UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

LDMEXTRACTOR: UN EXTRACTOR TANTO DEL MODELO LÓGICO DE DATOS COMO DE LAS ESTADÍSTICAS DE UNA BASE DE DATOS A PARTIR DE LOS CATÁLOGOS DEL SISTEMA

T E S I S

QUE PARA OBTENER EL TÍTULO DE:
LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:
ISRAEL PÉREZ SANJUAN

DIRECTOR DE TESIS:
M. EN C. JAVIER GARCÍA GARCÍA

2007

Agradecimientos

Terminar una licenciatura es un logro tan grande que no puede solamente involucrar a mi persona. Sería, sin embargo, demasiado complicado listar a todos aquellos que han sido, y son, partícipes de dicho logro. No trataré de listar, sino sólo mencionar a quienes me llegan a la mente en este instante, sin que con ello se entienda que son los únicos.

Agradezco, pues, a mi madre, quien siempre me ha apoyado en mis estudios en mi vida personal hasta donde ha podido; a mi hermano, por ser un gran compañero y amigo; a Maricela, mi amor, quien en un breve tiempo me ha dado tanto y a quien tengo que agradecerle tantas cosas que simplemente no tiene sentido ni mencionar una sola; al maestro Javier García García, mi tutor, un gran profesor y compañero cuyo apoyo en la recta final de mis estudios profesionales fue pieza clave e indispensable; a mi familia, a mis profesores (no solamente en la universidad, sino a través de toda mi vida), a mis amigos, a mis compañeros de clase, a quienes trabajaron conmigo en Refined, y al resto de la gente que, en mayor o menor medida, le hayan aportado algo a mi vida y hayan hecho posible este trabajo, este logro y este momento.

Expreso aquí un agradecimiento especial para el Macroproyecto: Tecnologías para la Universidad de la Información y la Computación, que hizo posible el proyecto de investigación del cual formé parte y que derivó en esta tesis.

Este logro es solamente uno de muchos que deben venir. Simplemente gracias a todos…gracias…

Índice general

Índice de figuras

Índice de tablas

Introducción

Usualmente, mientras un sistema (computacional o de cualquier tipo) funciona correctamente, pocas veces se preguntan los usuarios acerca de sus partes, su funcionamiento, su construcción, y aún menos de su motivación y del por qué de la estructura elegida para su construcción.

Sin embargo, en la gran mayoría de las ocasiones, el conocimiento de dichas características del sistema del que se hace uso nos puede llevar a un mejor aprovechamiento de los recursos que este provee, ya sea proporcionando información acerca de sí mismo, de los elementos que procesa, otorgando posibilidad a configuraciones más específicas, entre otras, dependiendo el tipo de sistema y su construcción.

Al conocer a fondo las características de los sistemas con los que trabajamos podemos optimizar tareas comunes, minimizar las probabilidades de errores al automatizar procesos, estandarizar estos últimos y establecer normas de trabajo para grupos, equipos o aún otros sistemas que tengan acceso a ellos.

Actualmente, todas las organizaciones, privadas y de gobierno, requieren almacenar en medios digitales las grandes cantidades de datos que generan, de manera que estos sean eficientemente aprovechados por toda la variedad de usuarios para satisfacer las necesidades de obtención de información pues, como bien sabemos, los datos por sí solos no son útiles.

Una de las formas más comunes, probadas y confiables para almacenar, administrar, proteger y explotar los datos es mediante el uso de *Sistemas Manejadores de Bases de Datos* o *SMBD*. Son particularmente populares aquellos SMBD que son además Relacionales (SMBDR).

Los SMBDR actuales poseen una enorme variedad de características que, en general, deben en primera instancia cumplir con las doce reglas que Edgar Frank Codd [12, 13] publicó en 1985 para evaluar si un SMBD puede ser considerado como relacional o no. Entre estas reglas, Codd establece que para que un SMBD sea considerado como relacional, tanto la base de datos misma como el *Modelo Lógico de Datos* (LDM por sus siglas en inglés) deben poder ser accesibles mediante el uso exclusivo de las características relacionales del SMBD. Esta característica normalmente es implementada mediante la inclusión de una base de datos de *metadatos* (datos acerca de los datos) conocida como catálogo del sistema.

Los catálogos de los SMBDR actuales que manifiestan cumplir con un conjunto "adecuado" de las reglas descritas en el estándar de SQL, contienen información muy variada referente no solamente al LDM de la base de datos y a la base de datos en sí, sino también mantiene una variedad de estadísticas referentes a cada una de las tablas de las bases de datos, información acerca de los dominios, de los usuarios y sus roles y permisos, de los tipos de datos que el SMBD implementa y provee para una base de datos en particular, de procedimientos, de funciones definidas por el usuario, y de muchas otras cosas, tanto referentes a elementos creados por el usuario, como aquellos que han sido creados y son administrados y utilizados principalmente por el SMBD en sí.

Siendo que siempre es posible (al menos en teoría) utilizar las propiedades relacionales de los Sistemas Manejadores para acceder al modelo lógico de los datos, podemos pensar que el acceso a dicha información puede volverse automático eligiendo un conjunto adecuado de relaciones y creando un conjunto de consultas en lenguaje SQL que extraiga la información concerniente a una base de datos especificando únicamente el nombre de la misma.

En el presente trabajo se expone la construcción de una herramienta que permite extraer de manera automática el LDM y las estadísticas de una base de datos a partir de los catálogos del sistema. Dicha herramienta, a la que se ha denominado **LDMExtractor**, provee al usuario de una interfaz sencilla que construye una representación esquemática, clara, portable y entendible del LDM y las estadísticas más relevantes que se encuentran presentes en los catálogos de tres SMBDR: MySQL, PostgreSQL y Microsoft SQL Server. Dicha representación puede ser visible directamente en la aplicación y puede también ser guardada en un archivo con formato XML para maximizar la portabilidad. **LDMExtractor** provee también al programador de un modelo para extender de manera sencilla la aplicación, haciéndola compatible con otros SMBD si así lo requiere.

Al proveer al usuario de un medio para construir el LDM únicamente con el nombre de la base de datos, se apoyan tareas como las realizadas en procesos de mantenimiento de aplicaciones y de bases de datos donde el LDM no se tiene a la mano y se requiere de una representación simple pero completa del mismo, una que incluya las restricciones de integridad más importantes, tales como son las llaves únicas, las llaves primarias y las llaves foráneas, así como los *Check* y *Not Null Constraints* y donde se requiera de portabilidad en dicha representación, como es el caso de algunos estudios que involucran procesos de Ingeniería Inversa [16].

Así mismo, si lo que desea conocer el usuario es la información concerniente a las estadísticas más importantes de la base de datos y, por ejemplo, planea portar dicha base de datos a un SMBD distinto al que actualmente la almacena y gestiona, puede comparar la compatibilidad de dichas estadísticas, así como la eficiencia de su uso y su mantenimiento utilizando la misma aplicación.

A través de las siguientes páginas y capítulos se estudiará a fondo algunos de los catálogos más importantes de los SMBDR ya mencionados y se explicará a detalle los más importantes elementos de la herramienta que es producto de esta investigación.

En el capítulo uno se estudia la historia en el desarrollo de las ideas que motivaron la conceptualización del LDM de una base de datos. En este mismo capítulo se analizan también las propiedades más generales y relevantes del LDM, pues en capítulos

posteriores se comparan las ideas de la teoría con lo que llega al usuario mediante los SMBD.

En el capítulo dos se amplía la visión de la teoría detrás del LDM. Así mismo, se expone ampliamente la motivación de almacenar estadísticas acerca de las tablas, así como su papel en las tareas realizadas por el SMBD.

En el capítulo tres se describe el *Information Schema*, que es un conjunto de tablas relacionales y/o vistas (dependiendo de la implementación) mantenidas por el SMBDR, que contienen toda la información concerniente a cada una de las bases de datos almacenadas por él y que es parte del estándar de SQL. Se explica cómo los tres SMBDR que este estudio abarca implementan dicho conjunto, así como qué tanto cumple con lo planteado en el estándar.

El capítulo cuatro se concentra en el subconjunto del *Information Schema* o su equivalente en catálogos del sistema que contienen la información referente al LDM específicamente, y se describen consultas específicas para cada uno de los SMBD que el presente trabajo abarca que permiten extraer dicha información.

El capítulo cinco se enfoca en el subconjunto del *Information Schema* o su equivalente en catálogos del sistema que contienen la información referente a las estadísticas que estén presentes en los tres SMBD mencionados. Se explican los elementos implementados, así como las limitantes encontradas en dos de los tres SMBD estudiados en el presente trabajo.

El capítulo seis describe a detalle los elementos de **LDMExtractor** como aplicación. Se expone el modelo utilizado para representar al LMD tanto como un documento XML como en forma de clase dentro de un lenguaje de programación.

El capítulo siete presenta el trabajo relacionado a esta investigación.

El capítulo ocho contiene una recopilación de datos interesantes y hallazgos, producto de esta investigación.

El capítulo nueve, finalmente, presenta las conclusiones de esta investigación, así como un resumen comparativo de los resultados obtenidos para cada uno de los SMBD estudiados.

I. El catálogo de un SMBD

En este capítulo se estudian las características generales que los postulados del Modelo Relacional expuestos ampliamente en [11] por Codd exigen del catálogo de un SMBDR. Se analizará el por qué de las características más relevantes para, posteriormente, analizar cuáles de ellas son implementadas adecuadamente por los SMBDR que en este trabajo se estudian.

I. 1. Motivación del catálogo

Una propiedad importante del modelo relacional es que tanto la base de datos como su descripción (el LDM) sean perceptibles por los usuarios como una colección de relaciones de manera que, siempre que un usuario tenga los permisos adecuados, este pueda acceder al LDM justo como accede a las bases de datos, utilizando el mismo lenguaje y las mismas reglas [12, 13], de manera que el catálogo del sistema debe contener una descripción de sí mismo.

Esto tiene la enorme ventaja de que para poder acceder a la descripción lógica de una base de datos no es necesario aprender ningún lenguaje especial ni estructuras particulares, sino que es posible utilizar los mismos conocimientos adquiridos para la manipulación de las bases de datos en general. Así mismo, todos aquellos métodos existentes para preservar la seguridad aplicables a la base de datos, son igualmente aplicables al catálogo del sistema, por lo que un usuario con dominio y conocimiento del mismo no puede obtener más control sobre la base de datos que aquel que le fue asignado.

I. 2. Acceso al catálogo

Los SMBD generalmente cuentan tanto con el catálogo del sistema (la entidad estudiada en el presente capítulo) como con un *diccionario*. La diferencia entre uno y

otro es que el diccionario, además de la información concerniente al modelo lógico de los datos y las estadísticas, contiene información propia del sistema, como son parámetros de configuración y representaciones binarias de objetos varios, por mencionar algunas, y que es administrada casi exclusivamente por él.

El acceso al catálogo también debe ser muy eficiente, para que los accesos a este no se vuelvan un cuello de botella que comprometa el desempeño del SMBD en el resto de las consultas que se realizan. Esta propiedad permite accesos y análisis remotos a la estructura de la base de datos, que facilitan las tareas del DBA.

I. 3. Descripción de dominios, relaciones y vistas

A continuación se describen los elementos que deben ser almacenados en el catálogo con respecto a dominios, relaciones (tablas) y vistas. Nótese que es precisamente esta serie de elementos la que compone la información que debe reunir **LDMExtractor** para poder construir el esquema lógico de las tablas, las columnas y sus tipos, así como algunas restricciones importantes, como la posibilidad o no de valores nulos en una columna en particular.

Según Codd [11], para cada dominio existente en la base de datos, el catálogo debe contener:

- su nombre
- su tipo de dato básico
- el rango de valores permitido
- si el operador "<" es significativo o no para este dominio, es decir, si tiene un orden bien definido

Para cada una de las relaciones, el catálogo debe contener al menos:

- su nombre
- los sinónimos para dicho nombre
- el nombre de cada columna
- el nombre de un dominio previamente definido para cada columna
- qué tipo de "valores faltantes" son permitidos (generalmente `NULL` o `NOT NULL`)
- si los valores de la columna deben ser distintos entre si
- las restricciones propias de cada columna (fuera de aquellas que representan restricciones de dominio)
- el tipo de dato básico de cada columna si es que hay tal
- si la columna es componente de la llave primaria
- para cada llave foránea, las columnas de las que se compone, así como la llave referenciada

Para cada una de las columnas compuestas definidas, el catálogo debe contener:

- su nombre
- el nombre de cada columna componente individual de la columna compuesta
- para cada columna componente, un entero que indique la posición que cada una de ellas ocupa dentro de la columna compuesta

Para cada una de las vistas, el catálogo debe contener al menos:

- su nombre
- los sinónimos para este nombre, de haberlos
- el nombre de cada columna
- para cada columna, el nombre de un dominio previamente declarado, a menos que la columna no sea derivada de una columna base
- si una columna es o no parte de la llave primaria de la vista (en caso de existir una llave primaria)
- la expresión en lenguaje relacional que define la vista
- si la inserción de nuevas tuplas es permitida
- si la eliminación de tuplas es permitida
- para cada columna de la vista, si la actualización de valores es permitida

I. 4. Restricciones de integridad

La segunda parte importante de la información recopilada por **LDMExtractor** es la concerniente a las restricciones de integridad. Son particularmente importantes las restricciones de integridad referencial (llaves foráneas) y aquellas restricciones que se denominan *check constraints*, que se encuentran dentro de las restricciones de integridad multivariada, pues ellas aseguran, bajo un uso adecuado de las mismas, un nivel más elevado de calidad en los datos.

Para cada restricción de integridad multivariada definida por el usuario (de tipo U según Codd, [11]), el catálogo debe contener una descripción completa conformada por:

- su nombre
- el evento que la activa
- el *timing type* (el momento en que la acción definida para esta restricción es ejecutada)
- la condición lógica que se prueba
- la respuesta a la violación de la restricción

Para cada restricción de integridad referencial, el catálogo debe contener:

- su nombre
- el evento que la activa

- el timing type
- las llaves que están involucradas
- la respuesta a la violación de la restricción

I. 5. Funciones definidas por el usuario

Las funciones definidas por el usuario (UDFs por sus siglas en inglés) proveen un mecanismo para extender la funcionalidad de un servidor de bases de datos agregando una función que puede ser evaluada en sentencias SQL. El estándar de SQL distingue entre funciones escalares y de tablas. Una función escalar devuelve un sólo valor (o `NULL`), mientras que una función de tabla devuelve una tabla relacional conformada por cero o más tuplas.

Para cada función definida por el usuario, el catálogo debe contener:

- su nombre
- el código fuente
- el código compilado
- los nombres de las relaciones para las cuales la función requiere acceso de sólo lectura
- si la función tiene inversa y, de ser así, el nombre de la inversa, su código fuente y el código compilado

I. 6. Características de seguridad y rendimiento

Con respecto a la seguridad de los datos contenidos en la base de datos, el catálogo debe contener:

- para cada usuario, terminal y aplicación que tenga acceso a la base de datos, los datos específicos a los que cada uno de estos tiene acceso, así como el tipo de acceso que les es permitido
- las condiciones bajo las que cada uno de los permisos descritos en el punto anterior son posibles

Con respecto al rendimiento, los catálogos de los SMBDR mantienen un conjunto básico de estadísticas, otro de los elementos que **LDMExtractor** es capaz de extraer y procesar de manera automática. En este caso, el catálogo debe contener al menos las siguientes estadísticas:

- el número de tuplas en cada relación
- el número de valores distintos para cada una de las columnas de cada una de las relaciones

Más adelante se estudiará que, en la mayoría de los casos, los SMBDR mantienen un conjunto más grande de estadísticas que, al igual que las dos mencionadas, son utilizadas en el proceso de optimización de consultas.

I. 7. Lo aprendido del catálogo de System R

System R es un sistema manejador de bases de datos relacional creado en el Laboratorio de Investigación de IBM en el año de 1976. Fue construido con la intención de realizar investigación en la arquitectura y los componentes de los sistemas manejadores de bases de datos; dado que fue uno de los primeros sistemas efectivos concebidos, muchas de las prácticas implementadas en este sistema fueron adoptadas por la mayoría de los SMBD que surgieron posteriormente con mayores o menores modificaciones.

Este sistema fue un parte aguas en el desarrollo de maneras efectivas para administrar bases de datos relacionales, y aunque su origen se remonta más de 30 años atrás en la historia, algunas de sus propuestas originales han permanecido prácticamente intactas desde entonces.

La descripción original de System R aparece en [5]. La descripción explícita del catálogo del sistema es muy breve. Textualmente, en la sección *Data Control Facilities*, se puede leer lo siguiente:

"*El RDS (Relational Data System) automáticamente mantiene un conjunto de relaciones del catálogo que describen las relaciones, vistas, imágenes, ligas, aserciones, y disparadores conocidos por el sistema. Cada usuario puede acceder a un conjunto de vistas de los catálogos del sistema que contienen información de los elementos que le pertenecen. El acceso a las relaciones del catálogo se realiza exactamente de la misma manera a como se realiza el acceso a cualquier relación (i.e. por medio de una consulta SEQUEL*[1]*). Por supuesto, ningún usuario está autorizado a modificar el contenido de un catálogo directamente, pero cualquier usuario autorizado puede modificar el catálogo indirectamente mediante acciones tales como crear una tabla. Además, un usuario puede agregar comentarios para cada una de las entradas del catálogo que le son visibles por medio de la sentencia* `COMMENT`".

En el párrafo anterior podemos apreciar algunos puntos importantes. En primer lugar, se menciona que el catálogo contiene información respecto a relaciones, vistas, imágenes, ligas, aserciones, y disparadores, lo cual es consistente con lo expuesto por Codd.

Con respecto a las relaciones, sin embargo, como es descrito en la sección Data Definition Facilities, "*Para cada campo de la nueva relación, el nombre del campo y su tipo de dato son especificados. Si así se desea, también puede ser especificado si son permitidos los valores nulos o no en uno o más de los campos*". Se puede observar que en esta descripción, no se menciona nada acerca de llaves primarias ni foráneas sobre las tablas. Se puede encontrar una referencia acerca de valores únicos en la misma

[1] Recuérdese que el nombre con el que se conocía originalmente al lenguaje relacional por excelencia era SEQUEL, y posteriormente se cambió su nombre a SQL.

sección: "*Una imagen puede ser declarada como* `UNIQUE`*, lo cual forza a cada combinación de valores de los sort field a ser única en la relación*", sin embargo, esta especificación da por hecho que un conjunto de campos son utilizados como *sort fields*, lo cuál contradice lo expuesto por Codd, quien especifica que no es necesaria la creación de un índice para poder establecer a un conjunto de campos como llave primaria. Más adelante veremos que, en la mayoría de los casos, un índice está relacionado a elementos que involucran pruebas de unicidad, como las llaves primarias y las llaves únicas.

El conjunto de aserciones y disparadores bien pudiera ser utilizado para especificar acciones como las que se pueden especificar en las restricciones de integridad referencial, sin embargo, no existen elementos en el catálogo que den soporte explícito para este tipo de restricciones. Es claro ver que este apartado hace referencia a las restricciones definidas por el usuario descritas por Codd.

En la sección *Data Definition Facilities* también podemos encontrar lo siguiente con respecto a las vistas: "*El poder de SEQUEL puede ser utilizado para definir una vista como una relación derivada de una o más relaciones. Esta vista puede entonces ser utilizada de la misma forma que una relación base... La sentencia SEQUEL que define una vista es guardada en un catálogo del sistema donde puede ser examinada por usuarios autorizados.*", lo cuál es consistente con lo expuesto por Codd.

Nótese que el soporte para *UDFs* no es descrito como parte de las especificaciones de System R, elemento requerido en las reglas de Codd.

No se darán detalles en la información acerca de imágenes y ligas, pues estos elementos no forman parte de los elementos que hacen de un SMBD un SMBDR.

Si bien System R fue ideado tomando en cuenta el enfoque relacional, este no puede ser identificado como un SMBDR, pues ni su catálogo ni el sistema manejador en sí, cumplen con algunos de los puntos básicos de las reglas de Codd.

El soporte para algunas de las características relacionales parece ser más complicado que el soporte para otras, por ejemplo las restricciones de integridad referencial, y esa es seguramente una de las razones por las cuales algunos SMBD, como es el caso de MySQL, han tardado más en implementarlas que otros.

En algunas otras ocasiones, como será expuesto en el capítulo 5, se ha omitido la implementación de algunas características por motivos de eficiencia.

I. 8. Resumen del capítulo

En este se estudiaron las características generales que los postulados del Modelo Relacional expuestos ampliamente en [11] por Codd exigen del catálogo de un SMBDR. Se explicó el por qué de las características más relevantes y se analizó cuáles de estas son adecuadamente implementadas por los SMBDR que en este trabajo se estudian.

En las secciones 2 a 6 se describe, como se hace en [11], la información que debe ser almacenada en el catálogo de un SMBDR, que es la concerniente a dominios, relaciones y vistas, la referente a las restricciones de integridad referencial y *check constraints*, la referente a las *UDFs*, las características de seguridad y rendimiento, y qué debe expresar tal información, así como las características del acceso a la misma.

En la sección 7 se exponen las características más relevantes del catálogo de *System R*, y cómo dicho catálogo implementa las especificaciones de Codd, así como la influencia que estas implementaciones tuvo sobre el diseño de los SMBDR actuales.

II. El LDM y las estadísticas: teoría y uso

Se ha estudiado la información acerca del modelo lógico de los datos que debe ser almacenada en los catálogos de los SMBDR. En este capítulo se analizará el por qué dicha información es requerida dentro del catálogo, así como los datos estadísticos que son almacenados en el mismo y la manera en que estos son utilizados.

II. 1. El Modelo Relacional

La teoría que sustenta al modelo lógico de datos mediante el cual es representada la información en los SMBDR es el *Modelo Relacional*. Fue propuesto en 1970 por Edgar Frank Codd, y está basado en la lógica de primer orden, así como en la teoría de conjuntos, ambas teorías muy sólidas y con un amplio estudio que las respalda.

Cuando Codd creó el modelo relacional, uno de sus objetivos era que fuera un modelo suficientemente simple para que aún los usuarios ocasionales tuvieran ante ellos una representación comprensible de la información almacenada. Es por esto que describir a grandes rasgos los elementos más importantes del modelo relacional es muy sencillo.

Es bien sabido que el modelo lógico de datos es utilizado para estructurar datos que por si solos no son capaces de generar información. Se estructuran los datos también para facilitar su comprensión y para proteger su integridad lógica, así como para facilitar las tareas de mantenimiento de los mismos y proveer de facilidades para establecer un sistema consistente y robusto de seguridad y recuperación.

En el Modelo Relacional solamente hay una estructura, que es la *relación n-aria*. Conceptualmente, una relación es representada por una tabla con un número fijo de *atributos* (columnas) y un número variable de *tuplas* (renglones) únicamente identificadas de manera asociativa, es decir, solamente por su contenido, haciendo completamente irrelevante el orden en que estas se encuentran almacenadas. Cada columna de una relación contiene valores que provienen de un *dominio* de valores

posibles para esa columna en particular. Una *llave primaria* de una relación es un atributo o conjunto de atributos que identifican a cada tupla del resto de ellas, es decir, la identifican de manera única [19]. Así mismo, una llave foránea consiste de una o más columnas que obtienen sus valores del dominio (simple o compuesto), sobre el que al menos una llave primaria se encuentra definida [11].

Sabiendo esto es claro por qué Codd describe al catálogo de la forma en la que lo hace. Para que el catálogo cumpla las necesidades bajo las cuales fue planteado, este debe poder describir aquellas entidades que el Modelo Relacional especifica. Son precisamente las relaciones, los atributos, los dominios, las llaves primarias, las foráneas y el tratamiento de información faltante (valores nulos) los elementos que **LDMExtractor** es capaz de extraer del catálogo pues, como podemos ver, son estos los elementos más importantes especificados en el Modelo Relacional.

II. 2. Las estadísticas

Se ha estudiado que los dos datos estadísticos requeridos en el catálogo, según las reglas de Codd, son el número de tuplas de cada una de las relaciones y el número de valores distintos para cada una de las columnas, pero no se ha explicado hasta el momento por qué es importante tener esta información a la mano.

Es bien sabido que una de las operaciones más importantes en el álgebra relacional (y por tanto en SQL) es la operación de `JOIN`. Esta operación es necesaria cuando la información de alguna entidad semántica se encuentra distribuida en un número grande de relaciones, generalmente porque la base de datos se encuentra normalizada. Supongamos, por ejemplo, que nos interesa guardar la información de un estudiante, los grupos en los que se encuentra inscrito y los profesores de dichos grupos. Una forma de guardar dicha información podría involucrar a las siguientes relaciones:

```
persona(curp, nombre, apellidop, apellidom, direccion)

alumno(curp, cuenta, año_ingreso, id_carrera, id_plantel)

carrera(id_carrera, nombre)

plantel(id_plantel, nombre, dirección)

carrera_plantel(id_carrera, id_plantel)

alumno_grupo(id_alumno, id_grupo)

grupo(clave, id_materia, id_profesor)

materia(id_materia, nombre, id_carrera)

profesor(curp, rfc, grado_estudios)
```

Si se deseara obtener la información completa de un alumno, su carrera, el plantel en el que estudia, los grupos en los que se encuentra inscrito, las materias de dichos cursos y los profesores que las imparten, sería necesaria una consulta como la siguiente:

```
SELECT *
FROM persona
     NATURAL JOIN alumno
     NATURAL JOIN carrera
     NATURAL JOIN carrera_plantel
     NATURAL JOIN plantel
     NATURAL JOIN alumno_grupo
     NATURAL JOIN grupo
     NATURAL JOIN materia
     NATURAL JOIN profesor
```

En la consulta podemos ver ocho operaciones de tipo `JOIN`, el cual, como veremos más adelante, es un número a considerar, y más si la cardinalidad de cada una de las relaciones es grande.

Es claro que el ejemplo anterior es un tanto exagerado para el marco en el que se presenta. Sin embargo, un número grande de operaciones de tipo `JOIN` es muy común en almacenes de datos o *data warehouses*, que generalmente presentan esquemas de tipo estrella, copo de nieve o galaxia [2] y que para poner la información a disposición de las herramientas de minería de datos u OLAP, se requiere generar tablas a partir de la información contenida en muchas de estas.

II. 2. 1. Conceptos en optimización de consultas

Se ha dicho ya que las estadísticas juegan un papel crucial en los SMBD, razón por la cuál **LDMExtractor** es capaz de extraerlas de los catálogos para poder ser estudiadas por el usuario. Su uso es extenso e involucra una cantidad considerable de elementos, razón por la cual será necesario profundizar en diversos conceptos para su correcta comprensión.

Para entender la importancia de dichas estadísticas en el procesamiento de las consultas, es importante comprender el proceso conocido como *Optimización de Consultas*. Sin embargo, dicho proceso involucra una serie de aspectos que es necesario analizar para su completa comprensión. La mayoría de estos conceptos surgieron con la creación de *System R* y se han mantenido prácticamente sin cambio desde que surgieron.

Durante el resto de la sección y por cuestiones de facilidad de exposición, se considerarán solamente consultas de tipo *select-project-join* o SPJ a menos que se especifique algo distinto.

