**Tesis de Licenciatura en Ciencias de la Computación**

# DESARROLLO DE UN LENGUAGE DE CONSULTA PARA INTEGRACION DE SISTEMAS DE INFORMACION GEOGRAFICA (GIS ) Y ON LINE ANALYTICAL PROCESSING (OLAP)

**Sebastián Javier Zich**
Agosto de 2012


Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Director:
**Alejandro Ariel Vaisman**

# RESUMEN

La presente tesis describe una propuesta de integración de sistemas de Información Geográfica (GIS) con sistemas OLAP (Online Analytical Processing). El modelo implementado está formado por información de aplicación, contenida en un data warehouse, e información geográfica, básicamente un mapa con elementos geométricos almacenados en diferentes capas. La información geométrica que compone el mapa se organiza en un conjunto de jerarquías que conforman lo que se denomina una dimensión GIS. Adicionalmente, los elementos de la parte geométrica se relacionan con información de aplicación almacenada externamente en un data warehouse. Este modelo soporta consultas que involucran agregación o sumarización de ambos tipos de información. Estas consultas se expresan tanto sobre la parte geométrica como sobre la información de aplicación, permitiendo navegar a posteriori los resultados con herramientas OLAP. Cabe destacar que en la actualidad no existe software comercial que implemente un modelo OLAP espacial como el propuesto en esta tesis, ni tampoco existe una solución para resolver consultas de agregación geométrica o que integre la información almacenada en sistemas GIS y OLAP.

*Piet* es un modelo de datos que integra datos espaciales y multidimensionales (i.e., GIS y OLAP). El presente trabajo, basado en Piet, presenta un nuevo lenguaje de consulta, Piet-QL, que soporta el modelo de datos. Piet-QL permite expresar consultas con la información geométrica filtrada por un cubo de datos (y viceversa) en forma elegante y relativamente sencilla. En la primera parte de este trabajo presentamos la sintaxis y semántica de Piet-QL, y proporcionamos ejemplos de su utilización. En la segunda parte del trabajo se presentan dos front-ends que permiten intereactuar con el motor de Piet-QL: (a) un asistente (Piet-QL Wizard) que permite la generación de consultas a través de una interfaz gráfica e intuitiva; (b) un cliente web que permite editar y ejecutar consultas Piet-QL.

# ABSTRACT

This thesis describes a proposal for integration of geographic information systems (GIS) with OLAP (Online Analytical Processing). The model implemented consists of application information contained in a data warehouse, and geographic information, basically a map with geometric elements stored in different layers. The geometrical information comprising the map is organized in a set of hierarchies which composes what is called a GIS dimension. Additionally, the geometric elements that are related to application information stored externally in a data warehouse. This model supports queries involving aggregation or summarization of both types of information. These queries are expressed both on the geometry and on the application information, allowing subsequent navigation of the resultsets with OLAP tools. Note that there is currently no commercial software that implements a spatial OLAP model as proposed in this thesis, nor is there any solution to solve geometric aggregation queries or that integrate information stored in GIS and OLAP systems.

Piet is a data model that integrates spatial and multidimensional data (i.e., GIS and OLAP). This paper, based on Piet, presents a new query language, Piet-QL, which supports the data model. Piet-QL can express queries with geometric information filtered by a data cube (and vice versa) in an elegant and relatively simple way. In the first part of this paper we presents the syntax and semantics of Piet-QL, and we provide some usage examples. In the second part of the paper we presents two front-ends that allow us to interact with the Piet-QL engine: (a) a wizard (Piet-QL Wizard) that allows us the generation of queries via an intuitive and graphical interface; ( b) a web client that allows you to edit and run Piet-QL queries.

# Índice de contenido

# CAPITULO 1

# INTRODUCCIÓN

En esta sección se describen las contribuciones de la tesis, y se introducen los conceptos básicos relacionados con los sistemas de información involucrados en este trabajo.

## 1.1 Sistemas de información geográfica (GIS)

Los Sistemas de Información Geográfica (GIS por sus iniciales en inglés, Geographic Information Systems) han sido ampliamente utilizados en diferentes aplicaciones, como por ejemplo, análisis económicos, análisis ecológicos, análisis demográficos, planificación de ciudades, y planificación de rutas. El National Center for Geographic Information and Analysis (NCGIA por sus iniciales en inglés) define a un GIS como "Un sistema de hardware, software y procedimientos para facilitar la gestión, manipulación, análisis, modelización y visualización, de la representación de datos georeferenciados para resolver problemas complejos en materia de planificación y gestión de recursos". La Figura 1 muestra un ejemplo de una aplicación GIS, ArcGis[1], donde se muestra el mapa de una parte de Europa.

[1] El ejemplo fue bajado de internet de la dirección: http://www.esri.com/software/arcgis/arcview/graphics/av_qualitymapping.gif

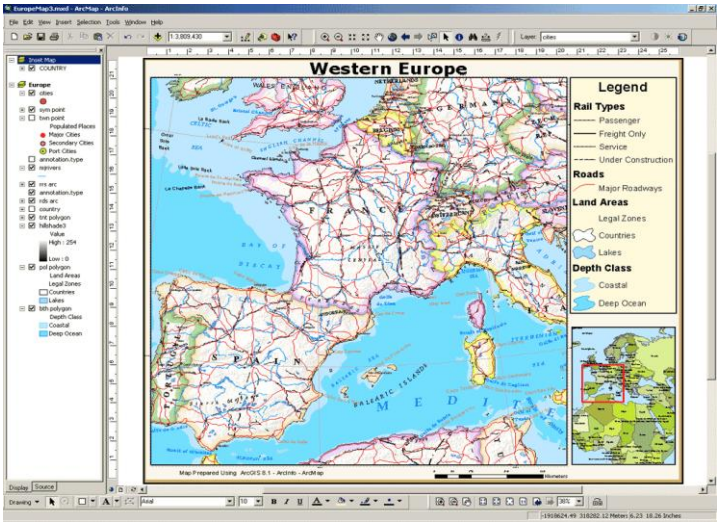


Figura 1: Una pantalla del software ArcGIS.

En GIS, los objetos geométricos de un mapa se organizan en capas temáticas (por ejemplo, las provincias en una capa, las ciudades en otra y los ríos en otra). Estos objetos espaciales pueden ser anotados con información numérica o categórica. Típicamente se consulta un GIS para recuperar objetos geométricos que cumplen alguna condición, la cual incluso puede expresar una agregación de medidas geográficas (por ejemplo, área, longitud, etc.).

Los objetos geométricos pueden almacenarse en diferentes estructuras de datos de acuerdo a los diferentes modelos de datos. Principalmente se utilizan dos modelos de datos para representación de la información espacial en una capa: el modelo vectorial y el modelo raster. La elección del modelo depende de la fuente de datos que se use para importar en el GIS. El modelo vectorial es el más utilizado en GIS. En este modelo, los conjuntos de puntos del espacio (infinitos) se representan como finitos en estructuras geométricas finitas, como puntos,

polilíneas y polígonos.

## 1.2 OLAP (On Line Analytical Processing)

OLAP (On Line Analytical Processing) comprende un conjunto de herramientas y algoritmos que permiten consultar, de manera eficiente e interactiva, grandes repositorios de datos multidimensionales, normalmente llamados data warehouses (DW). En el modelo multidimensional, los datos se pueden percibir como un cubo de datos, donde cada celda contiene una medida o un conjunto de medidas de interés (probablemente agregadas) que se analizan de acuerdo a un conjunto de dimensiones de interés. Estas dimensiones normalmente están organizadas en jerarquías que favorecen el proceso de agregación de datos. Existen tres formas típicas de implementar herramientas OLAP: MOLAP (Multidimensional OLAP), donde los datos se almacenan en estructuras de datos multidimensionales propietarias, ROLAP (Relacional OLAP), donde los datos se almacenan en bases de datos relacionales y HOLAP (Hybrid OLAP), que ofrece las dos soluciones en forma integrada. En un entorno ROLAP, los datos se organizan como un conjunto de tablas de dimensiones y tablas de hechos (fact tables). En el resto de este trabajo asumiremos esta organización, siguiendo el denominado Esquema Estrella.

Un conjunto de operaciones OLAP permite la explotación de las dimensiones y sus jerarquías, proporcionando un entorno interactivo para el análisis de datos. Los DW están optimizados para las operaciones OLAP, que habutualmente

implican la agregación o desagregación de datos a lo largo de una dimensión. Estas operaciones se denominan "roll-up" y "drill-down", respectivamente. Otras operaciones permiten seleccionar porciones del cubo (slice & dice) o reorientar la visión multidimensional de los datos (pivoting). Además de las operaciones básicas descritas anteriormente, estas herramientas proporcionan una gran variedad de operadores matemáticos, estadísticos y financieros para computar rankings, varianzas, etc.

## 1.3 Motivación y Contribuciones de la Tesis

Un Sistema de Soporte de Decisión (DSS por sus siglas en inglés) es una pieza de software destinada a ayudar al personal que toma las decisiones para recopilar información útil a partir de los datos brutos, documentos, conocimiento personal, y/o modelos de negocio. La mayoría de los DSS se basan en herramientas OLAP como las descritas anteriormente. Sin embargo, un DSS puede incluir otros tipos de herramientas de software, como por ejemplo, Sistemas de Información Geográfica.

Hoy en día, las organizaciones necesitan de DSSs cada vez mas sofisticados, que permitan combinar el análisis de datos agregados con operaciones geográficas. En este sentido, los proveedores de OLAP y GIS buscan permanentemente una mayor integración entre sus productos. Desde este punto de vista se pueden soportar consultas como *"Cantidad de visitantes, estadía promedio y promedio de gastos de hospedajes, agregados por tipos de personas, en destinos que*

*incluyen únicamente distritos intersecados por el río Dyle, en 2010".*

El desafío de integrar datos multidimensionales y espaciales se puede afrontar al menos con tres posibles enfoques: (a) modificar un sistema OLAP para proveer el almacenamiento y funcionalidad de un GIS; (b) modificar un GIS para proveer el almacenamiento y funcionalidad OLAP; (c) utilizar un motor intermedio con un conjunto de metadatos que relacione un sistema GIS puro con un sistema OLAP puro. A este modelo lo denominaremos débilmente acoplado, mientras que a los dos primeros los denominaremos fuertemente acoplados.

En este trabajo se describe el diseño e implementación de un motor débilmente acoplado que interactúa con un sistema OLAP y un GIS, basado en el modelo Piet [1]. Asimismo presentamos un lenguaje de consulta denominado Piet-QL, que permite expresar consultas sobre dicho modelo, de manera elegante y concisa. Piet-QL es compatible con los operadores propuestos por el Open Geospatial Consortium[2] para SQL. Además, incorpora la sintaxis necesaria para integrar las operaciones de OLAP, a través de un lenguaje similar a MDX4[3], el estándar de facto para OLAP.

Piet-QL admite cuatro tipos básicos de consultas: (a) consultas de tipo GIS; (b) consultas de tipo OLAP; (c) consultas GIS filtradas por una subconsulta de Piet-QL; (d) consultas de OLAP filtradas por una subconsulta de Piet-QL. Las subconsulta de Piet-QL pueden ser a la vez de cualquiera de los cuatro tipos

[2] http://www.opengeospatial.org
[3] http://msdn.microsoft.com/en-us/library/ms145506.aspx

anteriormente descritos. De esta manera la consulta *"Cantidad de visitantes, estadía promedio y promedio de gastos para hospedajes, por tipo de personas ofrecido en destinos en distritos intercecados por el río Dyle en 2007"* puede expresarse en Piet-QL como se muestra a continuación :

```
1   SELECT CUBE
2   [Measures].[visitors], [Measures].[stay],
3       [Measures].[expenditures] ON COLUMNS,
4                           [people].[all], [lodging].[all] ON rows
5   FROM [tourist]
6   WHERE [destination].[all] IN(
7                   SELECT GIS bel_dist
8                   FROM bel_river,bel_dist
9                   WHERE ST_INTERSECTS(bel_dist.the_geom,bel_river.the_geom)
10                  AND bel_river.name='Dyle'               )
11   SLICE [arrival].[2007]
```

El motor de Piet-QL es capaz de descomponer esta consulta en varias consultas spbre los sistemas OLAP y GIS subyacentes, emitir las consultas hacia sus respectivos motores y combinar los resultados para devolver al usuario.

Además del motor que interpreta Piet-QL se implementó un front-end que denominamos Piet-QL Wizard. Este es un asistente que permite la generación de consultas a través de una interfaz gráfica e intuitiva y permite generar consultas sin necesidad de conocer la sintaxis de Pietr-QL. El material desarrollado, así como información sobre el proyecto Piet, se puede encontrar el la siguiente URL: http://piet.exp.dc.uba.ar/pietql.

## 1.4 Organización de la Tesis

El presente documento se organiza de la siguiente manera. El Capitulo 2 se comentan los antecedentes y trabajos relacionados con el problema a tratar. En el Capitulo 3 se presenta la arquitectura y estructura del repositorio de datos de Piet-QL. En el Capitulo 4 presentamos el lenguaje de consulta Piet-QL describimos su sintaxis, semántica, y un conjunto de ejemplos que ilustran su uso. El detalle de los algoritmos y procedimientos más importantes se presenta en el Capitulo 5. En el Capitulo 6 presentamos la implementación de los clientes para PietQL, dos interfaces útiles para poder interactuar con el motor. El Capítulo 7 ilustra extensamente Piet-QL mediante ejemplos. Concluimos la tesis con el Capítulo 8.

# CAPITULO 2

## ESTADO DEL ARTE

Aunque algunos autores han señalado los beneficios de combinar GIS y OLAP, todavía queda mucho por hacer al respecto. Vega López et al. [2] presentan un estudio exhaustivo sobre la agregación espacio-temporal, que incluye una sección sobre agregación espacial. El trabajo de Bédard et al. [3] provee una revisión de esfuerzos para la integración entre OLAP y GIS. Rivest et al. [4] introducen el concepto de SOLAP (por Spatial OLAP), describiendo las propiedades y herramientas que un sistema SOLAP debería tener. Sin embargo, no presentan un modelo formal al respecto. Relacionado con el concepto de SOLAP, Shekhar et al. [5] introducen MapCube, una herramienta de visualización para cubos de datos espaciales. MapCube es básicamente, un operador que, dado un mapa base, las preferencias de cartografía y una jerarquía de agregación, produce un álbum de mapas que se puede navegar a través de operaciones roll-up y drill-down. Han et al. [6] utilizan técnicas OLAP para materializar objetos espaciales seleccionados, proponiendo un "Spatial Data Cube" que permiten realizar únicamente agregaciones sobre dichos objetos espaciales.

