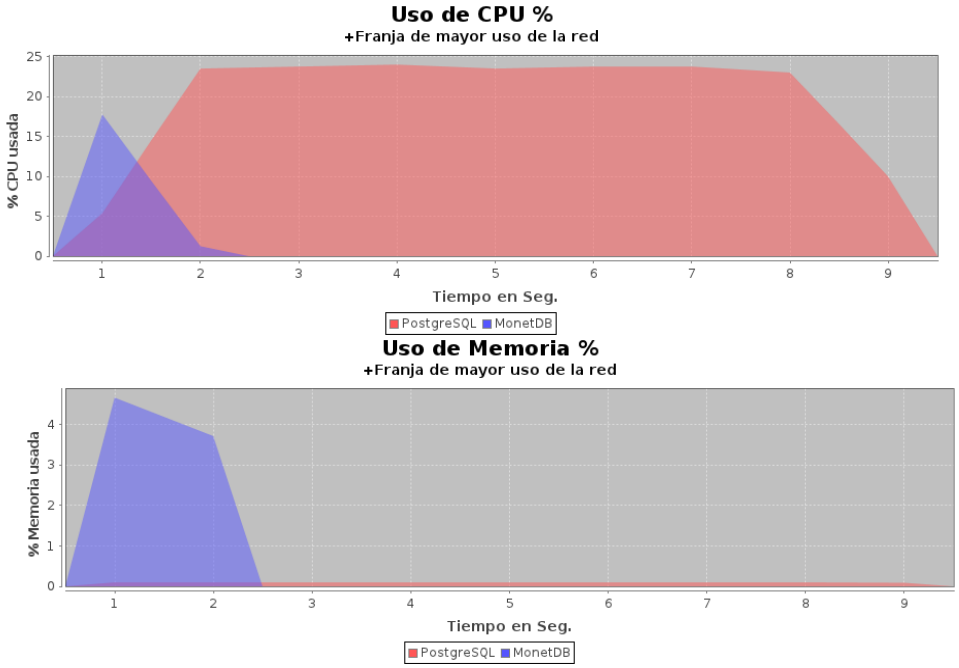


Figura 31: Uso de *CPU* y Memoria en los *DBMS*, para Q1 con 12 millones de registros en la tabla de hechos

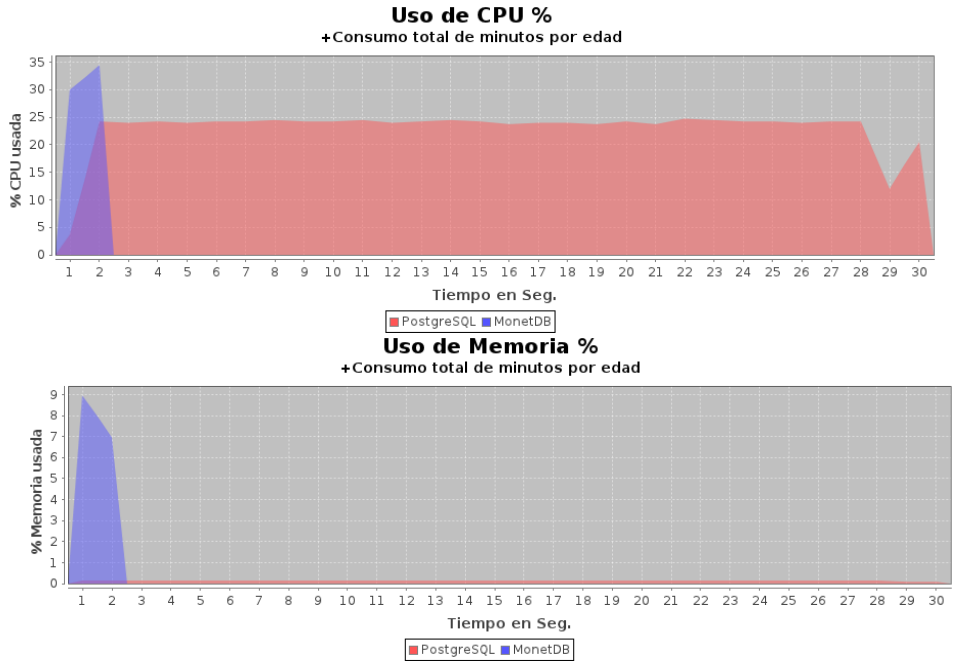


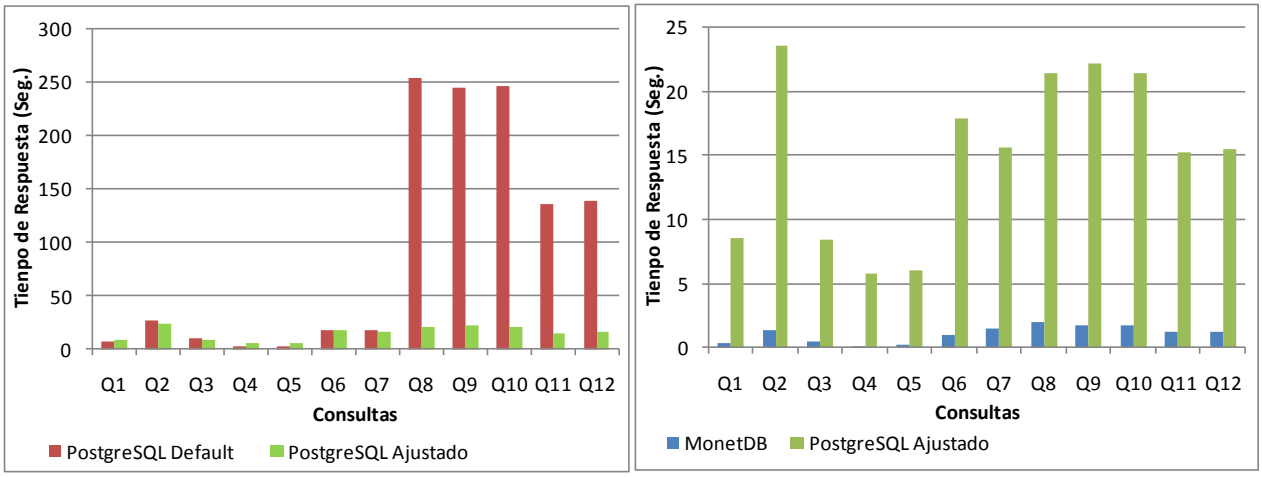
Figura 32: Uso de *CPU* y Memoria en los *DBMS*, para Q2 con 12 millones de registros en la tabla de hechos

A partir de las figuras 31 y 32, se puede concluir que *MonetDB* está diseñado para trabajar y sacar provecho de nuevas arquitecturas de hardware con multi-procesador y multi-hilos (ver características del equipo de pruebas en anexo 7). Esta es una de las optimizaciones

destacadas en beneficio de las cargas de trabajo de los *Data Warehouse*[5], donde plantean que *MonetDB* posee un diseño consiente de las nuevas arquitecturas, capaz de aprovechar los recursos disponibles de manera eficiente. Caso contrario a *DBMS* tradicionales que se construyeron para ser soportados en arquitecturas de hardware con características diferentes a las actuales, lo que no le permite usar todo el hardware para la ejecución de una consulta.

Como *PostgreSQL* no soporta multi-hilos[19], en las pruebas experimentales, contando con un equipo de pruebas con procesador capaz de ejecutar 4 hilos al tiempo. Nunca se registró que aprovechara más del 25% del uso de *CPU*, como se muestra en las figuras 31 y 32. El beneficio que puede obtener *PostgreSQL* de los recursos en las nuevas arquitecturas, es debido al multi-procesador. En el caso de entornos concurrentes como los operacionales, *PostgreSQL* asigna a cada transacción un proceso dentro del sistema operativo, razón por lo cual puede ejecutar varias transacciones a la vez, si dispone de un equipo servidor multi-procesador.

En cuanto al uso de memoria, se puede apreciar en las figuras 31 y 32, que *MonetDB* en sus procesos de ejecución de consultas, tiene un uso de memoria mayor al de *PostgreSQL*, pero en periodos más cortos. Ajustando parámetros que involucraban aumento del uso de memoria principal en *PostgreSQL*, se obtuvieron tiempos de respuesta para las consultas, con 12 millones de registros, como se muestra en la figura 33.



(a) (b)

Figura 33: Tiempos de respuesta ajustando *PostgreSQL* para las consultas, con 12 millones de registros en la tabla de hechos

La figura (a) muestra el comparativo de los tiempos de respuesta para *PostgreSQL* con la configuración por defecto y con justes para que use más memoria. La figura (b) muestra el comparativo de los tiempos de respuesta para *MonetDB* configurado por defecto y *PostgreSQL* con justes para que use más memoria.

Después de ajustar la configuración de *PostgreSQL* para que usara el 70% aprox. de la memoria principal, se pudo obtener tiempos de respuesta que mejoran el desempeño con

respecto a cuando usaba su configuración por defecto, aunque no lo suficiente como para superar el desempeño de *MonetDB*. La mejora es notoria en las consultas que hacen agregación con atributos muy selectivos, es decir que no compactan mucho la información, mientras que para las consultas que condensaban más la información haciendo uso de funciones de agregación, el desempeño fue casi el mismo.

Una de las consultas que más se benefició con el ajuste de poder usar más memoria, fue la consulta Q8, para la cual se muestra el uso de memoria durante la consulta en la figura 34.

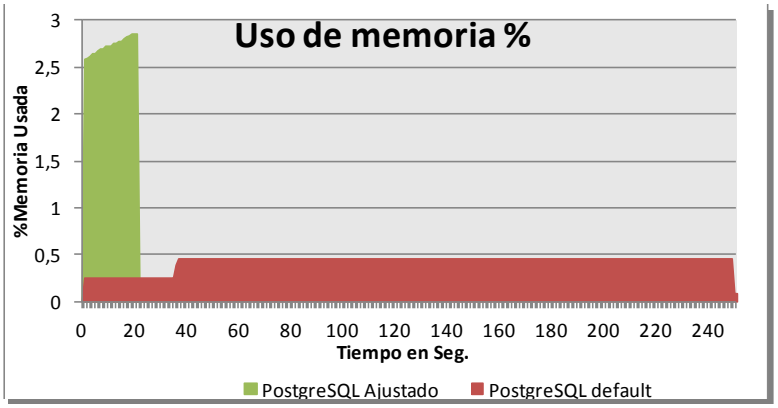


Figura 34: Uso de memoria de *PostgreSQL*, con la configuración por defecto y con justes para que use más memoria, durante la ejecución de Q8 con 12 millones de registros

En la figura 34, se observa que *PostgreSQL* pudo utilizar más memoria para procesar la consulta y reducir el tiempo de respuesta, pero también es importante resaltar que aun cuando se ajusto *PostgreSQL* para que pudiera usar el 70% de la memoria principal, este solo usó un 3% aproximadamente. Lo que indica que el procesamiento de toda la consulta no puede ser alojado en memoria principal y está usando memoria virtual para construir todo el conjunto de datos.

- **Análisis del aumento de las dimensiones involucradas**

Otra manera de comparar el desempeño es mostrada en la tabla 3, donde se muestran los resultados de un grupo de consultas seleccionadas de tal manera, que en cada una de ella se aumenta la cantidad de dimensiones involucradas. La finalidad es contrastar los resultados de estas consultas para evidenciar el esfuerzo de los *DBMS* en los procesos de ejecución de consultas que implican *join*. Los tiempos de respuesta en esta tabla se obtuvieron con 12 millones de registros.

Se usa la nueva consulta Q13 con cinco dimensiones involucradas, en lugar de la Q11 o Q12 que se tenía diseñadas. Esto se hace para dar continuidad a la tendencia de aumentar la complejidad añadiendo columnas de otras tablas, pero sin filtrar más información.

Consulta Q13
Resultado: Consumo de minutos mensuales de los usuarios que tienen plan de voz.
Observaciones: Se totaliza (suma) por cada línea telefónica las llamadas realizadas a nivel de mes y se incluyen las dimensiones plan_voz y plan_datos. Solo se aplica el filtro de plan de voz como se hace en la consulta Q8 y se añaden las columnas de los nombres de los planes de datos y voz .
Dimensiones Involucradas: 5

```
select sim_card.num_telefono, plan_datos.nombre, plan_voz.nombre_plan,
cliente.genero, dia.ano, dia.mes, dia.nombre_mes, sum(consumo.minutos) as minutos
from dwh_colmovil.consumo, dwh_colmovil.cliente, dwh_colmovil.dia,
dwh_colmovil.sim_card, dwh_colmovil.plan_voz, dwh_colmovil.plan_datos
where consumo.cliente_key=cliente.cliente_key and consumo.dia_key=dia.dia_key
and sim_card.id_sim_card=consumo.sim_card_key
and plan_voz.plan_voz_key=consumo.plan_voz_key
and  plan_voz.nombre_plan<>'Sin plan de voz'
and plan_datos.plan_datos_key=consumo.plan_datos_key
group by sim_card.num_telefono, plan_datos.nombre, plan_voz.nombre_plan, genero,
ano, mes, nombre_mes;
```

Consulta	# Dimensiones Involucradas	MonetDB Tiempo (ms)	PostgreSQL Tiempo (ms)
Q1	1	344	8533
Q6	2	1089	19538
Q7	3	1533	17901
Q8	4	2193	256870
Q13	5	2354	470945

Tabla 3: Tiempos de respuesta de los *DBMS* para las consultas seleccionadas, según la cantidad de dimensiones involucradas

Las figuras 35 y 36 muestran estos resultados. Debido a las diferencias en magnitud de los tiempos de respuesta, no es fácil ver en detalle el comportamiento (note la diferencia de escala en el eje de tiempo de respuesta, para ambas figuras).