Cuando se habla del *espacio de búsqueda* en la optimización de consultas, se habla de todas las expresiones algebraicas para las cuales, dada una consulta en SQL, el resultado es equivalente. Por ejemplo, el espacio de búsqueda de *System R* está conformado por árboles que corresponden a secuencias lineales de operaciones de tipo

JOIN. Si consideramos, por ejemplo, la secuencia JOIN(JOIN(JOIN(A,B),C),D), una manera lineal de representarla sería la mostrada en la Figura 1. Tales secuencias son lógicamente equivalentes gracias a las propiedades asociativas y conmutativas de la operación JOIN [9].

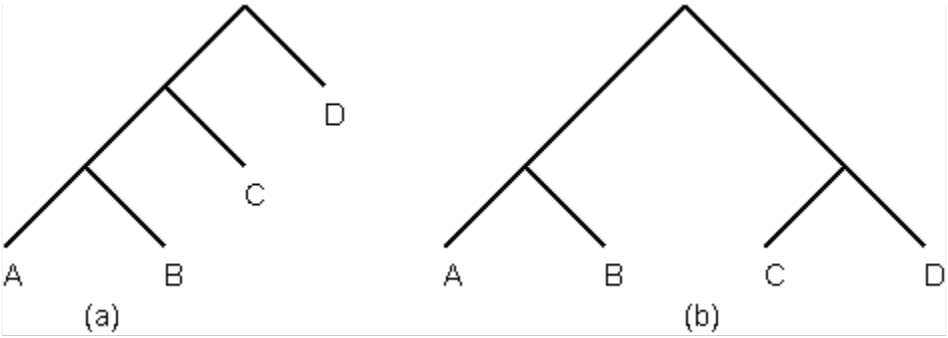

Figura 1. JOIN (a) lineal y (b) con estructura de arbusto.

El *modelo de costos* dentro del contexto de este trabajo asigna un costo estimado a cualquier plan de ejecución existente en el espacio de búsqueda, ya sea parcial o completo. También determina el tamaño estimado de los resultados obtenidos de cada una de las operaciones de los planes. El módulo encargado de obtener los costos de los planes en el espacio de búsqueda depende de tres elementos importantes:

- Un *conjunto de estadísticas* acerca de las relaciones y los índices, como número de páginas en una relación, número de páginas en un índice, número de valores distintos en una columna.
- Un *conjunto de fórmulas* para estimar la *selectividad de los predicados* y para *proyectar el tamaño de los resultados* para cada operador. Por ejemplo, el tamaño del resultado de una operación JOIN es estimado multiplicando el tamaño de dos tablas y aplicando la selectividad de todos los predicados especificados.
- Un *conjunto de fórmulas* para estimar los *costos de tiempo de procesamiento* y de E/S de una consulta para cada operador. Estas fórmulas deben tomar en cuenta las propiedades estadísticas de sus parámetros de entrada, los métodos de acceso disponibles, y cualquier orden disponible, pues ciertos algoritmos, como *sort-merge join*, son considerablemente más eficientes cuando los datos de entrada se encuentran ordenados. También se toma en cuenta si los datos en el resultado tendrán o no algún orden.

Finalmente, este módulo utiliza los mencionados elementos para asociar el tamaño del resultado, algún orden que pudiera este tener y el costo estimado del operador en cuestión, así como el costo parcial del plan hasta el momento, a un plan de ejecución, realizando los cálculos desde los niveles más bajos del árbol a los más altos.

Los algoritmos utilizados por los optimizadores de consultas pueden entonces utilizar *programación dinámica*. La idea central del enfoque de la programación

dinámica en el optimizador de consultas es *asumir* que el modelo de costos antes descrito cumple con el *principio de optimidad*.

Proposición II-1. (*Principio de optimidad.*)
*Sea **Q** una consulta de tipo* SPJ *compuesta por k joins y sea **P** un plan de ejecución para **Q**. **P** es un plan de ejecución óptimo si y sólo si todos los planes de ejecución para subexpresiones de **Q** que consten de k-1 joins son a su vez planes de ejecución óptimos para dichas subexpresiones.*

No es materia del presente trabajo demostrar la validez del principio de optimidad en cada uno de los distintos SMBD estudiados dado que, dependiendo del enfoque de sus diversas implementaciones, este principio puede o no ser válido para los planes de ejecución que componen sus respectivos espacios de búsqueda, pero el conocerlo permite tener una mejor noción de las opciones que tienen los SMBD para reducir el espacio de búsqueda adecuadamente, además de permitir una mejor comprensión del concepto de propiedad física, enunciado más adelante.

Lo que el principio de optimidad sugiere es que para obtener un plan de ejecución óptimo basta considerar los planes de ejecución óptimos de tamaño *k-1* y agregar a estos un join más o, lo que es lo mismo, no es necesario considerar planes subóptimos de tamaño *k-1* pues no es posible generar planes de ejecución óptimos de tamaño *k* a partir de ellos.

Así, la enumeración basada en programación dinámica ve a una consulta SPJ como un conjunto de relaciones $R_1,\ldots,R_n$ que serán reunidas. En el paso *j*, el algoritmo ha producido planes óptimos para todas las subconsultas con *j* joins.

Para obtener el plan de ejecución óptimo de una consulta de tamaño *(j+1)*, consideramos todas las posibles formas de construir un plan de ejecución para la subconsulta extendiendo los planes construidos en el paso *j*.

Ejemplo II-1.
*El plan de ejecución óptimo para la consulta **R_1,R_2,R_3,R_4** es obtenido tomando en cuenta el plan con el menor costo estimado de los siguientes posibles planes:*

$JOIN(R_1,R_2,R_3,R_4)$
$JOIN(R_1,R_2,R_4,R_3)$
$JOIN(R_1,R_3,R_4,R_2)$
$JOIN(R_2,R_3,R_4,R_1)$

una vez elegido el plan con menor costo, el resto de los planes es ignorado.

La programación dinámica es notablemente más eficiente que la búsqueda ingenua. En lugar de listar *O(n!)* planes utilizando la fuerza bruta, sólo se requiere listar $O(n2^{n-1})$ planes con programación dinámica. Aún así, un espacio de búsqueda de tamaño exponencial es aún demasiado grande para poder llevarlo a la práctica, reflejando una enorme necesidad de reducir aún más el espacio de búsqueda.

Considérese ahora la consulta que representa a la reunión de R_1,R_2,R_3. Supóngase que los costos del plan para la subconsulta R_1,R_2 son *x* y *y* utilizando *nested-loop* y *sort-merge* respectivamente, y que $x<y$. En tal caso, mientras consideramos el plan

para R_1,R_2,R_3, no consideraremos el plan en el que R_1 y R_2 son reunidos utilizando *sort-merge*. Sin embargo, es importante notar que si utilizamos *sort-merge* para reunir R_1 y R_2, el resultado de dicha reunión se encuentra ordenado, lo que provocaría que la reunión de dicho resultado con R_3 fuera mucho más eficiente si se vuelve a utilizar el mismo algoritmo. La capacidad del optimizador de consultas para no despreciar estos casos es el enfoque de *órdenes interesantes*. Este enfoque es generalizado en los SMBD al concepto de *propiedad física*.

Definición II-1. (*Propiedad física.*)
*Sea **Q** una expresión lógica de una consulta y sean **P_1** y **P_2** dos planes de ejecución cualesquiera para **Q**. Una propiedad física p de **P_1** es una característica de **P_1** que no aparece en **P_2** y que impacta en el costo de operaciones subsecuentes.*

II. 2. 2. Optimización de consultas

El volumen de datos que los Sistemas Manejadores de Bases de Datos administran ha crecido considerablemente desde que los primeros SMBD surgieron. La creación de mejores métodos para la captura de datos, así como la creciente necesidad de los usuarios (expertos y ocasionales) de la explotación de los mismos, ha provocado que el requerimiento de respuestas rápidas por parte de los SMBD a una consulta o a un número grande de estas no sólo no haya desaparecido, sino que se haya vuelto un problema cada vez más difícil de enfrentar.

El enfoque básico de optimización de consultas se encuentra fuertemente basado en el uso de estadísticas para realizar las estimaciones del costo de los planes de ejecución encontrados en el espacio de búsqueda. Sin embargo, muchos más aspectos pueden (y deben, en la mayoría de los casos) ser tomados en cuenta para realizar la complicada tarea de mejorar el tiempo de respuesta de los SMBD a las consultas.

Podemos clasificar a los enfoques para la optimización de consultas por el nivel en el que estos son aplicados:

- *Algoritmo*: aquí encontramos aquellas optimizaciones que son implementadas directamente sobre los algoritmos que llevan a cabo las operaciones descritas en el plan de ejecución.
- *Procesamiento de consulta*: en esta categoría encontramos aquellas optimizaciones que son realizadas sobre el plan de ejecución y su estructura. También se encuentran aquí aquellas optimizaciones que tienen que ver con realizar alguna tarea de preprocesamiento en los datos para mejorar el rendimiento de alguna operación sobre ellos.

Las optimizaciones sobre los algoritmos, originalmente las más complicadas de implementar pues requerían de la modificación del código de los optimizadores de consultas, procuran mejorar el rendimiento de las operaciones que representan los operadores presentes en el plan de ejecución. A este nivel se da por hecho que se está trabajando con un plan de ejecución óptimo, pues los algoritmos no pueden siquiera conocer el plan de ejecución que los invoca.

Actualmente se pueden llevar acabo optimizaciones en los algoritmos mediante *UDFs* (*User Defined Functions*), cuyo uso se ha popularizado gracias a que cada vez más SMBD soportan su utilización. Cada uno de los SMBD tiene un formato y restricciones específicas para estas funciones. Para más información en UDFs puede referirse a la documentación del SMBD de interés.

Las optimizaciones sobre los algoritmos generalmente son consideradas como una opción cuando se realiza un tipo particular de consultas sobre los datos o cuando el ambiente en el que se llevan a cabo las consultas tiene características y problemas específicos a razón de que, como es mostrado en [8], muchas veces el problema a tratar no puede ser optimizado e incluso puede no ser eficientemente aproximado tanto como se desearía.

Una de las primeras optimizaciones de este tipo fue la utilización de algoritmos que aprovechan los índices existentes sobre algún atributo para llevar a cabo JOINs, como es el caso de sort-merge join o hash-join.

En [24] los autores presentan las limitadas capacidades de los SMBD para realizar análisis estadístico multidimencional, *machine learning* o minería de datos, que requieren de manipulación de vectores y matrices. Así mismo, presentan una solución a estas limitaciones desarrollando UDFs, mostrando mejorar considerablemente las capacidades de un SMBD para responder a este tipo particular de consultas, que explotan exhaustivamente a las agregaciones. Un aspecto particularmente interesante de este trabajo es la manera en cómo fue medido el desempeño de las mejoras propuestas.

Al referirse a optimizaciones en el procesamiento de la consulta se está haciendo referencia a las optimizaciones que tienen que ver con el listado y la elección de planes de ejecución óptimos de manera eficiente y acertada, así como el preprocesamiento de datos para mejorar el desempeño de algún operador en particular. En general, los SMBD realizan esta tarea por medio de un optimizador de consultas que basa sus tareas en el uso de estadísticas. Sin embargo, algunas otras ideas pueden ser utilizadas para reducir el tiempo que le toma al SMBD determinar cuál de los posibles planes de ejecución es el mejor a seguir o para aumentar la calidad de la elección de dicho plan.

Un excelente ejemplo de esto es el enfoque tomado por el optimizador de consultas de PostgreSQL [18]. Aunque el núcleo del optimizador está básicamente construido para considerar las estadísticas del sistema contenidas en el catálogo, el optimizador de consultas tiene un módulo extra que está basado en un método heurístico de búsqueda basado en algoritmos genéticos.

El *algoritmo genético* (*GA* por sus siglas en inglés) es un método heurístico de optimización que realiza una búsqueda no determinista aleatorizada. El conjunto de posibles soluciones para el problema de optimización es interpretado como *los individuos de la población*. El grado de "adaptación" de un individuo a su ambiente (qué tanto mejor resulta un individuo como solución al problema) es especificado por su nivel de adaptación o *fitness*. Las características específicas de un individuo son expresadas por sus *cromosomas* (en este caso en particular, una cadena de caracteres), quienes también representan las "coordenadas" de un individuo en el espacio de búsqueda. Un *gen* es una subsección de cromosomas que representa un parámetro a optimizar. Mediante la simulación de operadores de evolución como *recombinación*, *mutación* y *selección*, nuevas *generaciones* de individuos cada vez más

adaptados son creadas a partir de los individuos existentes, siempre propagando las características de los individuos mejor adaptados, de manera que, después de un número determinado de generaciones, se espera que todos o la gran mayoría de los individuos, representen al espécimen mejor adaptado, es decir, la solución buscada.

Otras soluciones heurísticas pueden ser aplicadas para encontrar el plan de ejecución óptimo dentro del espacio de búsqueda pero, como se menciona en [9], también es necesario ampliar adecuada y eficientemente dicho espacio. Particularmente hablando de System R, el espacio de búsqueda que le es posible generar es muy limitado, pues la única manera que tiene de ampliarlo es mediante intercambios en el orden en que son realizados los joins. En el mismo texto, el autor expone algunas otras transformaciones algebraicas utilizadas en los sistemas actuales que pueden ser tomadas en cuenta para generar más planes de ejecución dada una expresión específica.

Para obtener más información acerca de los métodos de optimización de consultas, un buen compendio de técnicas de procesamiento de consultas puede ser encontrado en [17]. En el caso de las técnicas modernas es tan basto el trabajo que se ha realizado en el área que es complicado citar alguno o algunos en particular.

II. 2. 3. Optimización automática de consultas

La optimización automática de consultas se basa en el fundamento algebraico de SQL, y parte de la idea de que una expresión que representa a una consulta puede ser manipulada mediante operaciones algebraicas hasta otra expresión que es equivalente a la expresión original en el sentido en que ambas devuelven el mismo resultado al ser evaluadas.

Si bien el resultado de evaluar un par de expresiones algebraicas equivalentes será el mismo, el tiempo que le toma al SMBD evaluar cada una de ellas puede ser tan distinto que puede haber incluso varios órdenes de magnitud entre uno y otro, por lo cual un usuario desearía encontrar siempre aquella expresión que requiera menos tiempo de respuesta.

Es obvio que para poder elegir la expresión con mejor tiempo de respuesta lo ideal sería conocer dichos tiempos de antemano, pero para tener en nuestras manos dichos tiempos sería necesario evaluar cada una de las expresiones, cosa que claramente no es posible, razón por la cual existen módulos en todos los SMBD que en lugar de evaluar todas y cada una de las expresiones, estiman el tiempo que le tomará dar una respuesta para dicha expresión.

La forma más recurrida para calcular las ya mencionadas estimaciones es mediante el uso de estadísticas acerca de los datos con los que se está trabajando. Aunque las estimaciones realizadas por los SMBD en algunos casos son bastante buenas, en algunos otros, particularmente en aquellos donde se encuentra involucrada la operación `JOIN`, dichas estimaciones pueden ser excesivamente malas (decenas o incluso centenas de veces más lentas que las optimas), pues es necesario estimar no sólo el tamaño de las tablas iniciales (también llamadas tablas base) sino de las tablas que representan resultados parciales de la consulta o de las subconsultas realizadas.

Esto inmediatamente nos lleva a la idea de que es de suma importancia que las estadísticas mantenidas por el SMBD en el catálogo sean lo más realistas posibles.

Un optimizador de consultas cualquiera que utiliza el enfoque estadístico para elegir el plan de ejecución correspondiente a una consulta dada cumple con varias características generales que se encuentran presentes incluso desde System R. A grandes rasgos y de manera muy general, el proceso de optimización de consultas se puede dividir en dos tareas principales: reescritura y planificación [21].

La reescritura consiste en tomar una consulta proporcionada por el usuario y aplicar sobre ella transformaciones que dependen únicamente de las características declarativas o estáticas de la consulta, buscando mejorar su eficiencia esperada sin tomar en cuenta los costos específicos de dicha consulta en el SMBD que la procesa.

Es, sin embargo, la etapa de planificación la que nos resulta más interesante. El planificador explora el espacio de búsqueda determinado por el *Espacio Algebraico* y por el *Espacio de Estructuras y Métodos* utilizando una estrategia de búsqueda, calcula el costo estimado en base al *Modelo de Costos* y a la estimación de *Distribución y Tamaño* de cada uno de los planes generados y elige entre ellos al de menor costo.

En esta serie de procesos hay varias consideraciones importantes. Cómo se obtiene el espacio de búsqueda a partir del algebraico y del de estructuras y métodos de acceso es una de las cuestiones importantes que cada SMBDR afronta de manera distinta, pero como la construcción de dicho espacio no tiene que ver con la utilización de estadísticas del sistema, no será cubierta por el presente trabajo. Sin embargo, es importante analizar lo que sucede con el modelo de costos.

Para poder asignar un costo a cada uno de los planes existentes en el espacio de búsqueda, el modelo de costos calcula dicho valor tomando en cuenta las estadísticas, los métodos de acceso y la implementación de los algoritmos a utilizar.

La información estadística es un conjunto de datos guardados referentes a las relaciones e índices y existen de manera perdurable como parte del catálogo del SMBD, justo como Codd explica en la especificación del modelo relacional. Algunos de estos datos son generales, algunos otros para relaciones e índices específicos.

La unidad de costo por excelencia es el número de páginas leídas del disco duro. Esto quiere decir que para poder considerar el tiempo que tarda el CPU en procesar cada una de las tuplas, se requiere convertir el tiempo de procesador en lecturas de página. Para lograr esto, el catálogo o el diccionario de los SMBDR normalmente cuenta con una serie de factores que son utilizados en las fórmulas aritméticas correspondientes para realizar esta transformación. Los factores de conversión son los mismos para todas las consultas.

Es común que un usuario encuentre en los SMBD la opción de modificar los parámetros del factor de conversión de tiempo de procesador a lectura de páginas. Una mala elección de estos factores puede llegar a alterar las estimaciones realizadas por el planificador, quien podría hacer malas elecciones si un usuario inexperto los modifica de manera imprudente.

Los parámetros del modelo de costos obtenidos a partir de la información estadística están previamente definidos y son generales para cualquier modelo lógico de datos, lo que permite utilizarlos para todos y cada uno de los procesos de optimización sin distinción, pero también limita al optimizador al no permitirle considerar datos estadísticos específicos para una relación o un índice que podrían ser de vital importancia para encontrar el plan óptimo.

La actualización de estos datos es, en la mayoría de los casos, una tarea que debe ser llevada a cabo manualmente de manera periódica mediante un programa de mantenimiento, por lo que generalmente la información disponible para la optimización se encuentra caduca, es decir, no se encuentra actualizada; en algunos casos no es posible determinar qué tan vieja o nueva es la información estadística disponible. Para mantenerla al día, pueden existir dos tipos de herramientas presentes en los SMBDR: las estimativas y las precisas.

Al actualizar la información por medio de herramientas estimativas se sabe que las estadísticas serán actuales. Sin embargo, como el número calculado no fue obtenido tomando en cuenta todos y cada uno de los elementos presentes en la relación, sino que generalmente es calculado tomando una muestra aleatoria de la relación o índice en cuestión (es un número estimado), la información que representa no es completamente confiable. Un ejemplo de este tipo de herramientas es la sentencia `ANALYZE` que presentan PostgreSQL y MySQL[1].

Al actualizar los datos estadísticos con una herramienta precisa, se garantiza que la información es completamente confiable y que ésta se encuentra al día al momento de finalizar los procesos concernientes al cálculo de los datos. El problema con las herramientas precisas es que sus procesos son extremadamente costosos y tardados, tan costos como realizar un escaneo secuencial tupla por tupla sobre todas las tablas de la base de datos o aquellas que se indiquen. Recordemos que los datos en una base de datos se encuentran, en muchos casos, en constante cambio (bases de datos transaccionales), y en otros los datos son extremadamente numerosos (*data warehouses*), por lo que, para poder mantener al día los datos estadísticos acerca de ellos, sería necesario ejecutar una y otra vez las tareas de actualización precisa, lo cual resulta extremadamente costoso.

Para poder calcular el tamaño y la distribución de los elementos de una relación, la mayor parte de los optimizadores de consultas utilizan histogramas, que generalmente son creados dividiendo el dominio de un atributo en *cubetas* (*buckets* en inglés) que consideran que la distribución de los distintos valores de dicho atributo en estas cubetas es uniforme. Así pues, dichas cubetas son creadas de forma equidistante, de manera que cada una de ellas abarca un intervalo del dominio del atributo del mismo tamaño que el del resto de ellas.

Sin importar cuál sea la elección para la distribución a considerar, siempre que esta sea fija, habrá muchas ocasiones en que la distribución real de los distintos valores del atributo no sea la considerada por el SMBD y, en algunos casos, estar completamente errada, provocando serias deficiencias en el cálculo de los tamaños estimados de los resultados.

Esta es la principal razón por la que **LDMExtractor** presenta, siempre que el SMBD lo permite, un medio de consulta de las estadísticas, pues al tenerlas a mano, es más sencillo darse cuenta de cuándo es necesario llevar a cabo tareas de mantenimiento en las mismas, tareas en las que **LDMExtractor** puede ser útil.

[1] Solamente algunos de los sistemas de almacenamiento de MySQL soportan la actualización de estadísticas mediante este comando, ver la documentación para más información.

II. 3. Resumen del capítulo

En este capítulo se estudió el marco teórico sobre el cuál se basan los conceptos del modelo lógico de datos y del uso de estadísticas para la optimización de consultas.

Se estudió cómo Codd planteó la necesidad de un modelo sencillo pero robusto para la representación del modelo de los datos que eran almacenados en bases de datos.

Así mismo, se analizó la necesidad de proponer métodos que permitan realizar consultas cuyo costo (i.e. tiempo de respuesta) sea el menor entre las consultas equivalentes, así como la forma de encontrar estas últimas.

Finalmente, se describe de manera amplia y detallada la opción de utilizar estadísticas acerca de los elementos almacenados en los SMBD para realizar optimizaciones en los planes de ejecución de las consultas realizadas, ya sea de manera automática o utilizando la información estadística para plantear mejor las consultas desde el momento de su concepción. Se analizan conceptos como el *Principio de Optimidad*, la *Programación Dinámica*, los *Órdenes Interesantes* y las *Propiedades Físicas*, que son conceptos que, ya sea por su trascendencia histórica o por su utilidad actual, son sumamente importantes en la optimización de consultas. Se estudió también a qué niveles puede realizarse dicha optimización, y cómo se reflejan todas estas necesidades en lo especificado por Codd.

III. Descripción de los catálogos

En este capítulo se analizará de manera general la estructura de los catálogos del sistema. Se estudiará primeramente el *Information Schema*, que forma parte del estándar de SQL, y serán analizados también los diccionarios del sistema, siempre que estos participen en la reconstrucción del LDM o en la obtención de estadísticas.

III. 1. Breve historia del estándar

SQL se ha convertido en el lenguaje de consulta relacional más popular. El nombre SQL es una abreviatura de *Structured Query Language* (Lenguaje de Consulta Estructurado). En 1974 Donald Chamberlain y otros definieron el lenguaje SEQUEL (*Structured English Query Language*) en IBM Research [19]. Este lenguaje fue implementado inicialmente en un prototipo de IBM llamado SEQUEL-XRM en 1974-75. En 1976-77 se definió una revisión de SEQUEL llamada SEQUEL/2 y el nombre se cambió a SQL.

IBM desarrolló posteriormente System R. System R implementó un amplio subconjunto de SEQUEL/2 (llamado ahora SQL) y numerosos cambios que se le hicieron a SQL durante el proyecto. System R se instaló en diversos equipos de usuarios seleccionados, tanto internamente en IBM como exteriormente. Gracias al éxito y aceptación de System R en los mismos, IBM inició el desarrollo de productos comerciales que implementaban el lenguaje SQL basado en la tecnología de System R.

Durante los años siguientes, IBM y bastantes otros vendedores anunciaron productos SQL tales como SQL/DS (IBM), DB2 (IBM), ORACLE (Oracle Corp.), DG/SQL (Data General Corp.), SYBASE (Sybase Inc.) e Ingres (Ingres Corp., originalmente desarrollado por la universidad de California, en Berkeley, como un proyecto de software libre).

SQL es el estándar oficial actualmente. En 1982, la *American National Standards Institute* (ANSI) encargó a su Comité de Bases de Datos X3H2 el desarrollo de una propuesta de lenguaje relacional estándar. Esta propuesta fue ratificada en 1986 y consistía básicamente en el dialecto de IBM de SQL. En 1987, el estándar ANSI fue también aceptado por la Organización Internacional de Estandarización (ISO). Esta

versión recibió informalmente el nombre de *SQL/86*. En 1989, el estándar original fue extendido, y recibió el nuevo nombre, también informal, de *SQL/89*.

Los comités ISO y ANSI trabajaron durante muchos años en la definición de una versión muy ampliada del estándar original, llamado informalmente *SQL2* o *SQL/92*. Esta versión se convirtió en un estándar ratificado durante 1992: International Standard ISO/IEC 9075:1992, Database Language SQL. Se da una descripción detallada de SQL/92 en [14]. Al mismo tiempo, se desarrolla un nuevo estándar denominado informalmente como *SQL3* o *SQL/99*. Se plantea hacer de SQL un lenguaje de alcance completo (i.e. *Turing-complete language*), de manera que sean posibles todas las consultas computables, por ejemplo consultas recursivas.

Posteriormente se actualiza la definición de SQL/99 y se escribe SQL:2003, que es la quinta revisión del estándar, la cuál es, hasta el 2007, la última revisión. Esta revisión hace relativamente pocas modificaciones a las partes de SQL/99 y oficialmente incluye algunas nuevas características, tales como son las relacionadas con XML, *window functions*, el generador de secuencias que permite secuencias estandarizadas, dos tipos de columna nuevos: valores autogenerados y columnas de identidad, la sentencia MERGE, extensiones a la sentencia `CREATE TABLE` para permitir `CREATE TABLE AS` y `CREATE TABLE LIKE`, y la remoción de los pobremente implementados tipos de datos BIT y BIT VARYING.

III. 2. Information Schema según el estándar

La información contenida en esta sección es incluida como se encuentra en [4] y en [3]. El principal objetivo de exponerla en el presente trabajo es con motivos de comparación, pues en las siguientes secciones mostraremos cómo es que los SMBD que se estudian implementan lo expuesto en el estándar.

Los elementos del *Information Schema* son, en su mayoría, vistas definidas en términos de las tablas base del *Definition Schema*. El único propósito del *Definition Schema* es proveer de un modelo de datos para dar soporte al Information Schema y para apoyar el entendimiento. Una implementación de SQL no necesita más que simular la existencia del Definition Schema a través de las vistas del Information Schema.

Las vistas del Information Schema están definidas como si formaran parte de un esquema llamado `INFORMATION_SCHEMA`, permitiendo que sean accedidas en la misma forma en que se accede a cualquier otra tabla de cualquier otro esquema. La operación `SELECT` sobre la mayoría de estas vistas es permitida mediante el permiso `PUBLIC WITH GRANT OPTION`, de tal forma que puedan ser consultadas por cualquier usuario y para que el privilegio de `SELECT` pueda ser posteriormente otorgado a vistas que hagan referencia a estas. No se debe otorgar ningún otro permiso, de tal forma que no puedan ser actualizadas.

