Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
*Aplicaciones con Tecnología en Internet*

# GENERADOR GRÁFICO DE CONSULTAS PARA LA RECUPERACIÓN DE DATOS INTEGRADO EN ECLIPSE

Trabajo Especial de Grado
Presentado ante la ilustre
Universidad Central de Venezuela
Por los Bachilleres
Héctor Enrique Martínez Otero
Anely Nahir Poleo Marín
Para optar al título de
Licenciado en Computación

Tutores:
Prof. Sergio José Rivas Atanacio
Prof. Joselito de Sousa

Mayo, 2008.

# Reconocimientos y agradecimientos

En las próximas líneas plasmamos nuestros agradecimientos a los sitios y personas que otorgaron tan significativos aportes durante la elaboración de este Trabajo Especial de Grado.

Primeramente agradecemos a Dios, por hacer posible la culminación de este proyecto con éxito, y por darnos valor para continuar en sus etapas más difíciles.

Extendemos los agradecimientos a nuestros tutores, por ser parte clave en este proceso, por brindarnos su apoyo, conocimiento y guía para dirigir los objetivos planteados hacia el cumplimiento de la meta.

Agradecemos también al Círculo de compañeros de estudio (y de vida), que nos inyectaron fuerzas para animarnos en momentos de decaimiento; siempre tendrán un pedacito reservado en nuestros corazones.

Damos nuestros agradecimientos, a nuestros familiares y amigos, todo el apoyo brindando; especialmente a la señora Alba Martínez por brindarnos su apoyo y ofrecernos su oficina como la cuna de este proyecto.

Finalmente, agradecemos a la Escuela de Computación de la UCV, ya que cada una de las personas que a ella pertenecen, fueron partícipes en menor o mayor grado en los cimientos que influyeron en la formación de nuestras facultades como profesionales, como persona y como creadores de este proyecto.

# Resumen

En la actualidad, la mayoría de los sistemas de software almacenan su información haciendo uso de bases de datos. Cada uno de los Sistemas Manejadores de Bases de Datos, manejan esta información de diferentes maneras; lo cual hace que la recuperación de los datos dependa en gran parte del Sistema Manejador de Bases de Datos que se utilice.

El presente Trabajo Especial de Grado, ofrece realizar una herramienta que permita a los usuarios inexpertos en la manipulación de consultas de recuperación de datos (SQL). La construcción de una consulta se realizará de forma gráfica, permitiendo hacer abstracción del SMBD utilizado.

Esta herramienta gráfica de creación consultas de recuperación de datos, se implantó como un plug-in del entorno de desarrollo Eclipse, debido a que las consultas de recuperación de datos SQL son utilizadas en mayor parte en el momento en que se desarrollan los sistemas de software.

Para llevar a cabo la solución al problema, y desarrollar una herramienta de generación de consultas SQL independiente del SMBD y de su sintaxis SQL, se realizó lo siguiente:

- Se definió un proceso de desarrollo basado en Programación Extrema (XP) y tomando prácticas y principios de la metodología Modelación Ágil (AM).
- Como la herramienta debe permitir generar una sintaxis SQL para diferentes SMBD, se hizo que la misma fuera extensible mediante archivos de configuración basados en XML, en los cuales se especifican las características de cada base de datos. Las soportadas por el plug-in actualmente son MySQL 4.1, Postgres 8.1 y Oracle 9i.
- Se creó un módulo de base de datos genérico que permite establecer comunicación con los diferentes SMBD relacionales; además de poder obtener el diccionario de datos que en estos se encuentran.
- Se crearon interfaces de manejos de proyectos, que aplica los patrones de interfaces a las del entorno de desarrollo ECLIPSE.

**Palabras claves**: Plug-in, Eclipse, SQL, Base de datos.

**Contactos:**

Héctor Martínez (hectorenriquemo@gmail.com)
Anely Poleo (spidergirl672@gmail.com).
Sergio Rivas (srivas@gmail.com)
Joselito de Sousa (josedes@gmail.com)

# Tabla de contenido

# Tabla de imágenes

# Introducción

Los sistemas de bases de datos se han convertido en herramientas útiles para almacenar información, es por ello que los desarrolladores están construyendo frecuentemente aplicaciones que acceden a estos sistemas; debido a las potencialidades que estos ofrecen sobre el manejo de la información.

El proceso de recuperación de la información almacenada, se lleva a cabo utilizando el lenguaje SQL, el cual define reglas que pueden ser indicadas a través de los lenguajes de programación utilizados con el fin de obtener resultados desde las bases de datos.

En este sentido, el desarrollador ya cuenta con bibliotecas de alto nivel para los lenguajes de programación que permiten acceder a las bases de datos, pero sigue teniendo la necesidad de conocer el conjunto de reglas SQL para cada sistema de base de datos y muchas veces no aprovecha todas las características de estos sistemas por el desconocimiento de la sintaxis SQL.

El problema que se busca resolver con este trabajo especial de grado es la generación de estas reglas permitiendo hacer abstracción del lenguaje procedural que propone ANSI SQL. Así, se busca la realización de una herramienta de generación de consultas SQL, que permita recuperar información desde las bases de datos usando una interfaz gráfica sencilla y que esté integrada en un ambiente de desarrollo para facilitar su uso por parte de los desarrolladores. Adicionalmente, se desea que la herramienta sea extensible, para que en el futuro se puedan integrar a esta, nuevos sistemas de bases de datos de forma sencilla.

A través del desarrollo del documento se presentarán los conceptos y tecnologías implicadas en el desarrollo de la herramienta generadora antes mencionada. Este proceso se llevará a cabo en el entorno de desarrollo integrado Eclipse 3.1 y hará uso de la sintaxis SQL de los sistemas de bases de datos PostgreSQL, MySQL y Oracle.

El documento será estructurado en dos secciones: *Marco teórico*, el cual presenta la información asociada con la construcción de la herramienta, y el *Marco aplicativo*, que contiene información acerca del desarrollo de la herramienta.

El marco teórico será distribuido en varios capítulos:

- **Capítulo I**: Busca presentar información acerca del estándar ANSI SQL que propone el lenguaje que será utilizado por la herramienta, así como las diferentes sintaxis de los sistemas de bases de datos que deriva de este estándar.

- **Capítulo II**: Tiene como objetivo crear las bases para comprender el funcionamiento, la estructura y el proceso de desarrollo de una herramienta correspondientes al IDE Eclipse.

En cuanto al marco a aplicativo se contempla lo siguiente:

- **Capítulo III**: Muestra como se llevó a cabo la construcción de la herramienta, siguiendo el proceso de desarrollo XP y la metodología de desarrollo AM. Se toman en cuenta: definición de las funcionalidades, planificaciones, definición de diagramas de clases, realización de pruebas y otros aspectos relevantes del proceso de construcción.

Finalmente, se presentan las conclusiones y se sugieren futuras extensiones de la herramienta de generación de consultas, que permitan agregarles nuevas funcionalidades.

# Propuesta

Los sistemas de bases de datos guardan un conjunto de objetos que son manipulados (generalmente por aplicaciones) a través de consultas para obtener información.

Dada la complejidad y amplitud del lenguaje procedural SQL propuesto por el ANSI[1], los proveedores de bases de datos adaptan un subconjunto de este, de acuerdo a sus necesidades, formando así su propio lenguaje. Así es como surge la necesidad por parte del desarrollador de aprender a utilizar los sublenguajes del ANSI SQL para trabajar con cada sistema de bases de datos; lo cual tiende a provocar que los desarrolladores se enfoquen solo en las instrucciones que más necesitan y muchas veces desaprovechen las potencialidades del sistema de base de datos.

Existen aplicaciones que permiten realizar las consultas de las bases de datos de una forma más sencilla e intuitiva. Generalmente, son herramientas que están dirigidas a desarrolladores expertos en el manejo de bases de datos, ofrecen acceso a un solo sistema de base de datos y no se encuentran integradas en un entorno de desarrollo de software.

Se propone el desarrollo de una herramienta de generación de consultas para diferentes sistemas de bases de datos relacionales, haciendo uso de una interfaz gráfica de usuario que no requiere de amplio conocimiento de los lenguajes de consulta por parte del desarrollador. Dicha herramienta estará dirigida a la construcción de aplicaciones de software, por lo que estará integrada en un ambiente de desarrollo.

La herramienta estará dotada de una interfaz gráfica que permitirá al desarrollador indicar las consultas y los parámetros de la base de datos sobre la cual desea ejecutarlas. También contará con un módulo que genere la consulta en sintaxis SQL, de acuerdo a las entradas colocadas por el desarrollador y tomando en cuenta el sistema de base de datos seleccionado.

Adicionalmente, se permitirá al desarrollador ejecutar la consulta generada para que pueda verificar si es el resultado deseado.

La herramienta será construida, utilizando un conjunto de tecnologías que permitan a ésta ser multiplataforma y sencilla de integrar en un entorno de desarrollo. Estas tecnologías se presentan en la siguiente tabla.

[1] El Instituto Nacional Estadounidense de Estándares (por sus siglas en inglés: American National Standards Institute).

**Tabla 1: Tecnologías para construir el generador de consultas**

| Tecnología | Razón |
|---|---|
| Java | Por un ser lenguaje de programación multiplataforma. |
| Eclipse 3.1 | Porque permite agregar nuevos módulos (funcionalidades) y brinda soporte para la extensión de su plataforma, las cuales permiten crear y publicar nuevos módulos. |
| JDBC 3.0 | Para establecer la comunicación con las bases de datos y la obtención de metadatos, a través de Java. |
| SWT y JFace | Para crear los componentes multiplataforma para la interfaz de usuario. |
| GEF 3.1 y Draw2d 3.1 | Como herramientas de alto nivel para crear editores gráficos complejos. |
| MySQL 4.1 | Como base de datos soportada por la aplicación. |
| PostgreSQL 8.1 | Como base de datos soportada por la aplicación. |
| Oracle 9i | Como base de datos soportada por la aplicación. |

# Arquitectura de la solución propuesta

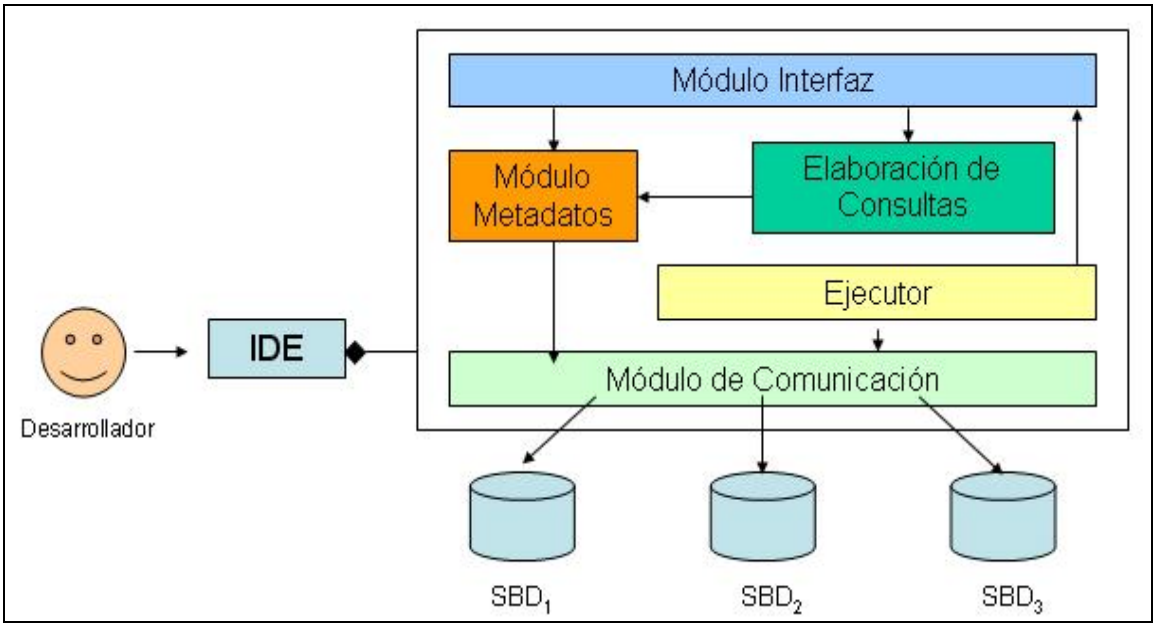


**Ilustración 1: Arquitectura del generador de consultas de recuperación de datos**

El **desarrollador** que interactúa con la herramienta gráfica de generación de consultas de recuperación de datos puede ser inexperto en la manipulación de bases de datos y tiene conocimientos básicos de las sintaxis de consulta.

Un **IDE** es un entorno de desarrollo en el cual se encuentran integradas un conjunto de funcionalidades para la creación de programas de aplicación en diferentes lenguajes de programación. Consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI.

La interfaz gráfica de usuario estará implementada por el **Módulo de Interfaz**, el cual permitirá al desarrollador indicar las consultas de recuperación de datos.

El **Módulo de Metadatos** es el encargado de consultar información acerca de los objetos que existen en la base de datos, por ejemplo las tablas existentes, sus atributos y tipos de datos asociados a estos.

La interpretación de la consulta indicada por el desarrollador se lleva a cabo en el módulo de **Elaboración de consultas**, quien además, crea la estructura de la consulta dependiendo del sistema de bases de datos que se esté utilizando.

La consulta elaborada por el módulo de elaboración de consulta es procesada por el módulo **Ejecutor**.

La comunicación con los diferentes sistemas de bases de datos se realiza a través del **Módulo de Comunicación**.

**$SBD_i$** son los diferentes sistemas de bases de datos sobre los cuales el desarrollador puede realizar las consultas a través de la herramienta.

# Objetivos

## *Objetivo general*

Desarrollar un módulo (plug-in) para la generación de consultas de recuperación de datos de forma gráfica, para diferentes sistemas de bases de datos e integrado en el entorno de desarrollo de software Eclipse.

## *Objetivos específicos*

- Aplicar el proceso de desarrollo XP[2] junto con AM[3] para la construcción de una herramienta gráfica de generación de consultas para la recuperación de datos.
- Desarrollar una interfaz gráfica que permita al desarrollador inexperto en la manipulación de bases de datos, crear consultas para la recuperación de datos.
- Crear una estructura común que permita utilizar de igual forma diferentes sistemas de bases de datos.
- Elaborar un módulo para el manejo de conexiones y obtención de metadatos de diferentes sistemas de bases de datos.

[2] eXtreme Programing: Proceso de desarrollo ligero.
[3] Agile Modeling: Conjunto de prácticas y principios para el desarrollo de software de forma ágil.

- Elaborar un módulo que genere y ejecute consultas de recuperación de datos, independientemente de la sintaxis SQL utilizada por los sistemas de bases de datos.
- Integrar los módulos anteriores en un entorno de desarrollo basado en plug-in.
- Realizar pruebas que contemplen un conjunto de estructuras de consultas de recuperación de datos sobre diferentes sistemas de bases de datos.

# Capítulo I. ANSI SQL

El Lenguaje de Consulta Estructurado, llamado SQL por las siglas en inglés de Structured Query Language, es un lenguaje de acceso a bases de datos que explota la flexibilidad y potencia de los sistemas relacionales, permitiendo gran variedad de operaciones sobre los mismos con el fin de recuperar información de interés de una base de datos, de una forma sencilla.

La historia de SQL, empieza en 1974 cuando IBM desarrolla un lenguaje para la especificación de las características de las bases de datos que adoptaban el modelo relacional. Este lenguaje se llamaba SEQUEL (Structured English Query Language), fue revisado entre 1974 y 1975 bajo el nombre SEQUEL/2, y luego cambio su nombre a SQL por problemas legales.

El prototipo (System R), basado en este lenguaje, tuvo éxito en IBM y en otras compañías.

En 1986, el ANSI[4] adoptó el dialecto SQL de IBM como estándar para los lenguajes relacionales y en 1987 se transformó en estándar ISO. Esta versión del estándar tenía el nombre SQL/86. En 1989, ANSI definió el SQL89, basado en el anterior pero con una serie de mejoras (definición de claves primarias, integridad de los datos, entre otras). Una característica importante definida era la posibilidad de utilizarse a través de dos interfaces: interactivamente o dentro de programas de aplicación.

La primera versión del SQL-89 tiene tres partes:

- **El Lenguaje de definición de datos (DDL):** Contiene todas las instrucciones para definir el esquema de una base de datos, como son: *create*, *alter* y *drop*.
- **El lenguaje de manipulación de datos (DML)**: Contiene las instrucciones de manejo de las tablas como son: *select*, *insert*, *delete* y *update*, y para control de concurrencia como: *commit* y *rollback*.
- **El lenguaje de control de datos (DCL)**: Contiene aquellas instrucciones para cambiar y revocar permisos de acceso a los datos de la base de datos, como son: *grant* y *revoke*.

En la actualidad el SQL es el estándar usado por la mayoría de los Sistemas Manejadores de Bases de Datos comerciales. Y, aunque la diversidad de elementos particulares que incluyen las distintas implementaciones comerciales del lenguaje es amplia, el soporte al estándar SQL-92 es general y muy amplio.

El SQL proporciona las siguientes funcionalidades para la definición y manipulación de una base de datos relacional:

- Consulta de datos.

[4] http://webstore.ansi.org/ansidocstore/default.asp

- Definición de datos.
- Definición de vistas.
- Manipulación de datos.
- Concesión y denegación de permisos.
- Implementación de restricciones de integridad y controles de transacción.
- Manipulación y alteración de esquemas.

## 1.1. Operación de consulta de datos

Debido a que la herramienta solo trabajará con consultas de recuperación de datos, se estudiará la operación del estándar que soporta este tipo de consultas.

La operación de consultas se realiza mediante una sentencia, la cual está compuesta de un conjunto de cláusulas. La herramienta que se desea construir realizará las consultas de bases de datos haciendo uso de esta operación.

De forma general, la sintaxis de la sentencia de consulta es la siguiente[5]:

```
SELECT [ALL | DISTINCT ]
{ * | expresión_columna_1 [AS c_alias_1]
[, expresión_columna_2 [AS c_alias_2] [,…]]}
FROM nombre_tabla_1 [t_alias_1][, nombre_tabla_2 [t_alias_2] [,…]]
WHERE condicionWhere
[GROUP BY expresión_columna_group_1 [, expresión_columna_group_2] [,…]]
[HAVING condicionHaving]]
[{UNION [ALL] | INTERSECT | EXCEPT} SELECT …]
[ORDER BY nombre_campoi1 [ASC | DESC]
      [nombre_campoj1 [ASC | DESC]][,…]]];
```

**Ilustración 2: Sintaxis SQL general**

A continuación se describen cada una de las cláusulas y ciertos elementos que se encuentran en la expresión de consulta:

### *1.1.1. Cláusulas*

- **SELECT**: Indica que la operación a realizar es una consulta, y además señala las columnas que se desean consultar. Se puede incluir la opción UNIQUE o DISTINCT para no obtener filas duplicadas en el resultado.
- **FROM**: Especifica las tablas en las que se encuentran los atributos implicados en la consulta.

[5] En el anexo A se puede encontrar esta sintaxis en notación BNF.

- **WHERE**: Especifica la condición de búsqueda que se requiere para seleccionar las filas deseadas.

- **GROUP BY**: Indica cuales filas deben agruparse sobre un valor común de las columnas especificadas. Agrupa el resultado de una consulta en base a uno o varios atributos, devolviendo una única fila por grupo. Los atributos que aparezcan en esta cláusula, deben estar en la cláusula SELECT.

- **HAVING**: Especifica una condición de grupo, seleccionándose sólo aquellos grupos que cumplan la condición especificada. Conceptualmente es como el WHERE, salvo que éste actúa a nivel de tuplas.

- **ORDER BY**: Permite ordenar el resultado en base a los atributos especificados, de forma ascendente o descendente. Los atributos de ordenación deben ser atributos o expresiones que aparezcan en la cláusula SELECT.

### *1.1.2. Funciones Agregadas*

Devuelven un valor único, numérico, como resumen de la información relativa a atributos. No se puede combinar una función agregada, con columnas que devuelvan más de un valor, a menos que la consulta contenga una cláusula GROUP BY.

- **COUNT**: Contador de filas (totalizador) específicas en una consulta.

- **AVG**: media aritmética de un conjunto de valores numéricos.

- **SUM**: Suma de atributos o expresiones numéricos.

- **MAX**: Valor máximo de un atributo o expresión numéricos.

- **MIN**: Valor mínimo de un atributo o expresión numéricos.

### *1.1.3. Operadores*

Los siguientes operadores se pueden utilizar para expresar condiciones de fila (cláusula WHERE) o de grupo (cláusula HAVING).

#### De comparación

- **BETWEEN**: Permite especificar un intervalo de valores en cadenas, números y fechas.

- **Lógicos (AND, OR, NOT)**: Permiten realizar conjunciones, disyunciones y negaciones de las expresiones de una consulta.

- **LIKE**: Utilizado en la comparación de cadenas para la búsqueda de un patrón.

- **IN**: Permite saber si el valor de un campo se encuentra en una subconsulta de selección de una columna.

- **Cuantificadores (ANY, SOME, ALL)**: Permiten conocer la existencia de valores en una subconsulta.
- **Existencial (EXISTS):** Devuelve verdadero si una subconsulta es vacía o no.
- **Otros**: <=, >, >=, <>, =.

### De combinación de consultas o de conjuntos

- **UNION**: combina el resultado de dos sentencias SELECT en un único resultado. Este resultado se compone de todos los registros devueltos en ambas sentencias.
- **EXCEPT**: Combina el resultado de dos sentencias SELECT en un único resultado. Este resultado se compone de los registros que se encuentra en la primera sentencia y no en la segunda.
- **INTERSECT**: Combina el resultado de dos sentencias SELECT en un único resultado. Este resultado se compone de los registros que se encuentran en ambas sentencias al mismo tiempo.

## 1.2. Diferentes sintaxis para la operación de consulta

Generalmente, los sistemas de bases de datos[6] permiten manipular las bases de datos a partir de lenguaje interno propio, el cual es definido a partir de un subconjunto del estándar propuesto por ANSI.

En esta sección se presentarán la sintaxis de la operación de consultas correspondiente al lenguaje interno de cada uno de los sistemas de bases de datos que fueron contemplados para la construcción de la herramienta.

Es importante contemplar las diferencias de la operación entre los sistemas de bases de datos, ya que se tiene como objetivo, que la herramienta ofrezca un procedimiento de generación de consultas independientemente del lenguaje interno de sistema de bases de datos utilizado.

### *1.2.1. Operación de consulta de Oracle*

```
WITH nombre_consulta AS (subconsulta) [, nombre_consulta AS (subconsulta) ]...
SELECT [hint] [ALL | [DISTINCT | UNIQUE]]
        * | expresión_columna_1 [AS c_alias_1]
[, expresión_columna_2 [AS c_alias_2] [,…]]
FROM nombre_tabla_1 [t_alias_1][, nombre_tabla_2 [t_alias_2] [,…]]
[WHERE condicion]
[GROUP BY expresion | [ROLLUP | CUBE] (expresion_1, expresion_2, ...)]
```

6 Es un sistema computarizado cuya finalidad general es almacenar datos y permitir a los usuarios recuperar y actualizar esos datos.

```
       [HAVING condicion]]
       [(UNION [ALL] | INTERSECT | MINUS) SELECT …]
       [ORDER BY expresión_columna_ob_1 [ASC | DESC]
             [expresión_columna_ob_2 [ASC | DESC]][,…]]]
       [FOR UPDATE opciones_for_update];


       nombre_tabla_i = (tabla | vista | vista_materializada)
               [ ( CROSS | NATURAL [INNER | (LEFT | RIGHT | FULL) OUTER] ) JOIN
               (tabla | vista | vista_materializada)]
```

**Ilustración 3: Operación de consulta de Oracle**

## *1.2.2. Operación de consulta de MySQL*

```
       SELECT [ALL | DISTINCT | DISTINCTROW ]
       [HIGH_PRIORITY] [STRAIGHT_JOIN] [SQL_SMALL_RESULT]
       [SQL_BIG_RESULT] [SQL_BUFFER_RESULT] [SQL_CACHE | SQL_NO_CACHE]
       [SQL_CALC_FOUND_ROWS]
       * | expresión_columna_1 [AS c_alias_1]
       [, expresión_columna_2 [AS c_alias_2] [,…]]
       [FROM nombre_tabla_1 [t_alias_1][, nombre_tabla_2 [t_alias_2] [,…]]]
       [WHERE condicion]
       [GROUP BY expresión [ASC | DESC] [,…] [WITH ROLLUP]]
       [HAVING condicion]
       [ORDER BY expresión_columna_ob_1 [ASC | DESC]
             [expresión_columna_ob_2 [ASC | DESC]][,…]]]
       [LIMIT {[offset,] row_count | row_count OFFSET offset}]
       [PROCEDURE procedure_name(argument_list)]
       [INTO OUTFILE 'file_name' export_options | INTO DUMPFILE 'file_name']
       [FOR UPDATE | LOCK IN SHARE MODE]]

       nombre_tabla_i = tabla [[INNER | CROSS] JOIN tabla [ON condición]]
                | tabla [[NATURAL] [LEFT | RIGHT] [OUTER] JOIN tabla [ON condición]]
                | tabla [STRAIGHT_JOIN tabla [ON condición]]
```

**Ilustración 4: Operación de consulta de MySQL**

## *1.2.3. Operación de consulta de PostgreSQL*

```
       SELECT [ ALL | DISTINCT [ ON ( expresión_columna_1 [, expresión_columna_2
[,…]) ] ]
           * | expresión_columna_1 [AS c_alias_1]
        [, expresión_columna_2 [AS c_alias_2] [,…]]
        [FROM nombre_tabla_1 [[AS] t_alias_1][, nombre_tabla_2 [[AS] t_alias_2]
[,…]]]
        [WHERE condition ]
        [GROUP BY expresion_1, expresion_2 [,...]]
        [HAVING condition]
        [ (UNION|INTERSECT|EXCEPT) [ALL] select]
        [ORDER BY expression [ASC|DESC|USING operator] [,…]]
        [LIMIT [contador | ALL]]
        [OFFSET inicio]
        [FOR UPDATE opciones_for_update]

        nombre_tabla_i = [ONLY] (tabla | vista | funcion)
                      [[NATURAL] ([INNER] | (LEFT | RIGHT | FULL) [OUTER]| CROSS)
JOIN
                       (tabla | vista | funcion)
                       [ON condicion | USING (columna_join [,…])]]
```

**Ilustración 5: Operación de consulta de PostgreSQL**

## Síntesis

El lenguaje estructurado de consultas, registrado como un estándar de ANSI, permite el acceso a sistemas de bases de datos relacionales. El acceso se lleva a cabo a través de un conjunto de operaciones utilizadas para establecer permisos de una base de datos, recuperar información, definir objetos de una base de datos, entre otras.

Los creadores de sistemas de bases de datos toman el lenguaje SQL como base para construir su propio lenguaje de consultas interno, incluyendo las adaptaciones que consideran necesarias. De esta forma, se pueden encontrar similitudes entre los lenguajes de los sistemas de base de datos.

En este capítulo se examinó la sintaxis formal de la operación de consulta haciendo uso del lenguaje. También se contemplaron las diferentes adaptaciones de la sintaxis para cada uno de los sistemas de bases de datos que fueron seleccionados como sistemas de bases de datos soportados por la herramienta que se quiere construir.

Una vez visto todo esto, el próximo capítulo muestra información acerca del entorno de desarrollo integrado Eclipse desde el punto de vista de desarrollo de plug-ins.

# Capítulo II. Entorno de desarrollo integrado: Eclipse

Este capítulo tiene como objetivo mostrar aspectos que se deben tomar en cuenta para realizar la implementación de un plug-in que permita generar consultas de recuperación de datos de forma gráfica.

Por ahora se definirá plug-in como el mecanismo que permite incorporar nuevas funcionalidades a una aplicación a demanda del usuario y de acuerdo con sus necesidades. Más adelante se establecerá el concepto de plug-in para Eclipse, al cual se hará regencia durante todo el capítulo.

En las próximas secciones se describen aspectos sobre Eclipse 3.1 que están involucrados en la creación de un plug-in. Primero se establecerán características y conceptos básicos del IDE, lo cual creará la base para comprender la implementación de un plug-in para eclipse, luego se definen aquellos aspectos de creación de un plug-in y, finalmente se muestran algunos módulos ya existentes en la plataforma.

## 2.1. Introducción al proyecto Eclipse[7]

Originalmente, Eclipse fue creado por IBM[8]. Desde el año 2001 lo desarrolla la Fundación Eclipse, una organización independiente sin fines de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Eclipse es una aplicación que provee un entorno de desarrollo integrado por diferentes herramientas, las cuales están dirigidas al desarrollo de software y pueden ser agregadas de acuerdo a las necesidades del usuario.

El proyecto eclipse está dividido en tres partes:

- **Núcleo de la aplicación**: Incluye el subsistema de ayuda (Help), la plataforma para trabajo colaborativo (Team), el Workbench (construido sobre SWT[9] y JFace[10]) y el Workspace para gestionar proyectos.
- **Java Development Tooling (JDT)**: Herramientas de desarrollo de Java[11].
- **Plug-in Development Enviroment (PDE)**: Está diseñado para proporcionar herramientas que ayudan en el desarrollo, prueba, depuración, construcción y despliegue de plug-ins.

---

[7] http://www.eclipse.org/
[8] IBM: Siglas de Internacional Business Machines, una empresa que fabrica y comercializa hardware, software y servicios relacionados con la informática.
[9] Stardar Widget Toolkit: Biblioteca para la construcción de interfaces portables e integradas a la plataforma.
[10] JFace: Biblioteca de alto nivel para la construcción de diferentes tipos de interfaces.
[11] Java: Lenguaje gratuito creado por Sun Microsystems, que distribuye gratuitamente el producto base, denominado JDK.

Mediante Eclipse se pueden crear diversas aplicaciones como: sitios Web, programas Java, programas C++, entre otras. Se pueden integrar a Eclipse otras aplicaciones en forma de plug-ins, que son reconocidos automáticamente al iniciar el mismo.

Como Eclipse está escrito en Java, para su funcionamiento se debe tener instalado el JRE (Java Runtime Enviroment). Eclipse detecta automáticamente su ubicación.

Las funcionalidades que otorga Eclipse se localizan de dos formas diferentes: en un pequeño núcleo conocido como Plataform Runtime o en forma de plug-ins. Existe un conjunto de plug-ins que ya vienen con la plataforma. Entre estos se encuentran: Ant, Compare, Core, CVS, Debug, Help, JDT, JFace, Releng, Scripting, Search, SWT, Text, UI, Update, Team y WebDAV.

La plataforma Eclipse está construida en base a plug-ins o módulos. Este mecanismo permite desarrollar, integrar y desplegar nuevas funcionalidades, que pueden ser utilizadas y/o mejorados por otros módulos.

También ofrece una interfaz de usuario común para todas las herramientas, la cual está diseñada para trabajar en cualquier sistema operativo.

## 2.2. Estructura de la plataforma Eclipse

Eclipse ha sido diseñado desde el principio para facilitar el desarrollo. La plataforma no ofrece gran funcionalidad por sí sola sino que su valor real se encuentra en el modelo basado en plug-ins que pone a disposición del usuario.

La plataforma Eclipse está estructurada como un conjunto de subsistemas, los cuales son implementados en uno o más plug-ins que se despliegan sobre una pequeña máquina de ejecución. A continuación se muestra una ilustración de la arquitectura de Eclipse y se describen sus componentes principales:

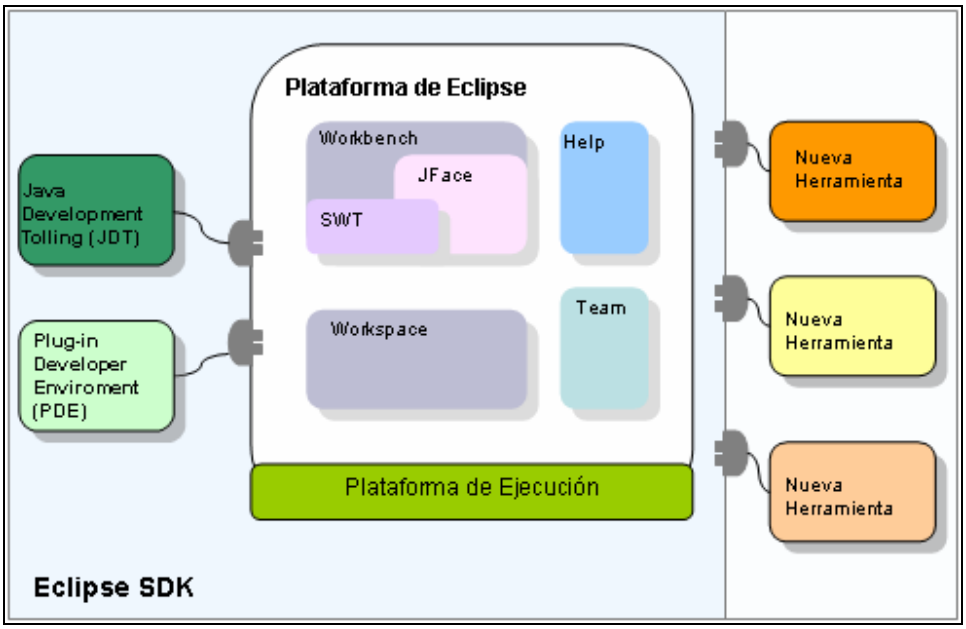


**Ilustración 6: Arquitectura de Eclipse**

A continuación se describen los componentes de la arquitectura de Eclipse:

### *2.2.1. Plataforma de ejecución*

Implementa la máquina de ejecución que se encarga de iniciar la base de la plataforma y de descubrir los plug-ins dinámicamente. Un objetivo muy importante de este componente es mejorar el desempeño de Eclipse, disminuyendo el uso de memoria por aquellos plug-ins que no están siendo usados. De esta manera un plug-in puede ser instalado y agregado al registro pero el mismo no será activado a menos que sea requerido un servicio que provea.

### *2.2.2. Workspace o área de trabajo*

El área de trabajo es una jerarquía de directorios que contiene tanto los archivos de usuario como los proyectos, código fuente, información de estado (preferencias) de los plug-in, entre otros. Las herramientas integradas a Eclipse operan sobre archivos localizados en esta área de trabajo.

#### Proyecto

Un proyecto es una estructura que se corresponde a una carpeta del sistema de archivos indicada por el usuario, en la cual se almacenan diferentes recursos necesarios para el desarrollo de una aplicación en Elipse.

Este componente define el API para la creación y gestión de recursos, como lo son proyectos, archivos y carpetas, que son producidos por herramientas y que son almacenados en el sistema de archivos.

En el entorno de desarrollo Eclipse todo archivo se almacena dentro de un proyecto. Esto quiere decir que todo documento, carpeta, archivo de código fuente (.java) y código compilado (.class) tiene que estar contenido dentro de un proyecto.

Es necesario crear un nuevo proyecto no sólo para desarrollar un nuevo programa, sino para editar también archivos ya existentes (como por ejemplo, un programa ".java" creado anteriormente).

Por defecto Eclipse permite la creación de varios tipos de proyectos[12], entre los cuales se pueden mencionar:

- **Java Project**: Permite crear o modificar programas desarrollados con Java. En este tipo de proyecto almacena información relacionada con el proyecto, código fuente, documentación y otros archivos relacionados.
- **Simple Project**: Son proyectos que almacenan documentos sencillos y otros archivos, los cuales no necesariamente son elementos Java.
- **Plug-in Development Project**: Se usan para agregar nuevos módulos y funciones al entorno Eclipse.

### *2.2.3. Workbench o banco de trabajo*

Implementa el aspecto visual que permite al usuario acceder a las funcionalidades de Eclipse y los recursos asociados a este. Está compuesto de un conjunto de interfaces de usuario: barra de menú, barra de herramientas, perspectivas, vistas, editores. En la siguiente ilustración se pueden apreciar la presencia de estos elementos dentro del banco de trabajo:

[12] Pueden existir otros tipos de proyectos dependiendo de los módulos que se tengan integrados al IDE.

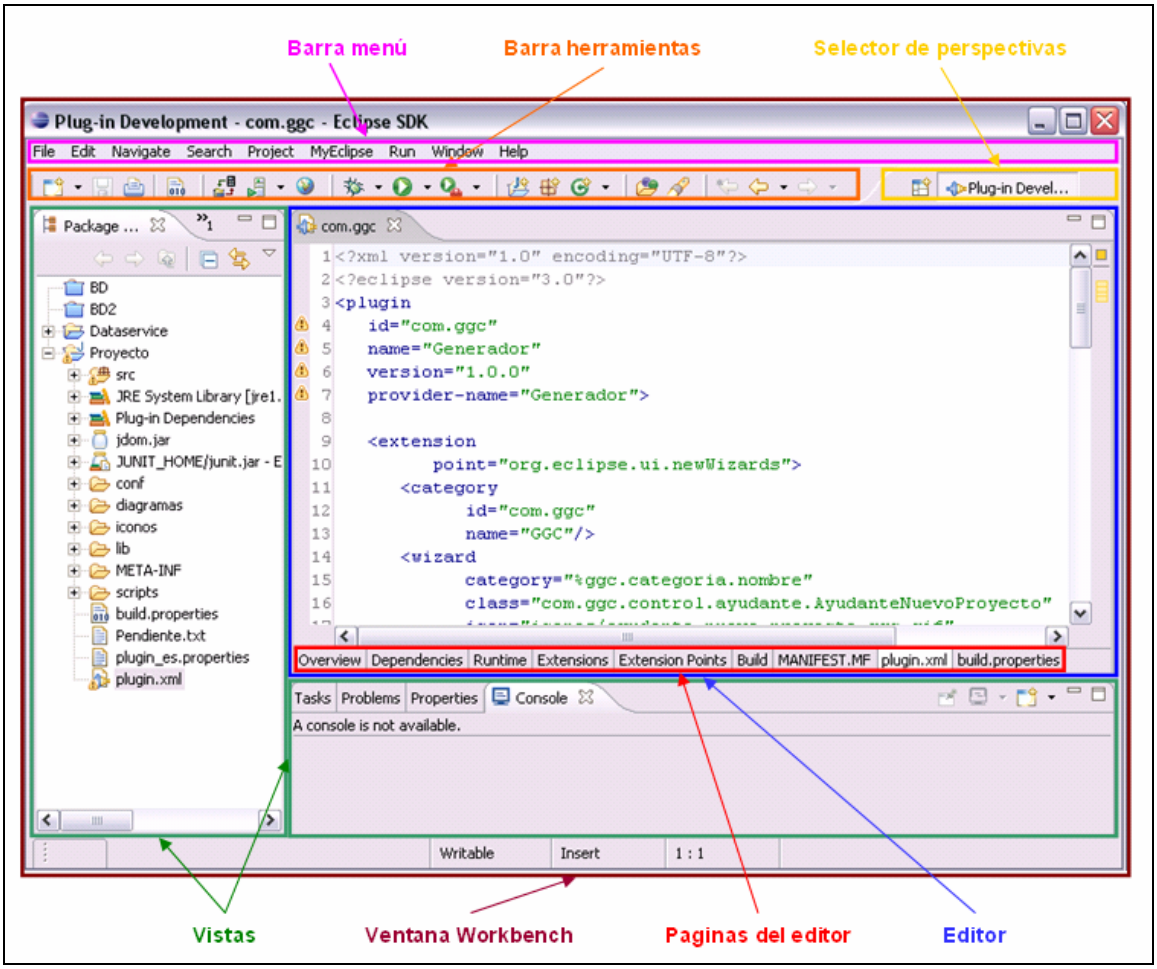


**Ilustración 7: Marco de trabajo de Eclipse**

- **Editores**: Un editor es un mecanismo que forma parte del marco de trabajo (Workbench) que permite modificar un recurso.

- **Vistas**: Proveen información acerca de algún objeto en el workbench con el cual esté trabajando el usuario. Cambian frecuentemente de contenido a medida que el usuario cambia de objeto.

- **Perspectivas**: Las perspectivas guardan información acerca de cuáles son las vistas predeterminadas para una actividad. Por ejemplo: si el desarrollador está depurando un código puede querer ver solo vistas y funcionalidades para esta actividad.

- **Diálogos**: Son utilizados para mostrar información al usuario y para permitir el ingreso de parámetros a la aplicación.

- **Ayudantes**: También llamados Wizards, guían al usuario durante la ejecución de un conjunto de pasos.

- **Páginas de preferencia**: Permiten establecer propiedades específicas de un plug-in para un recurso. Estos valores se mantienen al cerrar Eclipse. La

información obtenida a través de estas páginas son utilizadas para definir el comportamiento de un plug-in.

- **Páginas de propiedades**: Son interfaces a través de las cuales se establece información de un recurso en particular.

- **Contribuciones**: Representan una contribución a un recurso gráfico compartido como son menús y barras de herramientas. Los mismos son gestionados por un manejador de contribuciones. Por ejemplo, una contribución a una barra de herramientas puede ser un botón o un separador.

- **Páginas del editor:** Son vistas de las cuales está compuesta un editor. Estas vistas permiten editar un recurso y son accedidas a través de pestañas.

### *2.2.4. Help o ayuda*

Implementa un servidor de ayuda optimizado y la facilidad de integración de documentos, los cuales sirven al usuario para resolver dudas acerca del funcionamiento de cualquier funcionalidad de los plug-ins.

### *2.2.5. Team o equipo*

Permite a otros plug-ins definir y registrar implementaciones para la programación orientada a un equipo de desarrollo, acceso de repositorios y establecimiento de las versiones de artefactos.