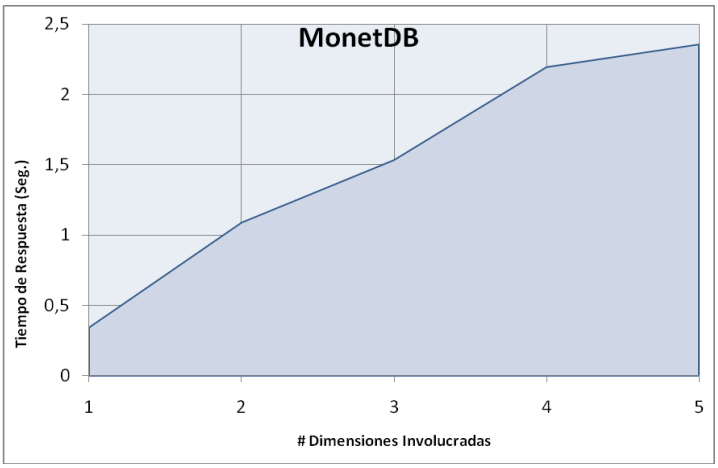


Figura 35: Tiempos de respuesta en *MonetDB* para las consultas seleccionadas

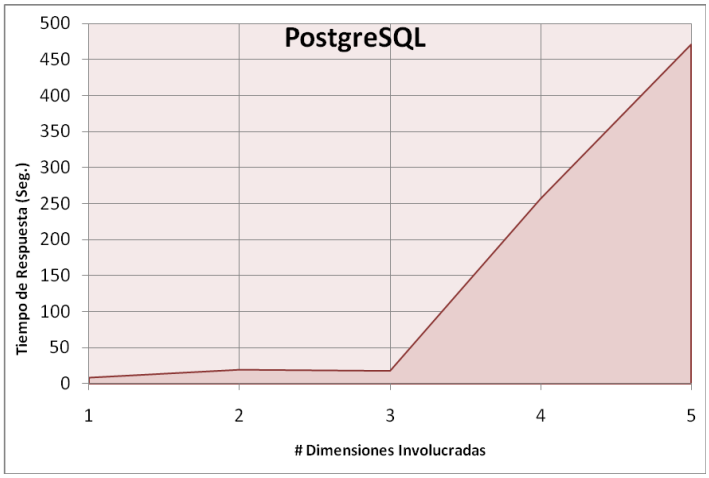


Figura 36: Tiempos de respuesta en *PostgreSQL* para las consultas seleccionadas

A pesar de de la gran diferencia de los tiempos de respuesta entre *MonetDB* y *PostgreSQL*, en las figuras 35 y 36, se puede observar que a medida que se accedan más tablas, se aumenta los tiempos de respuesta. Se observa también en la figura 35 (*MonetDB*), que a partir de la consulta con 2 dimensiones involucradas, es notoria la reducción de rendimiento, a diferencia de la figura 36 (*PostgreSQL*), que parece que aumentar a 2 y 3 dimensiones no altera mucho su rendimiento. Esta reducción se debe a que a partir de la consulta 2, no solo se acceden más tablas, sino que también se acceden más atributos dentro de una misma tabla, evidenciando el costo adicional de unir también los atributos de una misma tabla que *MonetDB* internamente ha separado por columnas.

Para confirmar que el hecho que al acceder más atributos de una misma tabla afecta más el desempeño de *MonetDB* que de *PostgreSQL*, se plantea la consulta Q14 que se muestra a continuación, la cual hace la reconstrucción de tupla con todos sus atributos, para los hechos registrados en el esquema dimensional del *Data Mart* "consumo".

Consulta Q14
Resultado: Reconstrucción de tupla
Observaciones: Se acceden los diferentes atributos que componen la tabla de hechos y las dimensiones.
Dimensiones Involucradas: 8

```
select dia.dia_key, hora.hora_key, operador_destino.operador_destino_key,
cliente.cliente_key, operador_roaming.operador_roaming_key, sim_card.sim_card_key,
plan_datos.plan_datos_key, plan_voz.plan_voz_key, id_cliente, tipo_documento,
num_documento, cliente.nombre, apellido, direccion, estrato, email,
fecha_nacimiento, genero, estado_civil, dia, nombre_dia, festivo,
dia.nombre_festivo, puente, mes, nombre_mes, ano, hora, jornada, id_operador,
operador_destino.nombre_operador, indicativos, operador_destino.pais,
id_operador_roaming, operador_roaming.nombre_operador, operador_roaming.pais,
id_sim_card, num_serie, num_telefono, pin, puk, capacidad_mem_kb, id_plan_datos,
plan_datos.nombre, tipo_tarifa, valor_tarifa, valor_kb_adicional,
```

```
id_plan_voz, nombre_plan, total_minutos, costo_otros_operadores,
costo_minuto_adicional, sms_incluidos, precio
from dwh_colmovil.consumo, dwh_colmovil.cliente,  dwh_colmovil.dia,
dwh_colmovil.hora,  dwh_colmovil.operador_destino,  dwh_colmovil.operador_roaming,
dwh_colmovil.sim_card,  dwh_colmovil.plan_datos,  dwh_colmovil.plan_voz
where cliente.cliente_key=consumo.cliente_key and dia.dia_key=consumo.dia_key and
hora.hora_key=consumo.hora_key and
operador_destino.operador_destino_key=consumo.operador_destino_key and
operador_roaming.operador_roaming_key=consumo.operador_roaming_key and
sim_card.sim_card_key=consumo.sim_card_key and
plan_datos.plan_datos_key=consumo.plan_datos_key and
plan_voz.plan_voz_key=consumo.plan_voz_key
limit 10000;
```

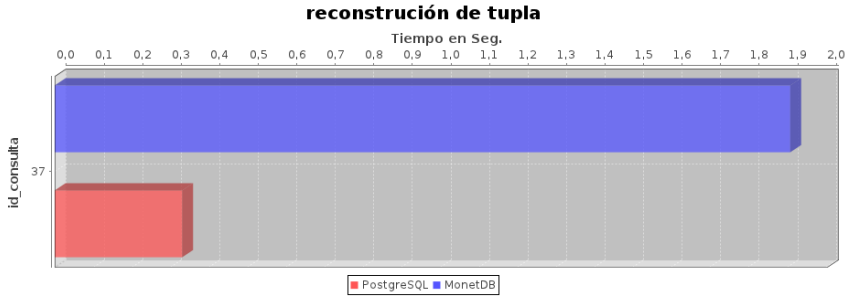


Figura 37: Gráfico de los tiempos de respuesta de *MonetDB* y *PostgreSQL*, para la consulta Q14.

Con los resultados mostrados en la figura 37, se evidencia el hecho que *MonetDB* disminuye su rendimiento al tener que hacer *join* por cada columna que haga parte del conjunto de salida de la consulta, hecho que en *PostgreSQL* no es necesario. En *PostgreSQL* cuando se accede a una columna en una tabla, también se accede todo el conjunto de columnas. Por lo tanto, se puede decir, que evita el proceso de unión de las demás columnas de la misma tabla.

La interpretación que se le puede dar a este comportamiento, desde el enfoque de interés de este trabajo investigativo, es que este tipo de consultas no son de mucha utilidad en el procesamiento analítico de la información, ya que no aportan datos relevantes en la búsqueda de información estratégica. Entonces no es de esperarse que el *DBMS* este constantemente resolviendo consultas como esta.

5 PROTOTIPO DE SOFTWARE PARA LA COMPARACIÓN

Para realizar las pruebas en ambos *DBMS* y llevar registro de los resultados, fue necesario construir un prototipo de aplicación que de soporte a esta tarea. El prototipo de aplicación, provee una interfaz que permite crear consultas, probarlas en ambos *DBMS*, obtener los tiempos de respuesta. cargar la información proveniente de la herramienta de software de monitoreo del uso de recursos, visualizar y almacenar los resultados para facilita su La figura 38, muestra una vista de la arquitectura del prototipo.

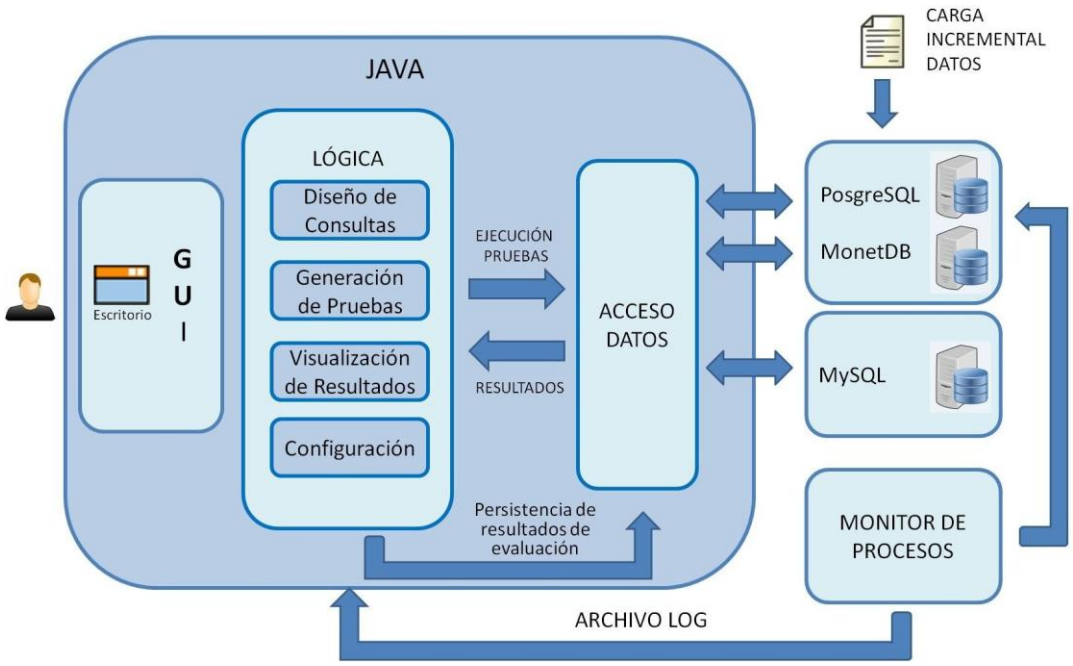


Figura 38: Arquitectura del prototipo

Se proponen cuatro módulos para las funcionalidades del prototipo. En el primero, denominado "Diseño Consultas", se gestionan (crear, modificar y borrar) las consultas, con la ayuda de opciones gráficas, que generan una versión *SQL* para la consulta, de manera automática. En el segundo, denominado "Generar Pruebas", se ejecutan y guardan los registros de las pruebas de rendimiento de cada consulta previamente diseñada. El tercer módulo, denominado "Resultados", provee la interfaz para la visualización de los resultados obtenidos en ambos *DBMS*. Para finalizar, el módulo de "Configuración", permite ajustar los parámetros de las conexiones a bases de datos, definir las tablas que conforman el *Data Mart* y generar los conjuntos de datos sintéticos para la carga incremental. (Para mayor detalle, ver anexo 6 definición de requerimientos para el prototipo de software de pruebas).

5.1 Diseño del prototipo

Para dar persistencia de las consultas diseñadas y a los resultados de las pruebas ejecutadas. Se definió la base de datos mostrada en la figura 39. El esquema se implementó en *MySQL*, para no asignar tareas ajenas a la ejecución de las pruebas de desempeño, en alguno de los *DBMS* de prueba (*MonetDB* y *PostgreSQL*).