El *Apéndice A: Elementos del Information Schema* presenta todos los elementos que conforman el Information Schema acompañados de su función. Al estudiar detenidamente la tabla de dicho Apéndice podemos notar que aquellos elementos que

no se encuentran en SQL/99 son, en su mayoría, elementos que dan soporte a las nuevas características de SQL, mencionadas en la sección anterior. Si se requieren las definiciones en lenguaje SQL o se desea ampliar la información acerca de estos elementos, puede referirse a [4] y a [3].

En las siguientes secciones se estudiarán las implementaciones del Information Schema en los tres SMBD estudiados por el presente trabajo.

III. 3. MySQL

En esta sección analizaremos la implementación del Information Schema que podemos encontrar en MySQL v5.0.

Como se describe en [1], la implementación del *Information Schema* es una "novedad" para este popular SMBD, que no había implementado esta característica del estándar en sus versiones anteriores, razón por la cual, como podremos observar, es una implementación aún en etapa de desarrollo con importantes carencias y fallas. El estándar principal sobre el que se encuentra basada esta implementación es SQL:2003.

La Tabla III-1 muestra una breve descripción de las tablas que conforman el Information Schema en MySQL. En dicha tabla se indica si la vista se encuentra implementada parcialmente (P), de manera completa o casi completa (C), no forma parte del estándar (NE) o no representa correctamente la información descrita en el mismo (NR).

Nombre del Catálogo	Estándar	Descripción
CHARACTER_SET	P	Carente de columnas que hacen referencia a estructuras no existentes en MySQL, así como a una columna que debe contener el repertorio de caracteres, dos columnas no estándar presentes
COLLATIONS	P	Únicamente presente el nombre del filtro y cinco columnas no estándar
COLLATION_CHARACTER_SET_APPLICABILITY	C	Como el Estándar
COLUMN_PRIVILEGES	C	Como el Estándar
COLUMNS	C	Como el Estándar
KEY_COLUMN_USAGE	C	Como el Estándar
ROUTINES	P	Múltiples columnas faltantes, columnas que hacen referencia a elementos no implementados en esta versión presentes, tres columnas no estándar presentes
SCHEMATA	C	Como el Estándar
SCHEMA_PRIVILEGES	NE	Privilegios de acceso a nivel base de datos
STATISTICS	NE	Información acerca de los índices de las tablas

TABLES	NR	Muchas características de las tablas de MySQL que aparecen en esta vista hacen referencia a elementos que no forman parte del estándar, mientras que algunos elementos estándar no aparecen
TABLE_CONSTRAINTS	C	Como el Estándar
TABLE_PRIVILEGES	C	Como el Estándar
TRIGGERS	C	Como el Estándar
USER_PRIVILEGES	NE	Información acerca de privilegios globales
VIEWS	C	Como el Estándar

Tabla III-1. Information Schema en MySQL, solamente conformado por vistas.

El Information Schema según el estándar contiene 61 tablas, mientras que la implementación de MySQL solamente presenta 16. La principal razón de esta enorme diferencia radica en el hecho de que MySQL no implementa, ni directa ni indirectamente, muchas de las características especificadas por el estándar en general, no solamente del Information Schema. Evidentemente no tiene sentido proporcionar vistas o columnas en estas que hagan referencia a información sobre elementos que no existen ni siquiera parcialmente (como más adelante veremos que sucede con PostgreSQL).

En la tabla se considera este hecho, y aunque a algunas de las vistas que fueron marcadas como *C* se encuentran carentes de una muy buena parte de las columnas especificadas por el estándar, los elementos que describen las columnas existentes son correctamente descritos según lo permiten las características del SMBD.

MySQL es, históricamente, uno de los **SMBD** que más ha tardado en adaptarse a los estándares (paradójicamente siendo uno de los *Open Source* más populares) por lo que muchas de las estructuras presentes en algunos otros proyectos más maduros no se encuentran presentes aquí o han sido recientemente incorporadas, como es el caso de la integridad referencial.

III. 4. PostgreSQL

En esta sección analizaremos la implementación del Information Schema que podemos encontrar en PostgreSQL v8.1, así como los catálogos de PostgreSQL, ya que estos participan en la reconstrucción del LDM.

El Information Schema se encuentra implementado en PostgreSQL por un esquema denominado `information_schema`, que existe de manera automática en todas las bases de datos que se crean. Por defecto, los elementos del Information Schema no se encuentran disponibles en la ruta de búsqueda, por lo que los accesos a éste tienen que ser mediante los nombre calificados (del tipo `information_schema.element`).

La Tabla III-2 muestra los elementos de la implementación del Information Schema presentes en PostgreSQL. En dicha tabla se indica si los elementos presentes se

encuentran implementados parcialmente (P) o de manera completa o casi completa (C).

Nombre del Catálogo	Estándar	Descripción
INFORMATION_SCHEMA	C	Como el Estándar
CARDINAL_NUMER	C	Como el Estándar
CHARACTER_DATA	C	Como el Estándar
SQL_IDENTIFIER	C	Como el Estándar
TIME_STAMP	C	Como el Estándar
ADMINISTRABLE_ROLE_ AUTHORIZATIONS	C	Como el Estándar
APLICABLE_ROLES	C	Como el Estándar
ATTRIBUTES	C	Como el Estándar
CHECK_CONSTRAINTS_ ROUTINE_USAGE	C	Como el Estándar
CHECK_CONSTRAINTS	C	Como el Estándar
COLUMN_DOMAIN_USAGE	C	Como el Estándar
COLUMN_PRIVILEGES	C	Como el Estándar
COLUMN_UDT_USAGE	C	Como el Estándar
COLUMNS	C	Como el Estándar
CONSTRAINT_COLUMN_ USAGE	C	Como el Estándar
CONSTRAINT_TABLE_ USAGE	C	Como el Estándar
DATA_TYPE_PRIVILEGES	C	Como el Estándar
DOMAIN_CONSTRAINTS	C	Como el Estándar
DOMAIN_UDT_USAGE	C	Como el Estándar
DOMAINS	P	La vista se encuentra carente de referencias a los conjuntos de caracteres, los *collations* y a otros elementos, de los cuales también la vista COLUMNS carece
ELEMENT_TYPES	P	Carente de elementos comúnmente no presentes en las vistas de esta implementación, así como valores nulos por hacer referencia a características no aplicables a este elemento
ENABLED_ROLES	C	Como el Estándar
KEY_COLUMN_USAGE	C	Como el Estándar
PARAMETERS	P	Carente de elementos comúnmente no presentes en las vistas de esta implementación, así como valores nulos por hacer referencia a características no aplicables a este elemento
REFERENTIAL_ CONSTRAINTS	C	Como el Estándar
ROLE_COLUMN_GRANTS	C	Como el Estándar

ROLE_ROUTINE_GRANTS	C	Como el Estándar
ROLE_TABLE_GRANTS	C	Como el Estándar
ROLE_USAGE_GRANTS	C	Como el Estándar
ROUTINE_PRIVILEGES	C	Como el Estándar
ROUTINES	P	Carente principalmente de referencias a elementos que no se encuentran implementados en el SMBDR
SCHEMATA	C	La vista se encuentra carente de referencias a los conjuntos de caracteres, los *collations* y algunos otros elementos
SECUENCES	C	Como el Estándar
SQL_FEATURES	C	Como el Estándar
SQL_IMPLEMENTATION_INFO	C	Como el Estándar
SQL_LANGUAGES	C	Como el Estándar
SQL_PACKAGES	C	Como el Estándar
SQL_PARTS	C	Como el Estándar
SQL_SIZING	C	Como el Estándar
SQL_SIZING_PROFILES	C	Como el Estándar
TABLE_CONSTRAINTS	C	Como el Estándar
TABLE_PRIVILEGES	C	Como el Estándar
TABLES	P	Carente principalmente de referencias a elementos que no se encuentran implementados en el SMBD
TRIGGERS	C	Como el Estándar
USAGE_PRIVILEGES	C	Como el Estándar
VIEW_COLUMN_USAGE	C	Como el Estándar
VIEW_ROUTINE_USAGE	C	Como el Estándar
VIEW_TABLE_USAGE	C	Como el Estándar
VIEWS	C	Como el Estándar

Tabla III-2. Information Schema en PostgreSQL, incluidos los dominios especificados por el estándar.

El Information Schema según el estándar contiene 61 tablas, mientras que la implementación de PostgreSQL presenta 45, todas ellas pertenecientes al estándar.

A diferencia de lo que sucede en MySQL, PostgreSQL ha procurado mantener su implementación del Information Schema lo más apegada al estándar que le es posible dados los elementos implementados por el SMBDR, poniendo a disposición todas las columnas de todas las vistas que implementa, aún cuando estas en ocasiones contengan valores nulos, solamente para hacer consistentes las consultas que se realicen sobre estas, es por esto que hemos sido un poco más estrictos con la calificación de la calidad de la implementación, dando P a las vistas que contienen un número grande de características ya sea no implementadas o que hacen referencia a elementos no presentes en el sistema.

En la documentación [18] se menciona que en muchas ocasiones, se ha optado por no seguir el estándar, pues algunos elementos son implementados de maneras muy

distintas a como es especificado en este. Esto, obviamente, se ve reflejado en las vistas con las implementaciones más pobres.

Un detalle interesante es que en la descripción de la vista `sql_features` y algunas otras se especifica que el equipo de desarrollo de PostgreSQL no realiza pruebas formales acerca de la conformidad con las características del estándar implementadas.

Algunos otros detalles importantes que podemos hacer notar refieren al hecho de que algunos elementos no tienen nombres únicos con respecto a la base de datos, sino a alguna otra entidad a la que se encuentran subordinados. Tal es el caso de las llaves foráneas, donde el nombre de una llave foránea puede estar presente múltiples ocasiones dentro del catálogo, pues el nombre de estas solamente es único respecto a la tabla que la contiene, no así al esquema y mucho menos a la base de datos. Esto es lo que, al menos en este caso, hace necesaria la participación del diccionario de datos en las tareas realizadas por **LDMExtractor**.

Como se describe en la sección 2 del capítulo 1 del presente trabajo, los SMBDR cuentan con un diccionario de datos además del catálogo. En el caso de PostgreSQL, el diccionario de datos es indispensable para integrar correctamente la información contenida en las diversas vistas del *Information Schema*.

En la Tabla III-3 se muestra una breve descripción de los elementos del diccionario disponibles en *PostgreSQL*.

Nombre del diccionario	Descripción
`pg_aggregate`	Las funciones de agregación disponibles
`pg_am`	Los métodos de acceso disponibles
`pg_amop`	Los operadores asociados con alguna clase de operadores de métodos de acceso específica
`pg_amproc`	Procedimientos de apoyo para los operadores de los métodos de acceso
`pg_attrdef`	Los valores por defecto para una columna en particular
`pg_attribute`	Cada una de las columnas de una tabla para todas las tablas
`pg_authid`	Los roles definidos en el sistema
`pg_auth_members`	Relaciones entre los distintos roles definidos
`pg_autovacuum`	Parámetros de configuración de *Autovacuum* para cada tabla
`pg_cast`	Información acerca de como llevar a cabo conversiones entre tipos de datos
`pg_class`	Información acerca de las tablas y la gran mayoría de las cosas que se asemejan a las tablas, como índices, secuencias, vistas, tipos compuestos y algunos tipos de relaciones especiales
`pg_constraint`	Las restricciones de integridad de las tablas, tales como llaves primarias, únicas y foráneas
`pg_conversion`	Procedimientos de conversión de codificación disponibles
`pg_database`	Información de las bases de datos disponibles en este *cluster*

pg_depend	Dependencias entre los distintos objetos de una base de datos
pg_description	Información adicional y comentarios acerca de los objetos de una base de datos
pg_index	Información adicional acerca de los índices, el resto se encuentra en pg_class
pg_inherits	Jerarquía de herencia en las tablas
pg_language	Lenguajes disponibles para escribir funciones o procedimientos
pg_largeobject	Objetos grandes partidos en páginas para ser guardados en renglones aquí
pg_listener	Soporte para los comandos LISTEN y NOTIFY
pg_namespace	Guarda *namespaces*. Cada namespace puede tener un conjunto de relaciones, tipo, etc., sin crear conflictos con sus nombres
pg_opclass	Contiene clases de operadores de métodos de acceso
pg_operator	Guarda operadores en general
pg_pltemplate	Provee plantillas para la creación de lenguajes procedurales de manera sencilla
pg_proc	Información sobre funciones o procedimientos
pg_rewrite	Reglas de reescritura de una consulta para una tabla. Corresponde al mencionado módulo de reescritura
pg_shdepend	Como pg_depend pero para objetos compartidos (*shared objects*)
pg_statistic	Guarda información estadística acerca de los datos contenidos en la base de datos. Sus tuplas son generadas por ANALIZE y utilizadas por el planificador del optimizador
pg_tablespace	Contiene espacios de tablas que pueden ser utilizados para ayudar a mantener orden en la organización física de las tablas en el disco duro
pg_trigger	Información acerca de los *triggers*
pg_type	Información acerca de los tipo

Tabla III-3. Los diccionarios de PostgreSQL.

III. 5. SQL Server

En esta sección analizaremos la implementación del Information Schema que podemos encontrar en Microsoft SQL Server 2005, como es especificado en [23].

El *Information Schema* se encuentra implementado en SQL Server por un esquema denominado information_schema. Al igual que en PostgreSQL, los elementos del Information Schema no se encuentran disponibles en la ruta de búsqueda por defecto, por lo que los accesos a éste tienen que ser mediante los nombre calificados (del tipo information_schema.element).

La Tabla III-4 muestra los elementos de la implementación del Information Schema presentes en SQL Server 2005. En dicha tabla se indica si la vista se encuentra implementada parcialmente (P), de manera completa o casi completa (C), no forma parte del estándar (NE) o no representa correctamente la información descrita en la misma (NR).

Nombre del Catálogo	Estándar	Descripción
CHECK_CONSTRAINTS	C	Como el Estándar
COLUMN_DOMAIN_USAGE	C	Como el Estándar
COLUMN_PRIVILEGES	C	Como el Estándar
COLUMNS	P	Algunas características (ej. columnas identidad) implementadas en SQL Server no reflejan su presencia en su implementación del Information Schema, algunas otras no están presentes por hacer referencia a elementos no implementados
CONSTRAINT_COLUMN_ USAGE	C	Como el Estándar
CONSTRAINT_TABLE_ USAGE	C	Como el Estándar
DOMAIN_CONSTRAINTS	C	Como el Estándar
DOMAINS	C	Como el Estándar
KEY_COLUMN_USAGE	C	Como el Estándar
PARAMETERS	C	Como el Estándar
REFERENTIAL_ CONSTRAINTS	C	Como el Estándar
ROUTINES	NR	Muchas columnas con valores NULL
ROUTINE_COLUMNS	NE	Vista que identifica las columnas que son devueltas por alguna rutina
SCHEMATA	C	Como el Estándar
TABLE_CONSTRAINTS	C	Como el Estándar
TABLE_PRIVILEGES	C	Como el Estándar
TABLES	P	Carente principalmente de referencias a elementos que no se encuentran implementados en el SMBD
VIEW_COLUMN_USAGE	C	Como el Estándar
VIEW_TABLE_USAGE	C	Como el Estándar
VIEWS	C	Como el Estándar

Tabla III-4. Information Schema en Microsoft SQL Server, conformado solamente por vistas.

El enfoque del Information Schema en SQL Server es muy similar al de MySQL, aunque en algunos casos presenta similitudes con PostgreSQL. Aquí, las columnas que hacen referencia a elementos no presentes en el SMBD algunas veces no aparecen en las vistas del Information Schema. Las veces que estos elementos aparecen, a veces se hace referencia al hecho de que estos campos serán posteriormente utilizados,

mientras que en otras ocasiones se especifica que dicha característica no es aplicable al elemento y, según parece, no será implementada.

Dado que en el caso de SQL Server también haremos uso de algunos catálogos del sistema, es necesario describir aquel conjunto de estos al que pertenecen las vistas que requeriremos más adelante. Si bien es cierto que haremos uso de un pequeño subconjunto de los elementos descritos a continuación, se incluye una breve descripción de aquellos que no haremos uso porque presentan, al igual que en PostgreSQL, una forma alternativa de acceder a la información de los LDM de las bases de datos que el presente trabajo trata. La Tabla III-5 muestra una descripción de estos catálogos. Para obtener más información puede consultarse la documentación en línea de SQL Server [23].

Nombre del diccionario	Descripción
`sys.allocation_units`	Las unidades de almacenamiento (*allocation unit*) en la base de datos
`sys.assembly_modules`	Las funciones, procedimientos o disparadores que se encuentre definidos por un *common language runtime assembly*
`sys.check_constraints`	Los objetos en la base de datos que representan a un *check constraint*
`sys.columns`	Las columnas que pertenecen a un objeto con columnas, como es el caso de tablas, vistas, algunos tipos de funciones, tablas internas o tablas de sistema
`sys.computed_columns`	Las columnas cuyos valores son calculados por el sistema
`sys.default_constraints`	Los objetos en la base de datos que representan la definición de un valor por defecto
`sys.events`	Los eventos para los cuales alguna notificación o *trigger* se dispara
`sys.event_notifications`	Los objetos en la base de datos que representan a alguna notificación de evento
`sys.extended_procedures`	Los objetos que representan a un *extended stored procedure* de SQL Server
`sys.foreign_keys`	Las llaves foráneas
`sys.foreign_key_columns`	Las columnas que participan en alguna restricción de tipo llave foránea
`sys.fulltext_indexes`	Los índices de texto completo existentes
`sys.fulltext_index_columns`	Las columnas que participan en algún índice de texto completo
`sys.identity_columns`	Las *identity columns* presentes en la base de datos
`sys.indexes`	Los índices definidos sobre objetos tabulares en la base de datos
`sys.index_columns`	Las columnas que forman parte de un índice
`sys.key_constraints`	Las llaves primarias y únicas definidas en la base de datos
`sys.numbered_procedures`	Los *stored procedures* que fueron definidos como

	procedimientos numerados
sys.numbered_procedure_parameters	Los parámetros de los *stored procedures* numerados
sys.objects	Contiene todos los objetos definidos por el usuario subordinados a un esquema
sys.parameters	Los parámetros de todos aquellos objetos que aceptan parámetros
sys.partitions	Las particiones de todas las tablas e índices de la base de datos
sys.procedures	Los objetos que representan a un procedimiento de algún tipo
sys.service_queues	Los objetos en la base de datos que representan a una cola de servicio de SQL Server
sys.sql_dependencies	Las dependencias sobre entidades independientes referenciadas en una expresión SQL o sentencia que define a alguna entidad referenciante dependiente
sys.stats	Estadísticas de los objetos tabulares presentes en la base de datos
sys.stats_columns	Las columnas que son parte de una estadística presente en sys.stats
sys.sql_modules	Los objetos que representan un módulo definido mediante SQL
sys.synonyms	Sinónimos para los diversos objetos de la base de datos
sys.tables	Las tablas de la base de datos
sys.views	Las vistas de la base de datos
sys.triggers	Los disparadores presentes en la base de datos
sys.trigger_events	Los eventos a los cuales está asociado un *trigger*
Sys.traces	Los "rastreos" (*traces*) que realiza el sistema al momento de hacer la consulta

Tabla III-5. Los diccionarios de SQL Server.

III. 6. Resumen del capítulo

En este capítulo se describe de manera breve el elemento del estándar de SQL conocido como Information Schema. También se realizó un resumen de toda la información referente a las implementaciones del Information Schema en los tres SMBD que se estudian en el presente trabajo, así como los diccionarios de aquellos en los que dicho elemento es relevante para las tareas de **LDMExtractor**.

IV. El modelo lógico de datos

Se ha estudiado en los capítulos anteriores que el catálogo contiene las descripciones de todos los elementos que conforman el LDM de las bases de datos contenidas en SMBDR. El contenido de este capítulo se enfoca en las partes del *Information Schema* y/o del diccionario, según sea el caso, que hacen referencia a los elementos estructurales básicos de la base de datos: tablas, columnas y tipos de datos, y las restricciones de integridad: tratamiento de nulos, integridad referencial y *check constraints*. Se analiza al conjunto de consultas en lenguaje SQL que son necesarias para reconstruir cada uno de estos elementos de una base de datos, pues es precisamente éste el utilizado por **LDMExtractor** para crear las representaciones abstractas de cada uno de los elementos presentes en el catálogo.

El lector podrá observar que en algunas de las consultas se puede cambiar el enfoque de contención de conjuntos a uno que utilice la operación de JOIN, pero dado el número reducido de registros que en la gran mayoría de los casos estará involucrado en los resultados de dichas consultas, siempre que se ha podido se ha optado por la primera opción para facilitar la exposición de las ideas que representa cada una de las consultas.

IV. 1. MySQL

Para estudiar la reconstrucción del LDM de las bases de datos almacenadas en MySQL, se dividirá dicho estudio en dos partes; primeramente, el análisis del LDM a través del catálogo con las tablas de la base de datos.

La vista del *Information Schema* que contiene la información referente a las tablas de la base de datos y sus columnas es `COLUMNS`. Para poder reconstruir una tabla, únicamente requerimos saber su nombre, el nombre de sus columnas y el tipo de dato de cada una de ellas. La Figura 2 muestra la sentencia SQL que extrae la información requerida. Nótese que, dada la estructura de la vista, en esta misma consulta es posible extraer la información acerca de una de las restricciones importantes, que es el tratamiento de valores nulos.

```
SELECT TABLE_NAME, COLUMN_NAME, COLUMN_DEFAULT,
       DATA_TYPE, IS_NULLABLE
FROM COLUMNS
WHERE TABLE_SCHEMA = '<base de datos>'
      AND TABLE_NAME IN
      (Select TABLE_NAME
       From INFORMATION_SCHEMA.TABLES
       Where TABLE_TYPE = 'BASE TABLE')
```

Figura 2. Sentencia SQL para extraer la información básica de la tabla y sus campos en MySQL.

El siguiente paso es extraer la información acerca de las restricciones de la base de datos. La vista del Information Schema que contiene la información de las restricciones es KEY_COLUMN_USAGE. Es importante recalcar que la versión 5.0 de MySQL no soporta *check constraints*, por lo que solamente se analizarán las llaves primarias, únicas y las restricciones de integridad referencial. La Figura 3 muestra la sentencia SQL que extrae la información requerida.

```
SELECT CONSTRAINT_NAME, TABLE_NAME,
       COLUMN_NAME, REFERENCED_TABLE_SCHEMA,
       REFERENCED_TABLE_NAME, REFERENCED_COLUMN_NAME
FROM KEY_COLUMN_USAGE
WHERE CONSTRAINT_SCHEMA = '<base de datos>'
```

Figura 3. Sentencia SQL para extraer información acerca de las restricciones en MySQL.

Una forma de extraer toda la información del LDM en una sola consulta es la mostrada en la Figura 4. Dicha consulta reúne los elementos acerca de las tablas, las columnas, llaves primarias, llaves únicas, llaves foráneas y tratamiento de valores nulos.

```
SELECT COLUMNS.TABLE_NAME, COLUMNS.COLUMN_NAME,
       COLUMN_DEFAULT, DATA_TYPE, IS_NULLABLE,
       CONSTRAINT_NAME, REFERENCED_TABLE_NAME,
       REFERENCED_COLUMN_NAME
FROM INFORMATION_SCHEMA.COLUMNS
     LEFT OUTER JOIN
     INFORMATION_SCHEMA.KEY_COLUMN_USAGE ON
     (COLUMNS.TABLE_NAME = KEY_COLUMN_USAGE.TABLE_NAME AND
      COLUMNS.COLUMN_NAME = KEY_COLUMN_USAGE.COLUMN_NAME)
WHERE TABLE_SCHEMA = '<base de datos>'
```

Figura 4. Sentencia SQL para extraer toda la información acerca del LDM de una base de datos en MySQL en una sola tabla.

Obsérvese que ninguna de las consultas hace referencia a la estructura *schema*. Esto es porque MySQL no da soporte a esta estructura en absoluto y, por tanto, no tiene sentido extraer dicha información. Para MySQL, referirse a un esquema es lo mismo que referirse a una base de datos.

IV. 2. PostgreSQL

Al igual que como se hizo en el caso de MySQL, para estudiar la reconstrucción del LDM de las bases de datos almacenadas en PostgreSQL, se dividirá dicho estudio en varias partes. Comenzaremos el análisis del LDM a través del catálogo con las tablas de la base de datos.

Al igual que en el caso de MySQL, la vista del *Information Schema* que contiene la información de las tablas de la base de datos y sus columnas es `COLUMNS`. En este caso, para poder reconstruir una tabla, requerimos saber el nombre del esquema al que pertenece, el nombre de sus columnas y el tipo de dato de cada una de ellas. La Figura 5 muestra la sentencia SQL que extrae la información requerida. Nótese que, dada la estructura de la vista, en esta misma consulta es posible extraer la información acerca el tratamiento de valores nulos. Analicemos un poco dicha consulta.

Obsérvese primero que, a diferencia de lo que sucede en MySQL, PostgreSQL hace un uso extenso de la estructura *schema*, por lo que la información referente a dicha estructura es muy importante.

La primera gran diferencia se encuentra en el primer predicado de la cláusula `WHERE`. En tal se debe establecer forzosamente el nombre de la base de datos que se está analizando. Esto sucede porque el Information Schema contiene información sobre todas las bases de datos presentes en el sistema; al realizar la consulta, un usuario propietario de más de una base de datos tiene acceso a las tablas y columnas de dichas bases de datos, así que para mantener la información aislada, se requiere proporcionar el nombre de la que se desea analizar.

```
SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME,
       COLUMN_DEFAULT, DATA_TYPE, IS_NULLABLE
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_CATALOG = '<base de datos>' AND
      TABLE_NAME NOT LIKE 'pg_%' AND
      TABLE_SCHEMA <> 'information_schema' AND
      (TABLE_NAME,TABLE_SCHEMA) NOT IN (Select table_name,
                                               table_schema
                                        From
                                    information_schema.views)
```

Figura 5. Sentencia SQL para extraer la información básica de tablas y campos en PostgreSQL.

El segundo predicado es necesario para no tomar en cuenta la información contenida en el diccionario del sistema pues, como sabemos, toda la información de un SMBDR debe poder ser accedida de manera relacional; así pues, la información utilizada propiamente por el SMBD se encuentra en esquemas y tablas cuyo nombre siempre incluye el prefijo `pg_`. Es este prefijo el que se busca para no considerar una tabla. Evidentemente si una tabla de usuario contiene el mencionado prefijo no será tomada en cuenta por esta consulta, aunque los desarrolladores de PostgreSQL exhortan a los usuarios a no utilizar este prefijo en sus tablas.

El tercer predicado tiene un fin similar al anterior, y es eliminar del análisis del LDM a las vistas del Information Schema.

El cuarto y último predicado tiene que ver con una decisión en el diseño de **LDMExtractor**. Se ha optado por no considerar a las vistas definidas por el usuario como parte del LDM de la base de datos, y es precisamente este predicado el que las elimina del análisis.

