RT-POSTGRESQL: EXTENSIÓN DE POSTGRESQL PARA EL MANEJO DE DATOS CON FRECUENCIAS TEMPORALES

■ Aguilera F., Ana I.
e-mail: aaguilef@uc.edu.ve
Universidad de Carabobo, Departamento de Computación, Carabobo 2005, Venezuela

■ García M., Luis D.
e-mail: ldgarc@gmail.com
Universidad de Carabobo, Departamento de Computación, Carabobo 2005, Venezuela

Fecha de Recepción: 08 de Febrero de 2010
Fecha de Aceptación: 01 de Diciembre de 2010

RESUMEN

Actualmente, la mayoría de los Sistemas Gestores de Bases de Datos (SGBD) se basan principalmente en el paradigma relacional y el lenguaje SQL para el almacenamiento y manipulación de los datos. De esta manera, tales sistemas no permiten consultar datos definidos de acuerdo al paradigma temporal de bases de datos y, mucho menos, sobre aquellos datos cuya temporalidad está basada en frecuencias de tiempo previamente definidas. El lenguaje TSQL2 es una extensión de SQL2 que permite gestionar datos basados en temporalidad de manera adecuada. Con base en esto, este trabajo enfocado en TSQL2 implementa sobre el núcleo del SGBD de código abierto PostgreSQL, mediante una Arquitectura Fuertemente Acoplada (AFA). Esta extensión desarrolla métodos adecuados para establecer frecuencias sobre el tiempo válido de los datos y optimiza la forma en la cual esos tiempos deben ser manipulados.

Palabras clave: Base de Datos, Temporalidad, Extensión, PostgreSQL, TSQL2

RT-PostgreSQL: PostgreSQL extension to data management with Temporal Frequencies

ABSTRACT

Nowadays, most of Database Management Systems (DBMS) are principally based on the relational paradigm and Structured Query Language (SQL) for data storage and management purposes. In this way, such systems do not allow querying data according to the temporal database paradigm, and even less, over those data temporality is based on a previously defined frequency. TSQL2 language is a SQL2 extension that allows data management based on temporality in correct form. Thus, this work focused on TSQL2 implements an extension on the kernel of the open source DBMS PostgreSQL, using a Strongly Coupled Architecture (SCA). This extension develops methods to establish frequencies over the valid time from data and optimizes the way in which those times must be manipulated.

Keywords: Databases, Temporality, Extension, PostgreSQL, TSQL2

1. INTRODUCCIÓN

El modelo relacional de datos [1] sobre el cual se encuentra definido el paradigma relacional para el manejo de bases de datos, se ha convertido en la base de la mayoría de los SGBD comerciales utilizados en la actualidad. Sin embargo, para dar soporte a necesidades más especificas, existen sistemas que hacen uso de otros tipos de paradigmas de bases de datos como los paradigmas deductivos, multimedia, difusos y temporales entre otros. Un ejemplo de este tipo de sistemas son los sistemas de tiempo real, los cuales se encargan de registrar datos correspondientes a hechos y fenómenos que se presentan en el mundo real, donde tanto los objetos como las relaciones existentes entre ellos se presentan en espacios específicos de tiempo.

En muchas aplicaciones del mundo real es importante modelar la dimensión temporal, como lo es la banca, el control de inventario, tráfico aeroportuario, servicios de telefonía, etc. En particular, en la industria petrolera se presentan situaciones en las cuales es conveniente que toda la información a almacenar se registre tomando en cuenta el tiempo en el cual se presentó cada situación, ya que de este dependerá la validez de los datos que se registran en cierto momento dentro de la base de datos. En este sentido, las bases de datos convencionales representan el estado de la organización en un solo instante o momento del tiempo, sin tomar en cuenta entonces que para los datos siempre se debe manejar la dimensión tiempo, durante el cual los datos son o han sido válidos en el mundo real.

Las investigaciones realizadas hasta ahora en el área de Bases de Datos Temporales se han centrado en incorporar la componente de temporalidad sobre el manejo de los datos a partir del modelo relacional. A partir de dichas investigaciones han surgido entonces una serie de lenguajes que haciendo uso del enfoque relacional, permiten gestionar aspectos de temporalidad sobre los datos. De tal manera, resaltan entonces entre estos, los lenguajes LEGOL 2.0 [2], TQuel [3], HQuel [4], TRM [5] y TSQL2 [6]. De estos, lenguaje TSQL2 es el que aporta un mayor número de funcionalidades de acuerdo con las características del paradigma de bases de datos temporales.

Debido a que los principales SGBD se encuentran implementados en base al modelo relacional básico, no cuentan con las funcionalidades necesarias para satisfacer ciertos requerimientos en aquellos casos en los cuales resulta conveniente manipular los datos de acuerdo al paradigma de bases de datos temporales. Ejemplo de esto son los SGBD OracleDB, Microsoft SQL Server y PostgreSQL, los cuales a pesar de ser reconocidos como algunos de los SGBD más avanzados del mercado, no soportan la mayoría de las definiciones descritas en ninguno de los lenguajes de consultas temporales citados anteriormente.

Con base en necesidades similares a esta, se da inicio a investigaciones enfocadas en integrar las características correspondientes a paradigmas de bases de datos distintos al relacional, con los SGBD existentes. Como resultado de estos estudios, surge un enfoque que permite integrar el proceso de Descubrimiento de Conocimiento en Bases de Datos (DCBD) con SGBD por medio de tres tipos de arquitecturas, débil, mediana o fuertemente acoplada [7]. Una arquitectura es débilmente acoplada cuando la componente encargada de realizar las tareas de manipulación de los datos de acuerdo al paradigma específico, se encuentra en una capa externa al SGBD, por fuera del núcleo, y su integración con éste se hace a partir de una interfaz. Una arquitectura es medianamente acoplada cuando las tareas específicas del paradigma forman parte del SGBD mediante el uso de procedimientos almacenados o funciones definidas por el usuario. Una arquitectura es fuertemente acoplada cuando la totalidad de las tareas que llevan a cabo la manipulación de los datos según el paradigma establecido, forman parte del SGBD como una operación primitiva, dotándolo de todas las capacidades que permitan llevar a cabo cada una de las tareas asociadas a un determinado enfoque o paradigma. Debido a que en esta última arquitectura todas las tareas son ejecutadas por el SGBD en el mismo espacio de direccionamiento que se encuentran los datos, la ventaja potencial de la misma es que resuelve los problemas de escalabilidad y rendimiento que poseen las otras arquitecturas antes mencionadas.

En función de lo antes planteado, se llevó a cabo la implementación de una extensión del SGBD PostgreSQL por medio de una arquitectura fuertemente acoplada [8], la cual permite llevar a cabo las tareas de creación de tablas e inserción y selección de datos de acuerdo a lo definido en el Lenguaje TSQL2.

