UNIVERSIDAD DE LOS ANDES
POSTGRADO DE COMPUTACIÓN

Desarrollo de un componente espacio temporal para un sistema manejador de bases de datos de código abierto.

Trabajo de Grado presentado como requisito para la obtención del grado de:

MAGISTER SCIENTIAE.

Autor: Ing. Derik A. Pinto E.

Tutora: Dra. Isabel Besembel.

Junio 2010

Resumen.

El propósito de la tesis es el desarrollo de un componente para la gestión de datos espacio-temporales para sistemas manejadores de bases de datos de código abierto el cual debe contener el manejo de dos aspectos fundamentales. El primero es la resolución del problema fundamental de base de datos y el segundo se basa en la recuperación de la data eficaz y eficientemente. Este trabajo ha elegido el manejador PostgreSQL para el desarrollo de dicho componente debido a la versatilidad y documentación que posee reutilizando el componente para manipulación de datos geográficos PostGIS y la capacidad de extender los mecanismos de indexación mediante el módulo GiST para PostgreSQL, Particularmente, se diseña y se elabora un prototipo del componente que lleva por nombre PosGist con un nivel de acoplamiento medio al manejador utilizando los lenguajes C y PLPGSQL para el desarrollo de las interfaces.

Contenido

Índice de Tablas

Índice de Figuras

CAPÍTULO I

Planteamiento del Problema

Introducción

En este capítulo se definirán los antecedentes que es la explicación de cómo se originó el interés por el problema, se realizará la definición del problema donde se establecerá de manera concreta la situación que se va a investigar, Se explicará de manera detallada la importancia de realizar la investigación, sus aportes y se definirán los objetivos, metodología, alcance y estructura del documento.

1.1Antecedentes

Isabel Besembel y Stuart Roberts 1998 en su artículo "***Indexación de Objetos Espacio-temporales"*** consideran que el árbol_R, es la estructura de datos que puede ser usada para indexar objetos espacio-temporales, por medio de las dimensiones espaciales y temporales que la aplicación necesita, sin efectuar ningún cambio en la estructura del método de acceso. Se tomó el enfoque orientado por objetos para implementar un índice árbol_R extendido como una clase denominada *RTree*. Esta clase soporta datos espaciales y temporales como su clave de indexación y extiende el árbol_R con un conjunto mínimo de operadores de búsqueda para soportar las búsquedas solamente espaciales, solamente temporales y las espacio-temporales. Encontramos definiciones tales como Índices,

Objetos Espacio-Temporales de igual manera se cita que los modelos de tiempo y espacio pueden ser discretos ó continuos. "en cualquiera de esas alternativas, dos objetos diferentes no pueden estar en el mismo punto en el espacio y en el tiempo", [8] por lo cual una clave espacio-temporal se considera una clave primaria. Para tratar tal clave se utiliza aquí el modelo continuo representado por intervalos que son datos unidimensionales compuestos de dos valores, inferior y superior. Este modelo permite la representación de punto e intervalos, ya que los puntos son intervalos de longitud cero. Se considera también, el mínimo rectángulo que cubre al objeto y las propiedades temporales de tiempo de vida en conjunto formando un rectángulo multidimensional denominado un hiper-rectángulo. La principal desventaja de esta consideración es el tratamiento homogéneo de las dimensiones espaciales y temporales. Finalmente, definen un conjunto mínimo de operadores con miras a minimizar el conjunto de operadores de la clase *RTree* extendida que allí se propone.

Según Paul M. Aoki (1991), en su Trabajo "***Implementación de índices extendidos en POSTGRES"***, plantea que extender los índices de acceso es especialmente útil cuando se usa con un sistema en el que nuevos tipos de datos y operadores pueden definirse, luego expone que el manejador de base de datos Postgres permite al usuario ampliar el sistema de muchas maneras diferentes. El usuario puede implementar fácilmente nuevos tipos de datos abstractos, funciones, operadores, y los métodos de acceso también, divide, las implementaciones en mayores que modifican internamente el manejador y deben ser compiladas y otras que pueden hacerse con el manejador en ejecución.

JF. Naughton y A. Pfeffer (1995) en el trabajo: "***Árboles generalizados de búsqueda para sistemas de bases de datos, para el VLD"***, plantea la opción GIST como una estructura de indexación que soporta un conjunto de consultas y tipos de datos extensibles implementando estructuras básicas como los árboles B+ y árboles R y propone al GIST como una estructura fácil de extender por medio de la implementación de 6 métodos fundamentales.

1.2 Definición del problema

En las bases de datos espacio-temporales existen dos objetivos fundamentales a resolver, uno de los cuales consiste en proveer métodos de acceso que permitan construir índices para datos de tipo espacio-temporal. En este punto existe variedad de métodos que se pueden clasificar en tres categorías: Métodos que tratan el tiempo como otra dimensión, métodos que incorporan la información temporal dentro de los nodos de la estructura sin considerar el tiempo como otra dimensión y métodos que usan sobreposición (*overlapping*) de la estructura con el objeto de representar el estado de la base de datos en diferentes instantes de tiempo. Como segundo objetivo, proporcionar con un conjunto mínimo de operadores para soportar las búsquedas solamente espaciales, solamente temporales y espacio-temporales. Estos operadores incluyen la búsqueda al compañero exacto y las extienden por rango permitiendo el uso de operadores más adecuados en una consulta particular. Ellos son los operadores topológicos: separación, contención, intersección y cobertura; los operadores direccionales: arriba, abajo, a la izquierda

y a la derecha; algunos operadores de proximidad: cerca, lejos y entre; y los operadores temporales: antes, después, durante y los de proximidad temporal.

El espacio y el tiempo son dos atributos propios de cualquier objeto del mundo real, un objeto espacio temporal es una abstracción de una entidad que tiene un identificador (clave primaria), una localización y una forma espacial, al menos una propiedad que varía en el tiempo y algunas otras características que lo describen. Un componente espacio-temporal introduce al usuario en el contexto de los objetos en movimiento por medio de la integración de las funcionalidades antes mencionadas.

Lo anterior ha derivado en la necesidad de estudiar, comparar y desarrollar un componente un componente espacio temporal para sistemas manejadores de bases de datos de código abierto.

1.3 Justificación

Actualmente, los principales proveedores de soluciones para sistemas manejadores de base de datos de código abierto cuentan con soportes espaciales básicos para tipos de datos geométricos y datos temporales por separado, y se ha adelantado un poco en los métodos de indexación pero aún no cuenta con un soporte para el desarrollo de bases de datos espacio-temporales que introduzca al usuario en el contexto de los objetos en movimiento y la resolución de consultas basadas en operadores propios de los objetos espacio temporales, las cuales sin una indexación

adecuada elevarían los tiempos de respuesta del manejador de base de datos. Dichas consultas implementadas por operadores definidos a la medida objetos cuyos acontecimientos modifican sus atributos a través del tiempo.

Las bases de datos espacio-temporales aportan dicha funcionalidad que es aplicable en hechos de interés cotidiano de organismos sin fines de lucro que, generalmente, no le es factible adquirir una solución comercial por los elevados costos de las mismas y con las cuales se pueden desarrollar aplicaciones tales como:

- Localización de vehículos.
- Comportamiento de las placas tectónicas.
- Tráfico aéreo
- Sistemas de plan urbanístico

1.4 Objetivos

A continuación se presentará la definición de los objetivos generales y específicos planteados durante la investigación previa a la ejecución del proyecto.

1.4.1 Objetivo General

- Esta investigación tiene como objetivo fundamental definir e implementar una propuesta de diseño para un componente espacio-temporal y su integración en un sistema manejador de bases de datos de código abierto contentivo de un componente que gestione la indexación adecuada de los datos.

1.4.2 Objetivos Específicos

- Desarrollo y/o Adaptación de los hiper-rectángulos y métodos de indexación para el soporte de los datos temporales en los componentes GIS para soporte de los hiper-rectángulos.
    - Selección del diseño del hiper-rectángulo.
    - Pruebas de concepto de implementación.
    - Especificación de implementación de hiper-rectángulos.
    - Implementación de los hiper -rectángulos.
    - Pruebas de la implementación de los hiper-rectángulos.
    - Selección del diseño de la estructura de soporte a la indexación.
    - Pruebas de concepto de implementación.
    - Especificación de implementación de árbol seleccionado.
    - Implementación de la estructura.
    - Integración de la estructura con los hiper –rectángulos.
    - Pruebas de la implementación de los hiper-rectángulos.
- Desarrollo y/o Redefinición de los operadores temporales, espaciales y espacio-temporales para que interactúen con los hiper-rectángulos definidos.
    - Integración de operadores Temporales sobre los tipos de dato espacio-temporales.

- Integración de operadores espaciales sobre los tipos de dato espacio-temporales.
- Implementación e Integración de Operadores espacio-temporales.
- Obtención y/o creación de datos de prueba para el componente integrado.
- Pruebas al componente espacio-temporal integrado

- Documentación de código, Redacción de Informe y API del componente.

1.5 Metodología

Según J. Barrios y J. Montilva (2009): El método *W_Watch* es la versión ligera del método *Watch*, es un marco metodológico que describe, el conjunto estructurado de actividades necesarias para producir un producto de software sencillo y pequeño con documentación precisa.

En esta versión se trata de disminuir la elaboración detallada de documentos y/o especificaciones que actúan como instrumentos de apoyo parcial al proceso de desarrollo, permitiendo, a un grupo de desarrollo pequeño (1 o 2 personas), dedicar más tiempo a actividades de implementación e implantación de versiones operativas y evolutivas del producto. Es por ello que las actividades gerenciales, de control de calidad y configuración, necesarias en todo proyecto de desarrollo, se limitan a prescribir las actividades esenciales de control de cambios, validación y verificación de especificaciones y productos. El rol de líder del proyecto puede ser

llevado en paralelo y sin sobrecarga durante la ejecución de otros roles técnicos del proyecto.

En el método *W_Watch* se describen una serie de procesos y subprocesos que generan productos por medio de actividades como lo son:

- Procesos Gerenciales
- Procesos de Desarrollo
  - Modelado de Negocio
  - Ingeniería de Requisitos
  - Diseño de Software
  - Aprovisionamiento de Componentes
  - Ensamblaje del Software
  - Pruebas del sistema

Es importante aclarar que los subprocesos del método *W_Watch* tiene un subconjunto de actividades y tareas enfocadas hacia implementación de lógicas de negocio empresariales y por la naturaleza basada en componentes del proyecto, algunas de las actividades y tareas planteadas como lo son el diseño de la base de datos, entre otras no serán especificadas en la presente investigación.

1.6 Alcance

En alcance de este proyecto se fundamenta en la implementación de un componente espacio-temporal para postgres-gis, que gestione las bases de datos abarcando la implementación e integración de estructuras de almacenamiento con la redefinición de los operadores mínimos temporal, espacial y espacio-temporales necesarios definidos en ellas.

1.7 Estructura del documento

Capítulo 1, Introducción. Se definen los antecedentes, problema, justificación, objetivos, metodología y el alcance del proyecto. Describiendo de donde y porque surge el problema, lo que intenta solucionar y la forma en que se desarrollara el proyecto.

Capítulo 2, Marco Teórico. Se describen los conceptos inherentes a la investigación desarrollada comenzando por la definición de las bases de datos espacio-temporales y su aplicación en el mundo real así como también se mostrará un panorama de las herramientas existentes. De igual manera, se describe el sistema manejador de base de datos seleccionado y los componentes adicionales empleados para el logro de la implementación.

Capítulo 3, Análisis y Diseño para la creación del componente espacio-temporal. Se describen una serie de productos generados en las iteraciones planteadas en el método de J. Barrios y J. Montilva denominado *W_Watch*.

Capítulo 4, Implementación del Prototipo. Se detalla la plataforma en la cual se realizó el componente y se presenta la adecuación de los componentes en las iteraciones desempeñadas dentro de las cuales se relata las razones de una iteración que no resultó factible así como también la implementación del prototipo denominado Posgist con sus características más resaltantes.

Capítulo 5, Pruebas y Análisis de Resultados. Se describen las pruebas de caja negra, caja blanca e integración del prototipo Posgist por medio de la utilización de una interfaz de comunicación con postgres denominada Pgadmin III.

Capítulo 6, Conclusiones y Recomendaciones.

CAPÍTULO II

Marco Teórico

En este capítulo se describen los conceptos inherentes a la investigación desarrollada comenzando por la definición de las bases de datos espacio-temporales y su aplicación en el mundo real según herramientas conceptuales y aplicadas existentes. De igual manera se describe el sistema manejador de base de datos seleccionado y los componentes adicionales empleados para el logro de la implementación.

2.1 Bases de Datos espacio-temporales

Una base de datos espacio-temporal es un sistema que gestiona el espacio y el tiempo de los datos allí almacenados. Básicamente es una generalización de las bases de datos espaciales donde las coordenadas o algunas características de los datos varían con el tiempo. Se trata de geometría cambiando con el tiempo o localización de objetos en movimiento de geometría invariante [5]. Actualmente, las bases de datos espacio-temporales en conjunto representan un tópico de estudio reciente, de mucho interés debido a la estrecha relación existente entre espacio y tiempo en una gran cantidad de aplicaciones en donde se debe modelar datos con componentes tanto espaciales como temporales [6]. Se pueden encontrar ejemplos claros en áreas como las de transporte (vigilancia de tráfico), Ciencias Sociales (demografía), telecomunicaciones (telefonía celular), multimedia (películas animadas) e información geográfica (cambios de límites en terrenos), entre otras [6].

2.2 Objetos espacio-temporales

Un objeto espacio temporal es una abstracción de una entidad que tiene un identificador (clave primaria), una localización y una forma espacial, al menos una propiedad que varía en el tiempo y algunas otras características que lo describen [1].

2.3 Relaciones espacio-temporales

Según *Besembel Carrera y Montilva Calderón* [7], Dentro de las relaciones espacio-temporales se utilizan las relaciones topológicas (separación, contenido, solapamiento y cobertura), las direccionales (arriba o norte, abajo o sur, izquierda o este y derecha u oeste) y las de proximidad (cerca, lejos y entre). Las relaciones topológicas han sido tratadas y descritas por M. Egenhofer. La tabla 2.1 presenta estas relaciones utilizando la notación orienta por objetos que representa la relación asociada, i. e. Si **k.separado(m) = cierto**, entonces los objetos **k** y **m** están separados. El símbolo **b** indica el borde del objeto espacial e **i** indica su interior. Cada una de las columnas siguientes describe el resultado de la intersección catalogado como: vacío (Ø) o no vacío (¬Ø). Las intersecciones pueden ser entre sus bordes (b∩b), entre sus interiores (i∩i) y sus combinaciones.

Operador	b∩b	i∩i	b∩i	i∩b

k.separado(m)	Ø	Ø	Ø	Ø
k.contenidoEn(	Ø	¬Ø	Ø	¬Ø
k.dentro(m)	Ø	¬Ø	¬Ø	Ø
k.toca(m)	¬Ø	Ø	Ø	Ø
k.igual(m)	¬Ø	¬Ø	Ø	Ø
k.cubre(m)	¬Ø	¬Ø	Ø	¬Ø
k.cubiertoPor(	¬Ø	¬Ø	¬Ø	Ø
k.solapa(m)	¬Ø	¬Ø	¬Ø	¬Ø

Tabla 2.1. Relaciones topológicas entre dos objetos k y m.

Las relaciones direccionales o de posición fueron presentadas por W. Kim, J. Garza, y A. Keskin. Este conjunto de relaciones se muestran en la tabla 2.2 donde **mbr** es el mínimo rectángulo que cubre el objeto, el cual se considera como **[xi, xs, yi, ys]** y cada componente del **mbr** está referida como **mbr.componente**. Un objeto espacial **k** está arriba de otro objeto espacial **m** si el **mbr** de **k** intercepta el **mbr** definido por [- ∞, ∞ , m.ys, ∞]. Cálculos similares se hacen para el resto de los operadores.

Operador	**Relación**
k.arriba(m)	k.yi m.ys
k.abajo(m)	k.xi m.xs
k.izquierda(m)	k.ys m.yi
k.derecha(m)	k.xs m.xi
k.dirArriba(m)	k.arriba(m) ^ m.x.cubre(k.x)
k.dirAbajo(m)	k.abajo(m) ^ m.x.cubre(k.x)
k.dirIzquierda(m)	k.izquierda(m) ^ m.y.cubre(k.y)
k.dirDerecha(m)	k.derecha(m)
k.arribaIzquierda(m)	k.arriba(m) ^ k.izquierda(m)

k.arribaDerecha(m)	k.arriba(m) ^ k.derecha(m)
k.abajolzquierda(m)	k.abajo(m) ^ k.izquierda(m)
k.abajoDerecha(m)	k.abajo(m) ^ k.derecha(m)

Tabla 2.2. Relaciones direccionales entre objetos espaciales.

Las relaciones de proximidad se mencionan también. De las allí mencionadas se escogieron tres, a saber: cerca, lejos y entre. La tabla 2.3 presenta estos operadores y sus relaciones con los operadores topológicos para un valor dado denominado delta (Δ), el cual indica la unidad especificada para calcular si un **mbr** solapa objetos cerca del objeto buscado.

Operador	Relación	P
k.cerca(m)	k.solapa(p)	(xim-Δ , xsm + Δ , yim -Δ , ysm + Δ)
k.lejos(m)	k.separado(p)	
k.entre(m,	k.solapa(p)	(Min(xim, xij), Max(xsm, xsj), Min(yim, yij),

Tabla 2.3. Relaciones de proximidad entre objetos espaciales.

Las relaciones temporales son mostradas en la tabla 4. Para los operadores: adyacente, sigue y precede se necesita la especificación de un borde para diferenciar entre el borde inferior (**bi**) y el superior (**bs**), y además una unidad de tiempo (TU) para evaluar la proximidad en tiempo como la utilizada en las relaciones espaciales. En esta tabla **q** y **t** son dos objetos temporales.

Operador	b∩	i∩i	b∩i	i∩b	(bi, bs)
q.antes(t)	Ø	Ø	Ø	Ø	q.b < t.b
q.despues(t)	Ø	Ø	Ø	Ø	q.b > t.b
q.durante(t)	Ø	¬Ø	¬Ø	Ø	
q.solapa(t)	Ø	¬Ø	¬Ø	¬Ø	
q.toca(t)	¬Ø	Ø	Ø	Ø	
q.igual(t)	¬Ø	¬Ø	Ø	Ø	
q.comienza(t)	¬Ø	¬Ø	Ø	¬Ø	
q.termina(t)	¬Ø	¬Ø	¬Ø	Ø	
q.adyacente(t)	Ø	Ø	Ø	Ø	(t.bi – q.bs = TU) (q.bi – t.bs
q.sigue(t)	Ø	Ø	Ø	Ø	q.bi – t.bs = TU
q.precede(t)	Ø	Ø	Ø	Ø	t.bi – q.bs = TU

Tabla 2.4. Relaciones temporales entre objetos.

Como relaciones espacio-temporales se consideran solamente las topológicas entre los rectángulos en d- dimensiones llamados aquí **hiper-rectángulos**, donde las dimensiones espaciales y temporales se tratan homogéneamente. Una dimensión por cada dimensión espacial, X e Y si se tienen objetos espaciales en 2D y una dimensión por cada dimensión temporal, tiempo válido si se tienen objetos temporales en 1D. Esto es posible porque se considera cada representación de las d-dimensiones como intervalos, así un hiper-rectángulo está compuesto de **d** intervalos, uno por cada dimensión.