En cuanto a un lenguaje de consulta comparable con nuestro trabajo, Silva et al. [7] proponen GeoMDQL, un lenguaje de consulta para los entornos de SOLAP, basado en normas similares que Piet-QL. Esta propuesta, sin embargo, además

de carecer de un modelo formal, no trabaja sobre datos espaciales, sino sobre un data warehouse que contiene al menos una dimensión geográfica. Así, el lenguaje se reduce a una extensión de MDX con operadores geográficos. Escribano et al. [1] introducen el modelo de datos Piet, que incluye un lenguaje de consulta basado en este modelo de datos. Este lenguaje, llamado GISOLAP-QL, tiene una sintaxis muy sencilla. Una consulta de GISOLAP QL es de la forma:

Consulta GIS | Consultas OLAP

El pipe "|" separa dos secciones de consulta: una consulta de GIS y una consulta OLAP. La sección OLAP de la consulta se aplica a la parte de OLAP en el modelo de datos (es decir, los datos del warehouse) y está escrita en MDX. La parte GIS de la consulta tiene la típica forma SELECT FROM :

*SELECT lista de capas o mediciones*
*FROM Piet-Schema*
*WHERE operaciones geometricas;*

La parte de OLAP es básicamente una consulta MDX. El término Piet-Schema se refiere a los metadatos de Piet utilizados para responder la consulta. A partir de esta idea, hemos desarrollado un lenguaje de consulta mucho más expresivo, que llamamos Piet-QL. En los siguientes capítulos definiremos la sintaxis y semántica del lenguaje, y proporcionaremos un amplio conjunto de ejemplos.

# CAPITULO 3

# ARQUITECTURA

Esta sección describe la arquitectura de la solución partiendo de una visión conceptual, para luego describir la arquitectura concreta, finalmente implementada. También se incluye una descripción de los componentes de software que forman parte de la implementación y la estructura del repositorio de datos utilizado.

## 3.1 Arquitectura conceptual

PietQL es una librería o Back-end capaz de recibir consultas en lenguaje Piet-QL, y realizar las operaciones de análisis sintáctico, combinación y traducción para obtener los resultados directamente de los motores de GIS u OLAP, y pasar estos resultados al Front-end. La Figura 2 muestra un esquema de la arquitectura.

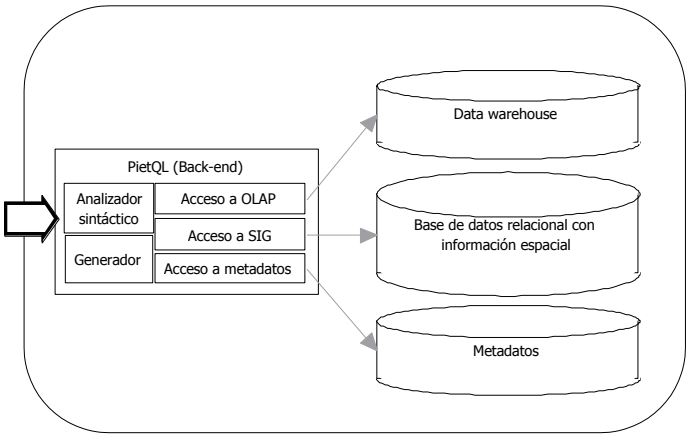


Figura 2: Esquema de arquitectura general de PietQL

Se desarrolló en JAVA y brinda un conjunto de funciones que permiten enviar al back-end consultas en lenguaje Piet-QL, y obtener los resultados correspondientes. Al ejecutar la función de traducción "RunTranslator(String query)" la librería invoca a una función de análisis sintáctico y a partir de ésta se obtienen las consultas y subconsultas. El proceso de ejecución comienza desde la subconsulta más interna y se propaga hacia la consulta general o más externa. Cada subconsulta se traduce al lenguaje requerido por los motores de GIS u OLAP, a través de las funciones internas de acceso a GIS y OLAP respectivamente. Con los resultados obtenidos se van completando las traducciones de las subconsultas externas. Para pasar de un motor a otro, es decir, de GIS a OLAP y viceversa, el programa utiliza metadatos que permiten utilizar los resultados de una consulta interna como filtro de una consulta externa. Estos metadatos contienen información sobre la asociación entre tablas y campos de GIS y OLAP.

## 3.2 Implementación de la Arquitectura

La solución implementada fue diseñada a partir de la arquitectura conceptual y las definiciones que se presentarán en el Capítulo 4. A partir de la función principal de traducción "RunTranslator(String query)" el programa llama a una función de parsing que procesa las consultas y subconsultas desde la más interna hacia la más externa, tal como se explicó anteriormente. Cuando un subquery es utilizado como filtro de una consulta o subconsulta más externa, se

utilizan dos funciones: "GIS_SQL_SubQuery" y "OLAP_SQL_SubQuery". Estas funciones encuentran la relación entre la subconsultas interna y externa, utilizando metadatos. Así, por ejemplo, si la consulta superior es de tipo GIS, y la subconsulta de OLAP (a través de los metadatos) "conoce" que hay que filtrar las geometrías correspondientes a los identificadores 17, 23 y 24, la consulta externa que se genere contendrá una condición en el WHERE de la forma "ID in (17,23,24)" , donde (17,23,24) es el resultado de la subconsulta OLAP. En el caso que la consulta externa sea de tipo OLAP, se procede análogamente. En el Capítulo 4 se detallan estos mecanismos.

Finalmente la librería provee algunas funciones para trabajar con el resultado de una traducción. El resultado de una traducción es una única consulta, expresada en un lenguaje de consulta soportado por los motores GIS u OLAP, según sea el tipo de la consulta más externa. La consulta finalmente será ejecutada sobre el motor correspondiente.

Alguna de las funciones provistas son:

getError: Permite obtener el error de una traducción no exitosa.

getTranslation: Permite obtener la traducción, solo si esta fue exitosa.

hasErrors: Permite consultar si la traducción fue exitosa.

isGisQuery: Indica, en caso de ser verdadero, que la traducción está expresada para ejecutarse en un motor GIS.

IsOLAPQuery: Indica, en caso de ser verdadero, que la traducción está expresada para ejecutarse en un motor OLAP.

RunGisTranslation: Permite ejecutar la consulta contra el servidor GIS en caso de ser una traducción expresada para GIS. Las traducciones de consultas OLAP se ejecutan directamente sobre el servidor Mondrian[4] para poder navegar los resultados con las mismas herramientas de dicho servidor OLAP.

## 3.3 Estructura de repositorio de datos PietQL

La estructura del repositorio de datos de PietQL está compuesta por una parte GIS y otra OLAP. Ambos repositorios son "puros" de su propia tecnología e independientes entre sí. La relación entre ambas estructuras propietarias se materializa mediante una estructura de metadatos.

La primera parte de los metadatos corresponde a un archivo de configuración que explica como leer los metadatos. Este archivo se denomina FoodMart.piet.xml[5].

La seguna parte de los metadatos lo componen las tablas de relación. Estas básicamente relacionan una capa o layer de GIS con una tabla de un nivel de una dimensión de OLAP.

La Figura 3 especifica el funcionamiento de los metadatos en el proceso de relación entre GIS y OLAP.

[4] http://mondrian.pentaho.com/
[5] Se mantuvo ese nombre y el formato por compatibilidad con otras versiones de Piet.

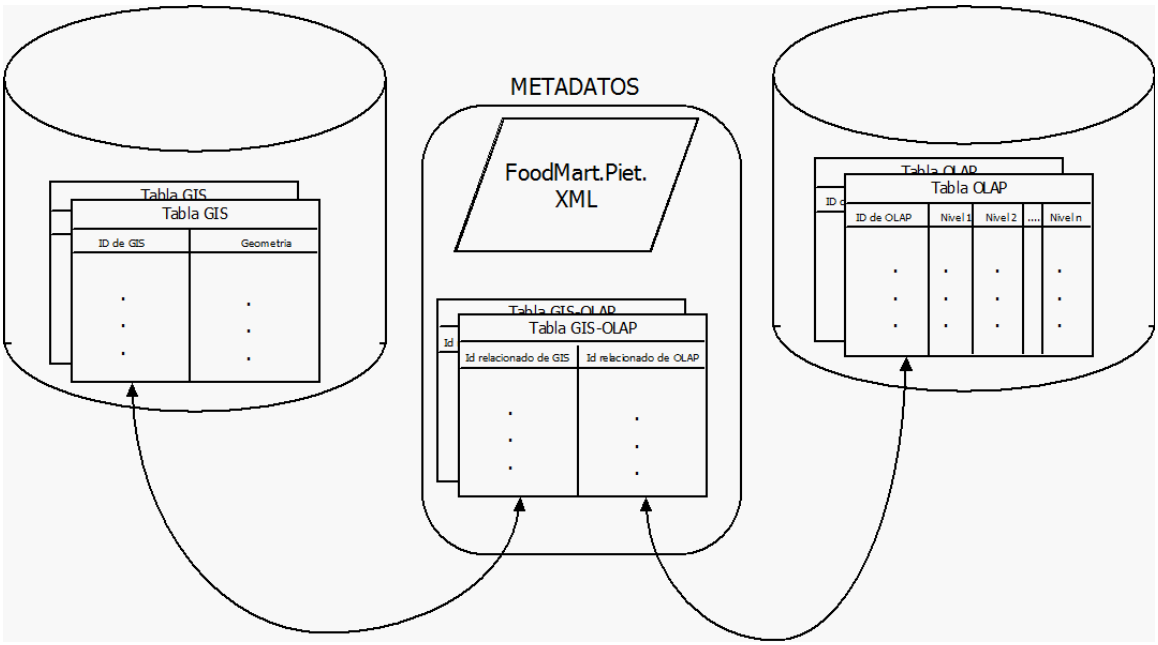


Figura 3: Relación entre metadatos, tablas de GIS y tablas de OLAP

El archivo xml contiene básicamente la información de cómo leer los metadatos y las tablas de las partes de GIS y OLAP. Por cada relación establecida contiene un tag "Layer" y cada uno de estos tags tienen la siguiente información :

- Tabla GIS (Propiedad:Name): Nombre de la tabla correspondiente a la capa de GIS relacionada con alguno de los niveles de una de las dimensiones del cubo.
- ID de GIS (Propiedad: PrimaryKey): Identificador único para una geometría del lado de GIS en la tabla indicada en "Tabla GIS".
- Para una tabla correspondiente a un layer en GIS, se especificará la relación con OLAP, dentro del tag "OLAPRelation" y este se define con la siguiente información :

- Tabla GIS-OLAP (Propiedad : Table): Para cada relación entre una capa GIS y un nivel de OLAP existe una tabla que contiene los identificadores de ambos lados para establecer la relación. Esta tabla relaciona los IDs de objectos OLAP con IDs de objetos GIS (espaciales).
- Id relacionado de GIS (Propiedad:GisId): Nombre del campo en la "Tabla GIS-OLAP" que contiene el ID del lado de GIS.
- Id relacionado de OLAP (Propiedad:OlapId): Nombre del campo en la "Tabla GIS-OLAP" que contiene el ID del lado de OLAP.
- Finalmente en un tag interno llamado OlapTable se definen los datos del lado de OLAP :
    - Tabla OLAP (Propiedad:Name): Tabla que contiene la dimensión del lado de OLAP relacionada con la "Tabla GIS".
    - ID de OLAP (Propiedad: Id): Nombre del campo de la "Tabla OLAP" que se usa como identificador de un Member del lado.
    - Campos de la Jerarquia OLAP (Propiedad: HierarchyNameAllFields): Es un listado jerárquico que define los niveles superiores al nivel relacionado.

- Elemento ALL (Propiedad:HierarchyAll): Define cómo se llama al elemento ALL en el caso de querer acceder a todos los miembros de un level.

# CAPITULO 4

# Piet-QL : UN LENGUAJE DE CONSULTA PARA PIET

En este capítulo presentamos el lenguaje de consulta Piet-QL, describimos su sintaxis y semántica, y un conjunto de ejemplos que ilustran su uso.

## 4.1 Diseño del lenguaje

El lenguaje fue diseñado basado en la sintaxis y operadores estándares de OpenGIS[6] y MDX. La idea subyacente para la combinación de ambos lenguajes es preservar los estándares de GIS y OLAP tanto como sea posible. En un escenario de OLAP y GIS integrado se deberían cubrir al menos las siguientes consultas:

- Consultas OLAP filtradas por condiciones geométricas.
- Consultas GIS filtradas por condiciones OLAP.
- Consultas GIS puras
- Consultas OLAP puras

Piet-QL no solo cubre los cuatro tipos antedichos, sino que para los dos primeros, soporta subconsultas Piet-QL, es decir:

- Consultas GIS filtradas por una sub-consulta Piet-QL.

---

[6] The Open Geospatial Consortium,Inc, http://www.opengeospatial.org/standards/sfs

- Consultas de OLAP filtradas por una sub-consulta de Piet-QL.

Las subconsultas de Piet-QL pueden ser a su vez de cualquiera de los cuatro tipos descritos anteriormente. Esto ofrece la posibilidad de realizar tantas subconsultas anidadas como se necesiten, por ejemplo, una consulta de tipo OLAP, filtrada por una subconsulta de tipo GIS , que a su vez esta filtrada por otra subconsulta de tipoOLAP y así sucesivamente.

## 4.2 Piet-QL en ejemplos

En esta sección presentamos un conjunto de ejemplos para ilustrar las capacidades de Piet-QL. El caso de estudio en el cual se basan los ejemplos se describe a continuación.

### Base de datos GIS

En el motor GIS contamos con una base de datos que contiene las diferentes capas de un mapa de Bélgica. Las capas definidas son Regiones, Ríos, Provincias, Ciudades y Distritos. Cada una de estas capas esta representada en el esquema público de la base de datos en las tablas bel_regn, bel_river, bel_prov, bel_city y bel_dist respectivamente.

### Base de datos OLAP

La base de datos multidimensional consiste en un cubo de 6 dimensiones y 3 métricas que contiene información sobre las visitas turísticas en Bélgica.

Las dimensiones son :

Personas: Clasifica los distintos tipos de personas. La clasificación se define por género y estado civil. Esta dimensión esta contenida en la tabla "People".

Alojamientos: Clasifica los distintos tipos de alojamiento. La clasificación tiene la jerarquía tipos, subtipos y descripción. Esta dimensión esta contenida en la tabla "Lodging".