En estudios futuros se propone una extensión de TSQL2 que permite manejar el tiempo válido de los datos de acuerdo a frecuencias de tiempo previamente definidas sobre estos, lo cual no es más que el tiempo transcurrido entre el momento en el cual se presentó un evento asociado a alguna variable o proceso, y el momento en el cual se presentó el evento siguiente para ese

mismo elemento. A su vez, se proponen dos tipos de correspondencias entre los datos y sus frecuencias, los datos con frecuencias en tiempo real e histórico[9]. Esta nueva extensión permite además manipular los datos de acuerdo a series de tiempo, definiendo tablas anidadas que permitan establecer valores de frecuencias distintas para atributos distintos de una misma tabla.

Basado en lo expuesto anteriormente, este artículo describe la implementación de las tareas de Creación de Tablas e Inserción y Selección de Datos de acuerdo a la sintaxis definida por Tineo y Gutierrez (2006) por medio de la extensión del SGBD PostgreSQL con base en una arquitectura fuertemente acoplada, de forma tal que se dé soporte al manejo de datos con frecuencias temporales.

2. TSQL2 Y EL MANEJO DE FRECUENCIAS TEMPORALES

En busca de recopilar las fortalezas presentadas por cada uno de los lenguajes temporales mencionados previamente, y por la imperiosa necesidad de adecuar dichas propuestas al Lenguaje de Consultas Estructurado SQL (*Structured Query Language*) considerado como un estándar para la ejecución de consultas sobre bases de datos relacionales, Snodgrass define el lenguaje TSQL2, el cual consiste en una extensión del estándar SQL-92 (también conocido por su álias, SQL2) para adecuarlo a la gestión de bases de datos temporales [6]. Las siglas TSQL hacen referencia a *Temporal Structured Query Language*, Lenguaje de Consultas Estructuradas Temporales, y el número dos (2) indica que fue desarrollado como una extensión del estándar SQL2.

El lenguaje TSQL2 es en sí, una extensión del Lenguaje de Definición de Datos (*Data Definition Language, DDL*) y del Lenguaje de Manipulación de Datos (*Data Manipulation Language, DML*) planteados por el lenguaje SQL2 para dar soporte a la dimensión temporal de los datos. A continuación se describen los principales aportes de TSQL2 y su respectiva extensión para el manejo de frecuencias de tiempo:

2.1. Creación de tablas con frecuencias temporales

El lenguaje TSQL2 permite definir Tablas Temporales con marcas de tiempo, de tal manera que cada registro que se almacene en alguna de estas tablas posea un tiempo válido y/o de transacción asociado al mismo. En la Figura 1 puede observarse un ejemplo de una tabla temporal básica. A su vez, la extensión realizada sobre dicho lenguaje permite definir frecuencias sobre los tiempos válidos definidos. En la Figura 2 se puede observar la sintaxis para la creación de tablas con frecuencias temporales.

Valor	Tiempo Válido
30.5	27/05/2007 03:00:05.00
30.7	27/05/2007 03:00:10.00
31	27/05/2007 03:00:15.00

Fig. 1. Ejemplo: Tabla temporal básica

```
CREATE TABLE <table name> <table
elements>
       [ <temp definition> ]      [
<freq definition> ]

<temp definition>::=
       AS { VALID [ STATE | EVENT ] }


              [ <timestamp_precision> ]
              [ AND TRANSACTION ]
       | AS TRANSACTION

<freq definition> ::=
WITH     FREQUENCY     <frequency
measure>
              AS { REAL TIME | HISTORIC
}
       |   WITH   FREQUENCY   <frequency
measure>
```

Fig. 2. Sintaxis para la creación de tablas con frecuencias temporales [9]

La sintaxis mostrada en la Figura 2 permite crear tablas temporales con dos tipos de frecuencias, en tiempo real e histórico, definida con una medida <frequency measure>. Estos dos métodos son descritos a continuación:

2.1.1. Tablas en Tiempo Real: permiten dar prioridad al proceso de almacenamiento y consulta de los datos durante la actividad asociada a sistemas críticos, ya que por estar sujetos a frecuencias de tiempo, se puede almacenar físicamente en la tabla cada uno de los datos generados pero basta con almacenar únicamente el tiempo válido del primer dato que se haya ingresado en dicha tabla, ya que los tiempos válidos de los datos siguientes pueden ser calculados a partir de éste y la frecuencia definida sobre la tabla. En la Figura 3 puede observarse un ejemplo de este método.

Fig. 3. Ejemplo: Tabla con frecuencias en tiempo real

2.1.2. Tablas Históricas: permiten definir tendencias sobre los valores registrados en una tabla temporal, almacenando únicamente aquellos registros que representen una variación en la tendencia actual de los datos almacenados en la tabla. De esta manera se prescinde de datos que puedan resultar redundantes dentro del proceso de toma de decisiones, pero que a su vez pueden ser inferidos luego por medio de los datos que sí han sido almacenados, sólo que con cierto margen de error. En la Figura 4 se puede observar la transformación de la tabla temporal básica mostrada en la Figura 1, en una tabla histórica.

Valor	Tiempo Válido
30,5	27/05/2007 03:00:05,00
31	27/05/2007 03:00:15,00

Fig. 4. Ejemplo: Transformación de la tabla de la Fig, 1 en una tabla histórica

2.2. Creación de tablas con datos en series de tiempo

En ciertos casos es posible que en un mismo punto en el tiempo, dos variables que deben almacenarse dentro de una misma tabla posean frecuencias de tiempo diferentes definidas sobre ellas. Por ende, una tabla debe permitir almacenar registros con diversos valores y tipos de frecuencias de tiempo definidos para cualquier atributo de ellos. Una solución para este problema es utilizar tablas anidadas que puedan representar cada una de estas variables, almacenándolas dentro de una tabla base o tabla padre relacional básica. En la Figura 5 puede observarse un ejemplo de este tipo de tablas.

Para dar soporte a esta funcionalidad, se propone la definición de tablas con frecuencias temporales como tipo de dato, lo cual permitirá luego utilizar el tipo de dato tabla definido como tipo de dato asociado a cualquier atributo de la tabla base. Esto puede llevarse a cabo por medio de la sintaxis expuesta en la Figura 6.

2.3. Inserción de datos en tablas con frecuencias temporales

En el caso de la inserción de datos, y en contraste con la operación de definición de tablas, la única modificación expuesta en la extensión de TSQL2, es con respecto al proceso de inserción en tablas con datos en series de tiempo, ya que al insertar datos en la tabla anidada, el proceso de inserción debe hacerse para un registro en específico de la tabla base correspondiente. Para realizar esto, la sintaxis propuesta se puede observar en la Figura 7.