En general es posible combinar las relaciones espaciales y temporales para ser usadas para responder preguntas o consultas espacio temporales, pero sólo se consideran aquí las relaciones topológicas en dD [7].

2.4 Índices de acceso

Son un conjunto de asociaciones entre una clave de indexación y la localización del dato que contiene dicha clave. Un índice permite la recuperación de objetos almacenados en memoria secundaria a través de la clave definida que es un atributo del objeto. Esta clave puede ser un único atributo que identifica unívocamente al objeto, o varios atributos no concatenados, donde cada uno de ellos es una clave secundaria [8].

2.5 Indexación de objetos espacio-temporales

El objeto espacio-temporal y los índices de acceso se puede que el problema de la indexación en dichos objetos se debe a que los datos espaciales y los datos temporales deben ser tratados homogéneamente para resolver las consultas que conllevan a resultados aplicando las relaciones espacio-temporales mencionadas anteriormente entonces la o las estructuras que mantiene el índice en memoria principal debe estar preparada para asumir dicho trato homogéneo extendido.

Existen varios métodos de acceso espacio-temporales que son adecuados para aplicaciones que consideran cambios espaciales de manera discreta. Una clasificación de los dichos cambios es la siguiente:

- Métodos que tratan el tiempo como otra dimensión.

- Métodos que incorporan la información temporal dentro de los nodos de la estructura sin considerar el tiempo como otra dimensión.
- Métodos que usan sobreposición de la estructura de datos subyacente.
- Métodos basados en versiones de la estructura.
- Métodos orientados a la trayectoria.[9]

2.6 Aplicaciones

Se pueden encontrar ejemplos claros en áreas como las de transporte (vigilancia de tráfico), Ciencias Sociales (demografía), telecomunicaciones (telefonía celular), multimedia (películas animadas) e información geográfica (cambios de límites en terrenos) [9].

2.7 Consultas espacio-temporales

Instante de tiempo o *TimeSlice*: consiste en: dado un instante de tiempo *ti*, y una ventana sobre el plano, informar todos los objetos que están o pasaron por ella durante ese instante de tiempo [10].

• Intervalo de tiempo: consiste en, dados dos instantes de tiempo *ti* y *tj*, y una ventana sobre el plano, informar todos los objetos que están o pasaron por ella durante ese intervalo de tiempo.

• Eventos: dado un instante de tiempo *ti*, y una ventana sobre el plano, informar los objetos que aparecieron/desaparecieron en ella durante ese instante de tiempo.

Trayectoria: dado un objeto y un intervalo de tiempo, informar las posiciones que ocupó dicho objeto durante ese intervalo de tiempo.

• Ensamble (*Join*): Este tipo de consulta consiste en: dados dos conjuntos de datos, recuperar todos los pares de objetos que satisfacen un predicado específico del tipo de dato que se trate (espacial, temporal, espacio-temporal). Existe una amplia variedad de predicados; sin embargo, el predicado de intersección tiene un rol muy importante ya que a partir de él se pueden obtener los restantes. Un ejemplo puede ser, "hallar todos los pares de objetos que se hallan espacialmente cerca, es decir, dentro de una distancia D, durante un intervalo o instante de tiempo específico". Otra aplicación inmediata es la detección de accidentes comparando la trayectoria de los vehículos: "hallar los pares de vehículos que estarán cerca, en una radio de 10 Km., dentro de 5 minutos".

Cercanía o proximidad: k-vecinos más cercanos: Dado un objeto X, encontrar los k objetos X' que tienen la mínima distancia de X. Aquí, la distancia se define entre atributos espaciales como la distancia (Euclidiana o de Manhattan) entre sus puntos más cercanos. Un ejemplo para este tipo de consulta puede ser: "hallar las 5 ambulancias más cercanas, con respecto al lugar del accidente"; y uno más complejo sería: "hallar las 5 ambulancias más cercanas con respecto al lugar del accidente en un intervalo de tiempo de 2 minutos antes y después del accidente,

sabiendo las direcciones y velocidades de las ambulancias y el mapa de las calles". Existen cuatro escenarios posibles teniendo en cuenta si el objeto X y el conjunto S de objetos de la base de datos se mueven o no: (1) no se mueve ni el objeto X ni los objetos de S; (2) se mueven los objetos del conjunto S pero no el objeto X; (3) se mueve el objeto X pero no los del conjunto S; y (4) tanto el objeto X como los del conjunto S se mueven.

• El par de vecinos más cercano: Dado un conjunto de objetos S, encontrar dos objetos cuya distancia mutua sea mínima. Un ejemplo típico en el control de tráfico aéreo sería "encontrar el par de aviones más cercano" ya que se determinaría dónde existe peligro de colisión [10].

Patrones espacio-temporales: Un patrón espacio-temporal se representa como una secuencia de distintos predicados espacio-temporales, donde interesa el ordenamiento temporal (exacto o relativo) de los predicados. Este tipo de consulta consiste en, dada una gran colección de trayectorias espacio-temporales y un patrón, recuperar todos los objetos que siguen ese patrón de movimiento definido en tiempo y espacio. Un ejemplo sería, "identificar todos los vehículos que estuvieron muy cercanos a los tres ataques de francotiradores en Maryland (las posiciones y tiempos de los ataques son conocidos)" o "localizar los productos que abandonaron la fábrica hace un mes, fueron almacenados en uno de los almacenes cerca del muelle y fueron cargados en un barco". Este tipo de consulta puede tener tipos arbitrarios de predicados espaciales (es decir, búsqueda por rango, vecino más cercano, etc.), donde cada predicado puede ser asociado con: (1) una restricción exacta temporal que es o un instante o un intervalo de tiempo (consulta

de patrones espacio-temporales con tiempo), o (2) generalmente, con un orden relativo (consulta de patrones espacio-temporales con orden) [10].

Por ejemplo, en un sistema de vigilancia satelital de una empresa de transportes puede ser interesante conocer la ubicación de las unidades a lo largo del tiempo. Así la información espacial se amplía con conceptos temporales. Para una aplicación como ésta resulta fundamental poder responder consultas con predicados espaciales y temporales, tales como: ¿Dónde se encuentra la unidad nº 12 ahora? haciendo preciso el instante ahora; o ¿Qué unidades estaban más cercanas a la intersección de las rutas A y B ayer a las 14:00 hrs? también haciendo precisos los conceptos más cercanas y ayer. Otras podrían ser ¿Cuál fue el recorrido de la unidad nº 8 en la última semana?, ¿Cuáles unidades se encontraban en la ruta A entre 04:00 hrs. y las 08:00 hrs. el día 14/02/05?, ¿Cuáles unidades llegaron y cuáles salieron en la zona Z a las 20:00 hrs?, entre otras [10].

2.8 Árbol R

Un árbol_R o *R_tree* es un árbol balanceado en altura cuyos nodos hojas contienen referencias a los objetos. El índice es completamente dinámico, donde las inserciones y las eliminaciones se pueden entremezclar con las búsquedas sin necesidad de reorganizaciones periódicas.

Un *R-tree* es una extensión natural de un *B-tree* para objetos espaciales (puntos y regiones) [11]. Cada nodo corresponde a una página o bloque de disco.

Los nodos hojas de un *R-tree* contienen entradas de la forma h<M*BR*,*oid*> donde *oid* es el identificador del objeto espacial en la base de datos y MBR (*Minimun Bounding Rectangle*) es un rectángulo multidimensional que corresponde al mínimo rectángulo que encierra al objeto espacial. Los nodos rama (nodos no-hojas) contienen entradas de la forma h<*MBR*, *ref*>, donde *re f* es la dirección del correspondiente nodo hijo en el *R-tree* y MBR es el rectángulo mínimo que contiene a todos los rectángulos definidos en las entradas del nodo hijo.

Sea *M* el número máximo de entradas que se pueden almacenar en un nodo y sea $m \leq M/2$ un parámetro especificando el número mínimo de entradas en un nodo. Un *R-tree* satisface las siguientes propiedades [11]:

1. Cada nodo contiene entre *m* y *M* entradas a menos que corresponda a la raíz.

2. Para cada entrada h<*MBR*,*oid*> en un nodo hoja, MBR es el mínimo rectángulo que contiene (espacialmente) al objeto.

3. Cada nodo rama tiene entre *m* y *M* hijos, a menos que sea la raíz.

4. Para cada entrada de la forma h<*MBR*, *re f*> de un nodo interno, MBR es el rectángulo más pequeño que contiene espacialmente los rectángulos definidos en el nodo hijo.

5. El nodo raíz tiene al menos dos hijos, a menos que sea una hoja.

6. Todas las hojas se encuentran al mismo nivel.

La figura 2.1 muestra para los mínimos rectángulos abarcadores como quedarían en un árbol_R Con m = 2 y M = 3

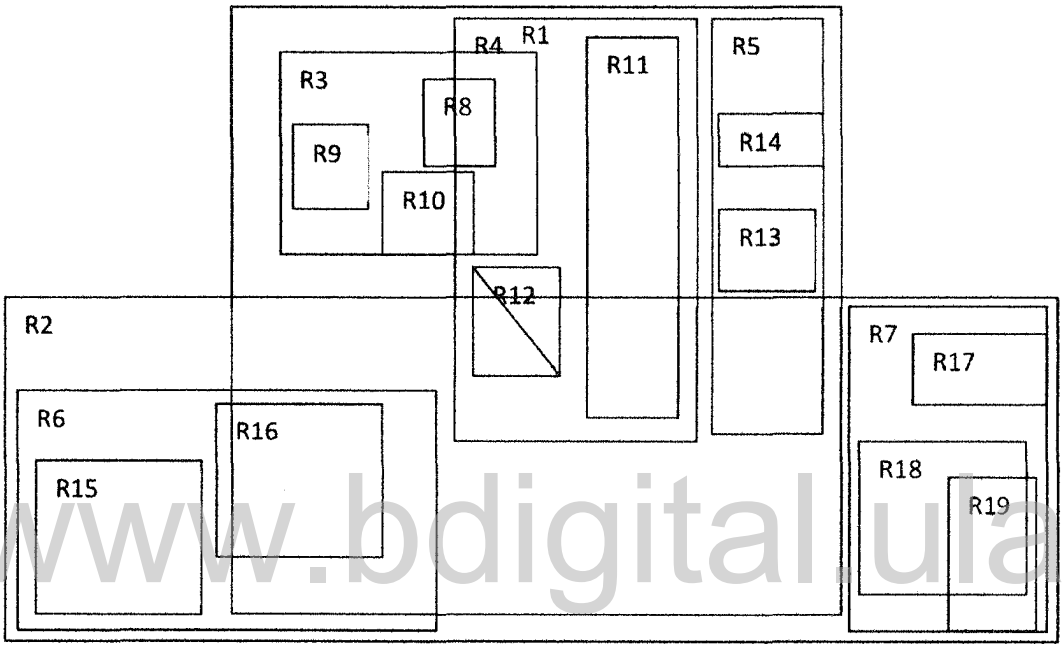


Figura 2.1: Agrupaciones de MBRs generadas por un *R-tree*.

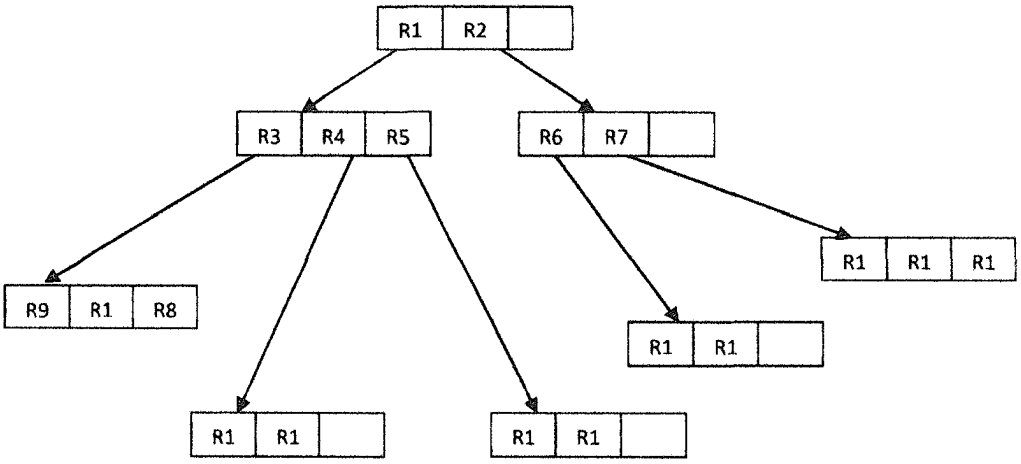


Figura 2.2: *R-tree* MBR de la figura 2.1

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional (ORDBMS) basado en el proyecto POSTGRES, de la universidad de Berkeley. Entre sus características se destacan:

Alta concurrencia

Mediante un sistema denominado MVCC (Acceso concurrente multiversión, por sus siglas en inglés) PostgreSQL permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Cada usuario obtiene una visión consistente de lo último a lo que se le hizo *commit,* es decir, cuando la transacción fue exitosa. Esta estrategia es superior al uso de bloqueos por tabla o por filas común en otras bases, eliminando la necesidad del uso de bloqueos explícitos.

Amplia variedad de tipos nativos

PostgreSQL provee nativamente soporte para:

- Números de precisión arbitraria.
- Texto de largo ilimitado.
- Figuras geométricas (con una variedad de funciones asociadas)
- Direcciones IP (IPv4 e Ipv6).
- Bloques de direcciones estilo CIDR.
- Direcciones MAC.
- *Arrays.*

Adicionalmente los usuarios pueden crear sus propios tipos de datos, los que pueden ser por completo indexables gracias a la infraestructura *GiST* de

PostgreSQL. Algunos ejemplos son los tipos de datos GIS creados por el proyecto PostGIS.

Otras características

Claves ajenas también denominadas llaves ajenas o claves foráneas (*foreign keys*).

- Disparadores (*triggers*): Un disparador o *trigger* se define en una acción específica basada en algo ocurrente dentro de la base de datos. En *PostgreSQL* esto significa la ejecución de un procedimiento almacenado basado en una determinada acción sobre una tabla específica. Ahora todos los disparadores se definen por seis características:
  - El nombre del disparador o *trigger*
  - El momento en que el disparador debe arrancar
  - El evento del disparador deberá activarse sobre la tabla donde el disparador se activará
  - La frecuencia de la ejecución
  - La función que podría ser llamada

Entonces combinando estas seis características, PostgreSQL le permitirá crear una amplia funcionalidad a través de su sistema de activación de disparadores (*triggers*).

- Vistas.
- Integridad transaccional.
- Herencia de tablas.
- Tipos de datos y operaciones geométricas.

Funciones

Bloques de código que se ejecutan en el servidor. Pueden ser escritos en varios lenguajes, con la potencia que cada uno de ellos da, desde las operaciones básicas de programación, tales como bifurcaciones y bucles, hasta las complejidades de la programación orientada a objetos o la programación funcional.

Los disparadores (*triggers* en inglés) son funciones enlazadas a operaciones sobre los datos.

Algunos de los lenguajes que se pueden usar son los siguientes:

- Un lenguaje propio llamado PL/PgSQL (similar al PL/SQL de oracle).
- C.
- C++.
- Java PL/Java web.
- PL/Perl.
- plPHP.
- PL/Python.
- PL/Ruby.
- PL/sh.
- PL/Tcl.
- PL/Scheme.
- Lenguaje para aplicaciones estadísticas R por medio de PL/R.

PostgreSQL soporta funciones que retornan "filas", donde la salida puede tratarse como un conjunto de valores que pueden ser tratados igual a una fila retornada por una consulta (*query* en inglés).

Las funciones pueden ser definidas para ejecutarse con los derechos del usuario ejecutor o con los derechos de un usuario previamente definido. El concepto de funciones, en otros DBMS, son muchas veces referidas como "procedimientos almacenados" (*stored procedures* en inglés) [13].

2.9 PostGis

PostGIS es un módulo que añade soporte de objetos geográficos a la base de datos objeto-relacional PostgreSQL, convirtiéndola en una base de datos espacial para su utilización en Sistema de Información Geográfica. Se publica bajo la licencia pública general de GNU. PostGIS agrega soporte para objetos geográficos al objeto de PostgreSQL la base de datos relacional. En efecto, PostGIS "habilita espacialmente" el servidor PostgreSQL, lo cual le permite ser utilizado como una base de datos espacial de fondo para sistemas de información geográfica (SIG). Muy similar a la SDE de ESRI o la extensión espacial de Oracle. PostGIS sigue la "*OpenGIS Simple Features Specification for SQL*" y ha sido certificado como compatible con los "tipos y funciones" del perfil [14].

PostGIS implementa las siguientes características:

- Tipos de datos geométricos para *points, linestrings, polygons, multipoints, multilinestrings, multipolygons y geometrycollections.*
- Predicados espaciales para determinar las interacciones de geometrías utilizando matriz 3x3.

- Operadores espaciales para determinar las medidas geoespaciales como el área, distancia, longitud y perímetro.
- Operadores espaciales para determinar el conjunto de operaciones geoespaciales, como la unión, diferencia, diferencia simétrica y buffer.
- Índices espaciales de árboles R para rápidas consultas espaciales.
- Soporte para índices selectivos, para proporcionar alto desempeño en la planeación de consultas para consultas mezcladas espaciales/no espaciales.

La implementación está basada en geometrías e índices "ligeros" optimizados para reducir el uso de disco y memoria. Utilizando geometrías ligeras ayuda al servidor a incrementar la cantidad de datos migrados desde el almacenamiento físico en el disco hacia la RAM, mejorando el desempeño de las consultas sustancialmente [14].

2.10 Tipos de datos espaciales y forma de almacenamiento

La representación *Well-Known Binary* (WKB) de valores geométricos está definida por la especificación *OpenGIS*. También está definida en el estándar ISO "SQL/MM Part 3: Spatial".

WKB se utiliza para intercambiar datos como cadenas binarias representadas por valores **BLOB** que contienen información geométrica WKB.

WKB utiliza enteros sin signo de un byte, enteros sin signo de cuatro bytes, y números de ocho bytes de doble precisión (formato IEEE 754). Donde un byte es una secuencia, son ocho bits.

Por ejemplo, un valor WKB que corresponde a un **POINT(1 1)** consiste en esta secuencia de 21 bytes (cada uno representado aquí por dos dígitos hexadecimales):

0101000000000000000F03F000000000000F03F

La secuencia puede descomponerse en los siguientes componentes:

Orden de byte: 01

Tipo WKB : 01000000

X : 000000000000F03F

Y : 000000000000F03F

La representación de componentes es como sigue:

- El orden de byte puede ser 0 ó 1, para indicar almacenamiento tipo *little-endian* o *big-endian*. Los órdenes de byte *little-endian* y *big-endian* son también conocidos como Representación de satos de red (*Network Data Representation* (NDR)) y Representación Externa de Datos (*External Data Representation* (XDR)), respectivamente.
- El tipo WKB es un código que indica el tipo de geometría.
  Los valores del 1 al 7 significan: *Point, LineString, Polygon, MultilineString MultiPolygon y GeometriCollection*.
- Un valor ***Point*** tiene coordenadas X e Y, cada una representada por un valor de doble precisión.