## 2.3. Módulos en Eclipse (Plug-ins)

La plataforma Eclipse está construida en base a plug-ins. Este mecanismo permite desarrollar, integrar y desplegar nuevas funcionalidades.

Un plug-in o módulo es un bloque fundamental de la plataforma Eclipse, el cual es activado cuando se necesita alguna de las funcionalidades que este implementa. La plataforma mantiene un registro de aquellos plug-ins instalados así como de las funcionalidades que proveen. Los módulos permiten agregar nuevas funcionalidades al sistema a través de un modelo de extensión común que será explicado más adelante.

Las herramientas construidas en base a plug-ins pueden estar compuestas de un solo plug-in o de un conjunto de plug-ins comunicados entre sí; esto generalmente varía dependiendo de la complejidad de la herramienta.

Existen por lo general, varias formas de instalar los plug-ins:

- En la mayoría de los casos, sólo hay que descomprimir el zip del plug-in en el directorio en el que se encuentra instalado Eclipse.

- Con un programa de instalación, el cuál integra la nueva herramienta.

- Utilizando la funcionalidad de actualización de la plataforma provista por Eclipse. De esta forma se pueden descargar nuevos plug-ins o versiones nuevas desde Internet.

### *2.3.1. Estructura de un plug-in*

Cada plug-in se encuentra en un subdirectorio de Eclipse llamado *plug-ins* y debe tener un nombre que lo identifique. Este nombre se separa con puntos si tiene más de una palabra y al final se coloca la versión separada del nombre por un subrayado. Por ejemplo, un posible identificador para la primera versión del plug-in de generación de consultas sería *com.ggc_1.0.0*.

Todo plug-in tiene un archivo descriptor META-INF/MANIFEST.MF que puede ser complementado con un archivo plugin.xml. Estos y otros componentes serán ampliados más adelante.

En la siguiente ilustración se muestra un posible ambiente de ejecución de eclipse con seis plug-ins: ggc, jdt, jface, ui, resources, jdtcore.

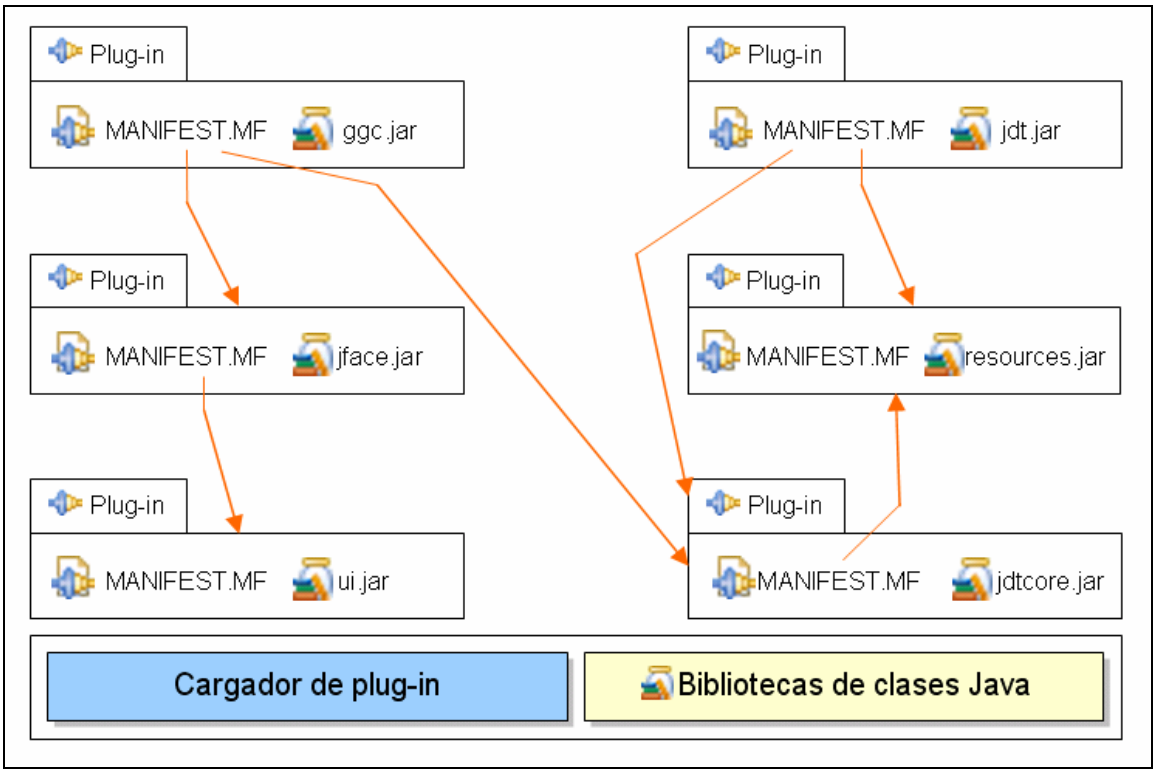


**Ilustración 8: Posible ambiente de ejecución de Eclipse**

En la ilustración anterior, se puede ver la comunicación entre los plug-ins y que cada uno de ellos está compuesto de dos archivos: MANIFEST.MF y un jar[13]. Estos componentes serán descritos más adelante.

[13] JAR: Formato de archivos de Java que permite empaquetar varios recursos en un sólo archivo

Este diseño modular de eclipse permite agregar, mejorar y quitar funcionalidades de acuerdo a lo que el usuario necesite. El generador gráfico de consultas que se quiere construir brindará sus funcionalidades para este entorno de desarrollo y será integrado en eclipse.

## Plug-in.xml:

Cuando se inicia Eclipse, el cargador de plug-ins se encarga de verificar los módulos que se encuentran en el sistema y de verificar la dependencia entre estos.

El archivo plugin.xml es un archivo XML que describe el plug-in, en él se indican las funcionalidades que se ofrecen, dependencias y recursos que se necesitan (tales como imágenes, archivos de configuración, entre otros). A continuación se presenta un ejemplo de este archivo y se describen algunos de sus elementos:

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin
   id="com.ggc"
   name="Generador"
   version="1.0.0"
   provider-name="Generador">
   <extension
         point="org.eclipse.ui.newWizards">
      <category
            id="com.ggc"
            name="GGC"/>
      <wizard
            category="%ggc.categoria.nombre"
            class="com.ggc.control.ayudante.AyudanteNuevoProyecto"
            icon="iconos/ayudante_nuevo_proyecto_ggc.gif"
            id="com.ggc.control.ayudante.AyudanteNuevoProyecto"
            name="%ggc.wizard.crearProyecto.nombre"
            project="true">
         <description>
            Este plug-in permite llevar a cabo la creación de un proyecto de consultas
         </description>
      </wizard>
   </extension>
   <extension
         id="com.ggc.naturalezaGGC"
         name="Proyecto GGC"
         point="org.eclipse.core.resources.natures">
   </extension>
...
</plugin>
```

**Ilustración 9: Ejemplo de un archivo plugin.xml**

**Tabla 2: Elementos de archivo plug-in.xml**

| Elemento | Descripción |
|---|---|
| Plugin | Es el elemento principal del archivo y tiene atributo que permite indicar información acerca del plug-in. |
| Extensión | Permite utilizar funcionalidades de otros plug-ins. |
| extension-point | Define el punto de entrada para que otros plug-in accedan a las funcionalidades del plug-in. |

## MANIFEST.MF:

A partir de la versión 3.1 de eclipse, cada plug-in tiene un archivo manifest.mf, el cual, guarda tanto la información general del plug-in como la información de exportación e importación de clases y paquetes (en tiempo de ejecución). Este archivo es quien permite al cargador de plug-ins conocer las características y dependencias de los plug-in sin necesidad de cargarlos, ahorrando así tiempo de ejecución y espacio en memoria. En la siguiente ilustración se presenta un ejemplo de este archivo:

```
Bundle-Name: %ggc.plugin.nombre
Bundle-SymbolicName: com.ggc; singleton:=true
Bundle-Version: 1.0.0
Bundle-Activator: com.ggc.GGC
Require-Bundle: org.eclipse.ui,
 org.eclipse.core.runtime,
 org.eclipse.core.resources,
 org.eclipse.ui.ide
Bundle-Vendor: hecane
Bundle-ClassPath: lib/jdom.jar,
 lib/ojdbc14.jar,
 lib/postgresql-8.2dev-503.jdbc3.jar
```

**Ilustración 10: Ejemplo de un archivo MANIFEST.MF**

**Tabla 3: Comandos de un archivo MANIFEST.MF**

| Elemento | Descripción |
|---|---|
| Bundle-Name | Nombre del plug-in. |
| Bundle-SymbolicName | Identificador único del plug-in, generalmente usando las convenciones de nombrado de paquetes de Java. |
| Bundle-Version | Tres números separados por períodos que indican la versión del plug-in. Se puede especificar un cualificador opcional que incluya caracteres alfanuméricos, tales como <bundle-version>.beta_1. |
| Bundle-Activator | Clase principal del plug-in, utilizada al momento de inicialización y finalización del plug-in. |
| Require-Bundle | Indica cuales son los plug-in que son requeridos para el funcionamiento del plug-in. |
| Bundle-Vendor | Creador del plug-in. |
| Bundle-ClassPath | Describe cuales son las bibliotecas que utiliza un plug-in para llevar a cabo sus funcionalidades. |

## Recursos estáticos:

Los plug-ins pueden incluir imágenes y otros recursos, los cuales son instalados en el directorio del plug-in junto con el resto de sus componentes (clases y archivos descriptores). Estos recursos son archivos estáticos que se comparten entre las múltiples instancias del IDE.

## Implementación:

Las clases que implementan las funcionalidades del plug-in, son empaquetadas en un archivo jar, el cual generalmente es nombrado usando el último segmento del identificador del plug-in.

Existe una clase especial que provee métodos para acceder a recursos estáticos asociados al plug-in, y permiten acceder e inicializar las preferencias específicas y otra información de estado y configuración. Además, es la primera clase notificada después de que el plug-in se carga, y la última cuando el plug-in es cerrado. Si esta clase no es creada, Eclipse asocia una por defecto.

### *2.3.2. Extensiones*

Las extensiones son el mecanismo para permitir la comunicación entre los módulos, y hacer posible que los módulos compartan sus funcionalidades entre ellos.

Para que un plug-in pueda compartir sus funcionalidades, este debe definir un punto de extensión, a través del cual se puedan acceder desde otros plug-ins para extender la plataforma. Cuando otros plug-ins hacen uso de algún punto de extensión debe definir una extensión en sus archivos descriptores.

Este mecanismo de extensión es la única manera de agregar funcionalidades a la plataforma. De esta forma todas las herramientas provistas con Eclipse SDK no utilizan ningún mecanismo privado para su implementación.

Las extensiones son típicamente escritas en Java utilizando el API de la plataforma. Sin embargo, algunos puntos de extensión tienen asociadas extensiones provistas en forma de ejecutables, componentes ActiveX, o incluso extensiones que han sido desarrolladas mediante lenguajes script. Es importante tener en cuenta que solo un subconjunto del total de las funcionalidades de la plataforma se encuentra disponible para extensiones no realizadas con Java.

A continuación se describen los puntos de extensión necesarios para agregar al sistema las nuevas funcionalidades provistas por el módulo que se desea desarrollar:

#### org.eclipse.ui.newWizards

Los wizards o ayudantes son usados para guiar al usuario a través de un conjunto secuencial de tareas. Los plug-ins desarrollados pueden contribuir con wizards en puntos de extensión predefinidos dentro de la plataforma. Desde el punto de vista estructural, un wizard se encuentra compuesto de varias partes:

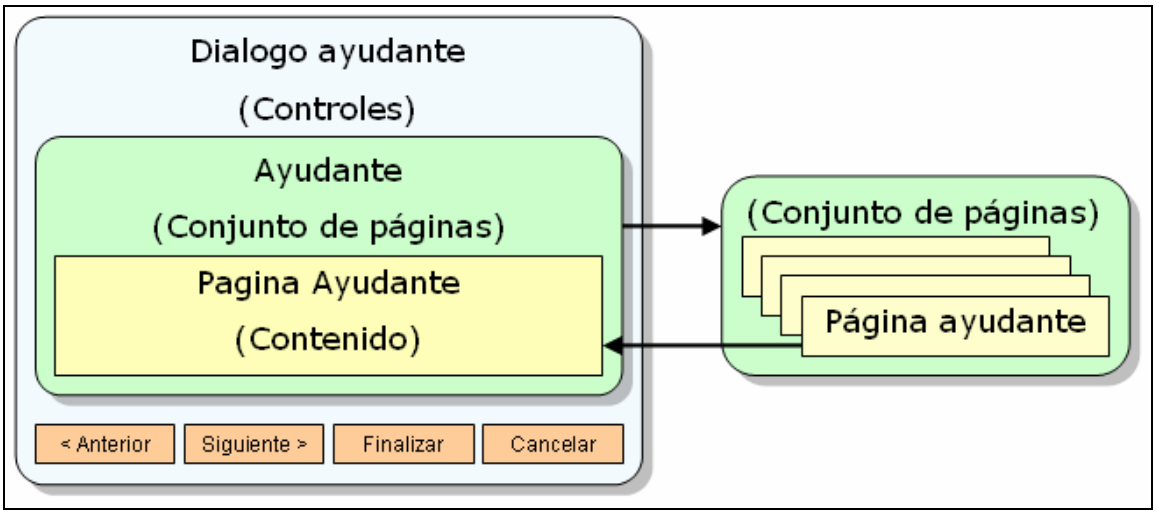


**Ilustración 11: Estructura de un ayudante**

**Tabla 4: Componentes de un ayudante**

| Parte | Descripción |
|---|---|
| Dialogo ayudante (WizardDialog) | Es el contenedor principal. Concretamente se encarga de definir los botones estándar del ayudante, así como administrar el conjunto de páginas que le son provistas. En término de los botones disponibles, gestiona la habilitación y desactivación de los mismos basándose en la información que obtiene desde la página actual y el Ayudante contenido. |
| Ayudante (Wizard) | Controla la apariencia y comportamiento general, como puede ser: el texto que se despliega en la barra de ventana, las imágenes que son mostradas o la disponibilidad o no de un botón de ayuda. |
| Página ayudante (WizardPage) | Define los controles que son utilizados para exhibir el contenido de cada página. Se encarga además de responder a eventos relativos a su contenido y determinar cuando la misma se ha completado. |

### org.eclipse.core.resources.natures

Las naturales son utilizadas para indicar los tipos de un proyecto (Java, simple, de desarrollo de plug-in, entre otros). Por ejemplo, en el caso del generador de consultas se define una nueva naturaleza para la generación de consultas la cual es asignada a cada proyecto creado a través del wizard. Esta naturaleza puede ser asignada para ejecutar acciones sobre los proyectos.

### org.eclipse.ui.editors

Este punto de extensión es usado para agregar nuevos editores al Workbench. En el caso de la nueva herramienta será utilizado para agregar un nuevo editor a través del cual se podrá editar una consulta de recuperación de datos.

Un editor es un componente visual dentro de una página de Workbench, el cual es utilizado para editar un recurso. Para abrir un editor, el usuario invocará la acción “Abrir” sobre un archivo. El Workbench provee soporte para la creación de editores internos, los cuales son integrados dentro del Workbench, y editores externos, los cuales son ejecutados en ventanas separadas.

#### org.eclipse.ui.views

Una vista es una parte del marco de trabajo (Workbench) que permite navegar en una jerarquía de información o desplegar propiedades de un objeto. Esto es posible a través de una instancia única para el marco. Cuando el usuario realiza selecciones u otros cambios en una vista, ellos son inmediatamente reflejados en el marco de trabajo. Las vistas son utilizadas para brindar soporte a un determinado editor. En nuestro caso, el presente punto de extensión es utilizado para proveer al usuario de una representación gráfica y navegable de las consultas gráficas generadas.

#### org.eclipse.core.runtime.preferences

Este punto de extensión es utilizado para permitir al usuario establecer los valores de configuración del plug-in a través de las interfaces de preferencias de Eclipse.

Los valores de una preferencia persisten entre diferentes sesiones del Workspace para permitir al usuario mantener el estado de un plug-in consistente entre sesiones.

El generador de consultas empleará este mecanismo para que el usuario pueda indicar el idioma de la herramienta y numero de filas del resultado de una consulta.

#### org.eclipse.ui.perpectives

Este punto de extensión es definido por Eclipse para permitir agregar una nueva perspectiva al Workbench. Será utilizado para establecer las interfaces que estarán a asociadas al generador de consultas.

### *2.3.3. Plug-ins de Eclipse*

Generalmente, los plug-ins son construidos basándose en uno o más módulos que forman parte de Eclipse. Estos módulos se separan en varios grupos: UI (para interfaces de usuario) y Core (Funcionalidades del nucleo). Algunos son:

- **Core**: Un grupo general de plug-ins sin interfaces gráficas que prestan servicios básicos.
- **SWT**: Biblioteca general de componentes para interfaces de usuario, que utiliza componentes del sistema operativo a través de un API independiente del mismo.
- **JFace**: Biblioteca general de interfaces gráficas adicionales funcionalmente construidas sobre SWT.
- **Workbench Core**: Plug-ins que proveen características específicas del IDE Eclipse, tales como proyectos, naturalezas de proyecto, entre otros.

- **Workbench UI**: Plug-ins que proveen características (gráficas) específicas de Eclipse, tales como editores, vistas, perspectivas, acciones y preferencias.
- **Team**: Grupo de plug-ins que proveen servicios para el control de diferentes tipos de integración de códigos fuentes en el IDE Eclipse.
- **Help**: Plug-ins que proveen documentación para Eclipse como parte del IDE.
- **JDT Core**: Provee herramientas para permitir el desarrollo con Java.
- **JDT UI**: Provee servicios de interfaces de usuario para el desarrollo con Java.
- **GEF**: Biblioteca general de interfaces gráficas para dibujar componentes personalizados; funcionalmente construida sobre SWT y JFace.

## Síntesis

Eclipse es una aplicación que ha sido diseñada para proveer diferentes herramientas que ayuden al desarrollo de software.

Este entorno de desarrollo propone un sistema de plug-ins para la extensión de sus funcionalidades. El elemento esencial para lograr todo esto es el núcleo de Eclipse, el cual es una arquitectura para el descubrimiento dinámico, carga y ejecución de plug-ins. La interfaz de usuario (UI) provee un modelo estándar de navegación, para ello se maneja el concepto Workbench que debe ser usado por cada herramienta nueva que se quiera desarrollar para Eclipse.

Una vez detallados los conceptos base para el desarrollo de la nueva herramienta, se lleva a cabo la especificación de los aspectos relacionados con el desarrollo de la herramienta de generación de consultas. Dicha especificación se realizará en el siguiente capítulo.

# Capítulo III. Desarrollo de la herramienta

En este capítulo se muestran aspectos referentes al desarrollo de la herramienta de generación de consultas. Primero, se toman como base las prácticas, principios y convenciones de XP[14] (eXtreme Programming) y AM[15] (Agile Modeling). Luego se adaptan y mezclan los anteriores para definir el proceso de desarrollo de la herramienta, el cual se llevará a cabo durante todo el capítulo.

## 3.1. Selección de prácticas, principios y valores

Para realizar la construcción de la herramienta se utilizará un proceso de desarrollo que consiste en una adaptación del proceso de desarrollo Programación Extrema y algunos principios, prácticas y valores de Modelación Ágil.

El proceso de desarrollo Programación Extrema es definido en su sitio oficial como una disciplina para el desarrollo de software, la cual utiliza un conjunto de técnicas y principios para permitir la construcción de un software de forma simple.

La Modelación Ágil está basada en prácticas para la modelación y documentación efectiva de sistemas basados en software. Posee un conjunto de valores, principios y prácticas para la modelación de software, que pueden ser aplicados en un proyecto de desarrollo de software de manera efectiva y ligera.

A continuación se listan los valores principios y prácticas de Programación Extrema y Modelación Ágil:

### *3.1.1. Principios XP:*

- El código es revisado continuamente mediante la programación en parejas.
- Las pruebas se realizan permanentemente (pruebas unitarias, pruebas de aceptación.
- Las pruebas de integración se efectúan siempre que se agregue una nueva clase o se modifique una existente (integración continua).
- Se rediseñará todo el tiempo, dejando siempre el código en el estado más simple y sencillo posible.
- Las iteraciones se reducen radicalmente, pudiendo aprovechar la retroalimentación.

[14] http://www.xprogramming.com/
[15] http://www.agilemodeling.com/

### *3.1.2. Prácticas XP:*

- Planificación.
- Versiones pequeñas.
- Diseño simple
- Pruebas.
- Refactorización.
- Programación en parejas.
- Propiedad colectiva.
- Integración continua.
- Cliente en el sitio.
- Estándares de codificación.

### *3.1.3. Valores Modelación Ágil:*

- Comunicación.
- Simplicidad.
- Retroalimentación.
- Coraje.
- Humildad.

### *3.1.4. Principios Modelación Ágil:*

- Modele con un propósito.
- Maximice la inversión de los colaboradores.
- Viaje ligero.
- Modelos múltiples.
- Retroalimentación rápida.
- Asuma la simplicidad.
- Abrazar el cambio.
- Cambio incremental.
- Trabajo de calidad.
- La meta principal es el software.
- La meta secundaria es hacer el siguiente esfuerzo.
- El contenido es más importante que la representación.
- Comunicación abierta y honesta.

### *3.1.5. Prácticas Modelación Ágil:*

- Participación activa de los colaboradores.
- Modelar con otros.
- Aplicar los artefactos correctos.
- Iterar a otro artefacto.
- Pruébelo con código.
- Use las herramientas más simples.
- Modele en pequeños incrementos.

- Información de fuentes simples.
- Propiedad colectiva.
- Cree muchos modelos en paralelo.
- Cree contenido simple.
- Represente los modelos de manera simple.
- Publique los modelos.
- Aplique estándares de modelación.
- Aplique patrones con cuidado.
- Descarte modelos temporales.
- Formalice los modelos de contrato.
- Actualice sólo cuando duela.

En este caso, la definición del proceso de desarrollo se basará en algunos de los valores, principios y prácticas anteriormente mencionadas.

Las tablas siguientes muestran el conjunto de valores, principios y prácticas que se utilizarán durante el proceso de desarrollo de la herramienta.

**Tabla 5: Valores a seguir en el proceso de desarrollo**

| Proceso / Metodología | Valor | Aplicación |
|---|---|---|
| Modelación Ágil | Simplicidad | Antes de comenzar la implementación de un conjunto de funcionalidades, se realizarán los artefactos del modelo necesarios para alcanzar la comprensión antes de codificar. |
| | Coraje | El equipo de desarrollo será capaz de corregir el trabajo hecho si hay cambios o se encuentra algún error. |
| | Humildad | Se respetarán las habilidades de cada uno de los miembros del equipo de desarrollo. La colaboración entre todos permitirá sacar provecho a estas habilidades. |

**Tabla 6: Principios a seguir en el proceso de desarrollo**

| Proceso / Metodología | Principio | Aplicación |
|---|---|---|
| Modelación Ágil | Modelar con un propósito | Sólo se crearán modelos con una finalidad, cuyo nivel de detalle sea correcto para quien lo va a utilizar. En nuestro caso se emplearán los casos de uso para determinar las funcionalidades iniciales del proyecto, y diagramas de clase para definir los componentes principales de cada una de las iteraciones. |
| | Viajar ligero | Se mantendrá una modelación sencilla y compuesta por pocos modelos (diagramas de las clases, diagramas de casos de uso) para agilizar los cambios en el futuro. |
| | Permitir el próximo esfuerzo | Se proveerá toda la documentación y las herramientas necesarias para la construcción de nuevas versiones de la herramienta. Además de la documentación se plantea la generación de la especificación de la herramienta utilizando Javadoc. |
| | Dar más importancia al contenido | Se realizarán los modelos, que aunque no sean muy representativos, resulten útiles para construcción de módulos y disminuyan la creación y mantenimiento del modelo. En los diagramas a realizar solo se detallará lo suficiente para comprender el problema. |

**Tabla 7: Prácticas a seguir en el proceso de desarrollo**

| Proceso / Metodología | Práctica | Aplicación |
|---|---|---|
| Programación Extrema | Diseño simple | El sistema se diseñará lo más simple posible. Con el fin de facilitar los cambios. |
| | Prueba | Se emplearán pruebas manuales y/o automatizadas al culminar la implementación de una tarea, esto permitirá asegurarnos de que las funcionalidades funcionan correctamente. En el caso de las pruebas automatizadas se utilizará el marco de trabajo JUnit[16]. El esqueleto de las pruebas se definirá antes de codificar, aunque se pueden agregar nuevos criterios de prueba al final para verificar el correcto funcionamiento.<br>En cuanto a las pruebas de aceptación serán realizadas de forma sencilla a final de cada iteración. Una vez que se finalicen todas las iteraciones se realizarán pruebas de aceptación completas, que contemplen la realización de diferentes tipos de consultas. |
| | Refactorización | El código se realizará siguiendo una estructura que facilite su modificación en caso de cambios e incorporación de nuevos requerimientos. |
| | Programación en parejas | Aunque el proceso de desarrollo original sostiene que todo el código será realizado en parejas, en nuestro caso, solo se aplicará durante la implementación de funcionalidades claves del negocio. |
| | Propiedad colectiva | El código se mantendrá centralizado de forma que los desarrolladores puedan acceder y modificar el mismo en cualquier momento. Está práctica también es adoptada por la Modelación Ágil. |
| | Integración continua | Al finalizar la implementación de un módulo, se integrará con el resto de los módulos del sistema. |
| | Estándares de codificación | La escritura del código seguirá estándares de codificación en cuanto a: organización, documentación y convenciones de nombrado. |
| Modelación Ágil | Modelar en pequeños incrementos | Para incrementar la agilidad, se aplicará la modelación de un conjunto de funcionalidades a la vez, particularmente las que corresponden a la construcción de un módulo del sistema. |
| | Crear contenido simple | Se modelará sólo lo necesario y sin llegar a un mayor nivel de detalle. |
| | Representar los modelos de manera simple | Se utilizará UML como lenguaje para el modelado, para comprender los puntos claves de las clases que conformarán la herramienta. |
| | Publicar modelos | Los modelos del sistema serán compartidos por todos los desarrolladores. |

## 3.2. Definición del proceso de desarrollo

El proceso de desarrollo de la herramienta comenzará con la definición de las características de la misma. Esto se realizará a través de un conjunto de historias de usuario indicadas por los clientes, con el fin de identificar las funcionalidades y requerimientos de forma global.

Se comenzará por identificar los requerimientos para la nueva herramienta y, tomando en cuenta principios y prácticas seleccionados anteriormente, se definirán las funcionalidades; utilizando para esto, modelos de casos usos[17].

[16] JUnit: Marco de trabajo para realizar pruebas automatizadas de software desarrollado con Java.
[17] La especificicación de los casos de uso iniciales se encuentran en el Anexo C.

De acuerdo a la característica “basado en iteraciones incrementales” del proceso de desarrollo XP, se desarrollará el sistema realizando iteraciones sobre las diversas funcionalidades del mismo.

Cada iteración tendrá asignadas un conjunto de tareas, las cuales serán definidas al comienzo de la misma. Se toman en cuenta las clases involucradas, detalles relevantes de implementación y las pruebas unitarias y de integración.

Siguiendo los principios y prácticas de la modelación ágil, se realizan los diagramas de clases principales asociados a las funcionalidades implementadas en cada iteración.

Durante este capítulo se muestran cada una de las iteraciones realizadas durante el desarrollo del sistema. Para cada iteración contempla lo siguiente:

### *3.2.1. Planificación*

Permite definir el conjunto de tareas principales que se realizarán para llevar a cabo el desarrollo de las funcionalidades de la iteración. Contempla un conjunto de historias de usuario para ser desarrolladas durante la iteración.

### *3.2.2. Diseño*

Definición de las clases principales asociadas a la implementación de las funcionalidades de la iteración. En el diseño se contemplarán los siguientes patrones:

#### Singleton (Instancia única)

Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.

El patrón *singleton* se implementa creando en una clase un método que crea una instancia del objeto sólo si todavía no existe alguna. Para asegurar que la clase no puede ser instanciada nuevamente se regula el alcance del constructor (con atributos como protegido o privado).

#### Proxy

Es un patrón que se utiliza como intermediario para acceder a un objeto, permitiendo controlar al acceso a él.

Proxy es utilizado cuando existe la necesidad de crear objetos que consumen muchos recursos, pero no se quiere instanciarlos a no ser que el cliente los solicite o se cumplan otras condiciones determinadas.

**Factory Method (Método de fabricación)**

Centraliza en una clase constructora la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la casuística para elegir el subtipo que crear. Fabricación de las estructuras del modelo.

**DAO (Data Access Object)**

Consiste en utilizar un objeto de acceso a datos para abstraer y encapsular todos los accesos a la fuente de datos. El DAO maneja la conexión con la fuente de datos para obtener y almacenar datos. Será empleado para realizar cualquier petición sobre la base de datos.

**Command**

Este patrón permite solicitar una operación a un objeto sin conocer realmente el contenido de esta operación, ni el receptor real de la misma. Para ello se encapsula la petición como un objeto, con lo que además se facilita la parametrización de los métodos. Posee los siguientes participantes:

- **AbstractCommand**: Clase que ofrece una interfaz para la ejecución de órdenes. Define los métodos do y undo que se implementarán en cada clase concreta.
- **ConcreteCommand**: Clase que implementa una orden concreta y sus métodos do y undo. Su constructor debe inicializar los parámetros de la orden.
- **Invoker**: Clase que instancia las órdenes, puede a su vez ejecutarlas inmediatamente (llamando a do) o dejar que el CommandManager lo haga.
- **CommandManager**: Responsable de gestionar una colección de objetos orden creadas por el Invoker. Llamará a los métodos doIt y unDoIt. Gestionará su secuenciación y reordenación (sobre la base de prioridades por ejemplo).

### *3.2.3. Codificación*

Se muestran los aspectos más importantes surgidos durante el desarrollo de las funcionalidades de la iteración, y que están relacionados con las tareas realizadas para implementar las funcionalidades.

Durante la realización de esta tarea se tomarán en cuenta diferentes patrones para la construcción de software:

**MVC (Modelo Vista Controlador)**

MVC son las siglas de Model View Control, y es un patrón arquitectónico que divide las funcionalidades en modelo de negocio, vistas y controladores. Estos tres componentes se definen de la siguiente forma:

- **Modelo**: Esta es la representación específica del dominio de la información sobre la cual funciona la aplicación. El modelo es otra forma de llamar a la capa de dominio.

- **Vista**: Este presenta el modelo en un formato adecuado para interactuar, usualmente un elemento de interfaz de usuario.

- **Controlador**: Este responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista.

### *3.2.4. Pruebas*

Definición de un conjunto de verificaciones sobre el código, tanto de forma automática (usando JUnit) como de forma manual. Para cada funcionalidad a probar se toman en cuenta:

- **Ejecución con excepción**: Se refiere a realizar una prueba que verifique cual es el comportamiento de una funcionalidad cuando recibe entradas o parámetros erróneos.

- **Ejecución correcta**: Verifica que la implementación de la funcionalidad es correcta. Se toman parámetros de entrada válidos.

## 3.3. Historias de usuario iniciales

En función de nuestro objetivo, se quiere desarrollar una herramienta para la generación de consultas de recuperación de datos almacenados en una base de datos. Además, se desea que esté integrado en el IDE Eclipse. Las historias de usuario indicadas son las siguientes:

**Tabla 8: Tabla de historias de usuarios**

| Id. | Fecha | Descripción | HU-Madre |
|---|---|---|---|
| HU-1 | 21-09-2006 | La estructura de la herramienta será lo suficientemente extensible, para permitir agregar una nueva base de datos a la lista de bases datos soportadas. | |
| HU-2 | 21-09-2006 | Desarrollar una herramienta integrada en Eclipse para la generación de consultas de recuperación de datos. | |
| HU-3 | 21-09-2006 | Las consultas estarán contenidas dentro de un proyecto que trabajará de forma similar a los proyectos de Eclipse. | |
| HU-4 | 21-09-2006 | La herramienta permitirá definir propiedades de configuración a través de las cuales el usuario podrá indicar, por ejemplo, el idioma de las interfaces. | |
| HU-5 | 21-09-2006 | La comunicación con las bases de datos se llevará a cabo a través de una conexión contenida dentro del proyecto. | |
| HU-6 | 21-09-2006 | Los objetos de la herramienta serán manejados como recursos de eclipse y serán agregados a través de ayudantes (wizards) que guíen al usuario a través del proceso de creación. | |

| | | | |
|---|---|---|---|
| HU-7 | 21-09-2006 | Se debe permitir cambiar los parámetros de conexión con la base de datos. | |
| HU-8 | 21-09-2006 | Se debe construir una interfaz gráfica que permita editar cada una de las cláusulas de la consulta. | |
| HU-9 | 21-09-2006 | El usuario podrá emplear, en la realización de sus consultas, las vistas y tablas existentes en la base de datos. | |
| HU-10 | 21-09-2006 | Construir una interfaz para mostrar resultados de la ejecución de la consulta en la base de datos. | |
| HU-11 | 21-09-2006 | El plug-in tendrá asociadas un conjunto de interfaces que serán abiertas por defecto. | |
| HU-12 | 21-09-2006 | La consulta debe ser generada de forma gráfica, permitiendo la edición de cada una de sus cláusulas. | |
| HU-13 | 21-09-2006 | La cláusula FROM debe ser generada a través de un editor gráfico que permita manipular las tablas y relaciones entre ellas. | HU-12 |
| HU-14 | 21-09-2006 | Los orígenes de datos deberán tener componentes gráficos que permitan seleccionar las columnas como parte del resultado de la consulta. | HU-13 |
| HU-15 | 21-09-2006 | Debe crearse una Tabla que muestre todos los campos (selecciones) que desplegará la consulta al ser ejecutada. | HU-12 |
| HU-16 | 21-09-2006 | Las condiciones o filtros a aplicar a una consulta deben mostrarse en un árbol para poder visualizar la jerarquía de los filtros. | HU-12 |
| HU-17 | 21-09-2006 | Para la vista que mostrará los grupos se necesitará una tabla que muestre los campos a través de los cuales se agruparán los registros y un árbol donde se pueden visualizar las condiciones que se le pueden aplicar a los grupos. | HU-12 |
| HU-18 | 21-09-2006 | Los resultados de una consulta pueden ser ordenados. Esto se realizará por medio de una tabla que contenga los campos por los cuales se ordenarán los resultados. | HU-12 |
| HU-19 | 21-09-2006 | La consulta generada después de que el usuario edite cada una de las secciones de la consulta, debe ser mostrada en un campo de texto sobre el cual el usuario pueda seleccionar para poder copiar y pegar. | HU-12 |
| HU-20 | 21-09-2006 | Crear el ejecutable del plug-in. | |

Para la completitud de estas historias de usuario serán seleccionadas por subconjuntos a medida que avancen las iteraciones. Se tomará en cuenta: la prioridad de las mismas, la prelación y relación entre ellas, y la complejidad en tiempo de desarrollo.

## 3.4. Iteración I

En esta iteración se definen los componentes que proporcionarán la característica extensible a la herramienta, la cual permitirá que en un futuro incremente el número de bases de datos soportadas.

El sistema de generación de consultas, creará las consultas SQL que el usuario indique sobre un sistema de bases de datos. Estos sistemas de bases de datos deben ser configurados para poder describir su sintaxis, palabras reservadas, funciones, operadores, entre otros elementos que estos manejen.

Para la configuración de los diferentes sistemas de bases de datos se crearán archivos XML, a partir de los cuales el generador de consultas obtendrá la información necesaria para generar las consultas SQL para dicho sistema de bases de datos.

### 3.4.1. Planificación

En esta iteración se trabajará sobre una sola historia de usuario:

**Tabla 9: Historias de usuario de la iteración I**

| Id. | Descripción |
|---|---|
| HU-1 | La estructura de la herramienta será lo suficientemente extensible, para permitir agregar una nueva base de datos a la lista de bases datos soportadas. |

**Tabla 10: Tareas de la iteración I**

| Historia de usuario | Tarea |
|---|---|
| HU-1 | 1. Crear los archivos de configuración que contienen la sintaxis de la consulta.<br>2. Crear los archivos con las características de los sistemas de base de datos.<br>3. Configurar las cadenas de conexión con las bases de datos.<br>4. Cargar configuración del sistema.<br>5. Crear módulo para la obtención de los metadatos. |

### 3.4.2. Diseño

A continuación de definen las clases principales de esta iteración:

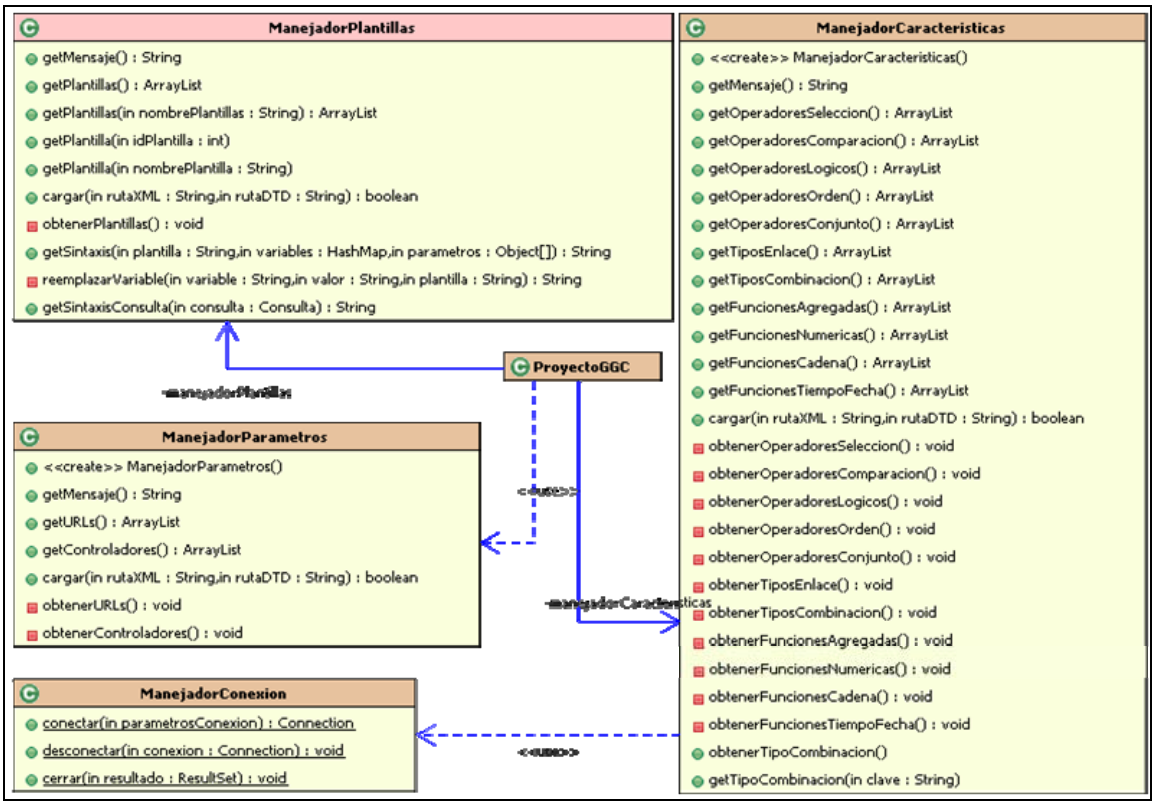


**Ilustración 12: Diagrama de clases de la iteración I**

- **ManejadorConexion**: Es el encargado de administrar las conexiones con la base de datos.
- **ManejadorCaracteristicas**: Utiliza el patrón Proxy para cargar las características (funciones, operadores, tipos de enlaces soportados, entre otras) del sistema manejador de bases de datos.
- **ManejadorPlantillas**: Utiliza el patrón Proxy para cargar las plantillas de construcción de la consulta, correspondiente al sistema manejador de bases de datos. Además, es la clase encargada de generar la sintaxis SQL a partir del objeto que contiene la información de la consulta. Los métodos de generación de sintaxis de cada una de las cláusulas de la consulta, serán implementados durante el desarrollo de la herramienta.
- **ManejadorParametros**: Utiliza el patrón Proxy para cargar los URL's de conexión, que muestra la herramienta para cada uno de los sistemas manejadores de bases de datos configurados.
- **ProyectoGGC**: Clase principal del proyecto de generación de consulta.

### 3.4.3. Implementación

#### Tarea 1: Crear los archivos de configuración que contienen la sintaxis de la consulta

Se agregará a la herramienta un archivo de configuración por cada proveedor de sistema de base de datos soportado. Este archivo permite la extensibilidad de la herramienta, y da al usuario la posibilidad de generar de la misma forma una misma consulta para sistemas de base de datos diferentes.

El archivo de configuración tiene como nombre plantillas.xml, y es un archivo XML describirá la sintaxis SQL para construir la sentencia SELECT. A través de un conjunto de plantillas, que indican cómo construir las diferentes opciones y cláusulas de esta sentencia. Todo literal precedido de un símbolo "$" indica que será sustituido por un operador, un tipo de enlace, una combinación o una función. Y un número entre llaves ({}) será sustituido por un parámetro. En las siguientes ilustraciones se puede ver como se define este archivo.

A continuación se muestra un ejemplo de este archivo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plantillas PUBLIC "-//hecane//DTD PLANTILLAS//ES" '../../DTDs/plantillas.dtd'>

<plantillas id="PostgreSQL" version="8.1">
        <sintaxis>
                <plantilla id="0" nombre="sintaxis_consulta" parametros="0">
                        $seleccion
                        $origen
                        $condicion
                        $grupo
                        $condicion-grupo
                        $orden
                        $limite
                </plantilla>
        </sintaxis>
        <clausulas>
                <plantilla id="23" nombre="origen" requerida="no" parametros="1">FROM
{0}</plantilla>
        </clausulas>
        <operadores>
                <plantilla id="3" nombre="operador-binario" parametros="2">({0} $operador
{1})</plantilla>
        </operadores>
        <combinaciones>
                <plantilla id="9" nombre="operador-combinacion" parametros="3">{0} $tipo-
enlace JOIN {1} ON ({2})</plantilla>
        </combinaciones>
        <funciones>
                <plantilla id="12" nombre="funcion" parametros="2"> ({0}, {1})</plantilla>
        </funciones>
        <otras>
                <plantilla id="29" nombre="expresion-alias" parametros="2">{0} AS
{1}</plantilla>
        </otras>
</plantillas>
```

**Ilustración 13: Ejemplo de un archivo plantillas.xml**

El archivo contiene los siguientes elementos:

**Tabla 11: Descripción de los elementos del archivo plantillas.xml**

| Elemento | Descripción |
|---|---|
| plantillas | Es el elemento raíz de este archivo y contiene todas las plantillas necesarias para crear una sentencia SQL. Contiene dos atributos: "id" (nombre del sistema de base de datos y "version" (versión del mismo). |
| plantilla | Indica la construcción de los elementos presentes en la sentencia SQL. Cada plantilla posee los atributos: identificador único (id), nombre (nombre) y número de operandos utilizados (parámetros). Todas las plantillas utilizadas para describir la construcción de una cláusula tienen un atributo "requerido" que indica si dicha cláusula es necesaria para la creación de la consulta SQL. |
| sintaxis | Este elemento define la sintaxis de una consulta SQL utilizada por el sistema de bases de datos. Contiene una plantilla con el nombre *sintaxis_ consulta*, la cual indica la construcción de la sentencia de consulta. |
| clausulas | Define la construcción de las diferentes cláusulas de una consulta SQL (select, from, entre otras), las cuales se especifican en *sintaxis_consutla*. Ejemplo: si en la plantilla *sintaxis_consulta* se definió una parte de consulta llamada *condicion* ($*condicion*) entonces dentro de *clausulas* debe existir una plantilla cuyo valor de su atributo nombre sea *condicion*. |
| operadores | Contiene las diferentes plantillas para definir el uso de los diferentes operadores provistos por el sistema de base de datos. |
| combinaciones | Este elemento define la forma de construir los diferentes enlaces soportados por el sistema de base de datos. |
| funciones | Contiene las plantillas que definen la construcción de las diferentes funciones provistas por el sistema. |
| Otras | Define otras plantillas que pudiera utilizar el sistema de base de datos para la generación de una consulta y que son comunes en las diferentes cláusulas o partes de la misma. |

Si el manejador de base de datos que se quiere configurar maneja alias, se debe definir una plantilla con el nombre *expresión-alias*, tal como se muestra en el ejemplo.