Figura 39: Diagrama físico de datos para la persistencia de las pruebas

El prototipo de software, en su diseño, se distribuye en tres capas (presentación, lógica y acceso a datos), mostradas como paquetes de clases a continuación en la figura 40:

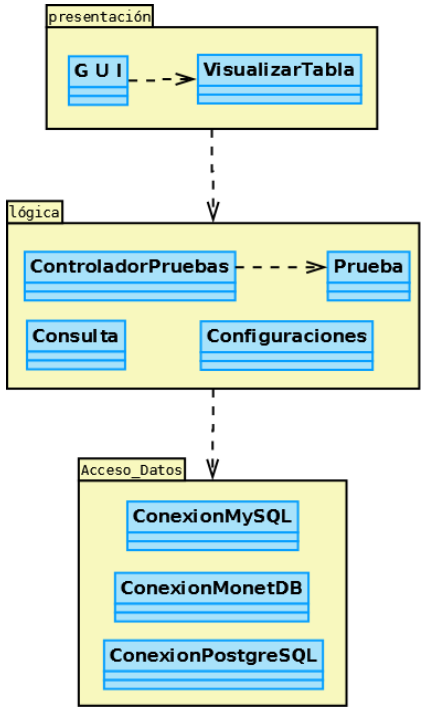


Figura 40: Diagrama de paquetes del prototipo

Internamente el prototipo de herramienta de software para las pruebas de desempeño. implementa el diagrama de clases mostrado en la figura 41. En este se muestra la relación y el uso de las clases involucradas en el sistema.

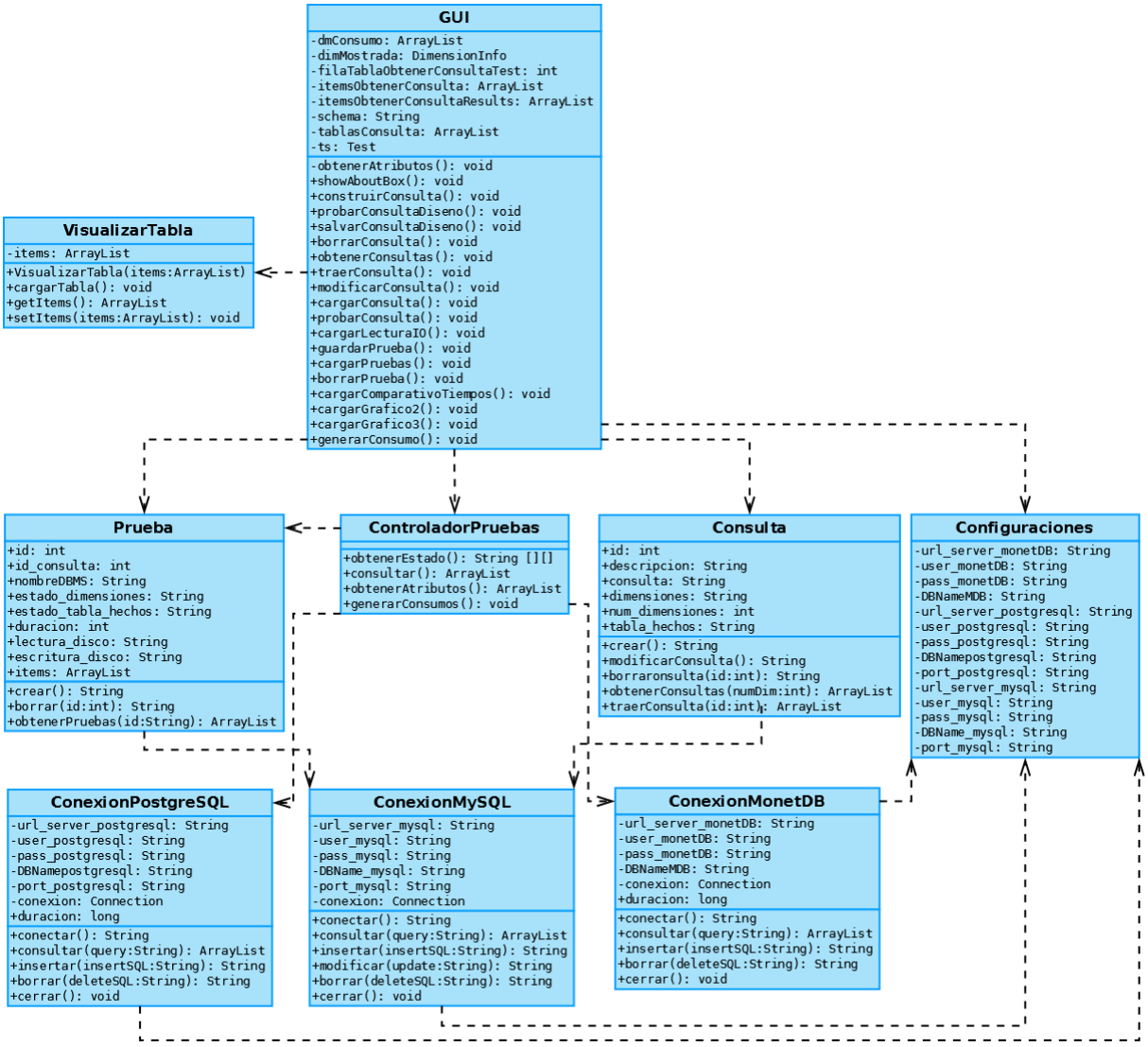


Figura 41: Diagrama de clases del prototipo

5.1.2 Modulo "Diseño Consultas"

Este modulo facilita la exploración y creación de las consultas, a través de un interfaz gráfica (ver figura 42) para la generación automática de consultas en *SQL*, permitiendo:

- Exploración a través de los atributos de las dimensiones y la tabla de hechos del *Data Mart*, con la posibilidad de seleccionar cuáles de ellos se requieren en la salida de la consulta. Para cada atributo se puede: introducir un alias que cambia el nombre del

atributo en la salida de la consulta, elegir la posición que tendrá en el conjunto de columnas de salida, elegir la función de agregación, escoger el tipo de ordenamiento y si se quiere que se le haga agrupamiento.

- Construcción automatizada del código *SQL* para la consulta.
- Prueba de la consulta, para verificar su correcta creación.
- Almacenamiento de la consulta diseñada.

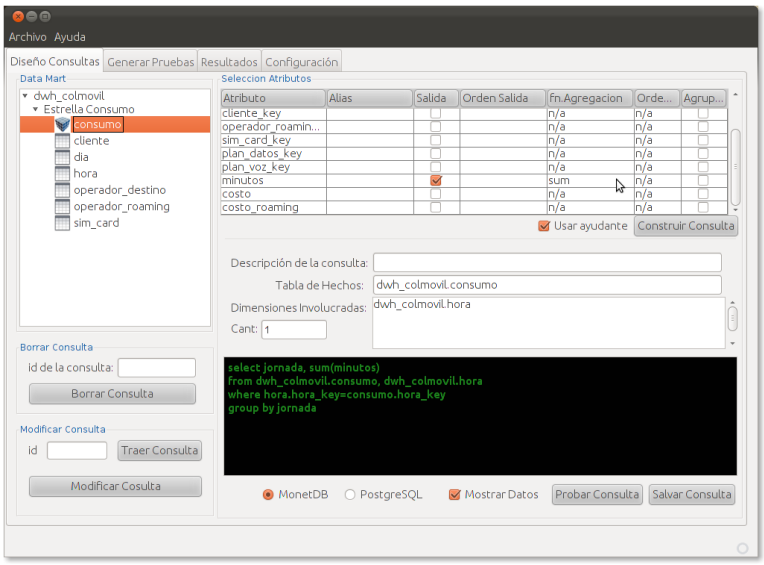


Figura 42: Interfaz gráfica del modulo, "Diseño Consultas".

5.1.3 Modulo "Generar Pruebas"

Es el modulo encargado de la ejecución de las pruebas de desempeño para cada consulta creada (ver figura 43). Con la ayuda de este modulo se puede ejecutar cada consulta y obtener evidencias comparativas. Las tareas que permite son:

- Hacer la consulta en ambos *DBMS*, registrando su tiempo de respuesta y el estado de las dimensiones involucradas.
- Cargar los datos resultantes de la tarea de monitoreo de los recursos del sistema usado por cada *DBMS*.
- Almacena los resultados de la prueba para su posterior análisis.

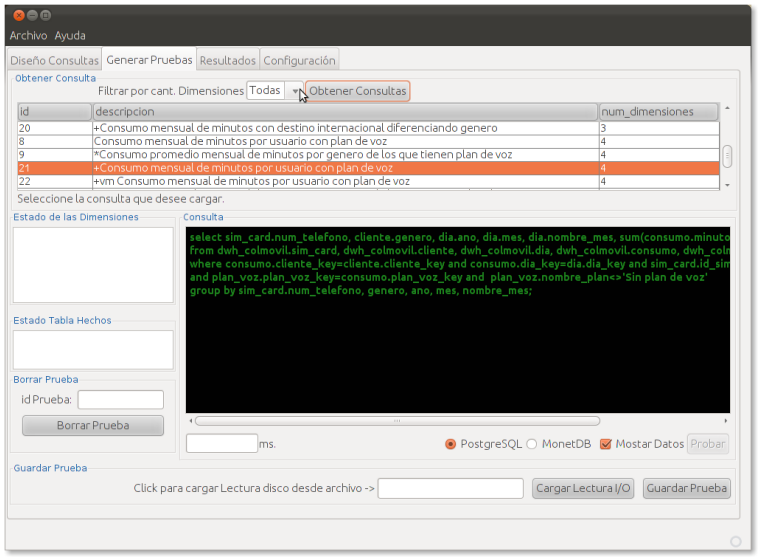


Figura 43: Interfaz gráfica del modulo, "Generar Pruebas"

5.1.4 Modulo "Resultados"

Con este modulo (ver figura 44), se puede visualizar los resultados de las consultas ejecutadas en los diferentes estados del *Data Mart*. Las visualizaciones ofrecidas son:

- Gráfico de barras para comparar los tiempos de respuesta de cada consulta.
- Gráfico para el uso de los recursos durante la consulta.
- Gráfico para ver la evolución del desempeño. (cantidad de filas afectadas vs tiempo de respuesta, para las pruebas hechas con una misma consulta).

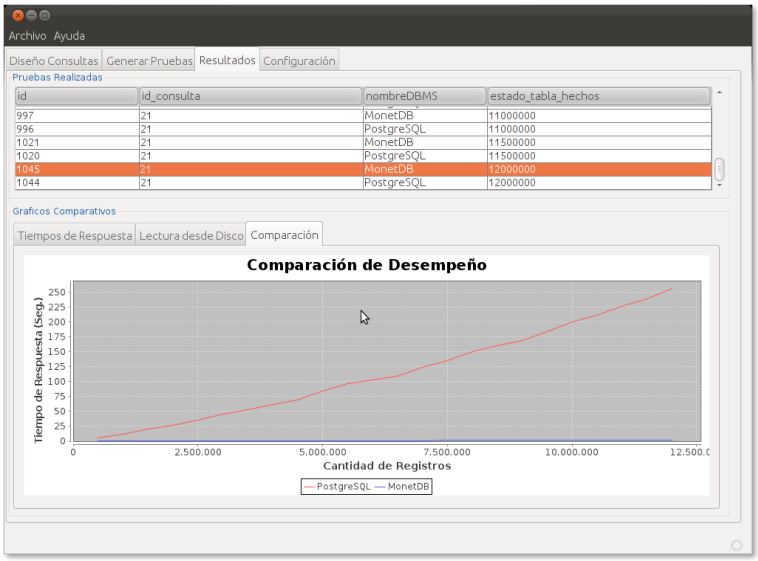


Figura 44: Interfaz gráfica del modulo, "Resultados"

5.1.5 Modulo "Configuración"

Ofrece la posibilidad de editar los parámetros de configuración, para las siguientes opciones:

- Parámetros de las conexión a los *DBMS*.
- Definición de las estrellas, que se pueden consultar.
- Generación de datos sintéticos.

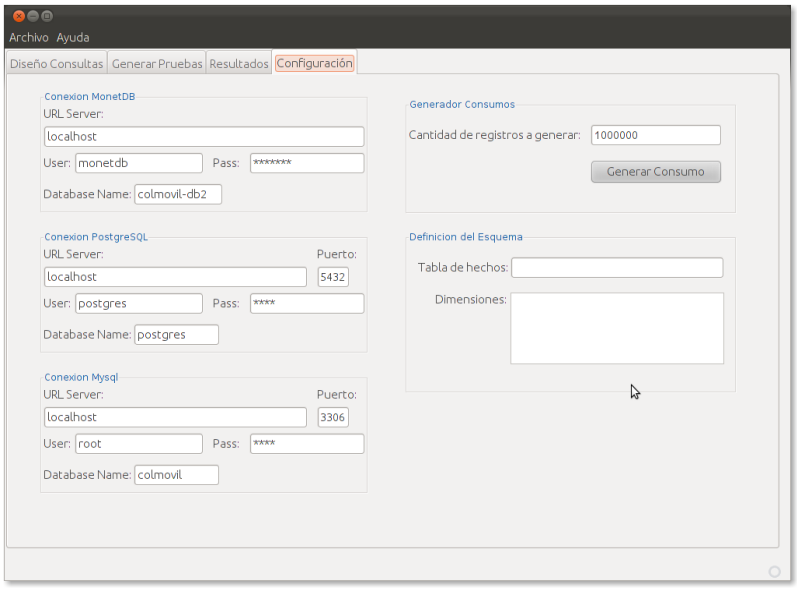


Figura 45: Interfaz gráfica del modulo, "Configuración"

6 CONCLUSIONES

Las pruebas de desempeño se hicieron sobre los *DBMS*, *PostgreSQL* y *MonetDB*, cada uno con un enfoque de diseño diferente. El primero, tradicionalmente orientado a filas y el segundo y poco usado, orientado a columnas en un esquema de datos diseñado para dar soporte a actividades típicas del procesamiento de datos analítico. Los resultados obtenidos muestran tiempos de respuesta bastante ventajosos para el *DBMS* orientado a columnas, evidenciando que esta es una alternativa que debe tenerse en cuenta en la implementación de soluciones que impliquen cargas de trabajo como las de los *DWH* .