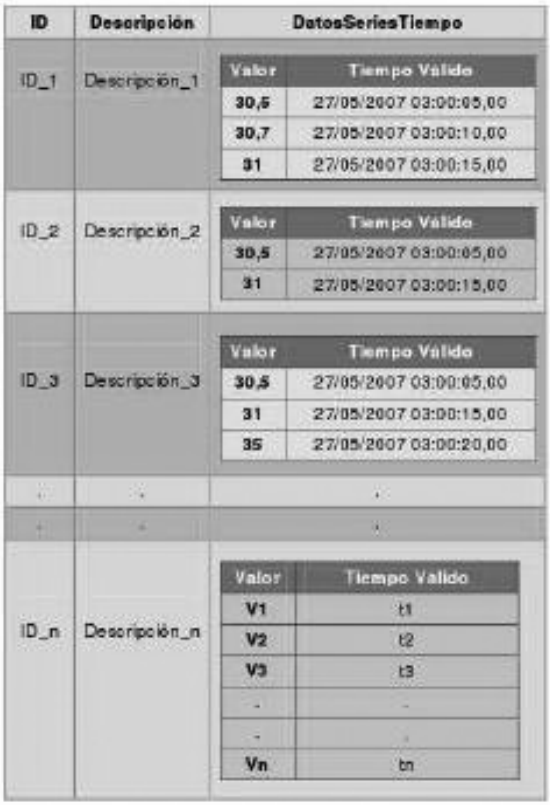

Fig. 5. Ejemplo: Tabla con datos en series de tiempo

```
CREATE TYPE TABLE <table type name>
       <table elements>
       <temp definition>  <freq
definition>
```

Fig. 6. Sintaxis para la definición de tablas con frecuencias temporales como tipos de dato [9]

```
INSERT        INTO <nested table name>
              FROM <base table name>
              WHERE <base table access
condition>

       VALUES (<values list>)
                                    <temp
insertion> ;

<temp insertion>:=
       [ VALID 'expr'
       | VALID \[  'initialExpr' –
'finalExpr' \]  ]
```

Fig. 7. Sintaxis para la inserción de datos en tablas con datos en series de tiempo [9]

La sintaxis expuesta en la Figura 7 permite realizar la inserción de datos sobre la tabla anidada *<nested table name>*, la cual se encuentra en el registro de la tabla *<base table name>* que cumpla con la condición *<base table access condition>*.

2.4. Selección de datos sobre tablas con frecuencias temporales

De manera similar al caso de la Inserción, la selección de acuerdo a la extensión de TSQL2 para el manejo de frecuencias sólo afecta la selección de datos almacenados en tablas con series de tiempo. Esto se debe a que al realizar consultas de datos sobre tablas con series de tiempo se debe tomar en cuenta que las mismas se deben realizar sobre la tabla representada por un registro o tupla en específico de la tabla base. Para realizar esto, la sintaxis propuesta se puede observar en la Figura 8.

```
SELECT        <target list>
              FROM <nested table name>
              FROM <base table name>
              WHERE <base table access
condition>
              WHERE    <nested    table
access condition>
              <group by clause> <having
clause>
```

Fig. 8. Sintaxis para la selección de datos sobre tablas con datos en series de tiempo [9]

3. POSTGRESQL

PostgreSQL es un SGBD objeto-relacional desarrollado bajo la ideología de Software Libre. Este sistema se origina a partir de un proyecto de investigación de la Universidad de California, el cual consistía en desarrollar un SGBD relacional. El sistema desarrollado fue denominado INGRES. Debido a la evolución constante de este sistema, después de recibir distintos nombres en el transcurso del tiempo, y en busca de satisfacer las definiciones establecidas por el lenguaje SQL, es a principios de 1997, cuando surge la primera versión oficial de PostgreSQL, el cual correspondía para ese entonces con la versión 6.0 de su predecesor INGRES. A partir de ahí el desarrollo del sistema ha estado a cargo del "PostgreSQL Global Development Group", un grupo de desarrolladores distribuidos a nivel mundial.

3.1 Estructura Interna

Internamente el SGBD PostgreSQL consta de cuatro módulos, denominados PARSER, REWRITER, PLANNER/OPTIMIZER y EXECUTOR. Cada uno de estos módulos se encarga de llevar a cabo funciones específicas de acuerdo a la consulta que se ha de ejecutar sobre la base de datos. En la Figura 9 se puede observar el esquema de procesamiento de una consulta en PostgreSQL .

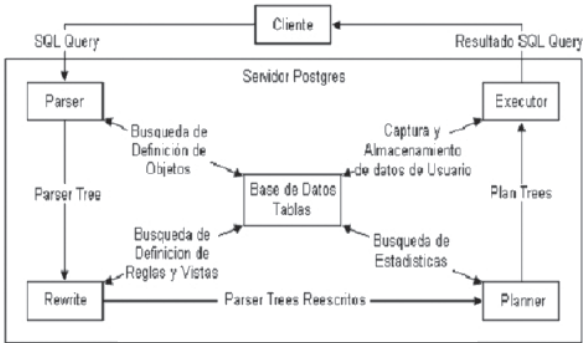

Fig. 9. Procesamiento de una consulta en PostgreSQL [10]

De manera general, el procesamiento de una consulta en PostgreSQL se lleva a cabo por medio de una conexión desde un programa aplicación al servidor Postgres, el cual se encarga de recibir las consultas del cliente y enviar los resultados de vuelta. Para enviar dicho resulta se realiza en el PARSER un análisis léxico, sintáctico y semántico a la consulta recibida, verificando entre otras cosas la existencia de las relaciones y atributos indicados en la consulta. Una vez hecho esto, se genera un árbol de consulta llamado query tree, el cual se envía al REWRITER para su modificación de acuerdo

a las Reglas o Vistas en las que puedan estar implicados los elementos involucrados en la consulta. Luego de esto se procede a la etapa de optimización dentro del PLANNER/OPTIMIZER, donde se toma el árbol de la consulta y se generan todos los planes de ejecución con rutas de acceso que conduzcan al mismo resultado, de manera tal que se pueda seleccionar aquel plan con menor costo de ejecución. Por último, para recuperar los registros solicitados en la consulta, el módulo EXECUTOR recupera de cada tabla las tuplas respectivas, de acuerdo a lo descrito en el plan desarrollado en el módulo anterior.

3.2. Importancia de PostgreSQL en la Implementación

En el transcurso de los años PostgreSQL ha tomado un puesto importante en el área de los SGBD, siendo uno de los sistemas más utilizados a nivel mundial en una amplia gama de aplicaciones. Adicionalmente, debido a la filosofía de Software Libre sobre la cual se basa el desarrollo de PostgreSQL, el código fuente del mismo se encuentra disponible para su revisión y modificación. A partir de esto, se selecciona entonces este sistema para llevar a cabo la implementación de la extensión descrita en este artículo.

4. RT-POSTGRESQL

La implementación de RT-PostgreSQL comprende la inclusión en el SGBD PostgreSQL de tres actividades u operaciones específicas de la extensión del lenguaje TSQL2 para el manejo de frecuencias temporales. Estas operaciones son la Creación de Tablas, Inserción y Selección de Datos, tanto para tablas con frecuencias básicas, como para aquellas con datos en series de tiempo. A continuación se detallan los aspectos asociados a la extensión de PostgreSQL para la implementación de la arquitectura:

4.1. Creación de Tablas con Frecuencias Temporales

La definición o creación de tablas en PostgreSQL se lleva a cabo mediante la sentencia CREATE TABLE. En el trabajo presentado por Chacón (2007) se integra a la definición de tablas en PostgreSQL, la componente de temporalidad, agregando a la sintaxis definida para ello, la sentencia *<temp definition>* (ver Figura.2).

