Trabajo de Fin de Grado

Grado en Ingeniería Informática

JITRAX: Intérprete en Java para Álgebra Relacional

JITRAX: Java Interpreter for Relational Algebra

Teguayco Gutiérrez González

La Laguna, 5 de junio de 2017

D. **Jesús Manuel Jorge Santiso**, con N.I.F. 42.097.398-S profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

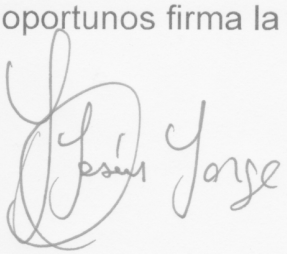
C E R T I F I C A

Que la presente memoria titulada:

"JITRAX: Intérprete en Java para Álgebra Relacional"

ha sido realizada bajo su dirección por D. **Teguayco Gutiérrez González**, con N.I.F. 54.117.339-H.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firma la presente en La Laguna a 5 de junio de 2017.



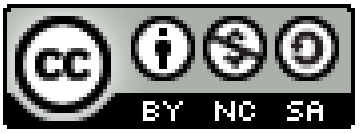
Fdo: Jesús M. Jorge Santiso

Agradecimientos

A mis padres, cuyos esfuerzos y sacrificios los convierten en los principales artífices de que a día de hoy pueda estar cursando estudios universitarios.

A mi tutor, Jesús Manuel Jorge Santiso, quien me ha prestado una valiosa ayuda durante todo el desarrollo de este trabajo.

Licencia



Resumen

El objetivo de este trabajo ha sido desarrollar una aplicación en Java que permita interpretar y ejecutar consultas del lenguaje teórico para bases de datos Álgebra Relacional.

Para alcanzar dicha meta, la técnica que se propone es la siguiente: traducir las consultas insertadas en Álgebra Relacional a consultas equivalentes del lenguaje comercial SQL y ejecutar estas últimas sobre un SGBD -en principio, PostgreSQL-, de manera que el usuario pueda comprobar de manera gráfica qué efectos ha tenido la ejecución de las consultas que ha escrito sobre la base de datos con la que trabaja.

El propósito fundamental de esta herramienta es académico y metodológico, puesto que se pretende mejorar el proceso de aprendizaje del citado lenguaje por parte de aquellos estudiantes que se inician en el estudio de las bases de datos relacionales.

Palabras clave: *Álgebra Relacional, Java, SQL, Sistema Gestor de Bases de Datos, SGBD, PostgreSQL, interpretación y reconocimiento de lenguajes.*

Abstract

The goal of this work has been the development of a desktop application written in Java which allows the interpretation and execution of Relational Algebra queries, which is a theoretical language for relational databases.

In order to reach this aim, the proposed technique is as follows: to translate the entered Relational Algebra queries to their equivalent in the commercial language SQL and, then, execute the latter on a DBMS -in principle, PostgreSQL-. Thus, the user will be able to check in a graphical way the effects of the execution of these queries on the database he is working with.

The fundamental purpose of this tool is academical and methodological, as it claims to enhance the learning process of the mentioned language by those students who begin in the study of relational databases.

Keywords: *Relational Algebra, Java, SQL, Database Management System, DBMS, PostgreSQL, language recognition and interpretation.*

Índice General

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

1.1 Introducción

Dentro del ámbito de las bases de datos, el Álgebra Relacional se erige como un lenguaje teórico de consultas cuya comprensión debe hacerse efectiva por parte de aquellos que pretenden iniciarse en el estudio de las consultas sobre bases de datos relacionales. Además, constituye una poderosa herramienta que permite, entre otros, establecer un marco para la medición de la potencia expresiva de los lenguajes comerciales, tales como el ampliamente conocido SQL.

Esta importancia del Álgebra Relacional es la que le confiere sentido al desarrollo de este TFG, donde nos proponemos desarrollar una herramienta gráfica que haga que el proceso de aprendizaje de este lenguaje de consultas sea más sencillo y eficiente. A la herramienta la hemos denominado JITRAX, acrónimo de *Java Interpretation Tool for Relational Algebra Expressions*, y es básicamente un traductor de Álgebra Relacional a SQL con capacidad de ejecutar consultas.

No obstante, estamos ante una herramienta cuyo uso puede llegar incluso más allá: el de proporcionar una capa de abstracción a los profesionales para realizar consultas sobre bases de datos alojadas en un Sistema Gestor real. Así, se lograría la ejecución de consultas sobre bases de datos reales sin tener en consideración aspectos de más bajo nivel como los que poseen los lenguajes comerciales.

Recientemente, una versión preliminar de esta herramienta desarrollada ha conseguido los siguientes reconocimientos: 2º premio en el Concurso Universitario de Software Libre de la ULL celebrado a nivel local y premio al *Mejor Proyecto Educativo* a nivel estatal organizado por la Universidad de Sevilla.

1.2 Objetivos

El principal objetivo de este TFG consiste en desarrollar una aplicación que permita interpretar y ejecutar sentencias de Álgebra Relacional, para así favorecer la comprensión del lenguaje.

Adicionalmente, la herramienta pretende:

- Mejorar la capacidad de escritura de consultas complejas.
- Facilitar el desarrollo de consultas más claras y eficientes mediante el estudio del código SQL que la herramienta genera.

1.3 Estado del arte

En este apartado comentaremos el estado actual de las herramientas existentes que, en mayor o menor medida, persiguen los mismos objetivos citados anteriormente y que, por tanto, comparten rasgos similares con el software desarrollado en este trabajo.

1.3.1 RelaX: Relational Algebra Calculator

Se trata de una aplicación en formato web desarrollada en la Universidad de Innsbruck, Austria. Implementa de por sí un motor de consultas en Álgebra Relacional que permite “simular” la ejecución de las consultas insertadas por el usuario sobre el esquema de tablas ficticio con el que se trabaja. Así, se mostrará posteriormente una vista con la tabla de resultados que genera la consulta introducida.

De manera complementaria, proporciona funcionalidad similar para experimentar con el lenguaje SQL.