Aunque en la mayoría de las consultas diseñadas en este trabajo investigativo, se evidenció un desempeño superior por parte de *MonetDB*, también se experimentó en algunos casos la disminución en su desempeño. En particular esto se dio cuando se incrementaban los atributos requeridos en la salida de la consulta, especialmente en los intentos por reconstruir las tuplas de todo el esquema y en ausencia de funciones de agregación que exijan procesamiento adicional de los datos. Sin embargo, esta situación de tener consultas que intenten extraer la información tal y como fue insertada, desde el punto de vista de entornos de trabajo como el evaluado (*DWH*), es poco probable que se presente. Lo anterior ratifica el uso de los *DBMS* orientados a columnas para fines analíticos y no operacionales, donde estas consultas son frecuentes.

El pobre desempeño obtenido en *PostgreSQL*, también es debido a que su planeador de consultas, en presencia de tablas tan grandes detecta que no se pueden alojar completamente en memoria principal para su procesamiento. Por esto, inicia la construcción del conjunto de atributos necesarios en la salida de la consulta, con el uso del espacio de intercambio ó memoria virtual. Esto implica recurrir a almacenamiento en disco, con costosos tiempos de acceso. En el caso de *MonetDB*, como los datos están particionados a nivel de columnas, el planeador de consultas obtiene solo las columnas necesarias y las procesa por separado, por lo que es posible ir construyendo el conjunto de atributos de salida evitando el uso del espacio de intercambio.

7 TRABAJO FUTURO

Algunas líneas de trabajo futuro, surgidas a partir de este trabajo de evaluación de desempeño, usando *DBMS* con diseños enfocados a columnas y los orientados a filas, se describen a continuación. Estos aspectos se podrían tener en cuenta para complementar la experiencia y confirmar los posibles beneficios que se pueden obtener con la implementación de las bases de datos en tareas que soportan la extracción y análisis de información en *DBMS* orientados a columnas en lugar de los tradicionales orientados a filas.

Los aspectos a considerar son:

- Analizar el desempeño de los *DBMS* ante múltiples consultas a la vez, simulando entornos concurrentes.
- Aplicar los procedimientos de pruebas en equipos servidores con características de hardware que permitan mayores volúmenes de datos.

8 REFERENCIAS

[1] D. J. Abadi, D. S. Myers, D. j. DeWitt and S. R. Madden, "Materialization Strategies in a Column-Oriented DBMS", Proceedings of ICDE, April, 2007, Istanbul, Turkey.

[2] D. J. Abadi, S. R. Madden, and N. Hachem, “Column-Stores vs. Row-Stores: How Different Are They Really?”, SIGMOD 2008, Junio 9–12, 2008, Vancouver, BC, Canada.

[3] D. J. Abadi, S. R. Madden and M. C. Ferreira, "Integrating Compression and Execution in Column-Oriented Database Systems", SIGMOD 2006, Junio 27–29, 2006, Chicago, Illinois, USA

[4] D. J. Abadi, “Query Execution in Column-Oriented Database Systems”, Ph.D. dissertation, Dept. Elect .Eng. Massachusetts Institute of Technology”, Massachusetts, USA, Febrero de 2008.

[5] P. A. Boncz, Martin L. Kersten, and Stefan Manegold, "Breaking the Memory Wall in MonetDB", Communications of the ACM, vol. 51, pp. 77-85, Diciembre de 2008.

[6] N. Bruno “Teaching an Old Elephant New Tricks”, 4th Biennial Conference on Innovative Data Systems Research (CIDR), Enero 4-7 de 2009, Asilomar, California, USA.

[7] D. J. DeWitt, From 1 to 1000 mips, PASS Summit 2009 Keynote. Available: http://pages.cs.wisc.edu/~dewitt/includes/passtalks/passtalks.html

[8] P. Ponniah, Data Warehousing Fundamentals: A Comprehensive Guide for IT Professionals. New York / Chichester / Weinheim / Brisbane / Singapore / Toronto: JOHN WILEY & SONS, INC, 2001, ISBN 0-471-22162-7 pp.1.

[9] P. Ponniah, Data Warehousing Fundamentals: A Comprehensive Guide for IT Professionals. New York / Chichester / Weinheim / Brisbane / Singapore / Toronto: JOHN WILEY & SONS, INC, 2001, ISBN 0-471-22162-7 pp.2.

[10] A. Silberschatz, H. Korth, S. Sudarshan, Fundamentos de Bases de Datos, 4th ed. Madrid: McGraw-Hill, 2002, ISBN 0-07-228363-7 pp. 221.

[11] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O’Neil, P. O’Neil, A. Rasin, N. Tran, and S. Zdonik, " C-Store: A Column-oriented DBMS", Proceedings of the 31st VLDB Conference, 2005, Trondheim, Norway.

[12] Datos sintéticos para proyecto de curso (2010), Descubrimiento del conocimiento en bases de datos - Univalle, Available: http://eisc.univalle.edu.co/cursos/web/ver/750061M/1

[13] Generador de Datos Sintéticos (2011, Noviembre), Advance Data Generator v2.0, Available: http://www.upscene.com/products.adg.moreinfo.php.

[14] Generador de Datos Sintéticos (2011, Noviembre), Generatedata v2.1, Available: http://www.generatedata.com/

[15] *MonetDB* (2011, Abril), Definición del sistema de base de datos *MonetDB*, Available: http://monetdb.cwi.nl/Home/index.html

[16] Motor de procesamiento numérico (2012, Mayo), Wolfram|Alpha, Available: http://www.wolframalpha.com/

[17] PostgreSQL (2012, Marzo), Acerca del sistema de base de datos PostgreSQL, Available: http://www.postgresql.org/about/awards/

[18] PostgreSQL (2012 Marzo), Historia de PostgreSQL, Available: http://www.postgresql.org/about/history/

[19] PostgreSQL (2012 Mayo), Multi-procesos y multi-hilos en PostgreSQL, Available: http://archives.postgresql.org/pgsql-es-ayuda/2006-07/msg00573.php

9 ANEXOS

ANEXO 1: Código *SQL* de implementación de la base de datos operacional

```
create schema colmovil;

CREATE  TABLE IF NOT EXISTS colmovil.cliente (
  idcliente BIGINT NOT NULL ,
  tipo_identificacion VARCHAR(45) NOT NULL ,
  numero_identificacion INTEGER NOT NULL ,
  nombre VARCHAR(45) NULL ,
  apellido VARCHAR(45) NULL ,
  direccion_residencia VARCHAR(200) NULL ,
  estrato INTEGER  NOT NULL ,
  email VARCHAR(45) NULL ,
  fecha_nacimiento DATE NOT NULL ,
  genero VARCHAR(45) NOT NULL ,
  estado_civil VARCHAR(45) NULL ,
  PRIMARY KEY (idcliente) );

CREATE  TABLE IF NOT EXISTS
colmovil.localizacion (
  idlocalizacion BIGINT NOT NULL ,
  ciudad VARCHAR(45) NOT NULL ,
  departamento VARCHAR(45) NOT NULL ,
  PRIMARY KEY (idlocalizacion) );

CREATE  TABLE IF NOT EXISTS colmovil.oficina (
  id_oficina BIGINT NOT NULL ,
  id_localizacion BIGINT NOT NULL ,
  direccion VARCHAR(80) NULL ,
  numero_empleados INTEGER NULL ,
  PRIMARY KEY (id_oficina),
  CONSTRAINT fk_oficina_1
    FOREIGN KEY (id_localizacion )
    REFERENCES colmovil.localizacion
(idlocalizacion ));

CREATE  TABLE IF NOT EXISTS
colmovil.operador_roaming (
  id_operador_roaming BIGINT NOT NULL ,
  nombre_operador VARCHAR(45) NULL ,
  pais_donde_opera VARCHAR(45) NOT NULL ,
  tarifa_voz_por_minuto FLOAT NULL ,
  tarifa_datos_por_kilobyte FLOAT NULL ,
  PRIMARY KEY (id_operador_roaming) );

CREATE  TABLE IF NOT EXISTS
colmovil.equipo_celular (
  id_equipo BIGINT NOT NULL ,
  marca VARCHAR(45) NOT NULL ,
  modelo VARCHAR(45) NOT NULL ,
  precio INTEGER NULL ,
  sistema_operativo VARCHAR(45) NULL ,
  pantalla VARCHAR(45) NULL ,
  procesador VARCHAR(45) NULL ,
CREATE  TABLE IF NOT EXISTS colmovil.contrato
(
  id_contrato SERIAL,
  id_sim_card BIGINT NOT NULL ,
  tipo_plan VARCHAR(45) NULL ,
  id_cliente BIGINT NOT NULL ,
  id_oficina BIGINT NOT NULL ,
  id_plan_voz BIGINT NOT NULL ,
  id_plan_datos BIGINT NOT NULL ,
  id_equipo_celular BIGINT  NULL ,
  fecha_contrato DATE NOT NULL ,
  PRIMARY KEY (id_contrato),
    CONSTRAINT fk_contrato_1
    FOREIGN KEY (id_sim_card )
    REFERENCES colmovil.sim_card (id_sim_card
)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_contrato_2
    FOREIGN KEY (id_cliente )
    REFERENCES colmovil.cliente (idcliente )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_contrato_3
    FOREIGN KEY (id_oficina )
    REFERENCES colmovil.oficina (id_oficina )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_contrato_4
    FOREIGN KEY (id_plan_voz )
    REFERENCES colmovil.plan_voz (id_plan_voz
)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_contrato_5
    FOREIGN KEY (id_plan_datos )
    REFERENCES colmovil.plan_datos
(id_plan_datos )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_contrato_6
    FOREIGN KEY (id_equipo_celular )
    REFERENCES colmovil.equipo_celular
(id_equipo )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);
CREATE  TABLE IF NOT EXISTS colmovil.operador
(
  id_operador BIGINT NOT NULL ,
  nombre VARCHAR(45) NULL ,
  indicativos VARCHAR(45) NULL ,
  PRIMARY KEY (id_operador));
```

```
  memoria VARCHAR(45) NULL ,
  almacenamiento_interno VARCHAR(45) NULL ,
  RED2G VARCHAR(100) NULL ,
  RED3G VARCHAR(100) NULL ,
  conectividad VARCHAR(100) NULL ,
  camara VARCHAR(100) NULL ,
  GPS VARCHAR(45) NULL ,
  radio VARCHAR(45) NULL ,
  bateria VARCHAR(100) NULL,
  PRIMARY KEY (id_equipo) );

CREATE  TABLE IF NOT EXISTS colmovil.plan_voz(
  id_plan_voz BIGINT NOT NULL ,
  nombre VARCHAR(45) NULL ,
  total_minutos INTEGER  NULL ,
  costo_otros_operadores FLOAT NULL ,
  costo_minuto_adicional FLOAT NULL ,
  sms_incluidos INTEGER NULL ,
  precio INTEGER NULL ,
  PRIMARY KEY (id_plan_voz));

CREATE  TABLE IF NOT EXISTS
colmovil.plan_datos (
  id_plan_datos BIGINT NOT NULL ,
  nombre VARCHAR(45) NULL ,
  tipo_tarifa VARCHAR(45) NULL ,
  valor_tarifa FLOAT NULL ,
  valor_kb_adicional INTEGER  NULL ,
  PRIMARY KEY (id_plan_datos));

CREATE  TABLE IF NOT EXISTS colmovil.sim_card
(
  id_sim_card SERIAL,
  numero_serie VARCHAR(20) NULL,
  numero_telefono VARCHAR(45) NOT NULL,
  pin VARCHAR(4)  NULL,
  puk VARCHAR(8)  NULL,
  capacidad_memoria_kb INTEGER NULL,
  PRIMARY KEY (id_sim_card));
```

```
CREATE  TABLE IF NOT EXISTS colmovil.llamada (
  fecha_inicio TIMESTAMP NULL ,
  fecha_finalizacion TIMESTAMP NULL ,
  id_contrato BIGINT NOT NULL,
  id_operador_destino BIGINT NOT NULL ,
  numero_destino VARCHAR(45) NOT NULL ,
  pais_destino VARCHAR(45) NOT NULL ,
  utilizo_roaming VARCHAR(45)  NULL ,
  operador_roaming BIGINT NOT NULL,
  CONSTRAINT fk_llamada_1
    FOREIGN KEY (id_contrato)
    REFERENCES colmovil.contrato (id_contrato)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_llamada_2
    FOREIGN KEY (id_operador_destino )
    REFERENCES colmovil.operador (id_operador
)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_llamada_3
    FOREIGN KEY (operador_roaming )
    REFERENCES colmovil.operador_roaming
(id_operador_roaming )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);

CREATE  TABLE IF NOT EXISTS colmovil.retiro (
  id_contrato BIGINT NOT NULL ,
  fecha DATE NULL ,
  causa VARCHAR(45) NULL ,
  CONSTRAINT fk_retiro_1
    FOREIGN KEY (id_contrato)
    REFERENCES colmovil.contrato (id_contrato)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);

CREATE TABLE IF NOT EXISTS colmovil.recarga (
  id_recarga SERIAL,
  id_contrato BIGINT NOT NULL ,
  fecha_recarga DATE NOT NULL,
  valor_recarga INTEGER DEFAULT NULL,
  medio_recarga VARCHAR(45) NOT NULL,
  PRIMARY KEY (id_recarga),
  CONSTRAINT fk_recarga_1
    FOREIGN KEY (id_contrato)
    REFERENCES  colmovil.contrato
(id_contrato)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);
```