Transporte : Clasifica los distintos tipos de transportes en los que se puede llegar a un destino. La clasificación tiene la jerarquía tipos, subtipos y compañía. Esta dimensión esta contenida en la tabla "Transport"

Razón : Permite discriminar por la razón para la visita (Turismo, Negocios, Luna de miel, Estudio, etc.). Esta dimensión esta contenida en la tabla "Reason".

Día de llegada : Es la dimensión temporal. Permite discriminar las consultas por el día de llegada. Esta dimensión esta contenida en la tabla "Arrival_day". Tiene declarada la típica jerarquía de tiempos (Año, Cuartos, Mes, Día y Fecha).

Destino : Es la dimensión geográfica. Permite discriminar las consultas por el lugar de llegada. La clasificación tiene la jerarquía estado, distrito y ciudad. Esta dimensión esta contenida en la tabla "Destination" y se relaciona a través de los metadatos con las capas GIS de Provincias, Distritos y Ciudades respectivamente.

Las metricas son : Cantidad de visitantes, estadía promedio y promedio de gastos.

## Diagrama de entidad relación de la base de datos

En la Figura 4 podemos ver el diagrama Entidad Relación de la estructura.

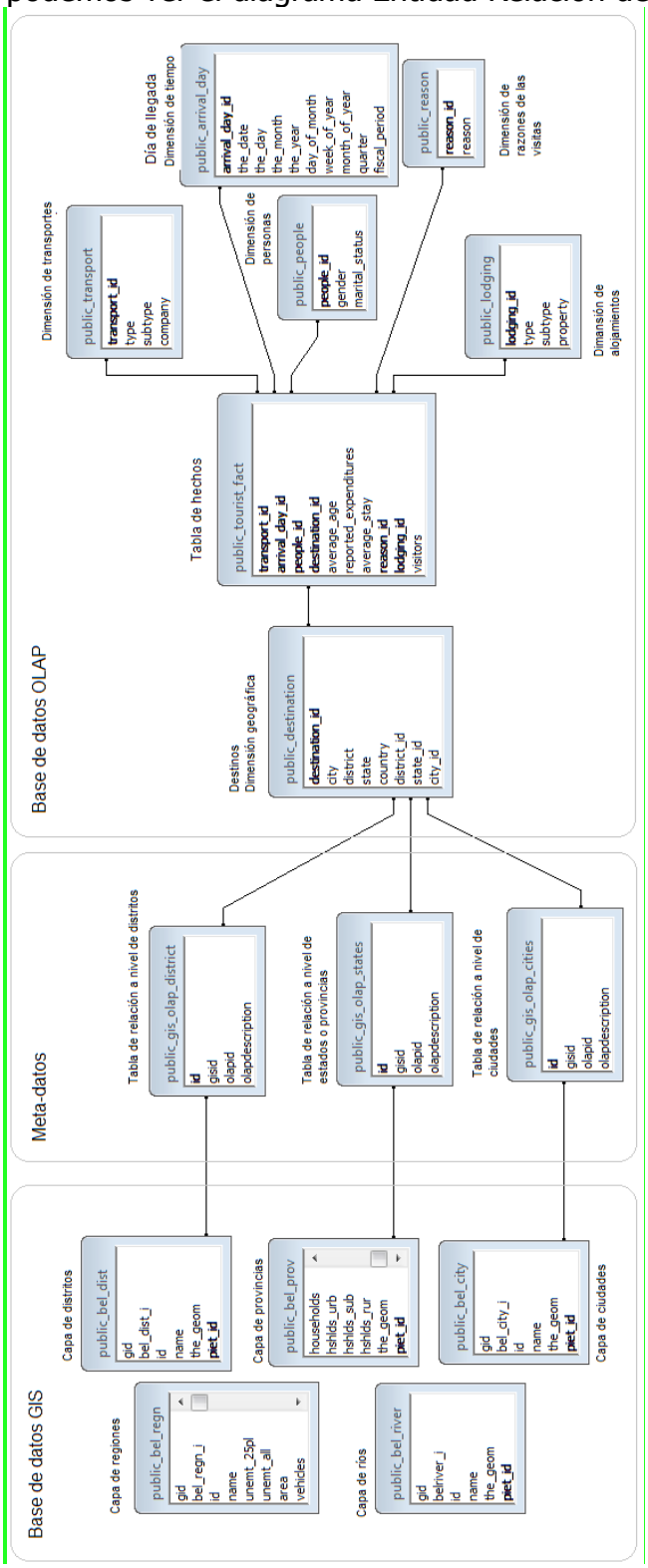


Figura 4: Diagrama de entidad relación de la base de datos

A continuación presentamos Piet-QL a través de una serie de ejemplos.

**Ejemplo 1 [OLAP-GIS]** Cantidad de visitantes, estadía promedio y promedio de gastos para los hospedajes y diferentes tipos de personas ofrecido en distritos intersectados por el río Dyle en 2007".

Esta consulta se expresa en Piet-QL de la siguiente manera :

```
1 SELECT CUBE
2     [Measures].[visitors], [Measures].[stay],
3         [Measures].[expenditures] ON COLUMNS,
4                                 [people].[all], [lodging].[all] ON rows
5     FROM [tourist]
6     WHERE [destination].[all] IN(
7                         SELECT GIS bel_dist
8                         FROM bel_river,bel_dist
9                         WHERE ST_INTERSECTS(bel_dist.the_geom,bel_river.the_geom)
10                        AND bel_river.name='Dyle'               )
11    SLICE [arrival].[2007]
```

La palabra clave CUBE indica que sigue una consulta de tipo OLAP (y por lo tanto se devolverán elementos del cubo). Análogamente, la palabra clave GIS indica que sigue una consulta espacial (y por lo tanto se devolverán elementos espaciales). La parte geométrica corresponde a la subconsulta interna que devuelve las provincias atravesadas por el río Dyle. La parte OLAP devuelve la información limitadas a dichas provincias, mediante el predicado IN.

**Ejemplo 2. [Consulta de tipo GIS]** Listar los distritos atravesados por ríos.

Esta consulta se expresa en Piet-QL de la siguiente manera :

```
1       SELECT GIS DISTINCT(bel_dist.name)
2       FROM bel_dist, bel_river
3       WHERE crosses(bel_river,bel_dist)
```

**Ejemplo 3 [Consulta GIS-OLAP]** Listar los nombres de los distritos interceptados por el río Nete que contengan ciudades con mas de 4000 visitantes durante el año 2007.

Esta consulta se expresa en Piet-QL de la siguiente manera:

```
1 SELECT GIS distinct(bel_dist.name,bel_dist.the_geom)
2         FROM bel_city,bel_dist,bel_river
3         WHERE Contains(bel_dist,bel_city)
4         AND Intersects(bel_dist,bel_river)
5                   AND bel_river.name='Nete'
6                   AND bel_city IN(
7                    SELECT CUBE filter([destination].[city].Members, [Measures].[visitors]>4000)
8                        FROM [tourist]
9                        SLICE [arrival].[2007]
10                 )
```

La consulta interior OLAP obtiene las ciudades cuyos visitantes superan las 4000 personas. La consulta externa tipo GIS utiliza esta listas en su cláusula WHERE.

**Ejemplo 4 [Consulta OLAP-OLAP]** Hospedajes y personas en ciudades de Bélgica, donde los visitantes superaron 4.000 visitantes durante el 2007.

Esta consulta se expresa en Piet-QL de la siguiente manera :

```
1         SELECT CUBE [people].[all], [lodging].[all] ON ROWS
2         FROM [tourist]
3         WHERE [destination].[all] IN (
4                           SELECT CUBE filter([destination].[city].Members, [Measures].[visitors]>4000)
5                           FROM [tourist]
6                            SLICE [arrival].[2007]
7                   )
```

Aquí, todas las partes se resuelven en OLAP. La subconsulta interior, a través de

la expresión MDX "SELECT CUBE filter([destination].[city].Members, [Measures].[visitors]>4000) from [tourist]" obtiene el listado de Ciudades cuyos visitantes superan las 4000 personas. Esta lista de ciudades se utiliza como filtro en la consulta externa.

## 4.3 Sintaxis de Piet-QL

El Listado 1 muestra la sintaxis de Piet-QL en forma Backus-Naur. El lenguaje utiliza como tipos de datos básicos Número (real, entero) y String (Cadenas de caracteres). Los datos geométricos, como puntos, lineas y polígonos, multi-puntos (colecciones de puntos), multi-lineas (colecciones de lineas) y multi-polígono (colecciones de polígonos) se expresan a través de las cadenas de OpenGIS. Por ejemplo, Punto "(10 20)", Línea "(5 10,16 2)" y Polígono "((10 20 30 30 50 20 10 20))" son válidos en geometrías Piet-QL.

Definimos los tipos y conjuntos:

**layers** = conjuntos de layers de un GIS
**geom_pol** = polígonos de GIS pertenece a un layer.
**geom_line** = líneas de GIS pertenece a un layer.
**geom_point** = puntos de GIS pertenece a un layer.
**geom** = geometría general. Es la union de geom_pol, geom_line y geom_point
**gisids** = identificadores de todos los elementos de geom.
**members** = Conjunto de todos los miembros de un cubo, por ejemploFlorida o Arg.BuenosAires.VteLopez.Florida.
**levels** = Conjunto de todos los niveles de un cubo, por ejemploLocalidad, País, Ciudad.

**dimensions** = Conjunto de todas las dimensiones del cubo.

**mesasures** = Conjunto de todas las mediciones del cubo.

**cubes** = Cubos de OLAP

**tuplasgisOLAP** = Conjunto de tuplas definidas en un archivo y un conjunto de tablas de configuración de GISOLAP que permiten relacionar un identificador de una geometría con un miembro de OLAP. Cada tupla está definida por m,i tal que m $\in$ members e i $\in$ gisids

Ademas el Listado 1 utiliza los siguientes Símbolos *Terminales*

GIS_ATTR$: GIS_ATTR$ $\in$ atributes. Referencia al atributo asignado a una geometría de un layer. Por ejemplolayer_city.name referencia a los nombre asignados a cada geometría del layer ciudad.

**OLAP_CUBE$**: OLAP_CUBE$ $\in$ cubes. Referencia al un cubo de OLAP.

**OLAP_MEMBER$**: OLAP_MEMBER$ $\in$ members. Referencia a los miembros del cubo de OLAP.

**OLAP_LEVEL$**: OLAP_LEVEL$ $\in$ levels. Referencia a los niveles del cubo de OLAP.

**GIS_FN$**: Cualquiera de las funciones definidas a continuación:

- Función Buffer

buffer(g, d) : operador sobrecargado cuyos dominios y rangos se definen de la siguiente manera :

geom_pol, real $\rightarrow$ geom_pol

geom_line, real $\rightarrow$ geom_pol

geom_point, real $\rightarrow$ geom_pol

- Función Length

length(g, u) : operador cuyo dominio y rango se definen de la siguiente manera :

geom_line $\rightarrow$ real

- Función Area

area(g, u) : operador cuyo dominio y rango se definen de la siguiente manera :

geom_pol → real

- Función Intersection

intersection( g1, g2): operador sobrecargado cuyos dominios y rangos se definen de la siguiente manera :

geom_pol, geom_pol →geom_pol

geom_pol, geom_line → geom_line

geom_line, geom_line → geom_point

**OLAP_FN$**: La función definida a continuación :

- Funcion Filter

filter( m , f) : operador sobrecargado cuyos dominios y rangos se definen de la siguiente manera :

m ∈ members, <f> → (m | ∅).

Sintaxis de <f>

<f> := <OLAP_mesuare_reference> <Predicados de comparación> <value>

<f> := <OLAP_mesuare_reference> <Predicados de comparación> <value>

- El Predicado de comparación se define de la siguiente manera :

v1 <operador> v2 : operador sobrecargado cuyos dominios y rangos se definen de la siguiente manera :

v1 ∈ integer, v2 ∈ integer → boolean

v1 ∈ integer, v2 ∈ real → boolean

v1 ∈ real, v2 ∈ integer → boolean

v1 ∈ real, v2 ∈ real → boolean

Donde <operador> puede ser = , > , < , > o <

**GIS_LAYER$**: l ∈ layers. Referencia a un layer.

**GIS_PRED$**: Cualquier predicado definidos a continuación:

- Predicado Intersecs

Intersecs( g1, g2) : operador sobrecargado cuyos dominios y rangos se definen de la siguiente manera :

g1 ∈ geom_pol, g2 ∈ geom_line → boolean

g1 ∈ geom_line, g2 ∈ geom_line → boolean

- Predicado Touches

Touches( g1, g2) : operador sobrecargado cuyos dominios y rangos se definen de la siguiente manera :

g1∈ geom, g2 ∈ geom → boolean

- Predicado Crosses

Crosses( g1, g2) : operador sobrecargado cuyos dominios y rangos se definen de la siguiente manera :

g1 ∈ geom_pol, g2 ∈ geom_line → boolean

g1 ∈ geom_line, g2 ∈ geom_line → boolean

- Predicado Within

Within( g1, g2): operador sobrecargado cuyos dominios y rangos se definen de la siguiente manera :

g1∈ geom_pol, g2∈geom_line → boolean

g1∈ geom_line, g2 ∈ geom_line → boolean

- Predicado Overlaps

Overlaps( g1, g2) : operador sobrecargado cuyos dominios y rangos se definen de la siguiente manera :

g1 ∈ geom_pol, g2 ∈ geom_line → boolean

g1 ∈ geom_line, g2 ∈ geom_line → boolean

- Predicado Contains

Contains( g1, g2) : operador sobrecargado cuyos dominios y rangos se definen de la siguiente manera :

g1 ∈ geom_pol, g2 ∈ geom_pol) → boolean

g1 ∈ geom_pol, g2 ∈ geom_line) → boolean

g1 ∈ geom_pol, g2 ∈ geom_point) → boolean

g1 ∈ geom_line, g2 ∈ geom_point) → boolean

- Predicado Covers

Covers( g1, g2) : operador sobrecargado cuyos dominios y rangos se definen de la siguiente manera :

g1 ∈ geom_pol, g2 ∈ geom_pol) → boolean

g1 ∈ geom_pol, g2 ∈ geom_line) → boolean

g1 ∈ geom_pol, g2 ∈ geom_point) → boolean

g1 ∈ geom_line, g2 ∈ geom_point) → boolean

Por comodidad si en algún predicado GIS se indica un layer el sistema considerará que se hace referencia al atributo geométrico de cada elemento del layer.

LITERAL$: Tipos de datos básicos como números o cadenas de caracteres.