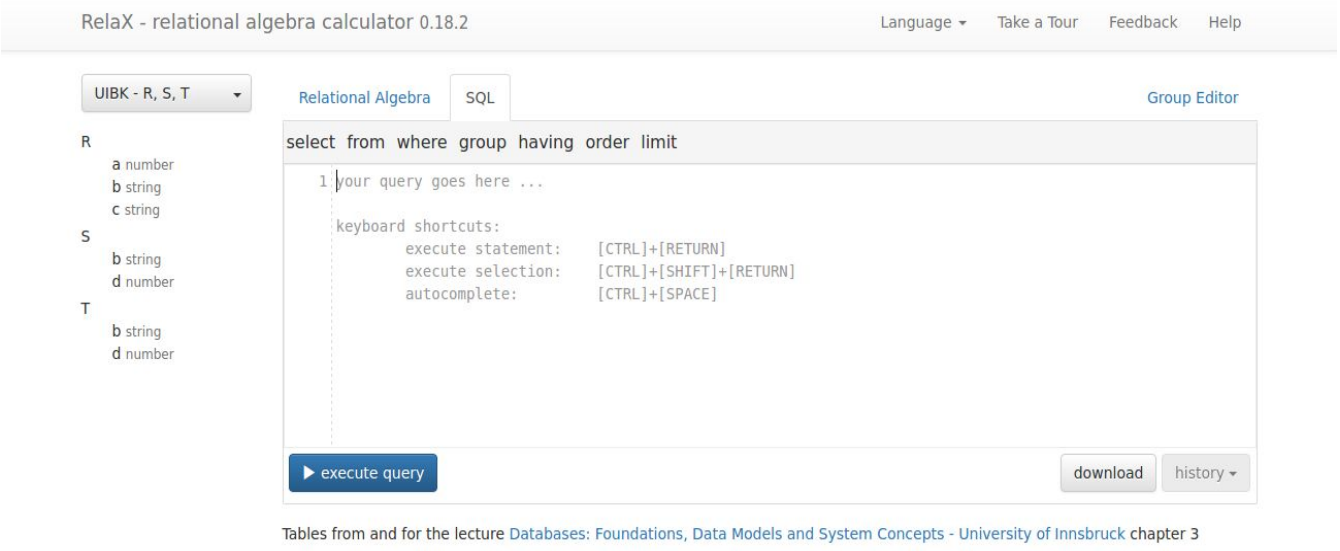


Figura 1.1: RelaX

Después de haber hecho uso de esta herramienta, se puede concluir que se trata de una opción muy completa: permite definir nuevos esquemas de

bases de datos al usuario y la lista de operadores soportados para el Álgebra Relacional incluye a los más comunes. Además, es posible utilizar vistas del Á. R. y comentarios en las consultas insertadas.

Sin embargo, implementa un motor de traducción de SQL a Álgebra Relacional que resulta muy básico -por ejemplo, no funciona cuando se introducen subconsultas anidadas-. Tampoco ofrece ningún mecanismo para el manejo de múltiples consultas -si se desea escribir una segunda consulta, la primera que se haya insertado desaparece del entorno-.

1.3.2 WinRDBI: Windows Relational Database Interpreter

WinRDBI es una aplicación de escritorio para Windows creada por la *Arizona State University* que permite experimentar tanto con el Álgebra Relacional como con el Cálculo Relacional de Tuplas y Dominios. Al igual que RelaX, las tablas con las que operar se definen dentro de la propia aplicación y son, por tanto, ficticias.

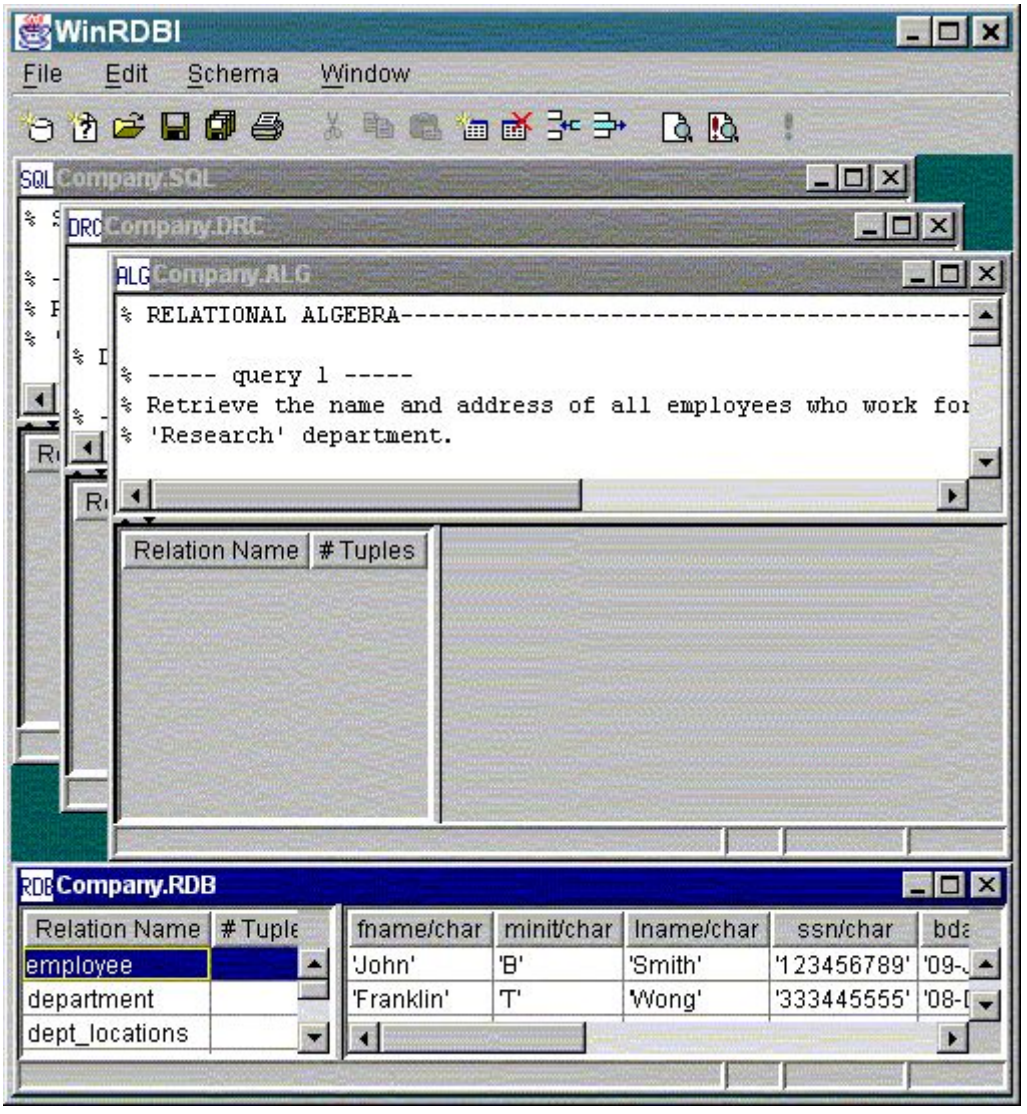


Figura 1.2: WinRDBI

Entre otros, permite el manejo simultáneo de consultas y la carga de bases de datos desde diferentes formatos de archivo, los cuales son posteriormente modificables desde el propio entorno. No obstante, no permite la optimización de consultas y no incluye funcionalidad relacionada con la traducción de lenguajes.

1.3.3 RAT: Relational Algebra Translator

Tal y como denotan sus siglas, RAT es un traductor de expresiones en Álgebra Relacional a lenguaje SQL, desarrollado en la Universidad Nacional de Costa Rica.

Permite también visualizar el árbol de análisis sintáctico que resulta del proceso de interpretar las expresiones en Álgebra y, opcionalmente, conectarse a un SGBD para, opcionalmente, mostrar los resultados de las consultas de forma visual.