A continuación se describen los cambios realizados dentro de la estructura interna PostgreSQL, para la modificación de la sentencia CREATE TABLE:

- Adición de palabras FREQUENCY e HISTORIC al archivo *keywords.c* (directorio *src/backend/parser/*), de manera tal que el módulo PARSER de PostgreSQL reconozca estás palabras como reservadas.
- Definición de gramática de la sentencia *<freq definition>* en el archivo *gram.y* (directorio *src/backend/parser/*).
- En el archivo *parsenodes.h* (directorio *src/include/nodes/*) se agregó la estructura FrequencyDefinition, encargada de almacenar los valores correspondientes a la sentencia <freq definition> del CREATE TABLE. Adicionalmente, se modificó la estructura *CreateStmt*, de manera tal que se diera soporte a la estructura añadida.
- La etiqueta *T_FrequencyDefinition* fue incluida dentro del archivo *nodes.h* (directorio *src/include/nodes/*), de manera tal que haga referencia a la estructura *FrequencyDefinition* ya incluida.
- En el archivo relcache.c (directorio *src/backend/utils/cache/*) se modificó la función *RelationBuildLocalRelation*(), de manera tal que la misma dé el soporte adecuado a la definición de tablas basadas en frecuencias de tiempo.
- Se extendió la estructura pg_class, definida en el archivo pg_class.h (directorio src/include/catalog/) de manera tal que se almacene en el catálogo de PostgreSQL la información asociada a la estructura de la tabla temporal definida.

4.2. Definición de Tablas con Frecuencias Temporales como Tipo de Dato.

Para dar soporte a la definición de tablas con datos en series de tiempo, es necesario definir tablas como tipos de datos, lo cual hace introducción al concepto de Tablas Anidadas. Para llevar a cabo la implementación de esta fincionalidad se llevaron a cabo las siguientes modificaciones dentro de la estructura interna de PostgreSQL:

- En el archivo *parsenodes.h* (directorio *src/include/nodes/*) se modificó la estructura *CompositeTypeStmt*, de forma tal que sea posible almacenar en ella las estructuras *tempDef* y *freqDef*, las cuales almacenan las sentencias *<temp definition>* y *<freq definition>* respectivamente, de manera

tal que se dé el soporte a la definición de tipos de dato tabla.

- Se incluyó la sintaxis correspondiente a la sentencia <*type table definition*> dentro en el archivo gram.y (directorio *src/backend/parser/*).
- En el archivo *typecmds.c* (directorio *src/backend/commands/*) se modificó la función *DefineCompositeType()*, de manera tal que la misma dé el soporte adecuado a las características definidas por las sentencias <*temp definition*> y <*freq definition*> utilizadas para la creación de tablas como tipos de dato.

4.3. Inserción de datos en base a Frecuencias de Tiempo.

La inserción de datos sobre tablas en PostgreSQL se lleva a cabo mediante la sentencia INSERT. En el trabajo realizado por Chacón (2007) se implementa la inserción de datos en tablas temporales dentro de PostgreSQL, por medio de la adición al <*insert definition*> de estructuras y funciones adicionales necesarias para llevar a cabo la inserción en tablas temporales de acuerdo al lenguaje TSQL2. Sobre dicha implementación, y para desarrollar la extensión aqui descrita, se definen dentro de la estructura interna de PostgreSQL una serie de procedimientos de compresión y manipulación de datos, los cuales permiten gestionar el método de almacenamiento de datos en tablas con frecuencias temporales. A continuación se describen los cambios realizados:

Se modificó el archivo *parsenodes.h* (directorio *src/include/nodes/*) agregando a la estructura InsertStmt dos campos adicionales para satisfacer los casos en los cuales la inserción deba hacerse sobre una tabla anidada. El primero de estos campos es *upper_rel*, el cual hace referencia a la tabla dentro de la cual se ha definido un atributo de tipo tabla temporal. El segundo es *where_clause*, el cual almacena la condición que debe cumplir el registro correspondiente a la tabla almacenada dentro de la tabla base.

Se incluyó en el archivo *gram.y* (directorio *src/backend/parser/*) la gramática asociada a la inserción sobre tablas con datos en series de tiempo.

En el archivo *analyze.c* (directorio *src/backend/parser/*) se definieron una serie de rutinas de evaluación y validación de los datos a insertar. Se modificó la función *transformInsertStmt()* y se incluyó dentro del código fuente la función *Nested_Table_Insert_Validation()*,dando así soporte a la componente de tablas con datos en series de tiempo. De igual forma, se llevaron a cabo extensiones de las funciones *TSQL_TargetList_modification()* y *TSQL_catalog_verification_and_InsertSt mt_modification()* definidas por Chacón (2008), de manera tal que dentro de ellas se dé el soporte correspondiente a la inserción de datos sobre tablas con frecuencias y en series de tiempo, aportando de esta manera la capacidad de compresión mencionada anteriormente propuesta por Tineo y Gutierrez.

4.4. Selección de datos en base a Frecuencias de Tiempo

El desarrollo de esta funcionalidad se lleva a cabo mediante la extensión de la sentencia SELECT de PostgreSQL, esto de acuerdo a la sintaxis definida por la extensión del lenguaje TSQL2 para el manejo de frecuencias de tiempo.

En este apartado no hubo necesidad de adicionar sentencia alguna para implementar la selección sobre tablas con frecuencias temporales básicas, sin embargo, se modificó el mecanismo de selección para aquellos casos en los cuales la selección de datos debe realizarse sobre tablas con datos en series de tiempo. Para esto se llevaron a cabo las siguientes modificaciones:

En el archivo parsenodes.h (directorio *src/include/nodes/*), se modificó la estructura SelectStmt, de manera tal que se almacenen en esta dos nuevos valores, el primero, *NestedFromClause*, indicará la tabla padre en caso de que sea una selección en una tabla anidada, y el segundo, *NestedWhereCondition*, indicará la condición que debe cumplir el registro correspondiente dentro de la tabla padre.

En el archivo *gram.y* (directorio *src/backend/parser/*) se extendió la sintaxis de la sentencia *simple_select* de manera tal que se evaluara la gramática definida para la selección de datos sobre tablas anidadas de acuerdo a la sintaxis propuesta.

En el archivo *analyze.c* (directorio *src/backend/parser/*) se definieron distintas rutinas de evaluación de las consultas a realizar, de manera que se valide que el acceso a los datos se haga sobre la tabla anidada adecuada. Se agregó a su vez la función *Nested_Table_Select_Validation()*, de manera tal que se encargue de analizar el SelectStmt proveniente del modulo de *PARSER* y en caso de que sea necesario haga las modificaciones pertinentes para dar soporte a la selección de datos sobre tablas anidadas. Adicionalmente, se hizo uso de rutinas internas de PostgreSQL para permitir realizar consultas sobre los datos de las tablas anidadas directamente desde el backend de este SGBD.