*Listado 1 : Sintaxis de la consulta Piet QL*

```
<query> ::=
        SELECT GIS <gis-heading> | SELECT GIS Distinct( <gis-heading> ) | SELECT CUBE <olap-heading>
<gis-heading> ::=
        <gis-list> <gis-from-clause> [<gis-where-clause> ] [<group-by-clause> ]
<gis-list> ::=
        GIS_ATTR$ [,<gis-list>] | GIS FN$ [,<gis-list>]
<gis-from-clause> ::=
        FROM <layer-list>
<layer-list> ::=
        GIS_LAYER$ [,<layer-list>]
<gis-where-clause> ::=
        WHERE <gis-filter>
<gis-filter> ::=
        <gis-predicate> AND <gis-filter> | <gis-predicate> OR <gis-filter> | [NOT] (gis-filter>)
<gis-predicate> ::=
        GIS_ATTR$ IN ( <single-result-subquery> ) | GIS_PRED$ |
        GIS_FN$ = LITERAL$ | GIS_ATTR$ = LITERAL$
<group-by-clause> ::=
        GROUP BY <gis-list>
<olap-heading> ::=
        <olap-list> ON ROWS [<olap-list> ON COLUMNS]
                <olap-from-clause> [<olap-where-clause> ] [ <olap-slice-clause> ]
        |
        <olap-list > ON COLUMNS [ <olap-list> ON ROWS ]
                <olap-from-clause> [<olap-where-clause> ] [ <olap-slice-clause> ]
<olap-list> ::=
        OLAP_MEMBER$ [,<olap-list>] | OLAP_LEVEL$ [ ,<olap-list >] | OLAP_FN$ [, <olap-list>]
<olap-from-clause> ::=
        FROM  OLAP_CUBE$
<olap-where-clause> ::=
        WHERE <olap-filter>
<olap-filter> ::=
        <olap-predicate> [ OR <olap-filter>] | [NOT] (<olap-filter>)
<olap-slice-clause> ::=
        SLICE <olap-list>
<olap-predicate> ::=
        OLAP_MEMBER$ IN (<single-result-subquery>) |
        OLAP_LEVEL$ IN ( <single-result-subquery> ) |
        OLAP_FN$ IN (<s i ng le-result-subquery> )
<single-result-subquery> ::=
        SELECT GIS GIS_LAYER$ <gis-from-clause> [<gis-where-clause> ] |
        SELECT CUBE OLAP_MEMBER$ <olap-from-clause> [<olap-where-clause>] [<olap-slice-clause> ] |
        SELECT CUBE OLAP_LEVEL$ <olap-from-clause> [<olap-where-clause>] [<olap-slice-clause> ] |
        SELECT CUBE OLAP_FN$  <olap-from-clause> [<olap-where-clause>] [<olap-slice-clause> ]
```

## 4.4 Semántica de Piet-QL

Para definir la semántica de la consulta Piet-QL comenzaremos definiendo la semántica de los predicados y funciones que utiliza el lenguaje.

### Predicados generales

**Operadores in-fijos de comparación**: v1 <operador> v2 : Equivalente a la operación = , > , < , > o <, respectivamente entre los números reales y o enteros en lógica de primer orden. Donde <operador> puede ser = , > , < , > o <.

**Operador in-fijo IN:** gml IN (<single−result−subquery>): IN es verdadero si y solo sí el identificador de gml pertenece al conjunto de identificadores devueltos por <single−result−subquery>. <single−result−subquery> es una sub-consulta cuya sintaxis y semántica se define en "Sintaxis de la consulta" y en "Semántica de la consulta" respectivamente.

El predicado IN materializa los filtros entre consultas GIS-OLAP o OLAP-GIS, este predicado infijo es de la forma: id IN (setofIDs), donde id puede ser un identificador de un objeto espacial o un miembro de OLAP. Esto significa que este predicado devuelve verdadero si 'id' pertenece al conjunto de identificadores . Estos identificadores, dependiendo del tipo de consulta, pueden se de tipos espaciales o no-espaciales. En otras palabras, a IN S devuelve verdadero si (a, x) es una tuplasgisOLAP, $x \in S$.

## Predicados GIS

Por comodidad, si en algún predicado GIS se indica un layer el sistema considerará que se hace referencia al atributo geométrico de cada elemento del layer.

**Intersecs( g1, g2) :** Devuelve TRUE si g1 y g2 se interceptan. Mapea a al operador ST_Intersects(geometry, geometry) de OpenGIS.

**Touches( g1, g2) :** Devuelve TRUE si el g1 se toca con g2. Mapea a al operador ST_Touches(geometry, geometry) de OpenGIS.

I(b) = {empty set} ) and (a intersection b) not empty

**Crosses( g1, g2) :** Devuelve TRUE si el g1 y g2 se cruzan. Mapea a al operador ST_Crosses(geometry, geometry) de OpenGIS.

**Within( g1, g2):** Devuelve TRUE si el g1 está dentro de g2. Mapea a al operador ST_Within(geometry, geometry) de OpenGIS.

**Overlaps( g1, g2) :**Devuelve TRUE si el g1 y g2 se se sOLAPan. Mapea a al operador ST_Overlaps(geometry, geometry) de OpenGIS.

**Contains( g1, g2) :** Devuelve verdadero si gl contiene a g2. Mapea a al operador ST_Contains(geometry, geometry) de OpenGIS.

**Covers( g1, g2) :** Devueleve verdadero si g1 contiene a todos los puntos de g2. Mapea a al operador ST_Covers(geometry, geometry) de OpenGIS.

## Funciones OLAP

**Filter( m , f)** : Devuelve *m* si *m* cumple la condición indicada en el filtro *f*, para la medición indicada caso contrario devuelve vacío.

## Funciones de GIS

Si en alguna función GIS se indica un layer el sistema considerará que se hace referencia al atributo geométrico del cada elemento del layer.

**Buffer(g, d) :** Para una geometría g y una distancia d, devuelve una geometría en la cual todos los puntos de la misma se encuentran a una distancia igual o menor que d de la geometría g. Mapea a al operador ST_Buffer(geometry, double, [integer]) de OpenGIS de la siguiente manera ST_Buffer(g, d).

**Length(g, u) :** Para una geometría g obtiene la longitud L aplicando ST_Length(geometry) de OpenGIS de la siguiente manera ST_Length(g).

**Area(g, u) :** Para una geometría g obtiene el area A aplicando ST_Area(geometry) de OpenGIS de la siguiente manera ST_Area(g).

**Intersection( g1, g2):** Devueleve la geometría de la intersección del elemento geométrico g1 con el elemento geométrico g2 aplicando la función ST_Intersection(geometry, geometry) de OpenGIS.

## Semántica de la consulta

Finalmente resumimos la semántica Piet-QL en la siguiente tabla. Dado que hemos definido cuatro tipos básicos de consultas, en las tres primeras columnas, de la izquierda, se indica a que tipo puede aplicar el componente de la cláusula

descrita en la cuarta columna. Por ejemplo, la cuarta línea en la tabla indica que la instrucción CUBE seleccionada se aplica a una sub-consulta en una consulta de tipo GISOLAP, y la quinta columna explica la semántica de la sub-consulta OLAP

| Aplica a consulta : | | | SEGMENTO DE LA SINTAXIS ANALIZADA | Semántica |
|---|---|---|---|---|
| GIS | OLAP | SUB | | |
| | | | | |
| NO | SI | NO | SELECT CUBE <olap-list><br>ON ROWS [<olap-list> ON COLUMNS<br><olap-from-clause> | Selecciona los miembros y/o niveles que se mostrarán del cubo definido en el <olap-from-clause><br>Ademas indica los ejes en la que se devolverán los resultados (Columnas o Filas)<br>Cada átomo de <olap-list> debe estar definido en el cubo indicado en <olap-from-clause> |
| SI | NO | NO | SELECT GIS <gis-list><br><gis-from-clause> | Selecciona los atributos que se mostrarán de los layers de GIS definidos en el <gis-from-clause><br>Cada átomo de <gis-list> debe estar definido en alguno de los layers indicados en <gis-from-clause> |
| NO | NO | SI | SELECT GIS GIS LAYER$<br><gis-from-clause> | Indica el layer del cual se obtendrá el identificador para usar en el predicado IN que hizo la llamada al subquery. |
| NO | NO | SI | SELECT CUBE<br>OLAP MEMBERS$<br><olap-from-clause> | Selecciona los miembros del cual se obtendrán los identificadores para usar en el predicado IN que hizo la llamada al subquery. |
| | | | | |
| NO | SI | SI | ...FROM OLAP CUBE$ | Define el cubo sobre el cual se realizará la consulta. Este lenguaje de consulta se define para un único cubo. |
| SI | NO | SI | ...FROM <layer-list> | Define el/los layers de GIS que participarán de la consulta.<br>Al ejecutarse la consulta se realizará el producto cartesiano de todas las geometrías de los layer indicados en <layer-list> |
| | | | | |
| NO | SI | SI | ... WHERE <olap-filter> | Filtra el eje de las filas del Select solo a aquellas tuplas que cumplan positivamente (resultado = true) la evaluación lógica del predicado <olap-filter> |
| SI | NO | SI | ... WHERE <gis-filter> | Filtra los resultados del producto cartesiano de <layer-list> solo a aquellas tuplas que cumplan positivamente (resultado = true) la evaluación lógica del predicado <gis-filter> |
| | | | | |
| NO | SI | SI | ...SLICE <olap-list> | Si se incluye este término, en vez de considerarse el cubo entero el motor toma el cubo limitado al slice determinado por <olap-list>.<br>Condición : (Todos los miembros de <olap-list> deben pertenecer a distintas dimensiones.)<br>Si $\exists$ d $\in$ dimensions y n y m $\in$ members /<br>n $\exists$ en <slicefilterlist> $\wedge$<br>m $\exists$ en <slicefilterlist> $\wedge$<br>n $\in$ d $\Rightarrow$ m $\notin$ d |

Tabla 1 – Semántica de la consulta Piet-QL

# CAPITULO 5

## DETALLES DE LA IMPLEMENTACIÓN

En nuestra aplicación, los datos son almacenados en el sistema administrador de bases de datos relacionales PostgreSQL[7]. Para resolver la parte relacionada con consultas geométricas del modelo de datos, utilizamos PostGIS[8], un plugin que añade soporte para objetos geográficos a PostgreSQL. PostGIS sigue la especificación OpenGIS "Simple Features Specification for SQL" y está certificado como compatible con el perfil en cuanto a "Tipos y funciones". Para el desarrollo se utilizó Java[9], en particular la biblioteca JTS Topology Suite[10], un plugin para la manipulación de geometrías lineales 2-dimensionales que también se ajusta a la especificación "Simple Features para SQL" publicado por Open GIS. Para resolver la parte de OLAP en el modelo de datos, usamos Mondrian[11], un servidor ROLAP escrito en Java. Por último, nuestros clientes se basan en un WebServer Apache y en OpenJUMP[12], un software GIS basado en Java.

Cuando una consulta de Piet-QL es enviada al sistema, un analizador descompone en las partes GIS y OLAP, y cada uno de ellas es ejecutado por el

---

7 http://www.postgresql.org.
8 http://www.postgis.org.
9 http://java.sun.com.
10 http://www.vividsolutions.com/jts.
11 http://mondrian.pentaho.org.
12 http://www.openjump.org.

motor correspondiente, la parte GIS en Postgres / PostGIS y la parte OLAP en Mondrian. Sintácticamente la consulta se divide en partes de GIS y OLAP mediante el predicado IN. La consulta interna que se ejecuta en primer lugar, devuelve un resultado intermedio que se utiliza para re-escribir la consulta externa. Este proceso se repite hasta llegar a la consulta más externa de todas. Esta última traducción es la final de este proceso, y se devuelve como respuesta.

Con el fin de aclarar el proceso, estudiaremos dos consultas. La primera es una consulta GIS filtrada por condiciones OLAP y la segunda es una consulta OLAP filtrada con una consulta GIS.

## Ejemplo de Consulta GIS-OLAP.

Empezamos con la consulta de GIS-OLAP: "Distritos (nombres y geometrias) de Bélgica intersecados por el río Nete, que contengan ciudades con más de 4000 visitantes durante el 2007".

Esta consulta se expresa en Piet-QL, como se muestra a continuación :

```
1       SELECT GIS distinct(bel_dist.name,bel_dist.the_geom)
2       FROM bel_city,bel_dist,bel_river
3       WHERE Contains(bel_dist,bel_city)
4       AND Intersects(bel_dist,bel_river)
5               AND bel_river.name='Nete'
6               AND bel_city IN(
7   SELECT CUBE filter([destination].[city].Members, [Measures].[visitors]>4000)
8   FROM [tourist]
9   SLICE [arrival].[2007])
```

Dado que la consulta externa es de tipo GIS , su resultado puede ser visualizado en un mapa. De hecho, hay 2 distritos en Bélgica (Antwerpen y Mechelen), que son atravesados por el río Nete. Pero sólo Antwerpen contiene ciudades con más

de 4000 visitantes durante el 2007.

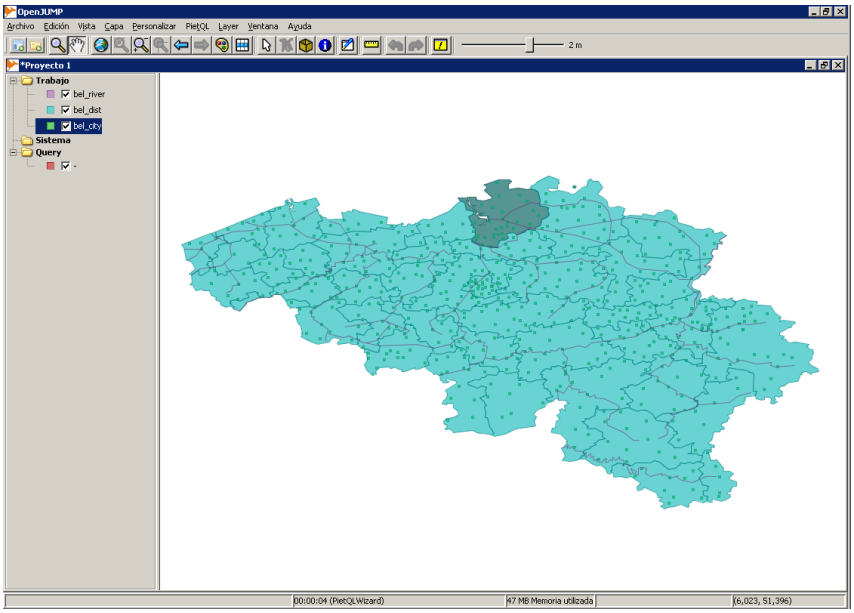


Figura 5. Visualización en OpenJump del resultado de la consulta GIS-OLAP del ejemplo anterior.