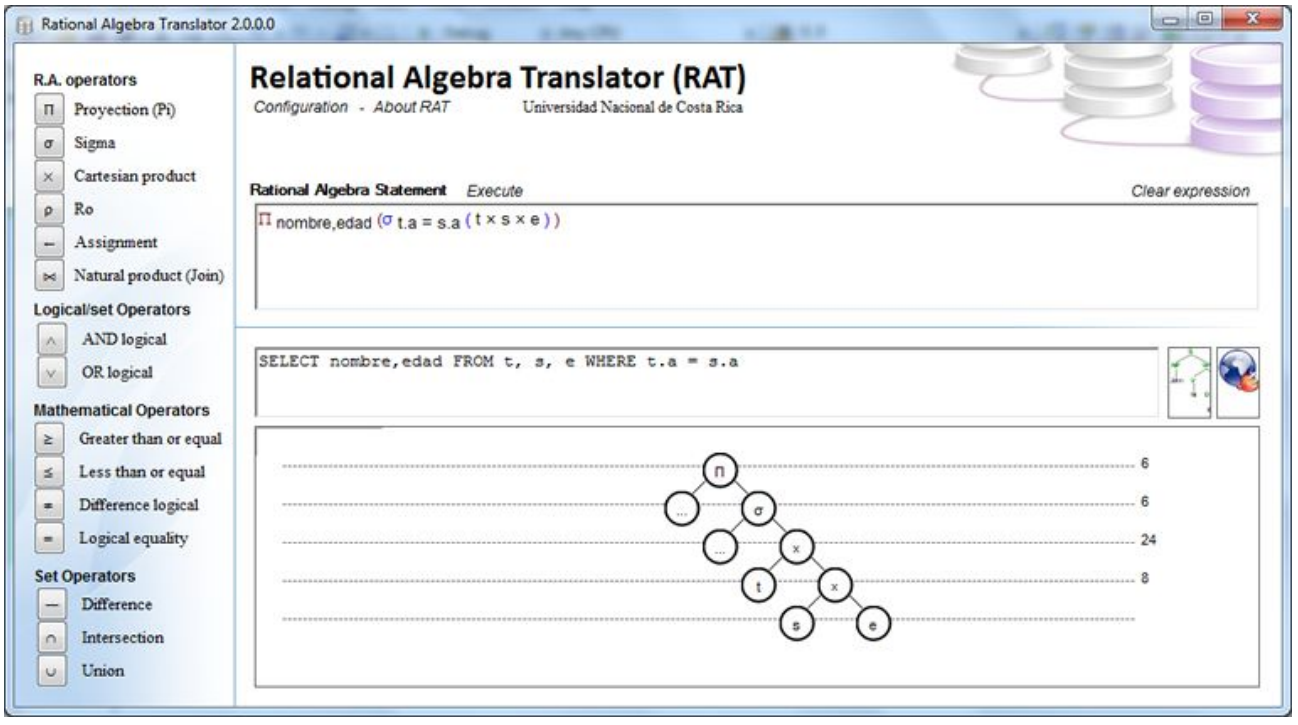


Figura 1.3: RAT

Cuenta con un conjunto de operadores más limitado, en el que destaca la ausencia del operador cociente. Tampoco permite el manejo simultáneo de consultas en Álgebra Relacional y el código SQL generado se muestra sin ningún formato.

1.4 Planificación

Se definen en este apartado cuáles han sido las actividades llevadas a cabo durante el desarrollo de este proyecto, así como las franjas temporales y recursos de los que se disponen para su cumplimiento.

1.4.1 Actividades realizadas

Las actividades llevadas a cabo se exponen a continuación, ordenadas por orden cronológico en función del momento en que fueron realizadas:

1. **Análisis y definición de los requerimientos**.
2. **Revisión bibliográfica.** Consulta y estudio del material bibliográfico y documentación acerca del estado del arte del problema y las herramientas que se utilizarán para acometer el desarrollo.
3. **Diseño e implementación de la aplicación de escritorio.** Ésta incluirá la funcionalidad descrita en el capítulo 4.
4. **Detección y corrección de errores**, como consecuencia de la realización de pruebas sobre el software desarrollado.
5. **Documentación.** Redacción de la memoria del TFG y documentación adicional sobre el uso de la herramienta implementada.

1.4.2 Plan de trabajo

La principal herramienta para poder realizar este proyecto es el entorno de desarrollo integrado Eclipse, PostgreSQL y se hará uso de otras herramientas software para realizar las actividades anteriormente especificadas, tales como: *ANTLRv4* para el reconocimiento del lenguaje del Álgebra Relacional, *RSyntaxTextArea* como editor de código con resaltado sintáctico para incrustar en la interfaz gráfica y *JDBC* para establecer conexión con el/los SGBD sobre los que ejecutar las sentencias traducidas.

En cuanto a los recursos temporales, se dispone de los cuatro meses que conforman el periodo comprendido entre el inicio del proyecto (febrero) y la convocatoria de exámenes de junio.

A continuación se muestra la planificación de las actividades a realizar, su secuenciación y la duración que se estima para cada una de ellas en base a las 15 semanas de las que se dispone para ejecutar el desarrollo -nótese que hay tareas que se ejecutaron en paralelo-:

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15
A1															
A2															
A3															
A4															
A5															

Tabla 1.1: Planificación de las actividades a realizar

Capítulo 2.
Fundamentos teóricos

Una vez que se ha puesto en contexto el desarrollo de la herramienta objeto de este trabajo, se pretende realizar en este capítulo un repaso de todos aquellos fundamentos teóricos que han sido necesarios para abordar la realización del mismo.

2.1 Modelo Relacional

2.1.1 Descripción

El denominado "Modelo Relacional" es el fundamento teórico de más importancia que ha contribuido al diseño de sistemas de bases de datos en la actualidad. Fue propuesto por Edgar F. Codd en 1970 en su artículo *"A Relational Model of Data for Large Shared Data Banks"* [1].

Este modelo de datos propuesto tiene como idea fundamental el uso de **relaciones**, que son aquellas estructuras en las que los datos se agrupan de manera lógica en forma de filas (tuplas). Así, podemos visualizar una *relación* en forma de tabla, tal y como se muestra a continuación:

atributo 1	atributo 2	…	atributo n	
XXX	XXX	...	XXX	➡ tupla 1
XXX	XXX	...	XXX	➡ tupla 2
...	...	...	...	
XXX	XXX	XXX	XXX	➡ tupla m

Figura 2.1: Representación de una relación en forma de tabla

Cada relación tiene un **nombre** que las identifica de manera unívoca en el modelo de datos y una lista de **atributos** que representan las propiedades de la tabla (correspondientes a las columnas de la tabla en la figura 2.1). Además, se tiene un conjunto de filas llamadas **tuplas** que contienen los valores que toman cada uno de los atributos para cada

elemento de la relación.

Otros conceptos importantes que cabe reseñar son el **grado** de una relación (el número de atributos que posee), la **cardinalidad** (número de tuplas) y **dominio de los atributos** (conjunto de todos los posibles valores que puede tomar un determinado atributo).

A continuación se muestra, a modo de ejemplo, una relación llamada *Autor*, compuesta por varias tuplas de datos relativos a distintos *autores*:

Nombre	Nacionalidad	Institución
Date, C. J.	**Norteamericana**	**Relational Institute**
Codd, E. F.	**Norteamericana**	**Relational Institute**
Ceri, S.	**Italiana**	**Politécnico de Milán**
Santos, F.	**Española**	**U.P.C**

2.2.2 Restricciones inherentes al modelo

En la práctica, todas aquellas relaciones de bases de datos de carácter relacional se representan a modo de tabla (tal y como se ha hecho en el apartado anterior). Esto ha provocado que se haya adoptado una terminología que en su día no recogió Codd cuando formuló el modelo: a la propia relación se la denomina *tabla*, a los atributos se les suele llamar *columnas* y, a las tuplas, *filas*.