5. RESULTADOS

A continuación se presenta un subconjunto de los resultados arrojados por la etapa de pruebas de la implementación de RT-PostgreSQL. Para dichas pruebas se consideraron los factores *Funcionalidad y Compresión*. Estos dos factores se evalúan a partir de un estudio experimental, en el cual se establece un conjunto de hipótesis asociadas a los resultados que se espera arroje el sistema RT-PostgreSQL, en pro de verificar el correcto funcionamiento del mismo. Así, el ambiente definido para las pruebas está compuesto por los siguientes elementos:

- *med_caldera:* tabla temporal con tiempo válido de tipo período y frecuencia en tiempo real de 1 hora.
- *temp_pozo:* tabla temporal con tiempo válido de tipo evento y frecuencia histórica de 5 segundos
- *TimeSeriesVol:* tipo de dato tabla con tiempo válido de tipo evento y frecuencia en tiempo real de 30 segundos.
- vol_data: tabla relacional que posee la tabla anidada *tsv* de tipo *TimeSeriesVol.*

5.1. Funcionalidad

Las pruebas aquí descritas se encuentran agrupadas en función de las operaciones implementadas: creación de tablas, inserción de datos y selección de datos. Cada operación se llevó a cabo por medio del terminal de PostgreSQL, psql y sobre una base de datos de prueba llamada RT-PostgreSQL.

5.1.1. Creación de Tablas

a) Tablas con Frecuencias Básicas: al crear una tabla por medio de la sintaxis definida por la extensión de TSQL2 para el manejo de frecuencias temporales, el resultado que debe arrojar RT-PostgreSQL es el mensaje “CREATE TABLE”, indicando que se ha creado la tabla en cuestión. En la Figura 10 se puede observar la prueba realizada.

```
RT-PostgreSQL=# CREATE TABLE
med_caldera (idm int, valor float,
planta varchar(20) ) AS VALID STATE
WITH FREQUENCY '00:00:05.00';
CREATE TABLE
RT-PostgreSQL=#      CREATE TABLE
temp_pozo (idp int, temp float) AS
VALID EVENT 2 WITH FREQUENCY
'00:00:05.00' AS HISTORIC;
CREATE TABLE
```

Fig. 10. Pruebas funcionales de creación de tablas con frecuencias temporales

Resultado: Funcionamiento adecuado

b) Tablas con Datos en Series de Tiempo:

- Tipo de Dato Tabla con Frecuencia de Tiempo: al definir un tipo de dato tabla por medio de la sintaxis definida por la extensión de TSQL2 para el manejo de frecuencias temporales, el resultado que debe arrojar RT-PostgreSQL es el mensaje “CREATE TYPE”, indicando de esta manera que se ha creado el tipo de dato tabla. En la Figura 11 se puede observar la prueba realizada.

```
RT-PostgreSQL=#     CREATE TYPE TABLE
TimeSeriesVol AS (vol_act float,
vol_min float, vol_max float) AS VALID
EVENT WITH FREQUENCY '00:00:30.00';
CREATE TYPE
```

Fig. 11. Pruebas funcionales de creación de tipo de dato tabla con frecuencia temporal

Resultado: Funcionamiento adecuado

Tabla con datos en series de tiempo: al definir una tabla con un atributo de tipo tabla con frecuencia de tiempo, el sistema RT-PostgreSQL debe, por tratarse de la creación de una tabla, enviar al usuario el mensaje “CREATE TABLE”, indicando que la creación de la tabla se llevo a cabo. En la Figura 12 se puede observar la prueba realizada.

```
RT-PostgreSQL=# CREATE TABLE vol_data
(idd int primary key, tsv
TimeSeriesVol);
CREATE TABLE
```

Fig. 12. Pruebas funcionales de creación de tablas con datos en series de tiempo

Resultado: Funcionamiento adecuado

c) Pruebas en el Catálogo del sistema: en vista de que es en la estructura pg_class del catálogo de RT-PostgreSQL, donde se almacena la información correspondiente a cada una de las tablas que se definan dentro de una base de datos, al realizar una consulta sobre ella, se debe corroborar que almacena la información correspondiente a los distintos elementos creados a partir de de la extensión del lenguaje TSQL2 para el manejo de frecuencias temporales. En la Figura 13 se puede observar la prueba realizada.

```
RT-PostgreSQL=#   SELECT relname,
relhasfrequency, relfrequencytype,
relfrequency, relhistorictrend,
rellastvalidtime, relnestedtable
FROM pg_class WHERE relname
IN('med_caldera', 'temp_pozo',
'vol_data');
relname         | relhasfrequency |
relfrequencytype
--------------- + -------------------- + ---------------------
med_caldera | t                    | REAL TIME
temp_pozo   | t                    | HISTORIC
vol_data      | f                  | NULL

relfrequency | relhistorictrend|
rellastvalidtime
--------------- + ------------------ + -------------
01:00:00.00    | 0                  |
 00:00:05.00  | 0                   |
 0             | 0                  |
```

```
relnestedtable
---------------
0
0
57490
```

Fig.13. Pruebas en el catálogo de RT-PostgreSQL

Resultado: Funcionamiento adecuado

5.1.2. Inserción de Datos

a) Inserción en Tablas con Frecuencias Básicas: al insertar datos dentro de alguna tabla alojada en bases de datos que se encuentren en el sistema RT-PostgreSQL, este enviará un mensaje indicando que el dato se ha insertado de manera adecuada. A su vez, para aquellas tablas que se encuentren definidas a partir de la extensión de TSQL2 para el manejo de frecuencias temporales, el tiempo válido de cada dato a almacenar debe concordar con la frecuencia que haya sido definida sobre la tabla en cuestión. En la Figura 14 se puede observar la prueba realizada.

b) Inserción en Tablas con Datos en Series de Tiempo: el sistema RT-PostgreSQL debe permitir que se lleve a cabo la inserción de datos sobre tablas con datos en series de tiempo por medio de la sintaxis definida por la extensión de TSQL2 para al manejo de frecuencias temporales. Una vez que un dato se haya almacenado de manera correcta, RT-PostgreSQL envía un mensaje indicando que el dato se ha insertado adecuadamente. En las Figuras 15 y 16 se pueden observar las pruebas realizadas.

```
RT-PostgreSQL=# INSERT INTO med_caldera
VALUES (1, 44.921650, 'Refinería el
Palito') VALID ['10-10-2004 11:00' -
'10-10-2004 12:00'];
   INSERT 0 1

INSERT INTO med_caldera VALUES (2,
59.921650, 'Refinería el Palito') VALID
['10-10-2004 12:00' - '10-10-2004
13:00'];
   INSERT 0 1

INSERT INTO med_caldera VALUES (3,
74.921650, 'Pequiven ORIENTE') VALID
['10-10-2004 13:00' - '10-10-2004
14:00'];
   INSERT 0 1

INSERT INTO med_caldera VALUES (4,
89.921650, 'Refinería el Palito') VALID
['10-10-2004 14:00' - '10-10-2004
15:00'];
   INSERT 0 1!
!
```