El siguiente paso es extraer la información acerca de las restricciones de la base de datos. Dado que en PostgreSQL es excesivamente complicado extraer toda la información de las restricciones en una sola consulta sin generar tuplas que pervierten el significado de la información obtenida haciéndola inútil, las restricciones se analizan mediante varias consultas, una para extraer la información acerca de las llaves primarias, otra para extraer aquella que habla de las llaves únicas, una más para los *check constraints* y una última para estudiar las restricciones de integridad referencial.

La vista del Information Schema que contiene la información de las llaves primarias y únicas es `KEY_COLUMN_USAGE`. Se realiza el análisis de cada una de estas restricciones por separado pues sus características son distintas. Cada llave primaria es un conjunto de uno o más atributos, mientras que las llaves únicas siempre se aplican sobre columnas de manera individual. La Figura 6 muestra la sentencia SQL para extraer la información acerca de las llaves primarias, mientras que la Figura 7 muestra la sentencia SQL para extraer la información acerca de las llaves únicas.

Nótese cómo ambas consultas son prácticamente iguales. La única diferencia se encuentra en el predicado de la subconsulta en la última línea, donde se especifica el tipo de llave que se está buscando.

```
SELECT constraint_schema, table_name,
       constraint_name, column_name
FROM INFORMATION_SCHEMA.key_column_usage
WHERE constraint_name IN
      (Select constraint_name
       From INFORMATION_SCHEMA.table_constraints
       Where constraint_type = 'PRIMARY KEY')
```

Figura 6. Sentencia SQL para extraer información acerca de las llaves primarias en PostgreSQL.

```
SELECT constraint_schema,table_name,constraint_name,column_name
FROM INFORMATION_SCHEMA.key_column_usage
WHERE constraint_name IN (Select constraint_name
                    From INFORMATION_SCHEMA.table_constraints
                    Where constraint type = 'UNIQUE')
```

Figura 7. Sentencia SQL para extraer información acerca de las llaves únicas en PostgreSQL.

Lo siguiente es extraer la información acerca de los *check constraints*. Las vistas del *Information Schema* que contienen los datos referentes a estos elementos son `CONSTRAINT_TABLE_USAGE`, de donde se extrae la información acerca de las restricciones de tabla, `CHECK_CONSTRAINT`, conteniendo la información de los *check constraints*, y `CONSTRAINT_COLUMN_USAGE`, de donde se extrae la información acerca de las restricciones de columna.

PostgreSQL hace distinción entre *checks* de tabla y *checks* de columna (al menos en la forma en que estos dos elementos son almacenados en el Information Schema), razón por la cuál son requeridas dos consultas, mostradas en la Figura 8 y en la Figura 9, respectivamente. Sin embargo, **LDMExtractor** utiliza una unión entre los resultados de ambas consultas para realizar sus funciones, como se estudiará más adelante.

```
SELECT table_schema, table_name,
       check_constraints.constraint_name, check_clause
FROM INFORMATION_SCHEMA.constraint_table_usage JOIN
     INFORMATION_SCHEMA.check_constraints
     ON (check_constraints.constraint_catalog =
         constraint_table_usage.constraint_catalog AND
         check_constraints.constraint_schema =
         constraint_table_usage.constraint_schema AND
         check_constraints.constraint_name =
         constraint_table_usage.constraint_name)
WHERE check_constraints.constraint_catalog = '<base de datos>';
```

Figura 8. Sentencia SQL para extraer información acerca de las check constraints de tabla en PostgreSQL.

```
SELECT table_schema, table_name,
       check_constraints.constraint_name, check_clause
FROM INFORMATION_SCHEMA.constraint_column_usage JOIN
    INFORMATION_SCHEMA.check_constraints
    ON (check_constraints.constraint_catalog =
         constraint_column_usage.constraint_catalog AND
         check_constraints.constraint_schema =
          constraint_column_usage.constraint_schema AND
        check_constraints.constraint_name =
         constraint_column_usage.constraint_name)
WHERE check_constraints.constraint_catalog = '<base de datos>';
```

Figura 9. Sentencia SQL para extraer información acerca de las check constraints de columna en PostgreSQL.

Para analizar las llaves foráneas se requiere de varias vistas tanto del *Information Schema* como del diccionario de datos de PostgreSQL. En la Figura 10 puede verse la consulta que extrae la información referente a las restricciones de integridad referencial. Se procede ahora a analizar dicha consulta.

La información que contiene la tabla resultante de la consulta es el nombre de la restricción (`conname`), el nombre del esquema al que pertenece la tabla referenciante (`refing_s`), el nombre de la tabla referenciante (`refing`), el nombre de la columna referenciante (`refing_col`), el nombre del esquema al que pertenece la tabla referenciada (`refed_s`), el nombre de la tabla referenciada (`refed`) y el nombre de la columna referenciada (`refed_col`).

Las vistas que participan en la consulta son `KEY_COLUMN_USAGE` que, como ya se ha visto, contiene la información acerca de las llaves en general, `COLUMNS` que contiene la información de las columnas de las tablas, y `TABLE_CONSTRAINTS` que contiene información de las restricciones sobre las tablas, todas ellas del *Information Schema*. Por parte del diccionario del sistema de PostgreSQL, las vistas participantes son `pg_constraint` que contiene información acerca de las restricciones en general, `pg_class` que contiene información acerca de cualquier objeto en la base de datos que sea o se parezca a una tabla, y `pg_namespace` que contiene información acerca de los espacios de nombres (esquemas).

A grandes rasgos, la consulta fue pensada de la siguiente manera: se toma a `pg_constraint` para tener a la mano la información acerca de las restricciones, se reúne con `pg_class` para obtener los datos del elemento referenciante, como son su nombre, su esquema y la tabla en la que reside, se reúne nuevamente con `pg_class` para obtener los datos del elemento referenciado, se reúne también con `pg_namespace` para obtener la información del esquema del elemento referenciante y una vez más para obtener la del esquema del elemento referenciado, finalmente se reúne con `information_schema.key_column_usage` y `information_schema.columns` para obtener la información que nos permitirá discriminar a las llaves que no nos interesan, como las llaves primarias y las únicas.

Recuérdese que las llaves primarias pueden estar conformadas por conjuntos de una o más columnas. En este sentido es importante hacer notar que, dadas las características del Information Schema y del diccionario de PostgreSQL, en aquellos casos en que la llave referenciada esté conformada por más de una columna o atributo, se obtendrá en los resultados algo similar a un producto cruz para cada una de las llaves.

Es importante recalcar que este resultado no es incorrecto pues, dado que las llaves son conjuntos, el orden de sus elementos dentro de dicho conjunto no tiene relevancia, simplemente sabemos que el *conjunto* conformado por las columnas en la tabla referenciante representa a una llave foránea que apunta a otro *conjunto* de atributos de la tabla referenciada representando a la llave primaria. Así pues, como ambas llaves son conjuntos, no puede haber elementos repetidos, por lo que incluir dos o más veces un atributo a una llave no tiene ningún efecto. Más adelante se estudiará como esta idea es utilizada en la implementación de **LDMExtractor**.

```
SELECT conname,
       key_column_usage.constraint_schema AS refing_s,
       ing_info.relname AS refing,
       key_column_usage.column_name AS refing_col,
       columns.table_schema AS refed_s,
       efed_info.relname AS refed,
       columns.column_name as refed_col
FROM pg_constraint
     JOIN pg_class AS ing_info
     ON (pg_constraint.conrelid = ing_info.oid)
     JOIN pg_class AS efed_info
     ON (pg_constraint.confrelid = efed_info.oid)
     JOIN pg_namespace AS nspace_info2
     ON (efed_info.relnamespace = nspace_info2.oid)
     JOIN pg_namespace AS nspace_info
     ON (pg_constraint.connamespace = nspace_info.oid)
     JOIN
     information_schema.key_column_usage
     ON (ing_info.relname = key_column_usage.table_name
         AND nspace_info.nspname =
key_column_usage.constraint_schema
         AND conname = key_column_usage.constraint_name)
     JOIN
     information_schema.columns
     ON (efed_info.relname = columns.table_name
         AND nspace_info2.nspname = columns.table_schema
         AND columns.ordinal_position = ANY
(pg_constraint.confkey))
WHERE key_column_usage.constraint_name IN
      (Select constraint_name
       From INFORMATION_SCHEMA.table_constraints
       Where constraint_type = 'FOREIGN KEY')
```

Figura 10. Sentencia SQL para extraer información acerca de las llaves foráneas en PostgreSQL.

Ejemplo IV-1

Supóngase que se tiene una relación **R** cuya llave primaria es (**R.a**,**R.b**,**R.c**) y que la relación **S** hace referencia a **R** mediante la llave foránea conformada por (**S.a**,**S.b**,**S.c**). La tabla resultante de realizar la consulta para obtener la información de las llaves foráneas de una base de datos almacenada en PostgreSQL que contiene estas dos relaciones contendría los siguientes datos:

	refing	refing_col		refed	refed_col
…	S	a	…	R	a
…	S	a	…	R	b
…	S	a	…	R	c
		.			
		.			
		.			
…	S	c	…	R	b
…	S	c	…	R	c

IV. 3. SQL Server

Para estudiar la reconstrucción del LDM de las bases de datos almacenadas en SQL Server, como en los otros SMBD, dividiremos dicho estudio en varias partes. Comenzamos el análisis del LDM a través del catálogo de nuevo con las tablas de la base de datos.

Para hacer referencia a las vistas del Information Schema en SQL Server 2005, es necesario escribir el nombre de la base de datos que se desea consultar, seguido de `INFORMATION_SCHEMA` y luego el nombre de la vista deseada, todo separado por puntos.

Como es lo esperado, la vista que contiene la información referente a las tablas de la base de datos y sus columnas es `INFORMATION_SCHEMA.COLUMNS`. Una vez más, para reconstruir una tabla, obtendremos el nombre del esquema al que pertenece, el nombre de sus columnas y el tipo de dato de cada una de ellas, así como la información acerca el tratamiento de valores nulos. La Figura 11 muestra la sentencia SQL que extrae la información requerida. Esta consulta, así como algunas de las subsecuentes, buscan que el esquema no sea el denominado `dbo`, que es un esquema creado por el sistema. Si bien es posible crear tablas dentro del esquema `dbo`, esto no es recomendado, y se exhorta a los usuarios a crear esquemas alternos con nombres distintos. Recuérdese que algo similar sucede en PostgreSQL y sus tablas y vistas nombradas con el prefijo `pg_`, y que no podemos evitar ignorar aquellas tablas que contengan este prefijo.

Al igual que PostgreSQL, SQL Server hace un uso extenso de la estructura *schema*, por lo que la información referente a dicha estructura también es importante en este caso. La consulta utilizada es construida considerando las mismas características

utilizadas en PostgreSQL. Sin embargo, por diferencias en los dialectos de SQL que soporta cada uno de los SMBD, no se puede utilizar la misma consulta en ambos SMBD. En el caso de PostgreSQL, la consulta está construida pensando en contención de conjuntos, se busca que el nombre de la tabla no se encuentre entre los nombres de las vistas utilizando la sentencia NOT IN de SQL. En el caso de SQL Server, la consulta está basada en el uso de una reunión externa, buscando en el predicado de la sentencia WHERE aquellas tuplas donde la definición de la vista sea un valor nulo, indicando que la tupla representa a una tabla y no a una vista.

```
SELECT COLUMNS.TABLE_SCHEMA, COLUMNS.TABLE_NAME,
       COLUMNS.COLUMN_NAME, COLUMN_DEFAULT,
       DATA_TYPE, IS_NULLABLE
FROM <base de datos>.INFORMATION_SCHEMA.COLUMNS
     LEFT OUTER JOIN
     <base de datos>.INFORMATION_SCHEMA.VIEWS
     ON (COLUMNS.TABLE_SCHEMA = VIEWS.TABLE_SCHEMA
         AND COLUMNS.TABLE_NAME = VIEWS.TABLE_NAME)
WHERE COLUMNS.TABLE_SCHEMA <> 'dbo'
      AND VIEW_DEFINITION IS NULL
```

Figura 11. Sentencia SQL para extraer la información básica de tablas y campos en SQL Server 2005.

El siguiente paso es extraer la información acerca de las restricciones de la base de datos. Nuevamente nos encontramos con un escenario como el que nos plantea PostgreSQL, así que extraeremos la información de las restricciones en varias consultas.

La vista del Information Schema que contiene la información de las llaves primarias y únicas es `KEY_COLUMN_USAGE`, aunque también se utiliza la vista `TABLE_CONSTRAINTS` para poder discriminar aquellas llaves que no son primarias ni únicas. Se realiza el análisis de cada una de estas restricciones por separado por la misma razón que en PostgreSQL. La Figura 12 muestra la sentencia SQL para extraer la información acerca de las llaves primarias, mientras que la Figura 13 muestra la sentencia SQL para extraer la información acerca de las llaves únicas.

Nótese cómo, al igual que en PostgreSQL, ambas consultas son prácticamente iguales. La única diferencia se encuentra en el predicado de la subconsulta, donde se especifica el tipo de llave que se está buscando.

En este punto es importante recalcar que SQL Server tiene algunos otros elementos (como es el caso de algunos tipos de índices) que tienen características similares a las llaves únicas pero que no aparecen en la consulta que extrae la información de estas últimas. El presente trabajo no tratará de abarcar tales elementos dado que, en general, se intenta trabajar siempre con elementos que pertenecen al estándar de SQL y a lo expuesto en la definición del modelo relacional de Codd.

```
SELECT constraint_schema, table_name,
       constraint_name, column_name
FROM <base de datos>.INFORMATION_SCHEMA.key_column_usage
WHERE constraint_name IN
      (Select constraint_name
       From <base de
datos>.INFORMATION_SCHEMA.table_constraints
       Where constraint_type = 'PRIMARY KEY')
```

Figura 12. Sentencia SQL para extraer información acerca de las llaves primarias en SQL Server 2005.

```
SELECT constraint_schema, table_name,
       constraint_name, column_name
FROM <base de datos>.INFORMATION_SCHEMA.key_column_usage
WHERE constraint_name IN
      (Select constraint_name
       From <base de
datos>.INFORMATION_SCHEMA.table_constraints
       Where constraint_type = 'UNIQUE')
```

Figura 13. Sentencia SQL para extraer información acerca de las llaves únicas en SQL Server 2005.

Se procede ahora a extraer la información acerca de los *check constraints*. Las vistas del Information Schema que contienen los datos referentes a estos elementos son `CONSTRAINT_TABLE_USAGE`, de donde se extrae la información acerca de las restricciones de tabla y `CHECK_CONSTRAINT`, conteniendo la información de los `check constraints`.

PostrgreSQL almacena de manera distinta aquellos *checks* que fueron creados como de tabla y aquellos que fueron creados como de columna, no así SQL Server, de manera que la consulta mostrada en la Figura 14 es suficiente para extraer toda la información de los `check constraints`.

```
SELECT table_schema, table_name,
       check_constraints.constraint_name,
       check_clause
FROM <base de datos>.INFORMATION_SCHEMA.constraint_table_usage
     JOIN
     <base de datos>.INFORMATION_SCHEMA.check_constraints
     ON(check_constraints.constraint_catalog =
         constraint_table_usage.constraint_catalog AND
        check_constraints.constraint_schema =
         constraint_table_usage.constraint_schema AND
        check_constraints.constraint_name =
         constraint_table_usage.constraint_name)
WHERE check constraints.constraint schema <> 'dbo'
```

Figura 14. Sentencia SQL para extraer información acerca de los *check constraints* en SQL Server 2005.

El análisis de las llaves foráneas utilizando únicamente el Information Schema en este caso, al igual que en PostgreSQL, resulta imposible. La documentación del SMBD en cuestión recomienda utilizar una consulta que utiliza funciones propias de SQL Server y algunas de las vistas del sistema que se muestran en la Tabla III-5 del capítulo anterior. Dicha consulta no hace referencia a los esquemas y, como sabemos, los esquemas son un elemento importante de las bases de datos. La consulta mostrada en la Figura 15 es la utilizada por **LDMExtractor** para realizar dicho análisis. Se han realizado modificaciones menores a la presentada en la documentación para incluirla en este trabajo. Las vistas del catálogo del sistema de SQL Server que participan en dicha consulta son `sys.foreign_keys`, `sys.foreign_key_columns` y `sys.objects`.

La información que la tabla resultante de la consulta contiene es el nombre de la restricción (`conname`), el nombre del esquema al que pertenece la tabla referenciante (`refing_s`), el nombre de la tabla referenciante (`refing`), el nombre de la columna referenciante (`refing_col`), el nombre del esquema al que pertenece la tabla referenciada (`refed_s`), el nombre de la tabla referenciada (`refed`) y el nombre de la columna referenciada (`refed_col`).

Las vistas `sys.foreign_keys` y `sys.foreign_key_columns` contienen toda la información referente a las llaves foráneas. Se requiere de `sys.objects` únicamente para extraer la información referente a los esquemas de los elementos que participan en la definición de cada una de las llaves foráneas.

En la consulta podemos observar a la función `SCHEMA_NAME` que toma como parámetro el identificador de un esquema y devuelve una cadena de caracteres que representa al nombre del mismo, a la función `OBJECT_NAME` que toma como parámetro el identificador de un objeto cualquiera y devuelve una cadena de caracteres que representa a su nombre, y a la función `COL_NAME` que toma como parámetros el identificador de un objeto que representa a una tabla y el identificador de un objeto que representa a una columna y devuelve una cadena de caracteres que representa el nombre de la columna en cuestión.

Obsérvese que, en la primera línea de la consulta, se encuentra presente el comando `USE`. Dicho comando indica a SQL Server qué catálogos del sistema se están consultando y es completamente indispensable para que la consulta devuelva la información que se desea.

```
USE <base de datos>;
SELECT f.name AS conname,
       SCHEMA_NAME(refing.schema_id) AS refing_s,
       OBJECT_NAME(f.parent_object_id) AS refing,
       COL_NAME(fc.parent_object_id,
    fc.parent_column_id) AS refing_col,
       SCHEMA_NAME(refed.schema_id) AS refed_s,
       OBJECT_NAME (f.referenced_object_id) AS refed,
       COL_NAME(fc.referenced_object_id,
    fc.referenced_column_id) AS refed_col
FROM sys.foreign_keys AS f
     JOIN sys.foreign_key_columns AS fc
     ON f.object_id = fc.constraint_object_id
     JOIN sys.objects AS refing
     ON f.parent_object_id = refing.object_id
     JOIN sys.objects AS refed
     ON f.referenced_object_id = refed.object_id;
```

Figura 15. Sentencia SQL para extraer información acerca de las llaves foráneas en SQL Server 2005.

IV. 4. Resumen del capítulo

En los capítulos previos al presente se analizó el marco teórico de los elementos que **LDMExtractor** es capaz de extraer de manera automática de los SMBD que abarca el presente trabajo. En este capítulo se hace un escrupuloso análisis de las consultas en SQL que permiten a **LDMExtractor** realizar sus tareas, dedicando una sección a cada uno de los SMBD estudiados por el presente trabajo.

Se hace, al principio del presente capítulo, la observación de que, si bien es posible crear consultas, mediante el uso de la operación JOIN, representan una versión más óptima que las presentadas en este trabajo, se ha optado por el enfoque de contención de conjuntos para facilitar la exposición de las ideas bajo las cuales fueron concebidas.

Se hace énfasis en las diferencias entre la implementación de las consultas para uno y otro SMBD con el objetivo de hacer notar las diferencias intrínsecas entre estos últimos, que se ven reflejadas precisamente en dichas consultas.

V. Las estadísticas

En el capítulo dos se estudió la enorme importancia del uso de estadísticas en la optimización de consultas. En este capítulo se analiza la forma en la que MySQL, PostgreSQL y SQL Server almacenan las estadísticas que Codd establece como indispensables en un SMBD que pretenda ser relacional: el número de tuplas en cada tabla y el número de valores distintos de cada columna.

Antes de entrar plenamente en materia cabe hacer algunas observaciones. Muchos SMBD han optado por no implementar algunas de las características especificadas por los estándares y por Codd mismo por cuestiones de eficiencia o dificultades de implementación. A través de las secciones de este capítulo podremos observar que el número de valores distintos de una columna es, en la mayoría de los casos, un dato estadístico que no se encuentra disponible. Se podría pensar que esto es una carencia, pero en la perspectiva de la eficiencia no lo es.

En la actualidad muchos SMBD implementan un número grande de tipos de datos, y más aún, soportan tipos de datos definidos por el usuario. Para algunos de estos tipos, XML que es soportado por SQL Server por ejemplo, la comparación entre elementos puede resultar extremadamente costosa, y para algunos otros esta pudiera ser imposible si no existe en el SMBD un operador de comparación entre los elementos de algún dominio, razón por la cual no es factible determinar el número de valores distintos en todos los casos.

Es por esto que la mayoría de los SMBD actuales han optado por almacenar solamente información estadística acerca de aquellas columnas sobre las cuales esté definido algún índice.

Se ha pensado también que, en la mayoría de los casos, no es necesario conocer el número de valores distintos de todas las columnas, y crear índices para aquellas donde esta información sea importante, procurando que el número de estas columnas sea pequeño para no entrar en problemas de falta de espacio de almacenamiento.

También se estudiará que, en SQL Server, a pesar de la existencia de estadísticas, estas no pueden ser accedidas de manera relacional, limitando el uso que se puede dar a esta información.

V. 1. MySQL

Una vez más, es MySQL el que presenta la implementación más simple, pero también la más escasa de entre los SMBD que el presente trabajo estudia.

Las estadísticas almacenadas por MySQL sólo refieren a la cardinalidad de los índices y las tablas base, y pueden encontrarse en dos vistas del Information Schema, `STATISTICS` y `TABLES` respectivamente. Recuérdese que, para mantener un estándar en las estadísticas a analizar, se buscan el número de tuplas por tabla y el número de valores distintos por columna.

Dado que ambas estadísticas mantenidas por MySQL hacen referencia a la cardinalidad de los elementos almacenados en la base de datos, se ha elegido la información de `TABLES` para formar parte la información recabada por **LDMExtractor**. La Figura 16 muestra la consulta que extrae la cardinalidad de cada una de las tablas presentes en la base de datos.

```
SELECT TABLE_NAME, TABLE_ROWS
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = '<base de datos>'
      AND TABLE_TYPE = 'BASE TABLE'
```

Figura 16. Sentencia SQL para extraer la cardinalidad de las tablas en MySQL.

A pesar de la simplicidad de las estadísticas almacenadas por MySQL, este no deja de ser un SMBD bastante eficiente. Sin embargo, su eficiencia parece estar más basada en sus sistemas de almacenamiento que en su optimización de consultas.

Los sistemas de almacenamiento que utilizan los SMBD no son tema de este trabajo, si se requiere más información al respecto puede consultar la documentación respectiva, que abunda ampliamente en el tema.

V. 2. PostgreSQL

La implementación del Information Schema de PostgreSQL no cuenta con una tabla o vista que contenga información acerca de los datos estadísticos de la tablas o columnas. Toda la información estadística almacenada por PostgreSQL se encuentra en sus catálogos de sistema.

Dos diccionarios contienen la información que requerimos: `pg_stats` y `pg_class`. En el primero de ellos encontramos información estadística de los índices existentes en la base de datos. El segundo de ellos, como hemos mencionado, contiene información acerca de todos aquellos elementos en la base de datos que tengan columnas o se parezcan de alguna forma a una tabla.

De pg_stats se puede obtener, entre otras cosas, el número de valores distintos en cada uno de los índices. Este número no es, como se podría pensar, un número entero en todos los casos. Como se especifica en la documentación, si este número es un entero positivo, el número representa al número total de valores distintos dentro de la columna. Si el número es menor a cero, este número representa el número de valores distintos dividido entre el número de tuplas. PostgreSQL utiliza la segunda opción cuando estima que el número de valores distintos aumentará conforme la tabla crezca, mientras que la primera opción es elegida cuando el número de valores distintos parece permanecer fijo o es más estable. Los motivos que llevaron a esta forma de implementar las estadísticas son cuestiones de optimización para los algoritmos que utilizan las estadísticas. De pg_class se obtiene el número total de tuplas presentes en cada una de las tablas.

Así pues, una sola consulta basta para extraer toda la información deseada. Dicha consulta puede apreciarse en la Figura 17. La consulta involucra también a pg_namespace para obtener el esquema al que pertenecen los elementos analizados.

```
SELECT schemaname, tablename,
       attname, n_distinct,
       reltuples
FROM pg_stats JOIN pg_class
     ON (tablename = relname)
     JOIN pg_namespace
     ON (pg_class.relnamespace = pg_namespace.oid
         AND pg_stats.schemaname = nspname)
WHERE schemaname NOT LIKE 'pg_%'
      AND schemaname <> 'information schema'
```

Figura 17. Sentencia SQL para extraer las estadísticas de tablas y columnas en PostgreSQL.

Aunque se puede extraer la información de ambos datos estadísticos por separado, es mejor extraerlos en una sola consulta, porque para poder conocer el número de valores distintos para las columnas con índices, es necesario reunir a pg_class para poder obtener el esquema al que pertenecen los elementos, y es precisamente este catálogo el que contiene el número de tuplas en cada relación.

V. 3. SQL Server

SQL Server presenta una amplia variedad de opciones para crear, almacenar y administrar estadísticas para la optimización automática de consultas. Sin embargo, a pesar de implementar los métodos más completos de optimización estadística estudiados por el presente trabajo, SQL Server no presenta un método puramente relacional para consultar el estado de dichas estadísticas, esto es, no existe una

consulta en lenguaje SQL mediante la cual se pueda extraer la información de las estadísticas almacenadas por el SMBD.

Si se tiene a mano el cliente proporcionado por Microsoft, solamente se requiere ejecutar un comando. Dicho comando pertenece a una familia de comandos administrativos que se ejecutan mediante el uso de DBCC[1]. Sin embargo, para poder extraer desde una conexión externa, ya sea el número de tuplas por tabla o el número de valores distintos por columna indexada, sería necesario manejar dos de los tres conjuntos de resultados o tablas que devuelve el comando en cuestión.

El comando que extrae la información acerca de las estadísticas almacenadas en la base de datos es `DBCC SHOW_STATISTICS`, que devuelve tres conjuntos de resultados[2], el primero conteniendo características de identificación y estado general de la estadística, el segundo sobre la densidad específica de las columnas participantes en el índice y el tercero conteniendo información de los histogramas. La Figura 18 muestra un ejemplo de cómo el SQL Server Management Studio de Microsoft despliega los tres conjuntos de resultados mencionados.