El analizador identifica una subconsulta utilizando el predicado IN: la subconsulta de las líneas 7 a 9 en el ejemplo, es la más interna y por lo tanto se resuelve en primer lugar. Dado que la expresión comienza con "SELECT CUBE", es una subconsulta OLAP, por lo tanto, es traducida a MDX, quedando de la siguiente manera :

```
1       SELECT {FILTER([DESTINATION.DESTINATION].[CITY].MEMBERS,
2                       ([MEASURES].[VISITORS] > 4000.0))} ON ROWS
3       FROM [TOURIST]
4       WHERE [ARRIVAL.ARRIVAL].[ANYTIME].[2007]
```

Una vez traducida se ejecuta en el servidor Mondrian. El conjunto de resultados obtenidos se muestra en la Tabla 2 y corresponde a las ciudades con más de 4000 visitantes durante el 2007.

| destination | | | |
|---|---|---|---|
| **Antwerpen** | **Brussel** | **Jalhay** | **Brugge** |
| 39,217 | 39,072 | 39,169 | 39,359 |

Slicer: [year=2007]

Tabla 2: Ciudades cuyos visitantes superan las 4000 personas.

El predicado IN necesita los identificadores GIS de estas ciudades. Por lo que es necesario consultar los metadatos que relacionan GIS-OLAP. La ciudad tiene una capa de GIS vinculante con datos OLAP, que se almacena en los metadatos de GIS-OLAP. En la Listado 2 se muestran solamente la porción de interés de estos metadatos.

*Listado 2: Porción de metadatos de relación GIS-OLAP*

```
1   <!--City layer-->
2   <Layer name="bel_city" hasAll="true" table="bel_city" primaryKey="PIET_ID"
3   geometry="the_geom" descriptionField="name">
4       <!--Layer properties-->
5       <Properties>
6           <Property name="Length" column="PIET_LENGTH" type="Double" />
7           <Property name="Area" column="PIET_AREA" type="Double" />
8       </Properties>
9       <!--List of subpolygonization levels included in the corresponding layer hierarchy-->
10      <SubpolygonizationLevels>
11          <!--Reference Subpolygonization definitions -->
12          <SubPUsedLevel name="Point" />
13      </SubpolygonizationLevels>
14      <OLAPRelation table="gis_olap_cities" gisId="gisid" olapId="olapid"
15      olapDimensionName="destination" olapLevelName="city">
16          <OlapTable name="destination" id="city_id" hierarchyNameField="city"
17  hierarchyNameAllFields="country,state,district,city" hierarchyAll="[destination].[all]" />
18      </OLAPRelation>
19  </Layer>
```

En la línea 14 del Listado 2 podemos ver que el nombre de la tabla es gis_olap_cities y los atributos que se relacionan son gisid y olapId para GIS y OLAP, respectivamente.

El traductor busca en la tabla de la dimensión "destination" de OLAP los identificadores de las ciudades resultantes de la consulta OLAP, con estos identificadores y la tabla gis_olap_cities, obtiene los identificadores de las ciudades del lado GIS. En nuestro ejemplo, estos valores son 182,111,351,180,352. Ahora podemos volver a escribir la subconsulta GIS con estos identificadores, y obtener una expresión SQL que se puede ejecutar en la base de datos PostgreSQL tal como se muestra a continuación :

```
1       SELECT DISTINCT bel_dist.name,bel_dist.the_geom
2       FROM bel_city,bel_dist,bel_river
3       WHERE ST_CONTAINS(BEL_DIST.THE_GEOM,BEL_CITY.THE_GEOM)
4               AND ST_INTERSECTS(BEL_DIST.THE_GEOM,BEL_RIVER.THE_GEOM)
5               AND bel_river.name='Nete'
6               AND (bel_city.PIET_ID IN (182,111,351,180,352))
```

El resultado de la ejecución de este SQL en PostgreSQL es "Antwerpen".

## Ejemplo de Consulta OLAP-GIS.

El análisis de una consulta OLAP-GIS es más complejo que el caso anterior, porque la expresión MDX debe ser reescrita porque no hay un predicado IN en el lenguaje MDX.

Supongamos la consulta "Cantidad de visitantes, estadía promedio y promedio de gastos para los hospedajes y diferentes tipos de personas en distritos interceptados por el río Dyle en 2007". Esta consulta se expresa en Piet-QL, como se muestra a continuación:

```
1   SELECT CUBE
2   [Measures].[visitors], [Measures].[stay],
3       [Measures].[expenditures] ON COLUMNS,
4                       [people].[all], [lodging].[all] ON rows
5   FROM [tourist]
6   WHERE [destination].[all] IN(
```

```
7                   SELECT GIS bel_dist
8                   FROM bel_river,bel_dist
9                   WHERE ST_INTERSECTS(bel_dist.the_geom,bel_river.the_geom)
10                  AND bel_river.name='Dyle'               )
11      SLICE [arrival].[2007]
```

La subconsulta externa indica que el resultado será de tipo OLAP. Podemos ver el resultado obtenido del mismo en la Tabla 3.

| | | | Medidas | | |
|---|---|---|---|---|---|
| **destination** | **people** | **lodging** | **visitors** | **stay** | **expenditures** |
| **Leuven** | **all** | **all** | 68.371 | 37,793 | 249.424.150,21 |
| **Nijvel** | **all** | **all** | 57.436 | 38,135 | 213.046.100,72 |

Slicer: [year=2007]

Tabla 3: Visualización en Mondrian del resultado de la consulta OLAP-GIS del ejemplo.

El analizador identifica la subconsulta con el predicado IN, entre las líneas de 7 a 10 en el ejemplo anterior, esta es la más interna y por lo tanto se resuelve en primer lugar. Dado que la expresión comienza con SELECT GIS , el analizador reconoce que es del tipo GIS y la traduce en SQL de la siguiente manera :

```
1                   SELECT bel_dist.PIET_ID
2                   FROM bel_river,bel_dist
3                   WHERE ST_INTERSECTS(bel_dist.the_geom,bel_river.the_geom)
4                   AND bel_river.name='Dyle'               )
```

Una vez traducida, la consulta se puede ejecutar en la base de datos PostgreSQL. El conjunto de resultados contiene los 2 distritos de Bélgica atravesados por el río Dyle (Leuven y Nijvel). De forma análoga al ejemplo anterior, se consulta los metadatos GIS-OLAP para averiguar el nombre de la tabla que contiene el enlace entre los identificadores de OLAP y los

identificadores GIS. En este caso la tabla es gis_olap_districts. Por lo tanto uniendo gis_olap_districts con la tabla donde se almacena la capa de distrito (bel_dist), podemos encontrar los identificadores de los distritos en el lado OLAP. Con esto, tenemos que construir la jerarquía en la dimensión "Destination" de OLAP usando los metadatos OLAP. La Figura 6 muestra la tabla de dimensiones que contiene esta información y que se utiliza para construir el siguiente fragmento de expresión MDX :

```
[destination.destination].[all].[Belgium].[Brabant].[Nijvel]
[destination.destination].[all].[Belgium].[Brabant].[Leuven]
```

| | store_country character varyi | store_state character varying(30 | store_district character varying(30) | store_city character varying(30) |
|---|---|---|---|---|
| 1 | Belgium | Antwerpen | Antwerpen | Antwerpen |
| 2 | Belgium | Antwerpen | Antwerpen | Essen |
| 3 | Belgium | Antwerpen | Antwerpen | Hove |
| 4 | Belgium | Antwerpen | Mechelen | Lier |
| 5 | Belgium | Antwerpen | Turnhout | Ravels |
| 6 | Belgium | Brabant | Brussel-Hoofdstad | Brussel |
| 7 | Belgium | Brabant | Brussel-Hoofdstad | Evere |
| 8 | Belgium | Brabant | Nijvel | Lasne |
| 9 | Belgium | Brabant | Nijvel | Rebecq |
| 10 | Belgium | Namur | Namen | Floreffe |
| 11 | Belgium | Namur | Namen | La Bruyere |

Figura 6: Extracto de los miembros y los niveles utilizados para construir la jerarquía

Por último, la consulta OLAP se reescribe con un CrossJoin de estas jerarquías tal como se muestra a continuación y se ejecuta en Mondrian, el servidor OLAP.

```
1   SELECT {[Measures].[visitors], [Measures].[stay],
2           [Measures].[expenditures]} ON COLUMNS,
3     Crossjoin(Hierarchize({
4           [destination.destination].[all].[Belgium].[Brabant].[Nijvel],
5           [destination.destination].[all].[Belgium].[Brabant].[Leuven]}),
6               {([people.people].[all], [lodging.lodging].[all])}) ON ROWS
7   FROM [tourist]
8   WHERE [arrival.arrival].[anytime].[2007]
```

# Capítulo 6

## IMPLEMENTACIÓN DE CLIENTES PIET-QL

La implementación de Piet-QL consiste en una librería JAVA que puede invocarse desde cualquier front-end. Para poder utilizar esta librería se desarrollaron dos clientes de Piet-QL: (a) una Interfaz WEB, simple, que permite ejecutar consultas sobre una base de datos de estudio proveyendo, además, un conjunto de consultas predefinidas a modo de ejemplo; (b) un plugin con estilo de asistente (Wizard) gráfico para OpenJUMP[13]. OpenJUMP es un software GIS basado en Java que brinda la posibilidad de graficar y trabajar con geometrías de GIS. La Figura 7 muestra en forma simplificada la abstracción entre las librerías y la visión del usuario final de Piet-QL.

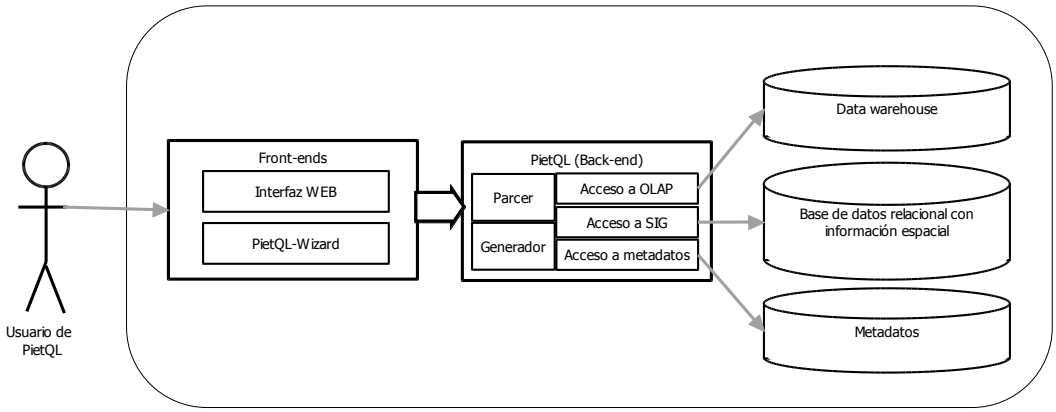


Figura 7: Abstracción entre las partes y la visión del usuario final de Piet-QL.

## 6.1 Interfaz WEB

Es un programa JAVA que actúa a través de un web-server y hace de interfaz de usuario para la utilización de la librería de Piet-QL. El usuario puede editar una

[13] http://www.openjump.org.

consulta Piet-QL, seleccionar una entre las predefinidas, o editar una consulta predefinida, modificándola. También puede elegir entre todas las consultas de ejemplo propuestas en este trabajo (Figura 8). En la parte inferior de la pantalla se encuentra una ventana de texto en donde se observa, en modo texto, la consulta seleccionada en el lenguaje Piet-QL.

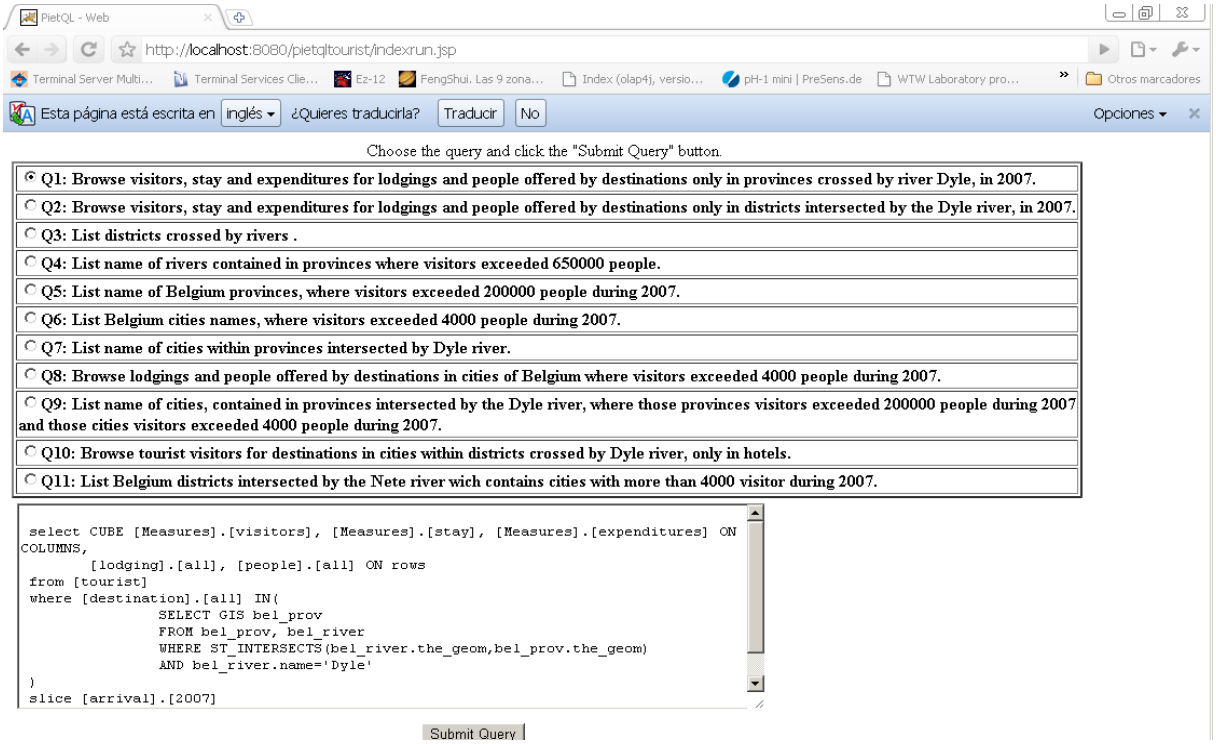


Figura 8 – Pantalla del Cliente WEB

### 6.1.1 Arquitectura de la interfaz Web

Es un programa JAVA que brinda servicio a través de un servlet, que puede correr en un servidor web. Para esta implementación, usamos Apache Tomcat[14]. La interfaz WEB utiliza la librería Piet-QL para procesar las consultas y solo accede al motor de Mondrian, para iniciar sesión, cuando una consulta genera un resultado OLAP. Esta sesión luego es entregada al servidor web para la navegación de los resultados OLAP. La Figura 9 muestra la arquitectura.