Fig. 14. Inserción de datos sobre la tabla *temp_pozo*

Resultado: Funcionamiento adecuado

```
RT-PostgreSQL=# INSERT INTO tsv FROM
vol_data WHERE idd=0  VALUES
(430.6,350.00,600.00)
VALID '03-11-2008 03:15:00.00';
   INSERT 0 1

RT-PostgreSQL=# INSERT INTO tsv FROM
vol_data WHERE idd=0  VALUES
(230.6,150.00,330.00)
VALID '03-11-2008 03:15:30.00';
   INSERT 0 1

RT-PostgreSQL=# INSERT INTO tsv FROM
vol_data WHERE idd=1  VALUES
(430.6,350.00,600.00)
VALID '03-11-2008 03:15:00.00';
   INSERT 0 1
```

Fig. 15. Inserción de datos sobre la tabla tsv anidadda en la tabla *vol_data*

Resultado: Funcionamiento adecuado

```
RT-PostgreSQL=# INSERT INTO vol_data
VALUES (0);
 INSERT 0 1

RT-PostgreSQL=# INSERT INTO vol_data
VALUES (0);
 INSERT 0 1!
```

Fig. 16. Inserción de datos sobre la tabla vol_data

Resultado: Funcionamiento adecuado

5.1.3. Selección de Datos

En esta sección se exponen las pruebas realizadas para evaluar la selección de datos por medio del sistema RT-PostgreSQL sobre tablas definidas a partir de la extensión de TSQL2 para el manejo de frecuencias temporales.

a) Selección en Tablas con Frecuencias Básicas: el sistema RT-PostgreSQL debe permitir que se ejecuten consultas sobre los datos almacenados en tablas con frecuencias de tiempo básicas. El resultado de estas consultas deben ser aquellos datos que cumplan con la condición impuesta por la selección en cuestión. En la Figura 17 se puede observar el resultado de "Listar todos los datos de la tabla med_caldera que hayan sido válidos entre las 8:00am del 01/01/2004 y las 8:00pm del 12/12/2004".

```
RT-PostgreSQL=#     SELECT * FROM
med_caldera AS M WHERE PERIOD
( TBEGIN('2004-01-01 08:00:00'),
TEND('2004-12-12 20:00:00') ) CONTAINS
VALID(M);

        idm     |   valor   |      planta
       -------+-------------+--------------------
                1    | 44.92165      | Refinería el
Palito

        initial_valid_time   |   final_valid_time
       -------------------------+-------------------------
       2004-10-10 11:00:00  | 2004-10-10 12:00:00
```

Fig. 17. Consulta sobre la tabla med_caldera

Resultado: Funcionamiento adecuado.

b) Selección en Tablas con Datos en Series de Tiempo: RT-PostgreSQL debe ser capaz de permitir consultas sobre tablas que han sido definidas con datos en series de tiempo. De manera similar al caso anterior, el resultado de estas consultas deben ser aquellos datos que cumplan con la condición impuesta por la selección en cuestión, sólo que en este caso se seleccionan los datos correspondientes a la tabla anidada dentro del registro que cumpla con a la condición impuesta sobre la tabla base. En la Figura 18 se puede observar el resultado de "Listar todos los datos de la tabla tsv de vol_data cuyo idd sea 0. Los datos a listar deben ser aquellos que posean un val_act mayor a cualquiera que se encuentre almacenado dentro de la tabla tsv de vol_data cuyo idd sea 1".

Observación: otra característica deseable del sistema RT-PostgreSQL es que el mismo permita llevar a cabo consultas sobre aquellos datos que hayan sido comprimidos en base al tipo de frecuencia definida sobre ellos, sin embargo, esta característica no se pudo incluir dentro del núcleo de funcionalidades del sistema.

```
RT-PostgreSQL=# SELECT * FROM tsv FROM
vol_data WHERE idd = 0 WHERE vol_act > ANY
(SELECT vol_act FROM tsv FROM vol_data
WHERE idd=1);
```

vol_act	vol_min	vol_max	valid_time
430.6	350	600	2008-03-11 03:15:00
333.42	300	350.3	
420.17	140	600	

Fig. 18. Consulta sobre la tabla anidada tsv de vol_data

Resultado: Funcionamiento adecuado.

5.2. Mecanismos de Compresión de Datos

Una de las características más relevantes del sistema RT-PostgreSQL es el uso de mecanismos de compresión de datos de acuerdo al tipo de frecuencia de tiempo que se encuentre definida sobre estos. De esta manera, el sistema permite llevar a cabo una reducción del volumen de información que se desee procesar y almacenar dentro de las bases de datos que el mismo gestione.

5.2.1. Hipótesis

A continuación una serie de hipótesis que permitirán evaluar si los mecanismos de compresión de RT-PostgreSQL cumplen con las expectativas fijadas durante su desarrollo.

a) Hipótesis A (HA): el tamaño que ocupa en disco una tabla T definida por medio del lenguaje TSQL2 es mayor al tamaño que ocuparía esa misma tabla T si estuviese definida sobre ella alguna frecuencia de tiempo de acuerdo a la extensión de TSQL2 para el manejo de frecuencias temporales.

b) Hipótesis B (HB): el tamaño que ocupa en disco una tabla T sobre la cual se encuentra definida una frecuencia en tiempo real es menor que el tamaño que ocuparía esa misma tabla T si estuviese definida por medio del lenguaje TSQL2 y mayor que el tamaño que ocuparía la tabla si sobre ella estuviese definida una frecuencia histórica.

c) Hipótesis C (HC): el tamaño que ocupa en disco una tabla T sobre la cual se encuentra definida una frecuencia de tipo histórico es menor que el tamaño que ocuparía esa misma tabla T si la frecuencia definida sobre ella fuese de tiempo real o si no existiese frecuencia alguna definida sobre ella.

5.2.2. Experimentos

A continuación se describen una serie de experimentos que permitirán evaluar si las hipótesis establecidas en el apartado anterior son verdaderas. De ser así, queda demostrado que cada uno de los mecanismos de compresión implementados dentro del sistema RT-PostgreSQL son efectivos.

Estos experimentos consisten en realizar una serie de inserciones de datos sobre distintas versiones de la tabla temp_pozo definida en la Figura 10. Luego, se evaluará el tamaño que ocupan en disco cada una de esas versiones. Para esto, se hace uso de las funciones nativas del gestor de bases de datos PostgreSQL, pg_relation_size() y pg_size_pretty(). La primera de estas permite, a partir del nombre de una tabla, obtener el tamaño de bytes en disco ocupados por dicha tabla. La segunda permite convertir cualquier medida en bytes a un formato de mayor legibilidad para el usuario.

A partir de esto se define la fórmula para el cálculo del tamaño de una tabla como pg_size_pretty(pg_relation_size("nombre")).