Sin embargo, se debe insistir en que una relación en este modelo de datos tiene una serie de restricciones que la distinguen del concepto de tabla:

- No admiten filas duplicadas.
- El orden de las filas es irrelevante.
- No se permiten atributos *multivaluados*; es decir, un determinado atributo no puede tener más de un valor.

2.2 Álgebra Relacional

2.2.1 Historia e importancia

El Álgebra Relacional surge de la mano de Edgar Frank Codd cuando este científico inglés propone el ya comentado “Modelo Relacional”.

Así, el Álgebra Relacional es un lenguaje de bases de datos constituido por un conjunto de operaciones que permiten realizar consultas sobre una base de datos relacional. Además, se dice que este lenguaje es de **tipo procedimental** ya que, mediante su uso, se define la forma en que se debe recuperar la información del sistema -en contraposición a los de tipo declarativo o no procedimental (como es el caso del Cálculo Relacional), donde únicamente se describe qué información se desea obtener-.

La importancia del Álgebra Relacional yace principalmente en que, a partir de él, fue creado el SQL (*Structured Query Language*), el lenguaje de consultas predominante en el panorama actual de las bases de datos. Por ello, aprender Álgebra Relacional antes que SQL supone realizar mejores consultas en el segundo (tanto en escritura como en eficiencia) una vez que hayan sido comprendidos a priori los fundamentos del primero.

2.2.2 Definición

El Álgebra Relacional es un sistema cerrado de operaciones definidas sobre relaciones. Es decir, tanto los operandos como los resultados son relaciones. Esto permite construir expresiones combinando unas operaciones con otras, de manera que los resultados de unas sean operandos de otras.

Con más precisión:

> Dado un conjunto de dominios, $D = (D_1, D_2, \ldots, D_n)$, sea **R** el conjunto de todas las relaciones posibles que se puedan construir sobre D, inclusivo relaciones sin nombre o sin nombres de atributos. El álgebra relacional es un sistema formado por **R** y unos operadores cuyos operandos y resultados también pertenecen a **R**.

2.2.3 Conjunto de operadores

Se describen a continuación cuáles son los operadores del Álgebra Relacional a los que la herramienta desarrollada ofrece soporte. Para ilustrar mediante ejemplos la aplicación de cada uno de los operadores, se describen las relaciones que se utilizarán como operandos, **R** y **S**:

R	**A**	**B**	**C**
	a	b	c
	d	a	f
	c	b	d

S	**D**	**E**	**F**
	b	g	a
	d	a	f

Por un lado, se encuentran los denominados **operadores básicos** del Álgebra Relacional: proyección, selección, producto cartesiano, unión y diferencia. Los operadores de este tipo se caracterizan por no poderse reescribir en función de otros operadores, de manera que su eliminación del conjunto de operadores implicaría la pérdida de potencia expresiva del lenguaje.

Por otro lado, existe otro tipo de operadores denominados **no básicos o derivados**. Son aquellos que pueden escribirse en función de operadores básicos y, por ello, su eliminación del conjunto de operadores no implicaría que el lenguaje perdiese potencia expresiva. Los operadores derivados a los que la aplicación desarrollada ofrece soporte son los siguientes: intersección, yunción, yunción natural y cociente.

2.2.3.1 Proyección

El operador *proyección* actúa sobre una relación dada y se representa mediante la palabra reservada `PROJECT`. La expresión

$$\texttt{PROJECT } [A_1, A_2, \ldots, A_n] \texttt{ (R)}$$

devuelve una relación cuyas columnas (A_1, A_2, …, A_n) y sus valores correspondientes se extraen a partir de R. Además, se eliminan luego las tuplas duplicadas.

Ejemplo. Sea la relación R(A, B, C) anteriormente descrita. La expresión `PROJECT [A, B] (R)` genera el siguiente resultado:

PROJECT [A, B, C] (R)	**A**	**B**
	a	b
	d	a
	c	b

2.2.3.2 Selección

El operador *selección* devuelve una relación anónima cuyas tuplas son aquellas de la relación **R** que satisfacen el predicado lógico **F**. El esquema de la relación resultante coincide con el de R.

```
SELECT [F] (R)
```

Ejemplo. Sea la relación R(A, B, C) anteriormente descrita. La expresión `SELECT [B = 'b'] (R)` genera el siguiente resultado:

SELECT [B = 'b'] (R)	A	B	C
	a	b	c
	c	b	d

2.2.3.3 Producto cartesiano

Efectuar un *producto cartesiano* entre dos tablas devuelve como resultado la concatenación de cada una de las tuplas de la primera relación con cada una de las tuplas de la segunda. La relación resultante tiene como esquema aquel que resulta de la concatenación de las listas de atributos de las dos relaciones que actúan como operando.

```
R x S
```

Ejemplo. Sean las relaciones R(A, B, C) y S(D, E, F) anteriormente descritas. El producto cartesiano entre ambas relaciones genera como resultado la siguiente relación:

R x S	A	B	C	D	E	F
	a	b	c	b	g	a
	a	b	c	d	a	f
	d	a	f	b	g	a
	d	a	f	d	a	f
	c	b	d	b	g	a
	c	b	d	d	a	f

2.2.3.4 Unión

El resultado de aplicar el operador *unión* sobre dos relaciones R y S es otra relación con el mismo grado y los mismos dominios que sus operandos. Así, su contenido lo forman todas aquellas tuplas que se encuentran en R o en S (sin duplicados).

```
R U S
```

Para poder aplicar este operador sobre dos relaciones, éstas deben ser ***compatibles***: ambas deben ser de igual grado y los dominios de los atributos i-ésimos deben coincidir.

Ejemplo. Sean las relaciones R(A, B, C) y S(D, E, F) anteriormente descritas. El resultado de la unión entre ambas es el siguiente:

R U S			
	a	b	c
	d	a	f
	c	b	d
	b	g	a

2.2.3.5 Diferencia

La relación que resulta de aplicar el operador *diferencia* sobre dos relaciones R y S es aquella que se compone de las tuplas que están en R pero no es S. Ambas relaciones deben ser compatibles.

```
R - S
```

Ejemplo. Sean las relaciones R(A, B, C) y S(D, E, F) anteriormente descritas. La operación `R - S` genera como resultado la relación siguiente:

R - S			
	a	b	c
	c	b	d

2.2.3.6 Intersección

El operador *intersección* efectuado sobre dos relaciones R y S devuelve una nueva relación compuesta por las tuplas de R que también se encuentran en S. Las dos relaciones que actúan como operandos deben ser compatibles.

```
R ∩ S
```

Ejemplo. Sean las relaciones R(A, B, C) y S(D, E, F) anteriormente descritas. El resultado de la intersección entre ambas es el siguiente

R ∩ S			
	d	a	f

2.2.3.7 Yunción

Efectuar una *yunción* entre dos relaciones es equivalente a realizar sobre ellas un producto cartesiano y, a partir de la tabla resultante, seleccionar aquellas tuplas que satisfacen una condición lógica F.

```
R Y S [F]
```

Ejemplo. Sean las relaciones R(A, B, C) y S(D, E, F) que a continuación se definen.