ANEXO 2: Descripción del modelo dimensional

Detalles dimensión DÍA

Figura 46: Dimensión Día

Nombre Atributo	Descripción del Atributo	Valores de ejemplo
DIA_KEY	Llave numérica ficticia de la tabla	1,2,3...
DIA	Numero de día	1,2,3...31
NOMBRE_DIA	Nombre del día	Lunes, Martes, Jueves, etc.
FESTIVO	Bandera que indica si el día es festivo	True, false
NOMBRE_FESTIVO	Nombre del la fiesta que se celebra	Día de la Independencia, Viernes Santo
PUENTE	Bandera que indica si el día cae en puente.	True, false
MES	Numero de mes	1,2,3...12
NOMBRE_MES	Nombre del mes	Enero, Marzo, Diciembre, etc.
ANO	Numero de año	2001,2002,2003...

Tabla 4: Descripción de atributos de la dimensión DIA

Detalles dimensión HORA

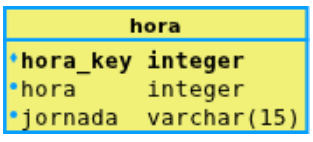


Figura 47: Dimensión Hora

Nombre Atributo	Descripción del Atributo	Valores de ejemplo
HORA_KEY	Llave numérica ficticia de la tabla	1,2,3...
HORA	La hora representada en dígitos del 0 al 23	0,1,2,3...
JORNADA	Jornada es un periodo del día que comprende 6 horas	Mañana, tarde, noche, madrugada

Tabla 5: Descripción de atributos de la dimensión HORA

Detalles dimensión OPERADOR_DESTINO

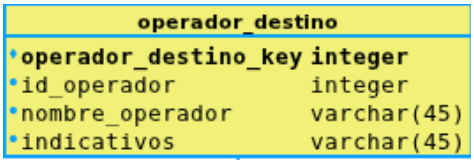


Figura 48: Dimensión operador_destino

Nombre Atributo	Descripción del Atributo	Valores de ejemplo
OPERADOR_DESTINO_KEY	Llave numérica ficticia de la tabla	1,2,3,4…
ID_OPERADOR	Llave traída del modelo operacional	1,2,3,4...
NOMBRE_OPERADOR	Nombre del operador al que el cliente realiza las llamadas	Movistar, Tigo...
INDICATIVOS	Indicativos que usa el operador al asignar sus números telefónicos	310, 311, 312

Tabla 6: Descripción de atributos de la dimensión operador_destino

Detalles Dimensión CLIENTE

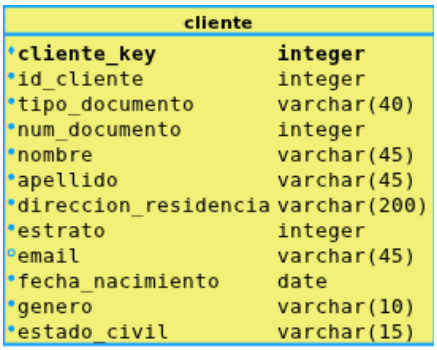


Figura 49: Dimensión cliente

Nombre Atributo	Descripción del Atributo	Valores de ejemplo
CLIENTE_KEY	Llave numérica ficticia de la tabla	1,2,3,4…
ID_CLIENTE	Llave traída del modelo operacional	1,2,3,4...
TIPO_DOCUMENTO	El tipo de documento con que se identifique en el país un usuario.	Cedula de Ciudadanía, Cedula de Extranjería
NUM_DOCUMENTO	Número de identificación	14624999
NOMBRE	Nombre del cliente	Pedro, María, Juan
APELLIDO	Apellido del cliente	Pérez, López, Ríos
DIRECCION_RESIDENCIA	Dirección de residencia del cliente	Calle 100 No 13 - 123
ESTRATO	El estrato social, al que pertenece la vivienda donde reside	1,2,3,4,5,6
EMAIL	Correo electrónico del cliente, para envío de correspondencia o promociones	juan@hotmail.com
FECHA_NACIMIENTO	Fecha nacimiento del cliente	1989-11-20
GENERO	Sexo del cliente	F,M
ESTADO_CIVIL	Estado Civil del cliente	Soltero, Casado, Viudo, etc..

Tabla 7: Descripción de atributos de la dimensión cliente

Detalles Dimensión OPERADOR_ROAMING

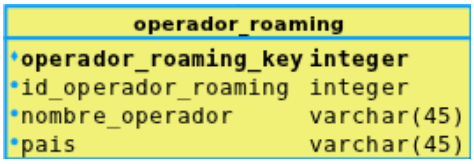


Figura 50: Dimensión operador_roaming

Nombre Atributo	Descripción del Atributo	Valores de ejemplo
OPERADOR_ROAMING_KEY	Llave numérica ficticia de la tabla	1,2,3,4…
ID_OPERADOR_ROAMING	Llave traída del modelo operacional	1,2,3,4...
NOMBRE_OPERADOR	Nombre del operador que presta su red para que clientes de otros operadores utilicen sus líneas	Movistar, Tigo...
PAIS	País donde funciona el operador.	USA, España

Tabla 8: Descripción de atributos de la dimensión operador_roaming

Detalles Dimensión SIM_CARD

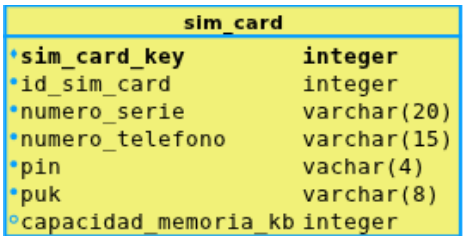


Figura 51: Dimensión sim_card

Nombre Atributo	Descripción del Atributo	Valores de ejemplo
SIM_CARD_KEY	Llave numérica ficticia de la tabla	1,2,3,4…
ID_SIM_CARD	Llave traída del modelo operacional	1,2,3,4...
NUM_SERIE	Es el nuero de serie que identifica la sim_card por parte del fabricante	12343212345678987654
NUM_TELEFONO	El numero asignado que identifica la línea telefónica	3016181145
PIN	El número de identificación personal que permite acceder a los datos guardados en la sim_card	0000
PUK	Clave personal de bloqueo para la sim_card	11224433
CAPACIDAD_MEM_KB	Capacidad en Kb de memoria de la Sim_card utilizada para guardar el directorio telefónico personal.	64

Tabla 9: Descripción de atributos de la dimensión sim_card

Detalles Dimensión PLAN_VOZ

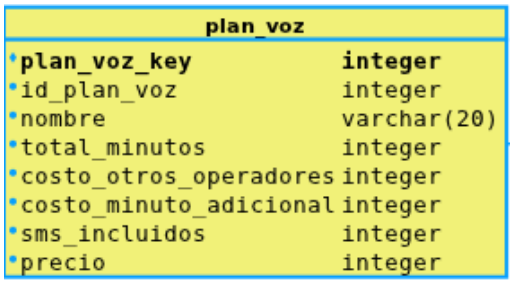


Figura 52: Dimensión plan_voz

Nombre Atributo	Descripción del Atributo	Valores de ejemplo
PLAN_VOZ_KEY	Llave numérica ficticia de la tabla	1,2,3,4…
ID_PLAN_VOZ	Llave traída del modelo operacional	1,2,3,4...
NOMBRE_PLAN	Nombre comercial del plan de voz	Plan Corporativo gold, Plan Estudiantil, etc...
TOTAL_MINUTOS	La cantidad de lo minutos incluidos en el plan de voz	500, 300, 1000, etc..
COSTO_OTROS_OPERADORES	El costo del minuto a un operador distinto a Colmovil	300, 500, 400, etc...
COSTO_MINUTO_ADICIONAL	El costo del minuto cuando exceda el total de minutos.	350, 400, 500, etc...
MSM_INCLUIDOS	Número de mensajes de texto, solo destinos Colmovil.	100, 150,200, etc...
PRECIO	El costo mensual del plan de voz	25000, 50000, 65000, etc...

Tabla 10: Descripción de atributos de la dimensión plan_voz

Detalles Dimensión PLAN_DATOS

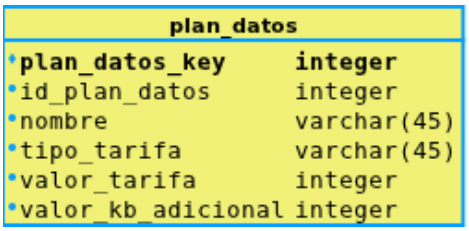


Figura 53: Dimensión plan_datos

Nombre Atributo	Descripción del Atributo	Valores de ejemplo
PLAN_DATOS_KEY	Llave numérica ficticia de la tabla	1,2,3,4…
ID_PLAN_DATOS	Llave traída del modelo operacional	1,2,3,4...
NOMBRE	Nombre comercial del plan de datos	Plan 250MB, Plan 1GB
TIPO_TARIFA	Indica la forma como de cobra el consumo de datos.	Fija, Por Volumen
VALOR_TARIFA	Valor fijo pagado mensual por el servicio o por unidad de consumo Kb.	40000, 5, etc...
VALOR_KB_ADICIONAL	Valor del kb consumido cuando exceda la capacidad fija adquirida.	10, 8, 9, etc...

Tabla 11: Descripción de atributos de la dimensión plan_datos

Detalles del Hecho CONSUMO

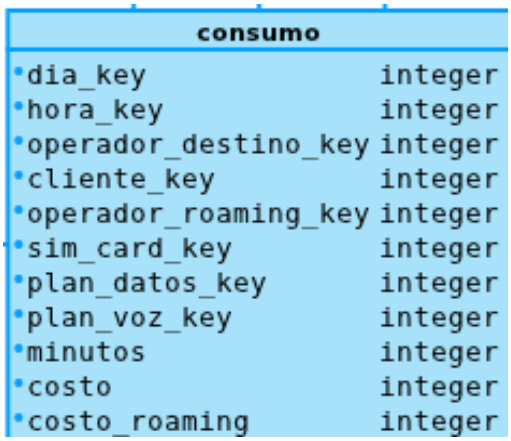


Figura 54: Tabla de hechos Consumo

Nombre Hecho	Descripción Hecho	Regla de agregación por defecto
DIA_KEY	Llave foránea de la tabla día	Count
HORA_KEY	Llave foránea de la tabla hora	Count
OPERADOR_DESTINO_KEY	Llave foránea de la tabla operador_destino	Count
CLIENTE_KEY	Llave foránea de la tabla cliente	Count
OPERADOR_ROAMING_KEY	Llave foránea de la tabla operador_roaming	Count
SIM_CARD_KEY	Llave foránea de la tabla sim_card	Count
MINUTOS	Duración de la llamada en minutos	Sum
COSTO	Costo total de la llamada sin usar roaming	Sum
COSTO_ROAMING	Costo por de la llamada si uso roaming.	Sum