Los valores WKB que representan valores geométricos más complejos son representados por estructuras de datos más complejas, tal como se detalla en la especificación *OpenGIS*. [15]

2.11 Base de datos y tipos de datos temporales

Una base de datos temporal es un sistema de gestión de base de datos (DBMS) el cual implementa y trata con especial énfasis aspectos temporales, teniendo un modelo de datos temporal y una versión temporal del lenguaje de consulta estructurado, (SQL). Entre las diversas propuestas de implementación, la más extendida es TSQL2

Especificando más profundamente, los aspectos temporales normalmente incluyen tiempo de validez y tiempo de transacción. La combinación de estos dos atributos forman un dato bi-temporal.

Tiempo de validez indica el periodo de tiempo en el cual un hecho es verdad en el mundo real.

Tiempo de transacción indica el periodo de tiempo en el cual un hecho está guardado en la base de datos.

Dato Bi-temporal es la combinación del tiempo de validez y el tiempo transaccional.

Existen un conjunto de relaciones temporales que se muestran a continuación en la figura 2.3 las cuales son necesarias en las bases de datos temporales cuando se debe considerar el tiempo de existencia [16].

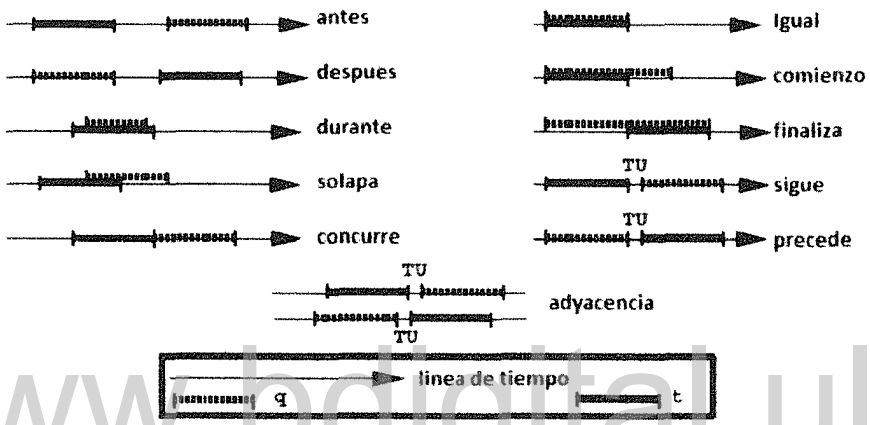


Figura 2.3 Relaciones temporales

2.12 Implementación de índices extendidos en Postgres GiST

GiST o árbol de búsqueda generalizada, es una estructura de datos con un API que puede ser utilizado para construir una variedad de búsquedas basadas en disco. GIST es una generalización del árbol B +, ofreciendo una estructura concurrente y recuperable de árbol de búsqueda balanceado en altura, sin hacer ninguna suposición sobre el tipo de dato almacenado, o las consultas que se atienden. GiST se puede utilizar para implementar fácilmente una serie de índices conocidos, incluidos los árboles B +, arboles R, árboles-HB, árboles-RD y otros,sino que

también permite el desarrollo fácil de índices especializados para nuevos tipos de datos. GiST puede ser utilizado para cualquier tipo de dato que puede ser naturalmente ordenado en una jerarquía de superseries. No sólo es extensible en términos de soporte de tipos de dato y de diseño de árboles, permite al que lo extiende apoyar cualquier consulta de predicados que elija.

GIST es un ejemplo de la extensibilidad del software en el contexto de los sistemas de base de datos: permite fácilmente la evolución de un sistema de base de datos para soportar nuevos índices basados en árboles. Esto se logra modificando su estructura básica, usando un pequeño API que es suficiente para captar la aplicación de los aspectos específicos de una amplia variedad de diseños de índice. El código de infraestructura *GiST* gestiona el diseño de las páginas de índice en el disco, los algoritmos de búsqueda y eliminación de los índices, detalles de transacciones complejas, tales como el bloqueo de página de alta concurrencia y registro de escritura anticipada para la recuperación de fallas. Esto permite a los autores del nuevo índice basado en árboles centrarse en la implementación de las novedades del nuevo tipo de índice - por ejemplo, la forma en que los subconjuntos de datos deben ser descritos para la búsqueda - sin llegar a ser expertos en el sistema de bases de datos internas.

Aunque originalmente diseñado para responder a las preguntas de selección de Boole, *GIST* también puede apoyar las formas de búsqueda de vecino más cercano, y varias aproximaciones estadísticas sobre los grandes conjuntos de datos.

La implementación GiST de PostgreSQL incluye soporte para claves de longitud variable, claves compuestas, control de concurrencia y recuperación, estas características son heredadas por todas las extensiones de GIST.

Hay varios módulos contribuidos desarrollados utilizando GIST y distribuido con PostgreSQL. Por ejemplo:

* rtree_gist, btree_gist - implementación GiST de R-Tree y B-Tree

* intArray - índice de apoyo para una matriz unidimensional de int4

* tsearch2 - una búsqueda (texto completo) tipo de datos con acceso indexado.

* ltree - tipos de datos, métodos de acceso y las consultas de datos organizados como una estructura tree-like.

*hstore - un dispositivo de almacenamiento de (clave, valor) de datos

*cube, en representación de cubos multidimensionales

La implementación GiST de PostgreSQL proporciona el soporte para la indexación de PostGIS (sistema de información geográfica) y el sistema de bioinformática BioPostgres.

CAPÍTULO III

Análisis y diseño para la creación del componente espacio-temporal

En este capítulo se describen una serie de productos generados en las iteraciones planteadas en el método de *J. Barrios y J. Montilva* W_Watch. El modelo de procesos de desarrollo se iniciaría directamente en la fase de Ingeniería de Requisitos debido a que se trata de una extensión para el manejo espacio-temporal del SMBD *PostgreSql*. Se realizó 2 iteraciones sobre procesos de diseño y aprovisionamiento de componentes para completar un ciclo que da como resultado el componente. En la primera iteración se obtiene como producto el diseño ideal conceptualizado en abstracto del componente, generando los requerimientos estimados de los componentes de los componentes que se ameritan en la fase de aprovisionamiento, seguidamente se realizó una segunda iteración en la cual se obtiene un diseño y especificación más definitiva del componente para utilizarla de insumo en la fase de implementación y ensamblaje del software que será descrita en el capítulo IV.

3.1 Primera Iteración en el proceso de diseño del método W_Watch

En el primer ciclo del método se realiza la ingeniería de requisitos para obtener como producto el diseño abstracto del componente verificando que los requerimientos y las interfaces sean cumplidas.

3.1.1 Ingeniería de Requisitos

Se plantea el desarrollo de un componente que gestione datos espacio-temporales para un sistema manejador de base de datos de código abierto que proporcione interfaces que resuelvan el problema fundamental de base de datos en objetos de tipo espacio-temporal, los cuales poseen un factor de riesgo elevado en lo que concierne a la recuperación de datos, y esto se debe a la forma de almacenamiento tradicional de los datos espaciales en sistemas manejadores de base de datos de código abierto actuales, que gestionan la memoria secundaria por medio de dos partes puntualizadas a continuación:

- Problema fundamental de base de datos a nivel de usuario.
- Problema de indexación de los datos espacio-temporales para mejorar la recuperación de los mimos.

3.1.1.1 Análisis de Dominio

El dominio de la solución propuesta se encuentra enmarcado en el de los componentes de apoyo a los sistemas manejadores de bases de datos que soportan el almacenamiento de tipos de datos espacio-temporales. Los datos procesados por el componente serán guardados en el sistema manejador de base de datos con la intensión de lograr el acceso a los mismos de manera adecuada.

3.1.1.2 Objetivos del componente espacio-temporal

- Aportar una generalización de las bases de datos espaciales donde se incluya el tiempo de existencia de los objetos.
- Indexar los objetos espacio-temporales.
- Proveer los operadores espacio-temporales mínimos necesarios para la realización de consultas.

3.1.1.3 Análisis de requisitos

En este punto se clasificaran los requisitos incluyendo un resumen del documento de definición y especificación de requisitos en el cual existe la particularidad de prioridad absoluta de los requisitos y la factibilidad de los mismos se encuentra enmarcada en el logro de este proyecto.

3.1.1.4 Clasificación y especificación de los requisitos

En esta sección se definen informalmente los requerimientos funcionales y no funcionales que debe satisfacer el componente a implementar. Los requerimientos funcionales tienen que ver con las operaciones que realizará el componente desde la perspectiva del usuario, mientras que los componentes no funcionales están asociados a las características que tendrá la aplicación.

3.1.1.5 Requisitos funcionales

La tabla 3.1 muestra e identifica los requisitos funcionales de la aplicación.

Ref #	Función	Categoría
R1.1	Crear columnas con objetos de tipo espacio-temporales en la base de datos.	Evidente
R1.2	Crear Índices de acceso para los objetos espacio-temporales.	Evidente
R1.3	Insertar en el SMBD objetos espacio-temporales	Evidente
R1.4	Modificar en el SMBD objetos espacio-temporales.	Evidente
R1.5	Eliminar Objetos espacio-temporales del SMBD	Evidente
R1.6	Ejecutar consultas de los datos espaciales por medio de operadores espaciales.	Evidente
R1.7	Ejecutar consultas de los datos temporales por medio de operadores temporales.	Evidente
R1.8	Ejecutar consultas de los tipos de datos espacio-temporales por medio de operadores particulares para el tipo de dato.	Evidente

Tabla 3.1 Funciones que debe ofrecer el componente

3.1.1.6 Requisitos no funcionales

La tabla 3.2 define informalmente los requisitos no funcionales de la aplicación.

Atributo	Detalles y Restricciones
Metáfora de Interfaz	Se provee una interfaz de interacción para el usuario con el manejador utilizando el lenguaje SQL con algunas extensiones para dar soporte a los operadores particulares de los objetos espaciales, temporales y espacio-temporales. Las interfaces de usuario a nivel de despliegue por ejemplo mapas e imágenes en general no serán objeto del componente sin embargo se brindará acceso independiente a los atributos espaciales para que puedan ser desplegados.
Lenguajes de Programación	PLPGSQL, C++, SQL.
Sistema Manejador de Base de Datos	PostgreSql
Sistema Operativo	Linux.

Tabla 3.2 Atributos del sistema.

3.1.2 Especificación de los requisitos

En esta sección se presentan los requisitos según el método W_Watch de manera técnica. En una primera parte se presentan los casos de uso de alto nivel que describen en forma narrativa los procesos del dominio y los diagramas de interacción para la asignación de las responsabilidades del componente.

3.1.2.1 Casos de Uso

El diagrama de casos de uso de la figura 3.1, describe los procesos para cumplir con el problema fundamental de base de datos donde se debe proveer una interfaz para creación de los tipos de datos espacio-temporales para que el usuario logre insertar, modificar y eliminar un tipo de dato compuesto por una parte espacial y otra temporal. Este diagrama presenta el único actor denominado como Usuario, que es la persona que interactúa con el componente.

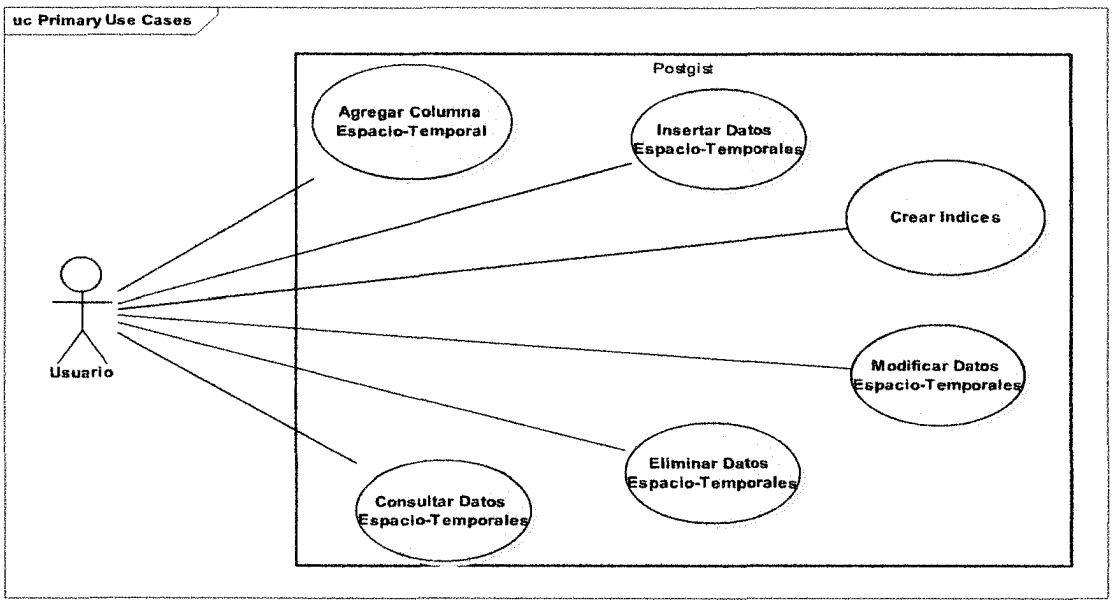


Figura 3.1 Diagrama de casos de uso

Las tablas 3.3, 3.4, 3.5, 3.6 explican cada caso de uso de los procesos planteados en el diagrama de la figura 3.1.

Caso de uso	**Agregar columna espacio-temporal**
Actores	Usuario
Propósito	Crea una columna con el tipo de dato espacio-temporal.
Resumen	El usuario crea una columna en una tabla con el tipo de dato espacio-temporal para almacenar objetos espacio-temporales.
Tipo	Primario
Referencias	Funciones: R1.1

Tabla 3.3 Caso de uso: agregar columna espacio-temporal

Caso de uso	**Crear índices**
Actores	Usuario
Propósito	Proveer una interfaz de creación de índices de acceso particulares preparados para el tipo de dato espacio-temporal de tal forma que soporte la recuperación eficaz de los datos en consultas espacio-temporales.

Resumen	El usuario crea índices de acceso a las columnas de tipo espacio-temporales que definió.
Tipo	Primario
Referencias	Funciones: R1.2

Tabla 3.4 Caso de uso: Crear índices

Caso de uso	**Insertar datos espacio-temporales.**
Actores	Usuario
Propósito	Proveer una forma de inserción de un tipo compuesto por una parte espacial que representa la figura de los objetos y otra temporal que representa el tiempo de existencia de los mismos.
Resumen	El usuario inserta datos espacio-temporales en la Base de Datos.
Tipo	Primario
Referencias	Funciones: R1.3

Tabla 3.5 Caso de uso: Insertar datos espacio-temporales

Caso de uso	**Modificar datos espacio-temporales.**
Actores	Usuario
Propósito	Proveer una forma para la modificación los objetos de tipo espacio-temporal.
Resumen	El usuario modifica datos espaciales y/o temporales en el Sistema Manejador de Base de Datos.
Tipo	Primario
Referencias	Funciones: R1.4

Tabla 3.6 Caso de uso: Modificar datos espacio-temporales

Caso de uso	**Modificar datos espacio-temporales.**
Actores	Usuario
Propósito	Proveer una forma para la eliminación de los objetos de tipo espacio-temporal de la base de datos.
Resumen	El usuario elimina los datos espacio-temporales de la Base de Datos.
Tipo	Primario
Referencias	Funciones: R1.5

Tabla 3.7 Caso de uso: Eliminar datos espacio-temporales

Caso de uso	Consultar datos espacio-temporales.
Actores	Usuario
Propósito	El usuario consulta los datos espacio-temporales por la parte espacial que representa la figura de los objetos o por la parte temporal que representa el tiempo de existencia de los mismos. Adicionalmente el usuario debe contar con un conjunto de operadores que implementan las relaciones espacio-temporales considerando las propiedades espaciales y temporales homogéneamente.
Resumen	El usuario consulta datos espacio-temporales
Tipo	Primario
Referencias	Funciones: R1.6, R1.7, R1.8

Tabla 3.8 Caso de uso: Consultar datos espacio-temporales

3.1.3 Diseño y arquitectura del componente.

El componente debe dotar al manejador de base de datos de código abierto PostgreSql de interfaces para el almacenamiento de datos espacio-temporales y debe poseer una estructura de almacenamiento adecuada para datos espaciales y temporales además de las interfaces de consulta para tratar la información espacial y temporal homogéneamente, debido a que las consultas espacio-temporales tratan el objeto espacial y temporalmente como un todo. Es bien conocido que los

manejadores de base de datos de código abierto están preparados para la recuperación de los datos cuando se hace de forma heterogénea es decir si se quiere recuperar el dato espacialmente o por sus atributos temporales, pero al haber planteado la resolución de consultas espacio-temporales se debe adecuar la forma de almacenamiento y proveer una estructura que identifique al objeto como un todo. La figura 3.2 ilustra como idealmente debe funcionar el componente en alto nivel de abstracción y seguidamente se presenta un diagrama de componentes que representa la implementación en la figura 3.3.

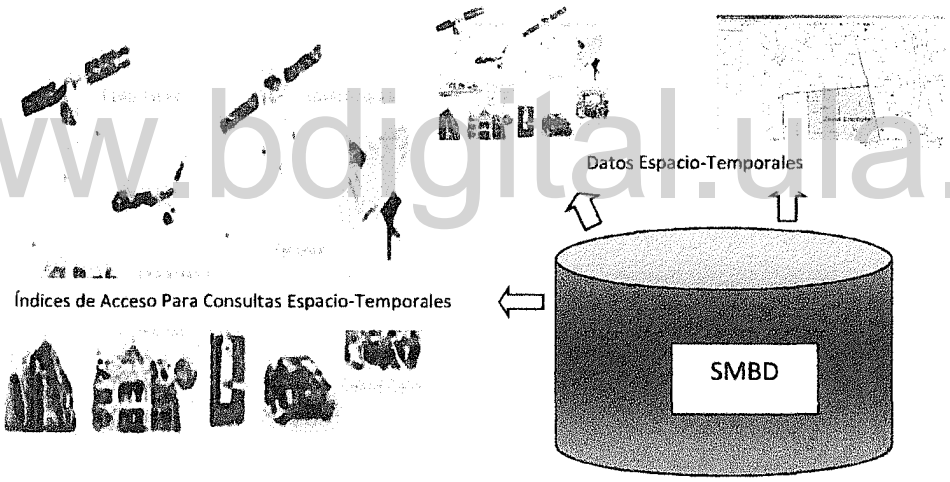


Figura 3.2 Bosquejo del funcionamiento del componente.

En principio se inició una prueba de conceptos que partió de hacer uso de una gran cantidad de características espaciales de Postgis y a la naturaleza extensible de *PostgreSQL* para extender *Postgis* con *wrappers* de sus métodos y extender los tipos de dato espaciales agregándole tipos nativos de *Postgres* para manejo del tiempo existencial y transaccional además de proveer dicha extensión con los

operadores necesarios reconfigurando los métodos de acceso para adecuarlos al componente espacio-temporal. El diseño de la extensión se muestra a continuación en la figura 3.2.1.