R	**A**	**B**	**C**
	1	2	3
	4	5	6
	7	8	9

S	**D**	**E**
	3	1
	6	2

La yunción `R Y S [B < D]` genera como resultado la siguiente relación:

R Y S [B < D]	**A**	**B**	**C**	**D**	**E**
	1	2	3	3	1
	1	2	3	6	2
	4	5	6	6	2

2.2.3.8 Yunción natural

El operador *yunción natural* efectúa un producto cartesiano entre R y S y selecciona aquellas tuplas con igualdad de valores en los atributos homónimos de ambas relaciones.

```
R * S
```

Ejemplo. Sean las relaciones R(A, B, C) y S(B, C, D) que a continuación se definen.

R	A	B	C
	a	b	c
	d	b	c
	b	b	f
	c	a	d

S	B	C	D
	b	c	d
	b	c	e
	a	d	b

El resultado de efectuar una yunción natural sobre las dos relaciones anteriores es el siguiente:

R * S	A	B	C	D
	a	b	c	d
	a	b	c	e
	d	b	c	d
	d	b	c	e
	c	a	d	b

2.2.3.9 Cociente

Para poder efectuar el operador *cociente* o *división* sobre entre dos relaciones R y S, el esquema de R debe contener estrictamente el esquema de S.

```
R / S
```

Sean R(A, B) y S(B), `R/S`(A) devuelve todos aquellos valores de *a* (sin duplicados) tal que, para todo valor *b* en S, existe una tupla de la forma 〈*a, b* 〉 en R.

Ejemplo. Sean las relaciones R(A, B, C, D) y S(C, D) que a continuación se definen.

R	A	B	C	D
	a	b	c	d
	a	b	e	f
	a	b	d	e
	b	c	e	f
	e	d	c	d
	e	d	e	f

S	C	D
	c	d
	e	f

El cociente entre ambas relaciones, `R/S`, da como resultado la relación siguiente:

R / S	A	B
	a	b
	e	d

2.2.4 Vistas

Otro aspecto incluido en la herramienta son las denominadas *vistas* del Álgebra Relacional, que permiten asignar un alias a una determinada consulta que permita su posterior reutilización en otra expresión. En la siguiente consulta en ÁR, se declara una vista llamada *T*, la cual es utilizada posteriormente en otra expresión:

```
T = PROJECT [A, B] (R)
SELECT [A >= C] (T x S)
```

2.3 SQL

El lenguaje SQL (*Structured Query Language*; del inglés *Lenguaje de Consulta Estructurado*) es el lenguaje de consultas para bases de datos relacionales predominante en el panorama actual.

En este apartado se pretende esbozar primeramente cuál ha sido la evolución del lenguaje y cuáles son las características principales que reúne para, posteriormente, comentar la analogía existente entre éste y el Álgebra

Relacional que ha servido como base para la implementación del motor de traducción que forma parte de la herramienta desarrollada en este TFG.

2.3.1 Orígenes y evolución

El SQL encuentra su origen en un lenguaje creado en los laboratorios de IBM en 1974 denominado SEQUEL (*Structured English QUEry Language*). Más tarde, fue introducido como lenguaje de consultas para un SGBD experimental llamado *System R* (1977), aunque no fue hasta 1979 cuando el SQL fue reconocido como un lenguaje comercial de la mano de la compañía estadounidense *Oracle*.

Con el paso del tiempo, han visto la luz multitud de versiones del lenguaje, hasta que éste haya llegado a imponerse como un estándar en la actualidad puesto que, con variaciones, la gran mayoría de productos lo incorporan. Así, esta circunstancia se presenta en nuestros días como el resultado de un largo e intenso proceso de normalización.

2.3.2 Características

Como se ha visto, el SQL fue originariamente creado a partir del Álgebra y el Cálculo Relacional propuestos por Codd de forma posterior al Modelo Relacional. Es por esto que el SQL puede ser descrito como un lenguaje de declarativo (característica que le confiere el Cálculo Relacional) y, también, como un lenguaje procedimental (dada su procedencia del Álgebra Relacional).

No obstante, el alcance de SQL va mucho más allá, puesto que constituye en sí mismo un lenguaje de definición, manipulación y control de datos:

- **Lenguaje de definición de datos** (DDL, *Data Definition Language*): ofrece funcionalidad para definir las estructuras que la base de datos utilizará para albergar los datos de manera lógica (fundamentalmente, las tablas). Destacan principalmente las funciones de creación de estructuras (CREATE), de borrado (DROP) y de actualización (ALTER).
- **Lenguaje de manipulación de datos** (DML, *Data Manipulation Language*): permite llevar a cabo operaciones de consultas o modificación sobre los datos contenidos en las estructuras comentadas en el punto anterior. En SQL, las sentencias más conocidas son SELECT, para la realización de consultas; INSERT, para añadir nuevos datos; DELETE, para borrarlos; y UPDATE, para actualizarlos.
- **Lenguaje de control de datos** (DCL, *Data Control Language*): permite establecer restricciones de acceso a los datos. Mediante los comandos GRANT y REVOKE se puede dar y eliminar permisos, respectivamente,

tantos a usuarios como a roles.

2.3.3 Álgebra Relacional como subconjunto del SQL

Se hará hincapié en este apartado en la similitud, en cuanto a operaciones, que guardan los lenguajes del SQL y el Álgebra Relacional (como resultado de que el primero se creó a partir del segundo) y la cual ha servido como base para la implementación del motor de traducción de consultas.

Algunos de los operadores del Álgebra Relacional (comentados en detalle en el apartado 2.2.3) encuentran su equivalente en el SQL de una forma casi directa, de modo que la traducción sea prácticamente literal. No obstante, otros operadores más complejos como el cociente, tienen traducciones que no se generan de forma tan trivial.

A continuación, se expone cuáles han sido las traducciones propuestas para cada uno de los operadores del Álgebra Relacional soportados por el intérprete desarrollado -pueden verse en el apéndice B algunos ejemplos de traducciones más complejas-.

2.3.3.1 Proyección

El operador proyección del Álgebra Relacional encuentra su equivalente en el SQL por medio de la cláusula SELECT (cuyo efecto en nada se compara con el operador selección del Álgebra Relacional).

```
PROJECT [A, B] (R) ➡ SELECT A, B
                     FROM R;
```

2.3.3.2 Selección

La cláusula WHERE del SQL toma su función a partir de la selección del AR:

```
SELECT [A > 1] (R) ➡ SELECT *
                     FROM R
                     WHERE A > 1;
```

2.3.3.3 Producto cartesiano

No hay ninguna palabra reservada en el SQL para el producto cartesiano, pero su ejecución se consigue enumerando las tablas implicadas en la cláusula FROM:

```
R x S ➡ SELECT *
        FROM R, S;
```

2.3.3.4 Unión

La traducción del operador unión del AR es prácticamente literal en SQL:

```
R U S ➡ SELECT *
        FROM R
        UNION
        SELECT *
        FROM S;
```

2.3.3.5 Diferencia

En la expresión equivalente en SQL para la diferencia del Álgebra Relacional se hace uso de la palabra reservada *EXCEPT*:

```
R - S ➡ SELECT *
        FROM R
        EXCEPT
        SELECT *
        FROM S;
```

2.3.3.6 Intersección

Al igual que para la unión y la diferencia, la intersección también tiene una traducción prácticamente literal a SQL:

```
R ∩ S ➡ SELECT *
        FROM R
        INTERSECT
        SELECT *
        FROM S;
```