## Tarea 2: Crear los archivos con las características de los sistemas de bases de datos

La finalidad de esta tarea es crear el archivo que contiene las funciones, operadores y demás elementos válidos dentro de una consulta para cada base de datos.

Este archivo contendrá los operadores, las combinaciones de los diferentes enlaces y las funciones del sistema empleando etiquetas XML. A continuación se muestra un ejemplo de este archivo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE caracteristicas-manejador PUBLIC "-//hecane//DTD CARACTERISTICAS-MANEJADOR//ES"
'../../DTDs/caracteristicas.dtd'>

<caracteristicas-manejador id="PostgreSQL" version="8.1">
        <operadores-seleccion>
                <operador>
                <nombre>Todos</nombre>
```

```
                        <nombre-bd>ALL</nombre-bd>
                        <descripcion>Muestra todos los registros</descripcion>
                </operador>
        </operadores-seleccion>
        <operadores-comparacion>
        ...
        </operadores-comparacion>
        <operadores-logicos>
                ...
        </operadores-logicos>
        <operadores-matematicos>
                ...
        </operadores-matematicos>
        <operadores-orden>
                ...
        </operadores-orden>
        <operadores-conjunto>
                ...
        </operadores-conjunto>
        <tipos-enlace>
                <tipo-enlace>
                        <conector>Ninguno</conector>
                        <nombre>Normal</nombre>
                        <nombre-bd>INNER</nombre-bd>
                        <descripcion>Une dos tablas a partir de los campos
enlazadores indicados</descripcion>
                </tipo-enlace>
        </tipos-enlace>
        <combinaciones>
                <combinacion plantilla="7">
                        <nombre>Cruzada</nombre>
                        <descripcion>Referencia cruzada</descripcion>
                </combinacion>
        </combinaciones>
        <funciones-agregadas>
                <funcion plantilla="10">
                        <nombre>Promedio</nombre>
                        <nombre-bd>AVG</nombre-bd>
                        <parametro>
                                <nombre>x</nombre>
                                <tipo>num&eacute;rico</tipo>
                        </parametro>
                        <descripcion>Media aritm&eacute;tica</descripcion>
                </funcion>
        </funciones-agregadas>
        <funciones-numericas>
                ...
        </funciones-numericas>
        <funciones-cadena>
                ...
        </funciones-cadena>
        <funciones-tiempo-fecha>
...
        </funciones-tiempo-fecha>
</caracteristicas-manejador>
```

**Ilustración 14: Ejemplo de un archivo caracteristicas.xml**

Descripción de los elementos del archivo:

**Tabla 12: Descripción de los elementos de caracteristicas.xml**

| Elemento | Descripción |
|---|---|
| caracteristicas-manejador | Este es el elemento raíz del archivo y contiene todas las características del sistema de base de datos. Este elemento contiene 2 atributos: "id" (nombre del sistema de base de datos y "version" (versión del mismo). |
| operador | Representa un operador soportado por el sistema de base de datos. Este puede ser: de selección (utilizado en la cláusula "SELECT"), de comparación (utilizado en expresiones condicionales), lógico (utilizado en las diferentes expresiones), de orden (utilizado en la cláusula "ORDER BY"), o de conjunto (utilizado para operar con conjuntos resultantes). Este elemento contiene un elemento hijo "nombre" que debe ser distinto para cada operador. |
| tipo-enlace | Define un tipo de enlace (JOIN) que soporta el sistema de bases de datos. Está contenido dentro de un elemento <tipos-enlace>. El atributo <conector> es utilizado para indicar en qué dirección se dibujará el enlace la aplicación; su dominio es: Ninguno, Derecho, Izquierdo, Ambos. El tipo de enlace con nombre Normal, es utilizado por defecto, en caso de no estar definido se toma el primero en ocurrencia. |
| combinacion | Especifica una opción, soportada por el sistema de base de datos, para especificar los enlaces entre tablas. Está contenido dentro de un elemento <combinaciones>. |
| funcion | Representa una función soportada por el sistema de bases de datos. Esta puede ser: agregada, numérica, de cadena o de tiempo-fecha. Tiene un atributo plantilla que indica el identificador de la plantilla de construcción de la función; esta última debe existir en el archivo *plantillas.xml*. Está contenido dentro de un elemento <funciones-agregadas>, <funciones-numericas>, <funciones-cadena> o <funciones-tiempo-fecha>, respectivamente. |
| nombre | Se encuentra dentro de un elemento padre que representa un característica del sistema de base de datos, e indica el nombre que esta característica poseerá dentro de la aplicación. |
| nombre-bd | Tiene como elemento padre una característica de la base de datos, e indica el nombre de una dicha característica en la base de datos. |
| descripcion | Establece una breve descripción para una característica de la base de datos. |
| parametro | Define el nombre y tipo de un parámetro recibido por una función del sistema de base de datos. El elemento <nombre> específica el nombre del parámetro en la aplicación y el tipo se refiere al tipo de datos asociado (fecha, cadena, número). |

## Tarea 3: Configurar las cadenas de conexión con las bases de datos

Esta tarea consiste en crear los archivos de configuración que indicarán a la herramienta como se formará la cadena de conexión con la base de datos seleccionada por el usuario. Se emplea un patrón proxy en la carga de este archivo, para que el proceso no se repita en futuras consultas de esta información.

Para realizar esta configuración, se debe crear un archivo XML, llamado parámetros.xml, para cada proveedor de sistemas de base de datos soportado por la herramienta. Este archivo está formado por un conjunto de URLs[18] y clases de controlados de bases de datos, los cuales definen la forma de construcción de los posibles URLs de conexión con la base de datos y las clases de conexión, respectivamente. El archivo será similar a la siguiente figura:

[18] URL son las siglas de localizador de recursos universal.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE parametros PUBLIC "-//hecane//DTD URLS//ES" '../../DTDs/parametros.dtd'>

<parametros id="PostgreSQL" version="8.1">
        <urls>
                <url>jdbc:postgresql://&lt;m&aacute;quina&gt;:&lt;puerto5432&gt;/&lt;base-
datos&gt;</url>
        </urls>
        <controladores>
                <controlador>org.postgresql.Driver</controlador>
        </controladores>
</parámetros>
```

**Ilustración 15: Ejemplo de un archivo parametros.xml**

El archivo contiene los siguientes elementos:

**Tabla 13: Descripción de los elementos del archivo parametros.xml**

| Elemento | Descripción |
|---|---|
| parametros | Es el elemento raíz de este archivo y contiene los parámetros que se precargan en el sistema al crear una conexión con la base de datos. Contiene dos atributos: “id”, nombre del sistema de base de datos y “version” (versión del mismo). |
| urls | Elemento que contiene los URLs posibles para establecer conexiones con las base de datos. |
| url | Contiene un esqueleto de URL de conexión con la base de datos asociada al atributo *id* del elemento *parametros*. |
| controladores | Elemento que contiene los nombres de las clases de los controladores de bases de datos que se encuentran dentro de la aplicación. |
| controlador | Guarda el nombre de la clase para un controlador de la base de datos indicada en el atributo *id* del elemento *parámetros*. El controlador asociado a esta clase debe estar cargado como biblioteca del plug-in en las rutas de cada sistema de base de datos. |

## Tarea 4: Cargar configuración del sistema.

Esta tarea lleva a cabo la obtención de la información de configuración de la base de datos, una vez que el proyecto es creado. Se emplea un patrón proxy, para que el proceso no se repita en futuras consultas de esta información.

La configuración se obtiene a partir de los archivos XML previamente definidos. Para cada uno de los archivos (*plantillas.xml*, *caracteristicas.xml* y *parametros.xml*) se crea una clase que extrae los datos y los guarda en estructuras de datos. Las clases se implementan haciendo uso de la biblioteca de JDom.

Se puede ver un ejemplo de la clase correspondiente al archivo *plantillas.xml* en la siguiente ilustración:

```
…
public class ManejadorPlantillas
{
   private SAXBuilder constructor;     //Construye la estructura usando un parser SAX.
   private Document documento;         //Crea la estructura de arbol del documento.
   private ArrayList plantillas;      //Lista de urls de plantillas.
   private String mensaje;             //Mensaje de resultado al realizar una operación.
                                       //Cuando está en blanco quiere decir que la
operación fue exitosa.
```

```
   public ManejadorPlantillas(){...}

   public String getMensaje(){ return mensaje;}

   public ArrayList getPlantillas(){...}

   public boolean cargar (String rutaXML, String rutaDTD)
   {
        constructor = new SAXBuilder(false);
                //usar el parser Xerces y no queremos
        constructor.setEntityResolver(new ManejadorDTD(rutaDTD));

        try
        {
                constructor.setValidation(true);
                        //Habilita la validación del documento.
                documento = constructor.build(rutaXML);
                        //Construye el documento a partir del archivo XML.
                mensaje = "";
                return true;
        }
        catch (JDOMException e) {...}
        catch (IOException e) {...}
        catch (Exception e) {...}
   }
   private void obtenerPlantillas ()
   {
        Element raiz;                   //Elemento principal del documento.
        Element elemento;               //Elemento del documento.
        Element elemento2;
        List elementos;                 //Lista de elementos urls.
        List elementos2;
        Iterator iterador;
        Iterator iterador2;
        Plantilla plantilla;            //Guarda  información  acerca  de  una  plantilla  de
construcción.

        try
        {
                raiz = documento.getRootElement();     //Obtiene el elemento raíz.
                elementos = raiz.getChildren();
                                //Obtiene os diferentes tipos de plantillas de construcción.
                                        //Obtiene las plantillas de sintaxis de consulta.

                iterador = elementos.iterator();


                plantillas = new ArrayList();

                while (iterador.hasNext())
                {
                        elemento = (Element)iterador.next();
                        elementos2 = elemento.getChildren("plantilla");
                //Obtiene un grupo de plantillas.
                        iterador2 = elementos2.iterator();

                        while (iterador2.hasNext())
                        {
                                elemento2 = (Element)iterador2.next();
                //Obtiene la próxima plantilla.

                                plantilla = new Plantilla();
                        //Incializa el objeto.
                                plantilla.setId(Integer

   .parseInt(elemento2.getAttributeValue("id")));            //Construye   el   objeto   a
partir del elemento.
                                plantilla.setNombre(elemento2.getAttributeValue("nombre"));
                                plantilla.setParametros(Integer.parseInt(elemento2
                                                .getAttributeValue("parametros")));
```

```
                                plantilla.setConstruccion(elemento2.getText());

                                plantillas.add(plantilla);
                                //Guarda la plantilla en la lista.
                        }
                }

                mensaje = "";
        }
        catch (Exception e) {...}
    }
}
```

**Ilustración 16: Fragmento de la clase Manejador Plantillas**

## Tarea 5: Crear módulo para la obtención de los metadatos.

Un proyecto de generación de consultas está asociado a una base de datos- Debido a esto tiene sus propias características es necesario obtener información acerca del diseño (tales como funciones y palabras reservadas), estructura (tablas, vistas, índices, entre otros) y capacidades (tales como el manejo de group by y enlaces), lo cual permitirá la construcción, generación y ejecución de las consultas realizadas.

Esta información será obtenida a través de JDBC 3.0[19], haciendo uso de la clase java.sql.DatabaseMetadata. Se emplea un patrón proxy, para que el proceso no se repita en futuras consultas de esta información.

Las funcionalidades de esta tarea son implementadas en la clase ManejadorCaracteristicas.java. Cada proyecto de generación de consultas tendrá asociado un objeto de este tipo:

```
public class ManejadorCaracteristicas
{
   /** Construye la estructura usando un parser SAX.*/
   private SAXBuilder constructor;

   /** Crea la estructura de arbol del documento.*/
   private Document documento;

   /** Mensaje de resultado al realizar una operación.*/
   private String mensaje;


....

   public ArrayList getOperadoresSeleccion()
   {
        if (operadoresSeleccion == null) obtenerOperadoresSeleccion();

        return operadoresSeleccion;
   }

   public ArrayList getOperadoresComparacion() {...}

   public ArrayList getOperadoresLogicos()  {...}

   public ArrayList getOperadoresOrden()  {...}

   public ArrayList getOperadoresConjunto()  {...}
```

[19] API de conectividad con bases datos en Java.

```
public ArrayList getTiposEnlace()  {...}

public ArrayList getTiposCombinacion()  {...}

public ArrayList getFuncionesAgregadas()  {...}

public ArrayList getFuncionesNumericas()  {...}

public ArrayList getFuncionesCadena()  {...}

public ArrayList getFuncionesTiempoFecha()  {...}

public boolean cargar (String rutaXML, String rutaDTD)
{
     constructor = new SAXBuilder(false);
     constructor.setEntityResolver(new ManejadorDTD(rutaDTD));
     try
     {
             constructor.setValidation(true);
                     //Habilita la validación del documento.
             documento = constructor.build(rutaXML);
                     //Construye el documento a partir del archivo XML.
             mensaje = "";
                                     //Si no ocurrio error el mensaje es blanco.
             return true;
     }
     catch (JDOMException e) {...}
     catch (IOException e) {...}

     catch (Exception e) {...}
}

private void obtenerOperadoresSeleccion()
{
     Element raiz;
                                     //Elemento principal del documento.
     Element elemento;
                                     //Elemento del documento.
     List elementos;
                                     //Lista de elementos.
     Iterator iterador;

     OperadorBD operador;

     try
     {
             raiz = documento.getRootElement();
                     //Obtiene el elemento raíz.
             elemento = raiz.getChild("operadores-seleccion");
     //Obtiene el grupo de operadores de selección.
             elementos = elemento.getChildren("operador");
     //Obtiene los operadores de selección.

             iterador = elementos.iterator();

             operadoresSeleccion = new ArrayList();

             while (iterador.hasNext())
             {
                     elemento = (Element)iterador.next();
             //Obtiene el próximo operador.

                     operador = new OperadorBD ();
                             //Inicializa el objeto.
                     operador.setNombre(elemento.getChildText("nombre"));
                     operador.setNombreBD(elemento.getChildText("nombre-bd"));
                     operador.setTipo("seleccion");
                     operador.setDescripcion(elemento.getChildText("descripcion"));

                     operadoresSeleccion.add(operador);
```

```
                        //Guarda el operador en la lista.
                }

                mensaje = "";
        }
        catch (Exception e) {...}
    }

    private void obtenerOperadoresComparacion() {...}
    private void obtenerOperadoresLogicos() {...}
    private void obtenerOperadoresOrden() {...}
    private void obtenerOperadoresConjunto() {...}
    private void obtenerTiposEnlace() {...}
    private void obtenerTiposCombinacion() {...}
    private void obtenerFuncionesAgregadas() {...}
    private void obtenerFuncionesNumericas() {...}
    private void obtenerFuncionesCadena() {...}
    private void obtenerFuncionesTiempoFecha() {...}

    public TipoCombinacionBD obtenerTipoCombinacion(TipoEnlaceBD tipoEnlace)
    {
        TipoCombinacionBD tipoCombinacion;
        String conector;

        if (tipoEnlace != null && tipoEnlace.getConector() != null)
        {
                conector = tipoEnlace.getConector();
                for (int i = 0; i < getTiposCombinacion().size(); i++) {
                        tipoCombinacion = (TipoCombinacionBD)getTiposCombinacion().get(i);
                        if (conector.equalsIgnoreCase(TipoEnlaceBD.NORMAL) &&

    tipoCombinacion.getNombre().equals(TipoCombinacionBD.NORMAL))
                                        return tipoCombinacion;
                        else
                                if (!conector.equalsIgnoreCase(TipoEnlaceBD.NORMAL) &&

    tipoCombinacion.getNombre().equals(TipoCombinacionBD.EXTERNA))
                                        return tipoCombinacion;
                }
        }

        return tipoCombinacion = (TipoCombinacionBD)getTiposCombinacion().get(0);
    }
}
```

**Ilustración 17: Fragmento de la clase ManejadorCarcteristicas**

### 3.4.4. Pruebas

Para realizar las pruebas de forma automatizada se implementaron una Suite de pruebas JUnit que contemplan un conjunto de casos de pruebas para las funcionalidades de la iteración.

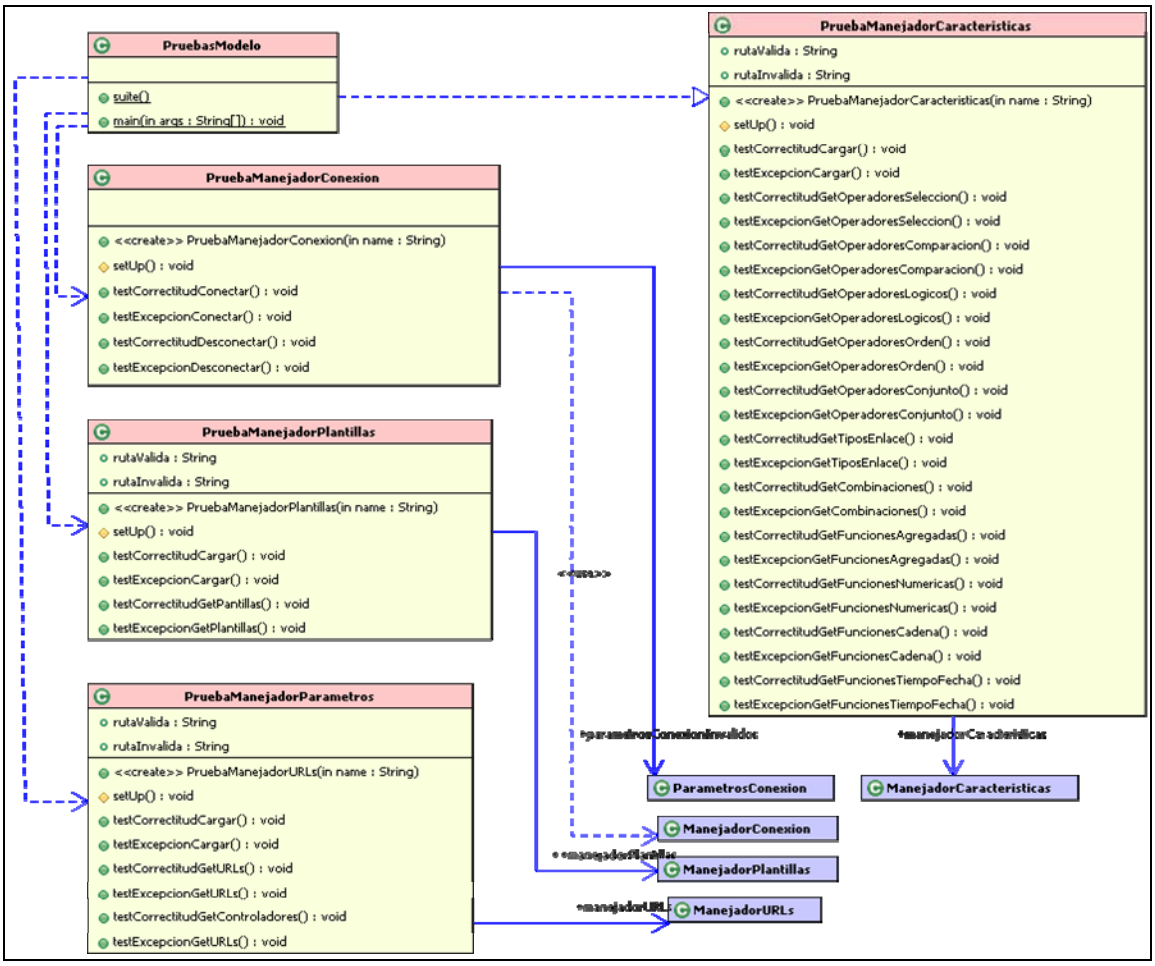


**Ilustración 18: Diagrama de clases de las pruebas automáticas de la iteración**

**Tabla 14: Pruebas de la iteración I**

| Nombre de la prueba | Descripción |
|---|---|
| PruebaManejadorCaracteristicas | Permite verificar el funcionamiento del Manejador de características. |
| PruebaManejadorConexion | Permite verificar el funcionamiento del Manejador de conexiones. |
| PruebaManejadorDiccionario | Permite verificar el funcionamiento del Manejador del diccionario de la base de datos. |
| PruebaManejadorParametros | Permite verificar el funcionamiento del Manejador de parámetros. |
| PruebaManejadorPlantillas | Permite verificar el funcionamiento de la clase ManejadorPlantillas. |

En las siguientes ilustraciones se pueden ver un ejemplo de la suite de pruebas y una de las pruebas automatizadas para esta iteración:

```
…
public class PruebasModelo
{
   public static Test suite()
   {
        TestSuite suite = new TestSuite("Pruebas del modelo");
        suite.addTestSuite(PruebaManejadorConexion.class);
        suite.addTestSuite(PruebaManejadorCaracteristicas.class);
        suite.addTestSuite(PruebaManejadorDiccionario.class);
```

```
        suite.addTestSuite(PruebaManejadorPlantillas.class);
        suite.addTestSuite(PruebaManejadorParametros.class);

        return suite;
    }

    public static void main(String[] args)
    {
        junit.swingui.TestRunner.run(PruebasModelo.class);
    }
}
```

**Ilustración 19: Clase principal de las pruebas automatizadas**

```
…
public class PruebaManejadorConexion extends TestCase
{
    ParametrosConexion parametrosConexionValidos; //Contiene parámetros válidos de conexión.
    ParametrosConexion parametrosConexionInvalidos; //Contiene parámetros de conexión
inválidos.
    ManejadorConexion manejadorConexion;       //Establece y libera conexiones con bases de
datos. Implementa las funcionalidades que se quieren probar.

    public PruebaManejadorConexion(String name) {...}

    protected void setUp() throws {...}

    public void testCorrectitudConectar()
    {
        parametrosConexionInvalidos.setControlador("");
                //Verifica exepcion de controlador.
        assertNull("Error: El método conectar no capturó la excepción Controlador.",
                        ManejadorConexion.conectar(parametrosConexionInvalidos));


        parametrosConexionInvalidos.setControlador("org.postgresql.Driver");
        parametrosConexionInvalidos.setUrl("");
                        //Verifica exepcion SQL.
        assertNull("Error: El método conectar no capturó la excepción SQL.",
                        ManejadorConexion.conectar(parametrosConexionInvalidos));


        assertNull("Error: El método conectar no capturó la excepción Nula.",
                        ManejadorConexion.conectar(null));
                        //Verifica que se captura la excepción.
    }

    public void testExcepcionConectar(){...}

    public void testCorrectitudDesconectar(){...}

    public void testExcepcionDesconectar(){...}
}
```

**Ilustración 20: Clase para el caso de prueba del ManejadorConexion**

## *Síntesis*

La realización de las tareas de esta iteración se llevó a cabo con éxito, resultando así en la creación de los archivos de configuración de la herramienta para los siguientes sistemas de bases de datos: MySQL, Postgres y Oracle.

Los archivos de configuración anteriormente manipulados, hacen que la herramienta sea extensible, lo cual permitirá al usuario de la misma poder incorporar un nuevo sistema de bases de datos a parte de los mencionados (MySQL, Postgres y Oracle).

El producto de la iteración fue mostrado y aprobado por el usuario completamente, por lo cual se procede a la creación de la herramienta dentro del entorno de desarrollo integrado de Elipse.

## 3.5. Iteración II

Durante esta iteración se llevarán a cabo un conjunto de tareas que permiten la integración de la herramienta en el entorno de desarrollo de Eclipse, creación de objetos base para la consulta, conexión, wizards, entre otros.

Se utilizarán las utilidades de eclipse para la creación de un nuevo plug-in, lo cual permitirá integrar la herramienta como parte de su entorno. Luego se crearán los wizards que guiarán al usuario en la creación de los proyectos de la herramienta y sus recursos. Finalmente se construirán las interfaces para permitir al usuario llevar a cabo la configuración de la herramienta y la elaboración de consultas.

### 3.5.1. Planificación

Las historias de usuario que se contemplarán durante esta iteración se listan a continuación:

**Tabla 15: Historias de usuario de la iteración II**

| Id. | Descripción |
|---|---|
| HU-2 | Desarrollar una herramienta integrada en Eclipse para la generación de consultas de recuperación de datos. |
| HU-3 | Las consultas estarán contenidas dentro de un proyecto que trabajará de forma similar a los proyectos de Eclipse. |
| HU-4 | La herramienta permitirá definir propiedades de configuración a través de las cuales el usuario podrá indicar, por ejemplo, el idioma de las interfaces. |
| HU-5 | La comunicación con las bases de datos se llevará a cabo a través de una conexión contenida dentro del proyecto. |
| HU-6 | Los objetos de la herramienta serán manejados como recursos de eclipse y serán agregados a través de ayudantes (wizards) que guíen al usuario a través del proceso de creación. |
| HU-7 | Se debe permitir cambiar los parámetros de conexión con la base de datos. |
| HU-8 | Se debe construir una interfaz gráfica que permita editar cada una de las cláusulas de la consulta. |
| HU-9 | Mostrar al usuario las tablas y vistas correspondientes a la base de datos indicada. |
| HU-10 | Construir una interfaz para mostrar resultados de la ejecución de la consulta en la base de datos. |
| HU-11 | El plug-in tendrá asociadas un conjunto de interfaces que serán abiertas por defecto. |

La forma en la que se definirán las pruebas para las tareas de esta iteración varían dependiendo de la tarea. Esto se debe a que existen diversos tipos de tareas, de interfaces, de creación de objetos, dependientes entre sí,

**Tabla 16: Tareas de la iteración II**

| Historia de usuario | Tarea |
|---|---|
| HU-2 | 1. Crear proyecto de Eclipse para creación de plug-in. |
| HU-3 | 2. Definir la naturaleza de los proyectos de generación de consulta. |
| HU-4 | 3. Definir internacionalización de mensajes.<br>4. Crear vistas para establecer preferencias. |

| | |
|---|---|
| HU-5 | 5. Crear el recurso que guarda los datos de la conexión.<br>6. Implementar modulo de conexión con la base de datos. |
| HU-6 | 7. Construir ayudante (wizard) para la creación del proyecto.<br>8. Construir ayudante para la creación de la consulta. |
| HU-7 | 9. Crear editor para los parámetros de conexión. |
| HU-8 | 10. Definir editor para la consulta. |
| HU-9 | 11. Implementar módulo para consultar diccionario de la base de datos.<br>12. Crear vista con las tablas y vistas de la base de datos. |
| HU-10 | 13. Crear modulo de ejecución de la consulta. |
| HU-11 | 14. Crear perspectiva del plug-in. |

### *3.5.2. Diseño*

Una vez que ya fueron definidas las historias de usuario, y las tareas que permiten completar dichas historias de usuario, se procede a diseño inicial de los principales componentes de la iteración actual:

#### Manejo de proyectos:

El plug-in incorporará a Eclipse un nuevo tipo de proyecto, en cual se diferencia por soportar las funcionalidades de Generación de consulta. A continuación se definen las dos clases principales:

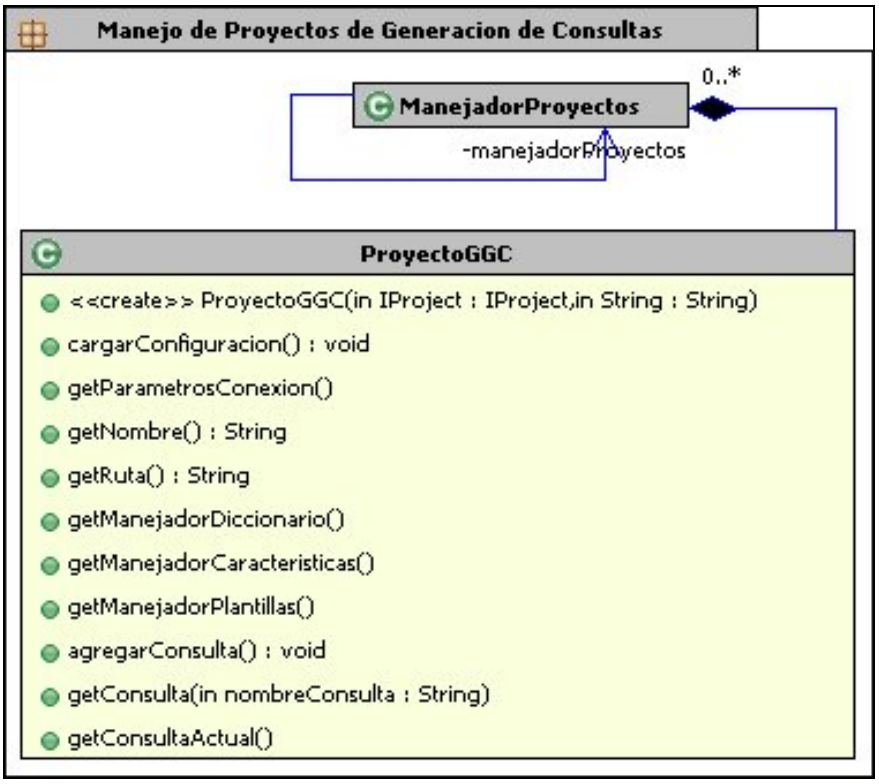


**Ilustración 21: Clases principales para el manejo de proyectos**

- **ProyectoGGC**: Es la estructura que engloba todas las propiedades del proyecto de generación de consultas y las características para representar un proyecto en el Workspace de Eclipse. Como puede verse en la siguiente figura, cada proyecto tendrá asociado el diccionario de datos, las características y las plantillas; las cuales deben corresponderse con el Sistema de base de datos escogido por el usuario durante la creación del proyecto.

- **ManejadorProyectos**: Es el encargado de administrar los proyectos de generación de consulta; su función es llevar a cabo la lógica que implica la creación, eliminación y modificación de un proyecto y sus recursos contenidos. Implementa el patrón Singleton, para garantizar una única instancia por Proyecto.

## Manejo de preferencias:

El usuario tendrá la posibilidad de modificar ciertas propiedades del plug-in, como lo son: idioma, número de filas resultantes y generación automática de la consulta. Las clases que implementan esta lógica se describen a continuación:


**Ilustración 22: Clases principales del módulo de preferencias**

- **PaginaPrincipal**: Es la clase que representa la primera página de preferencia en el plug-in, la cual se accede por el menú Windows→Preferences…, y muestra una imagen de la herramienta.

- **PaginaPreferencias:** Contiene los componentes visuales de la página de preferencias, a través de la cual el usuario puede establecer las propiedades o características.

- **InicializadorPreferencias y ConstantesPreferencias**: Contiene los posibles valores que pueden tomar las propiedades editables del plug-in, y permiten cargar las opciones por defecto.

## Edición de la consulta:

Durante esta iteración se creará la estructura inicial del editor de consultas, definiendo para ello las siguientes clases: EditorMultipagina (contiene los editores para cada una de las cláusulas de la consulta), y Editor<cláusula>.

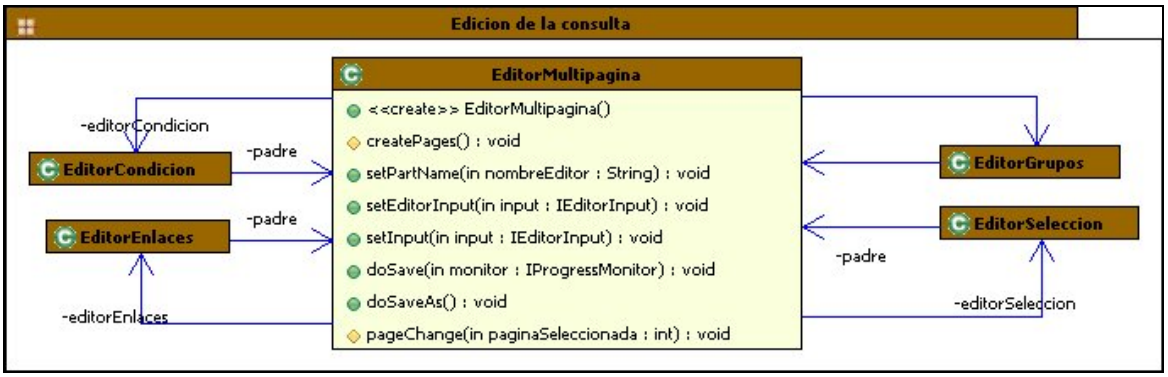


**Ilustración 23: Esqueleto del editor multipáginas**

## Generación de sintaxis SQL de la cláusula selección:


**Ilustración 24: Generación de la sintáxis SQL**

Para esta iteración el manejador de plantillas debe implementar los métodos que generan la sintaxis SQL correspondiente a la cláusula de orígenes de datos de la consulta.

## Ayudantes:

Se crearán dos ayudantes de creación de objetos, uno para la creación de la consulta, y el otro para la creación del proyecto. Para ello es necesario definir las clases que los representen y a cada una de sus páginas:

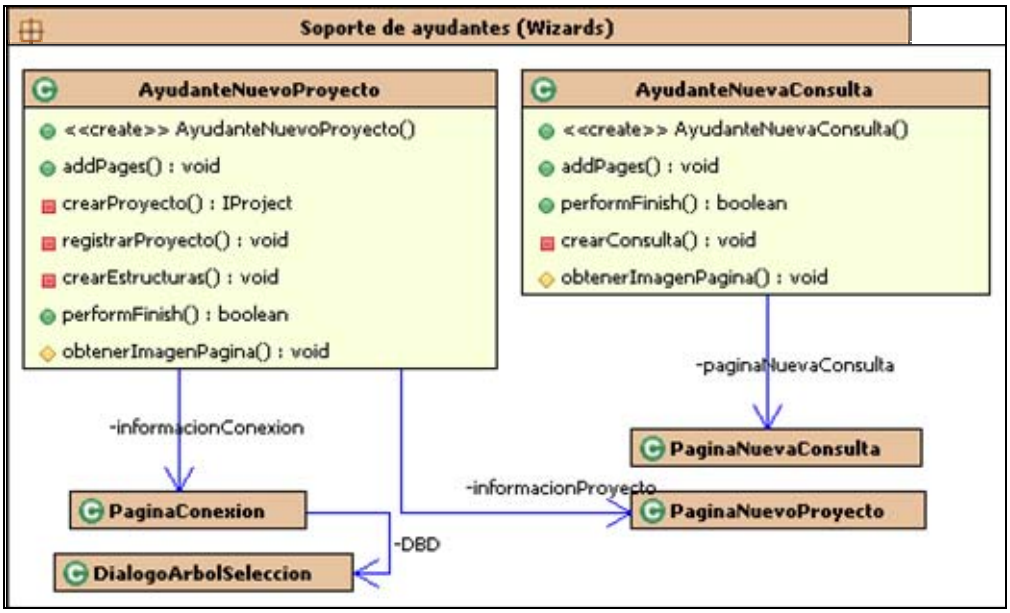


**Ilustración 25: Clases que implementan los ayudantes**

- **AyudanteNuevoProyecto**: Clase que implementa el ayudante para permitir al usuario crear un nuevo proyecto de generación de consulta. Contiene dos páginas: la página donde se establece la ruta y nombre del proyecto y la página

de configuración de la conexión, las cuales son representadas por las clases PaginaNuevoProyecto y PaginaConexion, respectivamente.

- **AyudanteNuevaConsulta**: Clase que implementa el ayudante para permitir al usuario crear una nueva consulta. Contiene una sola página, la cual solicita el nombre y ruta de la consulta, y es representada por la clase PaginaNuevaConsulta.

## Vistas del plug-in:

Se agregarán al Workbench de Eclipse dos vistas, a continuación se definen brevemente sus clases principales:

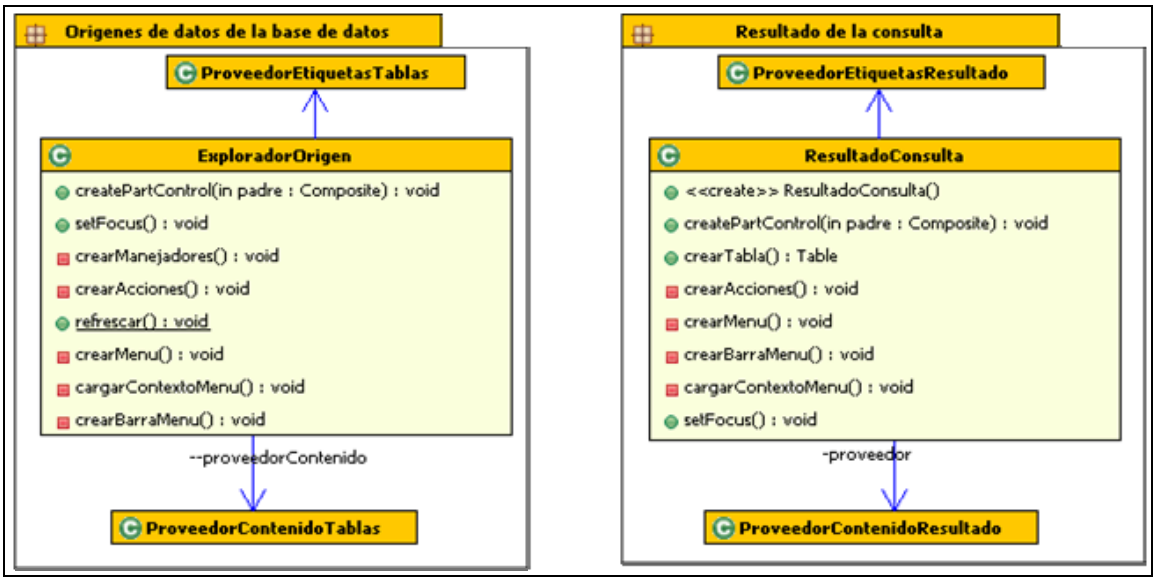


**Ilustración 26: Clases principales de las vistas del Workbench**

- **ExploradorOrigen**: Clase principal de la vista que muestra los orígenes de la base de datos seleccionada por el usuario. Aplica el patrón MVC, donde la vista es representada por ProveedorEtiquetasTablas y el modelo es representado por el ProveedorContenidoTablas.

- **ResultadoConsulta**: Clase principal de la vista que muestra el resultado de ejecución de la consulta. Aplica el patrón MVC, donde la vista es representada por ProveedorEtiquetasResultado y el modelo es representado por el ProveedorContenidoResultado.

### *3.5.3. Implementación*

#### Tarea 1: Crear proyecto de Eclipse para creación de plug-in

La primera tarea consiste en crear un nuevo proyecto en Eclipse el cual permitirá construir la nueva herramienta como un nuevo módulo.

Para la creación del proyecto se utiliza el ayudante de creación de proyecto provisto por Eclipse, cuyos pasos consisten en indicar la estructura del nuevo proyecto tipo plug-in.

Una vez realizada la tarea se tiene un proyecto para el desarrollo del plug-in a través del cual se implementará la nueva herramienta.