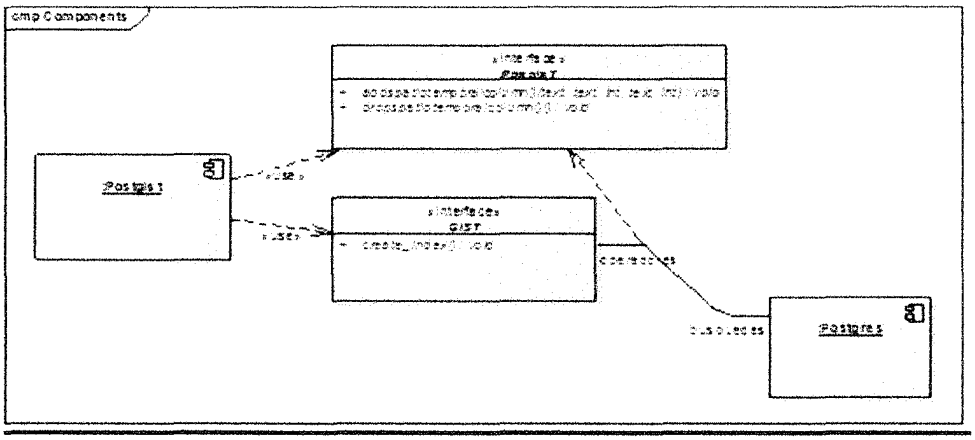


Figura 3.2.1 Adecuación del componente Postgis.

El diseño de las interfaces del componente que almacenan los tipos de dato espacio-temporales en memoria secundaria de la primera iteración, básicamente consiste en que a cada tabla con un atributo geométrico se le agregue por defecto los atributos temporales de existencia y se muestran a continuación en las tablas 3.9, 3.10

Interfaces de usuario para el almacenamiento en memoria secundaria		
Interfaz	Funcionalidad	Ejemplo
addspatiotemporalcolumn(*tablename*, *geometrycolumnname*, *srid*, *geometrycolumname*, *dimensiones*)	Aporta la creación de una columna espacio-temporal en el manejador de base de datos	AddSpatioTemporalColumn(*'aviones','avion',-1,'POLIGON',2*)
dropspatiotemporalcolumn(*oid*)	Elimina la columna espaciotemporal	Dropsqptiotemporalcomn(*oid*)

Tabla 3.9 Interfaces de usuario para el almacenamiento en memoria secundaria

Interfaces de usuario para la creación de índices de columnas espacio-temporales		
Interfaz	Funcionalidad	Ejemplo
Create index indexname on *tablename* using gist*(column_name* gist_spatiotemporal_option)	Aporta la creación de un índice para una columna espacio-temporal en el manejador de base de datos.	AddSpatioTemporalColumn(*aviones gist_spatiotemporal_option*)

Tabla 3.10 Interfaces de usuario para la creación de índices de columnas espacio-temporales

Posteriormente al planteamiento de diseño en la primera iteración, se procedió a la implementación, la cual resultó factible en los procesos de creación de la columna espacio-temporal y resolución del problema fundamental se refiere, pero cuando se buscó implementar el proceso de indexación se encontraron algunas restricciones en la manera como se realiza el pase de parámetros de las funciones de C en *Postgres*, y se debe a que los argumentos internos de las funciones de C no asumen la tupla completa para el pase por referencia a las funciones de indexación del componente *Gist* sino que reciben únicamente el tipo de dato como tal que para *Postgis* está definido genéricamente como tipo *geometry* el cual tiene longitud variable puesto que engloba los distintos tipos de geometrías por ejemplo *point, polygon, etc,* que por sus características propias son de longitud variable y al agregar al parte temporal complica la reinterpretación del tipo de dato compuesto fusionado en un tipo de dato espacio-temporal en el mismo atributo que incrementa considerablemente los tiempos de desarrollo. Por este motivo se decidió rectificar el modelo y utilizar un nivel de acoplamiento más débil planteando la separación del componente temporal y espacial integrándolos en un nivel más alto por medio de un tipo de dato compuesto, separando los procesos de indexación lo cual consideramos conceptualmente valido justificado en la n-dimensionalidad de las estructura de indexación expuestas en el capítulo II y en el hecho de que en la bases de datos espacio-temporales el intervalo de existencia configura un escenario para el cual los objetos existen a nivel de recuperación y dicho intervalo temporal podría estar indexado en un árbol R al igual que la información espacial organizada en otra estructura de tipo R para dividir la recuperación y la distribución de los escenarios temporal y espacial en dos sub-conjuntos que recuperables a nivel de indexación.

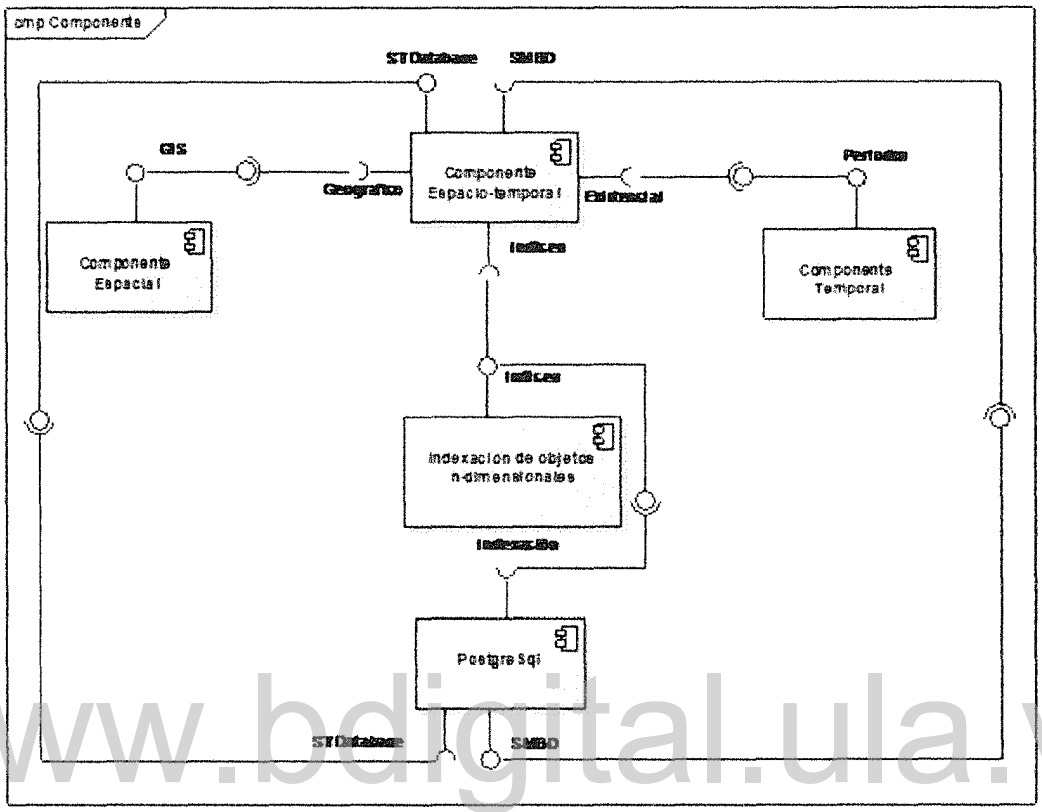


Figura 3.3 Diagrama de componentes a nivel abstracto

3.1.4 Diagramas de interacción

En ésta sección se muestran algunos aspectos dinámicos y deónticos del componente. Por medio de los diagramas de secuencia se establece, en primer lugar, el protocolo de interacción entre el componente y el entorno, para luego identificar los componentes participantes y establecer sus responsabilidades de cara a la colaboración que representa el componente.

3.1.4.1 Diagrama de Secuencia: crearColumnaEspacioTemporal

La operación crearColumnaEspacioTemporal ocurre cuando el usuario necesita crear la estructura para almacenar objetos espacio-temporales (ver figura 3.4).

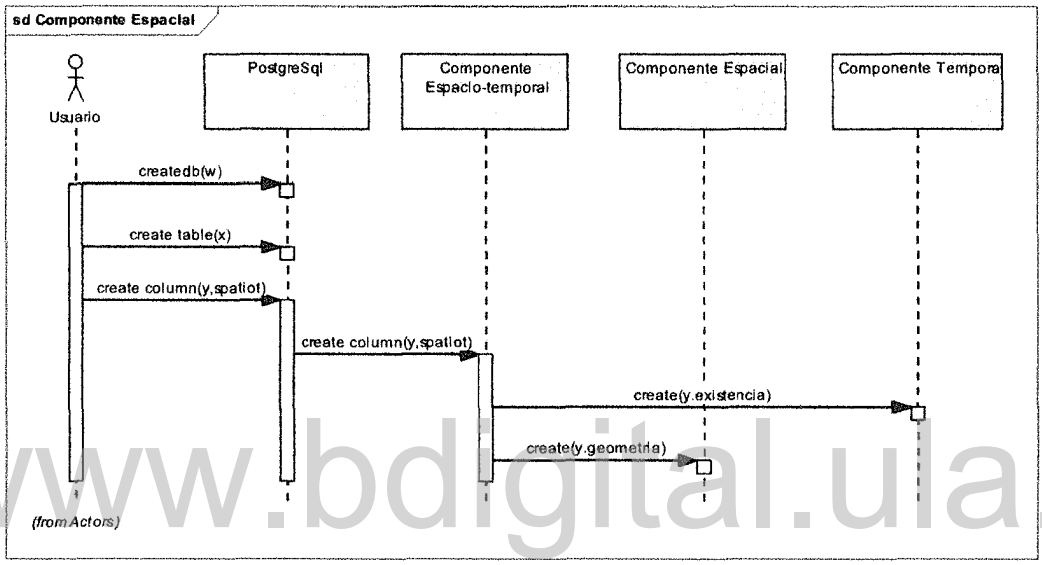


Figura 3.4. Diagrama de secuencia para creación de la estructura.

3.1.4.2 Diagrama de Secuencia: CrearIndices

La operación crearIndices ocurre cuando el usuario necesita crear los índices a las columnas espaciales y temporales (ver figura 3.5).

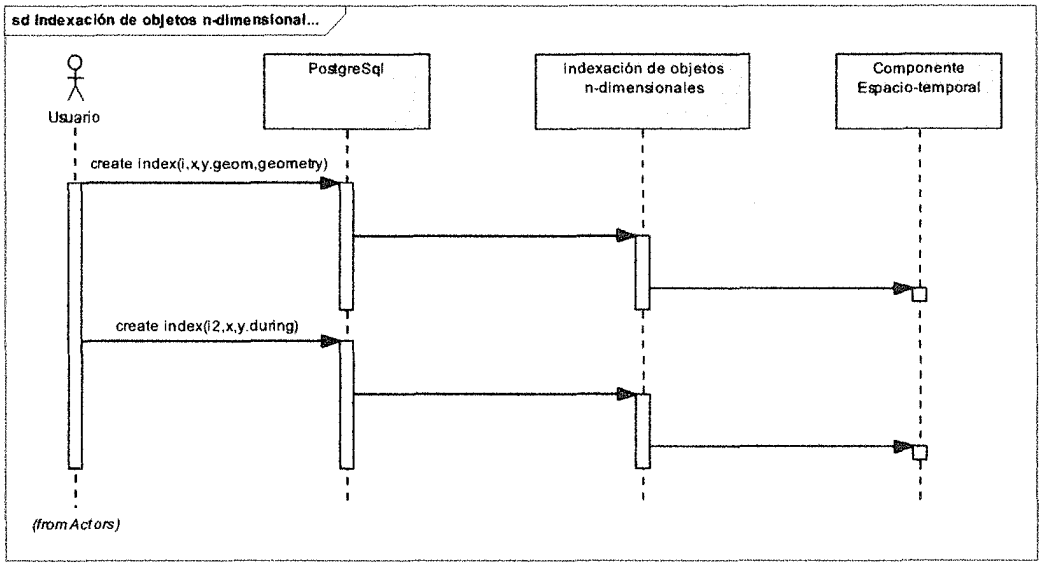


Figura 3.5. Diagrama de secuencia para creación de índices.

3.2 Aprovisionamiento de componentes

El método W_Watch propone en este proceso la selección y acondicionamiento de los componentes útiles o librerías que contribuyan la reutilización para el desarrollo de los componentes para luego definir la plataforma o infraestructura de desarrollo del componente.

3.2.1 Adquisición de componentes

En el modelo de componentes abstracto se plantea *Postgresql* como el sistema manejador de base de datos justificado en las investigaciones y basado en la extensibilidad, luego se presenta la necesidad de un componente que gestione los datos espaciales, otro que maneje los datos temporales y por último, un componente que maneje la indexación que pueda ser accedido desde los componentes espacial y el temporal de forma tal que al implementar las

adaptaciones y operaciones necesarias se pueda proveer la capacidad al SMBD de almacenamiento y recuperación eficaz de objetos espacio-temporales.

3.2.1.1 Componente para el manejo espacial.

El componente de principal importancia es el componente de datos espaciales existente para *PostgreSql* denominado *Postgis* el cual fue una de las razones de peso para la selección del manejador cuyas características se hicieron notar en el capítulo I, se obtuvo en la versión *open source* postgis-1.3.6SVN que además necesitaba librerías de complemento tales como *geos-3.1.0* y *proj-4.6.0* también de código abierto. El código fuente fue descargado los repositorios oficiales en internet y compilados en la plataforma Linux, luego fueron integrados a *PostgreSql* comprobando su adecuado funcionamiento y verificando que era de gran utilidad para la implementación de prototipo espacio-temporal.

3.2.1.2 Componente de indexación.

PostgreSql desde la versión 8.1 cuenta con el componente denominado árbol de búsquedas generalizadas GiST que brinda estructuras de indexación genéricas con un API compuesta de interfaces para ser extendidas hacia nuevos tipos de datos y así optimizar su indexación. Como se menciona en el capítulo II, la implementación GiST de PostgreSQL proporciona el soporte para la indexación de

PostGIS. El detalle de las adaptaciones e integraciones a nivel de implementación se expresará en el capítulo IV.

3.2.1.3 Componente temporal.

PostgreSql cuenta con tipos de dato SQL para almacenar fechas y tiempos bien definidos con operadores básicos, estructura de almacenamiento e indexación basada en árboles B, los cuales para manejar el tiempo transaccional al que se hizo referencia en la conceptualización es suficiente. Éstos tipos son los *timestamp* y sus variantes y el tipo de dato *interval* con todas sus funciones.

Con respecto a los intervalos de existencia de los objetos espacio-temporales en necesario la manipulación de la parte espacial y temporal homogéneamente, lo cual hace necesario la búsqueda y adaptación de un componente en el cual se puedan acceder a los operadores para redefinirlos y adaptarlos a los nuevos tipos de datos y agruparlos con los operadores espaciales y de igual forma con la indexación. En la iteración correspondiente a la implementación de dicho componente se paralelizaron pruebas de concepto con los tipos de datos nativos de PostgreSql y con una implementación que se consiguió realizada en la Universidad de California, la cual se explica en detalle en el capítulo IV.

3.3 Segunda Iteración en el proceso de diseño del método

En esta iteración se plantea un refinamiento de la estructura en concreto o más definitiva, basada en los procesos de la iteración de diseño anterior donde se eligieron los componentes y se presentaron los requerimientos que se deben cumplir.

3.3.1 Arquitectura del componente a nivel concreto

La arquitectura planteada hace uso de las características de un componente para en SMBD PostgreSql denominado Postgis que implementa las especificaciones del OGC (*Open Geospatial Consortium*) brindando estándares y calidad para el soporte de aplicaciones GIS y también de un componente adecuado para soportar el intervalo de existencia de los objetos espaciales, brindando así interfaces para la creación de objetos espacio-temporales como se ilustra en el diagrama de componentes de la figura 3.6.

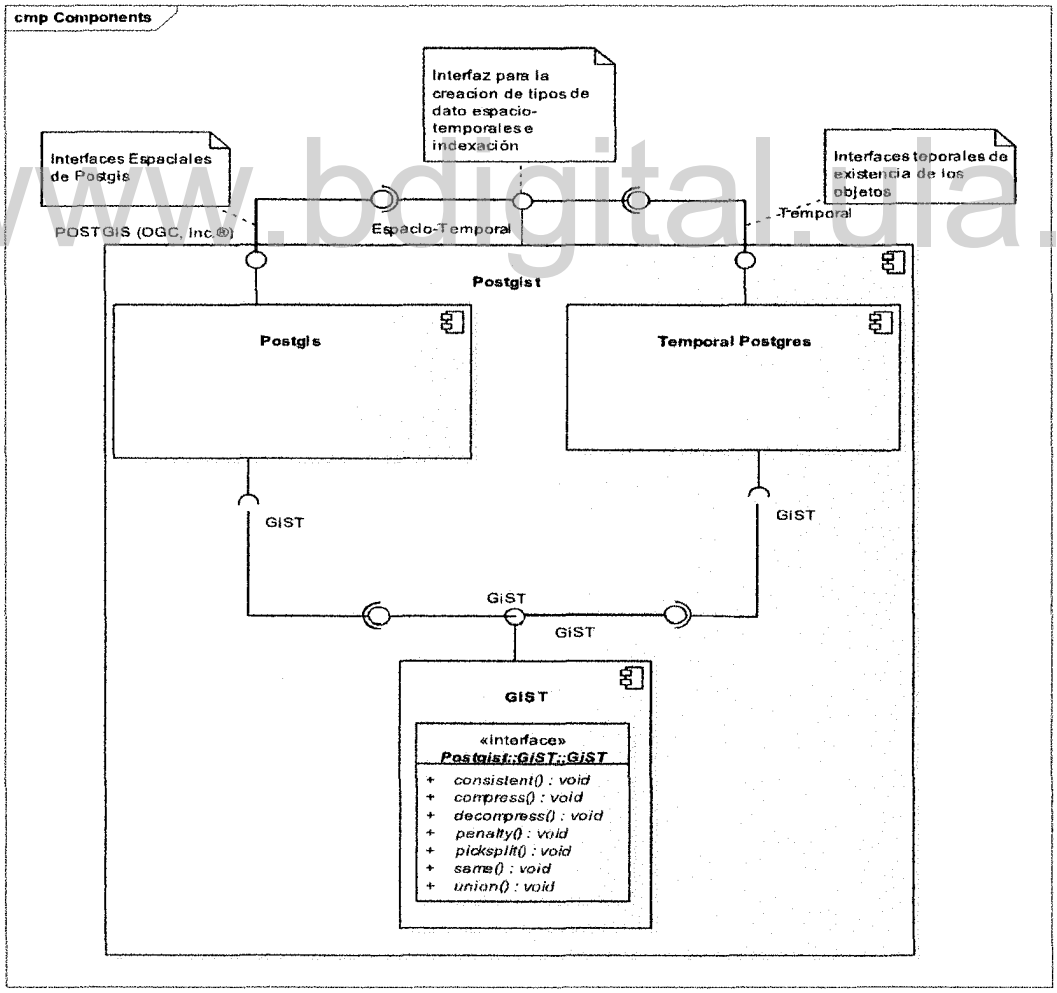


Figura 3.6 Diagrama del componente

3.3.2 Especificación del componente

A continuación se definen los tipos de dato e interfaces de consulta para la interacción con el componente a nivel de usuario.

3.3.3 Tipo de dato espacio-temporal

En principio un tipo de dato espacio-temporal a nivel de diseño lo podemos modelar como una clase de objetos denominado spatiot contentivo de un atributo geométrico reutilizando el tipo de dato genérico de *Postgis* denominado Geometry, que aporta al manejador de base de datos la capacidad de almacenar las geometrías básicas, tales como: puntos, polígonos, líneas, etc. Asimismo un atributo temporal que indica el tiempo en el que existe el dato geométrico para luego ser manipulado en forma de intervalos y poder interpretar las relaciones espacio-temporales, otro atributo posible en el que almacena el tiempo en el que se ejecutó la transacción de inserción ó última modificación del dato geométrico en el sistema manejador de base de datos pero este último puede ser manipulado genéricamente por el SMBD implementando un disparador o *trigger* en una columna y se puede dejar a disposición del usuario, aunque la mayoría de los manejadores poseen un registro de las transacciones en un log. El tipo de dato base para el componente en UML en la figura 3.7.