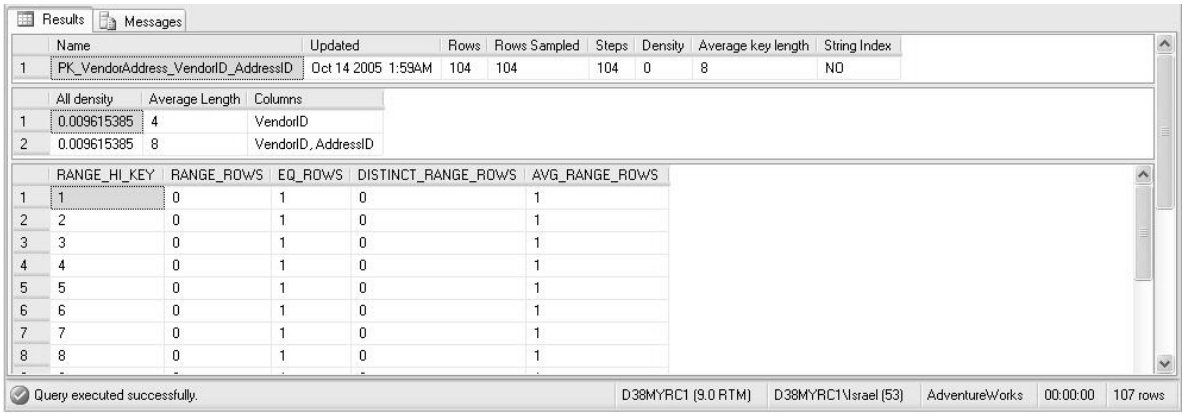

Figura 18. Los tres conjuntos de resultados que muestran la información referente a las estadísticas en SQL Server.

De estas tres tablas, la primera y la última son las que contienen los datos estadísticos especificados por Codd, el número de tuplas por tabla y el número de valores distintos por columna indexada, como hemos especificado desde el principio de este capítulo.

`DBCC SHOW_STATISTICS` recibe dos parámetros, el primero de ellos es el nombre de la tabla a la que pertenece la estadística que se está buscando en formato `<esquema>.<tabla>`, el segundo es el nombre de la estadística en cuestión.

Se ha expuesto ya que resulta muy complicado plantear una sentencia SQL mediante la cual se puedan extraer estos datos y, de hecho, no hay una serie de

[1] El lenguaje de programación Transact-SQL provee de sentencias DBCC que actúan como Comandos de Consola de Base de Datos para SQL Server 2000. Estas sentencias analizan la consistencia física y lógica de una base de datos [23].

[2] Conjuntos de resultados o *result sets* en la terminología de SQL Server, tablas en la terminología del Modelo Relacional.

operaciones relacionales que permitan extraer o crear una tabla con la información que se requiere, pero no se ha expuesto la razón de esto.

Los comandos de la familia DBCC no pueden ser ejecutados desde el interior de una sentencia SELECT de SQL, de manera que no es posible disponer de ellos para realizar múltiples operaciones. En el caso en particular del estudio que este trabajo representa, sería necesario utilizar la consulta mostrada en la Figura 19 para obtener los nombres de todas las estadísticas almacenadas en el SMBD y luego utilizar los datos de dicha consulta como parámetro para que `SHOW_STATISTICS` mostrara la información deseada, pero tal cosa no es posible.

```
SELECT SCHEMA_NAME(o.schema_id) as t_schema,
       OBJECT_NAME(o.object_id) as t_name,
       c.name AS column_name,
       s.name AS statistics_name
FROM sys.stats AS s
     JOIN sys.stats_columns AS sc
     ON (s.object_id = sc.object_id AND s.stats_id = sc.stats_id)
     JOIN sys.columns AS c
     ON (sc.object_id = c.object_id AND c.column_id =
sc.column_id)
     JOIN sys.objects AS o
     ON (sc.object_id = o.object_id)
WHERE SCHEMA NAME(o.schema id) <> 'dbo'
```

Figura 19. Sentencia SQL para extraer esquema, tabla y nombre de todas las estadísticas almacenadas en SQL Server.

Tratando de evadir estas limitantes, SQL Server ofrece la posibilidad de requerir una o dos de las tablas que devuelve `DBCC SHOW_STATISTICS` por separado mediante el uso de la sentencia `WITH` junto con uno o más de los nombres de las tablas que son devueltas, que son `STAT_HEADER`, `DENSITY_VECTOR` y `HISTOGRAM`, y más aún, permite insertar los resultados de dichas tablas en una tabla destinada a ese propósito.

Supóngase, por ejemplo, que se desea insertar en una tabla la información devuelta por la siguiente sentencia:

```
DBCC SHOW_STATISTICS ('Person.Address',
                      PK_Address_AddressID)
                      WITH STAT_HEADER
```

Dicha sentencia devuelve la primera de las tres tablas contenidas en el resultado del comando, entonces sería necesario crear una tabla con la siguiente estructura:

```
stat_headers(nombre, actualizada, tuplas, tuplas_muestra, pasos,
             densidad, tamano_prom, indice_cadena)
```

Una vez teniendo esa tabla, se tiene que ejecutar una sentencia de INSERT como la siguiente:

```
INSERT INTO stat_headers
   EXEC('DBCC SHOW_STATISTICS (''Person.Address'' ,
                               PK_Address_AddressID)
                               WITH STAT_HEADER');
```

Nótese el uso del comando EXEC, que es necesario dado que las funciones DBCC no forman parte del lenguaje relacional, sino que son consideradas un comando externo.

Esta forma de crear tablas con la información acerca de las estadísticas, si bien proporciona una forma relacional de consulta, no se presta para analizar toda la información de las estadísticas de manera eficiente, pues habría que ejecutar una sentencia INSERT por cada estadística existente en la base de datos.

Podría pensarse entonces en automatizar dichas inserciones mediante el uso de un programa de computación para crear o ejecutar las sentencias requeridas. Esta solución parece parcialmente adecuada, pero resulta que tampoco lo es. El problema radica en que los identificadores de las estadísticas son únicos solamente dentro de la tabla para la que fueron definidas, es decir, una estadística es identificada de manera única mediante el esquema de la tabla a la que pertenece, la tabla misma, y el nombre de dicha estadística.

Al observar el ejemplar expuesto en la Figura 18 se puede apreciar que el primer conjunto de resultados, `STAT_HEADER`, solamente contiene el nombre de la estadística, y no el resto de los datos necesarios para identificarla. Esto quiere decir que si se utiliza cualquier medio, automático o no, para insertar los datos en una tabla para poder acceder a ellos de manera relacional seguirá vigente el problema de que, en algunos casos (quizá pocos, pero siempre posiblemente existentes) habrá al menos una estadística que comparta nombre con otra. Se podría pensar entonces en utilizar una tabla con un valor de auto incremento para identificar a cada valor insertado y, de nuevo, apoyarse en medios externos para mantener registro de qué identificadores corresponden a qué estadísticas, pero lamentablemente esto tampoco es posible, pues el uso del comando EXEC no admite sentencias como la mostrada en la Figura 20, lo cual impide elegir a qué campos enviar la información proveniente de `SHOW_STATISTICS` y provocando errores que no permiten la ejecución de la consulta.

```
INSERT INTO stat_headers(nombre,actualizada,tuplas,
                         tuplas_muestra,pasos,densidad,
                         longitud_prom,indice_cadena) VALUES
   EXEC('DBCC SHOW_STATISTICS (“Person.Address”,
                               PK_Address_AddressID)
                               WITH STAT_HEADER');
```

Figura 20. Consulta que muestra un uso no posible del comando EXEC en SQL Server.

Así pues, al insertar los datos acerca de las estadísticas en una tabla llevando a cabo cualquiera de los procesos descritos, irremediablemente se pierde información imprescindible para su identificación, haciendo a dicha información inútil para el estudio realizado.

Es por todas estas razones que **LDMExtractor** no es capaz de extraer la información acerca de las estadísticas almacenadas por SQL Server, aunque esto no limita de ninguna manera el resto de su funcionalidad, como se estudiará más adelante.

V. 4. Resumen del capítulo

En capítulos anteriores se expuso la importancia de la información estadística que recaban los SMBD con respecto a los elementos que son almacenados en las bases de datos. En este capítulo se hace un escrupuloso análisis de las consultas en SQL que dotan a **LDMExtractor** con la capacidad de extraer dichos datos estadísticos desde los catálogos y/o diccionarios del sistema, para cada uno de los SMBD analizados en este trabajo.

Recuérdese que, al principio del capítulo anterior, se hace la observación de que, si bien es posible crear consultas que, mediante el uso de la operación JOIN, representan una versión más óptima que las presentadas en este trabajo, se ha optado por el enfoque de contención de conjuntos para facilitar la exposición de las ideas bajo las cuales fueron concebidas.

Se hace énfasis en las diferencias entre la implementación de las consultas para uno y otro SMBD con el objetivo de hacer notar las diferencias intrínsecas entre estos últimos, que se ven reflejadas precisamente en dichas consultas, diferencias que son particularmente marcadas en el contexto del almacenamiento y consulta de datos estadísticos.

VI. Implementación de la aplicación

Ya se ha analizado todo el marco teórico detrás del LDM de una base de datos y las estadísticas que los SMBD almacenan acerca de los datos que contienen, han sido presentadas las consultas que son requeridas para extraer todos los elementos que conforman el LDM almacenado por los tres SMBD que este trabajo estudia, hemos visto también la forma de extraer las estadísticas de la base de datos, así que ahora estamos listos para poner todo ese conocimiento junto y construir con él nuestra herramienta.

En este capítulo se expondrán las características de **LDMExtractor** como aplicación, se explicarán las ideas que fueron consideradas al elegir su arquitectura, así como los modelos utilizados para representar al LDM como un objeto en el lenguaje de programación Java y también como un documento XML.

Se hablará de los módulos específicos dedicados a llevar a cabo el análisis en cada uno de los distintos SMBD y cómo interactúan estos con la aplicación, explicando así también la forma correcta de crear nuevos módulos para extender el número de SMBD soportados por la aplicación.

Recuérdese que **LDMExtractor** es una herramienta que provee al usuario de una interfaz sencilla que construye una representación esquemática, clara, portable y entendible del LDM y las estadísticas más relevantes que se encuentran presentes en los catálogos de tres SMBDR: MySQL, PostgreSQL y Microsoft SQL Server. Dicha representación puede ser visible directamente en la aplicación y puede también ser almacenada en un archivo con formato XML para maximizar la portabilidad. **LDMExtractor** provee también al programador de un modelo para extender de manera sencilla la aplicación, haciéndola compatible con otros SMBD si así lo requiere.

VI. 1. Eligiendo un lenguaje de programación

En la actualidad existe una gran cantidad de lenguajes de programación que corresponden a diversos paradigmas y que tienen una cantidad enorme de similitudes y diferencias. Cuando se plantea la creación de una aplicación, siempre es importante conocer a detalle las tareas que serán llevadas a cabo por esta, el tipo de usuarios que la utilizarán y bajo qué ambientes será utilizada.

Las características esperadas de la aplicación que este trabajo expone son:

- ser capaz de conectarse a diversos SMBD mediante una interfaz común para todos ellos,
- ser modular para aislar las diferencias entre los distintos SMBD y poder así dar mantenimiento a módulos específicos para dar soporte a nuevas características en los SMBD sin tener que conocer ni modificar al resto de los módulos,
- ser fácilmente extensible para poder integrar soporte a nuevos SMBD sin necesidad de modificar el código de la aplicación,
- ser altamente portable, pues las plataformas soportadas por los SMBD son diversas,
- tener un manejo adecuado y estándar de XML,
- tener una interfaz sencilla de utilizar para el usuario y
- ser utilizada por usuarios expertos en bases de datos, y si tienen conocimientos de programación, ser extendida por ellos mismos según sus necesidades.

Una cosa clara es que es mucho más sencillo pensar en lenguajes orientados a objetos dadas las características de modularidad, mantenibilidad y extensibilidad que se esperan de la aplicación.

Así pues, se ha elegido a uno de los lenguajes orientados a objetos más robustos y populares de la estos días, que es Java, de Sun Microsystems. Nótese también que Java cuenta con una interfaz estándar para conectividad a bases de datos, que es JDBC, así como con una implementación de DOM, que es un estándar de W3C para realizar el manejo de XML.

Gracias a la extensa historia de Java, existen también numerosas herramientas para facilitar el desarrollo de aplicaciones en dicho lenguaje, como es el caso de NetBeans IDE de netbeans.org, o Eclipse de la comunidad de desarrollo que lleva el mismo nombre. Estas herramientas facilitan el desarrollo de aplicaciones, proporcionando también medios visuales para crear interfaces gráficas de usuario.

Java cumple también con la característica de portabilidad pues, al ser un lenguaje cuyos programas corren sobre la Java Virtual Machine, los programas creados en este lenguaje pueden ser ejecutados en cualquier plataforma para la cual exista dicha máquina virtual.

VI. 2. Representando el LDM

El LDM ya es un modelo abstracto de una base de datos, así que se puede utilizar la misma estructura que le fue otorgada en su definición para crear representaciones tanto en clases del lenguaje Java como en un documento XML.

Las bases de datos se encuentran organizadas en una estructura jerárquica pues se puede ver a una base de datos como una contenedor guardando tablas, que a su vez son contenedores de atributos, posiblemente una llave primaria, un conjunto de cero o más llaves foráneas, dependencias funcionales y/o *check constraints*, así como dependencias funcionales. La Figura 21 muestra un ejemplo gráfico de esta idea.

La idea intuitiva así descrita será utilizada para crear una representación en una DTD que exprese esas mismas características, y también para crear un conjunto de clases que representen las características de cada uno de los objetos, una vez que sean analizados aumentando la granularidad, partiendo desde el nivel de base de datos hasta llegar a los elementos más simples.

VI. 2. 1. Una DTD para los LDMs

Definición VI-1. (*DTD.*)

*La declaración de tipo de documento de un XML contiene o apunta a marcas declarativas (*markup declarations*) que proporcionan una gramática para una clase de documentos. A esta gramática se le conoce como* Document Type Definition (DTD)*. Dicha declaración puede apuntar a una liga externa (algún tipo de entidad externa) que contiene marcas declarativas, puede contenerlas directamente en un subconjunto interno, o ambas opciones. La DTD para un documento consiste de ambos subconjuntos considerados juntos.* [30]

Más intuitivamente, una DTD es una descripción de estructura y sintaxis de un documento XML o SGML. Su función básica es la descripción del formato de datos, para hacer uso de un formato común y mantener la consistencia entre todos los documentos que utilicen la misma DTD. Así, los documentos pueden ser validados, conocen la estructura de los elementos y la descripción de los datos que trae consigo cada documento, y pueden además compartir la misma descripción y forma de validación dentro de un grupo de trabajo que usa el mismo tipo de información.

La DTD mostrada en la Figura 22 representa el modelo lógico de datos de una base de datos relacional. Está diseñada considerando la estructura jerárquica de una base de datos descrita anteriormente. Es importante hacer notar que esta DTD contempla el novedoso concepto de *atributo foráneo* [25].

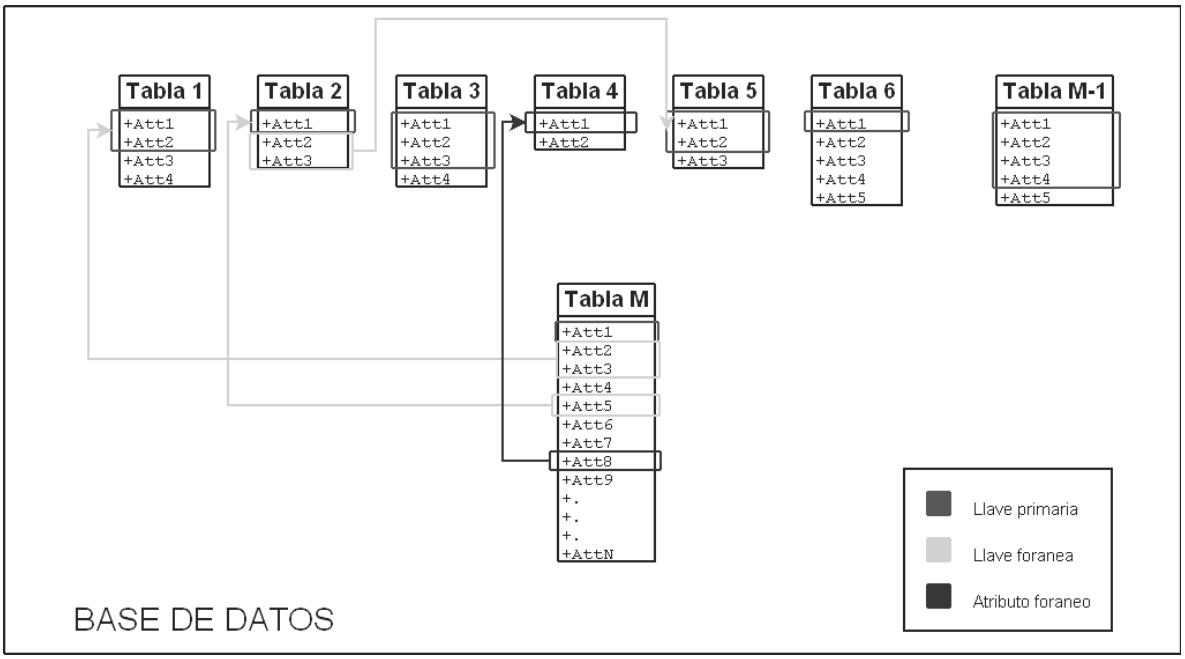

Figura 21. Ejemplo simple del LDM de una base de datos que además incluye representaciones gráficas para atributos foráneos.

Definición VI-2. (*Atributo Foráneo.*)

*Sea **R** una relación no normalizada y sea **a** un atributo en **R** que depende funcionalmente de una llave foránea **f** de **R** que referencía a la llave primaria de **S**. Se dice que **R.a** es un atributo foráneo si **a** es un atributo de **S** que depende funcionalmente de la llave primaria de **S**.*

Las tablas no normalizadas son comunes en el contexto de la integración de bases de datos de distintas fuentes que son frecuentes en los problemas de *data warehousing* y la minería de datos, o en las ocasiones en que las tablas sean *desnormalizadas* por razones de eficiencia.

En la Figura 21, la **TablaM** hace referencia a la **Tabla4** mediante el uso de un atributo foráneo, esto es, el atributo **TablaM.Att8** aparece con un valor distinto de nulo únicamente cuando existe una tupla en la **Tabla4** cuyo atributo **Tabla4.Att1** tiene ese valor.

```
<?xml version="1.1" encoding="iso-8859-1"?>
<!ELEMENT DATABASE (DBNAME,RELATION*) >
  <!ELEMENT DBNAME (#PCDATA) >
  <!ELEMENT RELATION (RNAME,ATTRIBUTE+,PK?,FK*,FA*,FUNCDEP*,
                      CHECKCONST*) >
    <!ELEMENT RNAME (#PCDATA) >
      <!ELEMENT ATTRIBUTE (ANAME,ATYPE) >
          <!ATTLIST ATTRIBUTE nullable (true|false) "false">
          <!ATTLIST ATTRIBUTE unique (true|false) "false">
        <!ELEMENT ANAME (#PCDATA) >
        <!ELEMENT ATYPE (#PCDATA) >
      <!ELEMENT PK (ANAME+) >
      <!ELEMENT FK (REFATTNAME+,RNAME,ANAME+) >
      <!ELEMENT FA (REFATTNAME,RNAME,ANAME) >
        <!ELEMENT REFATTNAME (#PCDATA) >
      <!ELEMENT CHECKCONST (#PCDATA)>
    <!ELEMENT FUNCDEP (FDETERMINANT,FDETERMINED) >
      <!ELEMENT FDETERMINANT (ANAME+) >
      <!ELEMENT FDETERMINED (ANAME+) >
```

Figura 22. DTD para el LDM de una base de datos relacional.

VI. 2. 2. La clase Database

En el paradigma de orientación a objetos, cada uno de los elementos significativos de una entidad abstracta que se desea representar debe ser descrito como una clase.

Para analizar la representación que se ha elegido para el LDM de una base de datos dentro de **LDMExtractor** se comenzará con el elemento en el nivel más alto de granularidad, que es el nivel base de datos, y posteriormente se analizarán los elementos más internos hasta llegar a las unidades más básicas.

Una base de datos es creada únicamente con su nombre y siempre está conformada por un conjunto de cero o más tablas. Así pues, para representar estos dos elementos, la clase `Database` cuenta con una variable de estado de tipo `String` para representar al nombre de la base de datos y una variable de tipo `LinkedHashSet<Table>`, que es una implementación de la entidad abstracta *conjunto* que tiene, además de las propiedades de los conjuntos, la facultad de mantener el orden en que los elementos son insertados.

El siguiente elemento a representar son las tablas de la base de datos. Una tabla es creada con su nombre y un conjunto no vacío de atributos, posiblemente una llave primaria formada por un subconjunto de atributos de la misma tabla, un conjunto de llaves foráneas y un conjunto de *check constraints*. Adicionalmente, el modelo utilizado por **LDMExtractor** contempla dos elementos más, que son las dependencias funcionales y los atributos foráneos. Además, dado el análisis de las estadísticas que es capaz de realizar **LDMExtractor**, también es contemplado el número de tuplas en la tabla para una instancia de dicha base de datos en específico. La clase `Table` cuenta

con ocho variables de estado, una de tipo `String` para almacenar el nombre de la tabla, una de tipo `LinkedHashSet<Attribute>` conteniendo a los atributos de la tabla, una más del mismo tipo para representar a la llave primaria, dos de tipo `LinkedHashSet<Reference>` que almacenan las llaves foráneas y los atributos foráneos respectivamente, una variable de tipo `LinkedHashSet<String>` para almacenar las definiciones de los *check constraints* y una última de tipo `int` para almacenar el número de tuplas en la tabla según las estadísticas.

Cada uno de los elementos que conforman a las tablas tiene a su vez una representación propia. Los atributos, siendo la unidad más fundamental, forman parte de la definición del resto de los elementos. Un atributo está formado por su nombre y su tipo o dominio. Adicionalmente se han considerado dos restricciones de integridad para formar parte de este modelo, si el atributo admite valores nulos y si sus valores deben o no ser únicos. Al igual que en el caso de las tablas, dado el análisis de las estadísticas que es capaz de realizar **LDMExtractor**, se contempla el número de valores distintos en la columna para una instancia de la base de datos representada. Entonces, la clase `Attribute` mantiene dos variables de tipo `String`, una para almacenar el nombre de un atributo y una para almacenar el tipo de dato del mismo, mantiene dos variables de tipo `boolean` para almacenar las restricciones referentes a valores nulos y a valores únicos, y una variable de tipo `int` para almacenar el número de valores distintos.

La clase utilizada para representar tanto a las llaves foráneas como a los atributos foráneos es la clase `Reference`. Tanto las llaves foráneas como los atributos foráneos son referencias a elementos ajenos a las tablas donde se encuentran definidos, así pues una referencia está conformada por dos conjuntos de atributos de la misma cardinalidad, uno en una tabla a la que se le denomina referenciante y el otro en una tabla a la que se le denomina referenciada. Un atributo foráneo en el contexto de la aplicación no es más que un caso particular de referencia, i.e. de la clase `Reference`, donde la cardinalidad de ambos conjuntos está obligada a ser igual a uno. Las referencias siempre se encuentran definidas en la tabla referenciante y deben contar con el nombre de la tabla referenciada. La clase `Reference` tiene dos variables de tipo `LinkedHashSet<Attribute>` para almacenar a los atributos referenciantes y referenciados, una de tipo `Table`, que es un apuntador a la tabla referenciada, y una variable de tipo `String` para almacenar el nombre de la referencia pues, en los SMBD, las referencias se encuentran nombradas en la mayoría de los casos.

Finalmente, la clase utilizada para representar a las dependencias funcionales es `FuncDep` que mantiene, al igual que la clase `Reference` dos variables de tipo `LinkedHashSet<Attribute>` para almacenar el conjunto de atributos determinante y el determinado. Aunque esta clase es similar a `Reference`, vale la pena crear una clase distinta para no almacenar referencias inútiles. En este caso, no es relevante el nombre de la tabla referenciada, pues siempre es la misma que la referenciante. Las dependencias funcionales no forman parte de las estructuras que son almacenadas en los SMBD, pero se le ha dado soporte a esta estructura para responder a las necesidades de implementación de otra aplicación conocida como Refined [25] que se encuentra actualmente en proceso de desarrollo.

En la Figura 23 se puede apreciar un diagrama de clases completo que muestra a las clases descritas junto con sus operaciones e interacciones; al paquete que contiene a dichas clases se le ha denominado `ldmbuilder.database`.

VI. 3. La interfaz LDMBuilder

Una vez que se han creado representaciones adecuadas para el LDM de una base de datos, se debe pensar en cómo acceder a la información de los catálogos, cómo procesarla, cómo construir con esta información instancias de las clases antes mencionadas para, finalmente, mostrarlas al usuario en la Interfaz Gráfica de Usuario, mejor conocida como *GUI* por sus siglas en inglés.

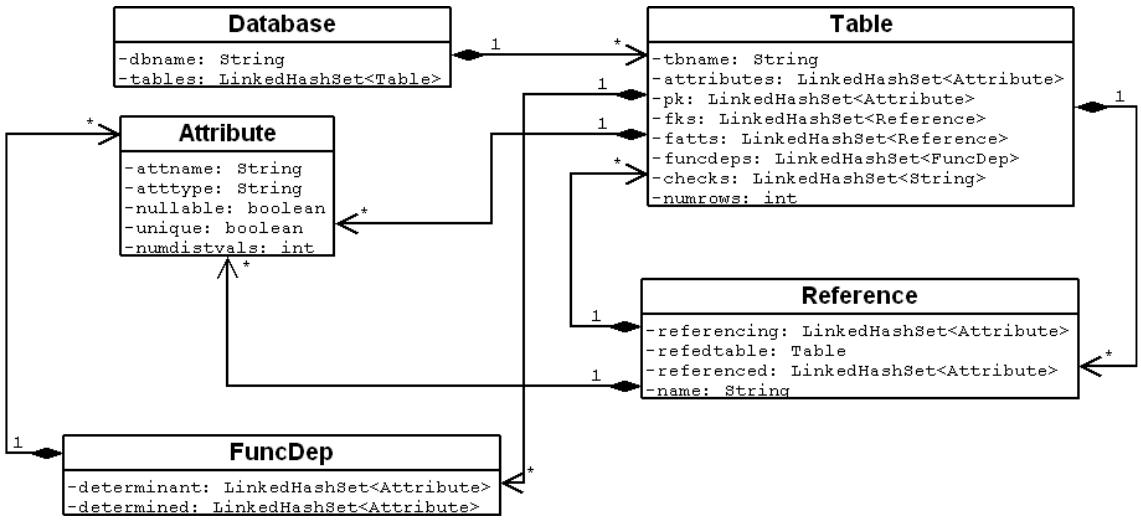

Figura 23. Diagrama UML del paquete Database.

Se ha estudiado ya que cada SMBD tiene características particulares que hacen que el acceso a la información acerca del LDM y las estadísticas en el catálogo no sea siempre igual, es decir, no se pueden utilizar exactamente las mismas consultas en un par cualquiera de SMBDs y, por tanto, el procesamiento de los datos devueltos por las consultas debe ser distinto. Sin embargo, se desea que **LDMExtractor** pueda trabajar de manera transparente con cualquier SMBD.

La interfaz `LDMBuilder` fue creada para forzar a todas las implementaciones de analizador automático del catálogo que pretendan trabajar con la interfaz gráfica de **LDMExtractor** a seguir un mismo esquema en la capacidad y estructura de sus operaciones y en la forma en que estas devuelven sus resultados, así como presentar a los usuarios una forma sencilla de extender la funcionalidad de la aplicación sin necesidad de modificar o conocer el resto del código.