Ahora, se definen entonces para realizar los experimentos, tres versiones correspondientes a la tabla temp_pozo:

- Versión 1: se define la tabla temp_pozo como una tabla temporal con tiempo válido de tipo evento con precisión de 2 milisegundos.

- Versión 2: se define la tabla temp_pozo como una tabla temporal con tiempo válido de tipo evento con precisión de 2 milisegundos y con frecuencia de tiempo de 5 segundos de en tiempo real.

- Versión 3: se define la tabla temp_pozo como una tabla temporal con tiempo válido de tipo evento con precisión de 2 milisegundos y con frecuencia de tiempo de 5 segundos de tipo histórico.

a) Experimento I:para este experimento se define el siguiente ambiente de prueba:

- Se almacenará el valor correspondiente a la temperatura de un pozo durante un (1) día completo de labores.

- Se asume que la temperatura del pozo se mantuvo invariante en 30.1 ºC durante todo el día.

A partir de este escenario, los resultados de realizar la inserción de datos en cada una de las 3 versiones de la tabla temp_pozo descritas anteriormente puede observarse en la Figura 19.

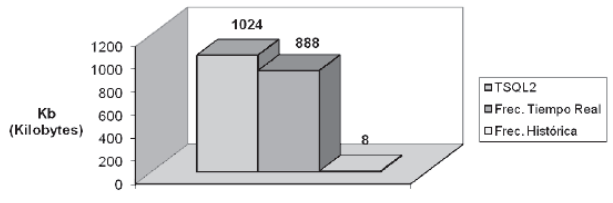

Fig. 19. Resultado del Experimento I

En la gráfica anterior se puede observar que los mecanismos de compresión, de acuerdo al escenario planteado para este experimento, satisfacen las hipótesis H1, H2 y H3. Esto puede determinarse a partir del valor en Kilobytes que ocupa en disco cada versión de la tabla temp_pozo, 1024Kb definida en TSQL2, 888Kb con frecuencia en tiempo real y 8Kb con frecuencia histórica.

Es importante acotar que el número de registros almacenados en las versiones 1 y 2 de temp_pozo es de 17279 registros, mientras que para la versión 3, la cantidad de registros almacenados son sólo 2, indicando que desde la primera hora del día de labores, hasta la última, la tendencia que se mantuvo fue la misma (30,1 ºC).

b) Experimento II: en vista de que en el Experimento I se almacenan datos únicamente durante un día, se plantea entonces un nuevo ambiente de prueba donde:

- Se almacenará el valor correspondiente a la temperatura de un pozo durante tres (3) días completos de labores.

- Se asume una vez más que la temperatura del pozo se mantuvo invariante en 30.1 ºC durante todo el día.

A partir de este escenario, puede observarse en la Figura 20, los resultados de realizar la inserción de datos en cada una de las 3 versiones de la tabla temp_pozo descritas anteriormente.

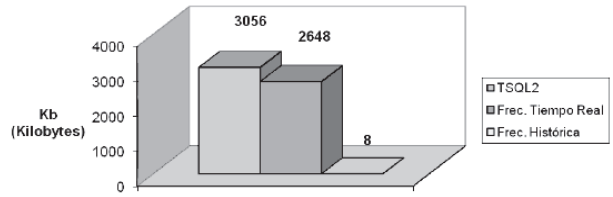

Fig. 20. Resultado del Experimento II

En la gráfica anterior, al igual que en la gráfica correspondiente al Experimento I, se puede observar que los mecanismos de compresión para el escenario planteado para este experimento satisfacen las hipótesis H1, H2 y H3. Puede observarse a su vez que la diferencia entre el tamaño en Kilobytes que ocupa en disco cada versión de la tabla sigue siendo considerable, sobretodo para la versión 3, aquella con frecuencia histórica. Siendo 3052Kb el tamaño para la versión de temp_pozo en TSQL2, 2648Kb para la versión con frecuencia en tiempo real y 8Kb para la definida con frecuencia histórica

Así mismo, el número de registros almacenados para las versiones 1 y 2 de temp_pozo es bastante grande, 51839 registros, en comparación con la versión 3, la cual almacena solo 2 registros, los cuales reflejan que la tendencia que mantuvo la temperatura durante los 3 día de labores fue la misma (30,1 ºC).

c) Experimento III:si bien los resultados de los Experimentos I y II reflejan mejoras considerables en el espacio que consumen en disco los datos almacenados sobre tablas con frecuencias de tiempo, los datos utilizados para estos experimentos pueden considerarse como ingenuos, ya que en el mundo real puede resultar poco probable que se mantenga un mismo valor de temperatura durante uno o varios días. Por esto, se plantea para este experimento un nuevo ambiente de prueba donde:

- Se almacenará el valor correspondiente a la temperatura de un pozo durante tres (3) días completos de labores.

- Se asume que, por verse afectada por factores climáticos, la temperatura del pozo varía de acuerdo a la hora del día y de la siguiente manera:

- 30.1 ºC de 6:00am a 8:00am
- 35.5 ºC de 9:00am a 10:00am
- 38 ºC de 11:00am a 1:00pm
- 49.5 ºC de 2:00pm a 3:00pm
- 40.5 ºC de 4:00pm a 5:00pm
- 37.5 ºC de 6:00pm a 7:00pm
- 20 ºC de 8:00pm a 5:00am

A partir de este escenario, los resultados obtenidos en la inserción de datos en cada una de las 3 versiones de la tabla temp_pozo puede

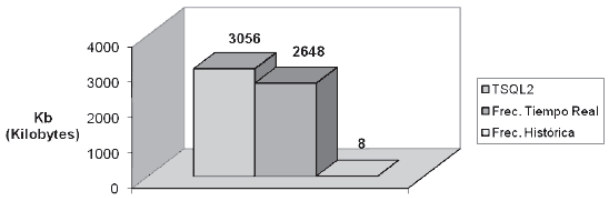

Fig. 21. Resultado del Experimento III

Tal y como se puede observar, los resultados arrojados por el Experimento III se mantienen invariantes con respecto a los obtenidos en el Experimento II. La única diferencia notable sería el número de datos que

se almacenan en la versión 3 de la tabla, en la cual se define una frecuencia de tipo histórica. Como resultado del Experimento II esta tabla almacenaba únicamente 2 registros, mientras que para este experimento son 43 el número de registros que se encuentran allí alojados. La explicación de por qué el tamaño de la tabla no varía a pesar de la variación en la cantidad de datos almacenados en ella, se debe a que el esquema físico utilizado por PostgreSQL se encuentra definido de manera tal que para almacenar datos en disco se reserven bloques de 8KB como medida mínima de almacenamiento.

Al igual que en los experimentos anteriores, para este experimento las hipótesis H1, H2 y H3 son satisfechas. Demostrando una vez más que los mecanismos de compresión han sido implementados con éxito y que actúan de la manera esperada.

d) Experimento IV: en busca de una posible variación en los resultados de los experimentos propuestos, se extiende el espacio de tiempo de prueba propuesto en el experimento anterior, de trés (3) a cinco (5) días.