#### Tarea 2: Definir a naturaleza de los proyectos de generación de consulta

La herramienta de generación de utilizará a los proyectos de generación de consultas con contenedor de las consultas creadas por el usuario. Estos proyectos tendrán una naturaleza propia, la cual se define en el archivo plugin.xml, agregando dos extensiones (una para definir el tipo de proyecto y otra para definir su imagen):

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin
   id="com.ggc"
   name="Generador"
   version="1.0.0"
   provider-name="Generador">
 ...
   <extension
      id="com.ggc.naturalezaGGC"
      name="Proyecto GGC"
      point="org.eclipse.core.resources.natures">
   </extension>
   <extension
      point="org.eclipse.ui.ide.projectNatureImages">
      <image
         icon="iconos/proyecto_ggc.gif"
         id="com.ggc.naturalezaGGC.imagenNuevoProyecto"
         natureId="com.ggc.naturalezaGGC"/>
   </extension>
</plugin>
```

**Ilustración 27: Archivo plugin.xml con extensiones de preferencias**

**Tabla 17: Elementos para definir la naturaleza**

| Elemento | Descripción |
|---|---|
| Extensión | Indica que se está extendiendo una funcionalidad implementada por otro plug-in, y la cual es accedida a través del punto de extensión definido en el atributo *point*. |
| Id | Identificador de la naturaleza dentro de Eclipse. |
| Name | Tipo de proyecto que define la naturaleza. |
| icon | Imagen que se asociará a los proyectos de tipo Proyecto GGC. |

| | |
|---|---|
| natureId | Identificador de la naturaleza asociada a la imagen. |

## Tarea 3: Definir internacionalización de mensajes

La finalidad de esta tarea es lograr que el plug-in muestre sus mensajes, etiquetas y demás componentes de la interfaz gráfica en el idioma que desee trabajar el usuario. Para ello se utiliza el mecanismo de internacionalización provisto por Java a través de archivos de propiedades que contienen las cadenas a mostrar en los diferentes idiomas.

Para la herramienta se definieron los siguientes archivos: propiedades_en_US.properties para el texto en Inglés y propiedades_en_VE.properties para el texto en español. La clase ManejadorPropiedades.java, implementa un conjunto de métodos para obtener los valores desde estos archivos.

La estructura de los archivos es similar a la siguiente:

```
…
general.idioma.espanol=Spanish
general.idioma.ingles=E&nglish
…
```

**Ilustración 28: Ejemplo de archivo de internacionalización**

Estos archivos serán modificados durante la construcción del Generador gráfico de consultas para agregar el texto de sus componentes gráficos.

## Tarea 4: Crear vistas para establecer preferencias

Eclipse ofrece un mecanismo para permitir establecer las preferencias de los módulos. La herramienta de generación de consultas hará uso de esta funcionalidad para permitir al usuario establecer los parámetros de configuración, para esto se agregan nuevas extensiones en el archivo plugin.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin
   id="com.ggc"
   name="Generador"
   version="1.0.0"
   provider-name="Generador">
 ...
   <extension
         point="org.eclipse.ui.preferencePages">
      <page
            class="com.ggc.vista.preferencias.PaginaPrincipal"
            id="com.ggc.vista.preferencias.PaginaPrincipal"
            name="Generador gráfico de consultas"/>
      <page
            category="com.ggc.vista.preferencias.PaginaPrincipal"
            class="com.ggc.vista.preferencias.PaginaPreferencias"
            id="com.ggc.vista.preferencias.PaginaPreferencias"
            name="Preferencias GGC"/>
   </extension>
   <extension
         point="org.eclipse.core.runtime.preferences">
```

```
      <initializer class="com.ggc.control.preferencias.InicializadorPreferencias"/>
   </extension>
</plugin>
```

**Ilustración 29: Archivo plugin.xml con extensiones de preferencias**

**Tabla 18: Elementos para definir una página de preferencias**

| Elemento | Descripción |
|---|---|
| Extensión | Indica que se está extendiendo una funcionalidad implementada por otro plug-in, y la cual es accedida a través del punto de extensión definido en el atributo *point*. |
| Page | Interfaz para establecer las preferencias de la herramienta. Se crean dos: una principal y otra dentro de esta para indicar las preferencias del usuario. |
| Category | Agrega una nueva categoría de preferencias a la plataforma de Eclipse. |
| Class | Ubicación de la clase que implementa: la construcción de los componentes gráficos de una página de preferencias, la restauración de los valores por defecto y la actualización de las preferencias. |
| initializer class | Ubicación de la clase que carga en las preferencias los valores por defecto. |

El usuario podrá acceder a la vista a través del menú de Eclipse Window → Preferences → Generador gráfico de consultas → Preferencias GGC. El resultado de la vista se muestra en la siguiente figura:

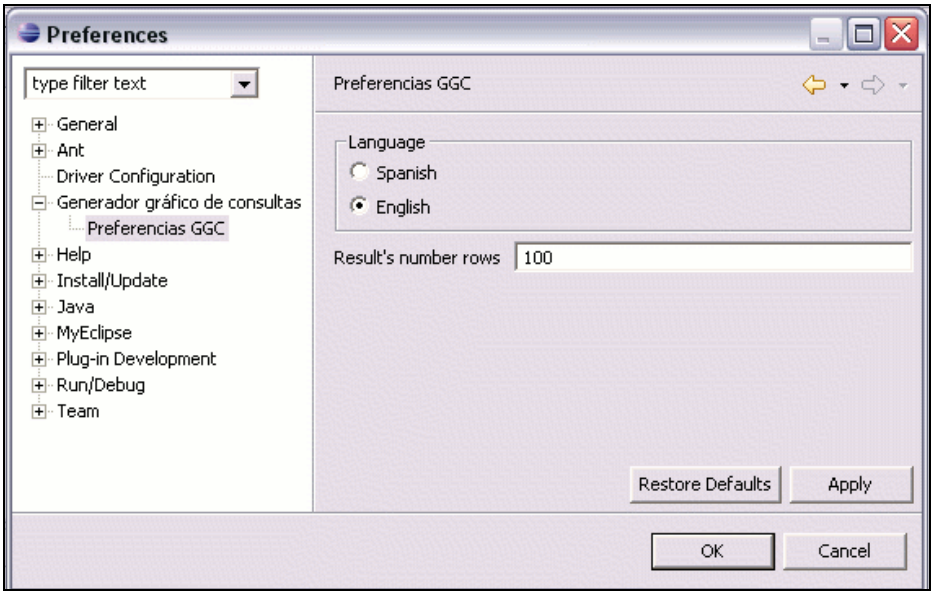


**Ilustración 30: Página de preferencias**

## Tarea 5: Crear el recurso que guarda los datos de la conexión

El archivo parametros_conexion.properties, representará la conexión del proyecto y guardará los parámetros de la conexión en forma de un archivo de propiedades.

En la siguiente ilustración se muestra un ejemplo del archivo:

```
url=jdbc:postgresql://127.0.0.1/prueba
usuario=hecane
clave=860607
controlador=org.postgresql.Driver
esquema=hecane
```

**Ilustración 31: Ejemplo del archivo parametros_conexion.properties**

Los valores almacenados son: Proveedor de base de datos (base-datos), ruta donde está alojado el controlador de conexión a través de JDBC (biblioteca), clase principal del controlador (controlador), URL de conexión (url), datos de usuario de la base de datos (usuario y clave).

Este archivo será incluido en el proyecto durante la creación del proyecto de Generación de consultas.

## Tarea 6: Implementar módulo de conexión con la base de datos

Esta tarea tiene como objetivo implementar las funcionalidades que llevan a cabo la conexión con la base de datos, empleando el patrón DAO.

Se creará una clase llamada ManejadorConexion.java, la cual implementará métodos para el manejo de las conexiones con la base de datos. Los parámetros para construir la conexión serán tomados del archivo parametros_conexion.properties, el cual estará definido como un recurso más del proyecto de generación de consultas.

## Tarea 7: Construir el ayudante (wizard) para la creación del proyecto

La creación de un proyecto de generación de consultas se llevará a cabo a través de un ayudante, el cual guiará al usuario en el proceso de creación y le permitirá indicar los parámetros de conexión de una de las bases de datos soportadas por la herramienta.

Para la creación del ayudante se utiliza el punto de extensión org.eclipse.ui.newWizards (explicado en la sección anterior del documento), el cual es provisto por Eclipse para soportar la creación de nuevos ayudantes que utilicen la misma presentación del resto de los ayudantes de la plataforma. Esta extensión se define en el archivo plugin.xml de la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin
   id="com.ggc"
   name="Generador"
   version="1.0.0"
   provider-name="Generador">

   <extension
         point="org.eclipse.ui.newWizards">
      <category
            id="%ggc.categoria.nombre"
            name="GGC"/>
```

```
      <wizard
            category="%ggc.categoria.nombre"
            class="com.ggc.control.ayudante.AyudanteNuevoProyecto"
            icon="iconos/ayudante_nuevo_proyecto_ggc.gif"
            id="com.ggc.control.ayudante.AyudanteNuevoProyecto"
            name="%ggc.wizard.crearProyecto.nombre"
            project="true">
         <description>
            Este ayudante permite llevar a cabo la creación de un proyecto de consultas
         </description>
      </wizard>
   </extension>
...
</plugin>
```

**Ilustración 32: Archivo plugin.xml con la nueva extensión para el ayudante**

**Tabla 19: Elementos para definir un nuevo ayudante en el plugin.xml**

| Elemento | Descripción |
|---|---|
| extension | Indica que se está extendiendo una funcionalidad implementada por otro plug-in, y la cual es accedida a través del punto de extensión definido en el atributo point. |
| category | Agrega una nueva categoría de ayudantes a la plataforma de Eclipse. |
| Wizard | Define un nuevo ayudante para el plug-in. Sus atributos indican: la categoría a la cual pertenece el ayudante (*category*), la clase que implementa el ayudante (*class*), el icono (*icon*), un identificador para el componente (*id*), título (*name*), si el ayudante se accede desde el menú new → Project (*project*). |
| descripcion | Breve descripción para el ayudante. |

El ayudante tiene dos páginas, una para indicar los datos de creación del proyecto, y otra para establecer los datos de conexión con la base de datos.

Para la implementación del ayudante se hace uso de la clase org.eclipse.jface.wizard.Wizard y org.eclipse.ui.INewWizard, y la clase org.eclipse.jface.wizard.WizardPage para las páginas que lo conforman.

A continuación se muestran las vistas asociadas al ayudante. La primera muestra un diálogo de creación de proyectos ya implementado por Eclipse y en el cual se puede escoger la categoría anteriormente definida:

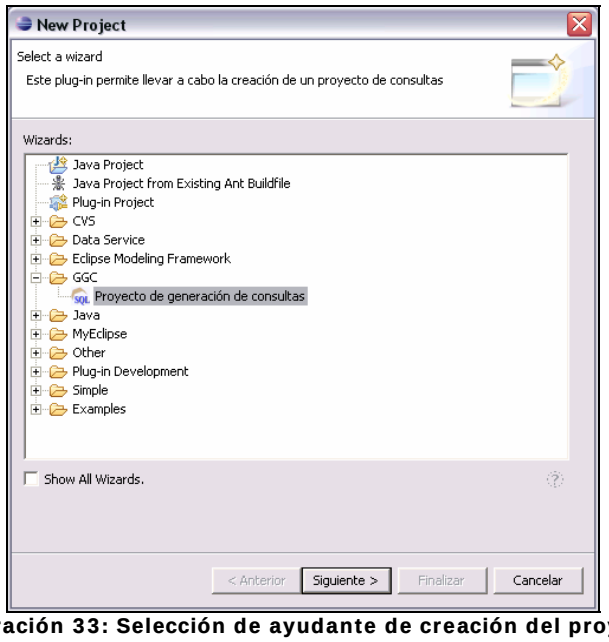


**Ilustración 33: Selección de ayudante de creación del proyecto**

La siguiente página obtiene información acerca del proyecto que se quiere crear (particularmente su ubicación y nombre):

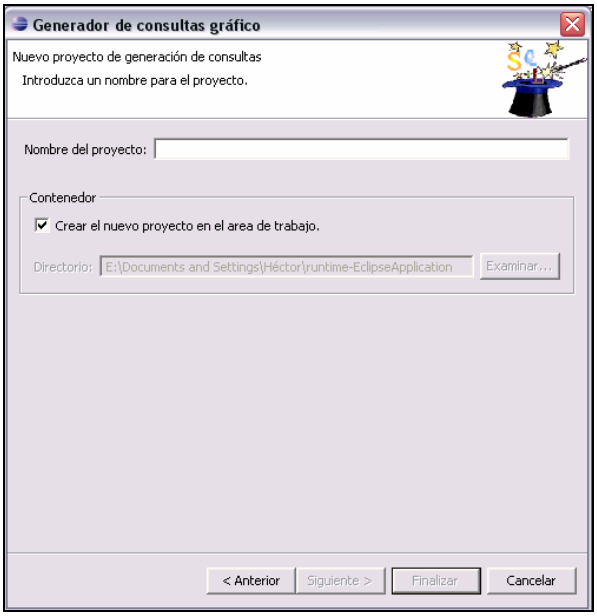


**Ilustración 34: Primera página del ayudante**

La siguiente página del ayudante permite seleccionar la base de datos sobre la cual se elaborarán las consultas y las propiedades acerca de la conexión con la misma.


**Ilustración 35: Segunda página del ayudante**

## Tarea 8: Crear ayudante (wizard) para crear una nueva consulta

La creación de una consulta se llevará a cabo a través de un ayudante, el cual guiará al usuario en el proceso de creación y le permitirá indicar el nombre del archivo de en el cual se guardará la información asociada a la consulta.

Para la creación del ayudante se utiliza el punto de extensión org.eclipse.ui.newWizards (explicado en la sección anterior del documento), el cual es provisto por Eclipse para soportar la creación de nuevos ayudantes que utilicen la misma presentación del resto de los ayudantes de la plataforma. Esta extensión se define en el archivo plugin.xml de la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin
   id="com.ggc"
   name="Generador"
   version="1.0.0"
   provider-name="Generador">

   <extension
         point="org.eclipse.ui.newWizards">
      <category
            id="%ggc.categoria.nombre"
            name="GGC"/>
          <wizard
           category="%ggc.categoria.nombre"
           class="com.ggc.ayudante.AyudanteNuevaConsulta"
           icon="iconos/editor_consulta.gif"
           id="com.ggc.ayudante.AyudanteNuevaConsulta"
           name="%ggc.ayudante.crearConsulta.nombre">
           <description>
              Este ayudante permite llevar a cabo la creación de una
consulta GGC
```

```
            </description>
        </wizard>
    </extension>
...
</plugin>
```

**Ilustración 36: Archivo plugin.xml con la nueva extensión para el ayudante**

**Tabla 20: Elementos para definir un nuevo ayudante en el plugin.xml**

| Elemento | Descripción |
|---|---|
| extension | Indica que se está extendiendo una funcionalidad implementada por otro plug-in, y la cual es accedida a través del punto de extensión definido en el atributo point. |
| category | Agrega una nueva categoría de ayudantes a la plataforma de Eclipse. |
| Wizard | Define un nuevo ayudante para el plug-in. Sus atributos indican: la categoría a la cual pertenece el ayudante (*category*), la clase que implementa el ayudante (*class*), el icono (*icon*), un identificador para el componente (*id*), título (*name*), si el ayudante se accede desde el menú new → Other →GGC → Consulta GGC. |
| descripcion | Breve descripción para el ayudante. |

A continuación se muestran las vistas asociadas al ayudante, la cual permite al usuario indicar la ubicación y el nombre del archivo:

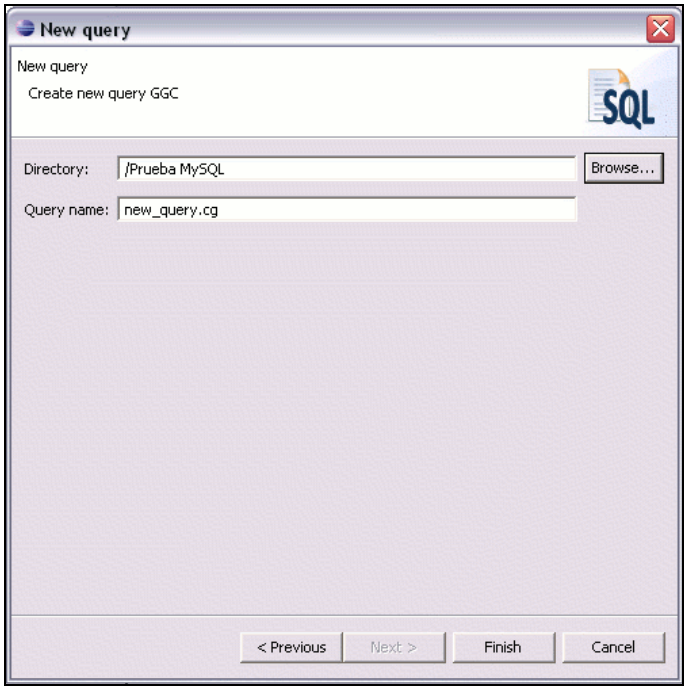


**Ilustración 37: Interfaz del ayudante para creación de consulta**

## Tarea 9: Crear editor para los parámetros de conexión

Esta tarea consiste en la construcción de un editor gráfico para modificar los parámetros de conexión con la base de datos, una vez que ya ha sido creado el proyecto.

Para incluir el editor, se define el punto de extensión en el archivo plugin.xml como sigue:

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin
   id="com.ggc"
   name="Generador"
   version="1.0.0"
   provider-name="Generador">
   ...
   <extension
      point="org.eclipse.ui.editors">
      <editor
            class="com.ggc.editor.EditorConexion"
            default="true"
            extensions="conexion"
            icon="iconos/conexion.gif"
            id="com.ggc.editor.EditorConexion"
            name="Editor de conexion"/>
    </extension>
   ...
</plugin>
```

**Ilustración 38: Archivo plugin.xml con la nueva extensión para el editor multipágina**

**Tabla 21: Elementos para definir un nuevo editor en el plugin.xml**

| Elemento | Descripción |
|---|---|
| extensión | Indica que se está extendiendo una funcionalidad implementada por otro plug-in, y la cual es accedida a través del punto de extensión definido en el atributo *point*. |
| Editor | Define un nuevo editor para el plug-in. Sus atributos indican: la clase que implementa el editor (class), el menú que se activa cuando se ejecuta el editor (contributorClass), los tipos de archivos que puede abrir (extensions), si es el editor por defecto de ese tipo de archivos (default), el icono (icon), un identificador para el componente (id), título (name). |
| Class | Referencia a la clase que implementa las funcionalidades principales de edición. |
| default | Establece al editor como editor por defecto para los archivos con extensión .conexion. |
| Icon | Ícono empleado por el editor. |
| Id | Identificador del editor en el Marco de trabajo de Eclipse. |
| Name | Nombre del nuevo editor. |

La interfaz del editor creado se muestra en la siguiente figura:

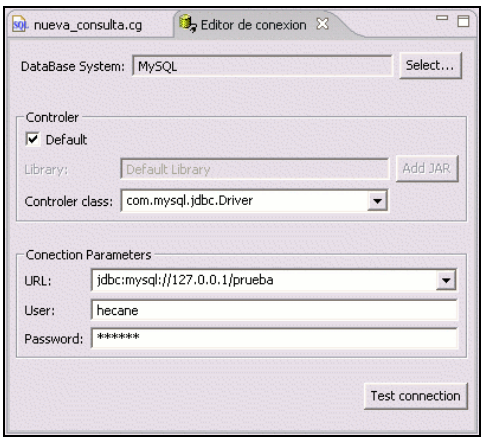


**Ilustración 39: Editor de parámetros de conexión**

## Tarea 10: Definir el editor de consultas

Esta tarea consiste en la definición del editor multipágina que permitirá la edición de la consulta.

Se crea una estructura base con diferentes vistas vacías para el editor de las consultas SQL. Esta estructura de editor tiene como finalidad trabajar con las diferentes cláusulas que conforman una consulta SQL. Está compuesto de diferentes páginas para trabajar con las diferentes cláusulas de la sintaxis SQL:

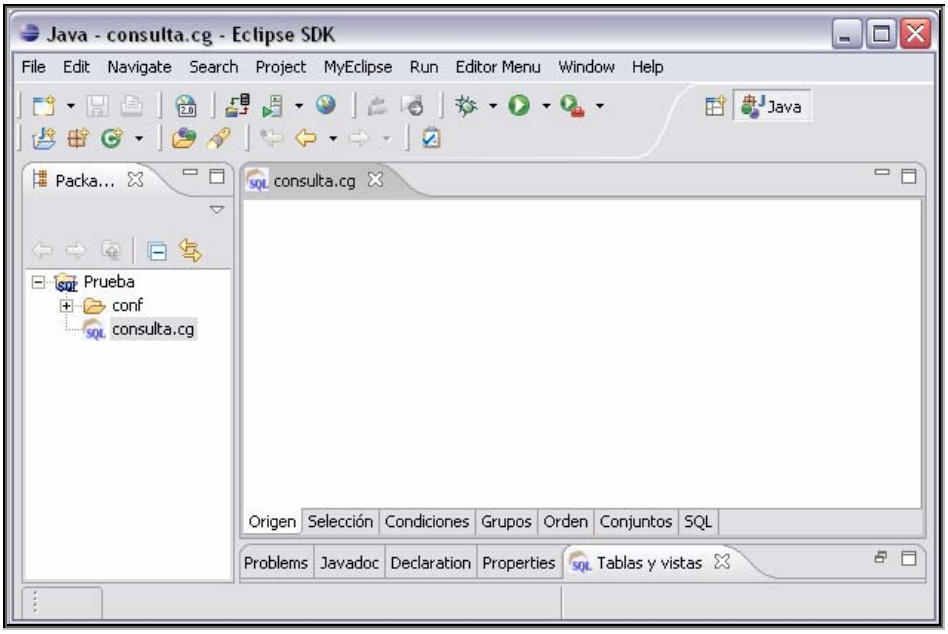


**Ilustración 40: Eclipse y el editor multipágina de GGC.**

- Página origen: Manipula el origen de datos (tablas, vistas, columnas, enlaces).
- Página selección: Columnas y campos que serán mostrados.
- Página condiciones: Condiciones de filtrado de la consulta.
- Página grupos: Definición de grupos para la consulta.
- Página orden: Criterios para ordenar el resultado.
- Página conjuntos: Operaciones sobre conjuntos.
- Página SQL: Sintaxis SQL de la consulta.

Un editor puede ser agregado al ambiente de eclipse utilizando el punto de extensión org.eclipse.ui.editors (mencionado en el marco teórico) ofrecido por Eclipse.

Para incluir el editor, se define el punto de extensión en el archivo plugin.xml como sigue:

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin
   id="com.ggc"
   name="Generador"
   version="1.0.0"
   provider-name="Generador">
```

```
   ...
   <extension
      point="org.eclipse.ui.editors">
      <editor
         class="com.ggc.editor.EditorMultipagina"
         contributorClass="com.ggc.editor.menu.MenuEditores"
         default="true"
         extensions="cg"
         icon="iconos/editor_consulta.gif"
         id="com.ggc.editor.EditorMultipagina"
         name="Editor de consultas"/>
   </extension>
   ...
</plugin>
```

**Ilustración 41: Archivo plugin.xml con la nueva extensión para el editor multipágina**

**Tabla 22: Elementos para definir un nuevo editor en el plugin.xml**

| Elemento | Descripción |
|---|---|
| Extensión | Indica que se está extendiendo una funcionalidad implementada por otro plug-in, y la cual es accedida a través del punto de extensión definido en el atributo *point*. |
| Editor | Define un nuevo editor para el plug-in. Sus atributos indican: la clase que implementa el editor (class), el menú que se activa cuando se ejecuta el editor (contributorClass), los tipos de archivos que puede abrir (extensions), si es el editor por defecto de ese tipo de archivos (default), el icono (icon), un identificador para el componente (id), título (name). |

## Tarea 11: Implementar módulo para consultar diccionario de la base de datos

Para permitir al usuario incluir en la consulta las tablas y vistas existentes en la base de datos, se construirá un módulo que permita cargar esta información en la aplicación.

La información será obtenida desde la base de datos; utilizando JDBC 3.0 para consultar el diccionario de datos de la base de datos seleccionada por el usuario y asociada al proyecto.

Estas funcionalidades estarán implementadas en la clase com.ggc.metadatos.ManejadorDiccionario.java y todos los proyectos tendrán asociado un objeto de esta clase.

## Tarea 12: Crear vista con tablas y vistas de la base de datos

El objetivo de esta tarea es realizar una vista que permitirá al usuario agregar los orígenes de datos disponibles para la consulta.

La vista tendrá como nombre *Tablas y vistas*, y contendrá los nombres de las tablas y vistas de la base de datos asociada al proyecto.

Además de mostrar los orígenes de datos, la vista tendrá un menú que permitirá al usuario recargarlas desde la base de datos.

Para la creación de la vista se utiliza el punto de extensión org.eclipse.ui.views (explicado en la sección anterior del documento), el cual es provisto por Eclipse para

soportar la creación de nuevas vistas para la plataforma. Esta extensión se define en el archivo plugin.xml de la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin
   id="com.ggc"
   name="Generador"
   version="1.0.0"
   provider-name="Generador">
   ...
        <extension point="org.eclipse.ui.views">
              <category
                 id="%ggc.categoria.nombre"
                 name="GGC"/>
              <view
                 category="%ggc.categoria.nombre"
                 class="com.ggc.vista.ExploradorOrigen"
                 icon="iconos/explorador.gif"
                 id="com.ggc.vista.ExploradorGGC"
                 name="%ggc.vista.tabla.nombre"/>
         </extension>
...
</plugin>
```

**Ilustración 42: Archivo plugin.xml con la nueva extensión para la vista**

| Elemento | Descripción |
|---|---|
| extension | Indica que se está extendiendo una funcionalidad implementada por otro plug-in, y la cual es accedida a través del punto de extensión definido en el atributo *point*. |
| category | Agrega una nueva categoría de vistas a la plataforma de Eclipse. |
| View | Define una nueva vista para el plug-in. Sus atributos indican: la categoría a la cual pertenece la vista (*category*), la clase que implementa la vista (*class*), el icono (*icon*), un identificador para el componente (*id*), título (*name*). |

**Ilustración 43: Elementos para definir una nueva vista en el plugin.xml**

Para la implementación de la vista se hace uso de la clase org.eclipse.ui.part.ViewPart, la cual es una clase abstracta con la implementación base para las vistas del marco de trabajo de eclipse.

Las tablas esquemas y columnas de la base de datos de la vista se mostrarán de forma jerárquica. Para esto se construye el componente de JFace TreeViewer, el cual es representado por la clase org.eclipse.jface.viewers.TreeViewer.

El TreeViewer implementa los métodos para la creación del árbol y hace uso de un proveedor de contenido (contiene la información completa de los nodos del árbol), y un proveedor de etiquetas (indica cómo se muestran los nodos del árbol).

Las tres principales clases que implementan la vista de tablas:

- com.ggc.vistas.ExploradorOrigen: Es una subclase de org.eclipse.ui.part.ViewPart la cual contiene un TreeViewer para desplegar una interfaz para mostrar al usuario las tablas disponibles en la base de datos. Contiene un árbol cuyos nodos son las tablas, columnas y esquemas (si son soportados por la base de datos).

- com.ggc.vistas.ExploradorOrigen.ProveedorContenido: Es una subclase de org.eclipse.jface.viewers.ITreeContentProvider y provee la información de los nodos a mostrar en el árbol a partir del modelo.

- com.ggc.vistas.ExploradorOrigen.ProveedorEtiqueta: Es una subclase de org.eclipse.jface.viewers.LabelProvider y quien indica al ExploradorOrigen como deben mostrarse cada uno de los nodos del árbol.

A continuación se muestra la vista de tablas, la cual se puede acceder a través del menú Window de Eclipse (Window → Show view → GGC → Tablas y vistas):

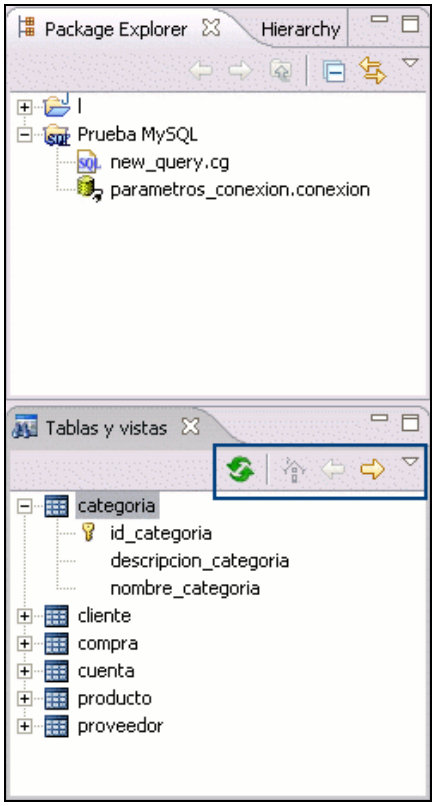


**Ilustración 44: Vista de tablas y vistas**

La vista posee una barra de menú que permite ejecutar acciones sobre ella, las cuales permiten al usuario navegar entre los esquemas de tablas (si son soportados por el sistema manejador de base de datos), tablas y atributos. Además se tienen una acción "Refrescar" que permite recargar las tablas desde la base de datos.

## Tarea 13: Crear modulo de ejecución de la consulta

Esta tarea contempla la creación de una interfaz que permita al usuario ejecutar y obtener resultados de la ejecución de una consulta SQL.

La vista tendrá como nombre *Resultado de la consulta*, y contendrá un menú que permitirá al usuario poder ejecutar la consulta actual que se está editando.

Para la creación de la vista se utiliza el punto de extensión org.eclipse.ui.views. Esta extensión se define en el archivo plugin.xml de la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin
   id="com.ggc"
   name="Generador"
   version="1.0.0"
   provider-name="Generador">
   ...
        <extension point="org.eclipse.ui.views">
              <category
                 id="%ggc.categoria.nombre"
                 name="GGC"/>
              ...
              <view
                 category="%ggc.categoria.nombre"
                 class="com.ggc.vista.ResultadoConsulta"
                 icon="iconos/resultado.gif"
                 id="com.ggc.vista.ResultadoConsulta"
                 name="%ggc.vista.resultado.nombre"/>
         </extension>
...
</plugin>
```

**Ilustración 45: Archivo plugin.xml con la nueva extensión para la vista**

| Elemento | Descripción |
|---|---|
| extension | Indica que se está extendiendo una funcionalidad implementada por otro plug-in, y la cual es accedida a través del punto de extensión definido en el atributo *point*. |
| category | Agrega una nueva categoría de vistas a la plataforma de Eclipse. |
| View | Define una nueva vista para el plug-in. Sus atributos indican: la categoría a la cual pertenece la vista (*category*), la clase que implementa la vista (*class*), el icono (*icon*), un identificador para el componente (*id*), título (*name*). |

**Ilustración 46: Elementos para definir una nueva vista en el plugin.xml**

Para la implementación de la vista se hace uso de la clase org.eclipse.ui.part.ViewPart la cual es una clase abstracta con la implementación base para las vistas del marco de trabajo de eclipse.

Dicha interfaz estará conformada de una tabla de datos que mostrará en su cabecera los nombres de las columnas que conforman la selección de la consulta SQL y en la sección de datos, la información de los registros recuperados al ejecutar la consulta. Para esto se construye el componente de JFace TableViewer, el cual es representado por la clase org.eclipse.jface.viewers.TableViewer.

El TableViewer implementa los métodos para la creación de una tabla y hace uso de un proveedor de contenido (contiene la información completa de los registros a resultantes), y un proveedor de etiquetas (indica cómo se muestran los registros de la tabla).

Las tres principales clases que implementan la vista:

- com.ggc.vistas.ResultadoConsulta: Es una subclase de org.eclipse.ui.part.ViewPart la cual contiene un TableViewer para desplegar una interfaz para mostrar al usuario el resultado de ejecución de una consulta.

- com.ggc.modelo.ProveedorContenidoResultado: Es una subclase de org.eclipse.jface.viewers.IStructuredContentProvider. Ejecuta la consulta actual y provee la información de las filas a mostrar en la tabla a partir del modelo.

- com.ggc.vistas.ExploradorOrigen.ProveedorEtiquetasResultado: Es una subclase de org.eclipse.jface.viewers.ITableLabelProvider y quien indica a la vista ResultadoConsulta como deben mostrarse los registros en la tabla.

A continuación se muestra la vista de resultado, la cual se puede acceder a través del menú Window de Eclipse (Window → Show view → GGC → Resultados de la consulta):


| Row Id | ID_CLIENTE | NOMBRE_CLIE... | EDAD_CLIENTE | DIRECCION_CL... | CORREO_CLIE... | ID_CLIENTE | CLAVE |
|---|---|---|---|---|---|---|---|
| 1 | 1 | Héctor Martínez | 27 | Sta. Mónica | hectorenriquem... | 1 | hemo |
| 2 | 2 | Anely Poleo | 19 | Coche | apoleo@jeje.com | 2 | polo |

**Ilustración 47: Vista de resultado de ejecución de una consulta**

La vista posee una barra de menú que permite ejecutar acciones sobre ella, las cuales permiten al usuario ejecutar la consulta que se está editando actualmente[20], a través de la acción Ejecutar (Botón con la flecha roja).

## Tarea 14: Crear perspectiva del plug-in

El objetivo de esta tarea es establecer cuál será la distribución de los componentes del Workbench, creados para el plug-in en las tareas anteriores.

Para definir la perspectiva se agregan una nueva extensión en el archivo plugin.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin
   id="com.ggc"
   name="Generador"
   version="1.0.0"
   provider-name="Generador">
   ...
   <extension
      id="%ggc.perspectiva"
      name="GGC"
      point="org.eclipse.ui.perspectives">
      <perspective
         class="com.ggc.perspectiva.PerspectivaGGC"
         id="com.ggc.perspectiva.PerspectivaGGC"
         name="%ggc.perspectiva.nombre"/>
   </extension>
```

[20] La consulta ejecutada para obtener esta imagen se tiene copiada directamente en el proveedor de contenido.

```
...
</plugin>
```

**Ilustración 48: Archivo plugin.xml con la nueva extensión para la vista**

| Elemento | Descripción |
|---|---|
| extension | Indica que se está extendiendo una funcionalidad implementada por otro plug-in, y la cual es accedida a través del punto de extensión definido en el atributo *point*. |
| perspective | Define la perspectiva a crear. |
| class | Ubicación de la clase que implementa la perspectiva. |
| id | Identificador de la perspectiva en Eclipse. |
| name | Nombre a través del cual se accederá a la perspectiva. |

**Ilustración 49: Elementos para definir una nueva perspectiva en el plugin.xml**

Para la implementación de la perspectiva se hace uso de la clase org.eclipse.ui.IPerspectiveFactory la cual es una clase abstracta con la implementación base para las perspectivas del marco de trabajo de eclipse. La clase que extiende de ella es llamada PerspectivaGGC.java.

La clase PerspectivaGGC, sobreescribe el método createInitialLayout, para indicar cuáles serán las vistas, editores y demás componentes gráficos a mostrar y su ubicación dentro del entorno de desarrollo:

```
public void createInitialLayout(IPageLayout layout)
{
	String areaEditor = layout.getEditorArea();
	IFolderLayout vistaSuperiorIzquierda;
	IFolderLayout vistaInferiorIzquierda;
	IFolderLayout vistaInferior;

	vistaSuperiorIzquierda = layout.createFolder("vistaSuperiorIzquierda",
		IPageLayout.LEFT, 0.25f, areaEditor);
	vistaSuperiorIzquierda.addView(IPageLayout.ID_RES_NAV);
		vistaSuperiorIzquierda.addPlaceholder(JavaUI.ID_PACKAGES);

	vistaInferiorIzquierda = layout.createFolder("vistaInferiorIzquierda",
		IPageLayout.BOTTOM, 0.50f, "vistaSuperiorIzquierda");
	vistaInferiorIzquierda.addView(VISTA_EXPLORADOR);