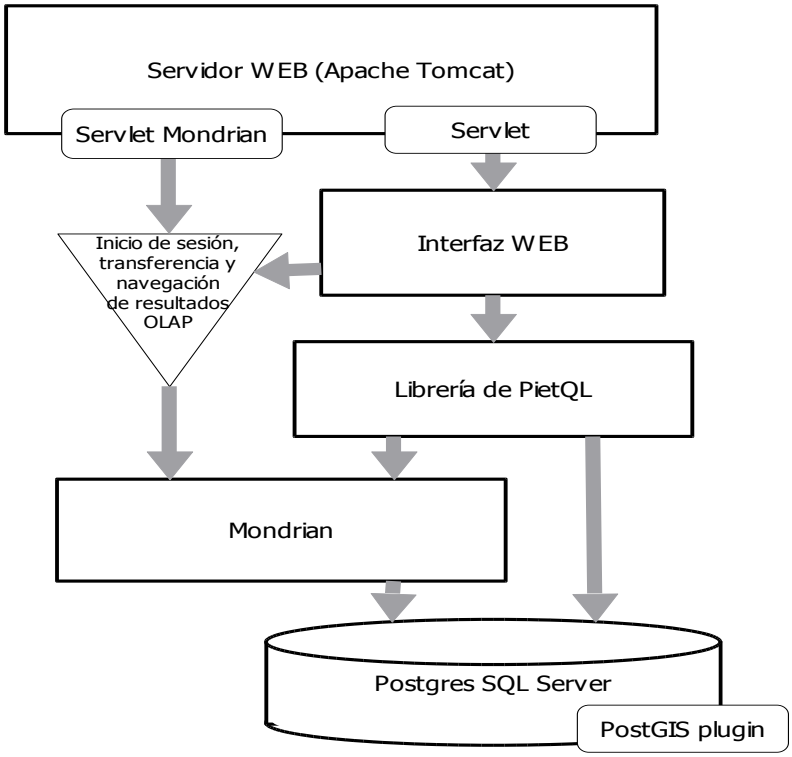


Figura 89 - Arquitectura de la interfaz Web

Apache utiliza el servlet del cliente Web para procesar las consultas y a su vez utiliza Mondrian para las derivaciones al navegador de cubos cuando una consulta es de tipo OLAP. El servlet del cliente Web a través de una clase Java

[14] http://tomcat.apache.org/

accede a la librería de Piet-QL para obtener las traducciones y las tuplas de resultados GIS. Finalmente la librería de Piet-QL utiliza Mondrian y Postgres para la traducción y ejecución de la consulta. Postgres utiliza a PostGIS como plugin para procesar las condiciones geométricas.

### 6.1.2 Descripción de la interfaz

En la pantalla principal de la interfaz web se observan las consultas de ejemplo de este trabajo. El usuario puede seleccionar cualquiera de los ejemplos. Al seleccionar uno de ellos, la consulta en lenguaje Piet-QL aparece en el cuadro de texto que se encuentra en la parte inferior de la pantalla. El usuario puede trabajar con la consulta seleccionada, cambiarla o escribir una completamente nueva en este cuadro, al presionar "Submit Query" el servidor toma la consulta escrita en la ventana de texto y la procesa a través de la librería de Piet-QL.

Si la consulta escrita es tal que su resultado es de tipo GIS, la Interfaz WEB, utiliza la librería de Piet-QL para traducir la consulta y luego utiliza esa misma librería para obtener los resultados. Los resultados se formatean en un cuadro de HTML dentro de la misma Interfaz WEB y se devuelven como respuesta al usuario. La interfaz WEB utiliza limitadores para la cantidad de registros a mostrar, de esta manera los resultados que contienen muchos registros se mostrarán en varias páginas.

Por el otro lado si la consulta escrita deriva en un resultado de tipo OLAP, la Interfaz WEB utiliza la librería de Piet-QL para traducir la consulta, y con esta traducción inicia una sesión Mondrian, para transferirle el control al front-end de

Mondrian, este provee de todas las herramientas de OLAP para la navegación de los resultados de la traducción obtenida.

## 6.2 Piet-QL Wizard

Piet-QL Wizard es un programa JAVA que se instala como plugin de OpenJUMP para la utilización de la librería de Piet-QL. OpenJUMP es un software GIS basado en Java que brinda la posibilidad de graficar y trabajar con geometrías de GIS. Piet-QL Wizard permite guiar paso a paso al usuario para crear consultas sencillas sin necesidad de conocer la sintaxis de Piet-QL. Los resultados de las consultas generadas se grafican en OpenJUMP o se envían al front-end de Mondrian dependiendo si las mismas devuelven resultados del tipo GIS u OLAP respectivamente.

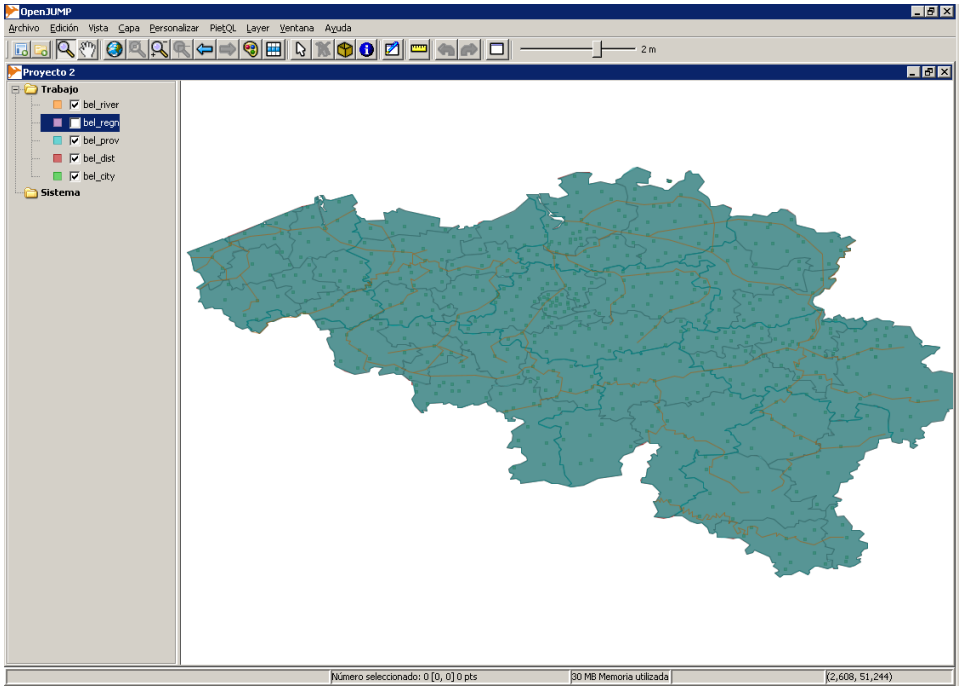

Figura 10 – Pantalla de Open Jump

### 6.2.1 Arquitectura de Piet-QL Wizard

Es un programa JAVA que brinda servicio como plugin de OpenJUMP. Utiliza la librería Piet-QL para procesar las consultas, y muestra los resultados mediante OpenJUMP si los resultados de la consulta son GIS o mediante el front-end de Mondrian, para la navegación de los resultados del cubo, cuando los resultados son de tipo OLAP.

También utiliza las librerías de PostGIS para la obtención de geometrías. Para la implementación de este front-end, además de las funciones estándares y declaradas de la librería de Piet-QL, se programaron algunas funciones adicionales para dar soporte a Piet-QL Wizard, ya que este, intenta graficar los mapas de las consultas realizadas independientemente de que el usuario haya seleccionado geometrías en el wizard o no. La figura 8 muestra la arquitectura de este programa.

OpenJUMP utiliza a Postgres para la obtención de geometrías y a PostGIS para el procesamiento de las mismas. A través de una registración como plugin, Piet-QL Wizard ofrece esta funcionalidad adicional. Piet-QL Wizard utiliza Apache para la derivación de resultados OLAP, quien a su vez utiliza Mondrian para la funcionalidad de navegación de cubos. Además, Piet-QL Wizard necesita de Mondrian y Postgres para presentar en pantalla al usuario, los campos, dimensiones, miembros y mediciones de GIS y OLAP disponibles para el armado de consultas. También utiliza la librería de Piet-QL para obtener las traducciones

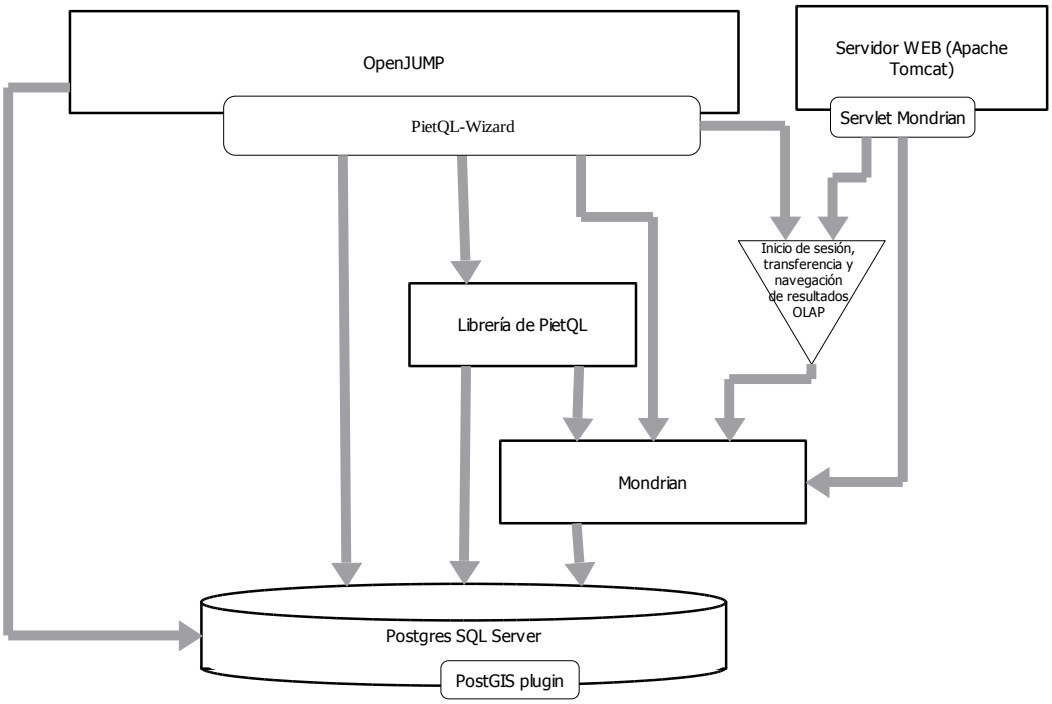

Figura 11 - Arquitectura de Piet-QL Wizard

y, adicionalmente, la lista de tablas GIS que puede venir en una consulta cuyos resultados finales son GIS.

Para la devolución de resultados, necesita de PostGIS para formatear y devolver las geometrías que se graficarán en OpenJUMP.

Finalmente la librería de Piet-QL utiliza Mondrian y Postgres para la traducción y ejecución de la consulta. Postgres utiliza a PostGIS como plugin para procesar las condiciones geométricas.

### 6.2.2 Detalles de la implementación

En Piet-QL Wizard se implementó un asistente que permite avanzar y retroceder entre diferentes pantallas las cuales guían paso a paso al operador para armar la consulta. Las pantallas son formularios gráficos hechos en Java. Al finalizar el asistente se regresa al formulario principal de Piet-QL Wizard donde se presenta

la consulta generada por el mismo. En la Figura 12 se muestra el diagrama de flujo de este asistente.

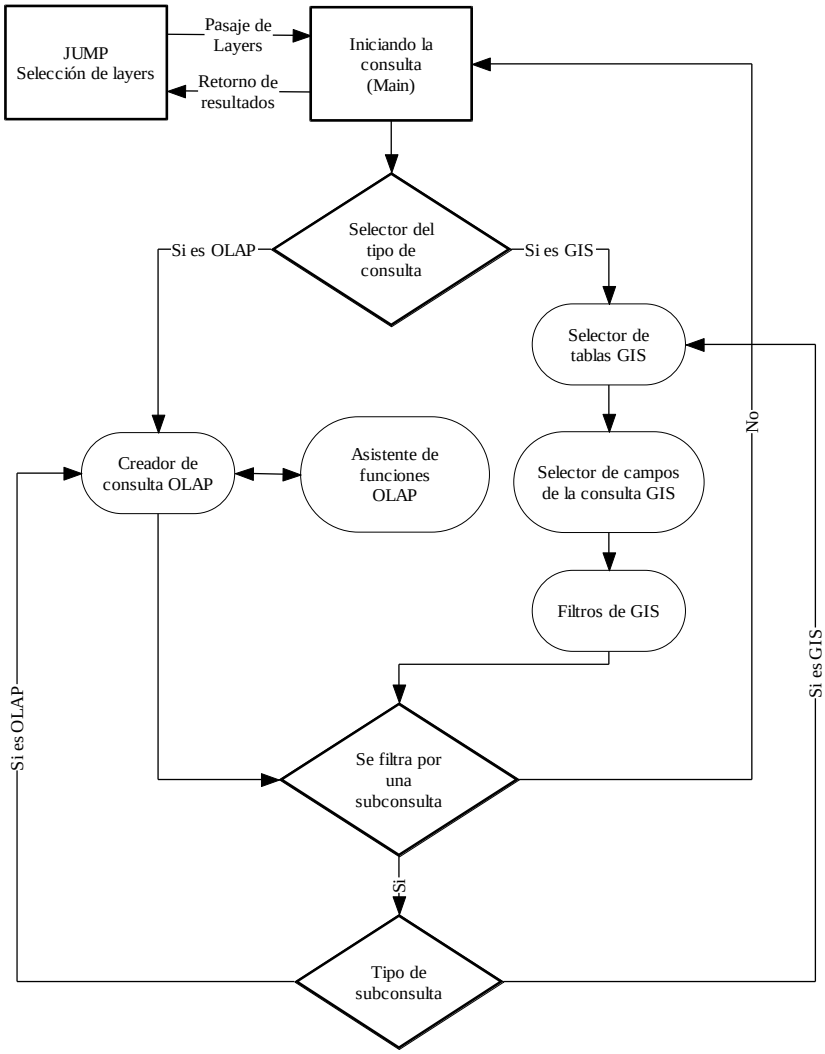


Figura 12 – Diagrama de flujo del asistente.