A partir de este escenario, los resultados obtenidos en la inserción de datos en cada una de las 3 versiones de la tabla temp_pozo puede observarse en la Figura 22.

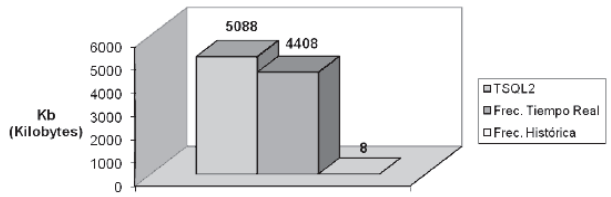

Fig. 22. Resultado del Experimento IV

Una vez más los resultados obtenidos demuestran que la implementación de los mecanismos de compresión satisfacen las hipótesis H1, H2 y H3. El consumo de espacio en disco resultado de este experimento es de 5088Kb para la tabla definida en TSQL2, 4408Kb para la tabla con frecuencia en tiempo real y 8Kb para la tabla con frecuencia histórica. La cantidad de registros almacenados es de 86399 registros para las tablas en TSQL2 y frecuencia en tiempo real y de 72 registros para la tabla con frecuencia histórica.

5.3. Resumen de Resultados

Como primer paso, se llevaron a cabo pruebas funcionales, las cuales permitieron determinar que efectivamente RT-PostgreSQL se encuentra en el tope de su funcionamiento, siendo capaz de llevar a cabo tareas de creación de tablas e inserción y selección de datos por medio de la propuesta de extensión de TSQL2 para el manejo de frecuencias temporales.

Otro de los resultados que se esperaba obtener por medio de la implementación de RT-PostgreSQL es la correcta definición de mecanismos internos de compresión, que fuesen capaces de optimizar los métodos de almacenamiento de datos propuestos por el lenguaje TSQL2. A partir de la definición de una serie de escenarios de prueba se pudo demostrar con un estudio experimental demostrar que se cumplió con el objetivo de integrar los mecanismos propuestos con el SGBD PostgreSQL de manera efectiva.

6. CONCLUSIONES

El desarrollo aquí descrito representa la primera integración en una misma arquitectura, de lo propuesto por Snoodgrass (1995), y Tineo y Gutiérrez (2006). A partir de esto, se concluye que RT-PostgreSQL:

- Hace de PostgreSQL el primer SGBD con la capacidad de definir y manipular datos basados en frecuencias y series temporales, todo esto en base a una arquitectura fuertemente acoplada.
- Representa un aporte significativo en el área de integración de paradigmas de bases de datos distintos al relaciona con los distintos SGBD existentes en el mercado.
- Demuestra que PostgreSQL es una de las herramientas más atractivas en lo que al desarrollo de este tipo de sistemas se refiere.
- Representa una solución factible para la gestión de tablas anidadas en PostgreSQL, gracias al método utilizado para la definición, inserción y selección de datos con frecuencias en series de tiempo.
- Desde el punto de vista del caso de estudio propuesto, la empresa PDVSA tendrá acceso a una herramienta desarrollada completamente en Software Libre y bajo estándares abiertos, la cual le permita tratar cada uno de los datos provenientes de sus procesos automatizados de una manera efectiva,

7. RECOMENDACIONES

- Continuar con el desarrollo de esta extensión, de manera que se puedan incluir distintas funcionalidades de definición y manipulación de datos con frecuencias temporales
- Dotar a RT-PostgreSQL de un mecanismo que permita exportar datos de tablas con frecuencias en tiempo real a tablas con frecuencias históricas,
- Aplicar el enfoque de extensión con arquitecturas fuertemente acopladas para satisfacer las necesidades correspondientes a distintos paradigmas de modelado de bases de datos.
- Definir un nombre formal para la propuesta hecha por Tineo y Gutiérrez (2006) para la extensión de TSQL2 para el manejo de frecuencias de tiempo. Esto permitiría adoptar un término que pueda considerarse como informáticamente correcto y que se adecúe, a su vez, a los estándares utilizados en el nombramiento de los distintos lenguajes de consultas de bases de datos que existen en la actualidad. Se propone para esto el nombre Time-Frequency Structured Query Language (TFSQL), Lenguaje de Consultas Estructuradas con Frecuencias Temporales.
- Continuar con los avances en esta propuesta de extensión del lenguaje TSQL2, de manera que se adopten en la misma distintas funcionalidades de los Lenguaje de Definición y Manipulación de Datos del estándar SQL (DDL y DML, respectivamente), tales como las operaciones ALTER y DROP en el caso del DDL, y las operaciones UPDATE y DELETE en el caso del DML.
- Se considera una necesidad imperiosa apoyar y fomentar el uso y desarrollo de Software Libre en pro de una evolución tecnológica económicamente sostenible y socialmente justa.

8. RECONOCIMIENTOS

Este desarrollo se llevó a cabo gracias al apoyo de la empresa Petróleos de Venezuela, S.A. bajo el N° 32001885.

9. REFERENCIAS

[1] Codd, E.F. (1970). “A Relational Model of Data for Large Shared Data Bases”, Communications of the ACM, 13, No. 6, 377-387.

[2] Jones, S., Mason, P. and Stamper, R.K. (1979). LEGOL 2.0: A relational specification language for complex rules, Information Systems, 4, No. 4, pp. 293-305.

[3] Snodgrass, R. (1987). “The Temporal Query Language TQuel”, ACM Transactions on Database Systems, 12, No. 2, pp. 247–298.

[4] Tansel, A. U., Arkun, M.E. (1986). “HQuel, A Query Language for Historical Relational Databases”, Proceedings of the Third International Workshop on Statistical and Scientific Databases.

[5] Ben-Zvi, J. (1982). “The Time Relational Model”, PhD. Dissertation, Computer Science Department, UCLA.

[6] Snodgrass, R.T. (1995). “The TSQL2 Temporal Query Language”, Kluwer Academic Publishers.

[7] Timarán, R. (2001). Arquitecturas de Integración del Proceso de Descubrimiento de Conocimiento con Sistemas de Gestión de bases de datos: un Estado del Arte, en revista Ingeniería y Competitividad, Universidad del Valle, Volumen 3, No. 2, Cali.

[8] Chacón, C. (2007). “Creación de un Sistema Manejador de Bases de Datos”, Tesis de grado, Universidad Simón Bolívar.

[9] Tineo, L. and Gutiérrez, L. (2006). “Extending TSQL2 for More Advanced Applications”. Sometido a IEEE Review.

[10] Timarán, R., Guerrero, M., Díaz, M., Cerquera, C., Armero, S. (2005). “Implementación de Primitivas SQL para Reglas de Asociación en una Arquitectura Fuertemente Acoplada”, en XXXI Conferencia Latinoamericana de Informática CLEI, Cali, Valle, Colombia.

10. OTRAS REFERENCIAS

• Arte, en revista Ingeniería y Competitividad, Universidad del Valle, Volumen 3, No. 2, Cali.