	vistaInferior = layout.createFolder("vistaInferior", IPageLayout.BOTTOM,
		0.75f, areaEditor);
	vistaInferior.addView(VISTA_RESULTADO);
	vistaInferior.addView(IPageLayout.ID_PROBLEM_VIEW);
	vistaInferior.addPlaceholder(IPageLayout.ID_TASK_LIST);
}
```

**Ilustración 50: Implementación de la distribución de las vistas y editores del plug-in**

Se explicará el funcionamiento estudiando las tres primeras instrucciones, en las cuales se incluyen como parte de la perspectiva las vistas de Eclipse Navegador y Explorador de paquetes:

1. Primero se define el área, dentro del Workbench, en la que serán abiertas el Navegador o el Explorador de paquetes. Se indica que estará a la izquierda del área de edición.
2. Luego se define al navegador como vista por defecto en esa área.
3. Finalmente se indica que, en caso de abrirse el Explorador de paquetes, este será ubicado en la misma área que el Navegador.

A continuación se muestra la vista de tablas, la cual se puede acceder a través del menú Window de Eclipse (Window → Open perspective → Other → GGC:

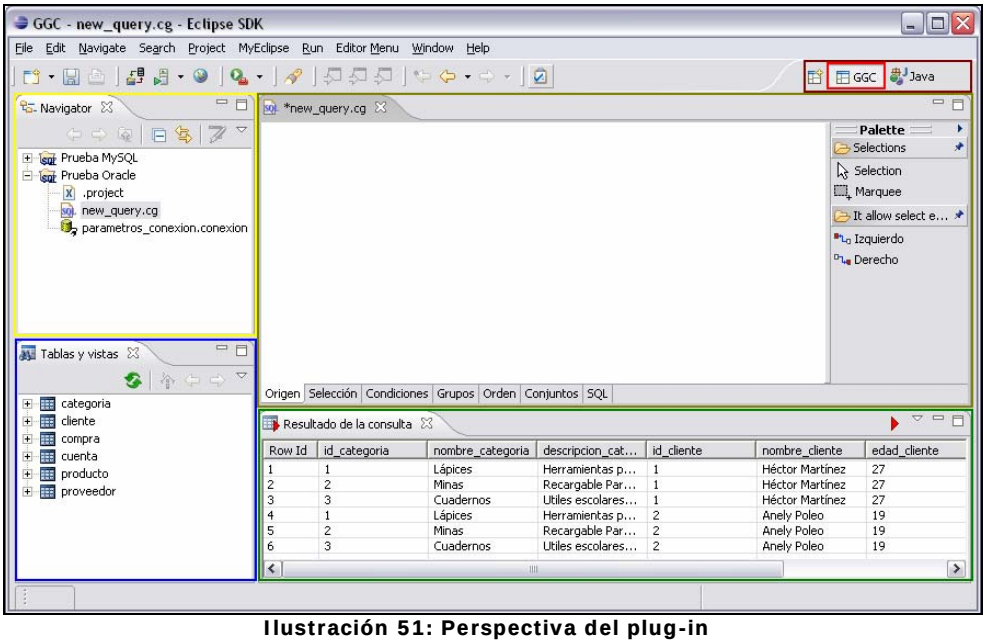


**Ilustración 51: Perspectiva del plug-in**

## 3.5.4. Pruebas

Para realizar las pruebas de forma automatizada se implementaron una Suite de pruebas JUnit que contemplan un conjunto de casos de pruebas para las funcionalidades de la iteración.

### Prueba de las preferencias

Esta prueba consiste en verificar que el módulo para establecer las preferencias funcione correctamente:

1. La interfaz debe permitir agregar los valores de configuración del plug-in: Se pudieron establecer todos los valores correctamente.

2. La configuración debe almacenarse permanentemente después de ser establecidos: Los valores de la configuración se mantienen después de aplicarlos.
3. Se debe permitir la restauración de los valores por defecto del plug-in: La restauración de los valores por defecto funciona correctamente.

### Prueba de la internacionalización

Se debe verificar que los mensajes del plug-in sean indicados en los dos idiomas establecidos (inglés o español), de acuerdo a las preferencias del usuario:

1. Mensajes en inglés: Fueron mostrados correctamente.
2. Mensajes en español: Fueron mostrados correctamente.

### Prueba de la conexión con la base de datos

Esta prueba consiste en verificar el comportamiento del módulo que establece la conexión con la base de datos:

1. Obtener parámetros de conexión del archivo de conexión: Son cargados correctamente en objetos que son utilizados para establecer la conexión con la base de datos.
2. Parámetros de conexión inválidos: Si el sistema encuentra algún parámetro de conexión incorrecto (por ejemplo, no encuentra la base de datos), se atiende el error y se retorna una conexión nula.
3. Parámetros de conexión válidos: Se establece correctamente la conexión y se retorna una referencia a esta, para que pueda ser empleado por la herramienta.

### Prueba del ayudante de creación del proyecto

Esta prueba consiste en verificar el funcionamiento correcto del ayudante para la creación del proyecto, las pruebas a realizar son:

1. Parámetros inválidos: En caso de indicar parámetros incorrectos o dejar en vacio parámetros obligatorios, no se permite al usuario avanzar al siguiente paso del ayudante y se muestra en la parte superior del mismo el error correspondiente.
2. Almacenamiento de pasos anteriores: Se permite al usuario navegar entre las páginas del ayudante y modificar valores que ya han sido establecidos.
3. Completitud de los pasos: Al completar los pasos del ayudante se crea correctamente un proyecto de generación de consultas con un archivo que define la conexión.

### Prueba del ayudante de creación de la consulta

Esta prueba consiste en verificar el funcionamiento correcto del ayudante para la creación de la consulta, las pruebas a realizar son:

1. Parámetros inválidos: En caso de indicar parámetros incorrectos o dejar en vacio parámetros obligatorios, no se permite al usuario avanzar al siguiente paso del ayudante y se muestra en la parte superior del mismo el error correspondiente.
2. Almacenamiento de pasos anteriores: Se permite al usuario navegar entre las páginas del ayudante y modificar valores que ya han sido establecidos.

3. Completitud de los pasos: Al completar los pasos del ayudante se crea correctamente una consulta, en la ruta del proyecto indicada por el usuario.

### Prueba de la modificación de los parámetros de conexión

Esta prueba consiste en verificar que se puedan cambiar los parámetros de conexión a través de una interfaz y que dichos cambios se efectúen permanentemente.

1. Componentes gráficos: Funcionan correctamente los componentes gráficos que permiten seleccionar los valores de los parámetros de conexión, y verificar si se establece o no la conexión.
2. Parámetros inválidos: Se indica el error en la parte inferior del editor.
3. Parámetros válidos: No se muestra el error.
4. Permanencia de cambios: Se marca el archivo como modificado y se da la opción de guardar al usuario.

### Prueba de asociación del archivo de consulta

Esta prueba consiste en verificar que los archivos de la consulta estén asociados al editor de consulta.

1. La asociación resultó ser correcta y al abrir una consulta su editor por defecto es el definido para su edición.

Para realizar la prueba se crea un nuevo archivo de consulta empleando el ayudante. El resultado de la operación fue el esperado, se abrió el editor multipáginas:

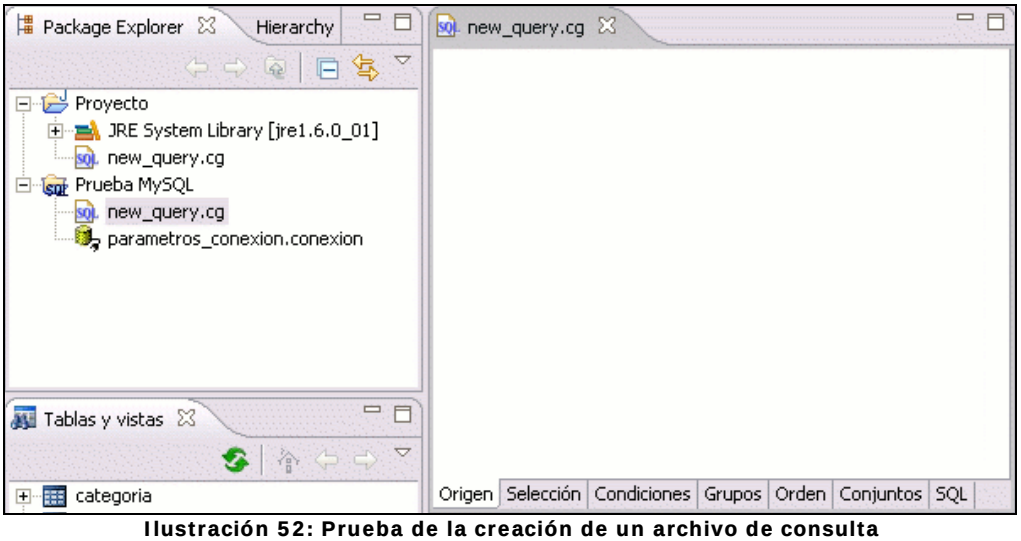


**Ilustración 52: Prueba de la creación de un archivo de consulta**

### Prueba del módulo de carga de vistas y tablas

Esta prueba consiste en verificar que se carguen correctamente en una vista de eclipse todas las tablas y vistas de la base de datos y que están disponibles para emplear en la construcción de las consultas de un proyecto.

1. No hay contenido para mostrar: Se muestra la vista en blanco. Esta situación puede ocurrir cuando no se tienen seleccionado ningún proyecto o no se pudo establecer conexión con la base de datos.
2. Hay contenido para mostrar: Se muestran las tablas y vistas en un árbol.
3. Refrescar contenido: Se verificó que al cambiar entre proyectos la vista se actualiza con el contenido correspondiente a la base de datos asociada al proyecto seleccionado.
4. Actualizar contenido: Se verificó que al actualizar la vista, a través de su menú, se obtiene nuevamente el contenido desde la base de datos asociada.

### Prueba del módulo de ejecución

Consiste en verificar el correcto funcionamiento de la ejecución de una consulta SQL y que los resultados se reflejen en la vista de resultados.

1. Ejecutar consulta: Si la ejecución de la consulta se lleva a cabo satisfactoriamente, se devuelve los registros resultantes; en caso contrario, se devuelve una lista vacía.
2. No hay contenido para mostrar: Se muestra la vista en blanco y se le indica al usuario la causa del error. Esta situación puede ocurrir cuando no se tiene seleccionada ninguna consulta, ocurra un error durante la ejecución de la misma o no se pudo establecer conexión con la base de datos.
3. Hay contenido para mostrar: Se muestra el conjunto de registros que se obtienen al ejecutar la consulta.
4. Límite de registros: Se verificó que la cantidad de registros mostrados fuese inferior a la cantidad de registros establecidos por el usuario.

### Prueba de la perspectiva

Verificar que la disposición de la perspectiva se lleve a cabo correctamente: Los componentes del generador se encuentran bien ubicados.

## Síntesis

Durante esta iteración se construyeron los componentes, que proveen al usuario utilidades para la generación de consultas de forma gráfica dentro del entorno integrado de eclipse.

Al completar esta iteración la herramienta tiene: ayudantes, editores, vistas e interfaces de configuración, aprobados por el usuario. En las siguientes iteraciones nos enfocaremos en las historias de usuario relacionadas con la edición de la consulta gráfica.

## 3.6. Iteración III

En esta iteración se construirán las funcionalidades que permiten la edición de la clausula FROM de una consulta de forma gráfica; lo cual consiste en indicar desde que tablas o vistas de la base de datos se tomaran los datos.

Para ello se emplearán las bibliotecas provistas por GEF y Draw2D, las cuales proveen mecanismos para desarrollar componentes gráficos con el comportamiento requerido por el usuario.

### *3.6.1. Planificación*

En esta iteración se trabajará sobre las siguientes historias de usuario:

**Tabla 23: Historias de usuario de la iteración III**

| Id. | Descripción |
|---|---|
| HU-12 | La consulta debe ser generada de forma gráfica, permitiendo la edición de cada una de sus cláusulas. |
| HU-13 | La cláusula FROM debe ser generada a través de un editor gráfico que permita manipular las tablas y relaciones entre ellas. |
| HU-14 | Los orígenes de datos deberán tener componentes gráficos que permitan seleccionar las columnas como parte del resultado de la consulta. |
| HU-19 | La consulta generada después de que el usuario edite cada una de las secciones de la consulta, debe ser mostrada en un campo de texto sobre el cual el usuario pueda seleccionar para poder copiar y pegar. |

**Tabla 24: Tareas de la iteración III**

| Historia de usuario | Tarea |
|---|---|
| HU-13 | 1. Definir la página de enlaces.<br>2. Construcción del área donde se dibujarán los elementos gráficos.<br>3. Construcción de la paleta de herramientas. |
| HU-14 | 4. Creación de la representación visual del origen de datos (tabla, vista). |
| HU-13 | 5. Crear políticas y comandos que permitan manipular los orígenes de datos.<br>6. Creación de la representación gráfica del enlace o relación entre los orígenes de datos.<br>7. Crear políticas y comandos que permitan manipular las relaciones entre los orígenes de datos.<br>8. Integración de la vista explorador al editor gráfico.<br>9. Creación del diálogo enlace. |
| HU-19 | 10. Generación de la sintaxis SQL de la cláusula de origen. |

### *3.6.2. Diseño*

A continuación se definen las clases principales de esta iteración. Las mismas fueron agrupadas para facilitar el diseño.

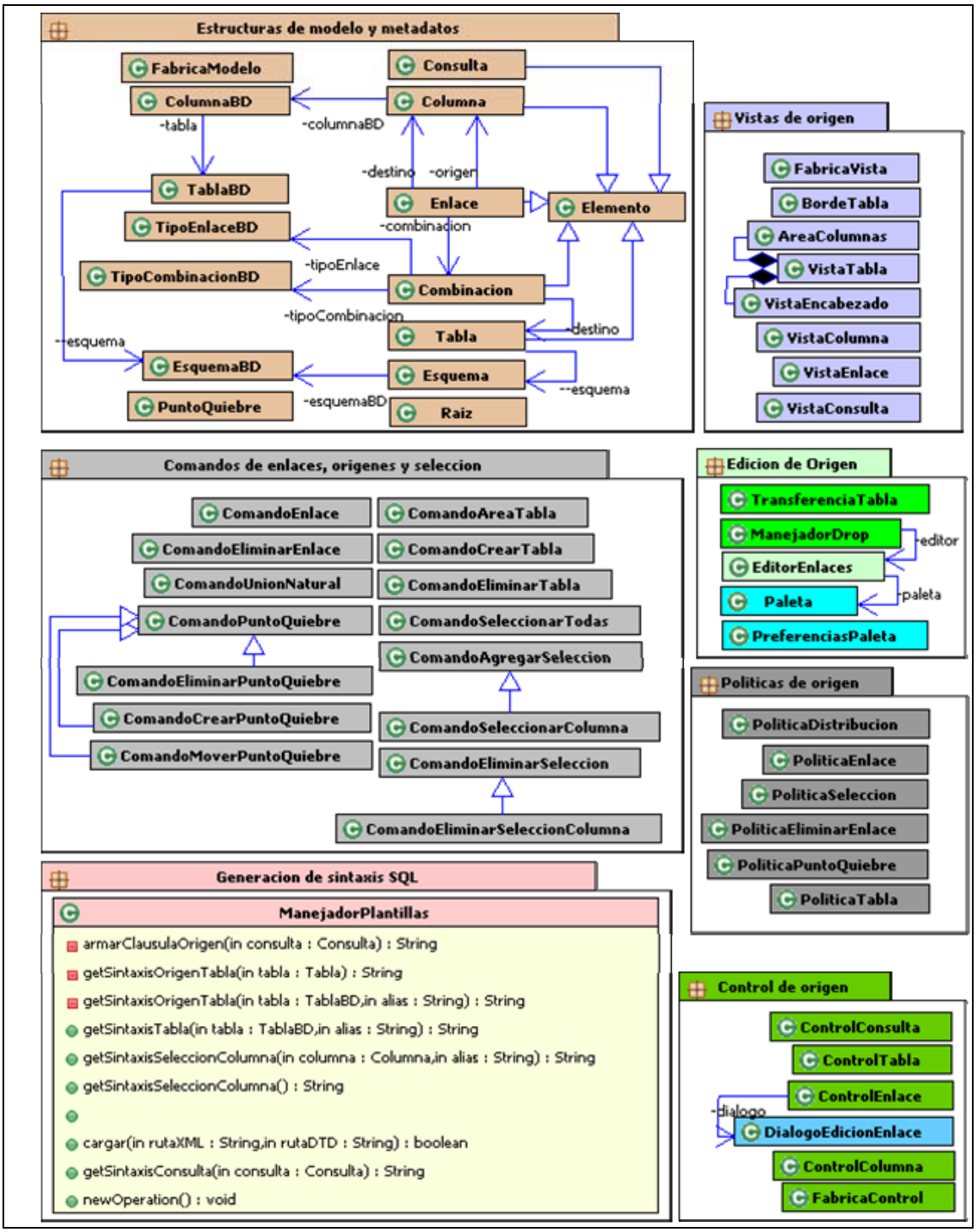


**Ilustración 53: Clases principales de la iteración III**

- **Vistas de las figuras del editor**: Estas clases representan la vista del patrón MVC, y contienen información acerca de cómo se deben mostrar cada uno de los componentes visuales del editor enlace. Por ejemplo la clase VistaColumna tiene la siguiente información acerca de la representación gráfica de la

columna: color de fondo, color de las letras, posición donde esta dibujada e imagen de la columna. Se emplea el patrón método de fabricación para crear objetos de estas clases.

- **Estructuras de modelo y metadatos**: Son un conjunto de clases que representan el modelo del patrón MVC y contienen la información acerca de cada una de las figuras a ser mostradas en el editor de enlaces. La clase Columna provee información tal como: el nombre de la columna, si es o no clave primaria, entre otras. Se emplea el patrón método de fabricación para crear objetos de estas clases.

- **Comandos de enlaces, orígenes y selección:** Son un conjunto de clases que implementan las acciones que el usuario puede ejecutar sobre cada una de las figuras del editor. Por ejemplo, la clase ComandoCrearTabla es quien lleva a cabo la inclusión de un origen en la cláusula FROM de la consulta.

- **Políticas de enlaces, orígenes y selección:** Son un conjunto de clases que indican cuales son las acciones (manejadas en GEF por comandos) que puede llevarse a cabo sobre cada una de las figuras del editor. Por ejemplo, la clase PoliticaTabla indica que una tabla se puede: agregar, mover de posición, cambiar de tamaño, eliminar y seleccionar.

- **Controladores de las figuras**: Representan el control en el patrón MVC y son los encargados de: establecer las políticas, refrescar la vista de las figuras, entre otras. Por ejemplo, la clase ControlTabla define sus políticas, comunica las clases VistaTabla y Tabla. Se emplea el patrón método de fabricación para crear objetos de estas clases.

- **Generación de sintaxis SQL**: Son métodos que generan la instrucción SQL correspondiente a la cláusula FROM.

### *3.6.3. Implementación*

#### Tarea 1: Definir la página de enlaces

Esta es la Página Origen del editor multipáginas creado en la iteración anterior. Es la página que se abre por defecto cuando se ejecuta el editor de consultas. Contiene toda la información sobre el origen de datos de una consulta SQL, es decir, tablas, vistas, columnas a mostrar y los enlaces entre las tablas y vistas.

Para ello se creó la clase “EditorEnlace”, que extiende de GraphicalEditorWithFlyoutPalette. Esta es una clase especializada que nos permite agregar una página al editor, orientada a la manipulación de gráficos, eventos e interacciones de usuario a través del teclado y el ratón.

## Tarea 2: Construcción del área donde se dibujarán los elementos gráficos

En Draw2d, para poder dibujar cualquier tipo de figura, hay que establecer área de dibujo. Es por ello que se elaboró una clase que funciona como una especie de lienzo donde se pueden dibujar las diferentes figuras que se utilizarán en el editor. En esta área se especifica el tipo de distribución (tipo de disposición en la pantalla) y se define el comportamiento de las figuras a través del mecanismo de políticas y comandos utilizado por de GEF.

La clase que implementa el lienzo (VistaConsulta) extiende a "FreeformLayerdPane" para permitir que las figuras que se dibujen en el mismo se puedan salir de los límites del área que está mostrando la ventana y se establece la distribución a "FreeformLayout" ya que da al lienzo la capacidad de colocar las diferentes figuras en cualquier punto del mismo.

La implementación del lienzo se muestra a continuación:

```
…
public class VistaConsulta extends FreeformLayeredPane
{
   public VistaConsulta()
   {
        setLayoutManager(new FreeformLayout()); //Establece la distribución
        setBorder(new MarginBorder(5)); //Establece el margen del lienzo
        setBackgroundColor(ColorConstants.white); //Color de fondo del lienzo
        setOpaque(true); //Indica que el lienzo es opaco.
   }
}
```

**Ilustración 54: Clase que implementa la interfaz del lienzo.**

Ahora se muestran el control relacionado al lienzo.

```
…
public class ControlConsulta extends AbstractGraphicalEditPart implements
PropertyChangeListener
{
   protected IFigure createFigure() //Crea el lienzo
   { return FabricaVista.crearVistaConsulta();}

   protected void createEditPolicies() //Establece las políticas a aplicar a las figuras
   {
        installEditPolicy(EditPolicy.LAYOUT_ROLE, new PoliticaDistribucion());
   }

   protected List getModelChildren(){ return getConsulta().getTablas();}

   protected Consulta getConsulta(){ return (Consulta)getModel();}

   public void propertyChange(PropertyChangeEvent evento) //Atiende los eventos de la
consulta
   {
        String propiedad = evento.getPropertyName();
        if (Consulta.TABLA.equals(propiedad))refreshChildren();
   }
…
}
```

**Ilustración 55: Clase que implementa el control del lienzo.**

## Tarea 3: Construcción de la paleta de herramientas

Después de definir el área donde se dibujarán los elementos gráficos, se creó una paleta de herramientas para que el usuario pueda seleccionar el tipo de operación que desea realizar sobre el editor.

Las funcionalidades agregadas a esta paleta son las siguientes:

- **Seleccionar**: Permite que el usuario seleccione una tabla que se encuentra en el editor.
- **Selección múltiple**: Esta funcionalidad provee al usuario una herramienta que selecciona un conjunto de tablas que se encuentren en el editor.
- **Enlace**: Permite establecer una combinación entre dos tablas de origen de datos que se encuentren en el editor.

La siguiente ilustración muestra el editor multipáginas con la paleta de herramientas.

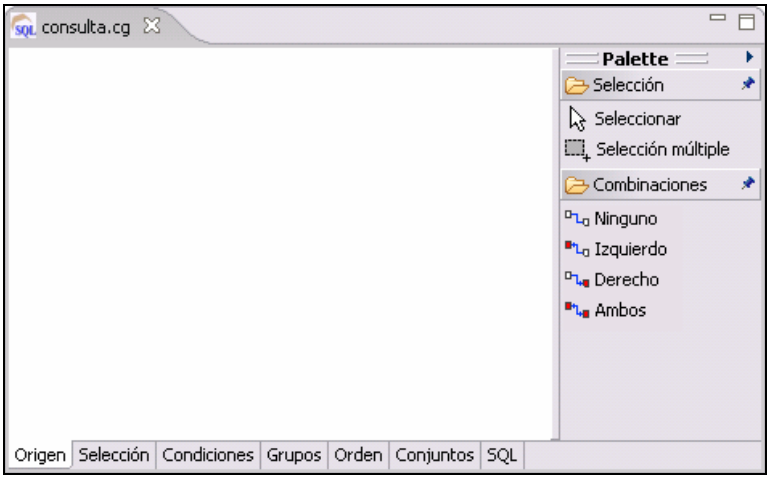


**Ilustración 56: Editor multipágina con la paleta de herramientas desplegada.**

Para la implementación de la paleta, la página "Origen" (EditorEnlace) tuvo que extender de *GraphicalEditorWithFlyoutPalette* para que soportara la inclusión de la paleta en el mismo. Además se agregó el siguiente código a la página.

```
protected PaletteRoot getPaletteRoot() //Obtiene una paleta para el editor.
{
     /*Obteniendo el archivo que se está cargando actualmente*/
     IEditorInput entrada = getEditorInput();
     IFile archivo = ((IFileEditorInput)entrada).getFile();

     /*Construyendo la paleta del editor.*/
     if (paleta == null)
        paleta = new Paleta(archivo.getProject().getName());

     return paleta;
}
```

**Ilustración 57: Fragmento de código incluido en el "EditorEnlace" para que soporte el uso de una paleta de herramientas.**

También se implemento la clase “Paleta.java” para definir el conjunto de herramientas que se quieren utilizar para el editor.

```
public Paleta (String proyecto)
   {
        …
        /* Construyen las herramientas de selección.*/
        clave = "Paleta.categoria.seleccion";
        categoria = new PaletteDrawer(ManejadorPropiedades.getValor(clave));
        herramientaSeleccion = new SelectionToolEntry();
        herramientaMarcar = new MarqueeToolEntry();

        clave = "iconos/selecciones.gif";
        categoria.setSmallIcon(GGC.getImageDescriptor(clave));
        clave = "iconos/selecciones.gif";
        categoria.setLargeIcon(GGC.getImageDescriptor(clave));
        clave = "Paleta.herramienta.seleccion.etiqueta";
        herramientaSeleccion.setLabel(ManejadorPropiedades.getValor(clave));
        clave = "Paleta.herramienta.seleccion.descripcion";
        herramientaSeleccion.setDescription(ManejadorPropiedades
                        .getValor(clave));
        categoria.add(herramientaSeleccion);
        …
        add(categoria);

        /* Construyen las herramientas de creación de conexión.*/
        categoria = new PaletteDrawer(ManejadorPropiedades.getValor(clave));
        clave = "Paleta.categoria.combinacion";
        clave = "iconos/combinaciones.gif";
        categoria.setSmallIcon(GGC.getImageDescriptor(clave));
        clave = "iconos/combinaciones.gif";
        categoria.setLargeIcon(GGC.getImageDescriptor(clave));
        …
        for (int i=0; i<tiposEnlace.size(); i++)
        {
                /* Obtiene informaciòn acerca del tipo del enlace soportado
                 * por la base de datos.*/
                tipoEnlace = (TipoEnlaceBD)tiposEnlace.get(i);
                nombre = tipoEnlace.getConector();
                descripcion = tipoEnlace.getDescripcion();

                /* Busca la imagen del enlace en la paleta de acuerdo a su tipo.*/
                if       (tipoEnlace.getConector().equalsIgnoreCase(TipoEnlaceBD.IZQUIERDO))
                        clave = "iconos/enlace_izquierdo.gif";
                else   if   (tipoEnlace.getConector().equalsIgnoreCase(TipoEnlaceBD.DERECHO))
                        clave = "iconos/enlace_derecho.gif";
                else if (tipoEnlace.getConector().equalsIgnoreCase(TipoEnlaceBD.COMPLETO))
                        clave = "iconos/enlace_completo.gif";
                else   if   (tipoEnlace.getConector().equalsIgnoreCase(TipoEnlaceBD.NORMAL))
                        clave = "iconos/enlace_normal.gif";

                /* Crea la entrada en la paleta.*/
                fabricaModelo = new FabricaModelo(tipoEnlace);
                fabricaModelo.setOperadorBD(operador);
                herramientaEnlace = new ConnectionCreationToolEntry(nombre
                                , descripcion
                                , fabricaModelo
                                , null
                                , null);
                herramientaEnlace.setLargeIcon(GGC.getImageDescriptor(clave));
                herramientaEnlace.setSmallIcon(GGC.getImageDescriptor(clave));
                categoria.add(herramientaEnlace);
        }

        add(categoria);
   }
}
```

**Ilustración 58: Clase que implementa la paleta de herramientas de la página "Origen" del editor.**

## Tarea 4: Creación de la representación visual del origen de datos (tabla, vista)

Se construyeron los componentes gráficos de las “vistas” y “tablas” de las bases de datos, haciendo uso de los API: draw2d para graficar las figuras y GEF para el manejo de eventos e interacciones sobre las figuras.

Las vistas y tablas constan de: un encabezado y un cuerpo. El encabezado contiene un objeto selección (checkbox) que será utilizado para seleccionar todas las columnas que conforman el origen de datos y el nombre o título de la figura para identificarla. El cuerpo contiene el conjunto de columnas que conforman la vista o la tabla. Este contiene un objeto selección (checkbox) por cada columna y el nombre de las mismas. Los objetos de selección serán utilizados para indicar si la columna seleccionada se mostrará en la consulta SQL; similar al QBE.

Las vistas y las tablas se diferencian en la interfaz por el color del encabezado y por una imagen descriptiva (ícono).

Cada figura consta de un modelo: el cual permite obtener los metadatos del mismo y, a su vez, la ubicación dentro del editor; un control: que indica la forma en que se construirá la interfaz que se mostrará al usuario; una interfaz para mostrar al usuario. A continuación se muestran las figuras resultantes:

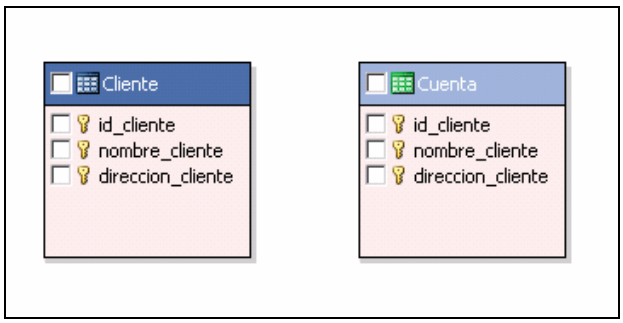


**Ilustración 59: Ejemplo de las interfaces de las figuras (vistas, tablas).**

Para la implementación de las figuras anteriormente mencionadas, se sigue el modelo MVC planteado por GEF. Además se utilizan las API’s de GEF y de Draw2d para definir los aspectos visuales y el comportamiento de cada una de las figuras.

### Vistas:

Los aspectos visuales fueron desarrollados haciendo uso del API de Draw2d y creando las siguientes clases:

| Clase | Descripción |
|---|---|
| VistaTabla | Contiene los aspectos gráficos para la vista de una tabla. |
| VistaEncabezado | Representa el encabezado que contiene el título (nombre) de la tabla o vista. |
| AreaColumnas | Área donde se dibujan las columnas de una tabla. |
| VistaColumna | Representación de una columna de una tabla o vista, a través de la cual se pueden realizar las selecciones de la consulta y los enlaces con otras tablas. |
| BordeTabla | Implementa el borde de las tablas. |

- **Modelo:**

Este contiene toda la información relacionada a las diferentes características de los componentes.

| Clase | Descripción |
|---|---|
| Consulta | Clase que contiene toda la información de la consulta que se está generando en la aplicación. |
| Tabla | Datos sobre una tabla o una vista (nombre, columnas). |
| Columna | Información sobre las columnas que conforman una tabla o vista. (Nombre, tipo de dato, clave primaria o foránea). |

- **Control:**

Los siguientes controles fueron desarrollados para poder manejar la interacción del usuario con las diferentes figuras.

| Clase | Descripción |
|---|---|
| ControlTabla | Clase que representa el control de un componente de tipo tabla. Encargada de manejar los cambios que pueden ocurrir con las tablas, es decir, cambio de posición, creación de una nueva tabla entre otros. |
| ControlColumna | Clase que representa el control de un componente de tipo columna. Tiene como finalidad verificar si una columna, de una tabla o vista fue seleccionada. |

## Tarea 5: Crear políticas y comandos que permitan manipular los orígenes de datos

Las políticas en GEF permiten asociar un conjunto de comandos que pueden ser aplicados a las diferentes figuras que se crean.

Las políticas que se definieron en esta tarea nos permiten crear una tabla para agregarla a la consulta, moverla de ubicación y cambiarle el tamaño en pantalla, eliminar una tabla de la consulta y realizar la selección de las columnas que se mostrarán en la consulta.

Para ello, se crearon las siguientes clases que implementan las políticas y los comandos:

- **Políticas:**

Estas definen el conjunto de comandos que se pueden llevar a cabo sobre las figuras.

| Clase | Descripción |
|---|---|
| PolíticaTabla | Política que permite eliminar tablas de la consulta gráfica. |
| PolíticaDistribucion | Esta clase permite agregar nuevas tablas al área de dibujo, mover las tablas existentes en el lienzo y cambiar los aspectos visuales de estas. Esta política se asigna al lienzo debido a que es en este donde se dibujarán las tablas. |

- **Comandos:**

Son los comandos que permiten ejecutar las diferentes funcionalidades sobre el editor, que afectan al origen de datos (tablas) y a la selección de una consulta.

| Clase | Descripción |
|---|---|
| ComandoCrearTabla | Comando que lleva a cabo la creación de una nueva tabla en el área de dibujo del editor. |
| ComandoEliminar | Comando que permite eliminar los componentes de una consulta (tablas). |
| ComandoAreaTabla | Se encarga de establecer la ubicación y el área de una tabla o vista. |
| ComandoSeleccionarColumna | Agrega la columna como parte de la selección o campos que mostrará la consulta. |
| ComandoSeleccionarTodas | Comando que permite agregar todas las columnas que conforman una tabla como parte de la selección o campos que mostrará la consulta. |
| ComandoEliminarSeleccion | Elimina la selección o campo de la consulta. |

## Tarea 6: Creación de la representación gráfica del enlace o relación entre los orígenes de datos

Una vez creadas la figura tabla, la cual representará un origen de datos en la consulta SQL, se creará una figura especial llamada enlace. Esta figura representará un enlace dentro de una combinación SQL entre dos tablas.

Un enlace será representado por una línea que conecta las columnas involucradas de las tablas que participan en la combinación, y, dependiendo del tipo de enlace SQL (FULL JOIN, LEFT JOIN o RIGHT JOIN) puede contener puntas de flecha para diferenciarlos.

Para permitir al usuario organizar mejor los enlaces y los orígenes en el editor, se crearon los “puntos de quiebre” de los enlaces. Estos permiten al usuario definir la ruta que seguirá la línea del enlace desde su origen hasta su destino.

La siguiente ilustración muestra un enlace entre dos tablas.

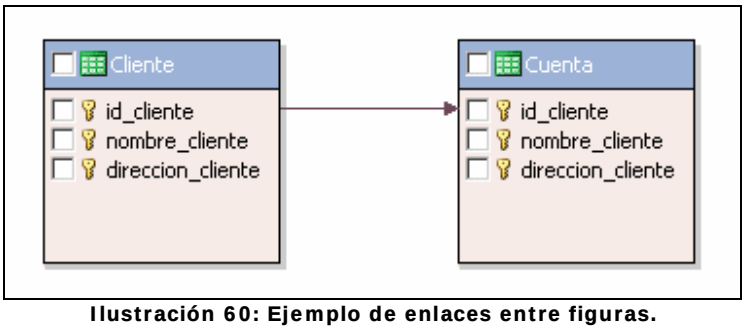


**Ilustración 60: Ejemplo de enlaces entre figuras.**

Al ser un enlace una figura especial, este también se implementará utilizando el modelo MVC planteado por GEF.

### Vistas:

Los aspectos visuales fueron desarrollados haciendo uso del API de Draw2d y creando las siguientes clases:

| Clase | Descripción |
|---|---|
| VistaEnlace | Esta clase representa la línea que se dibuja entre dos tablas del editor. |

### Modelo:

Contiene información sobre el tipo de enlace y las columnas entre las que se establece. También contiene los puntos de la ruta por la cual se dibujará la línea del enlace.

| Clase | Descripción |
|---|---|
| Enlace | Describe la información de un enlace entre dos orígenes de datos de una consulta SQL. |
| PuntoQuiebre | Contiene los aspectos de modelo a un punto de quiebre de un enlace. Define los puntos a través de los cuales se dibujará el enlace. |

### Control:

Para la implementación de los enlaces entre las tablas, se definieron y modificaron los siguientes controles.

| Clase | Descripción |
|---|---|
| ControlColumna (modificado) | Clase que representa el control de un componente de tipo columna. Se agregó la política enlace “PoliticaEnlace” para poder cambiar los enlaces. |
| ControlEnlace | Representa el control de un enlace. Utilizado para poder seleccionar los enlaces en el editor y eliminarlos. Permite controlar los puntos de quiebre. |

## Tarea 7: Crear políticas y comandos que permitan manipular las relaciones entre los orígenes de datos

De la misma forma que con las tablas y las vistas, a los enlaces, se les asoció un conjunto de políticas siguiendo el esquema de GEF para poder asociar un conjunto de comandos que se puedan aplicar sobre los enlaces.

Las políticas y comandos creados en esta tarea, nos permiten agregar un enlace entre las tablas, eliminar un enlace, cambiar el origen o destino de un enlace, seleccionar un enlace, crear los puntos de quiebre de un enlace para organizar las figuras del área de dibujo.

Las políticas y comandos implementadas en esta tarea son las siguientes:

### Políticas:

Las siguientes políticas fueron creadas para poder definir los comandos que se pueden ejecutar sobre los enlaces.

| Clase | Descripción |
|---|---|
| PolíticaEnlace | Política que define los comandos para crear o reconectar los enlaces. Utilizada para crear los enlaces entre las columnas de las tablas y para cambiar las columnas origen y destino una vez que se ha creado el enlace. |
| PolíticaSeleccion | Política que permite seleccionar los enlaces en el editor. |
| PolíticaEliminarEnlace | Esta clase provee la funcionalidad de eliminar los enlaces existentes en el editor cuando son seleccionados. |
| PolíticaPuntoQuiebre | Define el conjunto de comandos que controlan la creación de los puntos de quiebre y la movilidad de los mismos dentro del editor. |

### Comandos:

Los comandos definidos para ejecutar las diferentes funcionalidades sobre los enlaces son los siguientes.

| Clase | Descripción |
|---|---|
| ComandoEnlace | Permite crear o reconectar los enlaces. |
| ComandoEliminarEnlace | Permite eliminar los enlaces. |
| ComandoCrearPuntoQuiebre | Lleva a cabo la creación de un punto de quiebre. |
| ComandoMoverPuntoQuiebre | Cambia la ubicación de un punto de quiebre. |
| ComandoEliminarPuntoQuiebre | Permite eliminar los puntos de quiebre de los enlaces. |

## Tarea 8: Integración de la vista explorador al editor gráfico

Después de tener el editor enlace con todas las figuras y las diferentes funcionalidades, se creó la funcionalidad de arrastrar y soltar (drag and drop) desde la vista explorador; para agregar las tablas que se muestran en esta vista en el editor enlace.

Para que el editor contemplara esta funcionalidad se crearon las siguientes clases: ManejadorDrag, ManejadorDrop, TransferenciaTabla.

La clase ManejadorDrag, es la encargada de escuchar todos los eventos de arrastre (Drag) de elementos que ocurran sobre la vista explorador para poder exportarlos fuera de la vista hasta el editor enlace.

En la siguiente figura, se muestran los métodos más importantes de esta clase.

```
…
public class ManejadorDrag extends DragSourceAdapter
{
    …
    public void dragFinished(DragSourceEvent evento) {…}

        public void dragSetData(DragSourceEvent evento)
        {
           IStructuredSelection seleccion;     //Elementos que se están arrastrando.
           seleccion = (IStructuredSelection)observador.getSelection();

           if (TransferenciaTabla.getTransferencia().isSupportedType(evento.dataType))
           {
              List lista = seleccion.toList();
              TablaBD[] tablas = (TablaBD[])lista.toArray(new TablaBD[seleccion.size()]);
```

```
                evento.data = (TablaBD[])seleccion.toList().toArray(
                        new TablaBD[seleccion.size()]);
            }
        }

        public void dragStart(DragSourceEvent evento) {…}
    …
}
```

**Ilustración 61: Fragmento de la clase ManejadorDrag.java**

La clase ManejadorDrop, se utiliza para escuchar los eventos de soltar (Drop) que ocurren sobre el editor enlace.

A continuación se muestran los métodos relevantes de esta clase.

```
public class ManejadorDrop extends AbstractTransferDropTargetListener
{
        private EditorEnlaces editor;
        private FabricaModelo fabrica = new FabricaModelo("");

        protected void updateTargetRequest(){…}

        protected Request createTargetRequest()
        {
            CreateRequest peticion = new CreateRequest();
            peticion.setFactory(fabrica);
            return peticion;
        }

        protected void handleDragOver()
        {
            getCurrentEvent().detail = DND.DROP_COPY;
            super.handleDragOver();
        }

        protected void handleDrop()
        {
            TablaBD[] transferencia;
            transferencia = (TablaBD[])getCurrentEvent().data;
            fabrica.setTablaBD(transferencia[0]);
            super.handleDrop();
        }

        public boolean isEnabled(DropTargetEvent evento) {…}
}
```

**Ilustración 62: Fragmento de la clase ManejadorDrop.java**

Para poder transferir un objeto desde la vista explorador hasta el editor enlace se requiere de una clase que realice una traducción, es decir, se necesita convertir el objeto en un ente que, tanto la vista explorador como el editor puedan entender y poder obtener la información que se transmite; para ello se creó la clase TransferenciaTabla.

En la siguiente ilustración se muestran los métodos más importantes de esta clase.