A continuación se muestran a modo de ejemplo, algunas de las pantallas de Piet-QL Wizard.

Por ejemplo, si la consulta a generar es de tipo GIS, el asistente presentar'a al usuario una pantalla como la siguiente, para seleccionar los layers con información espacial.

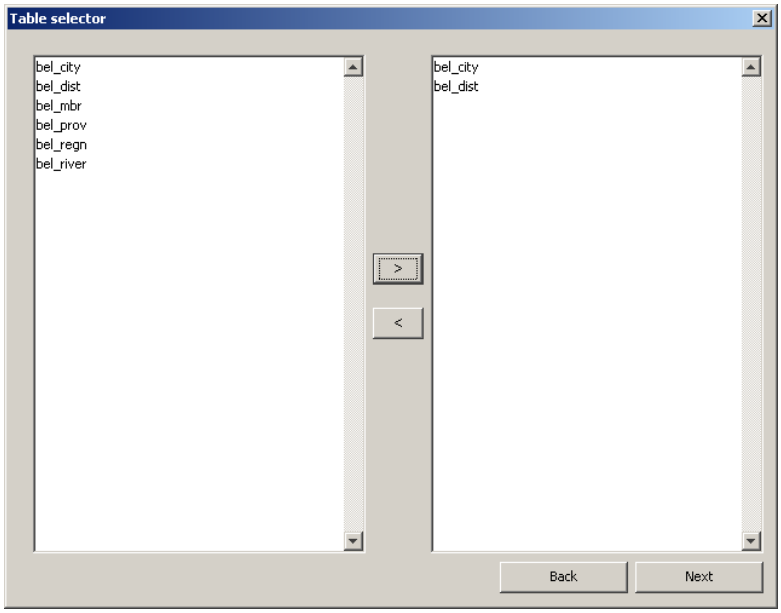

Figura 13 – Selector de tablas GIS.

Una vez elegidas las tablas, se pueden elegir los atributos pertenecientes a las tablas seleccionadas, mediante la pantalla de la Figura 14.

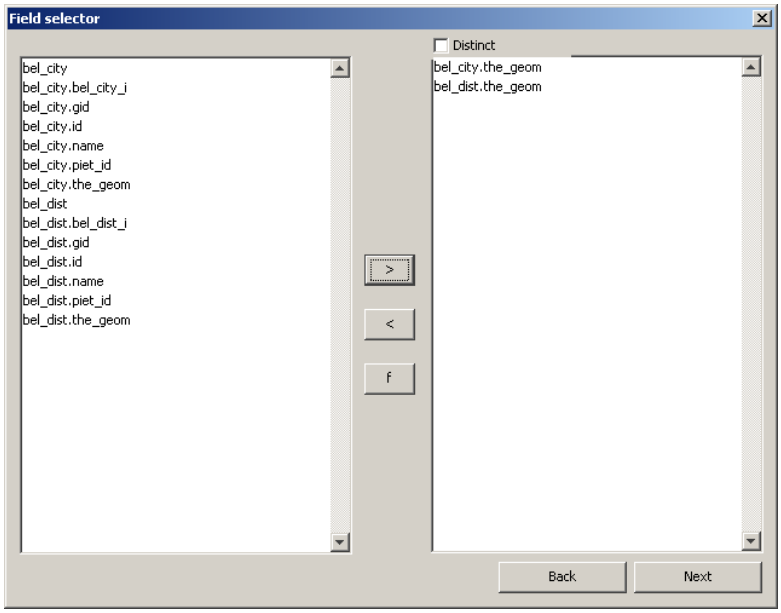

Figura 14 – Selector de campos de la consulta GIS.

La pantalla de filtros de GIS (Figura 15) se utiliza para indicar los filtros que servirán para seleccionar la información deseada (WHERE del SQL).

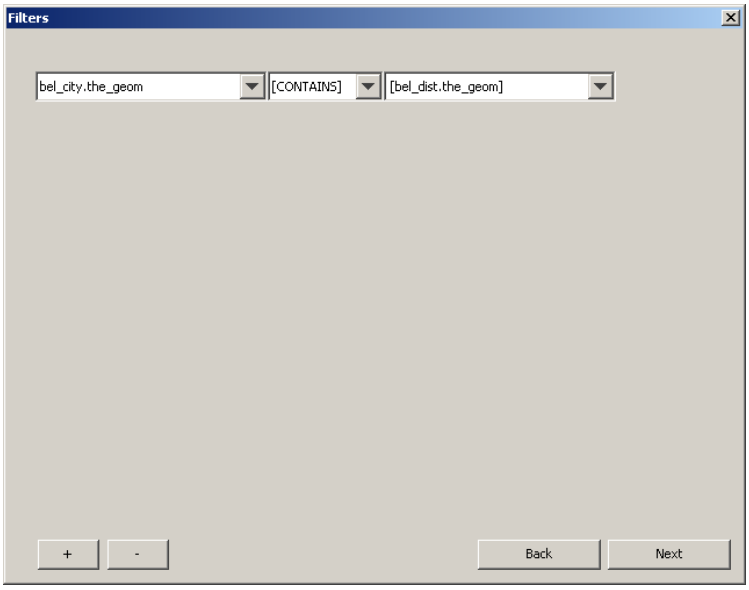


Figura 15 – Selector de filtros de GIS.

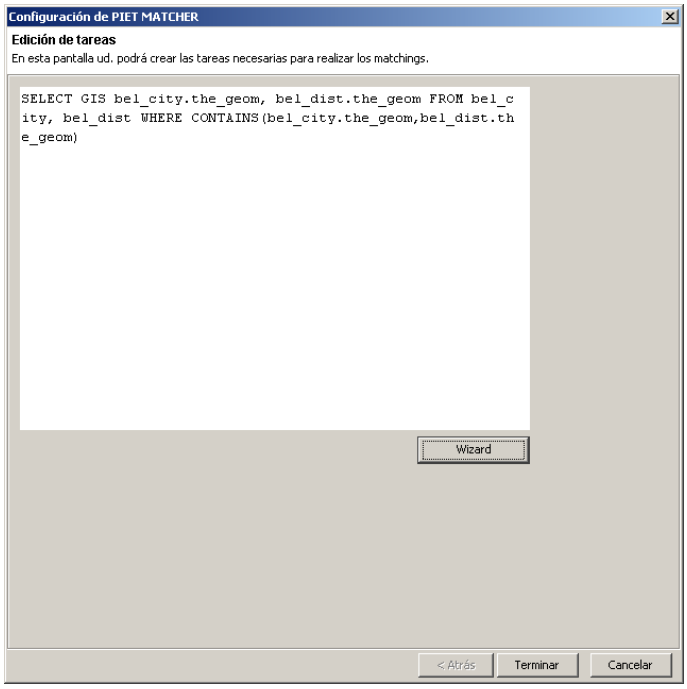


Figura 16 – La consulta GIS producida por el asistente.

La Figura 16 muestra la consulta GIS generada.

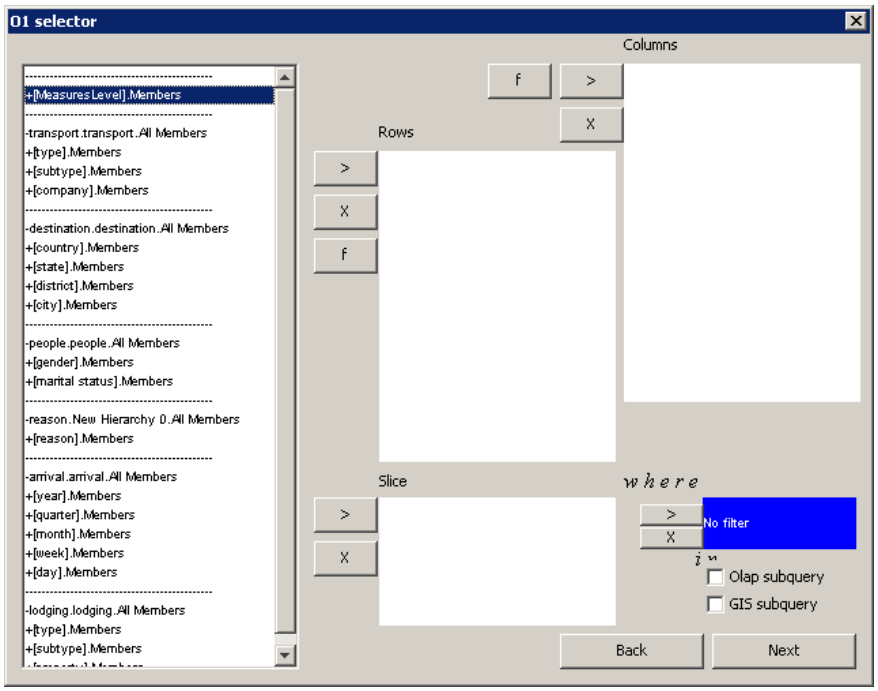

Figura 17 – Pantalla de creación de consultas OLAP.

Al elegir OLAP Query en el selector de consultas, el wizard nos lleva a la pantalla de creación de consultas OLAP, Figura 17, donde podemos elegir los campos procedentes de las distintas dimensiones del cubo. Para acceder a los miembros u otras jerarquías, se puede hacer doble click en los ítems que posean por delante un símbolo "+". Mediante este procedimiento se puede navegar entre los distintos miembros y jerarquías del cubo.

Al armar una consulta OLAP, se puede elegir cuales miembros o jerarquías se mostrarán en las columnas (Columns), y cuales a las filas (Rows). En la lista "Slice" también se pueden asignar miembros para el filtrado de la consulta (Equivalente al Slice de Piet-QL o Where de MDX). Finalmente, también es

posible asignar miembros o jerarquías a la cláusula Where, en este caso los elementos de la dimensión seleccionada se filtrarán por una sub-consulta que a su vez puede ser GIS u OLAP. En este último caso el asistente, al presionar "Next", continuará con otra pantalla para generar la subconsulta. Si no se incluye la clausula Where, el asistente asume que se llegó al final del proceso de generación de la consulta y vuelve a la pantalla inicial mostrando la consulta generada.

# Capítulo 7

## EJEMPLOS

Concluimos este documento mostrando una serie de ejemplos que ilustran la forma en que se evalúa una consulta Piet-QL. Se incluyen consultas de distintos tipos, para abarcar un amplio espectro de problemas a resolver.

### Query 1

*Visitantes, estadía y gastos para los distintos tipos de personas y alojamientos en provincias cruzadas por el río Dyle durante 2007.*

### Consulta Piet-QL:

```
SELECT CUBE [Measures].[visitors], [Measures].[stay], [Measures].[expenditures] ON COLUMNS,
        [lodging].[all], [people].[all] ON rows
 from [tourist]
 where [destination].[all] IN(
                    SELECT GIS bel_prov
                    FROM bel_prov, bel_river
                    WHERE ST_INTERSECTS(bel_river.the_geom,bel_prov.the_geom)
                    AND bel_river.name='Dyle'
 )
 slice [arrival].[2007]
```

### Subconsulta GIS

```
                    SELECT GIS bel_prov
                    FROM bel_prov, bel_river
                    WHERE ST_INTERSECTS(bel_river.the_geom,bel_prov.the_geom)
                    AND bel_river.name='Dyle'
```

| ID | NAME |
|---|---|
| 2 | Brabant |

### Traducción final

```
SELECT {[Measures].[visitors], [Measures].[stay], [Measures].[expenditures]} ON COLUMNS,
  Crossjoin(Hierarchize({[destination.destination].[all].[Belgium].[Brabant]}), {([lodging.lodging].[all],
[people.people].[all])}) ON ROWS
```

```
from [tourist]
where [arrival.arrival].[anytime].[2007]
```

## Resultado

| | | | Medidas | | |
|---|---|---|---|---|---|
| **destination** | **lodging** | **people** | **visitors** | **stay** | **expenditures** |
| **Brabant** | **all** | **all** | 267.695 | 36,771 | 944.263.862,84 |

Slicer: [year=2007]

# Query 2

*Cantidad de visitantes, estadía promedio y promedio de gastos para los hospedajes, por tipo de personas en distritos intersecados por el río Dyle en 2007.*

## Consulta Piet-QL:

```
SELECT CUBE [Measures].[visitors], [Measures].[stay], [Measures].[expenditures] ON COLUMNS,
        [people].[all], [lodging].[all] ON rows
 from [tourist]
 where [destination].[all] IN(
                    SELECT GIS bel_dist
                    FROM bel_river,bel_dist
                    WHERE ST_INTERSECTS(bel_dist.the_geom,bel_river.the_geom)
                    AND bel_river.name='Dyle'
)
 slice [arrival].[2007]
```

## Subconsulta GIS

```
                    SELECT GIS bel_dist
                    FROM bel_river,bel_dist
                    WHERE ST_INTERSECTS(bel_dist.the_geom,bel_river.the_geom)
                    AND bel_river.name='Dyle'
```

| ID | NAME |
|---|---|
| 24 | Leuven |
| 25 | Nijvel |

## Traducción final

```
SELECT {[Measures].[visitors], [Measures].[stay], [Measures].[expenditures]} ON COLUMNS,
  Crossjoin(Hierarchize({[destination.destination].[all].[Belgium].[Brabant].[Nijvel],
[destination.destination].[all].[Belgium].[Brabant].[Leuven]}), {([people.people].[all], [lodging.lodging].[all])})
ON ROWS
from [tourist]
where [arrival.arrival].[anytime].[2007]
```

## Resultado

| | | | **Medidas** | | |
|---|---|---|---|---|---|
| **destination** | **people** | **lodging** | **visitors** | **stay** | **expenditures** |
| **Leuven** | **all** | **all** | 68.371 | 37,793 | 249.424.150,21 |
| **Nijvel** | **all** | **all** | 57.436 | 38,135 | 213.046.100,72 |

Slicer: [year=2007]

# Query 3

*Distritos cruzados por ríos*.