Así pues, `LDMBuilder` presenta el esqueleto del analizador mediante doce operaciones, nueve de las cuales representan las diversas formas de acceder al catálogo directamente y extraer `ResultSet`s de JDBC, una de ellas para construir un modelo completo y representarlo por una instancia de la clase `Database` descrita en la sección anterior, y dos de ellas para forzar una característica de implementación que será discutida más adelante. `LDMBuilder` pertenece al paquete `ldmbuilder.builder`.

La Tabla 15, presente en el Apéndice B, muestra las operaciones que especifica la interfaz `LDMBuilder` así como la descripción de cada una de ellas. Sus valores de regreso serán analizados en las siguientes secciones, conforme sea necesario. Todos los métodos, a excepción de `getConn` que no recibe parámetros, y `setConn` que recibe como parámetro un objeto de tipo `DBConnection`, reciben como parámetro el nombre de la base de datos.

Dadas las características de los SMBD, no siempre es posible implementar alguna o algunas de las operaciones especificadas por `LDMBuilder`. Este es un aspecto que se debe tomar en cuenta al momento de crear una implementación de esta interfaz. Nótese también que las operaciones `getConn` y `setConn` son especificadas para forzar la inclusión de una variable de tipo `DBConnection` que maneje las conexiones a la base de datos y que sea accesible desde el exterior, es decir, una correcta implementación de la interfaz `LDMBuilder` debe contener al menos una variable de estado y esta debe ser de tipo `DBConnection`.

Un aspecto relacionado al anterior es la necesidad de un constructor sin parámetros para cada una de las implementaciones de `LDMBuilder`. La aplicación permite integrar nuevas implementaciones de `LDMBuilder` y utilizarlas sin necesidad de modificar el código fuente de la misma, pero no hay forma de que la aplicación sepa de antemano cómo es que estas implementaciones serán nombradas. Para hacer frente a este problema, se utilizan constructores anónimos y se cargan las clases que implementan a `LDMBuilder` por su nombre, de manera que una simple cadena de caracteres puede especificar el uso de una clase, pero como no es posible saber la estructura de diversos constructores, una correcta implementación de `LDMBuilder` debe contener un constructor sin parámetros.

La estructura de las tablas que son devueltas por las operaciones que crean objetos de tipo `ResultSet` varía de implementación a implementación. Si se tienen dos clases que implementan la operación `getCheckConstraintsMetadata` especificada por `LDMBuilder`, estas dos clases no están obligadas a devolver tablas con estructura idéntica, sino que pueden devolver tablas cuyos resultados sean completamente distintos uno del otro. Por esta razón es importante proporcionar un medio para estandarizar la forma de trabajar con la información que es extraída del catálogo de los SMBD. Este medio es precisamente la operación `buildDatabase`, que tiene como finalidad procesar toda la información obtenida por medio del resto de las operaciones y construir con ella un objeto que, sin importar qué estructura tenga esta en los catálogos o el Information Schema, tenga siempre la misma estructura, permitiendo entonces su análisis.

A continuación se describen tres implementaciones de `LDMBuilder` que corresponden a los tres SMBD que el presente trabajo contempla. Más adelante se describirán algunos otros elementos de **LDMExtractor** que fueron mencionados en esta sección pero que no fueron explicados.

VI. 3. 1. MySQLLDMBuilder

Se ha estudiado ya que la implementación del Information Schema de MySQL es, de las tres que contempla este trabajo, la más simple y escasa de todas. La implementación de la gran mayoría de las operaciones de la interfaz `LDMBuilder` no está presente, y la razón de esto no es que no sea posible llevarla a acabo sino que, al menos para el presente trabajo, no se consideró útil implementarlas.

La Tabla VI-1 muestra qué operaciones están implementadas y cuáles no en `MySQLLDMBuilder`.

Como se estudió en la sección 4.1, existe una consulta capaz de extraer todos los metadatos acerca de una base de datos almacenada en MySQL en una sola consulta. Esta consulta es utilizada por `getMetadata` para extraer la información requerida y convertirla en un `ResultSet` de JDBC.

Así mismo, los métodos `getTablesMetadata` y `getConstraintsMetadata` utilizan las consultas mostradas en la Figura 2 y en la Figura 3 para realizar sus tareas, devolviendo, al igual que `getMetadata`, objetos de tipo `ResultSet` de JDBC.

Nombre	Implementado
buildDatabase	Si
getMetadata	Si
getTablesMetadata	Si
getConstraintsMetadata	Si
getPrimaryKeyConstraintsMetadata	No
getUniqueKeyConstraintsMetadata	No
getCheckConstraintsMetadata	No
getForeignKeyConstraintsMetadata	No
getStatistics	No
getTableStatistics	Si
getColumnStatistics	No
getConn	Si
setConn	Si

Tabla VI-1. Métodos de la interfaz LDMBuilder implementados y no implementados por MySQLLDMBuilder.

La implementación de `buildDatabase` utiliza diversas operaciones internas que hacen uso de las operaciones de extracción de metadatos para poder construir el objeto `Database`. Lo que internamente sucede es que el método ejecuta

consecutivamente las operaciones para construcción de los diversos elementos del LDM de una base de datos en el siguiente orden:

1. construcción del objeto `Database` que será el resultado
2. construcción e inserción de las tablas de la base de datos
3. construcción e inserción de las restricciones
4. extracción y actualización de datos estadísticos en los objetos que representan las tablas

Obsérvese que estos pasos coinciden con los métodos que se encuentran implementados en `MySQLLDMBuilder`. La razón para no usar el método `getMetadata` es que resulta más simple realizar el análisis del LDM por partes, ya que se tendría que almacenarse en alguna variable el `ResultSet` y este tendría que ser restaurado y recorrido varias veces para poder reconstruir los elementos en el orden adecuado, siendo requeridas, además, más comparaciones y otras operaciones sobre los datos extraídos.

Es cierto que puede resultar ligeramente más eficiente para el SMBD atender una sola consulta en lugar de tres, pero se ha estimado que la ganancia es mínima y, por tanto, despreciable en un marco general. El orden del tamaño de los resultados extraídos crece en proporción del número de tablas en la base de datos y el número de columnas en dichas tablas. Para obtener una idea de estas proporciones, considérese una base de datos con 1000 tablas[1], cada una de las cuales con 10 columnas cada una, considérese también una consulta Q que extrae la información de las tablas y sus columnas. El resultado de dicha consulta contendría 10000 tuplas, número que es relativamente pequeño al tamaño que pueden alcanzar las bases de datos en la actualidad, con tablas con millones o aún miles de millones de registros en los casos más extremos.

Como se mencionó anteriormente, si así se requiere, pueden utilizarse versiones optimizadas de las consultas aquí descritas, así como una implementación de `LDMBuilder` que utilice dichas versiones o que implemente otras mejoras.

VI. 3. 2. PSQLLDMBuilder

Mucho más completa que la implementación de MySQL, la implementación de PostgreSQL del Information Schema provee de más información, aunque también aumenta la complejidad en los procesos que se requieren para extraer los metadatos de una base de datos almacenada en este SMBD.

La Tabla VI-2 muestra qué operaciones están implementadas y cuáles no en `PSQLLDMBuilder`.

[1] Aunque no hay una norma que especifique el número de tablas que debe tener una base de datos relacional, se estima que algunas aplicaciones de Planificación de Recursos Empresariales (ERP por sus siglas en inglés) alcanzan un pico de mil tablas aproximadamente.

Nombre	Implementado
buildDatabase	Si
getMetadata	No
getTablesMetadata	Si
getConstraintsMetadata	No
getPrimaryKeyConstraintsMetadata	Si
getUniqueKeyConstraintsMetadata	Si
getCheckConstraintsMetadata	Si
getForeignKeyConstraintsMetadata	Si
getStatistics	Si
getTableStatistics	No
getColumnStatistics	No
getConn	Si
setConn	Si

Tabla VI-2. Métodos de la interfaz LDMBuilder implementados y no implementados por PSQLLDMBuilder.

Como se estudió en la sección 4.2, no existe una consulta capaz de extraer todos los metadatos acerca de una base de datos almacenada en PostgreSQL en una sola consulta, por lo que la implementación de `getMetadata` no está presente. Lo mismo sucede en el caso de `getConstraintsMetadata`.

Sin embargo, el resto de las operaciones definidas por `LDMBuilder` pueden ser implementadas por `PSQLLDMBuilder`, aunque dos de ellas no son implementadas pues existe una operación que resume la información devuelta por las dos no implementadas, y no todas son utilizadas por `buildDatabase`. Las consultas mostradas en la sección 4.2 son utilizadas en los métodos que corresponden para crear objetos de tipo `ResultSet` conteniendo los resultados de cada una de ellas. Así mismo, las mostradas en la sección 5.2 son utilizadas para extraer los datos estadísticos presentes en la base de datos.

De manera similar a lo que sucede en `MySQLLDMBuilder`, la implementación de `buildDatabase` presente en `PSQLLDMBuilder` utiliza diversas operaciones internas que hacen uso de las operaciones de extracción de metadatos para poder construir el objeto `Database`. Internamente, el método ejecuta consecutivamente las operaciones para construcción de los diversos elementos del LDM de una base de datos en el siguiente orden:

1. construcción del objeto `Database` que será el resultado
2. construcción e inserción de las tablas de la base de datos
3. construcción e inserción de las llaves primarias
4. construcción e inserción de las llaves únicas
5. construcción e inserción de los *check constraints*
6. construcción e inserción de las llaves foráneas
7. extracción y actualización de datos estadísticos en los objetos que representan las tablas y a las columnas

Una vez más, estos pasos coinciden con los métodos que se encuentran implementados en `PSQLLDMBuilder`. Obsérvese que el último paso difiere del último paso estudiado en `MySQLLDMBuilder`, haciendo evidente una vez más la ausencia de datos estadísticos en MySQL.

VI. 3. 3. MSSQLLDMBuilder

La implementación del Information Schema de SQL Server, aunque no tan completa como la de PostgreSQL, presenta un escenario similar al estudiado en este último.

La Tabla VI-3 muestra qué operaciones están implementadas y cuáles no en `MSSQLLDMBuilder`.

Nombre	Implementado
buildDatabase	Si
getMetadata	No
getTablesMetadata	Si
getConstraintsMetadata	No
getPrimaryKeyConstraintsMetadata	Si
getUniqueKeyConstraintsMetadata	Si
getCheckConstraintsMetadata	Si
getForeignKeyConstraintsMetadata	Si
getStatistics	No
getTableStatistics	No
getColumnStatistics	No
getConn	Si
setConn	Si

Tabla VI-3. Métodos de la interfaz LDMBuilder implementados y no implementados por MSSQLLDMBuilder.

Prácticamente las mismas operaciones implementadas por `PSQLLDMBuilder` son implementadas por `MSSQLLDMBuilder`, excepto aquellas que extraen la información estadística de tablas y columnas. Las consultas mostradas en la sección 4.3 son utilizadas en los métodos que corresponden para crear objetos de tipo `ResultSet` conteniendo los resultados de cada una de ellas.

Ya en el capítulo 5, en la sección referente a SQL Server, se han estudiado las razones por las cuales no es posible extraer con un cliente distinto a los proporcionados por Microsoft los datos estadísticos de la base de datos.

La implementación de `buildDatabase` presente en `MSSQLLDMBuilder` utiliza operaciones internas que utilizan las operaciones de extracción de metadatos para

poder construir el objeto `Database`. Internamente, el método ejecuta una a una a las operaciones para construcción de los elementos del LDM de una base de datos en el siguiente orden:

1. construcción del objeto `Database` que será el resultado
2. construcción e inserción de las tablas de la base de datos
3. construcción e inserción de las llaves primarias
4. construcción e inserción de las llaves únicas
5. construcción e inserción de los *check constraints*
6. construcción e inserción de las llaves foráneas

Como es de esperarse, estos pasos coinciden con los métodos que se encuentran implementados en `MSSQLLDMBuilder`.

VI. 4. La GUI de LDMExtractor

Una GUI simple y completa es siempre lo que un usuario busca tener entre sus manos. Son precisamente esas dos características las que persigue el diseño de la GUI de **LDMExtractor**.

Al ejecutar el programa, la pantalla que el usuario puede apreciar es la mostrada en la Figura 24[1], en ella se pueden observar tres elementos importantes.

El primero de ellos es la barra de menús, conteniendo tres de ellos. El menú *File* contiene las opciones referentes al manejo de archivos XML dentro de **LDMExtractor**, que son abrir (Open), cerrar (Close), y guardar (Save). El menú *Connection* contiene solamente una opción, que es conectarse (Connect). Finalmente, el menú *Help* contiene solamente la opción acerca de (About) que muestra la información referente a la aplicación.

El segundo de ellos es un cuadro que muestra una estructura de tipo árbol que, al momento de iniciar la aplicación, solamente muestra la leyenda "[No database loaded]" indicando que ningún LDM ha sido cargado por ningún medio.

El tercero de ellos es un cuadro en el que se muestran las propiedades del elemento seleccionado. Al momento de iniciar la aplicación, este cuadro no muestra ninguna información.

[1] Algunos elementos visuales en la aplicación pueden diferir ligeramente de los mostrados en las imágenes de este trabajo. Todas las imágenes se muestran en escala de grises.

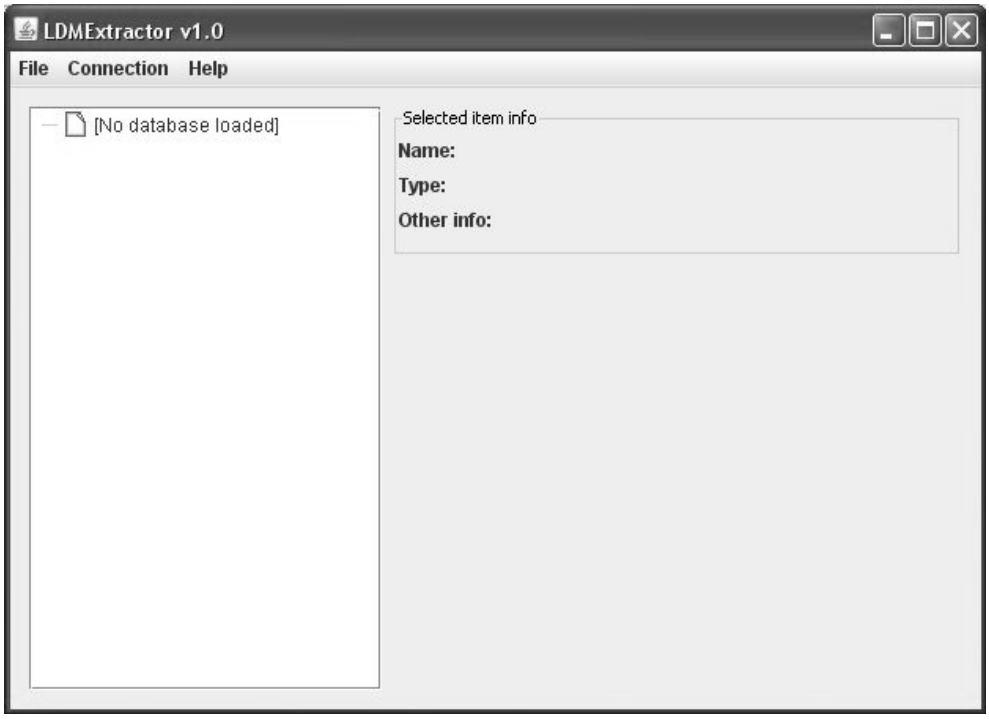

Figura 24. La pantalla principal de LDMExtractor.

Una de las principales características de **LDMExtractor** es su capacidad para conectarse a una base de datos y extraer el LDM proporcionando únicamente el nombre de la misma. Al seleccionar la opción Connect del menú Connection, aparece una ventana como la mostrada en la Figura 25. En esta ventana se debe indicar el nombre de la base de datos, la dirección del servidor donde se encuentra dicha base, el nombre del *driver* compatible con JDBC a utilizar, el nombre de la clase que implementa a `LDMBuilder` que debe ser utilizada para el tipo de SMBD con el que se está trabajando y los datos de conexión. Es precisamente esta ventana la que permite integrar nuevos SMBD al repertorio de **LDMExtractor**. Al presionar el botón *Connect*, el LDM de la base de datos indicada es extraído del catálogo y mostrado en pantalla.

Ya sea que se haya seleccionado la opción Open del menú File o que una base de datos haya sido cargada mediante la opción Connect del menú Connection, el LDM de la base de datos es cargado y mostrado en forma de árbol como se muestra en la Figura 26. En esta misma figura podemos observar como, al seleccionar un elemento, en el panel de la derecha aparecen datos descriptivos del mismo. Los datos que aparecen en el cuadro de información varían según el elemento seleccionado. Siempre que se seleccione una tabla o columna, entre estos datos aparecen las estadísticas almacenadas por los SMBD. Obsérvese que todas las tablas y columnas tienen esta característica, aún cuando algunas veces no haya estadísticas definidas para un elemento específico.

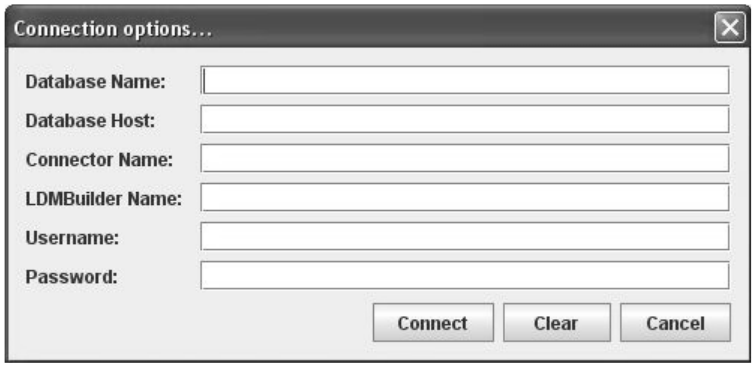

Figura 25. Opciones de conexión.

Figura 26. El LDM de una base de datos es mostrado en pantalla.

Una vez que un LDM se encuentra cargado en el sistema, este puede ser almacenado en formato XML para su posterior uso. Esto se logra utilizando la opción Save del menú File. Es importante hacer notar que, dado que las estadísticas no forman parte del LDM de una base de datos, estas no son almacenadas en el archivo XML una vez que este es guardado. La razón es que si se tienen varias instancias de una misma base de datos, es posible que ambas posean información estadística distinta, pero si son instancias del mismo LDM, sus tablas, columnas, y demás características serán exactamente las mismas.

No es necesario cerrar la conexión, pues esta no permanece abierta. Una vez que las tareas de **LDMExtractor** son realizadas, la conexión es cerrada para no retener recursos que ya no serán utilizados.

VI. 5. Otros elementos de LDMExtractor

LDMExtractor requiere de algunas otras clases, además de las ya mencionadas, para poder realizar sus funciones.

Se pueden destacar las clases contenidas en el paquete `DatabaseConnection`. Este paquete fue desarrollado por Eduardo Yoztaltépetl Ángel Hernández y René Alejandro Villeda Ruz, y forma parte del proyecto Refined [25], del cual también forma parte (al menos parcialmente) el presente trabajo. El paquete `DatabaseConnection` presenta una interfaz simple para encapsular las tareas comunes de conexión a una base de datos, así como presentar un manejo limpio de los datos de conexión. La clase `DBConnection` hace posible utilizar cualquier *driver* compatible con JDBC de manera similar a como se utilizan las implementaciones de `LDMBuilder`.

No menos importante es la clase `XMLDoc`, que le otorga a **LDMExtractor** la capacidad de leer archivos XML y transformarlos en objetos de tipo `Database`, realizando las validaciones necesarias en este proceso. La clase `XMLDoc` es un *parser* de documentos de tipo `Database` que cumplen con la DTD especificada para este tipo de documentos y que se expuso en la sección 6.2. Como todo *parser*, primero realiza una verificación sintáctica del documento, es decir, se verifica que la estructura gramatical del documento sea la correspondiente a la especificada por la DTD. Esta tarea se realiza automáticamente gracias a las propiedades de la clase `DocumentBuilder` que, al construir objetos de tipo `Document`, realiza la validación si el documento a procesar hace referencia a una DTD específica. Pero el trabajo de `XMLDoc` se extiende a la verificación semántica, que no es posible solamente analizando la estructura del documento. Esta clase se encarga de verificar que las referencias definidas por los elementos del documento XML sean semánticamente correctas, es decir, que no existan elementos que violen las reglas de integridad, como referencias a elementos inexistentes.

El paquete `ldmbuilder.visual` contiene todos los elementos que hacen posible la visualización de la GUI de la aplicación. No se entrará en detalles en este punto, puesto que no es relevante en el sentido del desarrollo teórico del presente trabajo.

Adicionalmente, el diagrama de clases completo de **LDMExtractor** puede verse en el Apéndice C.

VII. Experimentación

En este capítulo se presentan los resultados obtenidos en las pruebas de la aplicación al conectarse con una base de datos en cada uno de los SMBD estudiados por el presente trabajo. No se contempla el caso en el que se tiene un archivo en formato XML que contiene el LDM de una base de datos, pues dicho caso es irrelevante en el sentido teórico del presente trabajo. Todas las pruebas serán realizadas utilizando la base de datos sintética definida en TPC-H [27].

TPC-H es un *benchmark* de soporte para toma de decisiones. Está formado por un conjunto de consultas orientadas a negocios y modificaciones de datos concurrentes. Las consultas y los datos poblando la base de datos han sido elegidos para tener relevancia en un amplio sector de la industria manteniendo un grado adecuado de facilidad de implementación. Este *benchmark* ilustra sistemas de soporte de decisión que examinan grandes volúmenes de datos, ejecutan consultas con un alto grado de complejidad y dan respuesta a preguntas de negocios críticas [27].

El diagrama relacional de la base de datos especificada por TPC-H, así como todas las especificaciones pertinentes, pueden ser encontrados en [27].

En adelante, se dará por hecho que en cada SMBD existe una base de datos nombrada *tpch* que respeta la estructura definida en las especificaciones de TPC-H. En el caso de PostgreSQL y SQL Server, se ha elegido un esquema nombrado `tables` para almacenar todas las tablas de la base de datos. Todos los SMBD se encuentran instalados en el mismo equipo en que se ejecuta la aplicación, de manera que la dirección IP hacia los servidores es siempre `localhost` (127.0.0.1). La base de datos será poblada en el factor de escalamiento 1, según la notación de TPC-H. No se ha definido índice alguno sobre las columnas de las tablas, más que aquellos implícitamente creados al especificar las llaves primarias de las mismas. Siempre que se ha podido, se han homologado los nombres de aquellos elementos que no tienen asociado un nombre estándar en la definición de TPC-H, como es el caso de las llaves foráneas. No se han definido más restricciones que las llaves primarias y foráneas especificadas por TPC-H.

El primer paso es conectarse a la base de datos. La Figura 27 muestra la configuración para el caso de MySQL, la Figura 28 muestra la configuración para el caso de PostgreSQL y la Figura 29 muestra la configuración para el caso de SQL Server.

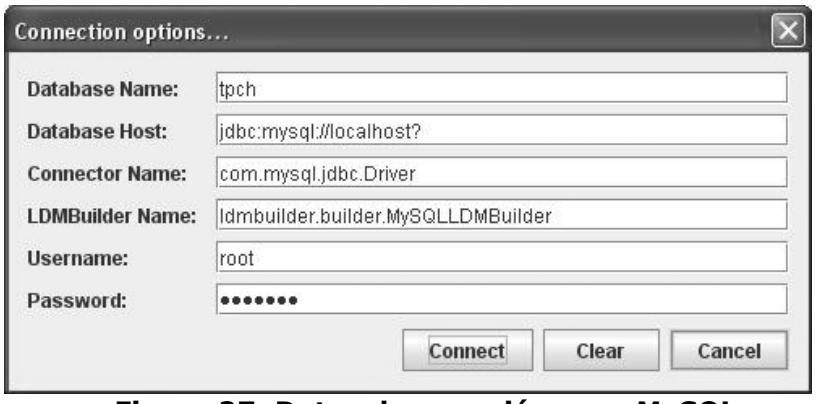

Figura 27. Datos de conexión para MySQL.

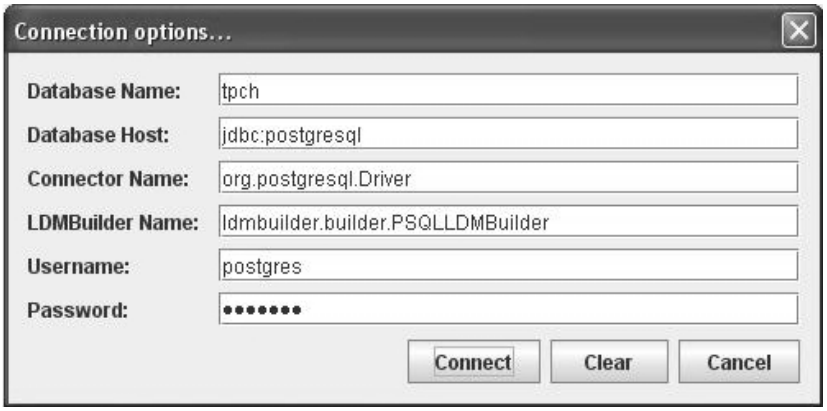

Figura 28. Datos de conexión para PostgreSQL.

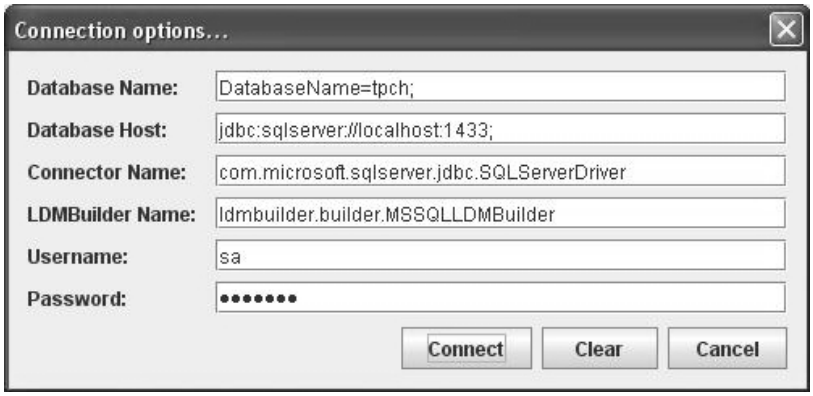

Figura 29. Datos de conexión para SQL Sever. Nótese que es necesario agregar `DatabaseName=` como prefijo del nombre de la base de datos y punto y coma al final de este.

Al oprimir el botón *Connect*, en cualquiera de los tres casos la pantalla que se obtiene es la mostrada en la Figura 30, con ligeras variaciones en los nombres de los elementos, producto de sus distintas representaciones internas.