```
…
protected void javaToNative(Object objeto, TransferData datosTransferencia)
{
        byte[] bytes = crearArregloByte((TablaBD[]) objeto);
        if (bytes != null)
        super.javaToNative(bytes, datosTransferencia);
}
…
protected Object nativeToJava(TransferData datosTransferencia)
```

```
{
        byte[] bytes = (byte[]) super.nativeToJava(datosTransferencia);
        return leerArregloBytes(bytes);
}
…
```

**Ilustración 63: Métodos de conversión de objeto de la clase TransferenciaTabla.java**

## Tarea 9: Creación del diálogo enlace

Esta tarea consiste en crear las funcionalidades e interfaces necesarias para permitir al usuario editar un enlace; para definir la dirección y otras propiedades del mismo.

A través del siguiente dialogo se permitirá al usuario: definir el operador de la condición de enlace y establecer si se quiere combinar las tablas a través de las columnas con el mismo nombre (combinación natural):

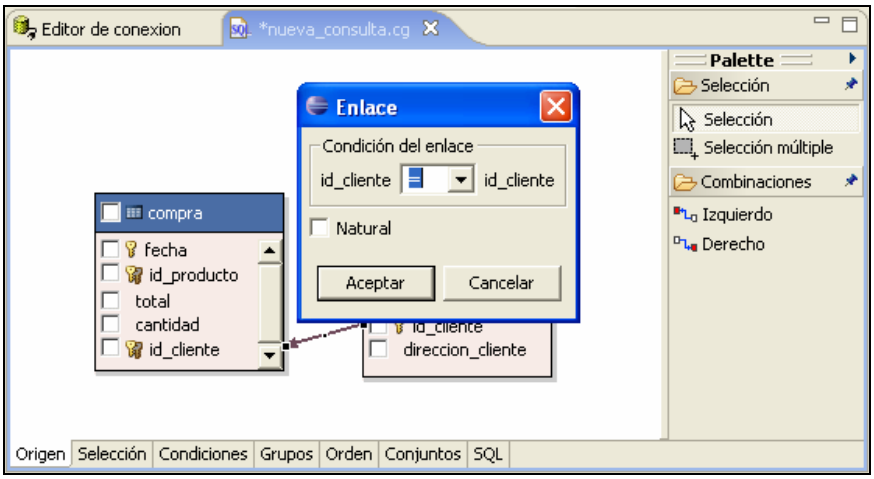


**Ilustración 64: Diálogo de edición de enlace**

## Tarea 10: Generación de la sintaxis SQL de la cláusula de origen

El objetivo de esta tarea es implementar los métodos que generaran la sintaxis SQL de la cláusula FROM. Se hace uso de las plantillas del sistema manejador de base de datos correspondiente (información previamente cargada al cargar el proyecto GGC), y de los orígenes de datos y selecciones de columnas asociados a este editor y almacenados en la consulta.

Se implementan los siguientes métodos de la clase ManejadorPlantillas:

- **armarClausulaOrigen**: Este método construye la cláusula Selección (SELECT) de la consulta a partir de las selecciones incluidas por el usuario.
- **getSintaxisOrigenTabla**: Construye la sintaxis de una tabla de la cláusula origen. Para esto determina como se representa una tabla o vista dentro de la cláusula, según la plantilla del sistema manejador de base de datos correspondiente.

- **getSintaxisSeleccionColumna**: Obtiene la sintaxis de una columna, desde las plantillas previamente cargadas.

- **getSintaxisTabla**: Obtiene la sintaxis de una tabla, desde las plantillas previamente cargadas. En este método se define como se muestra la información de la tabla (por ejemplo esquema.tabla o tabla).

El resultado del primer método es mostrado en la última página del editor multipáginas, el cual debe actualizarse cada vez que cambie la información del editor enlace, siguiendo el patrón observer.

### *3.6.4. Pruebas*

#### Prueba página origen

Consiste en verificar que la página diseñada para manipular los orígenes de datos muestre el área donde se dibujará la paleta de herramientas; Al abrir el editor consulta, la página origen si mostraba el área de la paleta.

#### Prueba de la paleta de herramientas

Verificar que se encuentren los elementos definidos para la paleta (seleccionar, selección múltiple, enlaces) de forma agrupada. Y que esta muestre todos los tipos de enlaces que soporta la base de datos asociada al proyecto.

1. Elementos y grupos: Al ejecutar el editor muestra los elementos definidos para la misma. Seleccionar y selección múltiple agrupados bajo la pestaña “Selección” y los tipos enlaces bajo la pestaña “Combinaciones”.

2. Tipos de enlace: Los diferentes tipos de enlace fueron mostrados satisfactoriamente cada vez que se abría el editor enlace con un sistema de BD diferente.

#### Prueba de fijación de la paleta

Esta prueba consiste en comprobar si el botón de fijación de la paleta evita que la misma no pueda ser visualizada por el usuario y que al desactivarlo esta pueda ser ocultada por el mismo.

1. Botón de fijación: Al activar el botón de fijación, efectivamente, la paleta no desaparece de la pantalla en ningún momento y al desactivar dicho botón el usuario puede ocultar la misma.

## Prueba de graficación

Consiste en el correcto despliegue gráfico de la figura que se generó (Nombre, columnas, Vista o tabla); Al abrir el editor se colocaron una vista y una tabla, las cuales mostraron su información correctamente, La vista mostró su nombre, las columnas que la integraban y su respectivo encabezado con su imagen. La tabla al igual que la vista, mostró todos sus componentes de forma satisfactoria.

## Prueba de selección de figuras

Verificar que efectivamente GEF permite seleccionar las figuras que se despliegan en el área de dibujo.

i. Seleccionar una figura: Una vez que las figuras son mostradas por pantalla, se procedió a seleccionar las figuras con el ratón; siendo satisfactorio el resultado ya que todas las figuras mostradas al hacerles clic con el puntero del ratón, les aparecen los puntos de selección.


**Ilustración 65: Imagen de una tabla seleccionada con los puntos de selección visibles.**

ii. Selección múltiple: De igual manera, se procedió a seleccionar un conjunto de figuras a través del puntero del ratón y las mismas fueron seleccionadas obteniendo así un resultado satisfactorio para esta prueba.

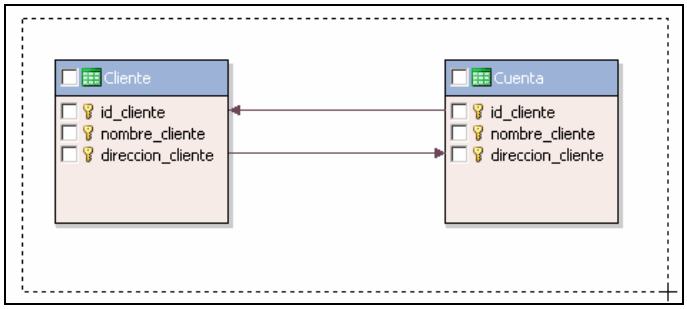


**Ilustración 66: Rectángulo de selección.**

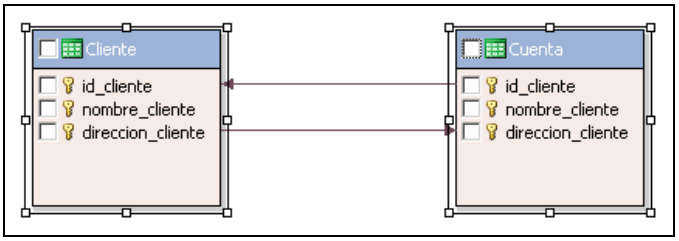


**Ilustración 67: Selección múltiple de tablas.**

## Prueba de desplazamiento de una figura

Esta prueba consiste en verificar que los elementos que han sido seleccionados en el área de dibujo, pueden ser desplazados de una ubicación a otra arrastrándolos con el ratón.

1. Desplazar la imagen dentro del área visible de la ventana: Al seleccionar una figura o un conjunto de figuras y moverlas sobre el área de dibujo, estas cambian de ubicación sin ningún problema.

2. Desplazar la figura fuera del área visible de la ventana: Al desplazar las figuras fuera del área de dibujo de la ventana, se activan las barras de desplazamiento vertical u horizontal según sea el caso, permitiendo de esta forma extender el área de dibujo que se muestra al usuario y poder visualizar un mayor número de figuras en la ventana del editor.

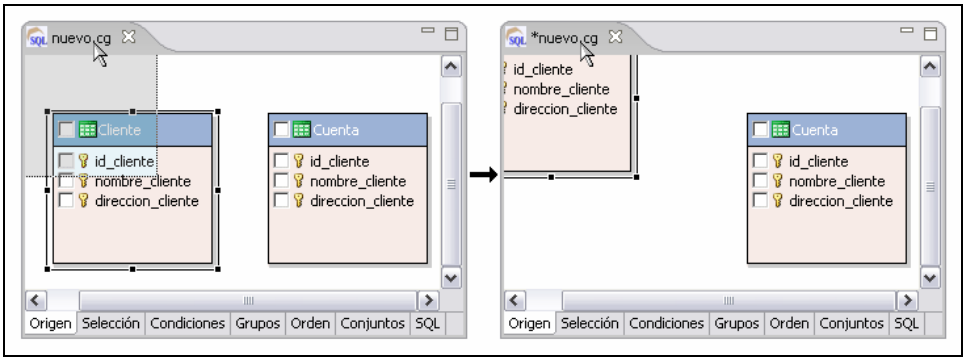


**Ilustración 68: Mover tablas fuera del área visible.**

## Prueba cambiar tamaño tabla

La prueba cambiar tamaño consiste en cambiar el tamaño del área que ocupan las diferentes tablas que se encuentran en el "*área de dibujo*" del editor; Esta prueba se realizó sobre todos los puntos de selección; cambiando el tamaño de las tablas en todas las direcciones y estableciendo diferentes tamaños para las mismas; obteniendo un resultado satisfactorio de esta prueba.

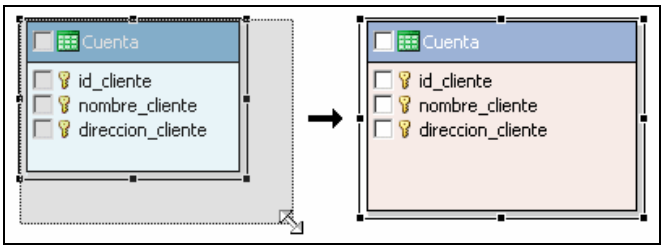


**Ilustración 69: Cambio de tamaño de una tabla.**

### Prueba eliminar tabla

Esta prueba consiste en eliminar una tabla que se haya agregado al área de dibujo del editor y a la consulta; Después de seleccionar una figura, esta se pudo eliminar desde los menús de la aplicación y también haciendo uso del teclado.

### Prueba de creación de enlaces

La prueba busca verificar que los enlaces que se crean se grafiquen correctamente, es decir, se muestre la línea en el editor y que esté dirigida correctamente desde la columna origen que se seleccionó hasta la columna destino seleccionada al crear el enlace; Se realizaron diferentes enlaces entre diferentes tablas y vistas, y todos estos enlaces se dibujaron correctamente en el editor.

### Prueba de reconexión de enlaces

Verificar si a los enlaces diferentes enlaces presentes en el editor, se les puede cambiar tanto de sus columnas orígenes como de sus columnas destinos.

1. Reconexión entre mismas tablas: Al reconectar los enlaces entre las mismas tablas o vistas sobre las cuales fueron creados, estos cambiaron satisfactoriamente sus orígenes, sus destinos y la ruta o dibujo del enlace.
2. Reconexión entre tablas diferentes: Al intentar cambiar el origen o el destino de un enlace con otra tabla sobre la cual él no estaba definido, la reconexión no se pudo llevar a cabo, siendo este comportamiento el esperado.

### Prueba puntos de quiebre

La prueba de los puntos de quiebre consiste en cambiar la ruta (“línea que representa el enlace”) del enlace estableciendo los puntos por los se quiere que pase el enlace. Para ello, se selecciona el enlace. Cuando aparecen los puntos de selección, también aparecen los posibles puntos de quiebre. Para cambiar la ruta, se coloca el puntero del ratón sobre este, y se mantiene oprimido el botón del ratón para arrastrarlo al lugar deseado.

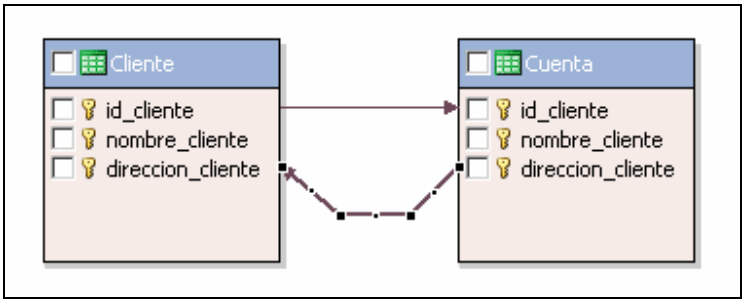


**Ilustración 70: Estableciendo una ruta para un enlace.**

Después de establecer diferentes rutas para diferentes enlaces se obtuvo el resultado esperado.

### Prueba agregar tabla desde la vista explorador

Esta prueba se consiste en agregar nuevas tablas desde la vista explorador hasta el editor enlace.

1. Drag and Drop: una vez que la vista explorador muestra las tablas que contiene la Base de Datos, se realizó un clic sobre la tabla o vista que se quería agregar al editor y, manteniendo el botón del ratón presionado, se arrastra la misma desde la vista explorador hasta el editor enlace. En el momento que el puntero del ratón se encontraba en la zona del editor donde se quería mostrar la tabla se soltaba el botón y se agregó la tabla o la vista. El resultado fue satisfactorio.
2. Doble clic: al realizar un doble clic en la vista explorador sobre la tabla que se quiere agregar en el editor, esta se agrega al editor de forma satisfactoria en la posición (0,0) el editor.

### Prueba validar objeto de la vista explorador

Consiste en validar que al hacer doble clic o arrastrar un objeto de la vista explorador, solo se agreguen al editor los objetos tablas de la misma.

1. Tablas: al transferir tablas al editor enlace, estas se transfieren satisfactoriamente.
2. Esquemas: al intentar transferir al editor enlace un esquema de la base de datos, este no se transfiere; siendo este resultado el correcto.
3. Columnas: Al igual que con los esquemas, si se intenta transferir una columna, esta no se agrega al editor; siendo este resultado satisfactorio.

### Prueba modificar enlace

Se debe verificar si la modificación sobre el enlace se lleva a cabo correctamente. Las pruebas para las posibles modificaciones son:

1. Modificación del operador: La condición del enlace tiene el operador seleccionado por el usuario en el diálogo de edición, construido para tal fin.
2. Modificación de la combinación: Si el usuario selecciona el CheckBox que indica que la combinación será natural, se deberán eliminar todos los enlaces existentes en el origen de datos y crear enlaces entre todas las columnas que tengan el mismo nombre en ambos orígenes.

### Prueba de la generación de sintaxis

Cada vez que se modifique la información del editor de enlace se debe generar la sintaxis SQL de la cláusula FROM. Esta nueva sintaxis generada debe mostrarse en la última página del editor de consultas.

### Prueba deshacer / rehacer

Esta prueba consiste en deshacer y rehacer algún cambio de una operación o acción que haya realizado el usuario en el editor.

Para realizar esta prueba, se procedió a realizar distintos tipos de operaciones sobre las figuras del editor (Mover, cambiar de tamaño, eliminar) y después de hacerlas, se ejecutó el comando deshacer (“Undo”) varias veces y se obtuvo el resultado deseado. Luego se ejecutó el comando rehacer (“Redo”) y se realizaron nuevamente todas las operaciones que había realizado el usuario; obteniendo un resultado satisfactorio de esta prueba.

## Síntesis

Esta iteración fue planificada para construir e integrar todos los componentes que permiten crear o generar la clausula origen (FROM) de la sintaxis SQL.

Según lo planificado, se construyó la página origen del editor multipáginas, además, se integró la vista explorador para permitir agregar las tablas.

Al finalizar esta iteración, la herramienta cuenta con: el editor enlace, la vista explorador integrada, la paleta de herramientas, el dialogo enlace para la modificación de los mismos y la generación de la sintaxis SQL de la cláusula de orígenes de datos; todos estos componentes fueron aprobados por el usuario.

Las historias de usuarios HU-12 y HU-19, se subdividen en otras historias de usuario que serán contempladas en las próximas iteraciones; por lo cual al terminar esta iteración solo se completó un porcentaje de estas.

La siguiente iteración estará enfocada en la creación del editor selecciones para poder construir la clausula selección (SELECT) de la sintaxis SQL.

# 3.7. Iteración IV

En esta iteración se construirán las funcionalidades que permiten la edición de la clausula SELECT de una consulta de forma gráfica.

Se trabajará sobre la página Selección del editor multipáginas; la cual permitirá al usuario indicar cuáles son los datos que se quieren recuperar de los orígenes agregados en la página de Enlaces.

## *3.7.1. Planificación*

Las historias de usuario correspondientes a esta iteración son:

**Tabla 25: Historias de usuario de la iteración IV**

| Id. | Descripción |
|---|---|
| HU-12 | La consulta debe ser generada de forma gráfica, permitiendo la edición de cada una de sus cláusulas. |
| HU-14 | Debe crearse una Tabla que muestre todos los campos (selecciones) que desplegará la consulta al ser ejecutada. |
| HU-19 | La consulta generada después de que el usuario edite cada una de las secciones de la consulta, debe ser mostrada en un campo de texto sobre el cual el usuario pueda seleccionar para poder copiar y pegar. |

**Tabla 26: Tareas de la iteración IV**

| Historia de usuario | Tarea |
|---|---|
| HU-14 | 1. Crear interfaz de la tabla de selecciones.<br>2. Agregar selección.<br>3. Eliminar selección.<br>4. Mover selección.<br>5. Guardar expresión. |
| HU-19 | 6. Generar sintaxis de la selección. |

### *3.7.2. Diseño*

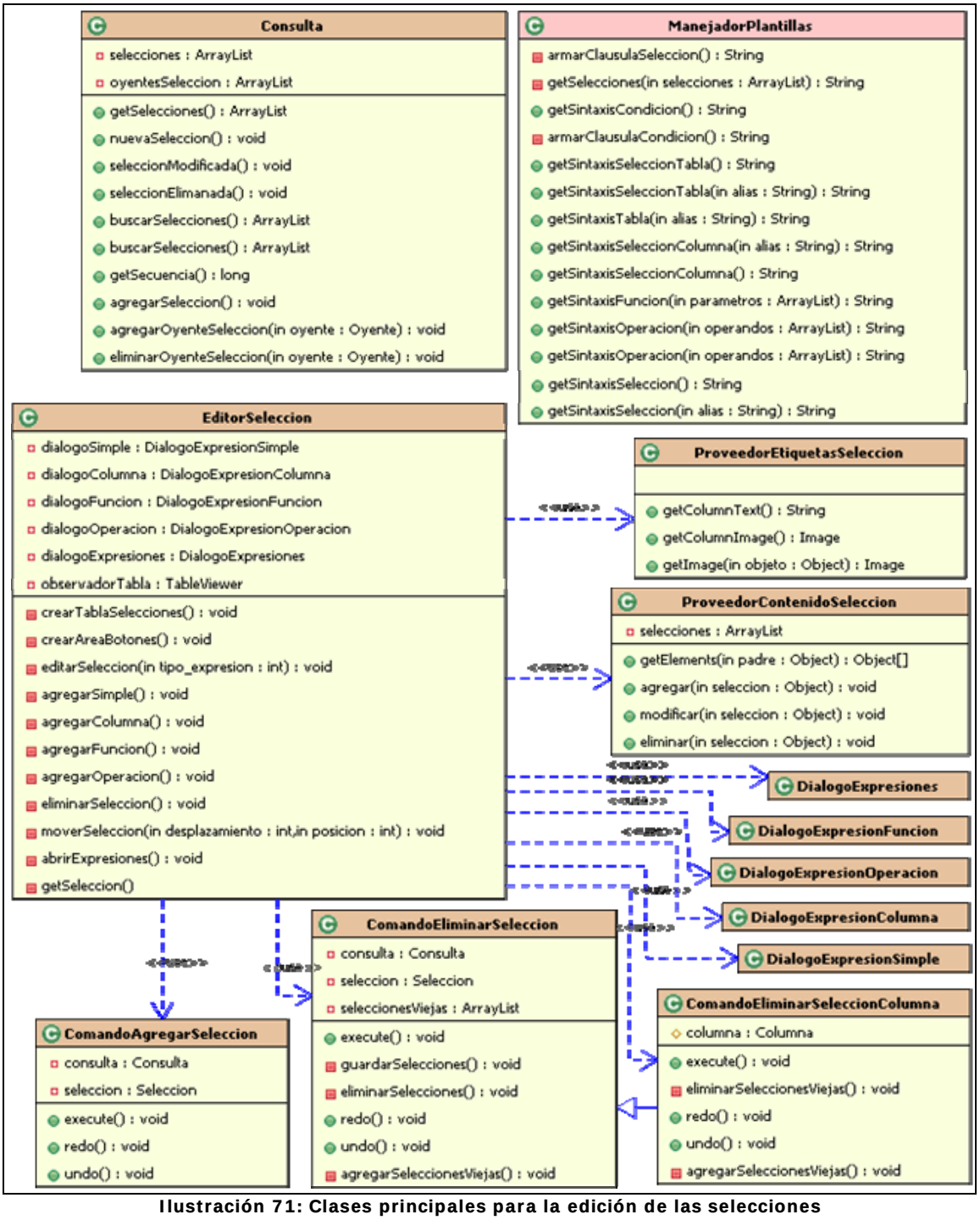


**Ilustración 71: Clases principales para la edición de las selecciones**

### *3.7.3. Implementación*

#### Tarea 1: Crear interfaz de la tabla de selecciones

Los campos de resultado que el usuario desea recuperar en la consulta, serán mostrados empleando una tabla de selecciones. En esta tabla e mostrarán tanto las selecciones realizadas sobre el editor de enlaces como las agregadas directamente en la tabla a construir.

Se crea un objeto de tipo TableViewer, que implementa los métodos para la creación de la tabla y hace uso de un proveedor de contenido (obtiene la información de las selecciones de la consulta), y un proveedor de etiquetas (indica como se muestran las selecciones en la tabla).

A continuación se muestra la interfaz construida:

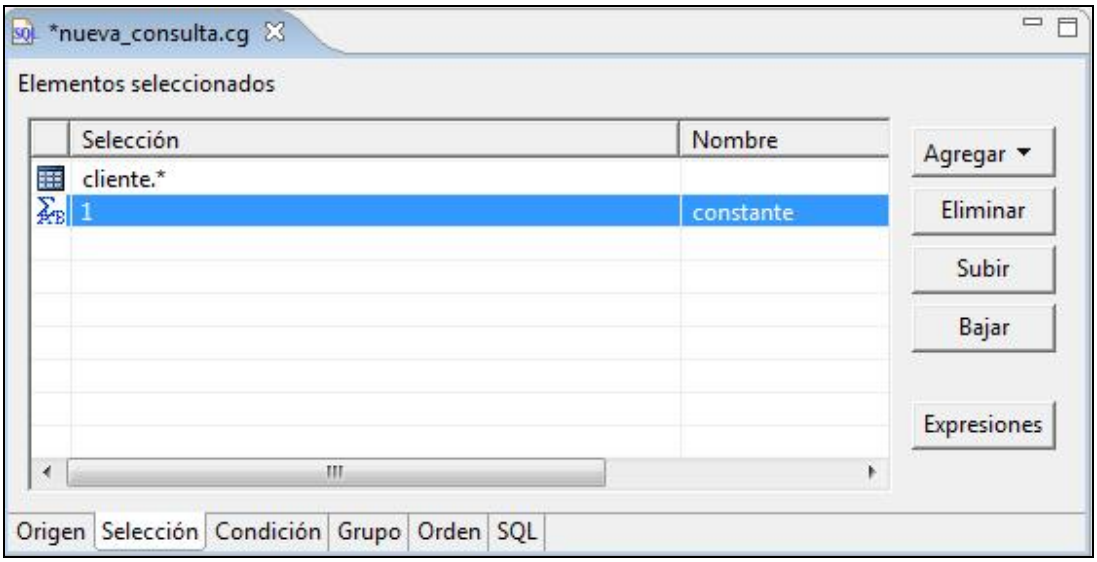


**Ilustración 72: Interfaz de la tabla de selecciones**

#### Tarea 2: Agregar selección

Esta tarea consiste en la implementación detrás de la adición de una nueva selección en la tabla construida en la tarea anterior, y posteriormente en la consulta. El usuario podrá agregar selecciones de diferentes tipos:

- **Simple**: Permite al usuario indicar en un cuadro de texto, el valor de una columna del resultado de la consulta.
- **Columna**: Permite al usuario seleccionar una columna como resultado de la consulta.
- **Función**: Permite al usuario utilizar una función (configurada para el sistema manejador de base de datos).
- **Operación**: Permite al usuario utilizar un operador (configurado para el sistema manejador de base de datos).

- ***Actualización de la clase Consulta***

Se modificará la clase Consulta para agregar el objeto que guardará las selecciones. Además se deben crear métodos que permitan notificar a las clases que dependen de los datos de la condición (por ejemplo el proveedor de contenido de la tabla), a través del patrón Observer, donde la consulta es el Subject y el resto de las clases involucradas son los Observers.

- ***Comandos***

La adición de la nueva condición se llevará cabo a través del comandoAgregarSeleccion.

El comando implementará el método que permite deshacer la acción realizada, en este caso, eliminar la selección agregada y dejar la tabla de selecciones en su estado anterior; y representa el participante ConcreteCommand del patrón Command. Se empleará la clase Command de GEF como AbstractCommand, el Invoker será el manejador de eventos del botón, y el editor de selección tendrá el rol de CommandManager.

## Tarea 3: Eliminar selección

Esta tarea consiste en la implementación detrás de la eliminación de la selección de consulta seleccionada en la tabla de selecciones.

- ***Actualización de la clase Consulta***

Se deben crear métodos que permitan notificar la eliminación, a las clases que dependen de los datos de la selección. Esto se hace través del patrón Observer, donde la consulta es el Subject y el resto de las clases involucradas son los Observers.

- ***Comandos***

La eliminación de una selección existente, se llevará cabo a través de los comandos ComandoEliminarSeleccion y su subclase ComandoEliminarSelecionColumna. Cada una de estas clases implementa el ConcreteCommand del patrón Command. Se empleará la clase Command de GEF como AbstractCommand, el Invoker será el manejador de eventos del botón, y el editor de selecciones tendrá el rol de CommandManager.

## Tarea 4: Mover selección

Esta tarea consiste, principalmente, en la implementación del comando que permitirá al usuario subir o bajar una seleccione (fila de la tabla de selecciones), con la finalidad de cambiar el orden de resultado de las consulta.

- ***Actualización de la clase Consulta***

Se deben crear métodos que permitan notificar la modificación de la posición antigua y la nueva posición de la selección, a las clases que dependen de los datos de la selección. Esto se hace través del patrón Observer, donde la consulta es el Subject y el resto de las clases involucradas son los Observers.

- ***Comandos***

Para mover una selección, se crea cabo El comando ComandoMoverSeleccion, el cual implementa el ConcreteCommand del patrón Command. Se emplea la clase Command de GEF como AbstractCommand, el Invoker será el manejador de eventos del botón, y el editor de selecciones tendrá el rol de CommandManager.

La clase que representa este comando no fue diseñada al comienzo de la iteración, porque surgió durante la etapa de implementación.

### Tarea 5: Guardar expresión

Todos los editores, exceptuando el editor de enlaces, pueden tener acceso a un diálogo de expresiones, en el cual se pueden guardar expresiones de la página actual que pudieran ser empleados posteriormente en otras páginas del editor multipáginas. A continuación se muestra la interfaz de usuario de este diálogo:


### Tarea 6: Generación de la sintaxis de la selección

El objetivo de esta tarea es implementar los métodos que generaran la sintaxis SQL de la cláusula SELECT. Se hace uso de las plantillas del sistema manejador de base de datos correspondiente (información previamente cargada al cargar el proyecto GGC), y de las selecciones al editor y almacenados en la consulta.

Entre los métodos creados en esta iteración para la clase ManejadorPlantillas, se encuentran los siguientes:

- **armarClausulaSeleccion**: Este método construye la cláusula Selección (SELECT) de la consulta a partir de las selecciones incluidas por el usuario.
- **getSintaxisSeleccion**: Obtiene la sintaxis SQL para una selección de la consulta.
- **getSintaxisFuncion**: Obtiene la sintaxis SQL de la invocación de una función del configurada para el sistema de base de datos.

Además se utilizan otros creados anteriormente, como **getSintaxisOrigenTabla y getSintaxisTabla.**

El resultado del primer método es mostrado en la última página del editor multipáginas, el cual debe actualizarse cada vez que cambie la información del editor enlace, siguiendo el patrón Observer.

### *3.7.4. Pruebas*

#### Agregar selección

Se verifica si la condición se lleva a cabo correctamente en los posibles casos:

- Agregar selección de columna.
- Agregar selección de tabla.
- Agregar selección simple.
- Agregar selección de función.
- Agregar selección de operación.

#### Deshacer agregar selección

Consiste en verificar si al agregar una selección, su comando realiza correctamente la eliminación de esta.

#### Mover selección

Se verifica que al subir o bajar una selección en la tabla, se lleve a cabo correctamente.

#### Deshacer mover selección

Se verifica que al subir o bajar una selección en la tabla, se lleve a cabo correctamente la función deshacer, la cual debe llevar la selección a su posición anterior en la tabla.

### Eliminar selección

Se verifica que la eliminación de una selección se lleve a cabo correctamente, una vez que el usuario confirma la acción.

### Deshacer eliminar selección

Consiste en verificar si al eliminar una selección, su comando realiza correctamente la creación de la selección eliminada en su posición correspondiente.

### Guardar expresión

Se debe verificar que al hacer click en el botón Guardar Expresión, se guarde y muestre correctamente la expresión seleccionada.

### Eliminar expresión

Se debe verificar que al hacer clic en el botón Eliminar Expresión, se elimine correctamente la expresión seleccionada en el diálogo de expresiones.

### Generación de sintaxis de la selección

Se verifica que la sintaxis de las selecciones se actualice correctamente en el editor SQL (última página del editor multipáginas).

## Síntesis

Esta iteración fue planificada para construir e integrar todos los componentes que permiten crear la clausula selección (SELECT), tanto a nivel gráfico como a nivel de la sintaxis SQL.

Según lo planificado, se construyó la página de selección del editor multipáginas, además, se integró la generación de la sintaxis SQL en la última página del editor.

Al finalizar esta iteración, la herramienta cuenta con: el editor selección, el dialogo de expresiones y la sintaxis SQL de las selecciones; todos estos componentes fueron aprobados por el usuario.

La siguiente iteración estará enfocada en la creación del editor condiciones para poder construir la clausula de condición (WHERE) de la sintaxis SQL.

# 3.8. Iteración V

El objetivo de esta iteración es permitir al usuario poder agregar a la consulta gráfica condiciones de filtrado sobre los datos que quiere recuperar.

Las condiciones serán manipuladas a través de un árbol, sobre el cual el usuario podrá agregar y eliminar condiciones en el orden que requiera.

El árbol estará contenido en la página de condiciones del editor de consultas (tercer editor del EditorMultipáginas).

## *3.8.1. Planificación*

En esta iteración se trabajará sobre las siguientes historias de usuario:

**Tabla 27: Historias de usuario de la iteración V**

| Id. | Descripción |
|---|---|
| HU-12 | La consulta debe ser generada de forma gráfica, permitiendo la edición de cada una de sus cláusulas. |
| HU-16 | Las condiciones o filtros a aplicar a una consulta deben mostrarse en un árbol para poder visualizar la jerarquía de los filtros. |
| HU-19 | La consulta generada después de que el usuario edite cada una de las secciones de la consulta, debe ser mostrada en un campo de texto sobre el cual el usuario pueda seleccionar para poder copiar y pegar. |

**Tabla 28: Tareas de la iteración V**

| Historia de usuario | Tarea |
|---|---|
| HU-16 | 1. Crear interfaz del árbol de condiciones.<br>2. Agregar condición.<br>3. Eliminar condición. |
| HU-19 | 4. Generar sintaxis de la condición. |

## *3.8.2. Diseño*

A través del siguiente diagrama se pueden conocer las clases principales que implementan la manipulación de condiciones de la consulta:

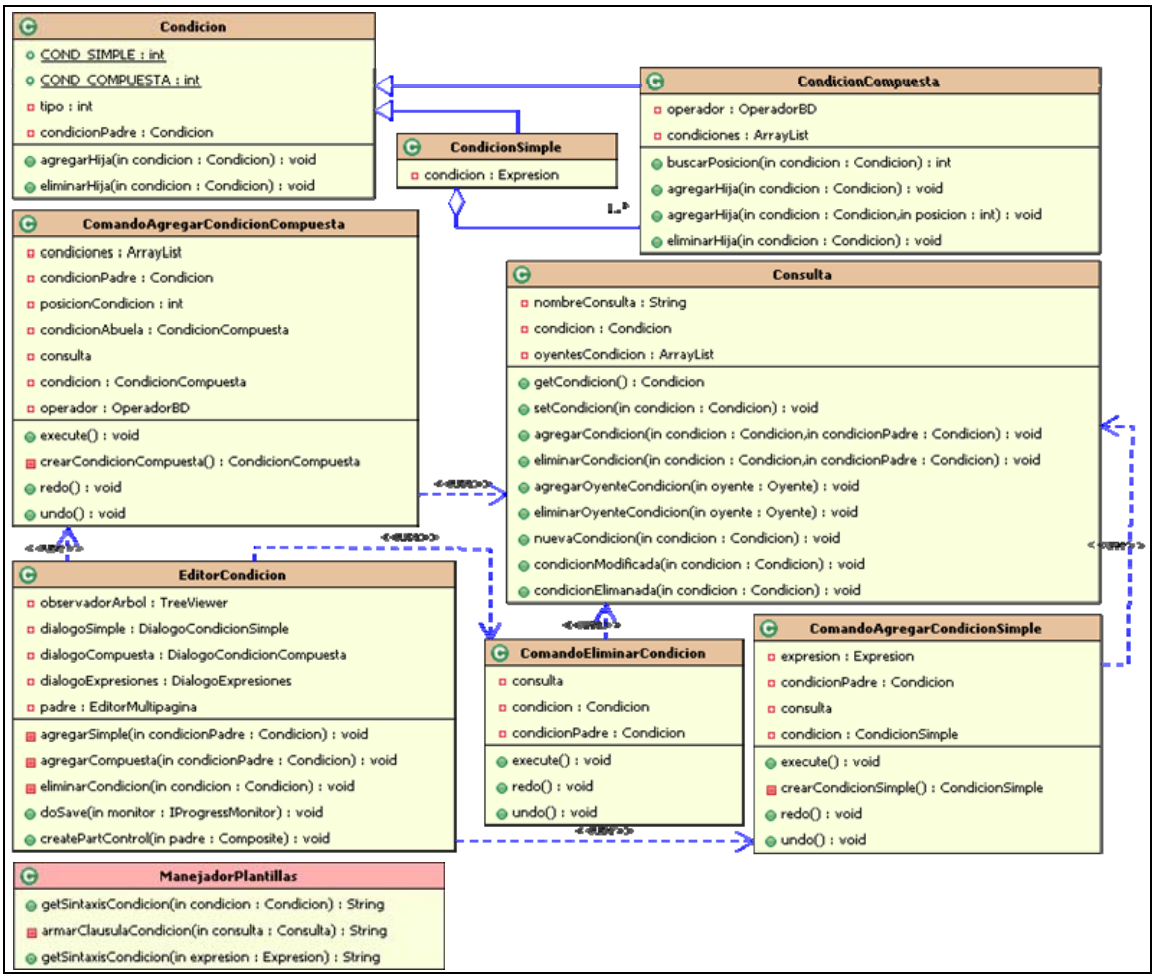


**Ilustración 73: Clases principales del módulo de condiciones**

- **Condicion**: Representa una condición de filtrado en la consulta; y tiene un conjunto de expresiones que permite almacenar los filtros indicados por el usuario.

- **CondicionSimple**: Es un subtipo de la condición y empleado para almacenar los datos de una condición de filtrado que solo tiene dos operandos simples y un operador. Un operando simple puede ser: una columna, un texto indicado por el usuario, entre otras expresiones simples.

- **Consulta**: Se modifica en esta iteración para incorporar el objeto que almacena las condiciones y los oyentes que (clases que dependen de los datos de las condiciones).

- **CondicionCompuesta**: Es un subtipo de la condición, que puede estar formada por un conjunto de condiciones simples.

- **ComandoAgregarCondicionSimple**: Esta clase lleva a cabo la adición de una nueva condición simple en el árbol de condiciones de la consulta.

- **ComandoAgregarCondicionCompuesta**: Esta clase lleva a cabo la adición de una nueva condición compuesta en el árbol de condiciones de la consulta.

- **ComandoEliminarCondicion**: Esta clase lleva a cabo la eliminación de una condición del árbol de condiciones de la consulta.

### *3.8.3. Implementación*

#### Tarea 1: Crear interfaz del árbol de condiciones

El objetivo de esta tarea es realizar un árbol que permitirá al usuario agregar condiciones a la consulta. Para esto se construye el componente de JFace TreeViewer, el cual es representado por la clase org.eclipse.jface.viewers.TreeViewer.

Se crea un objeto de tipo TreeViewer, que implementa los métodos para la creación del árbol y hace uso de un proveedor de contenido (contiene la información completa de los nodos del árbol), y un proveedor de etiquetas (indica cómo se muestran los nodos del árbol).

A continuación se muestra la vista construida:

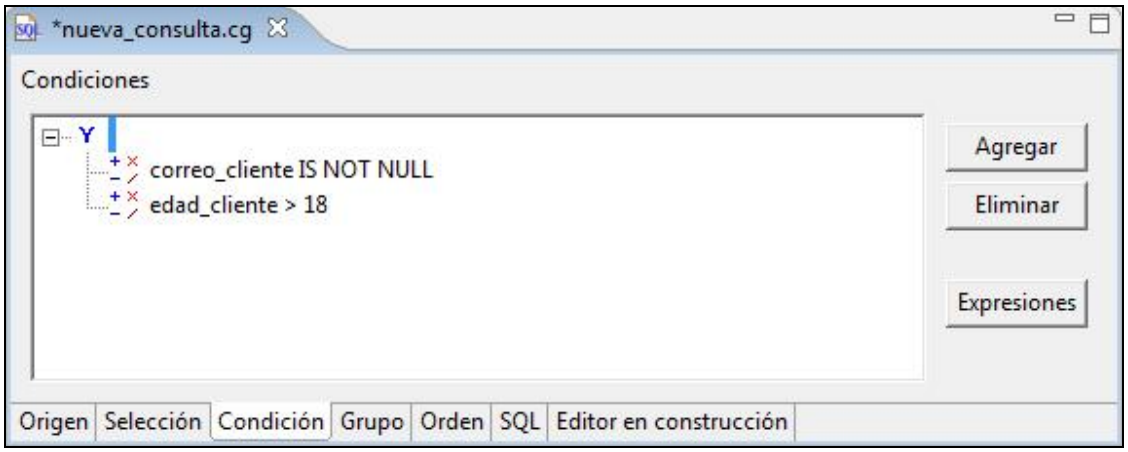


**Ilustración 74: Vista del árbol de condiciones**

#### Tarea 2: Agregar condición

Esta tarea consiste en la implementación detrás de la adición de una nueva condición en el árbol. El usuario podrá agregar condiciones de acuerdo a las siguientes reglas:

- Solo se agregará una sola condición a la vez.
- La condición se puede agregar sobre cualquier nodo del árbol.
- Si se agrega una condición sobre una condición simple, se debe crear en su lugar, una condición compuesta que contenga como hijas a esta y a la nueva condición.

- Si se agrega una condición sobre una condición compuesta, la condición agregada será simple.

- ***Actualización de la clase Consulta***

Se modificará la clase Consulta para agregar el objeto que guardará las condiciones. Además se deben crear métodos que permitan notificar a las clases que dependen de los datos de la condición (por ejemplo el proveedor de contenido del árbol), a través del patrón Observer, donde la consulta es el Subject y el resto de las clases involucradas son los Observers.

- ***Comandos***

La adición de la nueva condición se llevará cabo a través de dos comandos: ComandoAgregarCondicionSimple y ComandoAgregarCondicionCompuesta, para condiciones simples y compuestas respectivamente.

Cada uno de los comandos implementará el método que permite deshacer la acción realizada, en este caso, eliminar la condición agregada y dejar el árbol de condición en su estado anterior; y representan el participante ConcreteCommand del patrón Command. Se empleará la clase Command de GEF como AbstractCommand, el Invoker será el manejador de eventos del botón, y el editor de condiciones tendrá el rol de CommandManager.

## Tarea 3: Eliminar condición

Esta tarea consiste en la implementación detrás de la eliminación de la condición seleccionada del árbol de condiciones.

- ***Actualización de la clase Consulta***

Se crean métodos que permitan notificar a las clases que dependen de los datos de la condición, a través del patrón Observer, donde la consulta es el Subject y el resto de las clases involucradas son los Observers.

- ***Comandos***

La eliminación de una condición existente, se llevará cabo a través del comando ComandoEliminarCondicion. El cual implementará el método que permite deshacer la acción realiza, en este caso, agregar la condición eliminada y dejar el árbol de condición en su estado anterior. Esta clase implementa el ConcreteCommand del patrón Command. Se empleará la clase Command de GEF como AbstractCommand, el Invoker será el manejador de eventos del botón, y el editor de condiciones tendrá el rol de CommandManager.

### Tarea 4: Generar sintaxis de la condición

Esta tarea consiste en construir la sintaxis SQL correspondiente a la cláusula de condiciones de la consulta.

Para ello se emplean las plantillas del sistema manejador, previamente cargadas, y los datos de la estructura de las condiciones.

## *3.8.4. Pruebas*

### Agregar condición

Se verifica si la condición se lleva a cabo correctamente en los posibles casos:

- Agregar condición sobre la raíz.
- Agregar condición en cual nodo del árbol.

### Deshacer agregar condición

Consiste en verificar si al agregar una condición, su comando realiza correctamente la eliminación de la nueva condición.

### Eliminar condición

Se verifica que la eliminación de una condición se lleve a cabo correctamente, una vez que el usuario confirma la acción. Si es la única condición del árbol, entonces se debe establecer la condición de la consulta como nula.

### Deshacer eliminar condición

Consiste en verificar si al eliminar una condición, su comando realiza correctamente la creación de la condición eliminada en su posición correspondiente.

### Generación de sintaxis de la condición

Se verifica que la sintaxis de las condiciones se actualice correctamente en el editor SQL (última página del editor multipáginas).

## Síntesis

Al iteración finalizó luego que se construyeron e integraron todos los componentes que permiten crear la clausula de condición (WHERE), tanto a nivel gráfico como a nivel de la sintaxis SQL.

Según lo planificado, se construyó la página de condición del editor multipáginas. También, se integró la generación de la sintaxis SQL en la página de sintaxis SQL del editor de consultas.

El usuario aprobó el módulo planificado en esta iteración, el cual consistían en permitir al usuario editar gráficamente las condiciones de la consulta.

La siguiente iteración mostrará las etapas de construcción del módulo de grupo, el cual permitirá al usuario manejar las cláusulas GROUP BY y HAVING de la sintaxis SQL.

# 3.9. Iteración VI

En esta iteración se construyen los componentes que permitirán al usuario editar los grupos y condiciones de grupo de una consulta.

La manipulación de los grupos se llevará a cabo a través de una tabla, similar a la tabla construida para manipular las selecciones de la consulta. Mientras que para editar las condiciones de grupos se construye un árbol, similar al del árbol de condiciones construida en la iteración anterior.

El usuario podrá agregar eliminar y mover un grupo haciendo uso de la tabla de grupos que estará contenida en la página de grupos del editor de consultas (cuarto editor del EditorMultipáginas). Además podrá establecer condiciones sobre estos grupos haciendo uso del árbol de condiciones de grupo.

## *3.9.1. Planificación*

En esta iteración se trabajará sobre una sola historia de usuario:

**Tabla 29: Historias de usuario de la iteración VI**

| Id. | Descripción |
|---|---|
| HU-12 | La consulta debe ser generada de forma gráfica, permitiendo la edición de cada una de sus cláusulas. |
| HU-17 | Para la vista que mostrará los grupos se necesitará una tabla que muestre los campos a través de los cuales se agruparán los registros y un árbol donde se pueden visualizar las condiciones que se le pueden aplicar a los grupos. |
| HU-19 | La consulta generada después de que el usuario edite cada una de las secciones de la consulta, debe ser mostrada en un campo de texto sobre el cual el usuario pueda seleccionar para poder copiar y pegar. |

**Tabla 30: Tareas de la iteración V**

| Historia de usuario | Tarea |
|---|---|
| HU-17 | 1. Crear interfaz del editor de grupos.<br>2. Agregar grupo.<br>3. Eliminar grupo.<br>4. Agregar condición de grupo.<br>5. Eliminar condición de grupo.<br>6. Generar sintaxis de grupo. |
| HU-19 | 7. Generar sintaxis de la condiciones de grupo. |

## *3.9.2. Diseño*

El siguiente diagrama muestra las clases más importantes para implementar las funcionalidades de edición de grupos y condición de grupos. Los objetos que almacenan la información de las condiciones de grupo serán los mismos empleados para guardar las condiciones de filtrado de la consulta.

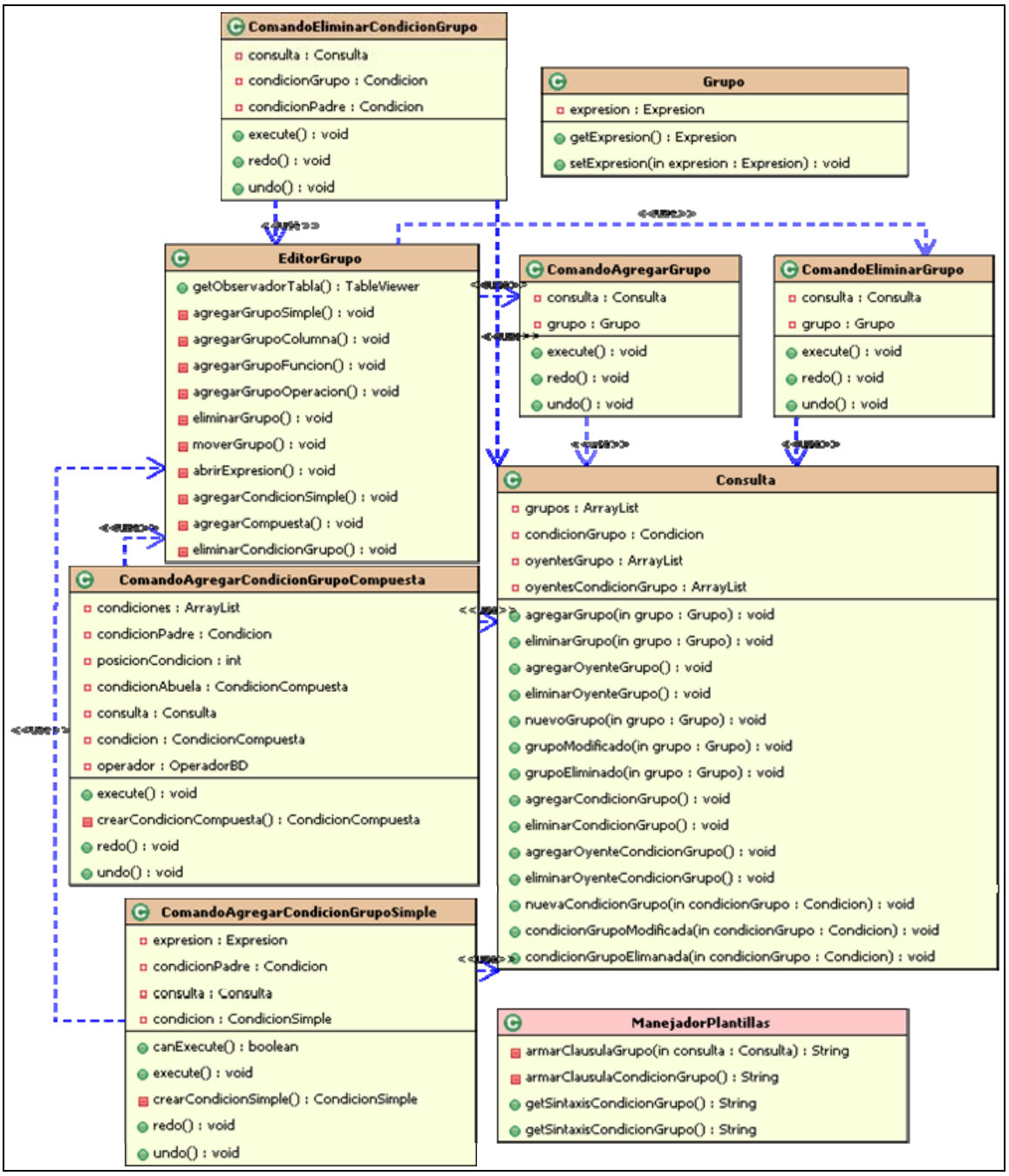


**Ilustración 75: Clases principales del módulo de grupos**

- **Grupo**: Representa una expresión de grupo en la consulta; y tiene una expresión que permite indicar el atributo por el cual se crea la expresión de grupo.

- **Consulta**: Se modifica en esta iteración para incorporar el objeto que almacena los grupos, condiciones de grupo y los oyentes.

- **ComandoAgregarGrupo**: Esta clase lleva a cabo la adición de una nueva expresión de grupo en la consulta.

- **ComandoEliminarGrupo**: Esta clase lleva a cabo la eliminación de un grupo desde la tabla de grupo.
- **ComandoAgregarCondicionGrupoCompuesta**: Esta clase lleva a cabo la adición de una nueva condición de grupo compuesta en el árbol de condiciones de grupo.
- **ComandoAgregarCondicionGrupoSimple**: Esta clase lleva a cabo la adición de una nueva condición de grupo simple (texto introducido directamente por el usuario) en el árbol de condiciones de grupo.
- **ComandoEliminarCondicionGrupo**: Esta clase lleva a cabo la eliminación de una condición del árbol de condiciones de grupo.

### *3.9.3. Implementación*

#### Tarea 1: Crear interfaz del editor de grupos

Se crean los dos objetos principales de este editor:

- Tablas de grupo: Se construye usando la clase TableViewer, el cual muestra los datos de los grupos de la consulta, empleando proveedor de contenido y proveedor de etiqueta para los grupos.
- Árbol condición de grupo: Se crea un objeto de tipo TreeViewer, que implementa los métodos para la creación del árbol y hace uso de un proveedor de contenido (contiene la información completa de los nodos del árbol) y un proveedor de etiquetas (indica cómo se muestran los nodos del árbol) para las condiciones de grupo.

  A continuación se muestra la vista construida:

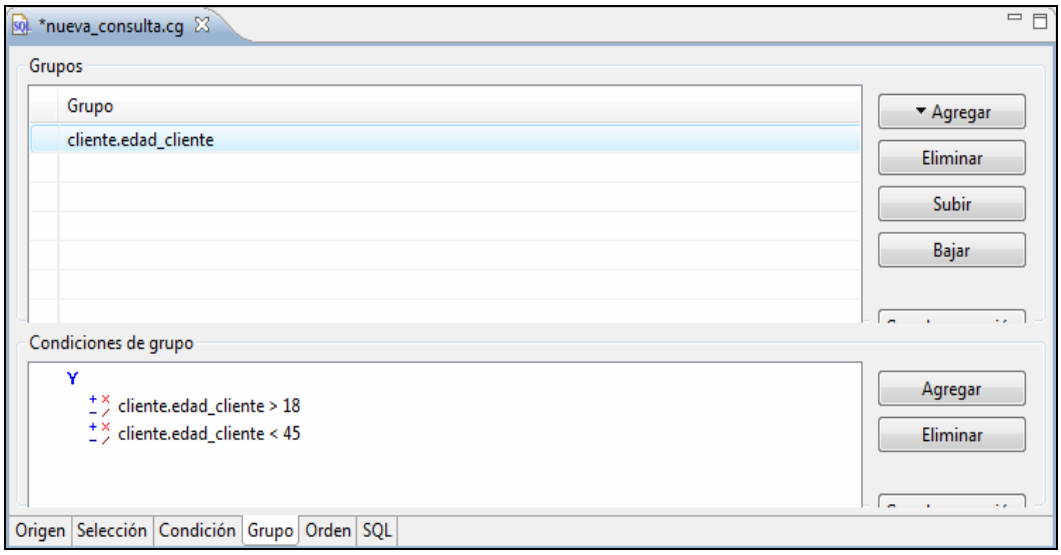


**Ilustración 76: Vista del editor de grupo**

## Tarea 2: Agregar grupo

Esta tarea tiene como objetivo implementar los métodos que llevarán a cabo la adicción de un nuevo grupo en la tabla de grupos. A continuación se describe el impacto significativo de implementar esta funcionalidad.

### *Actualización de la clase Consulta*

Se modificará la clase Consulta para agregar el objeto que guardará los grupos. Además se deben crear métodos que permitan notificar a las clases que dependen de los datos de la condición (por ejemplo el proveedor de contenido de la tabla de grupos), a través del patrón Observer, donde la consulta es el Subject y el resto de las clases involucradas son los Observers.

### *Comandos*

La agregación de un nuevo grupo se llevará a cabo a través del comando ComandoAgregarGrupo; quien además, implementará el método que permite deshacer la acción realizada, en este caso, eliminar el grupo agregado desde la tabla de grupos del editor de grupos.

Este nuevo comando será el participante ConcreteCommand del patrón Command. Se empleará la clase Command de GEF como AbstractCommand, el Invoker será el manejador de eventos del botón, y el editor de grupos tendrá el rol de CommandManager.

## Tarea 3: Eliminar grupo

La eliminación de un grupo desde la tabla de grupos, se llevará a cabo a través del patrón Command. Donde el comando construido implementa el ConcreteCommand. Se empleará la clase Command de GEF como AbstractCommand, el Invoker será el manejador de eventos del botón, y el editor de grupos tendrá el rol de CommandManager.

- ***Actualización de la clase Consulta***

Se crean métodos que permitan notificar a las clases que dependen de los datos de los grupos, a través del patrón Observer, donde la consulta es el Subject y el resto de las clases involucradas son los Observers.

- ***Comandos***

El ComandoEliminarGrupo, implementará la acción de eliminación y el método que permite deshacer la acción realiza; en este caso, agregar de nuevo a la tabla el grupo el grupo eliminado.

## Tarea 4: Agregar condición de grupo

La agregación de condiciones de grupo en la consulta se llevará a cabo a partir del árbol de condiciones de grupo del editor de grupos. Se toma en cuenta lo siguiente:

- Solo se agregará una sola condición de grupo a la vez.
- La condición de grupo se puede agregar sobre cualquier nodo del árbol.
- Si se agrega una condición de grupo sobre una condición simple, se debe crear en su lugar, una condición de grupo compuesta que contenga como hijas a esta y a la nueva condición.
- Si se agrega una condición de grupo sobre una condición compuesta, la condición agregada será simple.

- ***Actualización de la clase Consulta***

Se modificará la clase Consulta para agregar el objeto que guardará las condiciones de grupo. Además se deben crear métodos que permitan notificar a las clases que dependen de los datos de las condiciones de grupo (por ejemplo el proveedor de contenido del árbol), a través del patrón Observer, donde la consulta es el Subject y el resto de las clases involucradas son los Observers.

- ***Comandos***

La adición de la nueva condición de grupo se llevará cabo a través de dos comandos: ComandoAgregarCondicionGrupoSimple y ComandoAgregarCondicionGrupoCompuesta, para condiciones simples y compuestas respectivamente.

Cada uno de los comandos implementará el método que permite deshacer la acción realizada, en este caso, eliminar la condición de grupo agregada y dejar el árbol de condición de grupo en su estado anterior; y representan el participante ConcreteCommand del patrón Command. Se empleará la clase Command de GEF como AbstractCommand, el Invoker será el manejador de eventos del botón, y el editor de grupo tendrá el rol de CommandManager.

### Tarea 5: Eliminar condición de grupo

La eliminación de una condición de grupo desde el árbol de condiciones del editor de grupo, se llevará a cabo a través del patrón Command. Donde el comando construido implementa el ConcreteCommand. Se empleará la clase Command de GEF como AbstractCommand, el Invoker será el manejador de eventos del botón, y el editor de grupos tendrá el rol de CommandManager.

- ***Actualización de la clase Consulta***

Se crean métodos que permitan notificar a las clases que dependen de los datos de las condiciones de grupos, a través del patrón Observer, donde la consulta es el Subject y el resto de las clases involucradas son los Observers.

- ***Comandos***

El ComandoEliminarCondicionGrupo, implementará la acción de eliminación y el método que permite deshacer la acción realiza; en este caso, agregar de nuevo al árbol la condición de grupo eliminada.

### Tarea 6: Generar sintaxis de grupo

El objetivo de esta tarea es implementar los métodos de la clase ManejadorPlantillas que construyen la sintaxis SQL de la cláusula de grupo (GROUP BY).

### Tarea 7: Generar sintaxis de la condiciones de grupo

El objetivo de esta tarea es implementar los métodos de la clase ManejadorPlantillas que construyen la sintaxis SQL de la cláusula de grupo (GROUP BY).

## *3.9.4. Pruebas*

### Agregar grupo

Se verifica si agrega un nuevo grupo a la tabla de grupos del editor de grupos.

### Deshacer agregar grupo

Consiste en verificar si al agregar un grupo, su comando realiza correctamente la eliminación del mismo.

### Eliminar grupo

Se verifica que la eliminación de un grupo se lleve a cabo correctamente, una vez que el usuario confirma la acción. Si es el único grupo del árbol, entonces se debe establecer el grupo de la consulta como nulo.

### Deshacer eliminar grupo

Consiste en verificar si al eliminar un grupo, su comando realiza correctamente la creación del grupo eliminado en su posición correspondiente.

### Generación de sintaxis de grupo

Se verifica que la sintaxis de grupo se actualice correctamente en el editor SQL (última página del editor multipáginas).

### Agregar condición de grupo

Se verifica si la condición de grupo se lleva a cabo correctamente en los posibles casos:

- Agregar condición de grupo sobre la raíz.
- Agregar condición de grupo en cual nodo del árbol.

### Deshacer agregar condición de grupo

Consiste en verificar si al agregar una condición de grupo, su comando realiza correctamente la eliminación de la nueva condición.

### Eliminar condición de grupo

Se verifica que la eliminación de una condición de grupo se lleve a cabo correctamente, una vez que el usuario confirma la acción. Si es la única condición de grupo del árbol, entonces se debe establecer la condición de grupo de la consulta como nula.

### Deshacer eliminar condición de grupo

Consiste en verificar si al eliminar una condición de grupo, su comando realiza correctamente la creación de la condición eliminada en su posición correspondiente.

### Generación de sintaxis de la condición de grupo

Se verifica que la sintaxis de las condiciones de grupo se actualice correctamente en el editor SQL (última página del editor multipáginas).

## Síntesis

Durante esta iteración se realizó la construcción de las cláusulas de grupo de la consulta (GROUP BY y HAVING). Se construyeron e implementaron los métodos correspondientes a la edición gráfica de estas cláusulas; además, se implementaron los métodos de generación de su sintaxis SQL a partir de las plantillas asociadas al Sistema de Base de Datos y los datos de las condiciones de grupo asociados a la consulta.

El usuario aprobó los módulos contemplados en esta iteración: edición gráfica de los grupos de la consulta.

En la próxima iteración se llevará a cabo la construcción de las funcionalidades asociadas con el manejo de órdenes de la consulta, a través de la cláusula ORDER BY.

# 3.10. Iteración VII

Durante esta iteración se lleva a cabo la construcción de los componentes que permitirán al usuario editar los órdenes de la selección.

A través de la cláusula de orden de la consulta se podrá ordenar el resultado de la consulta. El usuario podrá establecer este orden a través de una tabla cuyas filas representan un campo de ordenamiento.

La tabla podrá ser vista en el editor de orden (sexto editor del EditorMultipáginas).

## *3.10.1. Planificación*

En esta iteración se trabajará sobre las siguientes historias de usuario:

**Tabla 31: Historias de usuario de la iteración VII**

| Id. | Descripción |
|---|---|
| HU-12 | La consulta debe ser generada de forma gráfica, permitiendo la edición de cada una de sus cláusulas. |
| HU-18 | Los resultados de una consulta pueden ser ordenados. Esto se realizará por medio de una tabla que contenga los campos por los cuales se ordenarán los resultados. |
| HU-19 | La consulta generada después de que el usuario edite cada una de las secciones de la consulta, debe ser mostrada en un campo de texto sobre el cual el usuario pueda seleccionar para poder copiar y pegar. |
| HU-20 | Crear el ejecutable del plug-in. |

**Tabla 32: Tareas de la iteración VII**

| Historia de usuario | Tarea |
|---|---|
| HU-18 | 1. Crear interfaz del editor de orden.<br>2. Agregar orden.<br>3. Eliminar orden.<br>4. Mover orden. |
| HU-19 | 5. Generar sintaxis del orden. |
| HU-20 | 6. Exportar la herramienta como una estructura de plug-in para Eclipse |

## *3.10.2. Diseño*

A continuación se muestra el diagrama con las clases principales que permiten el manejo de órdenes en la consulta:

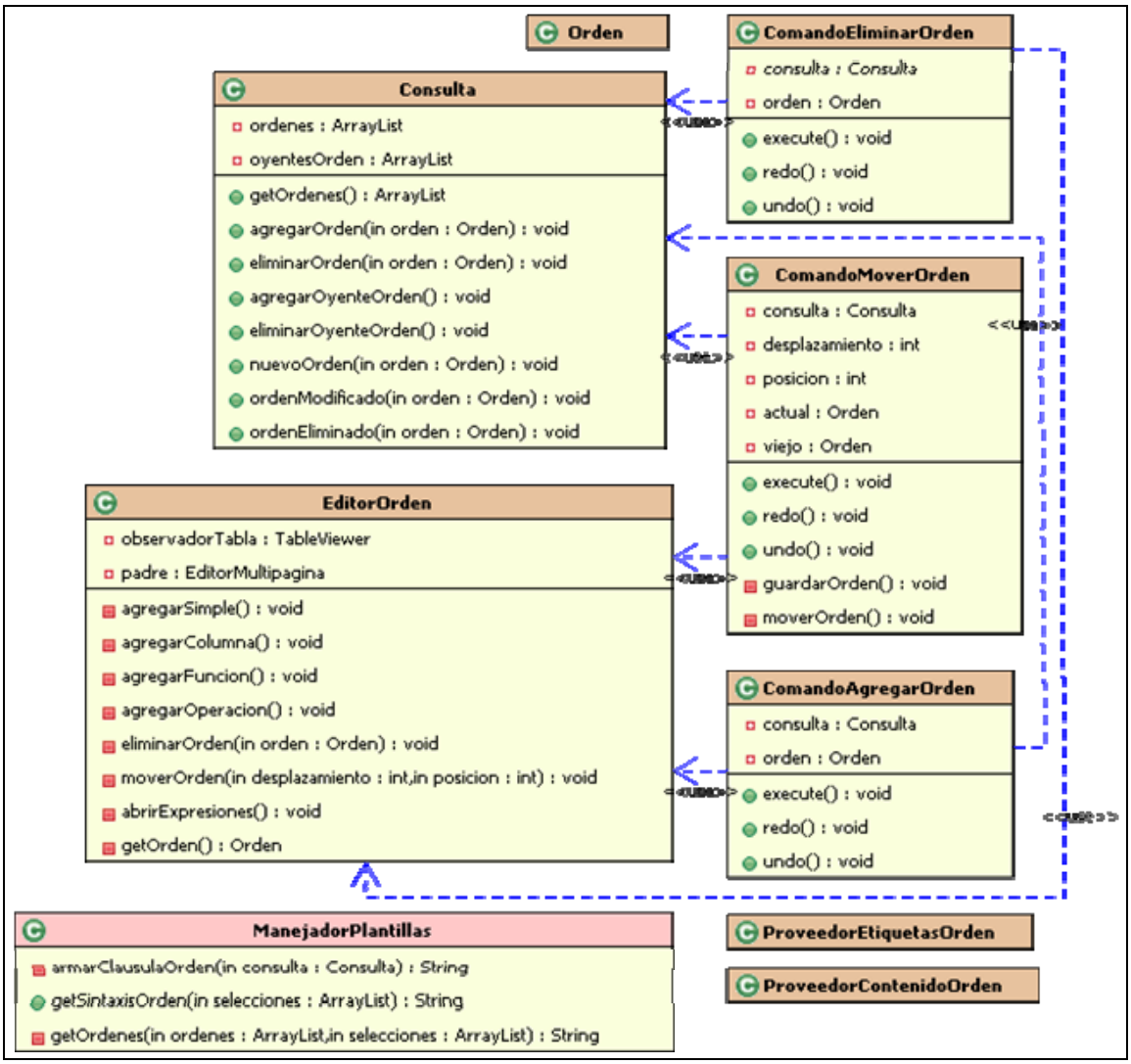


**Ilustración 77: Clases principales del módulo de orden**

- **Orden**: Representa un orden en la consulta. Tiene una expresión que permite almacenar la expresión por la cual se debe ordenar la consulta.
- **Consulta**: Se modifica en esta iteración para incorporar el objeto que almacena los y los oyentes que (clases que dependen de los datos de los órdenes).
- **ComandoAgregarOrden**: Esta clase lleva a cabo la adición de un nuevo orden en la consulta.
- **ComandoEliminarOrden**: Esta clase lleva a cabo la eliminación de un orden de la consulta.

### *3.10.3. Implementación*

#### Tarea 1: Crear interfaz del editor de orden

La interfaz a ser construidos en esta tarea permite establecer el orden del resultado de la consulta.

Los órdenes se manejarán a través de una tabla, de tipo TableViewer, que implementa los métodos para la creación de la tabla y hace uso de un proveedor de contenido (obtiene la información de los órdenes de la consulta), y un proveedor de etiquetas (indica como se muestran las órdenes en la tabla).

A continuación se muestra la interfaz construida:


**Ilustración 78: Interfaz de la tabla de órdenes**

#### Tarea 2: Agregar orden

Esta tarea consiste en la implementación detrás de la adición de orden en la tabla construida en la tarea anterior, y posteriormente en la consulta. Las modificaciones principales son:

- ***Actualización de la clase Consulta***

Se modificará la clase Consulta para agregar el objeto que guardará los órdenes. Además se deben crear métodos que permitan notificar a las clases que dependen de los datos los órdenes (por ejemplo el proveedor de contenido de la tabla), a través del patrón Observer, donde la consulta es el Subject y el resto de las clases involucradas son los Observers.

- ***Comandos***

La adición del nuevo orden se llevará cabo a través del comandoAgregarOrden.

El comando implementará el método que permite deshacer la acción realizada, en este caso, eliminar el orden agregado y dejar la tabla de órdenes en su estado anterior; y representa el participante ConcreteCommand del patrón Command. Se empleará la clase Command de GEF como AbstractCommand, el Invoker será el manejador de eventos del botón, y el editor de selección tendrá el rol de CommandManager.

## Tarea 3: Eliminar orden

Esta tarea consiste en la implementación detrás de la eliminación de un orden de la consulta seleccionada, desde la tabla de órdenes.

- ***Actualización de la clase Consulta***

Se deben crear métodos que permitan notificar la eliminación, a las clases que dependen de los datos del orden. Esto se hace través del patrón Observer, donde la consulta es el Subject y el resto de las clases involucradas son los Observers.

- ***Comandos***

La eliminación de un orden existente, se llevará cabo a través del comando ComandoEliminarOrden, cuya clase implementa el ConcreteCommand del patrón Command. Se empleará la clase Command de GEF como AbstractCommand, el Invoker será el manejador de eventos del botón, y el editor de orden tendrá el rol de CommandManager.

## Tarea 4: Mover orden

Esta tarea consiste, principalmente, en la implementación del comando que permitirá al usuario subir o bajar un orden (actualizar la fila de la tabla de órdenes), con la finalidad de establecer un nuevo orden en el resultado de la consulta.

- ***Actualización de la clase Consulta***

Se deben crear métodos que permitan notificar la modificación de la posición antigua y la nueva posición del orden, a las clases que dependen de los datos de los órdenes. Esto se hace través del patrón Observer, donde la consulta es el Subject y el resto de las clases involucradas son los Observers.

- ***Comandos***

Para mover un orden, se crea cabo El comando ComandoMoverOrden, el cual implementa el ConcreteCommand del patrón Command. Se emplea la clase Command de GEF como AbstractCommand, el Invoker será el manejador de eventos del botón, y el editor de selecciones tendrá el rol de CommandManager.

### Tarea 5: Generación de la sintaxis del orden

El objetivo de esta tarea es implementar los métodos que generaran la sintaxis SQL de la cláusula ORDER BY. Se hace uso de las plantillas del sistema manejador de base de datos correspondiente (información previamente cargada al cargar el proyecto GGC), y de los órdenes almacenados en la consulta.

Entre los métodos creados en esta iteración para la clase ManejadorPlantillas, se encuentran los siguientes:

- **armarClausulaOrden**: Este método construye la cláusula Orden (ORDER BY) de la consulta a partir de los órdenes establecidos por el usuario.
- **getSintaxisOrden**: Obtiene la sintaxis SQL para un orden de la consulta.

El resultado del primer método es mostrado en la última página del editor multipáginas, el cual debe actualizarse cada vez que cambie la información del editor de orden, siguiendo el patrón Observer.

### Tarea 6: Exportar la herramienta como una estructura de plug-in para Eclipse

Esta tarea consiste en empaquetar todos los recursos y clases que conforman la herramienta, en una unidad ejecutable.

Para llevarla a cabo se realizan los siguientes pasos a través del PDE de Eclipse:

Archivo → Exportar → Plug-in Development → Deplorable plug-ins and fragments:

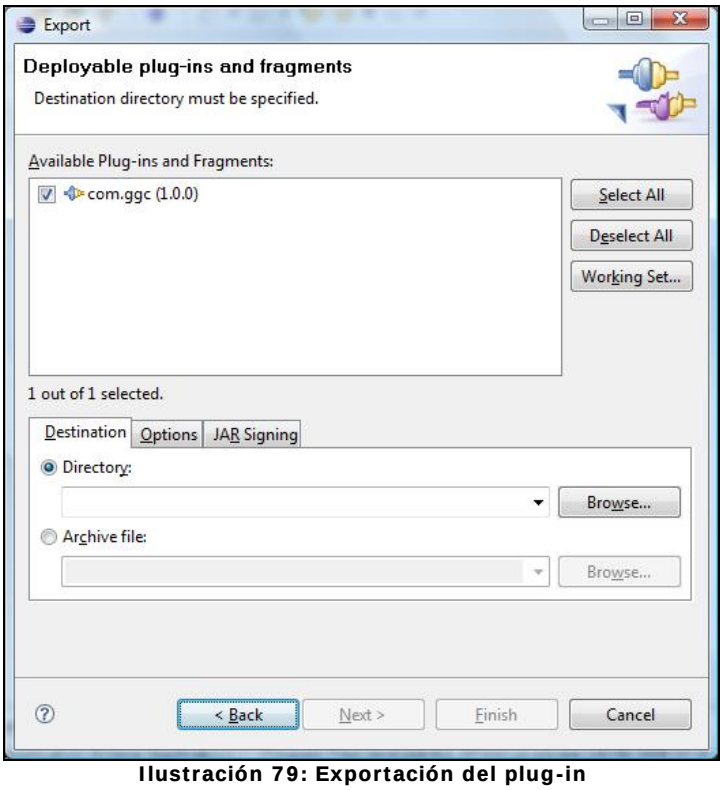


**Ilustración 79: Exportación del plug-in**

### *3.10.4. Pruebas*

#### Agregar orden

Se verifica que la agregación del orden se lleve a cabo correctamente en la tabla y en la consulta grafica.

#### Deshacer agregar orden

Consiste en verificar si al agregar un orden, su comando realiza correctamente la eliminación de este.

#### Mover orden

Se verifica que al subir o bajar un orden en la tabla, se lleve a cabo correctamente.

### Desplegar el plug-in generado en Eclipse

Se verifica que la herramienta se ejecute correctamente como un plug-in de Eclipse. Para ello se lleva a cabo lo siguiente:

1. Se copia el archivo com.ggc_1.0.0.jar en el directorio plug-ins del Eclipse donde se instalará el plug-in.
2. Se ejecuta el entorno de desarrollo integrado Eclipse.
3. Se accede a las funcionalidades de la herramienta.

### Deshacer mover orden

Se verifica que al subir o bajar un orden en la tabla, se lleve a cabo correctamente la función deshacer, la cual debe llevar el orden a su posición anterior en la tabla.

### Eliminar orden

Se verifica que la eliminación de un orden se lleve a cabo correctamente, una vez que el usuario confirma la acción.

### Deshacer eliminar orden

Consiste en verificar si al eliminar un orden, su comando realiza correctamente la creación del orden eliminado en su posición correspondiente.

### Generación de sintaxis de los órdenes

Se verifica que la sintaxis de los órdenes se actualice correctamente en el editor SQL (última página del editor multipáginas).

## Síntesis

Esta iteración fue planificada para construir e integrar todos los componentes que permiten crear la clausula de orden (ORDER BY), tanto a nivel gráfico como a nivel de la sintaxis SQL.

Según lo planificado, se construyó la página de orden del editor multipáginas, además, se integró la generación de la sintaxis SQL en la última página del editor.

Al finalizar esta iteración, la herramienta cuenta con todas las páginas del editor de consultas gráficas; todos los componentes han sido aprobados por el usuario.

Al realizar las pruebas integrales sobre la aplicación con todos los editores, se presentaron algunos errores, por lo cual se procede a refactorizar las funcionalidades que presentaron fallas.

## 3.11. Iteración VIII

Después de haber construido todas las funcionalidades y editores que permiten generar una consulta gráfica, se realizaron diferentes pruebas; y al generar diferentes tipos de consultas, utilizando todos los editores, se presentaron algunos errores.

En esta iteración se lleva a cabo la refactorización de las funcionalidades que presentaron fallas al intentar generar las consultas.

Estos errores generaron las siguientes historias de usuarios:

### *3.11.1. Planificación*

En esta iteración se trabajará sobre las siguientes historias de usuario:

**Tabla 33: Historias de usuario de la iteración VIII**

| Id. | Descripción |
|---|---|
| HU-21 | Las expresiones generadas por los diferentes diálogos, deben validar los tipos de datos de los diferentes componentes que las conforman. |
| HU-22 | Las operaciones soportadas por los diferentes manejadores de BD, deben trabajar con diferentes tipos de datos, es decir, deben manejar el polimorfismo de las operaciones. |
| HU-23 | Cuando se ejecuta una consulta y esta genera algún error de ejecución, el mismo se debe mostrar en pantalla para que el usuario pueda identificar el punto en que el error ocurre en la consulta. |
| HU-24 | Mejorar el tiempo de respuesta de la vista de tablas de la aplicación. |
| HU-25 | Cuando se define el origen de una consulta y se repite un origen de datos, es decir, aparece un mismo origen pero con un alias, el origen de datos no se genera correctamente. |
| HU-26 | La vista de tablas, debe mostrar las columnas de las tablas, ordenadas según fueron definidas por el usuario al crear las mismas. |
| HU-27 | Adaptar la aplicación a la última versión del IDE Eclipse. |

**Tabla 34: Tareas de la iteración VIII**

| Historia de usuario | Tarea |
|---|---|
| HU-21 | 1. Modificar archivo de características para soportar tipos de datos.<br>2. Modificar diálogos para validar tipos de datos. |
| HU-22 | 3. Agregar nuevos operadores para simular el polimorfismo. |
| HU-23 | 4. Crear diálogo de error de BD. |
| HU-24 | 5. Crear proxy para obtener el diccionario de datos de la BD.<br>6. Modificar puntos de acceso a datos del diccionario. |
| HU-25 | 7. Modificar métodos de comparación de alias. |
| HU-26 | 8. Cambiar estructura de almacenamiento de columnas.<br>9. Cambiar proveedor de contenido de la vista tablas. |
| HU-27 | 10. Modificar archivos de configuración del plugin. |

### *3.11.2. Implementación*

#### Tarea 1: Modificar archivo de características para soportar tipos de datos.

Para hacer que las expresiones puedan soportar tipos de datos, se modificaron los archivos de características de los diferentes manejadores de BD.

En la definición de las funciones, se agregó una etiqueta que indica el tipo de dato que retorna la función y a los parámetros de la misma, se agregó una etiqueta que indica el tipo de dato de los parámetros.

En la definición de las operaciones, también se agregó una etiqueta que indica el tipo de dato del resultado de la operación y además, se agregaron etiquetas para indicar la información de los operandos de la operación.

A continuación se muestra un fragmento del archivo de características.