2.3.3.7 Yunción

Para la *yunción*, se utiliza la estructura **INNER JOIN ON**:

```
R Y S [A > 1] ➡ SELECT *
                FROM R INNER JOIN S
                 ON A > 1;
```

2.3.3.8 Yunción natural

Para la *yunción natural*, se utiliza la palabra reservada *NATURAL JOIN*:

```
R * S ➡ SELECT *
        FROM R NATURAL JOIN S;
```

2.3.3.9 Cociente

El operador *cociente* no dispone de una traducción directa en SQL. Como se comentará en el apartado 2.4, se ha escogido una traducción que hace uso de funciones avanzadas del SQL como son el *GROUP BY* y el *HAVING*, en lugar de hacer una mera traducción literal del operador cociente en función de operadores básicos (lo cual generaría una expresión mucho menos simple y eficiente):

```
R(r) / S(s) ➡ SELECT r-s
              FROM R NATURAL JOIN S
              GROUP BY r-s
              HAVING COUNT(*) =
                  (SELECT COUNT(*)
                   FROM S);
```

Cabe destacar que la herramienta comprueba previamente que los esquemas de las relaciones -o expresiones- que intervienen en un cociente respetan la restricción vista en el apartado 2.1.2.9.

2.4 Reconocimiento de lenguajes con ANTLR

La herramienta que se utiliza para el reconocimiento de expresiones del lenguaje del Álgebra Relacional, ***ANTLR***, proporciona al programador una interfaz que le permite abstraerse de los conceptos de, entre otros, **análisis léxico, sintáctico y semántico** que tan ligados están a esta

temática. No obstante, en lugar de limitarse únicamente a explicar el modo de uso de esta herramienta para realizar las labores de traducción del Álgebra Relacional, se hará también un breve repaso teórico de los fundamentos anteriormente nombrados dado que puede resultar interesante conocer cómo funciona internamente una herramienta de estas características.

2.4.1 Especificación del lenguaje

Para comenzar a trabajar con ANTLR, en primer lugar debe definirse una gramática que especifique cuál será el lenguaje objeto de reconocimiento -en nuestro caso, el Álgebra Relacional-. En el apéndice A.1 de este documento puede encontrarse la **gramática independiente del contexto** en notación BNF extendida que reconoce este lenguaje.

A partir de esta definición del lenguaje del Álgebra Relacional (contenida en un fichero denominado *RelationalAlgebra.g4*), la herramienta ANTLR genera los siguientes ficheros Java:

- *RelationalAlgebra.tokens*
- *RelationalAlgebraLexer.tokens*
- *RelationalAlgebraLexer.java*
- *RelationalAlgebraParser.java*
- *RelationalAlgebraBaseVisitor.java*
- *RelationalAlgebraVisitor.java*

Es a partir de este momento cuando se está listo para comenzar con la implementación del motor de traducción para las expresiones del Álgebra Relacional. A continuación, se detalla el uso del código generado por ANTLR en los ficheros anteriores, así como el código que ha sido necesario proveer por parte del programador para conseguir las traducciones a SQL.

2.4.1 Análisis léxico

El análisis léxico es la primera fase de un compilador y el componente encargado de llevarla a cabo es el *lexer*. Toma como entrada la propia expresión origen (en este caso, la consulta en Álgebra Relacional que será objeto de traducción a SQL) y la convierte a una secuencia de *tokens*. Por ejemplo, la expresión en Álgebra Relacional `SELECT [A = 1] (R x S)` generaría la siguiente lista de tokens:

1. El operador `SELECT`.
2. El identificador `A`.
3. El operador de igualación =.
4. La constante numérica `1`.
5. El identificador R.

6. El operador x.
7. El identificador S.

Es esta secuencia de tokens la que constituye la entrada que toma la siguiente fase: el *análisis sintáctico*.

ANTLR implementa el lexer para el Álgebra Relacional en la clase contenida en el fichero fuente *RelationalAlgebraLexer.java* y se invoca en el siguiente fragmento de código:

```
19 ANTLRInputStream input;
20 RelationalAlgebraLexer lexer;
21 CommonTokenStream tokens;
22
23 // LEXICAL ANALYSIS
24 input = new ANTLRInputStream(relationalAlgebraInput);
25 lexer = new RelationalAlgebraLexer(input);
26 tokens = new CommonTokenStream(lexer);
```

Figura 2.1: Análisis léxico con ANTLR

2.4.3 Análisis sintáctico

El análisis sintáctico o, en inglés, *parsing* es la fase que tiene lugar tras el análisis léxico y éste toma como entrada la secuencia de tokens que se genera en la fase anterior. A partir de esta secuencia, se obtiene el denominado ***árbol de análisis sintáctico*** o ***árbol de parse*** (en inglés, *Parse Tree*). Este árbol de parse permite representar la estructura sintáctica de las expresiones que constituyen la expresión origen en Álgebra Relacional -para los tokens obtenidos anteriormente, conocer qué posición ocupan dentro del contexto de la expresión original tomada como entrada-.

Por ejemplo, la expresión en Álgebra Relacional `PROJECT [A, B] (R * S);` generaría el siguiente árbol de análisis sintáctico:

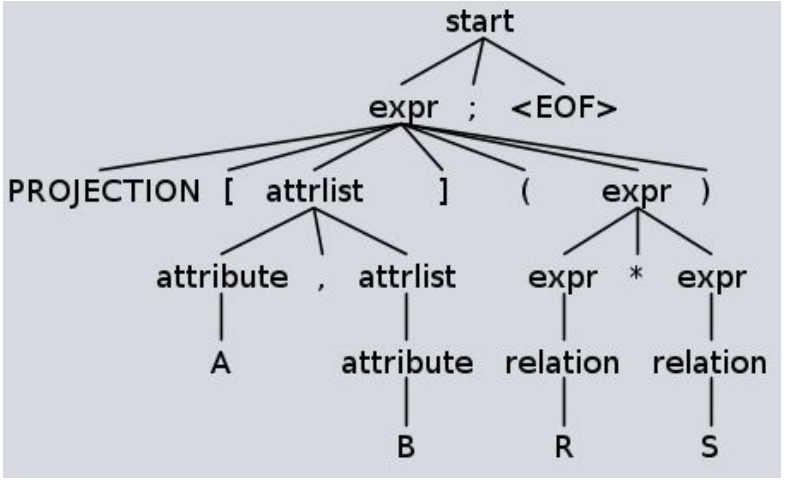


Figura 2.2: Árbol de parse de una consulta en ÁR

En ANTLR, el análisis sintáctico tiene lugar en el siguiente fragmento de código:

```
29 RelationalAlgebraParser parser;
30 ParseTree tree;
31
32 // SYNTAX ANALYSIS
33 parser = new RelationalAlgebraParser(tokens);
34 tree = parser.start();
```

Figura 2.3: Análisis sintáctico con ANTLR

2.4.4 Exploración del árbol de parse y análisis semántico

Para implementar la traducción de consultas, es necesario recorrer el árbol de parse obtenido durante el análisis sintáctico y efectuar ahora el denominado análisis semántico.