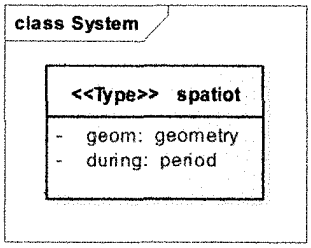


Figura 3.7 Definición del tipo EspacioTemporal

3.3.4 Resolución de Consultas

Seguidamente, el componente debe satisfacer el problema fundamental de base de datos aportando los operadores mínimos necesarios para las consultas en el contexto de las bases de datos espacio-temporales como lo muestra la figura 3.8.

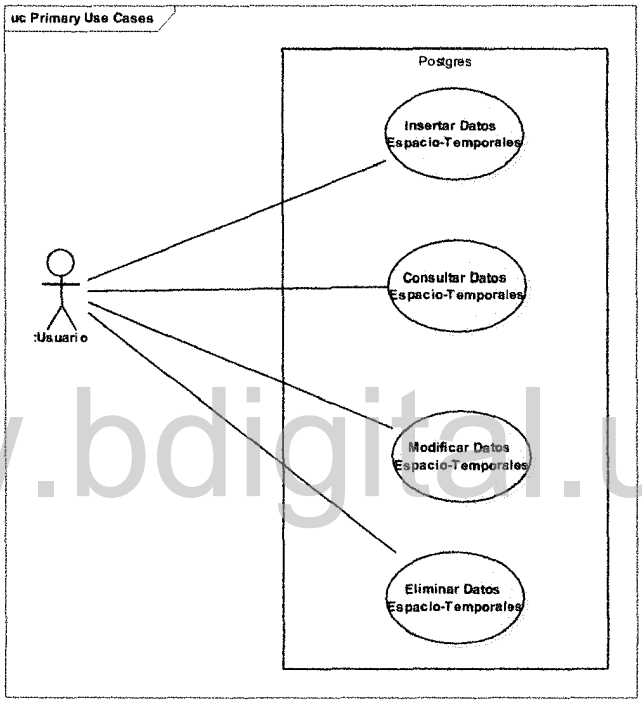


Figura 3.8 Casos de uso para el solucionar el problema fundamental de Base de Datos

3.3.5 Diagrama de secuencia: IngresarDatos espacio-temporales

La operación IngresarDatos ocurre cuando se necesita insertar datos espacio-temporales en la base de datos (ver figura 3.8).

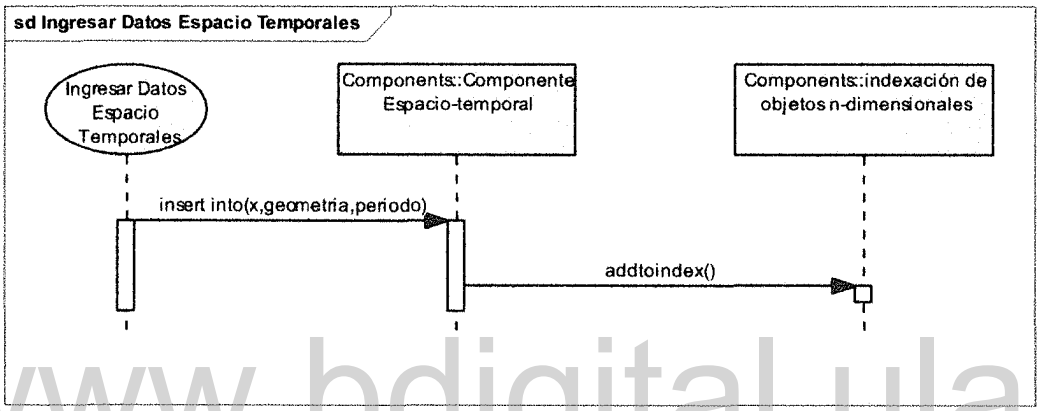


Figura 3.8 Diagrama de secuencia de IngresarDatos

Y análogamente al diagrama de la figura 3.8 se realiza las secuencias para la modificación y la eliminación de datos espacio-temporales

3.3.6 Diagrama de secuencia: ConsultarDatos espacio-temporales

La operación ConsultarDatos espacio-temporales ocurre cuando se necesita acceso a los datos ya almacenados en la base de datos que cumplen alguna condición espacial, temporal o espacio-temporal por ejemplo: los objetos que estaban en el sitio cercano al lugar X entre la el 23-12-2009 a las 03:30:20 (ver figura 3.9).

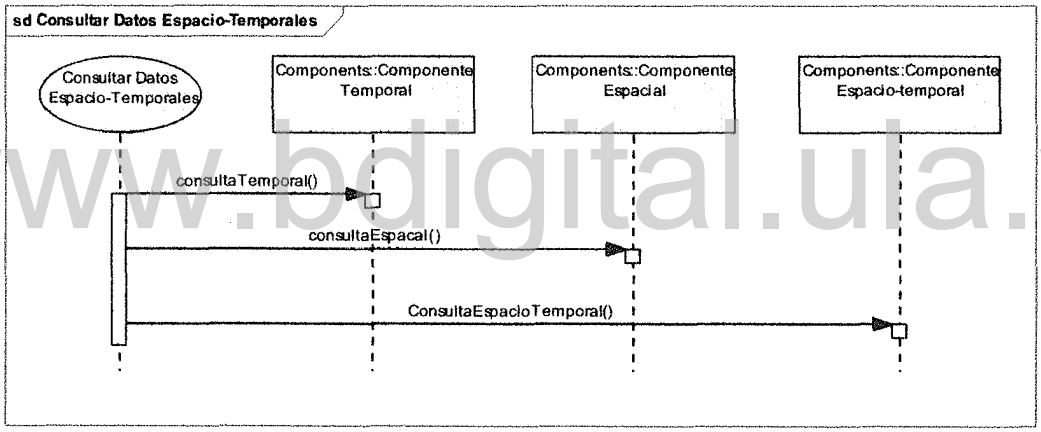


Figura 3.9 Diagrama de componentes de Consultar Datos

CAPÍTULO IV

Implementación del prototipo

En esta fase se realizó una primera iteración en el subproceso de aprovisionamiento de componentes del proceso de desarrollo del método *W_Watch* por medio de la extensión del componente *PostGis* en forma fuertemente acoplada sobrescribiendo sus métodos y atributos, la cual no tuvo éxito debido a la forma en que está implementado el pase de parámetros a las funciones de indexación escritas en lenguaje C, por ende se refinó el modelo y en una segunda iteración se describe la implementación del componente con un nivel de acoplamiento más débil, luego de refinar el modelo y tomando como insumo el resultado de la segunda iteración del proceso de diseño, en el cual se obtuvo la especificación en concreto de los componentes a adecuar e implementar.

4.1 Plataforma de desarrollo

En la fase de Selección de la plataforma se ratifica la selección del sistema manejador de base de datos *PostgreSQL* debido a su extensibilidad y documentación. Los tipos de dato, métodos de acceso, operadores que son almacenados en tablas catálogos, pueden ser implementados como módulos cargables. Muestra de ello lo son el componente *Postgis* que implementa funcionalidades de almacenamiento de tipos de datos espaciales y el componente *GiST* que implementa una forma generalizada de búsqueda por medio de la indexación extensible. La tabla 4.1 presenta un resumen de las herramientas utilizadas en el desarrollo del componente.

Nombre	Descripción
Postgresql	Sistema Manejador de Base de Datos. Compilado desde el código fuente. Versión 8.3.4
Mandriva Linux 2009.	Sistema Operativo
MonoDevelop	Entorno integrado de desarrollo.IDE Versión 1.0,
pgAdmin III	Versión 1.8.4 Interfaz de pruebas y desarrollo para Postgres
Vim	Editor de texto

Tabla 4.1 Herramientas utilizadas para el desarrollo de la aplicación

Los pasos de instalación y compilación de las herramientas base se encuentran descritos en el anexo A.

4.2 Adecuación de los componentes

En esta sección se muestran las 2 iteraciones de adecuación de componentes, la primera considerada como fallida y la segunda que representa la implementación del prototipo.

4.2.1 Adecuación de Postgis en primera iteración

El planteamiento de adecuación de Postgis en una primera iteración se realizó por medio de *wrappers*, agregando el atributo temporal de un tipo nativo de PostgreSQL para el almacenamiento en memoria secundaria y funcionaron correctamente a nivel de implementación, con excepción de las interfaces de integración con GiST que se encargarían de hacer la indexación, la cual falló debido a que la interfaz para el desarrollo de funciones en C de PostgreSQL no está preparada para tuplas con tipos de dato compuestos, puesto que la firma del método recibe un parámetro o argumento llamado PG_ARGUMENT, dicho argumento, puede ser reinterpretado si lleva un tipo de dato o un arreglo de un tipo de dato por medio de casting a un tipo de dato de los descritos en la tabla 4.2 de equivalencias entre tipos en postgresql (SQL) y tipos en lenguaje C, ya que en el caso de las interfaces de los índices en GiST es de tipo Datum que consiste en el tipo de dato para el que fue definido el índice y en Postgis son de longitud variable dependiendo de la figura geométrica por ende delimitar las tramas de bits para reinterpretarlo y extraer los valores temporales se hizo infactible bajo el escenario de la base de código existente.

Built-In Type	C Type	Definido en
Abstime	AbsoluteTime	utils/nabstime.h
Bool	Bool	include/c.h
Box	(BOX *)	utils/geo-decls.h
Bytea	(bytea *)	include/postgres.h
Char	Char	N/A
cid	CID	include/postgres.h
datetime	(DateTime *)	include/c.h or include/postgres.h
int2	int2	include/postgres.h
int2vector	(int2vector *)	include/postgres.h
int4	int4	include/postgres.h
float4	float32 or (float4 *)	include/c.h or include/postgres.h
float8	float64 or (float8 *)	include/c.h or include/postgres.h
Lseg	(LSEG *)	include/geo-decls.h
name	(Name)	include/postgres.h
Oid	Oid	include/postgres.h
oidvector	(oidvector *)	include/postgres.h
Path	(PATH *)	utils/geo-decls.h

point	(POINT *)	utils/geo-decls.h
regproc	regproc or REGPROC	include/postgres.h
reltime	RelativeTime	utils/nabstime.h
Text	(text *)	include/postgres.h
Tid	ItemPointer	storage/itemptr.h
timespan	(TimeSpan *)	include/c.h or include/postgres.h
tinterval	TimeInterval	utils/nabstime.h
uint2	uint16	include/c.h
uint4	uint32	include/c.h
Xid	(XID *)	include/postgres.h

Tabla 4.2 Tipos de C equivalentes para los tipos internos de PostgreSQL

4.2.2 Adecuación de componentes en la segunda iteración

En esta segunda iteración como se mencionó anteriormente se escoge el diseño refinado de la segunda iteración de diseño que contiene 3 componentes esenciales que son el Postgis, Temporal Postgres y el GiST, y se realizó una integración entre los componentes utilizando las interfaces que proveen y la capacidad de extensibilidad de PostgreSQL para la constitución de tipos de datos compuestos.

4.2.2.1 Adecuación de Postgis

El componente Postgis no amerita modificaciones de fondo para esta versión debido a que los tipos esenciales que posee el componente son suficientes para la representación de los datos espaciales, pero debe ser integrado en la carga dinámica de las funciones del catálogo de PostgreSQL para la base de datos espacio-temporal en el orden correcto de precedencia de los tipos de dato y operadores, dicha carga dinámica se realiza en el componente Posgist ejecutando el script *postgis.sql*, el cual debe ser suplantado por el archivo del prototipo encargado de gestionar la carga dinámica ordenada del catálogo espacio-temporal cuya implementación se describirá a continuación.

4.2.2.2 Codificación de funciones en C que implementan los operadores

La implementación necesita de una serie de pasos para hacer efectiva la integración. Parte del código en C++ para la clase **period** se muestra en la tablas 4.3 y 4.4 utilizando como ejemplo la función de solapamiento (*overlaps*) temporal que a su vez forma parte del solapamiento espacio-temporal.

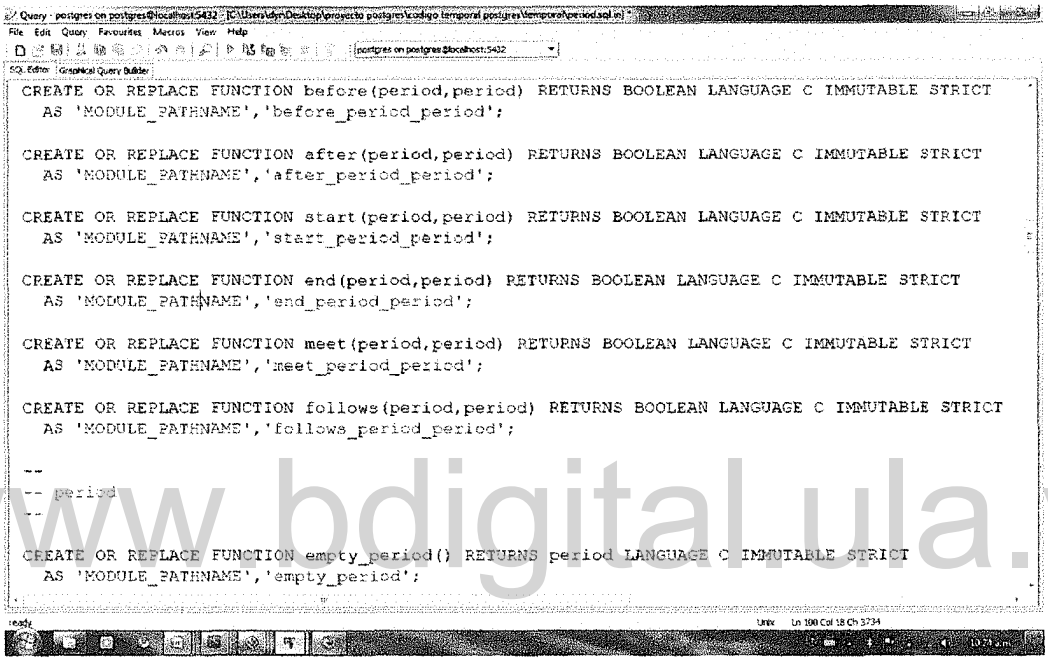
```
PG_FUNCTION_INFO_V1(overlaps_period_period);

Datum

overlaps_period_period(PG_FUNCTION_ARGS)

{

      period *p1 = (period*)PG_GETARG_POINTER(0);

      period *p2 = (period*)PG_GETARG_POINTER(1);

      PG_RETURN_BOOL(period_overlaps(p1,p2));

}
```

Tabla 4.3 Código en C de la clase period modificada en la función de sobreexposición

```
Bool

period_overlaps(period *p1, period *p2)

{

      TimestampTz last1,last2;

      if(period_is_empty(p1) || period_is_empty(p2))

            return false;

      // necesitamos un intervalo cerrado

      last1 = prior_timestamptz(p1->next);

      last2 = prior_timestamptz(p2->next);

      return ((p1->first <= last2 && last2 < p1->next) ||

                  (p2->first <= last1 && last1 < p2->next));

}
```

Tabla 4.4 Código en C de la clase period modificada para implementar el operador overlap

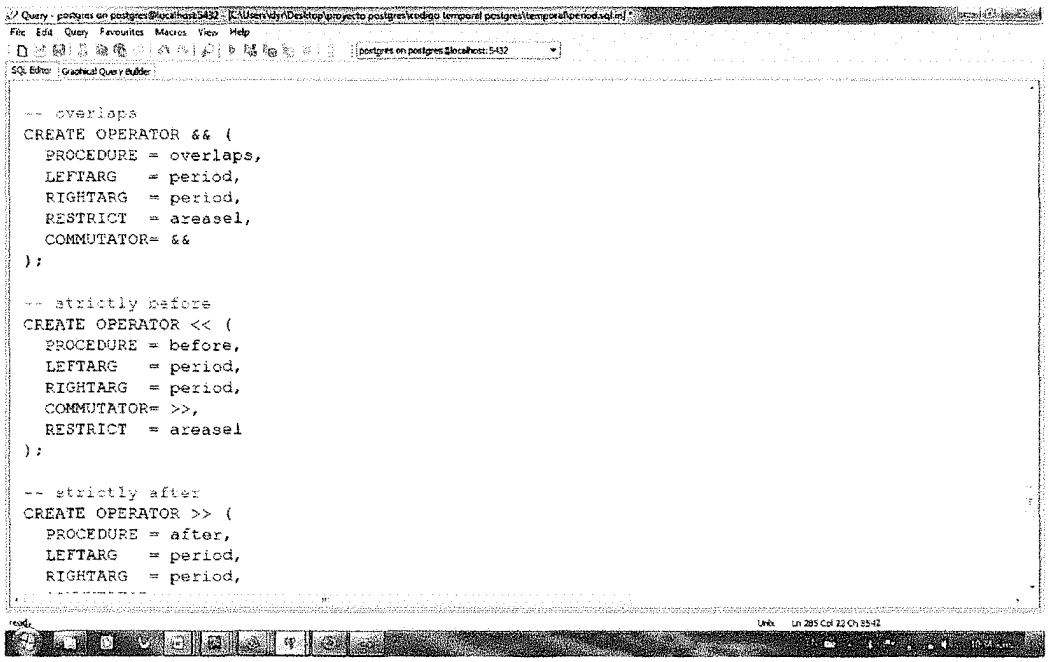
PostgreSQL ofrece la posibilidad de cargar dinámicamente funciones escritas en el lenguaje C compiladas. La codificación para la carga dinámica de las funciones y operadores temporales se puede observar en las figuras 4.1 y 4.2



```
CREATE OR REPLACE FUNCTION before(period,period) RETURNS BOOLEAN LANGUAGE C IMMUTABLE STRICT
  AS 'MODULE_PATHNAME','before_period_period';

CREATE OR REPLACE FUNCTION after(period,period) RETURNS BOOLEAN LANGUAGE C IMMUTABLE STRICT
  AS 'MODULE_PATHNAME','after_period_period';

CREATE OR REPLACE FUNCTION start(period,period) RETURNS BOOLEAN LANGUAGE C IMMUTABLE STRICT
  AS 'MODULE_PATHNAME','start_period_period';

CREATE OR REPLACE FUNCTION end(period,period) RETURNS BOOLEAN LANGUAGE C IMMUTABLE STRICT
  AS 'MODULE_PATHNAME','end_period_period';

CREATE OR REPLACE FUNCTION meet(period,period) RETURNS BOOLEAN LANGUAGE C IMMUTABLE STRICT
  AS 'MODULE_PATHNAME','meet_period_period';

CREATE OR REPLACE FUNCTION follows(period,period) RETURNS BOOLEAN LANGUAGE C IMMUTABLE STRICT
  AS 'MODULE_PATHNAME','follows_period_period';

--
-- period
--

CREATE OR REPLACE FUNCTION empty_period() RETURNS period LANGUAGE C IMMUTABLE STRICT
  AS 'MODULE_PATHNAME','empty_period';
```


Figura 4.1 Código para la carga dinámica de los funciones temporales

En la figura 4.1 se ilustra la forma de enlace dinámico. Las funciones definidas en lenguaje C con el manejador de Base de Datos *PostgreSQL* para ser utilizadas con el lenguaje SQL con la sintaxis de ejecución de un procedimiento almacenado por ejemplo para la sobreposición temporal **`SELECT overlap(period, period);`** luego se sobrescriben los operadores correspondientes para el tipo de dato temporal. En la figura siguiente se puede observar como por ejemplo el operador "**`&&`**" es asignado a la función de sobreposición (*overlaps*) enlazada anteriormente.



```
-- overlaps
CREATE OPERATOR && (
  PROCEDURE = overlaps,
  LEFTARG   = period,
  RIGHTARG  = period,
  RESTRICT  = areasel,
  COMMUTATOR= &&
);

-- strictly before
CREATE OPERATOR << (
  PROCEDURE = before,
  LEFTARG   = period,
  RIGHTARG  = period,
  COMMUTATOR= >>,
  RESTRICT  = areasel
);

-- strictly after
CREATE OPERATOR >> (
  PROCEDURE = after,
  LEFTARG   = period,
  RIGHTARG  = period,
```

Figura 4.2 Código para la carga dinámica de los operadores temporales

De esta manera al quedar definido el operador *&&* se puede enlazar con el componente de indexación *GiST* por medio de la codificación ilustrada en la siguiente tabla 4.5.

```
CREATE OPERATOR CLASS gist_period_ops

  DEFAULT FOR TYPE period USING gist AS

    OPERATOR   1    <<,   -- strictly before

    OPERATOR   2    &<,   -- overlaps or left of

    OPERATOR   3    &&,   -- overlaps

    OPERATOR   4    &>,   -- overlaps or right of

    OPERATOR   5    >>,   -- strictly after

    OPERATOR   6    =,    -- equal

    OPERATOR   7    @>,   -- contains
```

```
    OPERATOR  8     <@,   -- contained by
    OPERATOR 17     ~,    -- alias for contains
    OPERATOR 18     @,    -- alias for contained by
    OPERATOR 27     @>(period,TIMESTAMPTZ),
    OPERATOR 28     <@(TIMESTAMPTZ,period),
    FUNCTION  1     gist_period_consistent(internal, period, int4),
    FUNCTION  2     gist_period_union(internal, internal),
    FUNCTION  3     gist_period_compress(internal),
    FUNCTION  4     gist_period_decompress(internal),
    FUNCTION  5     gist_period_penalty(internal, internal, internal),
    FUNCTION  6     gist_period_picksplit(internal, internal),
    FUNCTION  7     gist_period_same(period, period, internal);
```

Tabla 4.5 Código para la carga dinámica en el catálogo de *PostgreSQL* en los operadores a nivel de indexación

El componente *Temporal Postgres* necesita la definición de operadores ilustrada anteriormente en la tabla 4.5 que complementan implementación de los operadores espacio-temporales por ende de se agregó algunas funciones ausentes como lo son: operador de comienzo denominado (start), el operador de finaliza (end), concurre (meet).