## Consulta Piet-QL:

```
SELECT GIS DISTINCT(bel_dist.name)
FROM bel_dist, bel_river
WHERE crosses(bel_river,bel_dist)
```

## Traducción final

```
SELECT GIS DISTINCT bel_dist.name
FROM bel_dist, bel_river
WHERE ST_CROSSES(bel_river,bel_dist)
```

## Resultado (parcial)

| NAME | NAME | NAME |
|---|---|---|
| Aalst | Gent | Oostende |
| Aarlen | Halle-Vilvoorde | Oudenaarde |
| ..... | ..... | ..... |

## Query 4

*Nombres de los ríos contenidos por provincias donde los visitantes superan las 650000 personas.*

## Consulta Piet-QL:

```
SELECT GIS distinct(bel_river.name)
FROM bel_river, bel_prov
WHERE ST_CONTAINS(bel_prov.the_geom,bel_river.the_geom)
AND ( bel_prov IN(
    SELECT CUBE filter([destination].[state].Members,[Measures].[visitors]>650000)
    FROM [tourist]
) )
```

## Subconsulta OLAP

```
SELECT CUBE filter([destination].[state].Members, [Measures].[visitors]>650000) FROM [tourist]
```

| **Medidas** | |
|---|---|
| **destination** | **visitors** |
| **Antwerpen** | 699.368 |
| **Brabant** | 1.051.968 |
| **Liege** | 722.800 |

## Traducción final

```
SELECT DISTINCT bel_river.name FROM bel_river,bel_prov WHERE
ST_CONTAINS(BEL_PROV.THE_GEOM,BEL_RIVER.THE_GEOM) AND ((bel_prov.PIET_ID IN
(84,83,88)))
```

## Resultado (parcial)

| NAME |
|---|
| Ambleve |
| Dyle |
| K.Bocholt-Herentals |
| .... |

## Query 5

*Listar los nombres de las provincias de Bélgica cuyos visitantes superan las 200000 personas durante el 2007.*

## Consulta Piet-QL:

```
SELECT GIS bel_prov.name
FROM bel_prov
WHERE ( bel_prov IN(
                SELECT CUBE filter([destination].[state].Members,
            [Measures].[visitors]>200000)
                FROM [tourist]
                SLICE [arrival].[2007]
        ))
```

## Subconsulta OLAP

```
SELECT CUBE filter([destination].[state].Members,
        [Measures].[visitors]>200000)
            FROM [tourist]
            SLICE [arrival].[2007]
```

| | **Medidas** |
|---|---|
| **destination** | **visitors** |
| **Brabant** | 267.695 |

Slicer: [year=2007]

## Traducción final

```
SELECT bel_prov.name FROM bel_prov WHERE  ((bel_prov.PIET_ID IN (84)))
```

## Resultado

| NAME |
|---|
| Brabant |

## Query 6

*Nombres de las ciudades de Bélgica cuyos visitantes superan las 4000 personas durante 2007.*

## Consulta Piet-QL:

```
SELECT GIS bel_city.name
FROM bel_city
WHERE ( bel_city IN(
SELECT CUBE filter([destination].[city].Members, [Measures].[visitors]>4000)
from [tourist]
slice [arrival].[2007]
        ))
```

## Subconsulta

```
SELECT CUBE filter([destination].[city].Members, [Measures].[visitors]>4000)
from [tourist]
slice [arrival].[2007]
```

| | **Medidas** |
|---|---|
| **destination** | **visitors** |
| **Antwerpen** | 14.724 |
| **Brussel** | 14.904 |
| **Jalhay** | 4.660 |
| **Brugge** | 13.740 |

Slicer: [year=2007]

## Traducción final

```
SELECT bel_city.name FROM bel_city WHERE  ((bel_city.PIET_ID IN (182,111,351,180,352)))
```

## Resultado

| NAME |
|---|
| Antwerpen |
| Brugge |
| Brussel |
| Jalhay |
| Jalhay |

# Query 7

*Nombre de las ciudades dentro de provincias intersectadas por el río Dyle*.

## Consulta Piet-QL:

```
SELECT GIS bel_city.name
FROM bel_city
WHERE ( bel_city IN(
 SELECT GIS bel_city
 FROM bel_city, bel_prov, bel_river
 WHERE INTERSECTS(bel_prov.the_geom, bel_river.the_geom )
 AND CONTAINS(bel_prov.the_geom,bel_city.the_geom)
AND bel_river.name='Dyle'
          ))
```

## Subconsulta

```
 SELECT GIS bel_city
 FROM bel_city, bel_prov, bel_river
 WHERE INTERSECTS(bel_prov.the_geom, bel_river.the_geom )
 AND CONTAINS(bel_prov.the_geom,bel_city.the_geom)
AND bel_river.name='Dyle'
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | Aarschot | 192 | Glabbeek | 316 | Liedekerke | 479 | Sint-Pieters-Woluwe |
| 4 | Affligem | 193 | Gooik | 325 | Linkebeek | 492 | Steenokkerzeel |
| 11 | Anderlecht | 196 | Grez-Doiceau | 326 | Linter | 498 | Ternat |
| 23 | Asse | 197 | Grimbergen | 331 | Londerzeel | 499 | Tervuren |
| 40 | Beauvechain | 199 | Haacht | 334 | Lubbeek | 505 | Tielt-Winge |
| 43 | Beersel | 203 | Halle | 339 | Machelen | 506 | Tienen |
| 44 | Begijnendijk | 218 | Helecine | 351 | Meise | 512 | Tremelo |
| 45 | Bekkevoort | 222 | Herent | 356 | Merchtem | 515 | Tubize |
| 47 | Berchem-Sainte-Agath | 226 | Herne | 367 | Molenbeek-Saint-Jean | 517 | Uccle |
| 53 | Bertem | 234 | Hoegaarden | 371 | Mont-Saint-Guibert | 523 | Villers |
| 56 | Bever | 235 | Hoeilaart | 391 | Nivelles | 525 | Vilvoorde |
| 59 | Bierbeek | 237 | Holsbeek | 401 | Opwijk | 539 | Walhein |
| 69 | Boortmeerbeek | 247 | Huldenberg | 403 | Orp-Jauche | 543 | Warve |
| 75 | Boutersem | 252 | Incourt | 404 | Oud-Heverlee | 545 | Waterloo |
| 76 | Braine-le-Alleud | 253 | Ittre | 408 | Oudergem | 546 | Watermaal-Bosvoorde |
| 77 | Braine-le-Chateau | 254 | Ixelles | 411 | Overijse | 550 | Wemmel |
| 88 | Brussel | 260 | Jette | 416 | Pepingen | 555 | Wezembeek-oppem |
| 98 | Chastre | 261 | Jodoigne | 419 | Perwez | 566 | Zaventem |
| 101 | Chaumont-Gistoux | 265 | Kampenhout | 432 | Ramillies | 569 | Zemst |
| 110 | Court-Saint-Etienne | 266 | Kapelle-op-den-Bos | 435 | Rebecq | 577 | Zoutleeuw |
| 127 | Diest | 270 | Keerbergen | 441 | Rixensart | | |
| 129 | Dilbeek | 278 | Koekelberg | 445 | Roosdaal | | |
| 135 | Drogenbos | 282 | Kortenaken | 446 | Rotselaar | | |
| 153 | Etterbeek | 283 | Kortenberg | 459 | Schaarbeek | | |
| 155 | Evere | 286 | Kraainem | 461 | Scherpenheuvel-Ziche | | |
| 171 | Forest | 291 | La Hulpe | 468 | Sint-Genesius-Rode | | |
| 176 | Galmaarden | 297 | Landen | 469 | Sint-Gillis | | |
| 177 | Ganshoren | 299 | Lasne | 471 | Sint-Joost-ten-Node | | |
| 182 | Geetbets | 306 | Lennik | 473 | Sint-Lambrechts-Woll | | |
| 184 | Genappe | 311 | Leuven | 478 | Sint-Pieters-Leeuw | | |

## Traducción final (extracto)

```
SELECT bel_city.name FROM bel_city WHERE  ((bel_city.PIET_ID='639' OR bel_city.PIET_ID='510' OR
bel_city.PIET_ID='637' OR bel_city.PIET_ID='380' OR bel_city.PIET_ID='249' OR bel_city.PIET_ID='377' OR
bel_city.PIET_ID='505' OR bel_city.PIET_ID='633' OR bel_city.PIET_ID='376' OR….
 ……OR bel_city.PIET_ID='391' OR bel_city.PIET_ID='134' OR bel_city.PIET_ID='644' OR
bel_city.PIET_ID='385' OR bel_city.PIET_ID='513' OR bel_city.PIET_ID='640'))
```

## Resultado (extracto)

| | | | | | |
|---|---|---|---|---|---|
| Aarschot | Court-Saint-Etienne | Herne | Lasne | Pepingen | Tienen |
| Braine-le-Alleud | Grimbergen | Kortenaken | Opwijk | Sint-Pieters-Woluwe | Zoutleeuw |
| Braine-le-Chateau | Haacht | Kortenberg | Orp-Jauche | Steenokkerzeel | |
| Brussel | Halle | Kraainem | Oud-Heverlee | Ternat | |
| Chastre | Helecine | La Hulpe | Oudergem | Tervuren | |
| Chaumont-Gistoux | Herent | Landen | Overijse | Tielt-Winge | |

# Query 8

*Hospedajes  y personas en ciudades de Bélgica, donde los visitantes superaron 4.000 visitantes durante el 2007.*

## Consulta Piet-QL

```
SELECT CUBE [people].[all], [lodging].[all] ON ROWS
from [tourist]
WHERE [destination].[all] IN(
SELECT CUBE filter([destination].[city].Members, [Measures].[visitors]>4000)
 from [tourist]
 slice [arrival].[2007])
```

## Subconsulta

```
SELECT CUBE filter([destination].[city].Members, [Measures].[visitors]>4000)
FROM [tourist]  SLICE [arrival].[2007]
```

| | **Medidas** |
|---|---|
| **destination** | **visitors** |
| **Antwerpen** | 14.724 |
| **Brussel** | 14.904 |
| **Jalhay** | 4.660 |
| **Brugge** | 13.740 |

Slicer: [year=2007]

## Traducción final

```
 SELECT Crossjoin( Hierarchize(
{[destination.destination].[all].[Belgium].[Antwerpen].[Antwerpen].[Antwerpen],[destination.destination].[all].[Belgium].[Brabant].[Brussel-
Hoofdstad].[Brussel],[destination.destination].[all].[Belgium].[Liege].[Verviers].[Jalhay],[destination.destination].[all].[Belgium].[West-Vlaanderen].[Brugge].[Brugge]}), {([people].[all],[lodging].[all])}) ON ROWS  FROM [tourist]
```

## Resultado

| destination | | | |
|---|---|---|---|
| **Antwerpen** | **Brussel** | **Jalhay** | **Brugge** |
| **people** | **people** | **people** | **people** |
| **All** | **all** | **all** | **all** |
| **lodging** | **lodging** | **lodging** | **lodging** |
| **All** | **all** | **all** | **all** |
| 39,12 | 38,905 | 39,066 | 38,967 |

Slicer: [subtype=HOTEL]

# Capítulo 8

## CONCLUSION

En este documento hemos descripto el diseño e implementación de Piet-QL, un lenguaje de consulta que soporta el modelo de Piet para la integración de GIS y sistemas OLAP. El modelo implementado permite expresar y resolver consultas que integran la información espacial y multidimensional, sin que ello implique modificar las estructuras de datos de los sistemas existentes. Esto significa que manteniendo las principales prestaciones de los sistemas GIS y OLAP actuales.

La experimentación realizada (reportada en [8], por lo que no la hemos repetido en este documento), demostró que sobrecarga de procesamiento de la integración y traducción es insignificante con respecto a la ejecución de las consultas por parte de GIS y OLAP.

El lenguaje desarrollado es simple e intuitivo, a la vez que fácil de aprender por personas que conozcan MDX y SQL. Una de las principales características de Piet-QL es que mantiene casi todas las características de los lenguajes que integra (MDX, SQL, PostGIS).

El concepto de implementar un motor débilmente acoplado permite una abstracción sobre los motores de GIS y OLAP. Esta abstracción lo hace fácilmente adaptable a otros motores de GIS y OLAP, así como al crecimiento individual de cada una de estas tecnologías.

Finalmente, el asistente PietQL-Wizard desarrollado permite la generación de consultas por parte de usuarios sin conocimiento de los lenguajes subyacentes.

Este lenguaje permite sentar las bases para resolver problemas más complejos aún, como el soporte a modelos temporales. De hecho, en [9], se utiliza Piet-QL para extenderlo en ese sentido.

# BIBLIOGRAFIA

[1] Ariel Escribano, Leticia Gomez, Bart Kuijpers, Alejandro A. Vaisman. Piet: a GIS-OLAP Implementation. In DOLAP, pages 73-80, 2007.

[2] Vega López, R. Snodgrass, B. Moon. Spatiotemporal Aggregate Computation: A Survey. IEEE Transactions on Knowledge and Data Engineering 17(2), 2005.

[3] Y. Bédard, S. Rivest, and M. Proulx. Spatial Online Analytical Processing (SOLAP): Concepts, Architectures, and Solutions from a Geomatics Engineering Perspective. In Wrembel-Koncilia, editors, Data Warehouses and OLAP: Concepts, Architectures and Solutions, Chapter 13, pages 298-319. IRM Press, 2007.

[4] S. Rivest, Y. Bédard, P. Marchand. Modeling Multidimensional Spatio-temporal Data Warehouses in a Context of Evolving Specifications. Geomatica, 55 (4), 2001.

[5] S. Shekhar, C. Lu, X. Tan, S. Chawla, R. Vatsavai. Mapcube: A Visualization

Tool for Spatial Data Warehouses. In Harvey Miller and Jiawei Han, editors, Geographic Data Mining and Knowledge Discovery (GKD), pages 74-109. Taylor and Francis, 2001.

[6] J. Han, N. Stefanovic, K. Koperski. Selective materialization: An Efficient Method for Spatial Data Cube Construction. In Proceedings of PAKDD'98, pages 144-158, 1998.

[7] Joel da Silva, Ausberto S. Castro Vera, Anjolina G. de Oliveira, Robson do Nascimento Fidalgo, Ana Carolina Salgado, Valéria Cesário Times. Querying Geographical Data Warehouses with GeomDQL. In SBBD, pages 223-237, 2007.

[8] Leticia I. Gómez, Alejandro A. Vaisman, Sebastián Zich. Piet-QL: a Query Language for GIS-OLAP integration. ACM-SIGSPATIAL, pages 27-36, 2008.

[9] Pablo Bisceglia, Leticia I. Gómez, Alejandro A. Vaisman. Temporal SOLAP: Query Language, Implementation, and a Use Case. In AMW, pages 102-113, 2012.