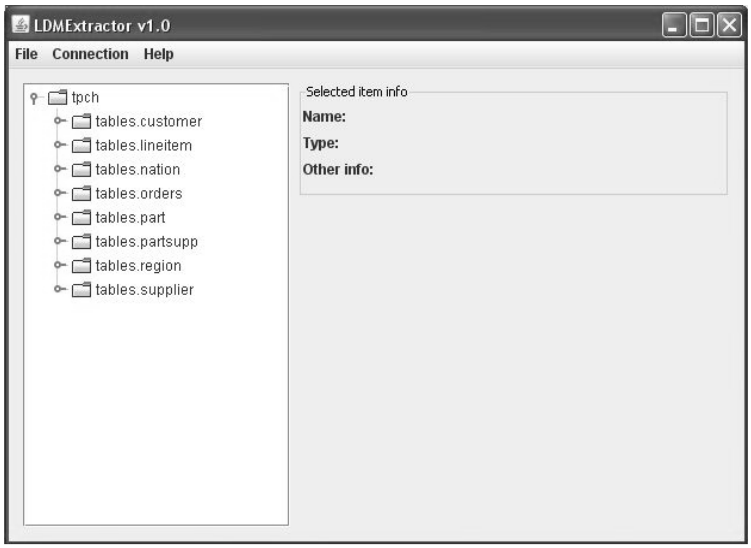

Figura 30. El LDM de TPC-H dispuesto en forma de árbol en pantalla.

Una vez que se tiene cargada la base de datos, se procede a realizar el análisis de los elementos de la misma. Al dar clic sobre la tabla `customer`, en el lado derecho de la ventana se muestran los datos respectivos al elemento seleccionado, como se muestra en la Figura 31, en la Figura 32 y en la Figura 33. En el caso de la base de datos almacenada en MySQL o PostgreSQL se puede apreciar la presencia de información acerca de las estadísticas, mientras que en el caso de SQL Server no se muestra correctamente el número de tuplas de la tabla, dadas las limitantes en el SMBD expuestas en la sección 5.3.

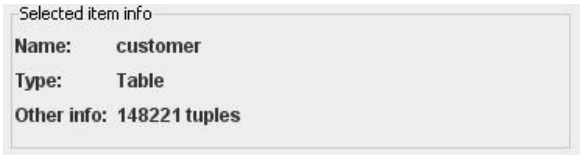

Figura 31. La tabla `customer` en MySQL

Figura 32. La tabla `customer` en PostgreSQL

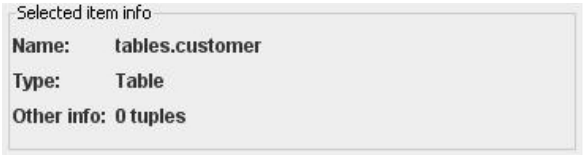

Figura 33. La tabla `customer` en SQL Server

Al ampliar la vista de la tabla y seleccionar el atributo `c_custkey` que representa la llave primaria de `customer`, SQL Server y MySQL presentan los mismos elementos en el lado derecho, no así PostgreSQL, que por permitir el acceso a estadísticas más completas, muestra un dato correcto en el número de valores distintos en la columna, que coincide con el número de tuplas de la tabla por tratarse de la columna que representa la llave primaria.

Figura 34. A la izquierda, lo mostrado por la aplicación relativo a la columna `custkey` de `customer` almacenada en MySQL y SQL Server, a la derecha lo mostrado en el caso de PostgreSQL.

En la Figura 35 se muestra el contenido de la tabla `customer` una vez que esta es expandida en el caso de MySQL, así como en la Figura 36 se muestra el caso de PostgreSQL, y en la Figura 37 se muestra el caso de SQL Server. Obsérvese las diferencias entre los nombres de los tipos de datos de PostgreSQL y los otros dos SMBD, así como otras pequeñas diferencias en el nombrado de los elementos.

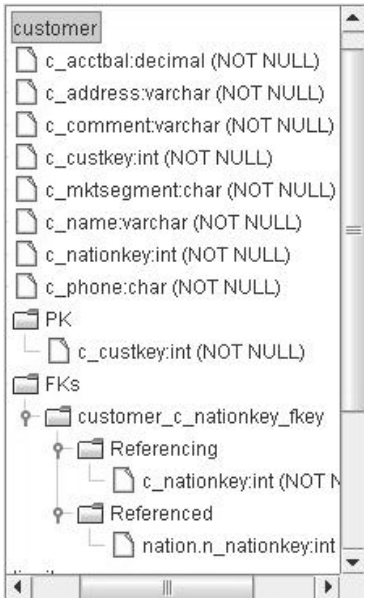

Figura 35. Vista expandida de la tabla `customer` en MySQL

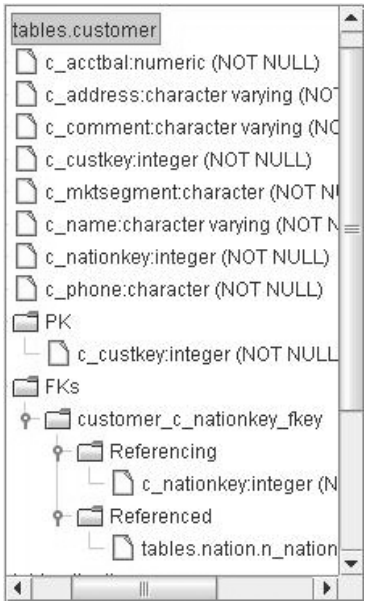

Figura 36. Vista expandida de la tabla `customer` en PostgreSQL

Para poder ejemplificar el resto de los elementos que **LDMExtractor** es capaz de extraer del catálogo y mostrar en su interfaz, se agregarán dos restricciones de integridad a la tabla `customer`, una de tipo *check* y una restricción de tipo llave única.

La Figura 38 muestra la vista ampliada de la tabla `customer` después de ser modificada. Recuérdese que MySQL no soporta las restricciones de tipo *check*, por lo que este elemento no aparece en la vista ampliada correspondiente a este SMBD.

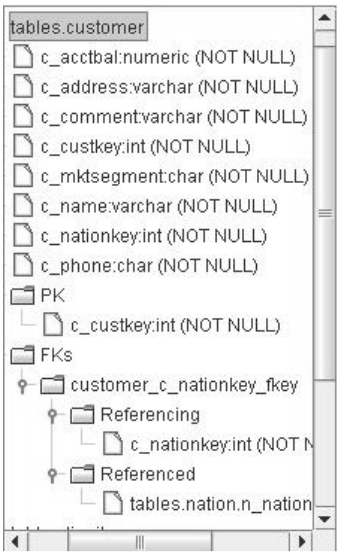

Figura 37. Vista expandida de la tabla `customer` en SQL Server

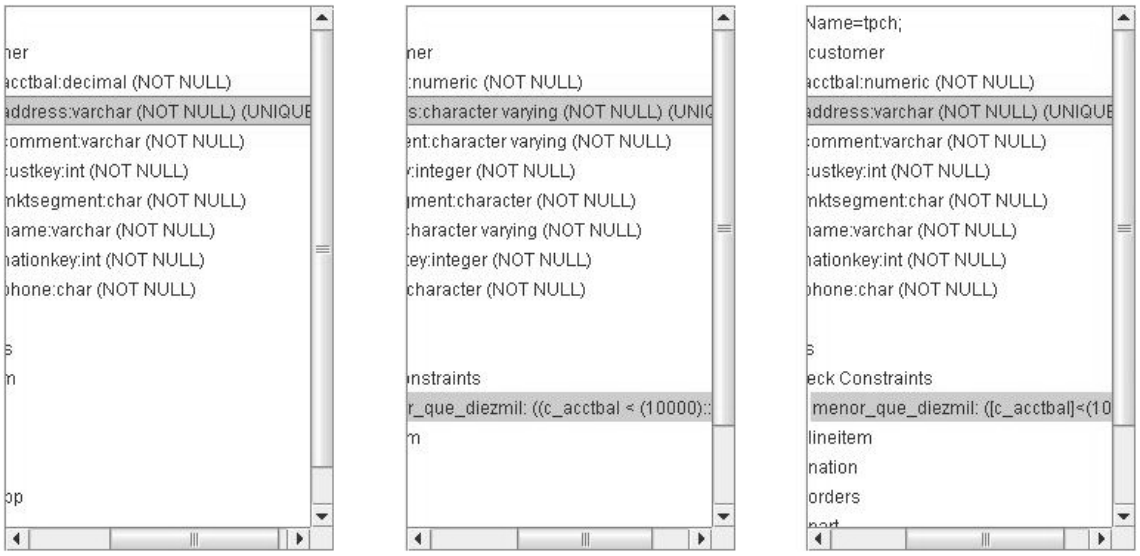

Figura 38. Vista expandida de la tabla `customer` tras ser modificada. Izquierda: MySQL, centro: PostgreSQL, derecha: SQL Server.

Pueden verse las restricciones agregadas seleccionadas en los respectivos árboles. La restricción de tipo *check* que ha sido definida especifica únicamente que el valor del atributo `customer.c_acctbal` debe ser menor a 10000, como puede verse en la Figura 38 la representación de dicho elemento en el árbol. La llave única ha sido definida sobre el campo `customer.c_address` y es mostrada como una característica de dicho atributo como parte de su representación en pantalla. Nótese que, si fuera seleccionado el atributo `customer.c_address`, solamente en el caso de estar analizando la base de datos almacenada en PostgreSQL podría apreciarse la estadística de columna implícitamente generada al crear una llave única sobre dicho atributo.

VIII. Trabajo relacionado

Además de lo establecido por Codd y sus principios a cumplir por los SMBDR, y la descripción del modelo relacional de Codd mismo, no muchos estudios han sido realizados con respecto al catálogo de los SMBD específicamente.

Existen algunas herramientas que son capaces de realizar ingeniería inversa sobre LDM de bases de datos almacenadas en múltiples SMBD aunque estas, en su mayoría, son software propietario.

Del software propietario, destaca *Azzurri Clay* de Azzurri, Ltd. [22], que presenta una herramienta integral de modelado de bases de datos para múltiples SMBD tanto comerciales como Open Source. Azzurri Clay es una herramienta de diseño de bases de datos que ha sido implementada como un *plug-in* para el ambiente de desarrollo de Eclipse (http://www.eclipse.org). Clay tiene una interfaz de usuario intuitiva para diseñar gráficamente una base de datos. Clay también puede crear el modelo de datos realizando ingeniería inversa sobre una base de datos existente. Además, Clay genera el código SQL (DDL) apropiado para cada base de datos [22]. El problema con Clay es que, a pesar de ser gratis, no es Open Source, por lo que no puede extenderse ni integrarse a otros desarrollos.

Entre las herramientas Open Source podemos encontrar a *PGDesigner* creado por Josh Bosh. En su sitio web [7], Bosh explica que PGDesigner es su intento de implementar un diseñador de modelos de datos para PostgreSQL y que, si bien es cierto que ya existen buenas herramientas de modelado de datos, ninguna de estas es Open Source. La documentación es prácticamente nula, pero se menciona que PGDesigner es capaz de importar tablas con llaves foráneas de bases de datos existentes a un diagrama. Sin embargo, no se menciona nada acerca de otros tipos de restricciones, y mucho menos se habla acerca de estadísticas, elementos que **LDMExtractor** es capaz de extraer del catálogo del sistema, no solamente de PostgreSQL, sino también de MySQL y de SQL Server.

En el área de bases de datos distribuidas, los autores describen en \cite{1017614} la creación de un modelo conceptual para el diccionario de datos de sistemas manejadores de bases de datos distribuidas. Plantean la problemática de los SMBDD con respecto a la existente en los SMBD convencionales. En este marco, una aplicación que analiza el catálogo del sistema debería también poder cubrir detalles propios de la distribución, quizá en el ámbito de las estadísticas, o en entidades separadas que

representen información como el servidor en que se encuentra un elemento en específico, o aún en tablas (vistas) con información específica para este problema.

Posteriormente, en [10] se propone un esquema de distribución del catálogo al que nombra *GOCAS (group-oriented catalog allocation scheme)*, que pretende reducir el número de mensajes entre cada uno de los sitios del sistema distribuido que tengan que ver con accesos al catálogo del sistema. Si bien es cierto que este trabajo no contempla ningún SMBD distribuido, es importante notar que, al menos en principio, los catálogos deberían tener las mismas características para poder ser relacionales y, por tanto, el presente trabajo debería poder ser extendido sin mayores problemas a esquemas de este tipo.

La herramienta descrita en el presente trabajo es capaz de generar archivos XML que representan al LDM de la base de datos que se está analizando. Los autores describen en [29] una herramienta que es capaz de integrar los datos de distintas fuentes mediante el uso de XML para estandarizar la información contenida en cada uno de ellos, de manera que, también mediante XML, se pueda intercambiar información entre las distintas fuentes de datos de manera transparente, haciendo las adaptaciones pertinentes no sobre los datos en sí, sino sobre su representación. Este enfoque nos da la idea de que **LDMExtractor** puede fácilmente ser ampliado para apoyar la migración de esquemas lógicos entre distintos SMBD o incluso para hacer las veces de puente entre la estructura de un LDM en un SMBD y la estructura del mismo LDM en otro, pues es bien sabido que, aunque el lenguaje de definición de datos es SQL en todos los SMBDR, en la gran mayoría de los casos es necesario realizar adaptaciones en las sentencias SQL que las herramientas de respaldo de cada SMBD genera antes de poder crear la misma base de datos en otro SMBD.

LDMExtractor también es capaz de extraer información acerca de las estadísticas mantenidas por el catálogo de los distintos SMBD. Dichas estadísticas son, generalmente, utilizadas en la optimización de consultas automática, aunque bien pueden ser utilizadas manualmente para realizar optimización analítica de las consultas que se realizan. Así mismo, en algunas ocasiones, un conocedor de los datos existentes en una base de datos en particular puede detectar si las estadísticas mantenidas por el SMBD no son realistas, y así realizar tareas de mantenimiento.

Con respecto a dichas estadísticas, algunos autores realizan trabajo que intenta mejorar la calidad de las mismas. Por ejemplo, en [31] se propone un método para mejorar la calidad de los datos estadísticos mantenidos por los SMBD en sus catálogos manteniéndolos al día. Este método fue denominado *piggyback method*[1] y su idea principal consiste en "sobrecargar" el procesamiento de las consultas de los usuarios con recolección automática implícita de información estadística acerca de los datos sobre los que estas operan. Mencionan que, aunque esta información puede no tener ninguna utilidad para el usuario que realiza la consulta, y que incluso la obtención de la misma puede hacer ligeramente más largo el tiempo de respuesta por parte del SMBD, las estadísticas recolectadas pueden ser utilizadas para, de manera efectiva y a bajo costo, mejorar el procesamiento de consultas subsecuentes en las que dichos datos estén involucrados. Claramente, este enfoque apoya tanto a los optimizadores de consultas clásicos, que involucran solamente estimaciones estáticas (realizadas antes

[1] El término *piggyback* se refiere a la idea de que las operaciones están "montadas" sobre la consulta mientras ésta es ejecutada

de comenzar el procesamiento de la consulta), como a aquellos optimizadores que lleven a cabo optimizaciones dinámicas (adaptativas). Versiones modificadas de las consultas para extraer estadísticas expuestas en este trabajo podrían fácilmente dotar a **LDMExtractor** con la capacidad de analizar estas y otras estadísticas que fueran creadas para los diversos elementos almacenados por los SMBD.

Esta misma idea fue retomada en [15], extendiendo el método propuesto para tomar en cuenta y calcular no solamente estadísticas a un solo nivel de granularidad, que es el de tabla, sino que un SMBD que implemente el citado método pueda determinar por sí mismo el grado de *interleaving*[1] o incluso, si el SMBD lo permite, el grado de paralelismo que es posible aplicar para una consulta en específico que tiene adheridas a ella una serie de operaciones *piggyback* dadas. Los autores exponen técnicas que permiten la integración automática de los procesos concernientes a la consulta y a las operaciones *piggyback* a los niveles apropiados de granularidad.

Finalmente, uno de los elementos más importantes del LDM de una base de datos son sus restricciones o *constraints*. Aquellos que son contemplados por **LDMExtractor** son las llaves primarias, únicas y foráneas, así como los *CHECK* y *NOT NULL constraints*. Los autores proponen en [20] la creación de una restricción más, una basada en las estadísticas recabadas acerca de los datos. Los autores denominan a este tipo de *constraint* como *restricción estadística*. Las restricciones estadísticas manifiestan relaciones implícitas entre los distintos valores de un atributo y están caracterizadas por su naturaleza probabilística, permitiendo la detección de errores que, con las restricciones comunes en los SMBD, no sería fácil detectar.

[1] Método para incrementar el rendimiento de la memoria permitiendo al procesador acceder a un sector de memoria mientras otro es recargado, de manera que el procesador nunca tiene que esperar a que la memoria se refresque, pues siempre tiene datos disponibles

IX. Resumen de hallazgos

El presente trabajo comenzó con la idea de crear una aplicación que fuera capaz de realizar un proceso de ingeniería inversa sobre las bases de datos para extraer de sus catálogos el LDM con el que fueron construidas, así como extraer las estadísticas básicas de las mismas. Para alcanzar esa meta, mucha información requirió ser analizada, por lo que los alcances de esta investigación se expandieron. Este capítulo pretende recopilar toda la información generada y a partir de ella encontrar datos interesantes sobre los tres SMBD que fueron estudiados en este trabajo.

La Tabla III-1 analiza el apego a las reglas de Codd referentes al catálogo de los SMBD, en ella se indica el nombre de una característica, se dice si el SMBD la implementa, y en caso de no ser así, se especifica qué es lo que hace falta para que dicha característica se encuentre correctamente implementada. Las características mencionadas en la tabla se encuentran ampliamente descritas en el capítulo 1.

	MySQL	PSQL	MSSQL
Acceso al catálogo	Implementada	Implementada	Implementada
Descripción de dominios, relaciones y vistas	Implementada	Implementada	Implementada
Restricciones de integridad	No da soporte a check constraints	Implementada	Implementada
UDFs	Implementada	Implementada	Implementada
Seguridad y rendimiento	Implementada	Implementada	Implementada

Tabla IX-1. Reglas de Codd referentes al catálogo y la forma en que son implementadas por los SMBD estudiados

Mucha documentación acerca de los SMBD tuvo que ser analizada para poder obtener la información requerida para el presente trabajo. Muchas veces una alta calidad de la documentación puede facilitar enormemente un trabajo de investigación, mientras que una documentación pobre o confusa puede dificultarla mucho. La Tabla IX-2 muestra información acerca de la calidad de la documentación estudiada[1]. Aunque

[1] El presente trabajo no pretende establecer un criterio formal para evaluar la calidad de la documentación de las herramientas estudiadas, simplemente pretende presentar una idea intuitiva de las cualidades positivas y las carencias detectadas en dicha documentación al realizar esta investigación.

los tres SMBD son populares, el soporte de la comunidad (foros de discusión, FAQs, etc.) es mejor en el caso de los Open Source que en el de SQL Server, obteniéndose respuestas más completas y mejor explicadas, generalmente más técnicas.

	MySQL	PSQL	MSSQL
Facilidad de uso	Buena	Buena	Regular
Completez	Regular	Buena	Buena
Actualizaciones	No frecuentemente	No frecuentemente	Frecuentemente
Soporte de la comunidad	Amplio	Amplio	Amplio, pero no siempre completo

Tabla IX-2. Calidad de la documentación analizada.

Con respecto a las estadísticas, las reglas de Codd especifican que se requiere el número de tuplas por tabla y el número de valores distintos por columna. En el capítulo 5 se especificó que, por cuestiones de implementación, los SMBD no implementan completamente la segunda de las estadísticas, pero que el no implementarla no tiene repercusiones en el rendimiento de los planificadores de consultas si se realiza un buen diseño y se agregan índices en las columnas adecuadas. La Tabla IX-3 muestra las estadísticas que se encuentran implementadas en cada SMBD, si el acceso a estas es el adecuado, y otras características relevantes.

	MySQL	PSQL	MSSQL
Tuplas por tabla	Implementada	Implementada	Implementada
Valores distintos por columna	No implementada	Implementada	Implementada
Actualización de estadísticas	Automática y manual estimativas	Automática estimativa y manual precisa	Automática y manual estimativas
Acceso a estadísticas	SQL	SQL	DBCC no compatible con SELECT

Tabla IX-3. Características referentes a las estadísticas implementadas por los distintos SMBD estudiados. Los valores distintos por columna sólo toman en cuenta columnas con índice por lo expuesto en el capítulo 5.

Con respecto al Information Schema, los tres SMBD estudiados cumplen con un rango distinto de elementos. La Tabla IX-4 muestra algunos datos estadísticos de las implementaciones de los tres SMBD.

	MySQL	PSQL	MSSQL
Implementados	9	38	16
No implementados	48	16	42
Parcialmente implementados	3	7	2
Incorrectamente implementados	1	0	1
No estándar	3	0	1

Tabla IX-4. Elementos del Information Schema implementados por los distintos SMBD estudiados. Sólo se toma en cuenta a las vistas.

Con respecto a las operaciones de `ldmbuilder.LDMBuilder` que es posible y útil implementar dadas las características de los SMBD, PostgreSQL es una vez más el que presenta las características más completas y compatibles con dichas operaciones. Un resumen de esto puede apreciarse en la Tabla IX-5.

	MySQL	PSQL	MSSQL
Implementadas	7	9	8
No implementadas	6	4	5

Tabla IX-5. Número de operaciones de `ldmbuilder.LDMBuilder` que es posible y útil implementar en cada SMBD estudiado.

X. Conclusiones

El presente trabajo estudió la motivación del uso del Modelo Relacional como base para crear un SMBD. Se analizó uno de los primeros SMBD que se hizo llamar relacional y se observó que, aunque el número de reglas y especificaciones de Codd acerca de lo que un SMBD debe tener para ser relacional que cumplía System R, al menos en lo referente a su catálogo, este no podía llamarse así, ya que dicho catálogo no daba soporte a las restricciones de integridad referencial.

Con respecto a los objetivos planteados para la aplicación, se puede decir que esta los cumple adecuadamente. Recuérdese que se esperaba que:

- fuera capaz de conectarse a diversos SMBD mediante una interfaz común para todos ellos,
- fuera modular para aislar las diferencias entre los distintos SMBD y poder así dar mantenimiento a módulos específicos para dar soporte a nuevas características en los SMBD sin tener que conocer ni modificar al resto de los módulos,
- fuera fácilmente extensible para poder integrar soporte a nuevos SMBD sin necesidad de modificar el código de la aplicación,
- fuera altamente portable, pues las plataformas soportadas por los SMBD son diversas,
- tuviera un manejo adecuado y estándar de XML,
- tuviera una interfaz sencilla de utilizar para el usuario y
- fuera utilizada por usuarios expertos en bases de datos, y si tienen conocimientos de programación, fuera extendida por ellos mismos según sus necesites

y como se estudió en el capítulo 6, estos objetivos fueron cuidadosamente considerados al momento de implementar.

En conclusión, tomando los datos expuestos en el capítulo anterior, las implementaciones de PostgreSQL son las más apegadas al estándar. Las implementaciones más pobres y, por tanto, más alejadas del estándar son las de MySQL. Un detalle importante a recalcar es el considerable bajo apego de SQL Server con el estándar, considerando además que se trata de software propietario. Una

posible razón de esto es el intento de algunos productores de software propietario de establecer normas de calidad para el área en el que su producto de software se encuentra, como es el caso de Oracle Corp., quien ha forzado en más de una ocasión a modificar los estándares de calidad, aunque esto implique restar compatibilidad con otros SMBD.

Así pues, el largo recorrido por la teoría detrás del LDM y las estadísticas, los catálogos, el Information Schema y las consultas para extraer los metadatos de las bases de datos, termina en una herramienta sencilla y completa para el estudioso del tema, aunque se pueden pensar varias formas de extender el presente trabajo.

La forma más sencilla de extender el presente trabajo es analizando los catálogos de otros SMBD y crear nuevas implementaciones de `LDMBuilder` para cada uno de ellos.

Otra de ellas puede ser ampliar la GUI de **LDMExtractor** y proporcionar a los usuarios una herramienta visual para la manipulación del LDM de las bases de datos, una forma de poder ver, a manera de diagrama de clases de UML, el esquema lógico de la base de datos analizada. De esta forma podrían crearse también nuevos modelos para después portarlos en formato XML.

Pero también la portabilidad puede ir más allá. Se puede extender la funcionalidad de **LDMExtractor** para que sea capaz de generar código SQL para generar la base de datos que se está analizando. Es bien sabido que, aunque SQL es el lenguaje de definición de datos estándar, cada SMBD utiliza dialectos del mismo que tienen pequeñas diferencias, que los hacen incompatibles unos con otros. Así pues, de la misma forma en que se utiliza una interfaz para especificar la estructura de las clases que representan al analizador del catálogo, también puede crearse una interfaz para constructores de código SQL para cada SMBD que se desee.

Finalmente, como sugiere uno de los trabajos relacionados, **LDMExtractor** puede modificarse hasta ser una interfaz estándar entre diversas fuentes de datos almacenadas en diversos SMBD.

A. Elementos del Information Schema

Este es un breve resumen de las especificaciones del Information Schema en los estándares SQL 99 y SQL 2003. En esta tabla pueden apreciarse los elementos que aparecen en ambos estándares, y se indica en la segunda columna a qué estándar pertenecen.

Nombre del elemento	Estándar	Descripción
INFORMATION_SCHEMA	99, 03	Esquema que identifica al esquema que contiene las tablas del Information Schema
INFORMATION_SCHEMA_CATALOG_NAME	99, 03	Tabla base que identifica al catálogo que contiene el Information Schema
CARDINAL_NUMER	99, 03	Define el dominio que contiene cualquier número no negativo menor que el máximo valor permitido para INTEGER en la implementación
CHARACTER_DATA	99, 03	Define el dominio que contiene cualquier carácter
SQL_IDENTIFIER	99, 03	Define el dominio que contiene todos los <identifier body>s y <delimited identifier body>s[1].
TIME_STAMP	99, 03	Define el dominio que contiene valores de tipo SQL timestamp
APLICABLE_ROLES	99, 03	Vista que identifica los roles aplicables al usuario actual
ASSERTIONS	99, 03	Vista que identifica los *assertions* definidos
ATTRIBUTES	99, 03	Vista que identifica los atributos de los tipos de datos definidos por usuarios

[1] Ver [4] y [3] para más información sobre estos elementos

CHARACTER_SETS	99, 03	Vista que contiene los conjuntos de caracteres disponibles
CHECK_CONSTRAINTS_ ROUTINE_USAGE	03	Vista que identifica todas las rutinas invocadas mediante SQL sobre las cuales las restricciones de dominio, los *table checks* o las *assertions* definidas en el catálogo se encuentran definidas
CHECK_CONSTRAINTS	99, 03	Vista que identifica los *check constraints* definidos en el catálogo
COLLATIONS	99, 03	Vista que contiene los *collations*[1] o filtros disponibles para cada conjunto de caracteres existente
COLLATION_CHARACTER_ SET_APLICABILITY	03	Vista que identifica los conjuntos de caracteres sobre los cuales cada *collation* es aplicable
COLUMN_COLUMN_USAGE	03	Vista que identifica cada caso donde una columna generada depende de una columna base perteneciente a una tabla base
COLUMN_DOMAIN_USAGE	99, 03	Vista que identifica las columnas dependientes de algún dominio presente en el catálogo
COLUMN_PRIVILEGES	99, 03	Privilegios de acceso a nivel columna
COLUMN_UDT_USAGE	99, 03	Vista que identifica las columnas que son dependientes de algún tipo de dato definido por usuarios
COLUMNS	99, 03	Las columnas de las tablas en el catálogo
CONSTRAINT_COLUMN_ USAGE	99, 03	Vista que identifica las columnas que son utilizadas por restricciones referenciales, llaves únicas, *check constraints* y *assertions*
CONSTRAINT_TABLE_ USAGE	99, 03	Vista que identifica las tablas que son utilizadas por restricciones referenciales, llaves únicas, *check constraints* y *assertions*
DATA_TYPE_PRIVILEGES	99, 03	Vista que identifica aquellos esquemas cuyos descriptores de tipo de dato se encuentran disponibles para un usuario dado
DIRECT_SUPERTABLES	99, 03	Vista que identifica las supertablas relativas a una tabla
DIRECT_SUPERTYPES	99, 03	Vista que identifica los supertipos directos relativos a los tipos definidos por usuarios

[1] Es la recopilación de información escrita en un orden estándar. En el uso común, se refiere también a este término como alfabetización, aunque el término no se encuentra limitado a ordenar letras o palabras alfabéticamente.