Con ANTLR, la exploración del árbol de parse constituye una tarea muy sencilla gracias a la interfaz que la herramienta proporciona. De hecho, el recorrido de la estructura arborescente ya es implementado por ANTLR, de manera que lo único debe hacer el programador es definir una nueva clase -en nuestro caso, en *RelationalAlgebraEvalVisitor.java*- que hereda de la clase *RelationalAlgebraBaseVisitor*. En esta última clase mencionada, la herramienta genera un método para cada posible construcción del lenguaje del Álgebra Relacional (un producto cartesiano, una lista de atributos, una relación, una condición lógica, etc). Así, se debe proveer por parte del programador una sobrescritura de estos métodos para traducir cada posible construcción de las anteriormente citadas. Por ejemplo, en el siguiente método de la clase RelationalAlgebraEvalVisitor se produce la traducción de una yunción natural del ÁR a SQL:

```
412 public String visitNaturalJoin(RelationalAlgebraParser.NaturalJoinContext ctx) {
413     String left = preppendSelectStarIfNeeded(ctx.expr(0), visit(ctx.expr(0)));
414     String right = preppendSelectStarIfNeeded(ctx.expr(1), visit(ctx.expr(1)));
415     return left + " NATURAL JOIN " + right;
416 }
```

Figura 2.4: Traducción de una yunción natural a SQL

En este apartado ya se encuentra prácticamente implementada la traducción de consultas del Álgebra Relacional a SQL, a falta de efectuar un análisis semántico. En un compilador, la fase de análisis semántico incluye la comprobación de ciertos aspectos que resultan imposibles describir mediante una gramática independiente del contexto como son, entre otros, la comprobación de tipos o la declaración de variables previa a su utilización en el código fuente.

Estas comprobaciones de carácter semántico se implementan en la misma clase en la que se realizan las traducciones

(*RelationalAlgebraEvalVisitor*) e incluyen la comprobación de las declaraciones de las vistas referenciadas en las consultas y la compatibilidad de esquemas para ciertas operaciones del Álgebra Relacional -como es el caso del cociente-. El resto de comprobaciones semánticas (existencia de las relaciones que actúan como operandos, compatibilidad en los dominios de los atributos durante comparaciones, etc) se delegan durante la etapa de ejecución de las consultas sobre el sistema gestor de PostgreSQL ya que éste, en caso de detectar algún error, proporciona una retroalimentación lo suficientemente detallada como para no tener que cubrir este tipo de aspectos.

En las siguientes líneas de código tiene lugar la exploración del árbol de parse y el análisis semántico comentados por parte de la interfaz que provee ANTLR:

```
36 RelationalAlgebraEvalVisitor eval;
37
38 // SEMANTIC ANALYSIS
39 eval = new RelationalAlgebraEvalVisitor(database);
40 String sqlTranslation = eval.visit(tree);
```

Figura 2.5: Exploración del árbol de parse y análisis sintáctico con ANTLR

2.5 Optimización de consultas

Optimizar es la acción por la cual se pretende mejorar el rendimiento de algo. Desde esta definición, la optimización de consultas en bases de datos hace referencia a la aplicación de una serie de procesos sobre una consulta dada para transformarla en otra equivalente que resulte más eficiente, bien en tiempo de cómputo o bien en espacio requerido para su procesamiento.

Uno de los aspectos más relevantes que se ha decidido incluir en la herramienta desarrollada es la **optimización básica de consultas**. Así, ésta es capaz de detectar determinados casos en los que, dada una consulta en Álgebra Relacional, se es capaz de obtener una traducción en SQL de su expresión óptima y no de la consulta que originalmente se escribe.

Cabe destacar que esta optimización se realiza siempre desde un punto de vista *teórico*: muchos SGBD de la actualidad aplican procesos de optimización sobre consultas de forma transparente al usuario haciendo uso de estadísticas que, con el paso del tiempo, van recolectando a partir de los datos almacenados. No obstante, a la hora de utilizar esta herramienta no puede hacerse uso de estas estadísticas porque sencillamente no se disponen de los datos necesarios para obtenerlas. En su lugar, se utilizan una serie de propiedades de los operadores del Álgebra Relacional que ayudan a obtener expresiones teóricamente más eficientes.

En el apartado 4.2 se comentará en más detalle la casuística que origina

que el mecanismo optimizador de la herramienta entre en acción (la cual está constituída principalmente por las cascadas de proyecciones y selecciones del Álgebra Relacional).

Capítulo 3. Herramientas utilizadas

El presente capítulo servirá para introducir las herramientas de las que se ha hecho uso durante el desarrollo de la aplicación JITRAX, objeto de este TFG.

3.1 De desarrollo

3.1.1 IDE Eclipse

Eclipse (https://www.eclipse.org/downloads/) es un Entorno de Desarrollo Integrado (IDE, Integrated Development Environment), utilizado principalmente para programar en lenguaje Java. No obstante, se pueden añadir una serie de *plug-ins* que permiten personalizar el entorno, así como añadir soporte para otros lenguajes de programación (C++, Cobol, Haskell, Javascript, Python, etc).

Figura 3.1: Eclipse

Entre las características que reúne este IDE, destacan las que a continuación se enumeran:

- **Editor de texto**: posee resaltado sintáctico de colores y autocompletado inteligente.
- **Compilación en tiempo real**.
- **Asistentes de creación**: permiten crear proyectos, clases, tests, etc de una manera rápida y fácil. También incorporan asistentes para la refactorización de código.
- **Integración con Apache Ant**.
- **Depurador**.

En concreto, se ha utilizado la versión *Oxygen Release Milestone 5 (4.7.0 M5)* durante el desarrollo de este TFG.

3.2 De apoyo

3.2.1 Apache Ant

Apache Ant (http://ant.apache.org/) es una herramienta utilizada en programación para la automatización de tareas, especialmente para aquellas de compilación y construcción (*build*) de proyectos Java.

Es utilizada en este proyecto para construir los ficheros binarios ejecutables de la aplicación a partir del código fuente. En este aspecto, es similar a la herramienta *Make*, utilizada cuando se programa en lenguaje C++.

Figura 3.2: Apache Ant

Para generar el fichero ejecutable con extensión .jar se debe ejecutar lo siguiente desde un terminal, situándose previamente en el directorio raíz del proyecto (véase el apartado 4.3 para conocer cómo acceder al código fuente):

```
$ ant build-jar
```

La ejecución del fichero generado se consigue mediante el siguiente comando:

```
$ ant run
```

3.3 Bibliotecas externas

En concreto, la herramienta desarrollada hace uso de tres componentes software externos, todos ellos con licencias de código libre: *ANTLRv4*, *RSyntaxTextArea* y *JDBC para PostgreSQL*.

3.3.1 ANTLR

ANTLR (http://www.antlr.org/) es una herramienta para el reconocimiento de lenguajes escrita por Terence Parr, profesor de informática en la Universidad de San Francisco. Por medio de esta herramienta ha sido posible llevar a cabo las fases de análisis léxico, sintáctico y semántico de las expresiones del Álgebra Relacional comentadas en el apartado 2.1 a partir de la gramática independiente del contexto que se ha confeccionado para dicho propósito.

Figura 3.3: ANTLR

3.3.2 RSyntaxTextArea

RSyntaxTextArea (http://bobbylight.github.io/RSyntaxTextArea/) es el editor de código que se ha incrustado en el entorno gráfico de la aplicación.