Tabla 12: Descripción tabla de hechos consumo

ANEXO 3: Código *SQL* de implementación del Modelo Dimensional

En *PostgreSQL*	En *MonetDB*
create schema dwh_colmovil; create table dwh_colmovil.operador_destino (operador_destino_key serial, id_operador bigint not null, nombre_operador varchar(45), indicativos varchar(45), pais varchar(45), primary key (operador_destino_key)); create table dwh_colmovil.dia (dia_key serial, dia integer, nombre_dia varchar(9), festivo bool, nombre_festivo varchar(50), puente bool, mes integer, nombre_mes varchar(10), ano integer, primary key (dia_key)); create table dwh_colmovil.hora (hora_key serial, hora integer, jornada varchar(15), primary key (hora_key)); create table dwh_colmovil.operador_roaming (operador_roaming_key serial, id_operador_roaming bigint not null, nombre_operador varchar(45), pais varchar(45), primary key (operador_roaming_key)); create table dwh_colmovil.cliente (cliente_key serial, id_cliente bigint not null, tipo_documento varchar(40), num_documento integer,	create schema dwh_colmovil; create table dwh_colmovil.operador_destino (operador_destino_key integer not null auto_increment, id_operador integer not null, nombre_operador varchar(45), indicativos varchar(45), pais varchar(45), primary key (operador_destino_key)); create table dwh_colmovil.dia (dia_key integer not null auto_increment, dia integer, nombre_dia varchar(9), festivo bool, nombre_festivo varchar(50), puente bool, mes integer, nombre_mes varchar(10), ano integer, primary key (dia_key)); create table dwh_colmovil.hora (hora_key integer not null auto_increment, hora integer, jornada varchar(15), primary key (hora_key)); create table dwh_colmovil.operador_roaming (operador_roaming_key integer not null auto_increment, id_operador_roaming integer not null, nombre_operador varchar(45), pais varchar(45), primary key (operador_roaming_key)); create table dwh_colmovil.cliente (cliente_key integer not null auto_increment, id_cliente integer not null, tipo_documento varchar(40),

```
nombre   varchar(45),
apellido   varchar(45),
direccion   varchar(200),
estrato   integer,
email   varchar(45),
fecha_nacimiento   date,
genero   varchar(10),
estado_civil   varchar(15),
primary key (cliente_key)
);

create table dwh_colmovil.sim_card
(
sim_card_key   serial,
id_sim_card   bigint not null,
num_serie   varchar(20) null,
num_telefono   varchar(45) not null,
pin   varchar(4)  null,
puk   varchar(8)  null,
capacidad_mem_kb   integer null,
primary key (sim_card_key)
);

create table dwh_colmovil.plan_datos
(
plan_datos_key   serial,
id_plan_datos   bigint,
nombre   varchar(45),
tipo_tarifa   varchar(45),
valor_tarifa   float,
valor_kb_adicional   integer,
primary key (plan_datos_key)
);

create table dwh_colmovil.plan_voz
(
plan_voz_key   serial,
id_plan_voz   bigint,
nombre_plan   varchar(20),
total_minutos   integer,
costo_otros_operadores   float,
costo_minuto_adicional   float,
sms_incluidos   integer,
precio   integer,
primary key (plan_voz_key)
);

create table dwh_colmovil.consumo
(
dia_key    bigint,
hora_key   bigint,
operador_destino_key   bigint,
cliente_key   bigint,
operador_roaming_key   bigint,
sim_card_key    bigint,
plan_datos_key   bigint,
plan_voz_key   bigint,
minutos   integer,
```

```
num_documento integer,
nombre varchar(45),
apellido varchar(45),
direccion varchar(200),
estrato integer,
email varchar(45),
fecha_nacimiento date,
genero varchar(10),
estado_civil varchar(15),
primary key (cliente_key)
);

create table dwh_colmovil.sim_card
(
sim_card_key integer not null
auto_increment,
id_sim_card integer not null,
num_serie varchar(20) null,
num_telefono varchar(45) not null,
pin varchar(4)  null,
puk varchar(8)  null,
capacidad_mem_kb integer null,
primary key (sim_card_key)
);

create table dwh_colmovil.plan_datos
(
plan_datos_key integer not null
auto_increment,
id_plan_datos integer,
nombre varchar(45),
tipo_tarifa varchar(45),
valor_tarifa float,
valor_kb_adicional integer,
primary key (plan_datos_key)
);

create table dwh_colmovil.plan_voz
(
plan_voz_key integer not null
auto_increment,
id_plan_voz integer,
nombre_plan varchar(20),
total_minutos integer,
costo_otros_operadores float,
costo_minuto_adicional float,
sms_incluidos integer,
precio integer,
primary key (plan_voz_key)
);

create table dwh_colmovil.consumo
(
dia_key              integer,
hora_key             integer,
operador_destino_key integer,
cliente_key          integer,
operador_roaming_key integer,
```

```
costo   integer,
costo_roaming   integer,
constraint fk_consumos_1 foreign key (dia_key)
references dwh_colmovil.dia (dia_key),
constraint fk_consumos_2 foreign key (hora_key)
references dwh_colmovil.hora (hora_key),
constraint fk_consumos_3 foreign key
(operador_destino_key)
references dwh_colmovil.operador_destino
(operador_destino_key),
constraint fk_consumos_4 foreign key
(cliente_key)
references dwh_colmovil.cliente (cliente_key),
constraint fk_consumos_5 foreign key
(operador_roaming_key)
references dwh_colmovil.operador_roaming
(operador_roaming_key),
constraint fk_consumos_6 foreign key
(sim_card_key)
references dwh_colmovil.sim_card (sim_card_key),
constraint fk_consumos_7 foreign key
(plan_datos_key)
references dwh_colmovil.plan_datos
(plan_datos_key),
constraint fk_consumos_8 foreign key
(plan_voz_key)
references dwh_colmovil.plan_voz (plan_voz_key)
);
```

```
sim_card_key         integer,
plan_datos_key       integer,
plan_voz_key         integer,
minutos              integer,
costo                integer,
costo_roaming        integer,
constraint fk_consumos_1 foreign key
(dia_key)
references dwh_colmovil.dia (dia_key),
constraint fk_consumos_2 foreign key
(hora_key)
references dwh_colmovil.hora (hora_key),
constraint fk_consumos_3 foreign key
(operador_destino_key)
references dwh_colmovil.operador_destino
(operador_destino_key),
constraint fk_consumos_4 foreign key
(cliente_key)
references dwh_colmovil.cliente
(cliente_key),
constraint fk_consumos_5 foreign key
(operador_roaming_key)
references dwh_colmovil.operador_roaming
(operador_roaming_key),
constraint fk_consumos_6 foreign key
(sim_card_key)
references dwh_colmovil.sim_card
(sim_card_key),
constraint fk_consumos_7 foreign key
(plan_datos_key)
references dwh_colmovil.plan_datos
(plan_datos_key),
constraint fk_consumos_8 foreign key
(plan_voz_key)
references dwh_colmovil.plan_voz
(plan_voz_key)
);
```

ANEXO 4: Generador de Datos sintéticos

Los datos que se usaron para cargar la tabla de hechos "consumo", fueron generados con una clase *JAVA* con conexión a base de datos, en el proceso de generación, fue necesario que se consultarán las dimensiones de la base de datos, para obtener las llaves reales de las dimensiones y combinarlas de forma semi-aleatoria para evitar infracciones a las restricciones de llaves foráneas en la tabla de hechos. A continuación se documenta el método encargado de generar los datos sintéticos en archivos de texto con valores separados por punto y comas. Se presentan las principales características implementadas en el método que genera los datos sintéticos.

La figura 55 obtiene de base de datos el último mes con consumos generados, para determinar el mes siguiente al que se va a generar los datos.

```
ArrayList a = new ArrayList();
String query = "select ano, mes from dwh_colmovil.dia, dwh_colmovil.consumo"+
               " where dia.dia_key=consumo.dia_key" +
               "order by ano desc, mes desc limit 1;";
a = consultar(query, "PostgreSQL");
if(a.size()>1){
ArrayList fila = (ArrayList)a.get(1);
    if(mes<12){
       mes = (Integer.valueOf((String)fila.get(1)))+1;
    }else{
        mes=1;
        ano++;
    }
}
```

Figura 55: Calculo del siguiente mes

Es importante que los datos generados cumplan las restricciones de llave foránea, por esto se consultan las dimensiones para obtener sus llaves y después combinarlas de manera semi-aleatoria.

```
ArrayList keys_mes = new ArrayList();
query = "select dia_key, dia from dwh_colmovil.dia where ano="+ano+" and mes="+mes;
keys_mes = consultar(query, "PostgreSQL");
ArrayList consumidores = new ArrayList();
query = "select contratos.cliente_key, sim_card_key, plan_datos_key, "+
"contratos.plan_voz_key, colmovil.plan_voz.precio, colmovil.plan_voz.total_minutos,"+
" genero from dwh_colmovil.contratos, dwh_colmovil.plan_voz, colmovil.plan_voz, "+
"dwh_colmovil.cliente where contratos.plan_voz_key=dwh_colmovil.plan_voz.plan_voz_key"+
" and dwh_colmovil.plan_voz.id_plan_voz=colmovil.plan_voz.id_plan_voz and "+
"cliente.cliente_key=contratos.cliente_key";
consumidores = consultar(query, "PostgreSQL");
ArrayList operadores_destino = new ArrayList();
query = "select operador_destino_key from dwh_colmovil.operador_destino";
operadores_destino = consultar(query, "PostgreSQL");
ArrayList operadores_roaming = new ArrayList();
query = "select operador_roaming_key from dwh_colmovil.operador_roaming";
operadores_roaming = consultar(query, "PostgreSQL");
```

Figura 56: Obtener llaves de las dimensiones

Como el método permite generar la cantidad de datos que se desee, en base a este límite se calculan las cantidades a generar por hora.

```
int totalConsumidores = consumidores.size();
int cantidadRegMes=limite; //límite es el parametro de entrada del metodo
int cantDiaria = (int)(cantidadRegMes/keys_mes.size());
int cantHora = (int)(cantDiaria/24);
```

Figura 57: Calculo de la cantidad de consumos a generar por hora

La figura 58 muestra como se usan las funciones aleatorias para obtener usuarios aleatoriamente y para la generación de los minutos y los costos de los mismos consumos.

```
int indice = (int) (Math.random()*totalConsumidores); //indice aleatorio
for(int cant=0; cant<cantHora;cant++){
    ArrayList consumidor = (ArrayList)consumidores.get(indice);//consumidor aleatorio
    String cliente_key = (String)consumidor.get(0);
    String sim_card_key = (String)consumidor.get(1);
    String plan_datos_key = (String)consumidor.get(2);
    String plan_voz_key = (String)consumidor.get(3);
    int precio_plan = Integer.valueOf((String)consumidor.get(4));
    int total_minutos = Integer.valueOf((String)consumidor.get(5));
    String genero = (String)consumidor.get(6);
    int index_op = (int)(Math.random()*(operadores_destino.size()-3));
    int operador_destino_key = Integer.valueOf((String)((ArrayList)operadores_destino.get(index_op)).get(0));
    int mul_min = 25;
    int minutos = (int)(Math.random()*mul_min+1); //minutos consumidos aleatoriamente
    if(genero.equals("Femenino")){
        minutos+=10;
    }
    int r = (int)(Math.random()*100);
    int costo = 0;
    int costo_roaming=0;
    int operador_roaming_key=1;
    if(r>95){ //el 5% de los consumos usará roaming
        int index_op_r = (int)(Math.random()*(operadores_roaming.size()-1)+1);
        int tarifa = (int)(Math.random()*(3000)+1000);
        operador_roaming_key = Integer.valueOf((String)((ArrayList)operadores_roaming.get(index_op_r)).get(0));
        costo_roaming = minutos*tarifa;
    }else{
        if(plan_voz_key.equals("1")){
            costo = minutos*300; //costo del minuto prepago
        }else{
            costo = (int)minutos*(precio_plan/total_minutos); //costo del minuto en plan
        }
    }
```

Figura 58: Uso de función aleatoria en los consumos

La salida del método son dos archivos con los valores separados por punto y coma, cada uno con la mitad de los datos.

```
String url = (new File ("").getAbsolutePath ());
fichero = new FileWriter(url+"/consumos"+ano+mes+"_1.csv");
fichero2 = new FileWriter(url+"/consumos"+ano+mes+"_2.csv");
pw = new PrintWriter(fichero);
pw2 = new PrintWriter(fichero2);
```

Figura 59: Creación de archivos de salida

ANEXO 5: Generación de datos para Dimensiones DIA y HORA.

Generador de datos para cargar la dimensión DIA.