DOMAIN_CONSTRAINTS	99, 03	Vista que identifica las restricciones de dominio
DOMAIN_UDT_USAGE	99	Vista que identifica los dominios que son dependientes de tipos definidos por usuarios
DOMAINS	99, 03	Vista que identifica los dominios definidos
ELEMENT_TYPES	99, 03	Vista que identifica los tipos permitidos para conformar arreglos
ENABLED_ROLES	99, 03	Vista que identifica los roles activos presentes en el catálogo
FIELDS	99, 03	Vista que identifica los tipos de campos definidos en el catálogo
KEY_COLUMN_USAGE	99, 03	Vista que identifica las columnas definidas en este catálogo que están restringidas como llaves
METHOD_SPECIFICATION_PARAMETERS	99, 03	Vista que identifica los parámetros SQL de los métodos especificados en METHOD_SPECIFICATIONS
METHOD_SPECIFICATIONS	99, 03	Vista que identifica las rutinas invocadas mediante SQL presentes en el catálogo
PARAMETERS	99, 03	Vista que identifica los parámetros SQL de rutinas invocadas mediante SQL
REFERENCED_TYPES	99, 03	Vista que identifica los tipos referenciados por tipos referenciantes definidos en el catálogo
REFERENTIAL_CONSTRAINTS	99, 03	Vista que identifica las restricciones de integridad referencial definidas en el catálogo
ROLE_COLUMN_GRANTS	99, 03	Vista que identifica los privilegios sobre columnas definidas en el catálogo que se encuentran disponibles para o que fueron otorgados por un usuario dado
ROLE_ROUTINE_GRANTS	99, 03	Vista que identifica los privilegios sobre rutinas invocadas mediante SQL definidas en el catálogo que se encuentran disponibles para o que fueron otorgados por un usuario dado
ROLE_TABLE_GRANTS	99, 03	Vista que identifica los privilegios sobre tablas definidas en el catálogo que se encuentran diponibles para o que fueron otorgados por un usuario dado
ROLE_TABLE_METHOD_GRANTS	99, 03	Vista que identifica los privilegios sobre métodos de tablas de tipo estructurado presentes en el catálogo que se encuentran diponibles para o que fueron otorgados por un usuario dado
ROLE_USAGE_GRANTS	99, 03	Vista que identifica los privilegios de USO

		sobre los objetos definidos en el catálogo que se encuentran disponibles para o que fueron otorgados por algún rol actualmente activo
ROLE_UDT_GRANTS	99, 03	Vista que identifica los privilegios sobre tipos definidos que se encuentran disponibles para o que fueron definidos por los roles actualmente activos
ROUTINE_COLUMN_USAGE	99, 03	Vista que identifica las columnas pertenecientes a un usuario dado de las cuales alguna rutina es dependiente
ROUTINE_PRIVILEGES	99, 03	Vista que identifica los privilegios sobre rutinas invocadas mediante SQL y que se encuentran disponibles o fueron otorgados por un usuario dado
ROUTINE_ROUTINE_USAGE	03	Vista que identifica las rutinas invocadas mediante SQL de las cuales alguna rutina es dependiente
ROUTINE_SECUENCE_ USAGE	03	Vista que identifica cada generador externo de secuencias sobre el cual alguna rutina está definida
ROUTINE_TABLE_USAGE	99, 03	Vista que identifica las tablas pertenecientes a un usuario dado de las cuales alguna rutina es dependiente
ROUTINES	99, 03	Vista que identifica las rutinas invocadas por medio de SQL que se encuentran en el catálogo
SCHEMATA	99, 03	Vista que identifica los esquemas existentes en el catálogo que pertenecen a un usuario dado
SECUENCES	03	Vista que identifica cada generador externo de secuencias existente en el catálogo que son accesibles a un usuario o rol dado
SQL_FEATURES	99, 03	Lista las características y subcaracterísticas del estándar e indica cuáles están presentes en la implementación de SQL en cuestión
SQL_IMPLEMENTATION_ INFO	99, 03	Lista los elementos informativos de la implementación de SQL definidos por el estándar y, para cada uno de ellos, indica el valor soportado por la implementación en cuestión
SQL_LANGUAGES	99, 03	Vista que identifica los niveles de conformidad, opciones y dialectos soportados por la implementación de SQL en cuestión
SQL_PACKAGES	99, 03	Lista los paquetes del estándar e indica

		cuáles son soportados por la implementación de SQL en cuestión
SQL_PARTS	03	Lista las partes de la definición del estándar e indica cuáles se encuentran disponibles en la implementación
SQL_SIZING	99, 03	Lista los elementos de dimensionamiento definidos en el estándar y, para cada uno de estos, indica el tamaño soportado por la implementación de SQL
SQL_SIZING_PROFILES	99, 03	Lista los elementos de redimensionamiento definidos en el estándar y, para cada uno de estos, indica el tamaño requerido por uno o más de los perfiles del estándar
TABLE_CONSTRAINTS	99, 03	Describe qué tablas tienen restricciones y de qué tipo son estas
TABLE_METHOD_ PRIVILEGES	99, 03	Vista que identifica los privilegios sobre métodos de tablas de tipo estructurado definidas en aquellos catálogos disponibles para u otorgados por un usuario dado
TABLE_PRIVILEGES	99, 03	Privilegios de acceso a nivel tabla, así como a quién y por quién fueron asignados
TABLES	99, 03	Las tablas en las bases de datos
TRANSFORMS	99, 03	Vista que identifica las transformaciones sobre los tipos definidos por el usuario que se encuentran disponibles
TRANSLATIONS	99, 03	Vista que identifica las traducciones de caracteres disponibles en el catálogo
TRIGGERED_UPDATE_ COLUMNS	99, 03	Vista que identifica las columnas que son identificadas mediante un evento de tipo UPDATE disparado por un disparador
TRIGGER_COLUMN_USAGE	99, 03	Vista que identifica las columnas sobre las cuales están definidos los disparadores y de las cuales son dependientes ya sea por su referencia definida en la condición de búsqueda o por su aparición en una sentencia SQL disparada por un trigger
TRIGGER_ROUTINE_USAGE	03	Vista que identifica las rutinas invocadas mediante SQL sobre las cuales están definidos los disparadores y de las cuales son dependientes
TRIGGER_SECUENCE_ USAGE	03	Vista que identifica los generadores externos de secuencias sobre los cuales están definidos los disparadores y de las cuales son dependientes
TRIGGER_TABLE_USAGE	99, 03	Vista que identifica las tablas sobre las cuales están definidos los disparadores y de las cuales son dependientes

TRIGGERS	99, 03	Definiciones de disparadores
UDT_PRIVILEGES	99, 03	Vista que identifica los permisos sobre los tipos de datos definidos por usuarios
USAGE_PRIVILEGES	99, 03	Vista que identifica los permisos de USO sobre los objetos definidos presentes en el catálogo que se encuentran disponibles o que fueron otorgados por un usuario dado
USER_DEFINED_TYPES	99, 03	Vista que identifica los tipos definidos por usuarios presentes en el catálogo
VIEW_COLUMN_USAGE	99, 03	Vista que identifica las columnas sobre las que las vistas están definidas y son dependientes
VIEW_ROUTINE_USAGE	03	Vista que identifica las rutinas invocadas mediante SQL sobre las que las vistas están definidas y son dependientes
VIEW_TABLE_USAGE	99, 03	Vista que identifica las tablas sobre las que las vistas están definidas y de las que son dependientes
VIEWS	99, 03	Las vistas creadas en las bases de datos

Tabla 6. Los elementos del Information Schema disponibles para un usuario dado.

B. Operaciones definidas por LDMBuilder

LDMBuilder es la interfaz mediante la cual **LDMExtractor** especifica qué operaciones deben ser implementadas para poder tener acceso al modelo lógico de datos de un SMBD específico. Esta interfaz tiene como objetivo forzar un uso estándar de las operaciones, de manera que cada implementación de dicha interfaz es completamente independiente del resto, así como independiente de la forma en que **LDMExtractor** utiliza la información devuelta por las clases que implementan LDMBuilder.

La tabla mostrada a continuación muestra las operaciones definidas por LDMBuilder.

Nombre	Descripción
buildDatabase	Construye un objeto Database representando el LDM de la base de datos especificada según lo encontrado en el catálogo
getMetadata	Obtiene todos los metadatos referentes a la base de datos especificada de una sola vez
getTablesMetadata	Obtiene todos los metadatos referentes a las tablas de la base de datos especificada
getConstraintsMetadata	Obtiene los metadatos referentes a todas las restricciones de la base de datos especificada
getPrimaryKeyConstraintsMetadata	Obtiene todos los metadatos referentes a las llaves primarias de la base de datos especificada
getUniqueKeyConstraintsMetadata	Obtiene todos los metadatos referentes a las llaves únicas de la base de datos especificada
getCheckConstraintsMetadata	Obtiene todos los metadatos referentes a los *check constraints* de la base de datos especificada
getForeignKeyConstraintsMetadata	Obtiene todos los metadatos referentes a las

	llaves foráneas de la base de datos especificada
getStatistics	Obtiene todos los datos estadísticos referentes a tablas y columnas de la base de datos especificada
getTableStatistics	Obtiene todos los datos estadísticos referentes a las tablas de la base de datos especificada
getColumnStatistics	Obtiene todos los datos estadísticos referentes a las columnas de la base de datos especificada
getConn	Intenta, al igual que `setConn`, forzar la existencia de una variable de tipo `DBConnection` en las implementaciones de `LDMBuilder`
setConn	Ver `getConn`

Tabla 15. Métodos de la interfaz LDMBuilder y sus descripciones.

C. Diagrama de clases de LDMExtractor

LDMExtractor es explicado como aplicación en el capítulo 6 del presente trabajo. En este apéndice se muestra el diagrama de clases de cada uno de los paquetes.

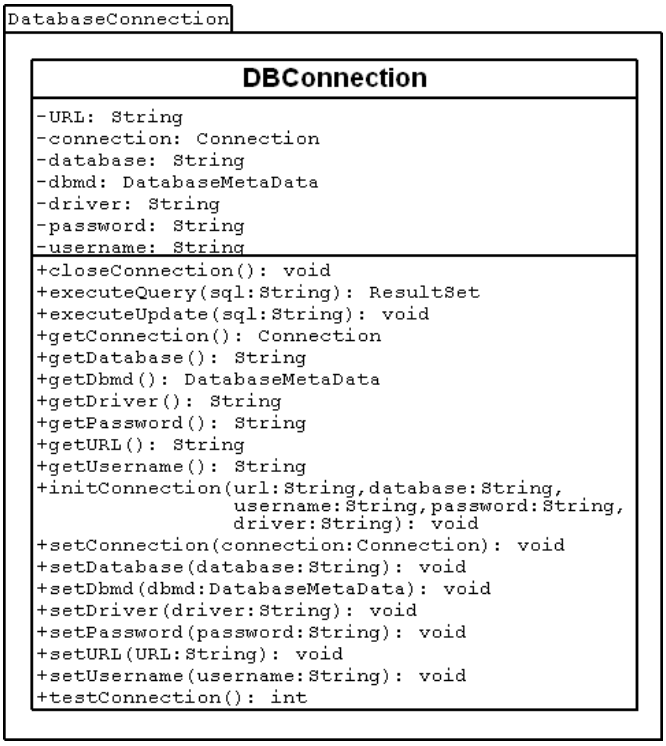

Figura 39. Diagrama del paquete DatabaseConnection

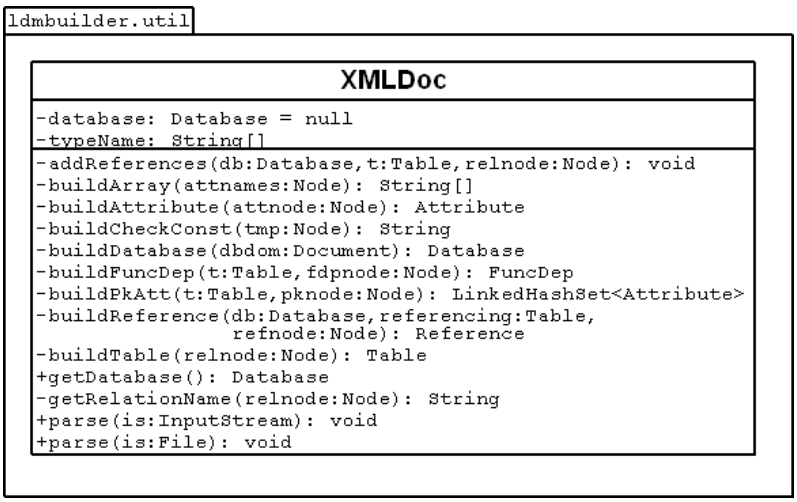

Figura 40. Diagrama del paquete ldmbuilder.util

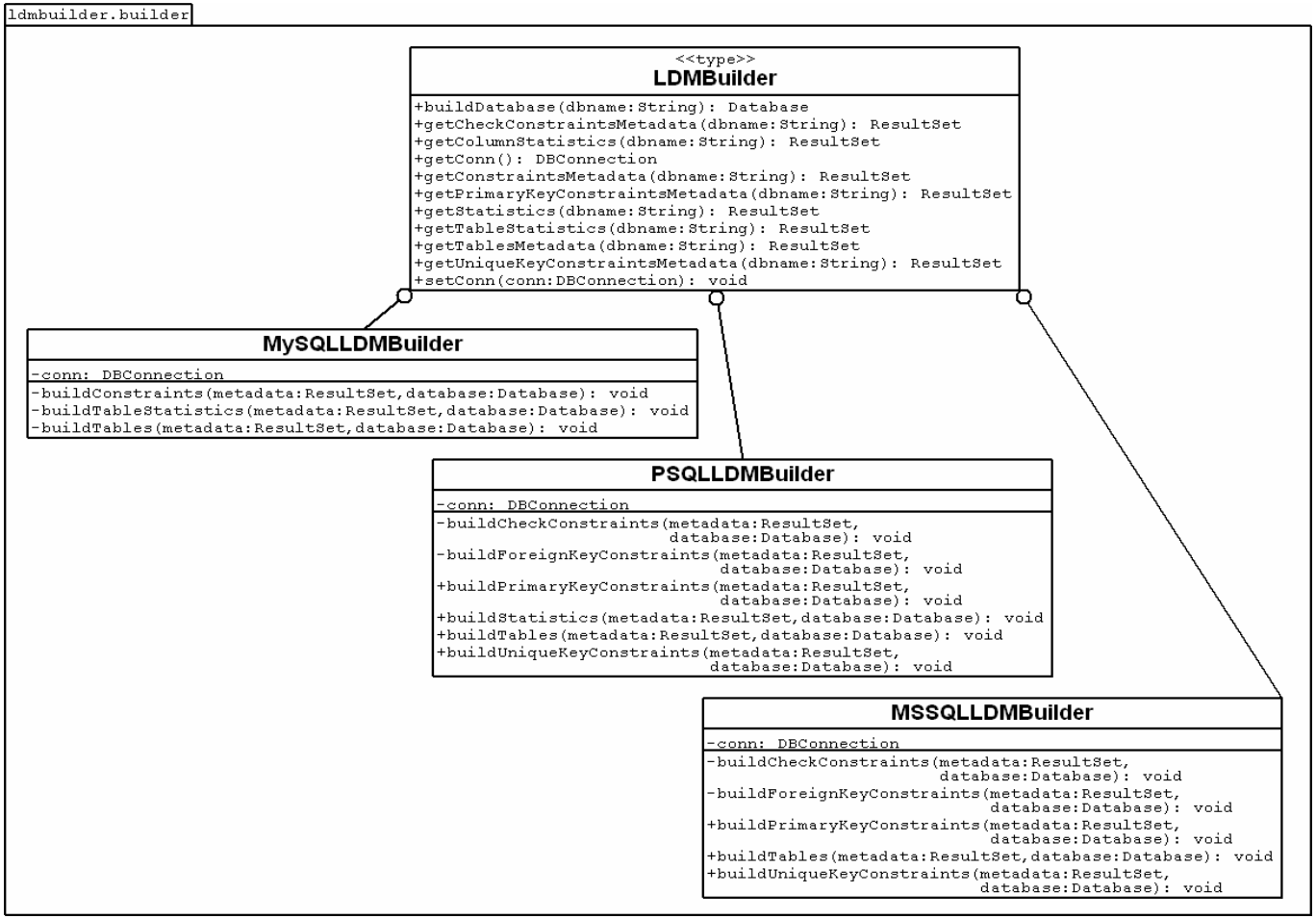

Figura 41. Diagrama del paquete ldmbuilder.builder

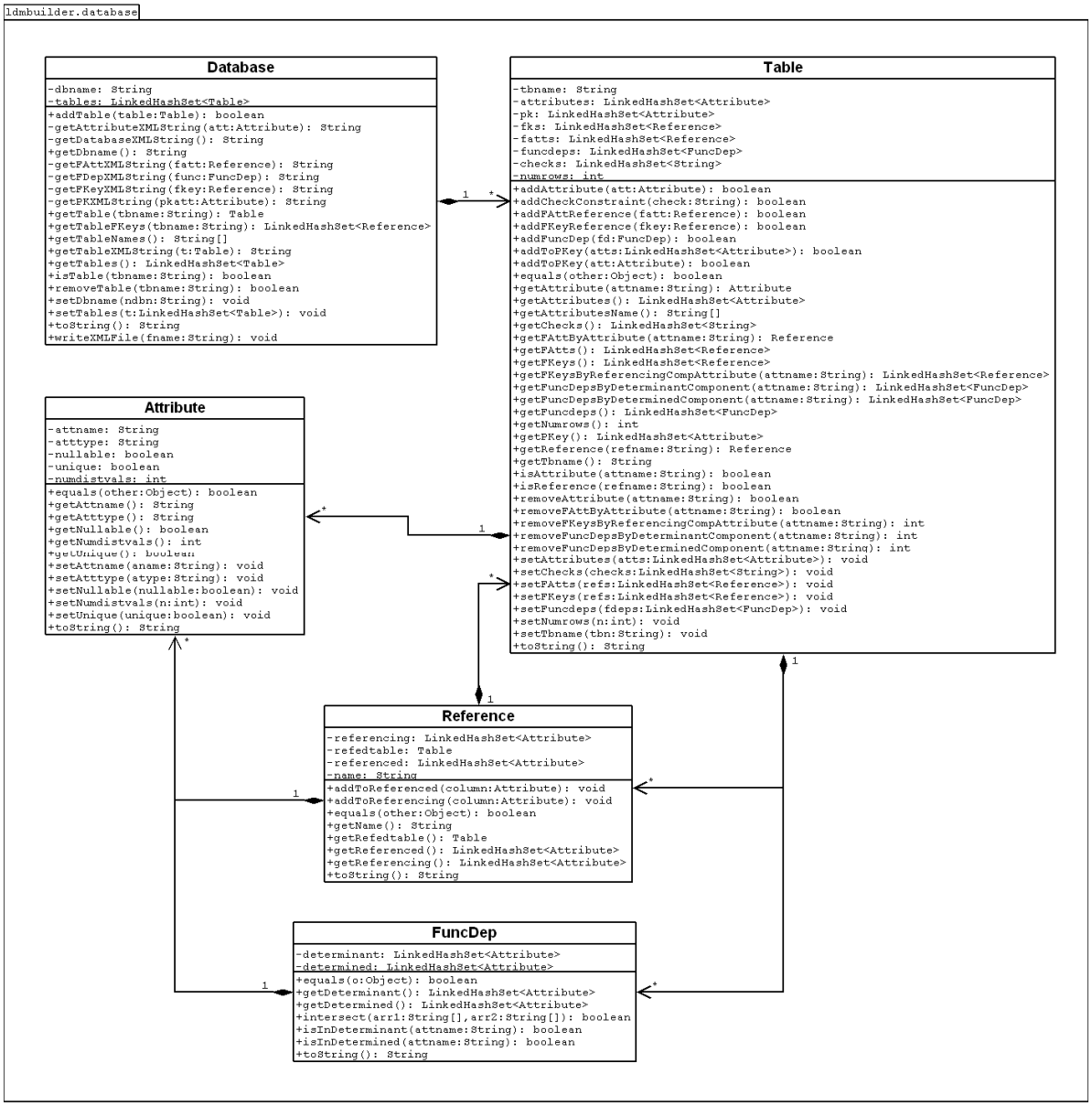

Figura 42. Diagrama del paquete ldmbuilder.database

Bibliografía

[1] MySQL AB. MySQL 5.0 new features: Data dictionary. http://dev.mysql.com/tech-resources/articles/mysql-datadictionary.html.

[2] S. Anahory and D. Murray. Data Warehousing in the Real World: A Practical Guide for Building Decision Support Systems. Addison-Wesley Professional, 1997.

[3] ANSI/ISO/IEC. Database Language SQL - Part 11: Information and Definition Schemas. ANSI/ISO/IEC International Standard (IS), 2003.

[4] ANSI/ISO/IEC. Database Language SQL - Part 2: Foundation(SQL/Foundation). ANSI/ISO/IEC International Standard (IS), 1999.

[5] M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J.N. Gray, P. P. Gri-ths, W. F. King, R. A. Lorie, P. R. McJones, J. W. Mehl, G. R. Putzolu, I. L. Traiger, B. W. Wade, and V. Watson. System r: relational approach to database management. ACM Trans. Database Syst., 1(2):97-137, 1976.

[6] Philip A. Bernstein and Dah-Ming W. Chiu. Using semi-joins to solve relational queries. J. ACM, 28(1):25-40, 1981.

[7] Josh Bosh. PGDesigner. http://www.hardgeus.com/projects/pgdesigner/.

[8] Jin-Yi Cai, Venkatesan T. Chakaravarthy, Raghav Kaushik, and Jeffrey F. Naughton. On the complexity of join predicates. In PODS '01: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pages 207-214, New York, NY, USA, 2001. ACM Press.

[9] Surajit Chaudhuri. An overview of query optimization in relational systems. In PODS '98: Proceedings of the seventeenth ACM SIGACT-SIGMODSIGART symposium on Principles of database systems, pages 34-43, New York, NY,

USA, 1998. ACM Press.

[10] Haengrae Cho. Catalog management in heterogeneous distributed database systems. In Communications, Computers and Signal Processing, 1997. '10 Years PACRIM 1987-1997 - Networking the Pacific Rim'. 1997 IEEE Pacific Rim Conference on, volume 2, pages 659-662, 1997.

[11] E. F. Codd. The relational model for database management: version 2. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[12] Edgar F. Codd. Does your DBMS run by the rules? ComputerWorld, October 21, 1985.

[13] Edgar F. Codd. Is your dbms really relational? ComputerWorld, October 14, 1985.

[14] C. J. Date and Hugh Darwen. A user's guide to the standard database language SQL. Addison-Wesley, 1997.

[15] Brian Dunkel, Qiang Zhu, Wing Lau, and Suyun Chen. Multiple-granularity interleaving for piggyback query processing. In CASCON '99: Proceedings of the 1999 conference of the Centre for Advanced Studies on Collaborative research, page 2. IBM Press, 1999.

[16] J.H. Cross II, E.J. Chikofsky. Reverse engineering and design recovery: A taxonomy in IEEE software. IEEE Computer Society, (2):13-17, January 1990.

[17] Goetz Graefe. Query evaluation techniques for large databases. ACM Comput. Surv., 25(2):73-169, 1993.

[18] PostgreSQL Global Development Group. PostgreSQL documentation. http://www.postgresql.org/.

[19] Mark N. Haynie. Tutorial: The relational data model for design automation. In DAC '83: Proceedings of the 20th conference on Design automation, pages 599-607, Piscataway, NJ, USA, 1983. IEEE Press.

[20] Wen-Chi Hou and Zhongyang Zhang. Enhancing database correctness: a statistical approach. In SIGMOD '95: Proceedings of the 1995 ACM SIGMOD international conference on Management of data, pages 223-232, New York, NY, USA, 1995. ACM Press.

[21] Yannis E. Ioannidis. Query optimization. ACM Comput. Surv., 28(1):121-123, 1996.

[22] Azzurri Ltd. Database modeling in eclipse.

http://www.azzurri.jp/en/software/clay/index.jsp.

[23] Microsoft Developer Network. SQL Server 2005 documentation. http://msdn2.microsoft.com/en-us/library/, 2007.

[24] Carlos Ordonez and Javier García-García. Vector and matrix operations programmed with udfs in a relational dbms. In CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management, pages 503-512, New York, NY, USA, 2006. ACM Press.

[25] Carlos Ordonez and Javier García-García. Referential integrity quality metrics. In Decision Support Systems Journal (DSS). Elsevier, 2007.

[26] Fabio A. Schreiber and G. Martella. Creating a conceptual model of a data dictionary for distributed data bases. SIGMIS Database, 11(1):12-18, 1979.

[27] Transaction Processing Performance Council (TPC). TPC Benchmarck H (Decision Support) Standard Specification. Transaction Processing Performance Council (TPC), revision 2.6.0 edition, 2006.

[28] Patrick Valduriez. Semi-join algorithms for multiprocessor systems. In SIGMOD '82: Proceedings of the 1982 ACM SIGMOD international conference on Management of data, pages 225-233, New York, NY, USA, 1982. ACM Press.

[29] Iraklis Varlamis and Michalis Vazirgiannis. Bridging xml-schema and relational databases: a system for generating and manipulating relational databases using valid xml documents. In DocEng '01: Proceedings of the 2001 ACM Symposium on Document engineering, pages 105-114, New York, NY, USA, 2001. ACM Press.

[30] W3C XML CoreWorking Group (WG). Extensible markup language (xml) 1.0 (fourth edition). http://www.w3.org/TR/REC-xml/.

[31] Qiang Zhu, Brian Dunkel, Nandit Soparkar, Suyun Chen, Berni Schiefer, and Tony Lai. A piggyback method to collect statistics for query optimization in database management systems. In CASCON '98: Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research, page 25. IBM Press, 1998.