```
…
<operadores-comparacion>
        <operador plantilla="1">
                <nombre>Es nulo</nombre>
                <nombre-bd>IS NULL</nombre-bd>
                <descripcion>Verifica si una expresi&oacute;n es nula</descripcion>
                <operando>
                        <nombre>Operando_1</nombre>
                        <tipo>Cadena</tipo>
                </operando>
                <retorno>valor-verdad</retorno>
        </operador>
        …
</operadores-comparacion>
…
<funciones-tiempo-fecha>
        <funcion plantilla="16">
                <nombre>Suma</nombre>
                <nombre-bd>ADDTIME</nombre-bd>
                <parametro>
                        <nombre>marca de tiempo</nombre>
                        <tipo>tiempo-fecha</tipo>
                </parametro>
                <parametro>
                        <nombre>d&iacute;s</nombre>
                        <tipo>Num&eacute;rico</tipo>
                </parametro>
                …
                <descripcion></descripcion>
                <retorno>tiempo-fecha</retorno>
        </funcion>
</funciones-tiempo-fecha>
…
```

**Ilustración 80: Archivo características**

#### Tarea 2: Modificar diálogos para validar tipos de datos.

Para hacer que la aplicación valide los tipos de datos de las funciones y las operaciones que el sistema de BD soporta, se agregó un método en los diálogos función y operación respectivamente.

El método que se agregó en ambos diálogos es parámetroValido(). Este método se encarga de verificar el tipo de dato que devuelve una expresión y compararlo con el tipo de dato del parámetro o operando que se está analizando.

En la siguiente ilustración se muestra un fragmento de esté método.

```
…
case Expresion.EXP_COLUMNA:
        ColumnaBD columna = ((ExpresionColumna)expresion).getColumna()
                                        .getColumnaBD();
        tipo = TiposSQL.obtenerTipo(columna.getTipo());
        if (tipo.equalsIgnoreCase("Cadena") && columna.getTamano() == 1)
                tipo = "Caracter";
        break;
case Expresion.EXP_FUNCION:
        FuncionBD funcion = ((ExpresionFuncion) expresion).getFuncion();
        tipo = funcion.getRetorno();
        break;
case Expresion.EXP_OPERACION:
        OperadorBD operador = ((ExpresionOperacion) expresion)
                                        .getOperador();
        tipo = operador.getRetorno();
        break;
…
```

**Ilustración 81: Método parametroValido() de las clases DialogoExpresionFuncion y DialogoExpresionOperacion.**

## Tarea 3: Agregar nuevos operadores para simular el polimorfismo.

Para poder simular el polimorfismo de las operaciones que soportan los diferentes sistemas de BD, se agregaron nuevas características en el archivo de características.

Se agregaron las mismas operaciones pero con diferentes operandos o con diferentes tipos de datos para los operandos.

Ahora, se muestra un ejemplo del polimorfismo de operaciones en el archivo de características.

```
…
<operador plantilla="3">
        <nombre>Igual</nombre>
        <nombre-bd>=</nombre-bd>
        <descripcion>Comprueba la igualdad de dos expresiones</descripcion>
        <operando>
                <nombre>Operando_1</nombre>
                <tipo>Num&eacute;rico</tipo>
        </operando>
        <operando>
                <nombre>Operando_2</nombre>
                <tipo>Num&eacute;rico</tipo>
        </operando>
        <retorno>valor-verdad</retorno>
</operador>
<operador plantilla="3">
        <nombre>Igual</nombre>
        <nombre-bd>=</nombre-bd>
        <descripcion>Comprueba la igualdad de dos expresiones</descripcion>
```

```
        <operando>
                <nombre>Operando_1</nombre>
                <tipo>Num&eacute;rico</tipo>
        </operando>
        <operando>
                <nombre>Operando_2</nombre>
                <tipo>Cadena</tipo>
        </operando>
        <retorno>valor-verdad</retorno>
</operador>
…
```

**Ilustración 82: Polimorfismo de operaciones.**

## Tarea 4: Crear diálogo de error de BD.

Cuando se presentaba un error durante la ejecución de la BD, se mostraba un diálogo que indicaba que ocurrió un error.

Esta tarea consiste en crear un diálogo de error que muestre el error que se genera al ejecutar una consulta. El error es obtenido a través del mensaje que genera la excepción que se captura al ejecutar la consulta.

Para crear el diálogo se crea la clase DialogoErrorBD que extiende de la clase Dialog de SWT. En este diálogo se agrega un cuadro de texto que mostrará el mensaje de error del sistema de BD.

A continuación se muestra la interfaz que fue generada en esta tarea.

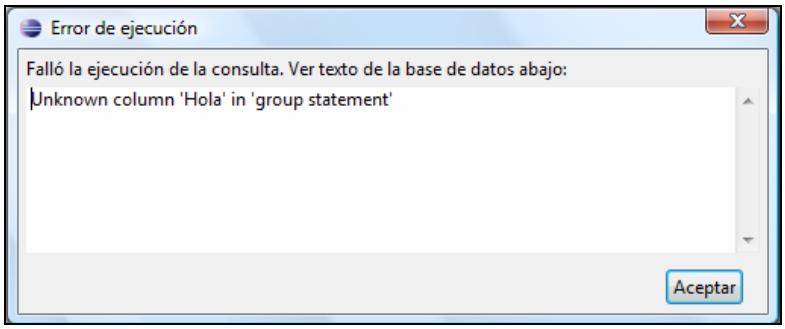


**Ilustración 83: Diálogo de Error de la BD.**

## Tarea 5: Crear proxy para obtener el diccionario de datos de la BD.

Esta tarea consiste en hacer que la información del diccionario de datos, se carga bajo demanda. Esto mejora la respuesta al usuario ya que no se carga toda la información de la BD al momento de acceder a la BD, sino que se carga según lo requiera el usuario.

Esto se logra implementando el patrón proxy, el cual nos permite cargar la información cuando esta es solicitada.

### Tarea 6: Modificar puntos de acceso a datos del diccionario.

Como el diccionario fue cambiado ya que ahora el mismo se carga bajo demanda, se modificaron los puntos del código en que se utiliza la información del diccionario.

Se modifica la clase ProveedorContenidoTablas ya que la misma es la que carga la vista de tablas de la aplicación.

La clase VistaExplorador fue cambiada ya que en la misma existe un manejador que permite agregar una tabla en la consulta al hacer doble clic y se requiere verificar si ya la información fue cargada para que no agregue tablas vacías a la consulta.

Y por último, se cambió la clase Transferencia tabla por la misma razón que se modificó la clase VistaExplorador. Ya que esta clase es la que realiza la transferencia de la información de la tabla cuando las tablas son arrastradas hacia el editor enlace.

### Tarea 7: Modificar métodos de comparación de alias

Esta tarea se lleva a cabo ya que cuando se generan los orígenes de una consulta; y una tabla o vista contiene un alias, la generación de la sentencia falla por error al comparar los alias de las tablas o vistas.

Se verificaron los métodos que comparan las tablas o vistas. Y al verificar estos métodos se detectó que no estaban funcionando correctamente ya que podían comparar el nombre de la tabla con el alias.

Se modificaron estos métodos para que en el caso de que las tablas tuviesen un alias, estas se compararán a través del mismo y si no, se compararan a través del nombre.

### Tarea 8: Cambiar estructura de almacenamiento de columnas.

Esta tarea consiste en cambiar el orden en que se muestran las columnas de las tablas tanto en la vista de tablas como en la vista enlace.

El usuario indica que las columnas de las tablas deben ser mostradas ordenadas como el usuario creó las mismas. Esto porque en la mayoría de los casos, los usuarios de BD crean las claves de las tablas primero que las columnas que contienen la información.

Para esto, se cambió la estructura que almacenaba las columnas. Se cambió de un HashMap a un ArrayList, este último, permite almacenar las columnas en el orden en que son leídas del diccionario de datos. Obteniendo así el resultado deseado por el usuario.

### Tarea 9: Cambiar proveedor de contenido de la vista tablas.

Una vez que fue cambiada la estructura que almacenaba las columnas, fue necesario, cambiar el proveedor de contenido de la vista de tablas para que pudiese manejar la nueva estructura (ArrayList).

A continuación se muestra la vista tabla con las columnas de las tablas ordenadas:

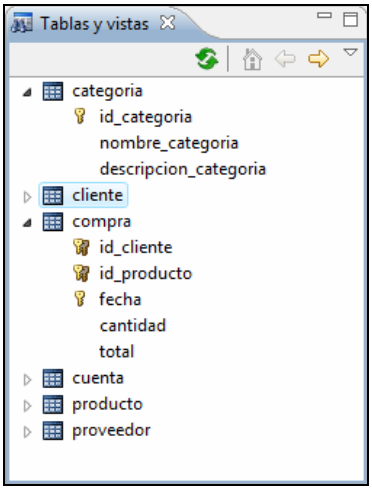


**Ilustración 84: Vista tabla con las columnas ordenadas.**

### Tarea 10: Modificar archivos de configuración del plugin.

Debido a las exigencias del usuario, hay que hacer una actualización al plug-in para que se pueda instalar en la última versión de eclipse (Eclipse 3.3.2). Después de estudiar las diferencias entre las versiones de los IDE, se observó que estos eran compatibles; pero en la última versión eran más estrictos con los archivos de configuración y estos habían cambiado el estándar de internacionalización del plug-in.

Las modificaciones que se realizaron en los archivos de configuración, fueron:

Se eliminó la redundancia entre los archivos, es decir, en la nueva versión, no se puede tener definida una variable en 2 archivos diferentes, porque esto genera un error al desplegar el plug-in en el entorno de desarrollo.

Y se definió la variable “Bundle-Localization” en el archivo MANIFEST.MF para indicar la ruta donde se encuentra el archivo de internacionalización del plug-in.

## *3.11.3. Pruebas*

Se realizaron nuevamente las pruebas de integración que se ejecutaron cuando se terminaron todos los artefactos del plug-in, es decir, se realizaron diferentes consultas completas y se analizaron las sintaxis de las mismas. Cada una de las consultas que se realizaron, fueron ejecutadas para probar la ejecución de las mismas desde el plug-in.

Una vez realizadas estas pruebas, el usuario dio su aceptación final; dando por terminado el producto.

## Síntesis

Esta iteración nace para corregir todos los errores o modificaciones solicitadas por el usuario final.

Esta última iteración fue una iteración de Refactorización para realizar los últimos ajustes de la aplicación.

En ella se corrigieron los errores que se generaban debido a la última integración de los diferentes editores o funcionalidades y se modificaron todas las características indicadas por el usuario final.

Al finalizar esta iteración, se cuenta con la aprobación del usuario para liberar el producto, ya que el mismo cuenta con todas las características solicitadas al inicio del proyecto. Dando por terminado el producto.

# Conclusiones

Este Trabajo Especial de Grado consistió en la implementación de un Plug-in para el entorno de desarrollo Eclipse. El Plug-in fue construido como una herramienta que permite la creación de una consulta SQL de forma gráfica, facilitando así el uso de plug-in a desarrolladores inexpertos en la sintaxis SQL.

La construcción de la herramienta se llevó a cabo siguiendo un proceso desarrollo basado en los conceptos de programación ágil. Se empleó Programación Extrema como proceso de desarrollo base, se le hicieron ajustes y se incorporaron principios y prácticas de la Modelación Ágil. La combinación de los dos derivó en la formación de un proceso de desarrollo que se adecuó al proyecto.

Se realizó la implementación de la herramienta, partiendo de los conceptos de extensibilidad de Eclipse, y tomando en cuenta el estándar de la sintaxis SQL de ANSI para la elaboración de consulta sobre bases de datos. De acuerdo esto, y siguiendo los requerimientos solicitados por el usuario la herramienta cuenta con las siguientes características:

- **Configurable:** Adición de nuevos sistemas manejadores de bases de datos realizando pocos cambios en el plug-in. Esto es posible gracias al uso de la tecnología XML para construir archivos de configuración genéricos con la información referente al sistema de base de datos; tal como sintaxis, funciones, operadores, URL de conexión.

- **Extensibilidad de funcionalidades:** Posibilidad de incorporar nuevos requerimientos con mínimo esfuerzo. Gracias al empleo de patrones (por ejemplo Factory Method, Command, MVC, entre otros empleados), es factible generar otras instrucciones, como actualizar, eliminar y modificar información sobre una base de datos.

- **Independencia de base de datos:** La herramienta es capaz de obtener metadatos y establecer conexiones, sobre SMBD soportados por JDBC.

- **Independencia de sintaxis:** Se cuenta con interfaces que permiten al usuario crear una consulta gráficamente, para diversos SMBD, independientemente de si estos difieren o no en la forma de construcción de la instrucción de consulta.

- **Probado**: Está formado por un conjunto de funcionalidades integradas, validadas (a través de pruebas automatizadas y manuales), las cuales fueron aceptadas por el usuario como un artefacto adecuado para la solución del problema inicial. Se implementó un conjunto de pruebas automatizadas empleando el framework JUnit, con el objetivo de validar las principales funcionalidades del modelo; el resto de las pruebas se realizó ejecutando la aplicación.

- **Orientado al desarrollo de aplicaciones**: Está integrado en la plataforma Eclipse, para facilitar a los desarrolladores realizar aplicaciones que requieran la consulta de datos desde una base de datos.

Como valor agregado de este Trabajo Especial de Grado, se pueden extraer los aspectos relevantes que se tomaron en cuenta para llevar a cabo con éxito el cumplimiento de los objetivos planteados. Además se mencionan prácticas que pueden resultar útiles para sobrellevar las complicaciones que se presentaron durante el proceso de desarrollo.

### Definición de requerimientos del usuario

Es importante definir los requerimientos iniciales del usuario, con el fin medir el tamaño del proyecto, definir el alcance, establecer un proceso de desarrollo y administrar el recurso tiempo.

La definición de los requerimientos iniciales no tiene por qué contemplar a detalle todas las funcionalidades del sistema. Tampoco deben ser modeladas formalmente; si se tiene como meta un desarrollo ágil lo importante es que se tenga conocimiento de los deseos del usuario, no extender en una documentación que pudiera convertir la fase de análisis en un proceso pesado y tedioso.

### Manejo de patrones

Los patrones de software pueden ser útiles para facilitar la búsqueda de buenas soluciones, debido a que cada patrón representa un esquema de solución para un tipo de problema en particular.

La aplicación de patrones no se realiza de forma mecánica, sino que se adaptan sobre las funcionalidades del nuevo software. Entre los patrones empleados en este TEG se encuentran: Observer, MVC, Command, Factory Method, Proxy y Singleton.

Gracias a la estructura de Eclipse, basada en plug-ins, las funcionalidades se pudieron integrar en un entorno de desarrollo junto con otras herramientas heterogéneas. Esta integración, facilita el acceso a las funcionalidades de generación de consulta para los usuarios del IDE, debido a que se emplean las interfaces predefinidas en Eclipse.

## *Aportes*

- El plug-in construido constituye una herramienta útil en la construcción de software que requiera recuperar datos desde una base de datos.
- El proceso de desarrollo definido con la mezcla de Programación Extrema y Modelación Ágil, puede ser empleado para la construcción de aplicaciones que no requieran extensa documentación y se cuente con un equipo de desarrollo pequeño.
- Se demostró que es posible establecer una estructura común de la sintaxis de consulta de los diferentes manejadores de bases de datos del mercado.

# Referencias bibliográficas


- Ambler, Scott. Agile Modelling (AM). Effective Practices for Modeling and Documentation, http://www.agilemodeling.com.
- ANSI. ANSI Standards Store, http://webstore.ansi.org/ansidocstore/default.asp.
- Microsoft Corporation. Arquitectura de Software: Métodos Heterodoxos en Desarrollo de Software, http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/heterodox.asp.
- Leffler, Jonathan (2005, Julio). BNF Grammars for SQL, http://savage.net.au/SQL.
- ECLIPSE Fundation. Eclipse.org home, http://www.eclipse.org.
- ECLIPSE Fundation. Eclipse Plugin Central, http://www.eclipseplugincentral.com/.
- Java Source and Support. Eclipse Plugin: SWT JFace Eclipse : Java examples, http://www.java2s.com/Code/Java/SWT-JFace-Eclipse/Eclipse-Plugin.htm.
- Alarcón, Pedro (1994). El estandar ANSI SQL (86), http://www.oei.eui.upm.es/Asignaturas/BD/ABD/doc/prac/SQL86.pdf.
- Oracle Corporation. Enterprise Applications | Oracle, http://www.oracle.com.
- Autor desconocido. Extreme Programming: A Gentle Introduction, http://www.extremeprogramming.org.
- Berzal, Fernando. Interfaces gráficas de usuario, http://elvex.ugr.es/decsai/java/pdf/D0-gui.pdf.
- iServicePro Technologies Inc (2007). iServicePro—Eclipse Plug-in Solution, Offshore Software Outsourcing, http://www.iservicepro.com/index.html.
- Sun Microsystems. Java SE Technologies – Java Database Connectivity (JDBD), http://java.sun.com/products/jdbc.
- Comunidad MySQL. MySQL Hispano – La comunidad de usuarios de MySQL, http://www.mysql-hispano.org.
- MySQL AB. MySQL Reference Manual, http://dev.mysql.com/doc/refman/4.1/en/index.html.
- Oracle Corporation (2002). Oracle 9i Database Online Documentation, http://www.stanford.edu/dept/itss/docs/oracle/9i
- Wikimedia Foundation, Inc. Portada – Wikipedia, la enciclopedia libre, http://es.wikipedia.org/wiki/Portada.

- Network World, Inc. Rich clients with the SWT and Jface, http://www.javaworld.com/javaworld/jw-04-2004/jw-0426-swtjface.html

- EMS Database Management Solutions, Inc. SQL Manager: Database Management Tools, http://www.sqlmanager.net.

- Froute, Agustín (1999). Tutorial de Java – Conectividad JDBC, http://www.itapizaco.edu.mx/paginas/JavaTut/froufe/parte21/cap21-3.html.

- Calero, Manuel (2003). Una explicación de la programación extrema (XP), http://www.willydev.net/descargas/prev/ExplicaXP.pdf

- Teknoda ( 2004, Enero). Una introducción a JDBC (Java Database Connectivity) http://www.teknoda.com/tips/java/TIPJA02B.PDF.

# Anexos

# Anexo A. BNF de la operación de consultas del SQL

En este anexo se formaliza la sintaxis de la operación de consulta mencionada en el capítulo I. A continuación se presenta el BNF propuesto por ANSI para esta operación:

**Tabla 35: Notación de BNF utilizada**

| Símbolo | Significado |
|---|---|
| < > | Representa los símbolos no terminales del lenguaje. |
| ::= | Operador de definición. |
| [] | Indica elementos opcionales. |
| { } | Agrupa elementos en una fórmula. |
| \| | Indica una alternativa. |
| ... | Indica repetición. |

## Sintaxis de la sentencia de consulta