Para generar los datos que se insertarán en la dimensión DIA, es necesario identificar el año, mes, día, día de la semana, el nombre del mes, identificar si es festivo y si hace parte de un puente. La figura 60 muestra el código fuente de como se hizo.

```
while(ano!=2011&&continuar){//se generan los datos hasta llegar al 2011
    ano = cal.get(Calendar.YEAR);//se extrae día, mes y año
    mes = cal.get(Calendar.MONTH);
    dia = cal.get(Calendar.DAY_OF_MONTH);
    int dia_semana = cal.get(Calendar.DAY_OF_WEEK);
    String nombre_dia="";
    String nombre_mes="";
    if(mes==0){nombre_mes = "ENERO";}
    if(mes==1){nombre_mes = "FEBRERO";}
    .
    .//se idenifica el nombre del mes
    .
    if(mes==11){nombre_mes = "DICIEMBRE";}
    if(dia_semana==1){nombre_dia = "DOMINGO";}
    .
    .//se indentifica el nombre del día
    .
    if(dia_semana==7){nombre_dia = "SABADO";}

    boolean festivo=false;
    String nombre_festivo="N/A";
    if(dia==1&&mes==0){festivo=true; nombre_festivo="Ano Nuevo";}
    if(dia==1&&mes==4){festivo=true; nombre_festivo="Dia del Trabajo";}
    ...//con los demas festivos se identifican igualmente
    boolean puente=false;
    if(festivo&&(nombre_dia.equals("LUNES")||nombre_dia.equals("VIERNES"))){
            puente=true; //se veifica si es puente
    }

    String query="";
    query="INSERT INTO dwh_colmovil.dia (dia, nombre_dia, festivo, "+
    "nombre_festivo, puente, mes, nombre_mes, ano ) "+
    "VALUES("+dia+", \'"+nombre_dia+"\', "+festivo+", \'"+nombre_festivo+
    "\', "+puente+", "+(mes+1)+", \'"+nombre_mes+"\', "+ano+")";
    String insert = con.insertar(query);//inserta el código SQL
    continuar=insert.equals("inserted");

    cal.add(Calendar.DATE, 1);//se avanza 1 día
}
```

Figura 60: Generación de registros para la dimensión DIA

Generador de datos para cargar la dimensión HORA.

Igualmente a la dimensión HORA se le generan los 24 registros que contendrá y se clasifica cada hora dentro de una jornada (mañana, tarde, noche y madrugada).

```
public void generar(){
   Calendar cal = Calendar.getInstance();
   cal.set(2000, 00, 00, 00, 00);
   int hora = cal.get(Calendar.HOUR_OF_DAY);
   ConexionPosgreSQL con = new ConexionPosgreSQL();
   String msj = con.conectar();
   if(msj.equals("conectado")){
       String resetSeq =  "ALTER SEQUENCE dwh_colmovil.hora_hora_key_seq RESTART WITH 1";
       con.insertar(resetSeq);//se resetea el auto increment de la tabla
       int count = 0;
       boolean continuar = true;
       while(continuar){
           hora = cal.get(Calendar.HOUR_OF_DAY);//se identifica la hora del día
           String query="";
           String jornada="";
           if(hora>=0&&hora<6){jornada="madrugada";}//se identifica la jornada
           if(hora>=6&&hora<12){ jornada="mañana"; }
           if(hora>=12&&hora<18){jornada="tarde";}
           if(hora>=18&&hora<=23){jornada="noche";}
           query="INSERT INTO dwh_colmovil.hora (hora, jornada) VALUES("+hora+", \'"+jornada+"\')";
           String insert = con.insertar(query);

           count++;
           cal.add(Calendar.HOUR_OF_DAY, 1);//se incrementa 1 hora

           continuar=count<24; //limite de generación
       }
   }else{
   JOptionPane.showMessageDialog(null, msj, "Mensaje  de PostgreSQL", JOptionPane.ERROR_MESSAGE);
   }
   con.cerrar();   //cerrar conexión a DB
}
```

Figura 61: Generación de registros para la dimensión HORA

ANEXO 6: Definición de requerimientos para el prototipo de software para de pruebas.

El propósito de este anexo, es presentar el conjunto de descripciones funcionales que orientan el diseño y desarrollo del prototipo de aplicación, para la realización de pruebas de desempeño en manejadores de bases de datos, necesario para la experimentación y consecución de evidencias que sustenten el presente trabajo investigativo.

Alcance del prototipo

El prototipo de aplicación a diseñar y desarrollar, será usado como herramienta de apoyo, en la experimentación emprendida para la realización del proyecto investigativo, en el se evaluará y comparará el desempeño de los manejadores de bases de datos orientados a filas contra los orientados a columnas. Se espera que el uso del prototipo, facilite la creación de las consultas en el modelo de almacenamiento de datos especificado en el proyecto de investigación. El prototipo se enfoca principalmente en la gestión de consultas de prueba, la ejecución de las pruebas, el registro de sus tiempos de respuesta y los gráficos comparativos para cada manejador evaluado.

Características de los usuarios

El prototipo de software será diseñado para el usuario investigador, interesado en gestionar consultas fácilmente y ejecutarlas en los *DBMS* a evaluar, para luego registrar sus tiempos de respuesta como pruebas de desempeño. No es necesario definir perfiles especializados, por lo cual, toda la funcionalidad estará disponible para su uso.
El usuario podrá realizar las siguientes funciones descritas a continuación:

Actores	Funciones
Usuario	• Diseñar y gestionar consultas • Ejecutar consultas y gestionar sus resultados. • Visualizar gráficos comparativos. • Configurar conexiones a los *DBMS* evaluados. • Generar datos sintéticos para la carga del modelo de datos especificado en el proyecto investigativo.

Restricciones generales

- La construcción de las consultas de prueba con el prototipo, estará ligado al diseño del esquema de la base de datos, la cual debe ser del tipo dimensional o en estrella, como el usado en la definición de los *Data Warehouse*.
- El prototipo debe realizar consultas en los *DBMS PostgreSQL* y *MonetDB*.
- Los archivos log que puede integrar, provienen de la herramienta pidstat, disponible para el monitoreo del uso de recursos en el sistema operativo Linux. El uso de otros archivos deberá coincidir con el formato empleado por pidstat.
- El generador de datos sintético funciona para la tabla de hechos definida en el proyecto investigativo y necesita que las dimensiones del esquema estén pobladas de datos.

Requerimientos Funcionales

A continuación se describen detalladamente cada uno de los requisitos funcionales del Prototipo.

Revisión Histórica

REV	Descripción del cambio	Autor	Fecha
001	Creación del documento	Julio César Tenganán Daza	7/11/11

Modulo Diseño y gestión de consultas

R.1 Modulo Diseño y gestión de consultas	
Función	Diseñar y gestionar consultas.
Descripción	Permitir al usuario de la aplicación diseñar, validar y gestionar consultas, con la ayuda de herramientas visuales, para la elección de los atributos y las funciones de agregación y ordenamiento.
Entradas	Selección de atributos de las tablas que componen el esquema, elección de funciones de agregación y ordenamiento, definición de alias para los atributos.
Fuentes	Teclado y mouse.

Salidas	Confirmación o error en la definición.
Proceso	El sistema despliega un árbol con las tablas que componen el esquema. Cuando el usuario elige una de las tablas, el sistema le presenta los atributos que componen la tabla. El usuario puede elegir cuales de estos atributos estarán en la salida de la consulta con opciones como: definición de alias para cambiar el nombre a mostrar del atributo, su posición en el conjunto de atributos de salida de la consulta, función de agregación, ordenamiento y si debo agrupar. Una vez, el usuario termina la selección y ajuste de los atributos de salida de su consulta. Puede pulsar el botón “Construir consulta” para que el sistema le construya una propuesta de consulta, la cual puede ser editada si se requiere. Adicionalmente el sistema le reconoce y le presenta cantidad de dimensiones involucradas en la consulta con sus nombres y cual es la tabla de hechos. El usuario debe proporcionar una descripción para la consulta. El usuario valida la consulta ejecutándola en cada *DBMS* con el botón “Probar Consulta”. Si la consulta esta correcta, puede guardar la consulta con el botón “Salvar Consulta”. También el usuario puede traer sus consultas previamente guardadas con el botón “Traer Consulta” para editarla y después guardarla con el botón “Modificar Consulta”. Si el usuario deberá borrar una consulta, debe digitar el id correspondiente y pulsar el botón “Borrar Consulta”.
Restricciones	El esquema de la base de datos debe estar creado en ambos *DBMS* para usar la ayuda visual. La consulta debe cumplir con el estándar *SQL*.
Precondiciones	La consulta a modificar o borrar debe existir.
Poscondiciones	La consulta queda guardada, modificada o eliminada en la base de datos del prototipo.
Efectos Colaterales	-
Tipo	Esencial y primario.

Realizado por:	Firma	Fecha
Julio César Tenganán Daza		7/11/2011
Aprobado por:	**Firma**	**Fecha**
Martha Millán		7/11/2011

Modulo de Ejecución de consultas y gestión de resultados.

R.2 Modulo ejecución de consultas y gestión de resultados	
Función	Ejecutar consultas y gestionar resultados.
Descripción	Permitir al usuario de la aplicación ejecutar las consultas diseñadas previamente en cada uno de los *DBMS*, registrando los tiempos de respuesta, importar los archivos log de la herramienta de monitoreo pidstat, guardar los resultados. Una vez guardado el resultado, podrá ser borrado.
Entradas	Selección de la consulta a probar, ruta del archivo log
Fuentes	Teclado y mouse.
Salidas	Confirmación o error en la ejecución
Proceso	El sistema muestra las consultas que han sido creadas. El usuario selección la consulta que desea probar, el *DBMS* y la ejecuta. El sistema envía la consulta al *DBMS* seleccionado, muestra la información consultada, el tiempo que tardo en procesar la consulta, el estado de las tablas involucradas(cantidad de registros que contiene cada tabla) El usuario opcionalmente puede definir la ruta del archivo log donde están los registros del uso de recursos durante la consulta y cargarlos. El sistema hace una lectura del archivo, reconociendo los datos que corresponden al proceso del *DBMS* seleccionado para la prueba. El usuario decide guardar la prueba como evidencia, presionando el botón “Guardar Prueba” Si el usuario decide borrar una consulta previamente guardada, digita el id de la prueba, presiona el botón “Borrar Prueba”. El

	sistema la borra entonces de la base de datos.
Restricciones	El archivo de log donde están la información del uso de recursos durante la consulta, debe estar en el formato que entrega pidstat.
Precondiciones	La consulta a ejecutar debe existir. La prueba a borrar debe existir.
Poscondiciones	La prueba queda guardada o eliminada en la base de datos del prototipo.
Efectos Colaterales	-
Tipo	Esencial y primario.

Realizado por:	**Firma**	**Fecha**
Julio César Tenganán Daza		7/11/2011
Aprobado por:	**Firma**	**Fecha**
Martha Millán		7/11/2011

Modulo de Visualización de gráficos comparativos.

R.3 Modulo visualización de gráficos comparativos	
Función	Mostrar gráficos comparativos a partir de las pruebas realizadas.
Descripción	Permitir al usuario de la aplicación visualizar gráficos construidos con los resultados obtenidos de la ejecución de las consultas.
Entradas	Selección de la consulta a visualizar.
Fuentes	Teclado y mouse.
Salidas	Representación visual de resultados.
Proceso	El sistema muestra las pruebas realizadas. El usuario selecciona una de las pruebas y gráfica los resultados obtenidos de esa prueba, si la prueba contiene datos del uso de recursos en esa consulta los presenta también.

Restricciones	-
Precondiciones	Las pruebas deben existir.
Poscondiciones	-
Efectos Colaterales	-
Tipo	Esencial y primario.

Realizado por:	**Firma**	**Fecha**
Julio César Tenganán Daza		7/11/2011
Aprobado por:	**Firma**	**Fecha**
Martha Millán		7/11/2011

Modulo de configuraciones y generación de datos

R.4 Modulo configuraciones y generación de datos	
Función	Definir los parámetros de configuración de los *DBMS* usados y de la generación de datos sintéticos.
Descripción	Brindar la posibilidad de ajustar los parámetros de conexión a los servidores de bases de datos. Permitir ajustar la cantidad de datos sintéticos y generarlos
Entradas	Url del servidor, puerto, usuario, nombre de la base de datos y password. Cantidad de registros a generar.
Fuentes	Teclado y mouse.
Salidas	Para los datos generados sintéticamente, archivos de texto con los valores separados por punto y coma.

Proceso	El sistema muestra los campo para el ajuste de los parámetros. (Url del servidor, puerto, usuario, nombre de la base de datos y password) El usuario digita estos campos, el sistema mantiene estos datos para usarlos en las conexiones. El usuario digita la cantidad de registros a generar y presiona el botón “Generar”. El sistema genera los datos en un intervalo de tiempo no mayor a un mes. Los datos son generados en dos partes, en archivos de texto con valores separados por punto y comas.
Restricciones	Las dimensiones del esquema de la base de datos deben estar pobladas para poder generar los datos.
Precondiciones	Las pruebas deben existir.
Poscondiciones	Los parámetros configurados se usan en todas las conexiones
Efectos Colaterales	-
Tipo	Esencial y primario.

Realizado por:	**Firma**	**Fecha**
Julio César Tenganán Daza		7/11/2011
Aprobado por:	**Firma**	**Fecha**
Martha Millán		7/11/2011

ANEXO 7: Características de Hardware y Software del equipo de pruebas.