Los componentes *Postgis* y *Temporal Postgres* modificado integrados al manejador de base de datos son precondición para el funcionamiento del componente espacio-temporal.

4.3 Prototipo

El componente se denomina Posgist en honor a las componentes en los que se apoya los cuales son Postgis, GiST y *Teporal Postgres* el cual es un componente que permite realizar las siguientes funciones:

1. Agregar columnas del tipo de dato "spatiot" que posee la implementación de los operadores en el contexto de base de datos espacio-temporales.
2. Creación de índices de acceso para los datos espaciales y temporales que son accedidos por los operadores espacio-temporales a nivel de inserción, modificación, eliminación y consulta de datos.
3. Realizar consultas espaciales, temporales y espacio-temporales con tiempos de respuesta aceptables.
4. Soporta modelos de objetos que varían su posición y/o su forma en el tiempo.
5. Es genérico para soportar particularidades de los modelos espacio-temporales planteados en el mundo real.

Una vez adecuados loa componentes anteriores se puede definir el tipo compuesto que es la base fundamental del componente ***Posgist*** bajo la premisa de la segunda iteración del diseño se procede a introducir en el catálogo el tipo de dato compuesto con la sentencia **`CREATE TYPE`** como se ilustra a continuación.

```
CREATE TYPE spatiot AS (geom geometry, during  period);
```

De igual manera que la adecuación del componente temporal se deben definir los operadores del componente espacio-temporal el cual se ilustrará con fines de

ejemplificar la definición de la función de solapamiento (overlap) para el tipo de dato compuesto de *Posgist* para el tipo "***spatiot***".

```
CREATE OR REPLACE FUNCTION "equals"(spatiot, spatiot)
  RETURNS boolean AS
$BODY$BEGIN
IF (equals($1.during,$2.during)) THEN
        IF (equals($1.geom,$2.geom)) THEN
                RETURN TRUE;
        ELSE
                RETURN FALSE;
        END IF;
ELSE
        RETURN FALSE;
END IF;
END;$BODY$
  LANGUAGE 'plpgsql' VOLATILE;



-- Function: "overlaps"(spatiot1, spatiot1)

-- DROP FUNCTION "overlaps"(spatiot1, spatiot1);

CREATE OR REPLACE FUNCTION "overlaps"(spatiot, spatiot)
  RETURNS boolean AS
$BODY$BEGIN
```

```
IF (overlaps($1.during,$2.during)) THEN

IF (overlaps($1.geom,$2.geom)) THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

ELSE

RETURN FALSE;

END IF;

END;$BODY$

  LANGUAGE 'plpgsql' VOLATILE;


-- equals

CREATE OPERATOR = (

  PROCEDURE = equals,

  LEFTARG   = spatiot,

  RIGHTARG  = spatiot,

  NEGATOR   = !=,

  RESTRICT  = eqsel

);
```


4.4Entorno del componente prototipo

Las interfaces del ***Posgist*** fueron diseñadas de forma tal que se puede interactuar de la manera tradicional como se interactúa con PostgreSQL por medio de los lenguajes que provee tales como *SQL, PLPGSQL* como si fuese un tipo nativo de *PostgreSql.*

Con respecto a la visualización de la parte gráfica se puede usar cualquiera de las interfaces que se conectan a ***Postgis*** para sistemas de información geográfica como: *Mapserver, Grass, Arcview*, debido a que una vez obtenidas las consultas se puede tener acceso a la parte geométrica del tipo espacio-temporal como un tipo nativo de ***Postgis*** y comunicarlo con las herramientas de visualización.

4.5Creación de un esquema básico de base de datos espacio-temporal

Para generar un modelo básico que contiene una clase con un atributo identificador denominado "oidst" y un atributo espacio-temporal denominado "spatiotcolumn" como se ilustra en el diagrama de clases de la figura 4.5.

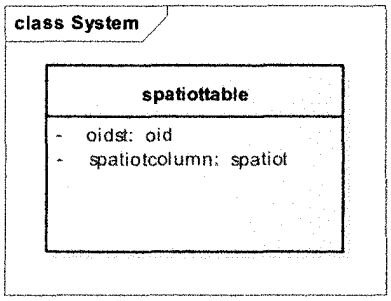


Figura 4.3 Clase spatiottable

El esquema relacional para el modelo planteado se muestra a continuación.

Spatiottable(<u>oidst</u>, spatiotcolumn);

Luego de tener el modelo definido y el esquema relacional se procede a la creación de la base de datos que exige como prerrequisito los componentes ***Postgis*** y

Temporal Postgres precompilados y configurados a *PostgreSQL*. La sintaxis para la creación de la base de datos se ilustra seguidamente en la figura 4.6

```
File Edit View Scrollback Bookmarks Settings Help
[root@dayana ~]# createdb gist_test
[root@dayana ~]# createlang plpgsql gist_test
[root@dayana ~]# psql -d gist_test -f posgist.sql > salida.out
psql:posgist.sql:56: NOTICE:  type "spheroid" is not yet defined
DETAIL:  Creating a shell type definition.
psql:posgist.sql:62: NOTICE:  return type spheroid is only a shell
psql:posgist.sql:68: NOTICE:  argument type spheroid is only a shell
psql:posgist.sql:74: NOTICE:  argument type spheroid is only a shell
psql:posgist.sql:91: NOTICE:  type "geometry" is not yet defined
DETAIL:  Creating a shell type definition.
psql:posgist.sql:97: NOTICE:  return type geometry is only a shell
psql:posgist.sql:103: NOTICE:  argument type geometry is only a shell
psql:posgist.sql:109: NOTICE:  argument type geometry is only a shell
psql:posgist.sql:127: NOTICE:  return type geometry is only a shell
psql:posgist.sql:133: NOTICE:  return type geometry is only a shell
psql:posgist.sql:139: NOTICE:  argument type geometry is only a shell
psql:posgist.sql:145: NOTICE:  argument type geometry is only a shell
psql:posgist.sql:326: NOTICE:  type "box3d" is not yet defined
DETAIL:  Creating a shell type definition.
psql:posgist.sql:332: NOTICE:  argument type box3d is only a shell
psql:posgist.sql:338: NOTICE:  return type box3d is only a shell
psql:posgist.sql:344: NOTICE:  argument type box3d is only a shell
```


Figura 4.6 sintaxis para crear la base de datos

La sintaxis en SQL para la creación de la clase planteada se muestra en la tabla 4.3.

```
-- Table: spatiottable

-- DROP TABLE spatiottable;

CREATE TABLE spatiottable
(
  oidst oid NOT NULL,
  spatiotcolumn spatiot,
  CONSTRAINT pkoidstl PRIMARY KEY (oidst)
)

-- Index: during_idx
```

```
-- DROP INDEX during_idx;

CREATE INDEX during_idx

  ON spatiottable

  USING gist

  (((spatiotcolumn).during));

-- Index: spatial_idx

-- DROP INDEX spatial_idx;

CREATE INDEX spatial_idx

  ON spatiottable

  USING gist

  (((spatiotcolumn).geom));
```

Tabla 4.6 Sintaxis para crear la estructura en *PostgreSQL*

Seguidamente en la tabla 4.4, se muestra la sintaxis para la manipulación del problema fundamental de base de datos en SQL, donde los atributos espacial y temporal del tipo de dato espacio-temporal se acceden empleando notación de punto `spatiotcolumn.geom, spatiotcolumn.during`.

```
INSERT INTO spatiottable (oidst,spatiotcolumn.geom,spatiotcolumn.during)

VALUES

(1,GeomFromText('POLYGON((85 191,197 173,75 60,85 191))',-1),
period('2008-01-01 14:00:00', '2009-01-01 16:00:00'))


INSERT INTO spatiottable (oidst,spatiotcolumn.geom,spatiotcolumn.during)

VALUES

(1,GeomFromText('POINT(0 0 0)',-1), period('2008-01-01 14:00:00', '2009-
01-01 16:00:00'))


INSERT INTO spatiottable (oidst,spatiotcolumn.geom,spatiotcolumn.during)

VALUES
```

```
(2,GeomFromTextEWKT('LINESTRING(0 0,1 1,1 2)',-1), period('2008-01-01
14:00:00', '2009-01-01 16:00:00'))


INSERT INTO spatiottable (oidst,spatiotcolumn.geom,spatiotcolumn.during)

VALUES

(3,GeomFromText('POLYGON((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2
0,1 2 0,1 1 0))',-1), period('2008-01-01 14:00:00', '2009-01-01
16:00:00'))


INSERT INTO spatiottable (oidst,spatiotcolumn.geom,spatiotcolumn.during)

VALUES

(4,GeomFromText('MULTIPOINT(0 0 0,1 2 1)',-1), period('2008-01-01
14:00:00', '2009-01-01 16:00:00'))


INSERT INTO spatiottable (oidst,spatiotcolumn.geom,spatiotcolumn.during)

VALUES

(5,GeomFromText('MULTILINESTRING((0 0 0,1 1 0,1 2 1),(2 3 1,3 2 1,5 4
1))',-1), period('2008-01-01 14:00:00', '2009-01-01 16:00:00'))


INSERT INTO spatiottable (oidst,spatiotcolumn.geom,spatiotcolumn.during)

VALUES

(6,GeomFromText('MULTIPOLYGON(((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1
0,2 2 0,1 2 0,1 1 0)),((-1 -1 0,-1 -2 0,-2 -2 0,-2 -1 0,-1 -1 0)))',-1),
period('2008-01-01 14:00:00', '2009-01-01 16:00:00'))
```

Tabla 4.7 Sintaxis de inserción para distintos tipos nativos de postgis

```
UPDATE spatiottable SET spatiotcolumn.geom =
GeomFromText('POLYGON((85 191,197 173,75 60,85 191))',-1)
WHERE oidst = 1
```

Tabla 4.8 Sintaxis de modificación

```
DELETE FROM spatiottable WHERE oidst = '1'
```

Tabla 4.9 Sintaxis de eliminación

```
SELECT * FROM spatiottable m1, spatiottable m2 WHERE
overlaps((m1.spatiotcolumn).geom, (m2.spatiotcolumn).geom);
```

Tabla 4.10 Sintaxis de consulta con funciones y operadores espaciales


```
SELECT * FROM spatiottable m1, spatiottable m2 WHERE
overlaps((m1.spatiotcolumn).during,
(m2.spatiotcolumn).during)
```

Tabla 4.11 Sintaxis de consulta con funciones y operadores temporales

```
SELECT * FROM spatiottable m1, spatiottable m2 WHERE
overlaps(m1.spatiotcolumn,m2.spatiotcolumn)
```

Tabla 4.12 Sintaxis de consulta con funciones y operadores espacio-temporales.

CAPÍTULO V

Pruebas y Análisis de Resultados

Introducción

En este capítulo se describen las pruebas del prototipo Posgist por medio de la utilización de una interfaz de comunicación con postgres denominada ***Pgadmin III***. Donde comenzaremos por la creación de la base de datos y una tabla con atributos espacio-temporales en la cual se ejecutaran pruebas caja negra para verificar los requisitos funcionales que enfocan el problema fundamental de base de datos a alto nivel, hasta la verificación de la indexación y la aplicación de los operadores espacio-temporales.

5.1 Plan de pruebas

Debido a las características del prototipo Posgist es necesario ejecutar un plan de pruebas que incluya la funcionalidad separada de los componentes y luego la integración de los mismos así como también se debe realizar pruebas caja blanca para los mecanismos de indexación y caja negra para la funcionalidad a nivel de usuario.

5.1.1 Bases para las pruebas.

En primera instancia crearemos una base de datos a la cual se le debe definir el lenguaje ***PLPGSQL*** y se le debe cargar el prototipo ***PosGist*** para base de datos espacio-temporal que contiene las funciones, los tipos de dato espaciales,

temporales para los periodos de existencia, y espacio-temporales como se ilustra en la tabla 5.1.

Prueba de Instancia del Postgist	**Usuario** - Crear Base de datos - Agregar lenguaje de procedimientos. - Instanciar el componente Posgist.
Prueba	**Descripción**
Creación de la base de datos con el Posgist precargado.	Después de que se tienen los componentes compilados se desea crear una base de datos, probar la carga efectiva de las funciones y los tipos de dato espaciales, periodos de existencia, espacio-temporales.
Resultados Esperados	**Resultados Obtenidos**
Base de datos creada, con las funciones	Se creó la base de datos y se cargo exitosamente.

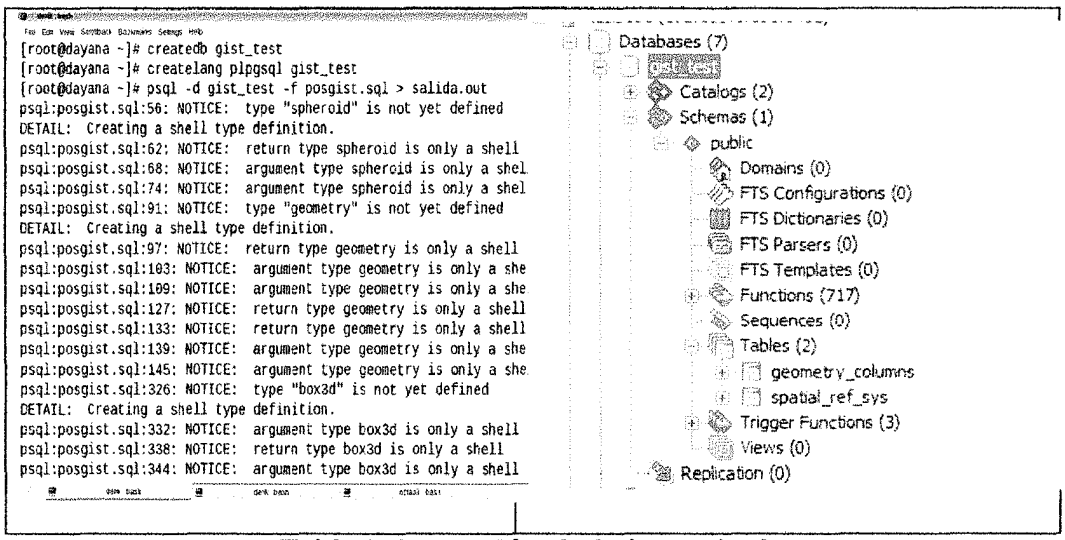


Tabla 5.1 creación de la base de datos.

5.1.2 Pruebas de caja negra y de integración

En esta etapa se realizaran las pruebas a nivel de usuario en referencia a la lista de requisitos funcionales que además prueban que la integración de los componentes funciona. De igual forma al ejecutar consultas utilizando el componente espacial y el componente temporal de forma separada probaremos el funcionamiento independiente en las tablas 5.7 y 5.8.

Prueba de Creación de tabla con atributo de tipo de dato espacio-temporal	**Usuario** - Crear Tabla - Agregar atributo con el tipos de dato espacio-temporal.

Referencia cruzada	R1.1
Prueba	**Descripción**
Creación de tabla con atributos espacio-temporales	Después de crear la base de datos con el prototipo Postgist precargado probar la creación de tablas con atributos de tipo espacio-temporal.
Resultados Esperados	**Resultados Obtenidos**
Tabla creada	Se creó exitosamente la tabla spatiottable en la base de datos gist_test y la columna spatiotcolumn se agrego exitosamente.

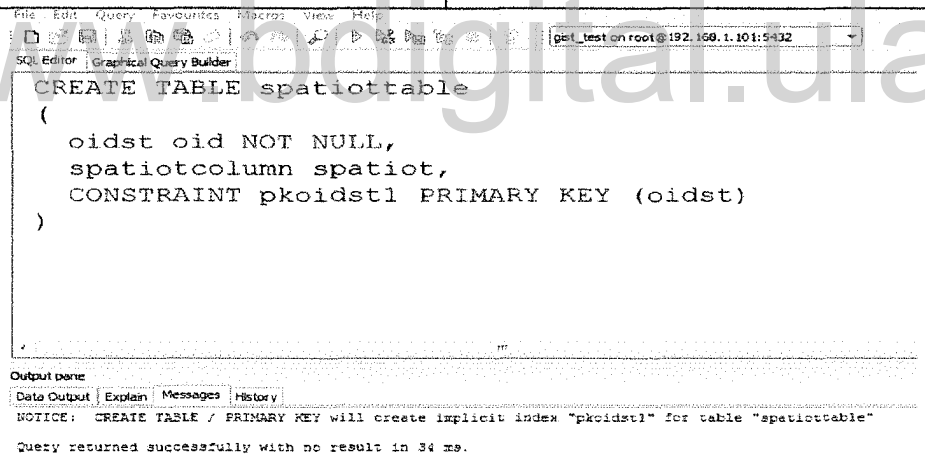


Tabla 5.2 creación tabla con atributo espacio-temporal.