```
<expresión-de-consulta> ::=
     <expresión-de-consulta-de-no-combinación>
   | <tabla-combinada>

<expresión-de-consulta-de-no-combinación> ::=
     <término-de-consulta-de-no-combinación>
   | <expresión-de-consulta> UNION [ALL]
   [ <especificación-de-correspondencia>] <término-de-consulta>
   | <expresión-de-consulta> EXCEPT [ALL]
   [<especificación-de-correspondencia>] <término-de-consulta>

<término-de-consulta> ::=
     <término-de-consulta-de-no-combinación>
   | <tabla-combinada>

<término-de-consulta-de-no-combinacion> ::=
     <primario-de-consulta-de-no-combinacion>
   | <término-de-consulta> INTERSECT [ALL]
   [<especificación-de-correspondencia> ] <primario-de-consulta>

<primario-de-consulta> ::=
     <primario-de-consulta-de-no-combinación>
   | <tabla-combinada>

<primario-de-consulta-de-no-combinación> ::=
  <tabla-simple>
| <parent-izq> <expresión-de-combinación-de-no-consulta><parent-dcho>

<tabla-simple> ::=
     <especificación-de-consulta>
   | <constructor-de-valor-de-tabla>
   | <tabla-explicita>

<tabla-explicita> ::= TABLE <nombre-de-tabla>

<especificación-de-correspondencia> ::=
   CORRESPONDING
[BY <parent-izq> <lista-de-columnas-que-correspoden> <parent-dcho>]

<lista-de-columnas-que-corresponden> ::= <lista-de-columnas>
```

```
<constructor-de-valor-de-tabla> ::=
   VALUES <lista-de-constructores-de-valores-de-tabla>

<lista-de-constructores-de-valores-de-tabla> ::=
   <constructor-de-valor-de-fila>
[ {<coma> <constructor-de-valor-de-fila> …} ]

<constructor-de-valor-de-fila> ::=
     <elemento-constructor-de-valor-de-fila>
   | <parent-izq> <lista-de-constructores-de-valor-de-fila> <parent-dcho>
   | <subconsulta-de-fila>

<lista-de-constructores-de-valor-de-fila> ::=
     <elemento-constructor-de-valor-de-fila>
   [ {<coma><elemento-constructor-de-valor-de-fila>} …]

<elemento-constructor-de-valor-de-fila> ::=
     <expresión-de-valor>
   | <especificiación-de-nulos>
   | DEFAULT

<especificación-de-nulos> ::= NULL

<subconsulta-de-fila> ::= <subconsulta>

<subconsulta> ::=
<parent-izq> <expresión-de-consulta> <parent-dcho>

<especificación-de-consulta> ::=
SELECT [<cuantificador-de-conjunto>]
<lista-de-consulta> <expresión-de-tabla>

<lista-de-consulta> ::=
  <asterisco>
| <sublista-de-consulta>
[{<coma> <sublista-de-consulta>} ...]

<sublista-de-consulta> ::=
<columna-derivada> | <calificador> <punto> <asterisco>

<columna-derivada> ::= <expresión-de-valor> [<cláusula-as>]

<cláusula-as> ::= [AS] <nombre-de-columna>

<cuantificador-de-conjunto> ::= DISTINCT | ALL

<expresión-de-tabla> ::=
<cláusula-from>
[ <cláusula-where> ]
[ <cláusula-group-by> ]
[ <cláusula-having>]
[<cláusula-order-by>]

<cláusula-from> ::=
FROM <referencia-a-tabla>
[ { <coma> <referencia-a-tabla> } ... ]

<referencia-a-tabla> ::=
<nombre-de-tabla> [ [AS] <nombre-de-correlación>
[<parent-izq> <lista-de-columnas-derivadas> <parent-dcho]] |
<tabla-derivada> [AS] <nombre-de-correlación>
[<parent-izq> <lista-de-columnas-derivadas> <parent-dcho] |
<tabla-combinada>

<tabla-derivada> ::= <subconsulta-de-tabla>

<lista-de-columnas-derivadas> ::= <lista-de-nombres-de-columnas>

<lista-de-nombres-de-columnas> ::=
<nombre-de-columna> [ { <coma> <nombre-de-columna> } ... ]
```

```
<nombre-de-correlación> ::= <identificador>

<calificador> ::= <nombre-de-tabla> | <nombre-de-correlación>

<tabla-combinada> ::= <combinación-cruzada>|
     <combinación-calificada> |
     <parent-izq> <tabla-combinada> <parent-dcho>

<combinación-cruzada> ::=
<referencia-a-tabla> CROSS JOIN <referencia-a-tabla>

<combinación-calificada> ::=
<referencia-a-tabla>
[NATURAL] [<tipo-de-combinación>] JOIN <referencia-a-tabla>
[<especificación-de-combinación>]

<especificación-de-combinación> ::=
<condición-de-combinación> |
<combinación-de-columnas-nominadas>

<condición-de-combinación> ::= ON <condición-de-búsqueda>

<combinación-de-columnas-nominadas> ::=
USING <parent-izq> <lista-de-columnas-de-combinación> <parent-dcho>

<tipo-de-combinación> ::=
INNER
| <tipo-de-combinación-externa> [OUTER]
| UNION

<tipo-de-combinación-externa> ::= LEFT | RIGHT | FULL

<lista-de-columnas-de-combinación> ::= <lista-de-nombres-de-columnas>

<cláusula-where> ::= WHERE <condición-de-búsqueda>

<cláusula-group-by> ::=
GROUP BY <lista-de-referencias-a-columna-de-agrupamiento>

<lista-de-referencias-a-columna-de-agrupamiento> ::=
<referencia-a-columna-de-agrupamiento>
[{<coma><referencia-a-columnas-de-agrupamiento>}...]

<referencia-a-columna-de-agrupamiento> ::= <referencia-a-columnas>

<referencias-a-columnas> ::=
[ <calificador> <punto> ] <nombre-de-columnna>

<cláusula-having> ::= HAVING <condición-de-búsqueda>

<condición-de-búsqueda> ::=
  <término-logico>
| <condición-de-búsqueda> OR <término-logico>

<término-logico> ::=
  <factor-logico>
| <factor-logico> AND <factor-logico>

<factor-logico> ::= [ NOT ] <evaluacion-logica>

<evaluacion-logica> ::=
<primario-logico> [ IS [ NOT ] <valor-lógico> ]

<valor-lógico> ::= TRUE | FALSE | UNKNOWN

<primario-logico> ::=
  <predicado>
| <parent-izq> <condición-de-búsqueda> <parent-dcho>

<predicado> ::= <predicado-de-comparación> |
     <predicado-entre> |
```

```
     <predicado-en> |
     <predicado-parece> |
     <predicado-de-nulidad> |
     <predicado-de-comparación-cuantificada> |
     <predicado-existe> |
     <predicado-de-unicidad> |
     <predicado-de-concordancia> |
     <predicado-de-solapamiento>

<predicado-de-comparación> ::=
<constructor-de-valor-de-fila>
<operación-de-comparación>
<constructor-de-valor-de-fila>

<predicado-entre> ::=
<constructor-de-valor-de-fila> ::= [ NOT ] BETWEEN
<constructor-de-valor-de-fila> AND <constructor-de-valor-de-fila>

<predicado-en> ::=
<constructor-de-valor-de-fila>
[ NOT ] IN <valor-de-predicado-en>

<valor-de-predicado-en> ::=
  <subconsulta-de-tabla>
| <parent-izq> <lista-de-valores-en> <parent-dcho>

<lista-de-valores-en> ::=
<expresión-de-valor>
{ <coma> <expresión-de-valor> } ...

<predicado-parece> ::= <valor-de-concordancia> [ NOT ] LIKE <patrón>

<valor-de-concordancia> ::= <expresión-de-valores-caracter>

<patrón> ::= <expresión-de-valores-caracter>

<predicado-de-nulidad> ::=
<constructor-de-valor-de-fila> IS [ NOT ] NULL

<predicado-de-comparación-cuantificada> ::=
<constructor-de-valor-de-fila> <operación-de-comparación>
<cuantificador> <subconsulta-de-tabla>

<cuantificador> ::= <todos> | <algunos>

<todos> ::= ALL

<algunos> ::= SOME | ANY

<predicado-existe> ::= EXISTS <subconsulta-de-tabla>

<predicado-de-unicidad> ::= UNIQUE <subconsulta-de-tabla>

<predicado-de-concordancia> ::=
<constructor-de-valor-de-fila>
MATCH[ UNIQUE ][ PARTIAL | FULL]<subconsulta-de-tabla>

<predicado-de-solapamiento> ::=
<constructor-de-valor-de-fila 1>
OVERLAPS
<constructor-de-valor-de-fila 2>

<constructor-de-valor-de-fila 1> ::= <constructor-de-valor-de-fila>

<constructor-de-valor-de-fila 2> ::= <constructor-de-valor-de-fila>

<cláusula-order-by> ::=
   ORDER BY <lista-de-especificación-de-orden>

<lista-de-especificación-de-orden> ::=
   <especificación-de-orden>
```

```
   [ {<coma> <especificación-de-orden>}…]

<especificación-de-orden> ::=
   <clave-de-orden>
   [<espcificación-de-ordenamiento>]

<clave-de-orden> ::= <nombre-columna>

<espcificación-de-ordenamiento> ::= ASC | DESC
```

**Ilustración 85: Sintaxis SQL de la expresión Select**

# Anexo B: Casos de uso

Una vez identificados los requerimientos para la nueva herramienta, se definen las funcionalidades que permiten establecer estas características de una manera más formal, definiendo sus casos de uso correspondientes.

Tomando en cuenta principios y prácticas propuestas por la modelación ágil se realizara la especificación de los casos de usos de forma genérica con el fin de representar las características principales del sistema.

### *3.4.1. Nivel 0*

De las historias de usuario anteriores se puede ver que se quieren manejar consultas de recuperación de datos dentro de un proyecto de Eclipse. De allí se deriva el primer nivel de los casos de usos:

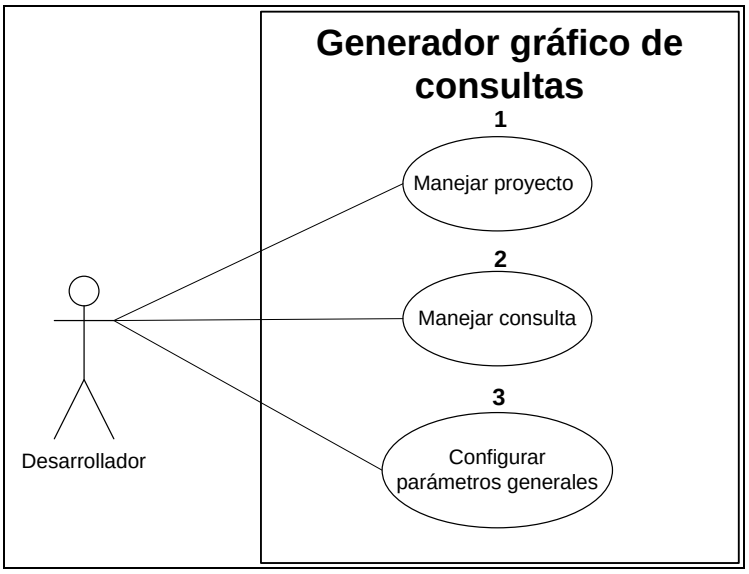


**Ilustración 86: Nivel 0 - Casos de uso**

| **Nombre** | **Manejar proyecto** |
|---|---|
| **Número** | 1 |
| **Actor** | Desarrollador |
| **Descripción** | Permite al usuario crear, seleccionar, abrir, guardar, eliminar un proyecto, así como también especificar los parámetros generales del mismo. |
| **Flujo de eventos** | Para seleccionar crear un nuevo proyecto:<br>1.- El actor cualquier opción relacionada con el manejo de un proyecto.<br>2.- El sistema lleva a cabo la operación e indica sus resultados. |
| **Precondición** | - |
| **Post condición** | El actor ha realizado cualquier operación relacionada con la administración de proyectos. |

| **Nombre** | **Manejar consulta** |
|---|---|

| | |
|---|---|
| **Número** | 2 |
| **Actor** | Desarrollador |
| **Descripción** | Permite al actor realizar cualquier operación asociada a una consulta, tales como crear, abrir, modificar, guardar, generar, ejecutar y eliminar una consulta. |
| **Flujo de eventos** | Para manejar una consulta se realiza lo siguiente:<br>1.- El actor escoge cualquier funcionalidad que implique el manejo de consulta.<br>2.- El sistema lleva a cabo la operación y le indica el resultado al actor. |
| **Precondición** | El actor ha seleccionado la base de datos. |
| **Post condición** | El actor ha indicado los parámetros de la consulta. |

| **Nombre** | **Configurar parámetros generales** |
|---|---|
| **Número** | 3 |
| **Actor** | Desarrollador |
| **Descripción** | Permite al actor establecer los parámetros generales de la aplicación, tales como idioma, generación automática de consulta. |
| **Flujo de eventos** | Para establecer los parámetros generales del sistema se realiza lo siguiente:<br>1.- El actor accede a la sección de configuración de parámetros generales.<br>2.- El sistema solicita los datos asociados a la configuración que desea realizar el actor.<br>3.- El actor indica los datos solicitados.<br>4.- Si los datos son válidos, el sistema lleva a cabo la configuración y muestra el resultado al actor. |
| **Precondición** | - |
| **Post condición** | Se ha configurado algún aspecto del sistema. |

### *3.4.2. Nivel 1*

En este nivel se busca detallar los casos de uso anteriores, para definir de forma más específica las funcionalidades a ser desarrolladas.

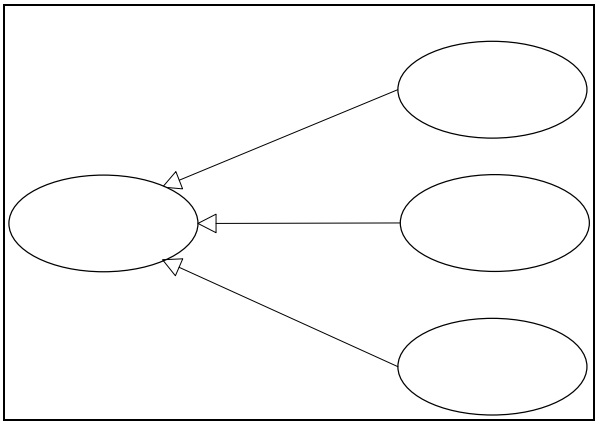

**Ilustración 87: Caso de uso Manejar proyecto**

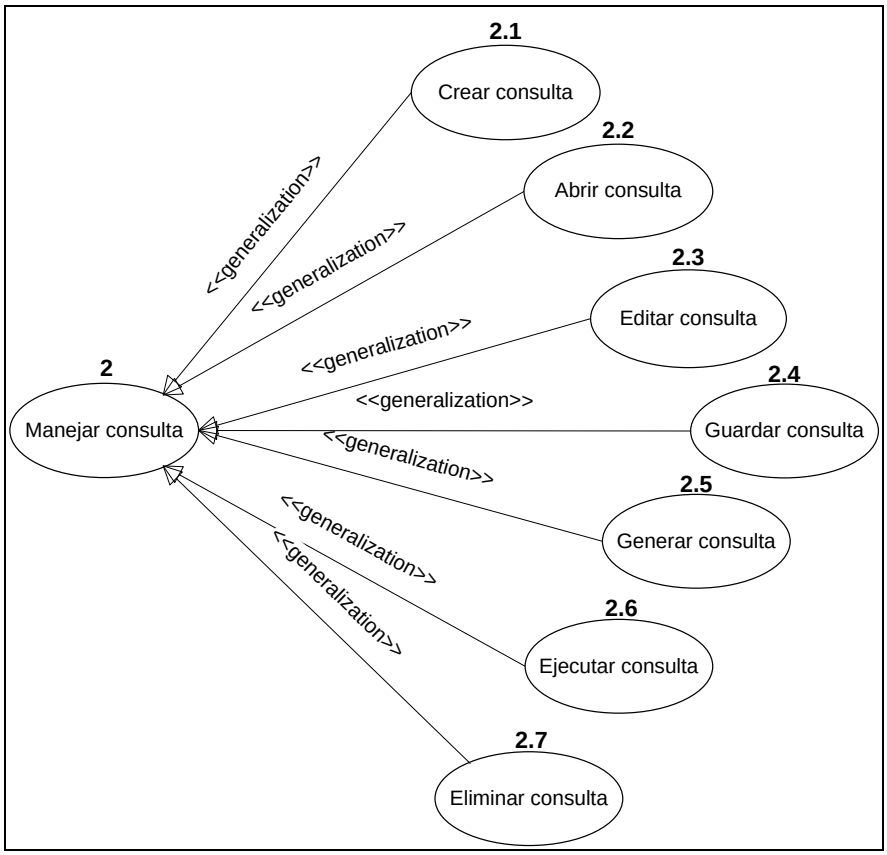


**Ilustración 88: Caso de uso Manejar consulta**

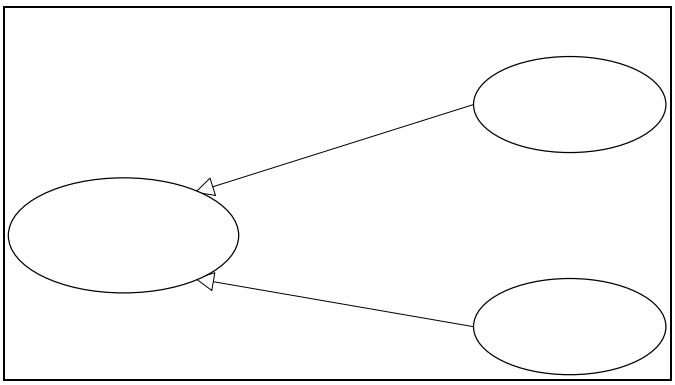

**Ilustración 89: Caso de uso configurar parámetros generales**

| Nombre | Crear proyecto |
|---|---|
| **Número** | 1.1 |
| **Actor** | Desarrollador |
| **Descripción** | Permite al usuario crear un proyecto, dentro del cual podrá crear las consultas deseadas. |
| **Flujo de eventos** | Para crear un nuevo proyecto se realiza lo siguiente:<br>1.- El actor escoge la opción de creación de un nuevo proyecto.<br>2.- El sistema solicita los parámetros de creación del proyecto (parámetros de conexión y ruta del proyecto).<br>3.- El actor indica los datos.<br>4.- El sistema verifica los datos introducidos. Si son válidos el sistema obtiene los metadatos de la base de datos seleccionada, carga las estructuras con la sintaxis del SELECT correspondiente y las estructuras correspondientes al proyecto. |
| **Precondición** | - |
| **Post condición** | El actor ha creado un proyecto en el cual podrá crear consultas sobre la base de datos indicada. |

| Nombre | Abrir proyecto |
|---|---|
| **Número** | 1.2 |
| **Actor** | Desarrollador |
| **Descripción** | Permite al usuario abrir un proyecto. Durante esta operación se llevan cabo la obtención de los metadatos y la carga de las estructuras de archivos asociadas al proyecto. |
| **Flujo de eventos** | Para abrir un proyecto se realiza lo siguiente:<br>1.- El actor escoge la opción de apertura de un proyecto existente.<br>2.- El sistema carga las estructuras y obtiene los metadatos. Si las operaciones son exitosas se abre y selecciona el proyecto indicado. |
| **Precondición** | El proyecto que se desea abrir existe. |
| **Post condición** | El actor ha abierto un proyecto existente. |

| Nombre | Eliminar proyecto |
|---|---|
| **Número** | 1.3 |
| **Actor** | Desarrollador |
| **Descripción** | Permite al usuario eliminar un proyecto existente. |
| **Flujo de eventos** | Para eliminar un proyecto se realiza lo siguiente:<br>1.- El actor escoge la opción eliminar proyecto existente.<br>2.- El sistema elimina las estructuras asociadas al proyecto. |
| **Precondición** | El proyecto que se desea eliminar existe y está seleccionado. |
| **Post condición** | El actor ha eliminado un proyecto existente. |

| Nombre | Crear consulta |
|---|---|
| **Número** | 2.1 |
| **Actor** | Desarrollador |
| **Descripción** | Permite al actor crear una nueva consulta de recuperación de datos. |
| **Flujo de eventos** | Para crear una consulta se realiza lo siguiente:<br>1.- El actor escoge la opción de creación de consulta.<br>2.- El sistema solicita el nombre de la consulta y crea una consulta vacía. |
| **Precondición** | El actor ha seleccionado un proyecto abierto. |
| **Post condición** | Se ha creado una nueva consulta vacía. |

| Nombre | Abrir consulta |
|---|---|
| **Número** | 2.2 |
| **Actor** | Desarrollador |
| **Descripción** | Permite al actor abrir una consulta existente. |
| **Flujo de eventos** | Para abrir una consulta se realiza lo siguiente:<br>1.- El actor escoge la consulta que desea abrir y escoge la opción de apertura.<br>2.- El sistema lleva a cabo la apertura y en caso de ser exitosa muestra la consulta. |
| **Precondición** | El actor ha seleccionado una consulta. |
| **Post condición** | Se ha abierto una consulta existente. |

| Nombre | Editar consulta |
|---|---|
| **Número** | 2.3 |
| **Actor** | Desarrollador |
| **Descripción** | Permite al actor modificar una consulta existente, específicamente: indicar la consulta de recuperación de datos, lo cual consiste en establecer las diferentes cláusulas del select (select, from, where, group by, having, order by) y establecer las propiedades de la consulta. |
| **Flujo de eventos** | Para editar una consulta se realiza lo siguiente:<br>1.- El actor escoge la modificación que desea realizar sobre la consulta.<br>2.- El sistema solicita los datos necesarios para realizar la modificación.<br>3.- El actor indica los datos.<br>4.- Si los datos son válidos, el sistema lleva a cabo la acción indicada. |
| **Precondición** | El actor ha abierto una consulta. |
| **Post condición** | Se ha modificado una consulta. |

| Nombre | Guardar consulta |
|---|---|
| **Número** | 2.4 |
| **Actor** | Desarrollador |
| **Descripción** | Permite al actor guardar una consulta. |
| **Flujo de eventos** | Para guardar una consulta se realiza lo siguiente:<br>1.- El actor selecciona la opción de guardar consulta.<br>2.- El sistema lleva a cabo la actualización de las estructuras de archivo asociadas a la consulta e indica al actor el resultado. |
| **Precondición** | La consulta que se desea guardar está abierta y seleccionada. |
| **Post condición** | Se ha guardado la consulta. |

| Nombre | Generar consulta |
|---|---|
| **Número** | 2.5 |
| **Actor** | Desarrollador |
| **Descripción** | Permite generar la sintaxis SQL a partir de la consulta gráfica. |
| **Flujo de eventos** | Para generar una consulta se realiza lo siguiente:<br>1.- El actor selecciona la opción de generar consulta.<br>2.- El sistema genera la consulta SQL a partir de los parámetros establecidos por el usuario en la consulta gráfica.<br>3.- En caso de ser exitoso, se muestra la consulta generada. Sino se muestra el error al actor. |
| **Precondición** | La consulta gráfica debe está abierta y seleccionada. |
| **Post condición** | Se ha generada la sintaxis SQL asociada a la consulta. |

| Nombre | Ejecutar consulta |
|---|---|
| **Número** | 2.6 |
| **Actor** | Desarrollador |
| **Descripción** | Permite al actor ejecutar la consulta generada o una indicada por él mismo. |
| **Flujo de eventos** | Para ejecutar una consulta se realiza lo siguiente:<br>1.- El actor selecciona la opción “ejecutar consulta”.<br>2.- El sistema ejecuta la consulta generada, o indicada por el desarrollador, sobre la base de datos indicada por el actor.<br>3.- En caso de ser exitoso, se muestra el resultado de la consulta. Sino se muestra el error. |
| **Precondición** | La consulta gráfica debe estar abierta y seleccionada. La consulta indicada es distinta de vacío. |
| **Post condición** | Se ha ejecutado una consulta. |

| Nombre | Eliminar consulta |
|---|---|
| **Número** | 2.7 |
| **Actor** | Desarrollador |
| **Descripción** | Permite al actor eliminar una consulta existente. |
| **Flujo de eventos** | Para eliminar una consulta se realiza lo siguiente:<br>1.- El actor selecciona la opción eliminar consulta.<br>2.- El sistema lleva a cabo la eliminación de todas las estructuras asociadas a la consulta y muestra el resultado al actor. |
| **Precondición** | La consulta gráfica debe estar seleccionada. |
| **Post condición** | Se ha eliminado una consulta. |

| Nombre | Configurar idioma |
|---|---|
| **Número** | 3.1 |
| **Actor** | Desarrollador |
| **Descripción** | Permite al actor configurar el idioma del sistema. |
| **Flujo de eventos** | Para establecer el idioma del sistema se realiza lo siguiente:<br>1.- El actor accede a la sección de configuración de parámetros generales y selecciona la configuración del idioma.<br>2.- El sistema solicita el idioma.<br>3.- El actor indica el idioma.<br>4.- Si los datos son válidos, el sistema lleva a cabo la configuración y muestra el resultado al actor. |
| **Precondición** | - |
| **Post condición** | Se ha configurado el idioma del sistema. |

| Nombre | Configurar generación |
|---|---|
| **Número** | 3.2 |
| **Actor** | Desarrollador |
| **Descripción** | Permite al actor configurar la forma de generación de consulta. |
| **Flujo de eventos** | Para establecer la forma de generación de consulta se realiza lo siguiente:<br>1.- El actor accede a la sección de configuración de parámetros generales y selecciona la configuración de generación.<br>2.- El sistema solicita el tipo de generación (automática o manual).<br>3.- El actor indica el tipo de generación.<br>4.- Si los datos son válidos, el sistema lleva a cabo la configuración y muestra el resultado al actor. |
| **Precondición** | - |
| **Post condición** | Se ha configurado la forma de generación de la consulta. |

### *3.4.3. Nivel 2*

Debido a que la complejidad más compleja de la herramienta es editar la consulta de recuperación de datos, se realiza este nivel para detallar las funcionalidades correspondientes a la edición de este tipo de consulta.

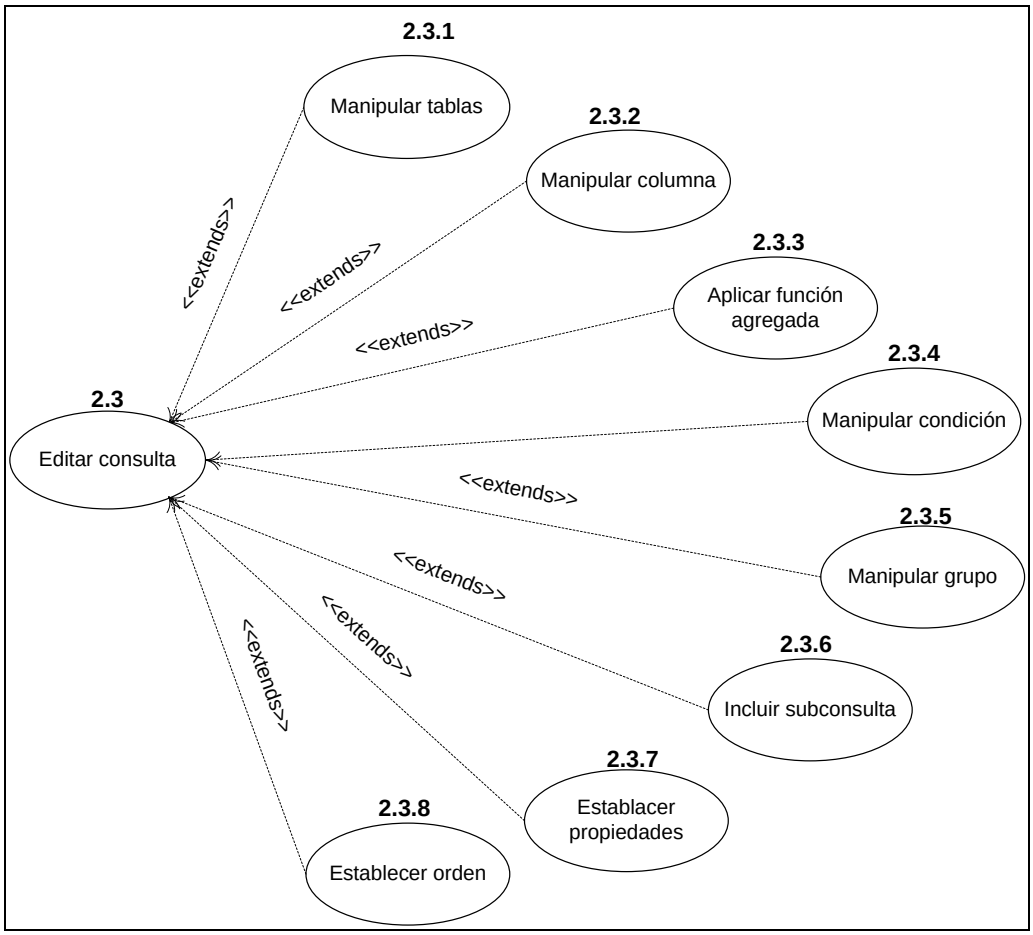


**Ilustración 90: Caso de uso Editar consulta**

| **Nombre** | **Manipular tablas** |
|---|---|
| **Número** | 2.3.1 |
| **Actor** | Desarrollador |
| **Descripción** | Permite al actor realizar acciones con las tablas de la base de datos indicada, tales como: agregarlas a la consulta como origen de datos, enlazarlas, entre otras. |
| **Flujo de eventos** | Para manipular una tabla se llevan a cabo los siguientes pasos:<br>1.- El actor selecciona las tablas de la base de datos y realiza cualquier acción sobre ellas<br>2.- El sistema actualiza estas operaciones en la consulta gráfica. |
| **Precondición** | Hay una consulta abierta y seleccionada. |
| **Post condición** | El actor realiza cualquier operación referente a la manipulación de tablas de la consulta activa. |

| Nombre | Manipular columna |
|---|---|
| **Número** | 2.3.2 |
| **Actor** | Desarrollador |
| **Descripción** | Permite al usuario realizar acciones con columnas de la consulta gráfica, tales como: incluir, eliminar, mostrar u ocultar columna. |
| **Flujo de eventos** | Para manipular una columna se llevan a cabo los siguientes pasos:<br>1.- El actor realiza una operación de manipulación de columna.<br>2.- El sistema actualiza estas operaciones en la consulta gráfica. |
| **Precondición** | Hay una consulta abierta y seleccionada. |
| **Post condición** | El actor realiza cualquier operación referente a la manipulación de columna. |

| Nombre | Aplicar función agregada |
|---|---|
| **Número** | 2.3.3 |
| **Actor** | Desarrollador |
| **Descripción** | Permite al actor aplicar una función agregada a la consulta gráfica. |
| **Flujo de eventos** | Para aplicar una función se llevan a cabo los siguientes pasos:<br>1.- El actor selecciona la función e indica sus parámetros.<br>2.- El sistema actualiza estas operaciones en la consulta gráfica. |
| **Precondición** | Hay una consulta abierta y seleccionada. |
| **Post condición** | Se aplica una función agregada a la consulta gráfica. |

| Nombre | Manipular condición |
|---|---|
| **Número** | 2.3.4 |
| **Actor** | Desarrollador |
| **Descripción** | Permite al actor agregar o eliminar una condición de filtrado sobre la consulta gráfica. |
| **Flujo de eventos** | Para manipular una condición se llevan a cabo los siguientes pasos:<br>1.- El actor selecciona la operación que desea realizar e introduce los datos necesarios.<br>2.- El sistema lleva a cabo la operación y muestra el resultado al actor. |
| **Precondición** | Hay una consulta abierta y seleccionada. |
| **Post condición** | El actor ha manipulado las condiciones asociadas a la consulta gráfica. |

| Nombre | Manipular grupo |
|---|---|
| **Número** | 2.3.5 |
| **Actor** | Desarrollador |
| **Descripción** | Permite al actor agregar y eliminar una agrupación sobre las tuplas resultantes de la consulta. |
| **Flujo de eventos** | Para agregar o eliminar un grupo sobre los resultados de una consulta se llevan a cabo los siguientes pasos:<br>1.- El actor indica la acción a realizar y los parámetros necesarios.<br>2.- El sistema establece o elimina l la agrupación y actualiza estas operaciones en la consulta gráfica. |
| **Precondición** | Hay una consulta abierta y seleccionada. |
| **Post condición** | El actor ha agregado o eliminado un campo de agrupación sobre el resultado de la consulta gráfica. |

| Nombre | Incluir subconsulta |
|---|---|
| Número | 2.3.6 |
| Actor | Desarrollador |
| Descripción | Permite al actor relacionar la consulta gráfica actual con otra previamente creada. |
| Flujo de eventos | Para incluir una subconsulta se llevan a cabo los siguientes pasos:<br>1.- El actor indica la consulta con la cual se desea establecer la relación.<br>2.- El sistema incluye la nueva consulta y actualiza estas operaciones en la consulta gráfica. |
| Precondición | Hay una consulta abierta y seleccionada. |
| Post condición | El actor ha relacionado la consulta gráfica con otra consulta. |

| Nombre | Establecer propiedades |
|---|---|
| Número | 2.3.7 |
| Actor | Desarrollador |
| Descripción | Permite al actor indicar las propiedades generales de la consulta, tales como tamaño, descripción y repeticiones. |
| Flujo de eventos | Para establecer las propiedades de la consulta se realiza lo siguiente:<br>1.- El actor introduce las preferencias con respecto a la consulta a generar.<br>2.- El sistema realiza la verificación de los datos introducidos.<br>3.- Si son válidos el sistema guarda las propiedades de la consulta.<br>4.- El sistema indica al actor el resultado de las operaciones. En caso de ser satisfactorio, muestra la vista de edición de la consulta. |
| Precondición | Hay una consulta abierta y seleccionada. |
| Post condición | El actor ha indicado las propiedades de la consulta. |

| Nombre | Manipular orden |
|---|---|
| Número | 2.3.8 |
| Actor | Desarrollador |
| Descripción | Permite al actor establecer orden, ascendente o descendente, sobre las tuplas resultantes de la consulta. También puede eliminarse un orden establecido anteriormente. |
| Flujo de eventos | Para establecer un orden en una consulta se llevan a cabo los siguientes pasos:<br>1.- El actor indica el orden deseado.<br>2.- El sistema establece el orden y actualiza estas operaciones en la consulta gráfica.<br>Para eliminar un orden en una consulta se llevan a cabo los siguientes pasos:<br>1.- El actor selecciona el orden deseado e indica que desea eliminarlo.<br>2.- El sistema elimina el orden y actualiza estas operaciones en la consulta gráfica. |
| Precondición | Hay una consulta abierta y seleccionada. |
| Post condición | El actor ha establecido el orden del resultado de la consulta. |