Hardware	
Característica	**Descripción**
Procesador:	*Intel® Core™ i3-370M processor* *# de Núcleos: 2* *# de Hilos: 4* *Velocidad: 2.4GHz* *Cache: 3MB*
Memoria:	*6 GB of dual-channel DDR3 667 MHz*
Disco Duro:	*250GB - 7200 RPM*

Tabla 13: Características de hardware del equipo de pruebas

Software	
Característica	**Descripción**
Sistema Operativo:	*Linux : Ubuntu oneiric* *Versión: 11.10*
PostgreSQL:	*Versión: PostgreSQL 9.1.2*
***MonetDB*:**	*Versión: MonetDB 5 server v11.7.5 "Dec2011"*
***MySQL*:**	*Versión: 8.42 Distribución 5.1.58*

Tabla 14: Características de software del equipo de pruebas

ANEXO 8: Tabla con los tiempos de respuesta para las consultas en ambos DBMS

Prueba	# Tuplas	Consulta Q1 Tiempo (ms.)		Consulta Q2 Tiempo (ms.)		Consulta Q3 Tiempo (ms.)		Consulta Q4 Tiempo (ms.)		Consulta Q5 Tiempo (ms.)		Consulta Q6 Tiempo (ms.)		Consulta Q7 Tiempo (ms.)		Consulta Q8 Tiempo (ms.)		Consulta Q9 Tiempo (ms.)		Consulta Q10 Tiempo (ms.)		Consulta Q11 Tiempo (ms.)		Consulta Q12 Tiempo (ms.)	
		MonetDB	PostgreSQL	MonetDB	PostgreSQL	MonetDB	PostgreSQL	MonetDB	PostgreSQL	MonetDB	PostgreSQL	MonetDB	PostgreSQL	MonetDB	PostgreSQL	MonetDB	PostgreSQL	MonetDB	PostgreSQL	MonetDB	PostgreSQL	MonetDB	PostgreSQL	MonetDB	PostgreSQL
1	500000	27	953	62	1647	32	963	21	640	21	739	84	1486	80	1343	158	5705	146	5661	180	5866	126	3574	121	3830
2	1000000	50	1311	109	2972	53	1297	35	806	23	780	132	2098	149	1978	220	12159	246	12387	214	12561	176	6976	179	7317
3	1500000	58	1624	174	3882	81	1586	29	931	34	876	191	2879	193	2760	307	20295	315	19674	289	20527	263	11688	236	13095
4	2000000	77	1870	211	4836	104	1937	40	989	36	1232	234	3829	274	3414	368	27113	471	27209	352	27920	287	16057	281	16424
5	2500000	81	2160	267	5843	99	2273	41	1096	42	1230	257	5115	272	4176	446	36121	436	36459	433	35846	368	20682	338	21663
6	3000000	99	2522	290	7158	117	2915	46	1189	43	1497	311	5718	338	4690	507	45482	482	45912	486	46689	412	25703	402	26073
7	3500000	112	2768	330	8072	132	2981	50	1356	58	1411	319	5840	394	5625	566	53586	544	53906	544	54720	461	28615	443	29649
8	4000000	137	3022	404	8853	146	3305	70	1493	48	1678	438	6623	515	6006	653	62258	590	62203	631	63692	523	34974	567	35400
9	4500000	127	3336	463	10302	176	3769	58	1445	60	1726	377	7482	485	6954	755	69066	726	68298	698	71450	543	39724	552	40487
10	5000000	213	4000	563	11029	202	3922	62	1745	61	1526	448	7956	695	7282	825	84421	797	82568	765	85059	654	48501	612	48868
11	5500000	175	3820	600	11786	207	4418	67	1625	65	1638	490	8425	715	8113	969	96171	913	96271	893	97305	769	54268	645	54658
12	6000000	202	4193	576	12587	243	4541	65	1758	67	1749	510	9622	676	8613	1147	103237	934	105644	939	107040	727	60613	759	60761
13	6500000	208	4758	662	14818	249	5165	83	2209	82	1852	539	10271	723	9598	1126	109171	1009	108783	1102	111552	775	63778	764	66217
14	7000000	194	5048	688	15060	315	5395	78	2387	82	1981	570	11620	851	10208	1139	123878	1092	126087	1089	125482	824	71237	797	70908
15	7500000	237	5234	881	16670	268	5730	86	2378	130	2068	611	11887	914	11103	1348	135669	1192	136188	1262	137109	855	75996	849	77590
16	8000000	292	5752	903	18037	336	6888	89	2462	91	2239	661	12635	970	12415	1292	150883	1279	147883	1237	148640	1019	83899	969	83648
17	8500000	270	6095	930	18419	325	6544	96	2345	99	2357	700	13667	983	12970	1471	161015	1302	166340	1372	167109	1097	91573	965	94977
18	9000000	254	6537	979	20360	361	7099	89	2780	111	2392	724	14617	1054	13291	1636	169135	1399	168972	1428	170657	1049	97783	1129	97941
19	9500000	280	6722	918	21537	384	7806	98	2769	104	2668	828	15431	1205	14212	1638	184543	1532	181406	1517	186304	1204	106376	1218	105765
20	10000000	319	7474	1156	22470	381	7568	95	2767	101	2748	877	16369	1184	14710	1791	200320	1617	194995	1665	199483	1256	114201	1222	116008
21	10500000	305	7304	1118	22754	407	8538	116	2798	114	2799	907	16996	1389	15707	1805	212093	1726	217089	1656	212338	1315	118934	1358	119515
22	11000000	298	7848	1099	24666	403	8677	118	2994	111	2954	881	18090	1362	16441	1858	226289	1796	224612	1809	229308	1323	123595	1347	130212
23	11500000	323	7704	1311	25031	526	8667	110	3105	116	3166	948	18397	1513	17595	1880	238981	1778	239856	1860	241456	1415	132312	1291	131745
24	12000000	347	7721	1373	27564	529	9434	120	3180	152	3247	968	18335	1533	17901	2003	254958	1704	245533	1729	247016	1251	136508	1274	139650

Tabla 15: tiempos de respuesta para las consultas en ambos DBMS

ANEXO 9: Pruebas con conjuntos de datos reducidos

Para evaluar el desempeño de las consultas en presencia de conjuntos de datos más pequeños y evidenciar si el desempeño sigue siendo igual al que se presentó con intervalos de datos mayores a 500.000 registros, se ejecutaron algunas de las consultas a medida que se incrementaban los datos en la tabla de hechos. El intervalo de datos entre los que se probaron las consultas inicia en 5000 registros, que se incrementaban en 5.000 hasta alcanzar un máximo de 50.000 registros. El resultado se muestra a continuación:

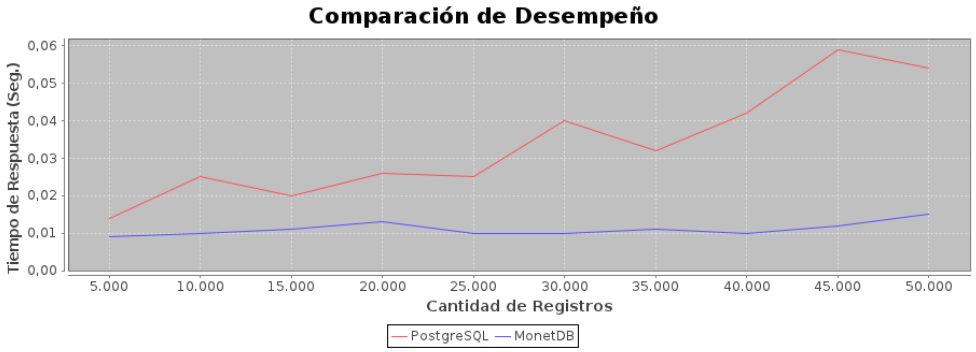


Figura 62: Tiempos de respuesta para Q1 en presencia de pocos datos, MonetDB vs PostgreSQL

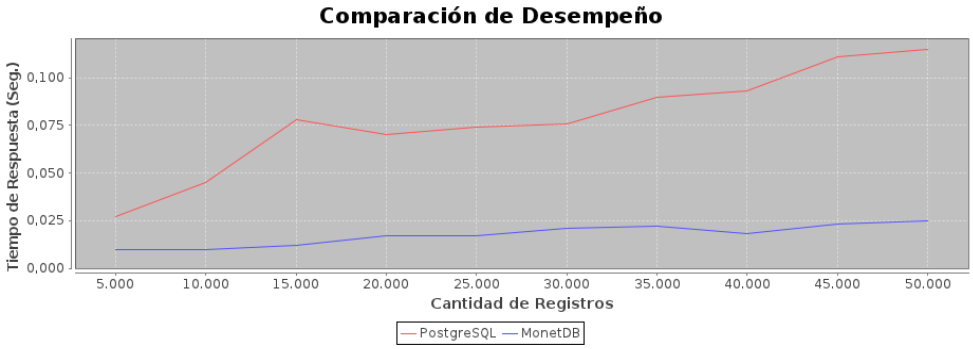


Figura 63: Tiempos de respuesta para Q2 en presencia de pocos datos, MonetDB vs PostgreSQL.

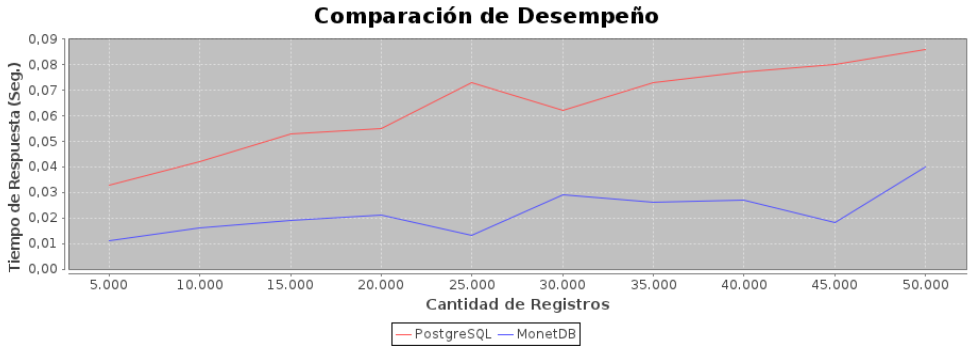


Figura 64: Tiempos de respuesta para Q7 en presencia de pocos datos, MonetDB vs PostgreSQL

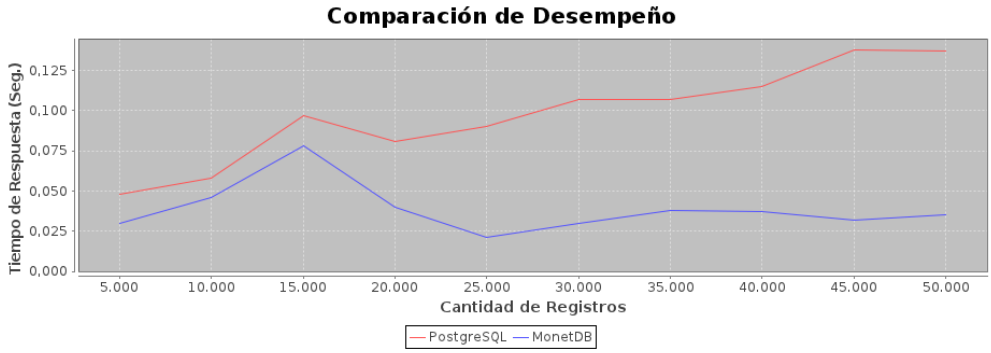


Figura 65: *Tiempos de respuesta para Q8 en presencia de pocos datos, MonetDB vs PostgreSQL*

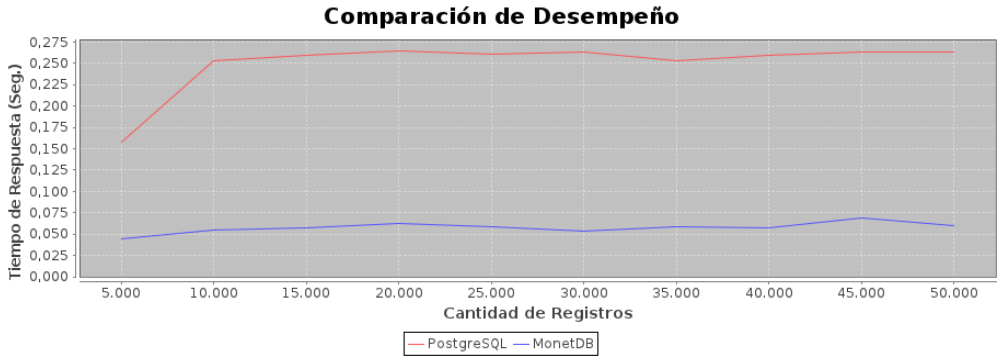


Figura 66: *Tiempos de respuesta para la consulta denominada "reconstrucción de tupla" en presencia de pocos datos, MonetDB vs PostgreSQL.*

Como se puede observar el desempeño en la mayoría de las pruebas no cambia con respecto a los resultados experimentados en las pruebas de desempeño con mayor cantidad de datos. También es posible evidenciar que para la consulta denominada reconstrucción de tupla, la cantidad de datos no es tan grande como para generar complejidad o costos adicionales en MonetDB debido a la unión o join de atributos.