Prueba de creación de índices de acceso.	**Usuario** - Crear índices al atributo espacio-temporal
Referencia cruzada	R1.2
Prueba	**Descripción**
Creación de índices a la parte temporal y a la parte espacial del atributo espacio-temporal.	Después de crear la tabla spatiottable con la columna spatiotcolumn de tipo espacio-temporal se procede a crearle los índices respectivos a la parte temporal y a la parte espacial.
Resultados Esperados	**Resultados Obtenidos**
Índices Creados	Se creó exitosamente el índice espacial y el índice temporal la tabla spatiottable en la base de datos gist_test base de datos y la columna spatiotcolumn se agregó exitosamente.

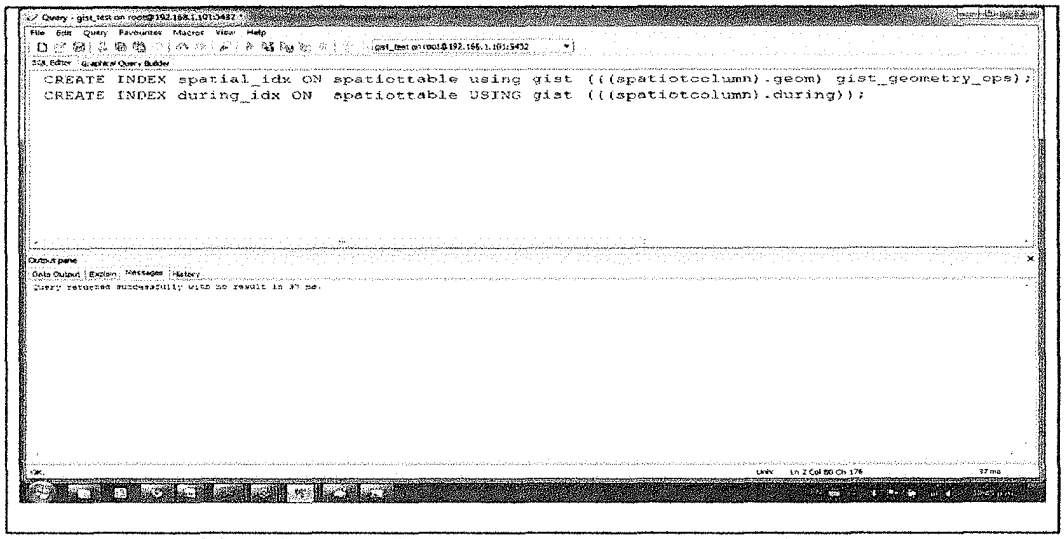


Tabla 5.3 creación tabla con atributo espacio-temporal.

Prueba de inserción de un tipo espacio-temporal.	**Usuario** - Inserta en la base de datos en una tabla con atributos espacio-temporales.
Referencia cruzada	R1.3
Prueba	**Descripción**
Inserción de una figura de tipo poligonal de postgis con tiempo de existencia	Se intenta insertar un polígono cuyos puntos delimitadores son POLYGON((85 191,197 173,75 60,85 191)) un periodo abierto desde 2008-01-01 14:00:00 hasta '2010-01-01 16:00:00'

Resultados Esperados	**Resultados Obtenidos**
El dato espacio-temporal insertado	Se insertó correctamente.

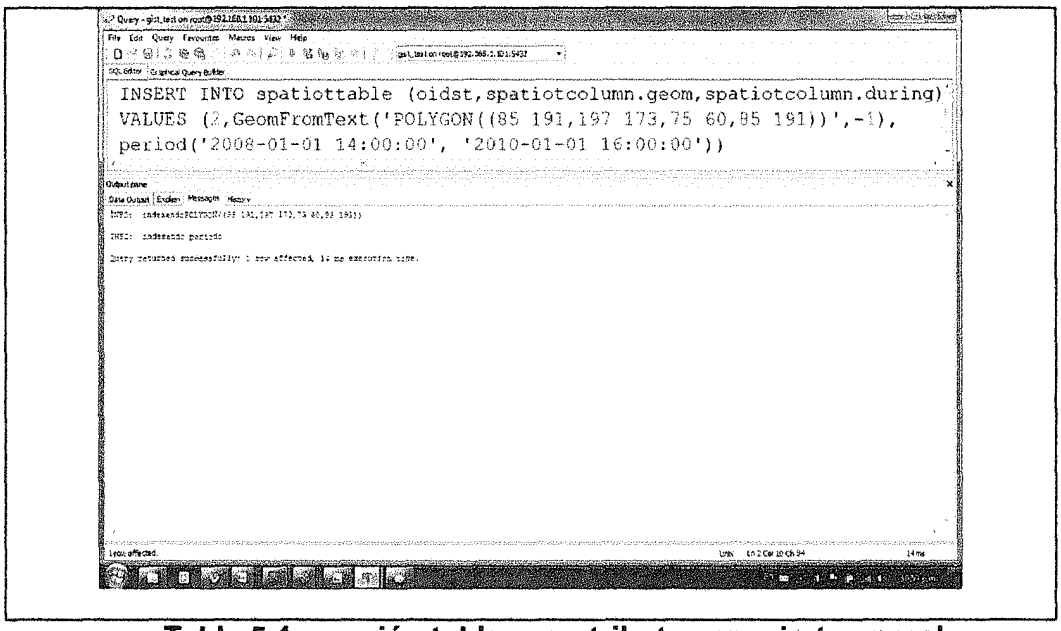


Tabla 5.4 creación tabla con atributo espacio-temporal.

Prueba de Modificación de un tipo espacio-temporal.	**Usuario** - Modifica en la base de datos en una fila con atributos espacio-temporales.
Referencia cruzada	R1.4
Prueba	**Descripción**
Modificación de una figura de tipo poligonal	Se intenta modificar un polígono cuyos puntos delimitadores son POLYGON((85 191,197 173,75 60,85 191)) un periodo abierto desde 2008-01-01 14:00:00 hasta '2010-01-01 16:00:00' para cambiar su coordenada inicial a 82.
Resultados Esperados	**Resultados Obtenidos**

El dato espacio-temporal fue modificado	Se modificó correctamente.

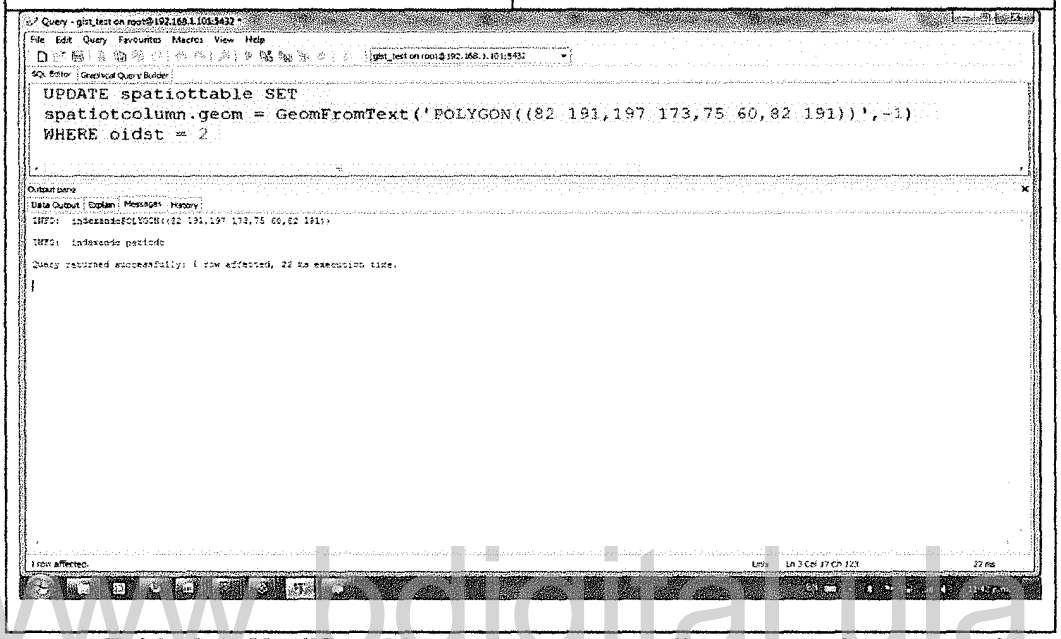


Tabla 5.5 Modificación de datos con atributo espacio-temporal.

Prueba de Eliminación de un tipo espacio-temporal.	**Usuario**- Elimina en la base de datos en una fila con atributos espacio-temporales.
Referencia cruzada	R1.5
Prueba	**Descripción**
Eliminación de una figura de tipo poligonal	Se intenta modificar un polígono cuyos puntos delimitadores son POLYGON((85 191,197 173,75 60,85 191)) un periodo abierto desde 2008-01-01 14:00:00 hasta '2010-01-01 16:00:00' oid = 2
Resultados Esperados	**Resultados Obtenidos**

El dato espacio-temporal fue eliminado	Se elimina correctamente.

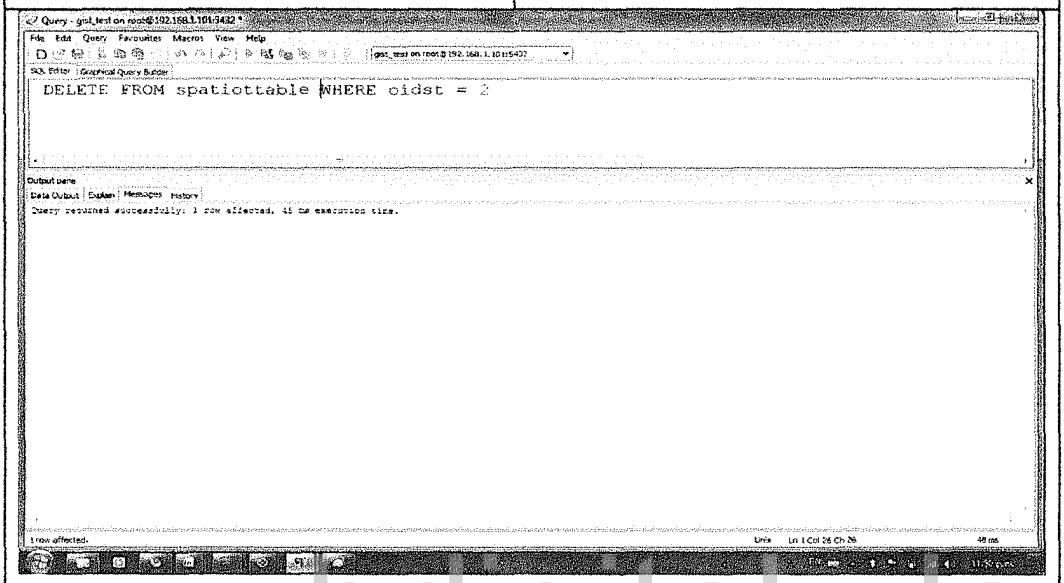


Tabla 5.6 Eliminación de la base de datos espacio-temporal

Prueba de Consultas de tipo espacial.	**Usuario** - Consulta en la base de datos en una fila con atributos espacio-temporales considerando la parte espacial.
Referencia cruzada	R1.6
Prueba	**Descripción**
Consulta de una unión donde las figuras sean geométricamente iguales.	Se intenta consultar utilizando la función de igualdad para las figuras geométricas de los atributos espacio-temporales.

Resultados Esperados	Resultados Obtenidos
Resultado con los objetos geométricamente iguales.	Se ejecutó correctamente.

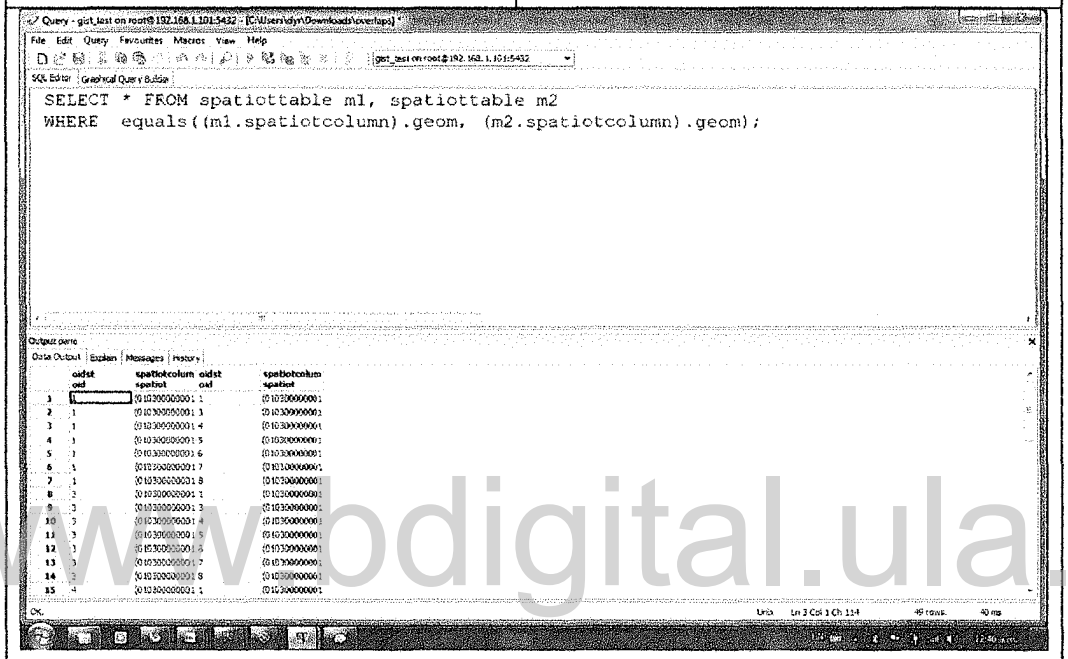


Tabla 5.7 Consulta espacial a la base de datos espacio-temporal

Prueba de Consultas de tipo temporal.	Usuario - Consulta en la base de datos en una fila con atributos espacio-temporales considerando la parte temporal.
Referencia cruzada	R1.7
Prueba	**Descripción**
Consulta de una de tipo temporal	Se consulta la unión de una tabla con ella misma luego de haber insertado una serie de datos espacio-temporales

	que se cuyos periodos de existencia se solapan.
Resultados Esperados	**Resultados Obtenidos**
La consulta retorne los datos esperados.	Se consultó correctamente en tiempos aceptables.

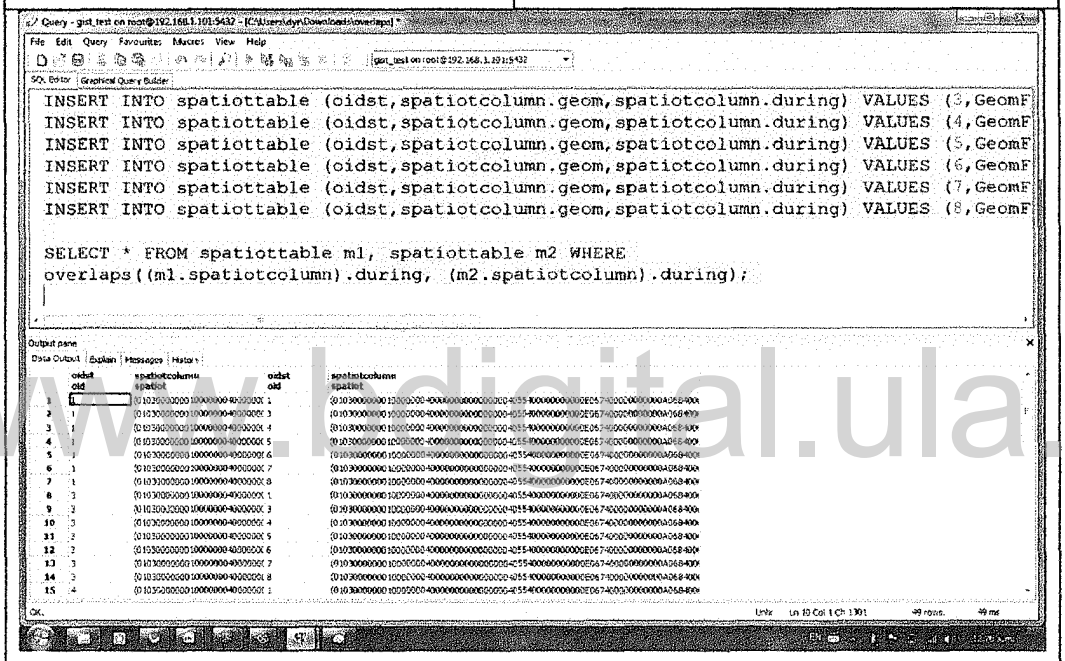


Tabla 5.8 Consulta temporal a la base de datos espacio-temporal

Prueba de Consultas de tipo espacio-temporal.	**Usuario** - Consulta en la base de datos en una fila con atributos espacio-temporales considerando el tipo de dato espacio-temporal homogéneamente.
Referencia cruzada	R1.8
Prueba	**Descripción**

Consulta de una de tipo espacio-temporal	Se consulta la unión de una tabla con ella misma luego de haber insertado una serie de datos espacio-temporales y se consulta por el solapamiento espacio-temporal.
Resultados Esperados	**Resultados Obtenidos**
La consulta retorne los datos esperados.	Se consultó correctamente en tiempos aceptables.
	

Tabla 5.9 Consulta temporal a la base de datos espacio-temporal

5.1.3 Pruebas caja blanca

En esta sección se muestran las pruebas de algunos de los métodos internos fundamentales para los mecanismos de indexación de los tipos de dato espacio-temporales y algunos operadores para el tipo espacio-temporal.

La estrategia este tipo de pruebas se aplica utilizando la función de log visualizando la salida de ***PostgreSql*** en la consola con la impresión de ***NOTICE*** en los métodos que hacen funcionar el árbol generalizado de búsqueda ***GIST***.

Se realizó un generador de datos pseudo-aleatorios que siguen una distribución uniforme validados para la inserción de datos masivamente por medio de un programa cuya implementación está realizada en el lenguaje C++ y se ilustra a continuación.

```
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
using namespace std;
//fechas validas
bool esValida(int year, int month, int day)
{
            if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0)
               if( month == 2 && day>28)
                   return false;
            if( month == 2 && day>28)
                return false;
            return true;

}
//validación del inicio y fin de los intervalos
bool esMayor(int y1, int y2,int m1,int m2,int d1,int d2){
```

```
if(y1>y2)
      return true;
if(y1==y2 && m1>m2)
return true;
if(y1==y2 && m1==m2 && d1>d2)
return true;
return false;
}
int main(){
  int i;
  //100000 numero de iteraciones para generar la data
  for(i=0; i<100000; i++)
    {
//srand ( time(NULL) );
//numeros aleatorios para generar la data de prueba uniformemente
           int init1, init2, anio, mes, dia, anio1, mes1, dia1, hora,
minuto, segundo, hora1, minuto1, segundo1;
           init1 = rand()/10000000;
           init2 = rand()/10000000;
           anio = rand() % 30 + 1985;
           mes = rand()% 10 + 1 ;
           dia = rand() % 30 + 1;
           anio1 = rand() % 30 + 1985;
         mes1 = rand()% 10 + 1 ;
           dia1 = rand() % 30 + 1;
           hora = rand() % 24;
           minuto = rand() % 60;
           segundo =    rand() % 60;
```

```
            hora1 = rand() % 24;

            minuto1 = rand() % 60;

            segundo1 =  rand() % 60;


if((esValida(anio,mes,dia) && esValida(anio1,mes1,dia1) ) &&
esMayor(anio1,anio,mes1,mes,dia1,dia))

      {

//cadena de insercion

//INSERT INTO spatiottable1
(oidst,spatiotcolumn1.geom,spatiotcolumn1.during) VALUES
(1,GeomFromText('POLYGON((85 191,197 173,75 60,85 191))',-1),
period('2008-01-01 14:00:00', '2009-01-01 16:00:00'))


cout<<"INSERT INTO spatiottable (oidst,
spatiotcolumn.geom,spatiotcolumn.during) VALUES ('"<<rand() <<"',
GeomFromText('POLYGON(("<<init1<<" "<<init2<<","<<rand()/10000000<<"
"<<rand()/10000000<<","<<rand()/10000000<<"
"<<rand()/10000000<<","<<init1<<" "<<init2<<"))',-1)";


cout<<",period('"<<anio<<"-"<<mes<<"-"<<dia<<"
"<<hora<<":"<<minuto<<":"<<segundo <<"', '";


cout<<anio1<<"-"<<mes1<<"-"<<dia1<<"
"<<hora<<":"<<minuto<<":"<<segundo <<" '));"<<endl;

      }

      }

return 0;


}
```

A continuación se muestra un subconjunto de la data generada con el programa ilustrado anteriormente para las pruebas de indexación.

```
INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('13156340227', GeomFromText('POLYGON((204 196,180 110,46
214,204 196))',-1),period('1985-7-23 15:9:2', '2001-2-9 15:9:2 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('1159126505', GeomFromText('POLYGON((197 14,57 182,193
208,197 14))',-1),period('1985-4-17 9:2:50', '1986-1-27 9:2:50 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('1780695788', GeomFromText('POLYGON((80 163,188 94,35
147,80 163))',-1),period('1998-8-25 14:27:14', '2000-3-16 14:27:14 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('1975960378', GeomFromText('POLYGON((70 49,204 46,103
195,70 49))',-1),period('1996-5-20 0:19:52', '1997-1-27 0:19:52 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('1373226340', GeomFromText('POLYGON((125 52,207 140,157
171,125 52))',-1),period('1991-6-17 16:51:32', '2008-10-15 16:51:32 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('116087764', GeomFromText('POLYGON((163 20,214 26,191
43,163 20))',-1),period('2008-9-26 11:59:28', '2011-2-2 11:59:28 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('653468858', GeomFromText('POLYGON((186 15,162 113,21
147,186 15))',-1),period('2002-7-4 3:16:52', '2006-8-26 3:16:52 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('1782436840', GeomFromText('POLYGON((213 123,35 106,143
211,213 123))',-1),period('1992-6-14 3:48:9', '1993-9-4 3:48:9 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('1544617505', GeomFromText('POLYGON((190 16,117 60,208
109,190 16))',-1),period('2003-9-10 20:3:30', '2012-8-7 20:3:30 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('434248626', GeomFromText('POLYGON((103 103,159 70,156
134,103 103))',-1),period('1988-7-9 10:45:1', '1995-3-12 10:45:1 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('824272813', GeomFromText('POLYGON((143 62,147 26,150
164,143 62))',-1),period('1993-2-4 13:14:40', '1996-2-11 13:14:40 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('260401255', GeomFromText('POLYGON((58 185,107 47,38
155,58 185))',-1),period('2003-8-3 9:56:50', '2013-3-10 9:56:50 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('882160379', GeomFromText('POLYGON((104 33,29 192,72
211,104 33))',-1),period('1995-3-9 12:50:29', '2008-9-21 12:50:29 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('1795519125', GeomFromText('POLYGON((178 15,134 167,55
7,178 15))',-1),period('1991-9-10 5:29:10', '2005-7-19 5:29:10 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('1144522535', GeomFromText('POLYGON((114 33,1 138,66
149,114 33))',-1),period('2001-3-15 23:2:48', '2012-10-14 23:2:48 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('12260289', GeomFromText('POLYGON((181 132,0 121,150
132,181 132))',-1),period('1987-6-28 12:5:49', '2009-10-20 12:5:49 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('847228023', GeomFromText('POLYGON((65 56,41 194,145
178,65 56))',-1),period('1997-6-24 18:20:21', '2009-10-11 18:20:21 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('696947386', GeomFromText('POLYGON((151 186,200 213,141
127,151 186))',-1),period('1994-1-16 18:30:46', '2011-8-18 18:30:46 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('1369356620', GeomFromText('POLYGON((88 131,82 174,10
109,88 131))',-1),period('1992-8-3 8:50:12', '2001-5-26 8:50:12 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('427355115', GeomFromText('POLYGON((55 163,80 23,185
130,55 163))',-1),period('2004-8-20 10:13:7', '2014-4-23 10:13:7 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('1269400346', GeomFromText('POLYGON((43 142,40 173,63
161,43 142))',-1),period('1991-4-24 11:17:42', '2009-5-4 11:17:42 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('659639006', GeomFromText('POLYGON((36 125,86 112,44
74,36 125))',-1),period('1993-1-25 21:49:33', '1996-2-4 21:49:33 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('537322532', GeomFromText('POLYGON((146 138,182 113,143
69,146 138))',-1),period('2004-1-8 12:21:25', '2011-4-28 12:21:25 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('1089653714', GeomFromText('POLYGON((85 15,176 152,62
104,55 15))',-1),period('2006-1-15 16:3:32', '2007-8-25 16:3:32 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('372160269', GeomFromText('POLYGON((102 162,59 160,20
173,102 162))',-1),period('1997-4-12 16:42:54', '2013-4-15 16:42:54 '));
```

```
INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('1891252715', GeomFromText('POLYGON((201 163,24 141,29

:28:26', '2010-1-22 13:28:26 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('1658698394', GeomFromText('POLYGON((122 98,116 21,166
118,122 98))',-1),period('1995-6-12 1:59:32', '2008-1-2 1:59:32 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('618250231', GeomFromText('POLYGON((97 165,49 164,182
84,97 165))',-1),period('1999-6-20 22:54:21', '2001-5-20 22:54:21 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('1632730758', GeomFromText('POLYGON((207 52,126 161,29
61,207 52))',-1),period('2006-6-3 1:15:56', '2009-4-6 1:15:56 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('150238230', GeomFromText('POLYGON((93 138,53 97,201
183,93 138))',-1),period('1989-8-22 13:57:34', '2008-10-9 13:57:34 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('2124236872', GeomFromText('POLYGON((159 46,46 146,73
84,159 46))',-1),period('2000-7-4 10:38:45', '2007-2-21 10:38:45 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('1436796072', GeomFromText('POLYGON((89 146,86 57,180
66,89 146))',-1),period('2006-2-6 18:36:12', '2010-10-21 18:36:12 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('322044317', GeomFromText('POLYGON((141 102,12 96,29
11,141 102))',-1),period('1985-1-21 19:24:35', '2003-8-26 19:24:35 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('998171884', GeomFromText('POLYGON((161 185,142 88,33
39,161 185))',-1),period('1991-7-13 6:45:37', '1993-9-2 6:45:37 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('1703610268', GeomFromText('POLYGON((177 3,105 213,178
94,177 3))',-1),period('1994-10-5 3:51:47', '2004-1-6 3:51:47 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('768356649', GeomFromText('POLYGON((124 105,169 192,167
138,124 105))',-1),period('1990-2-22 2:26:4', '2005-10-16 2:26:4 '));

INSERT INTO spatiottable (oidst, spatiotcolumn.geom,spatiotcolumn.during) VALUES ('1886873446', GeomFromText('POLYGON((159 27,140 173,29
54,159 27))',-1),period('1985-7-3 2:27:55', '1989-6-21 2:27:55 '));
```


Y para completar la prueba de las funcionalidades de la indexación de los objetos en la cual análogamente con la salida de por consola los ***INFO*** constatan el funcionamiento de la indexación en la ***tabla 5.9.***

Prueba de indexación espacial y temporal de un objetos espacio-temporales.	**Posgist** - Indexa y ejecuta los métodos de indexación tales como los picksplit, compress, calcula los bouding box cuando es necesario.

Referencia cruzada	R1.8
Prueba	**Descripción**
En la inserción se deben ejecutar los métodos GiST	Se inserta un polígono.
Resultados Esperados	**Resultados Obtenidos**
Salida por consola de los INFO esperados a saber: Indexando geometría	Se presentaron los INFO esperados

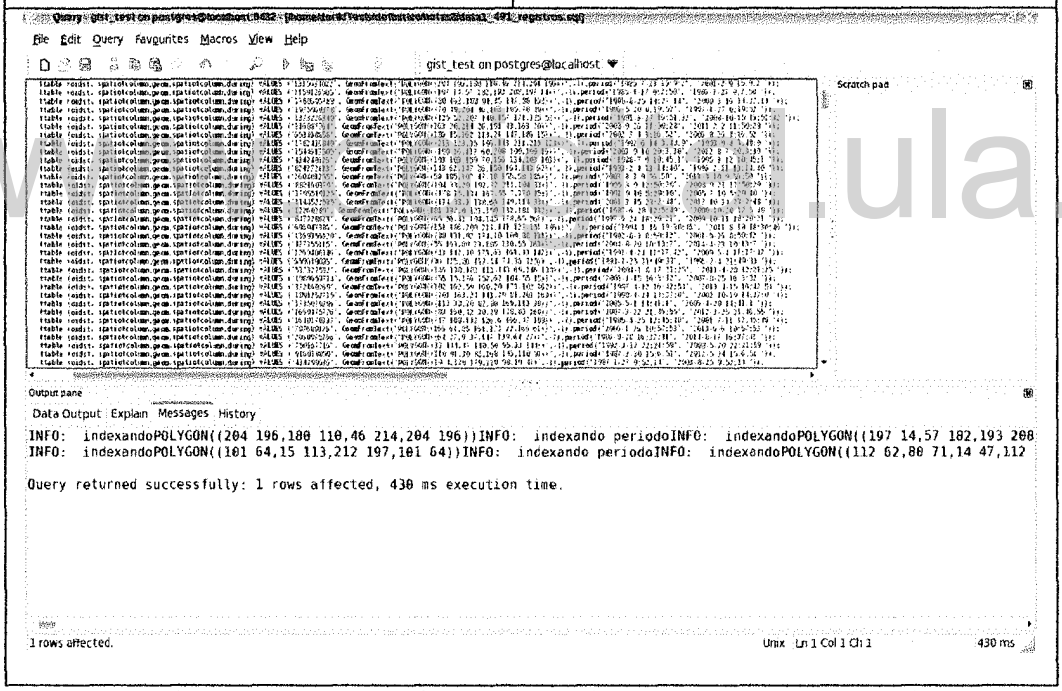


Tabla 5.10 Indexación de los objetos espacio-temporales.

CAPÍTULO VI

Conclusiones y Recomendaciones

Conclusiones

El desarrollo de prototipos para base de datos espacio-temporales en manejadores de base de datos *open source* cobra gran importancia debido a la creciente evolución de las tecnologías de alta movilidad y localización como lo son **GPS** (*Global Position System*) o Sistema de Posicionamiento Global. El sistema manejador de base de datos ***PostgreSql*** ha presentado una gran evolución debido a que la comunidad de software libre le ha dedicado muchos esfuerzos para lograr mantenerlo en la vanguardia tecnológica de las herramientas del dominio de los sistemas manejadores de base de datos, pero aún no cuenta con un manejador oficial para base de datos espacio-temporales razón por la cual se realizó el prototipo ***Posgist*** que incluye toda las funcionalidades del componente oficial para manejo de datos espaciales en ***PostgresSql*** denominado ***PostGis*** el cual está basado en estándares, es uno de los componentes más completos y de notable madurez, lo cual aporta que al sumar componentes con cualidades como las que mencionamos de ***PostGis*** podría hacer factible que el prototipo aquí desarrollado forme parte a futuro de una versión de ***PostgreSql*** que incluya el manejo para base de datos espacio-temporales.

Los objetivos planteados durante la investigación fueron cumplidos satisfactoriamente realizando el producto final como fue propuesto desde un principio, con la salvedad de un cambio en el enfoque de los mecanismos de

indexación basados en hiper-rectángulos para proveer un mecanismo de indexación, el cual fue suplantado por una separación interna de los atributos temporales y los espaciales, resaltando que la separación no es notable a nivel de usuario final, debido a que los operadores que ejecutan las consultas a los índices operan con interfaces que tratan el objeto espacio-temporal homogéneamente. Dicho cambio de enfoque en los mecanismos de indexación se realizó para acortar los tiempos en el proceso de desarrollo debido a que durante una de las iteraciones de implementación se observó que la forma en que se realiza el pase de argumentos entre las funciones definidas en el lenguaje ***C*** con ***PostgreSql*** no contemplan la tupla completa, entonces, para el enfoque de crear el tipo de dato espacio-temporal en dos campos separados no funcionaba de manera estándar con el código de ***PostgreSql*** por ésta razón se decidió implementar con un tipo de dato compuesto que separa la parte espacial y temporal y hacer la manipulación de los índices de forma separada también.

En el capítulo I se pudo observar que existían fundamentos que avalan la implementación desde el punto de vista de necesidades en cuanto al tema espacio-temporal se refiere, Se plantean los objetivos generales y específicos en función a las necesidades básicas que debe tener un componente para almacenar y recuperar datos espacio-temporales, que a lo largo de los capítulos siguientes evolucionan y son afectados en su mayoría en las metas de indexación resaltando que lejos de ser un problema de una meta no cumplida se convierte en un aporte debido a que el planteamiento de la tendencia teórica más fuerte para la indexación de datos de esta naturaleza es la de los hiper-rectángulos.. En el

capítulo II se esquematizan las bases conceptuales por las cuales se selecciona el manejador ***PostgreSql*** y se ampara el futuro diseño con algunas bases teóricas como los son las estructuras de indexación, las bases de datos espaciales y las bases de datos temporales así como también se idealiza el tipo de dato espacio-temporal del prototipo denominado "spatiot" para conllevar a un diseño refinado planteado en el capítulo III y finalmente los capítulos IV y V muestran la implementación, forma de uso y pruebas del componente Posgit basado en el diseño planteado.

Recomendaciones

- Implementar modelos de la vida real utilizando el prototipo a fin de probar el desempeño con gran cantidad de datos. El objetivo principal del proyecto se basa en que el prototipo se realizaría de forma genérica aportando las interfaces necesarias para la gestión de información espacio-temporal por ende las pruebas realizadas avalan que las interfaces funcionan de la forma como estaba estipulado pero no evalúan su desempeño ante datos masivos.
- Recomendamos el uso de metodologías como la W_Watch. En la implementación de prototipos el uso de metodologías enfocadas a producir software sencillo y pequeño con documentación precisa, ayudan a disminuir la elaboración detallada de documentos y/o especificaciones y así dedicar más tiempo a actividades de implementación e implantación de versiones operativas y evolutivas del producto.

- Al implementar prototipos de componentes que extiendan a PostgreSql el desarrollo del este proyecto arroja la recomendación de diseñar en primera instancia con un esquema basado en un nivel de acoplamiento al manejador de base de datos débil, es decir, PostgreSql al ser de código abierto proporciona la posibilidad de ser extendido y recompilado para que el catálogo de tipos de datos y funciones sean adaptados a medida de las necesidades, esto según Ana Aguilera y Leonid Tineo, representa generalmente una ganancia a nivel de desempeño debido a que existe un nivel de acoplamiento fuerte al sistema manejador de base de datos. Pero PostgreSql también brinda algunos lenguajes de programación para definir funciones a la medida que pueden ser cargadas dinámicamente al momento de necesitarlas sin necesidad de recompilar el manejador tales como PLPGSQL, C, SQL, que por el hecho de brindar un nivel de acoplamiento más débil posiblemente presentan un desempeño inferior al acoplamiento fuerte pero aporta facilidades y disminuye los tiempos de desarrollo por ende es recomendable comenzar a desarrollar esta clase de prototipos en un nivel de acoplamiento bajo.
- En el prototipo *Posgist* la definición de los operadores que implementa las relaciones espacio-temporales se priorizó la evaluación de las condiciones temporales de existencia de los objetos antes de evaluar las condiciones espaciales por ende el prototipo se comporta de forma óptima entonces cuando los datos varían más en factor tiempo que en su forma. Si se diera el caso de objetos que cuyo atributo geométrico varía más que el atributo

temporal sería conveniente invertir la forma en que se evalúan las condiciones de los operadores espacio-temporales a nivel de las funciones del catálogo.

- Se recomienda medir el desempeño del componente con herramientas que ya lo posean de tipo propietario y hacer una comparación en los mecanismos de indexación. Esto debido a que a indexación en el prototipo se realiza separando los atributos temporales y espaciales y no de forma multidimensional como se plantea en estudios realizados anteriormente.

REFERENCIAS

[1] Isabel Besembel y Stuart Roberts, "***Indexación de objetos espacio-temporales"***. Pág 2-4,1998.

[2] Paul M. Aoki. "***Implementation of Extended Indexes in POSTGRES"***, Pág 2-9, 1991, ACM NY USA, V-25

[3] JF. Naughton y A. Pfeffer. "***Árboles generalizados de búsqueda para sistemas de bases de datos, para el VLD"***, Pág 2, 1995.

[4] J. Barrios y J. Montilva. "***W Watch: Método White Watch para el desarrollo de Proyectos Pequeños de Software"***. Pág 2-5, 2009.

[5] ***"Spatiotemporal database"***, Recuperado de el 11 de marzo de 2009 de http://en.wikipedia.org/wiki/Spatio-temporal_database

[6] María G. Dorzán, Edilma O. Gagliardi, Natalia J. Ortiz, Gilberto A. Gutiérrez Retamal. ***Procesamiento de consultas espacio-temporales complejas sobre diferentes escenarios.*** Pág 2., año 2007, Chile

[7] *Isabel Besembel Carrera y Jonás Montilva Calderón,* "***Modelado de relaciones espacio temporales en aplicaciones orientadas por objetos"***, Pág 2-5, 1998, Mérida-Venezuela.

[8] Isabel Besembel y Stuart Roberts, "***Indexación de objetos espacio-temporales"***. Pág 4-5,1998.

[9] Gómez, J.; Gutiérrez G.; Gagliardi E.; Dorzán, M., Carrasco, F. y García, J. "***Métodos de Acceso Espacio-Temporal."*** , 2005, Chile.

[10] María G. Dorzán, Edilma O. Gagliardi, Natalia J. Ortiz, ***"Procesamiento de consultas espacio-temporales complejas sobre diferentes escenarios"***, Pág 2-3, Argentina.

[11] Antonin Guttman, "***R-Trees: A Dynamic Index Structure for Spatial Searching".***, Pág 47-57, 1984, ACM Press, Boston.

[12] Wang, X., Zhou, X. y Lu, S. (2000). ***Spatiotemporal data modeling and management: a survey. Proceedings 36th.***
International Conference on Technology of Object-Oriented Languages and System, pp. 202-211.

[13] ***"PostgreSQL"***, Recuperado el 08 de abril de 2009 de http://es.wikipedia.org/wiki/PostgreSQL.

[14] "***PostGIS"***, Recuperado el 12 de mayo de 2009 de http://postgis.refractions.net/

[15] ***"Formato Well-Known Binary (WKB)"***, Recuperado el 30 de mayo de 2009 de http://dev.mysql.com/doc/refman/5.0/es/gis-wkb-format.html

[16] Isabel M_ Besembel Carrera, ***"An integrated index structure for object oriented and spatio temporal information systems"***, Pág 57, 1998, U.K

[16] ***"GIST Implementation"***, Recuperado el 08 de septiemmbre de 2009 de http://www.postgresql.org/docs/8.4/static/gist-implementation.html