Cuenta con una gran cantidad de funcionalidades que hacen que sea altamente personalizable, de entre las que destacan el resaltado sintáctico a color. Además, junto a esta característica se permite añadir soporte para el lenguaje que el usuario desee y, en esta ocasión, se ha hecho con el Álgebra Relacional.

Incorpora también funciones de edición como las que pueden encontrarse en cualquier editor de código convencional: cortar, copiar y pegar texto, deshacer y rehacer, buscar y reemplazar, etc.

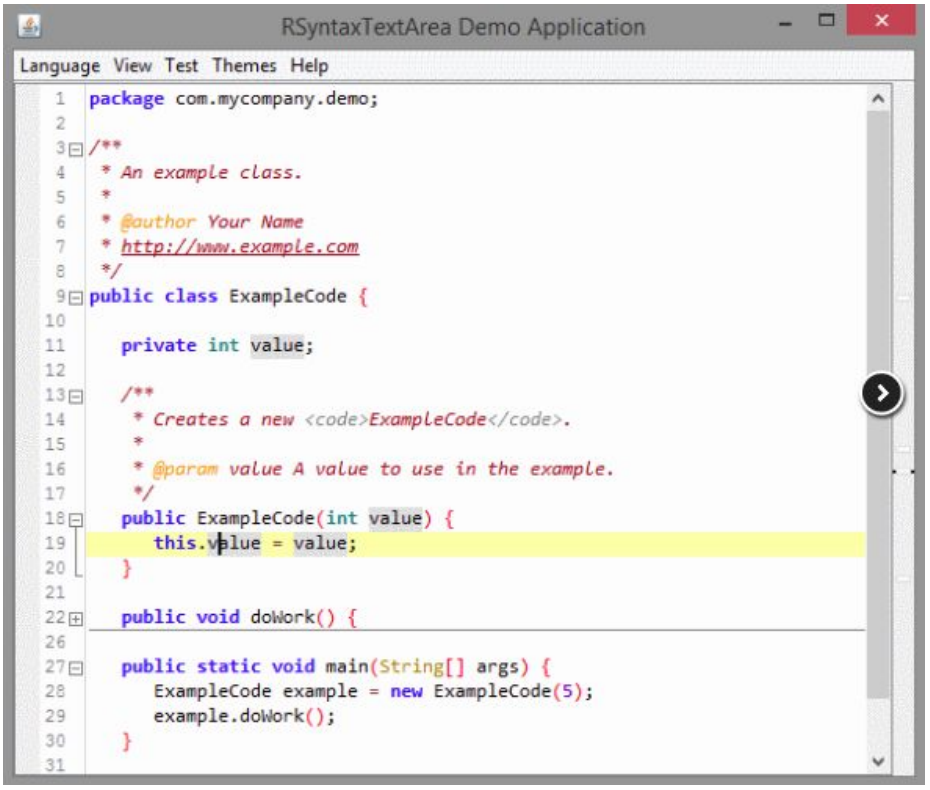


Figura 3.4: RSyntaxTextArea

3.3.3 JDBC para PostgreSQL

JDBC (https://jdbc.postgresql.org/) permite establecer conexión con un SGBD como es ***PostgreSQL*** y ejecutar sobre él todas las consultas que se generan en la aplicación como resultado de la actividad del usuario (ejecución de las consultas en Álgebra Relacional, creación o modificación del esquema de la base de datos con la que se trabaja, etc) de manera completamente transparente para éste.

Figura 3.5: PostgreSQL

Entre las clases y métodos principales proporcionados por esta API se encuentran:

- **Clase *Connection***: permite establecer conexión con el SGBD especificando su dirección IP y puerto, así como el nombre de usuario y contraseña de la cuenta con la que se quiere iniciar sesión.
- **Clase *Statement***: se utiliza para ejecutar consultas sobre la base de datos. Estas consultas pueden retornar un resultado vacío -para las cuales se utiliza el método *executeUpdate()*- o pueden devolver una tabla de resultados -en las que se utiliza en método *executeQuery()*-, la cual se procesa a través de la clase *ResultSet*.
- **Clase *ResultSet***: permite almacenar el resultado de una consulta (tabla) para posteriormente procesarla fila por fila -a modo de *cursor*-.

El siguiente ejemplo muestra cómo se realiza una conexión con un SGDB haciendo uso de JDBC, así como la ejecución sobre éste de una consulta y el posterior procesamiento del resultado devuelto:

```
10 try {
11     String connectionString = "jdbc:postgresql://localhost:5432/MyDatabase";
12     Connection conn = DriverManager.getConnection(connectionString, username, password);
13     Statement stmt = conn.createStatement();
14     ResultSet rs = stmt.executeQuery("SELECT * FROM R");
15 
16     while (rs.next()) {
17         int numColumns = rs.getMetaData().getColumnCount();
18         for (int i = 1; i <= numColumns; i++) {
19             // Show the returned data
20             System.out.println("--> " + rs.getObject(i));
21         }
22     }
23 }
24 
25 catch (SQLException SQL) {
26     e.printStackTrace();
27 }
```

Figura 3.6: Ejemplo de uso de JDBC

3.3.4 Swing

Swing (https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html) es una de las bibliotecas gráficas más conocidas para Java. Permite incluir elementos en la interfaz gráfica como cajas de texto, botones y tablas, entre otros.

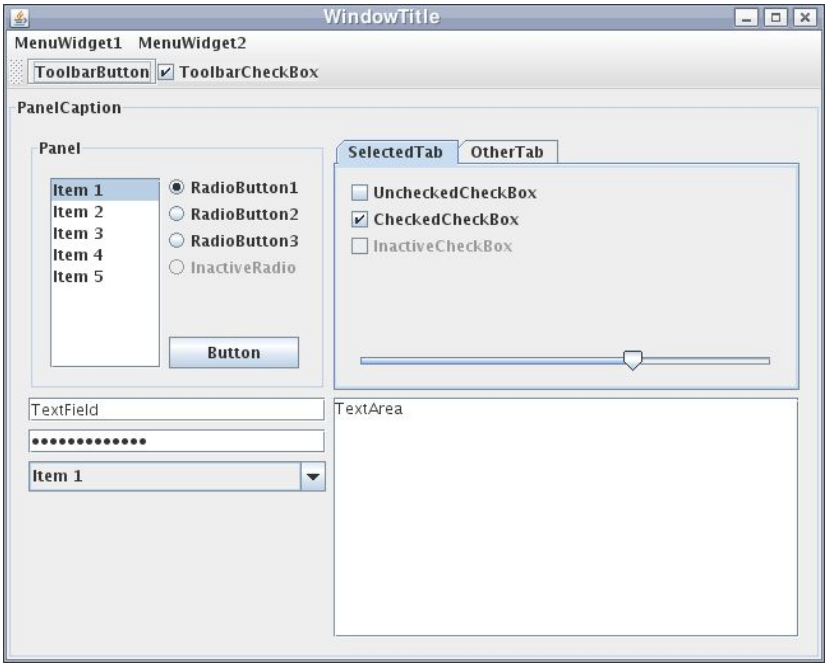


Figura 3.7: Apariencia de los elementos gráficos de Swing

Capítulo 4.
JITRAX

Se destacan en este capítulo los aspectos de mayor importancia relativos a la aplicación que se ha desarrollado como parte de este TFG.

Se trata de una aplicación de escritorio multiplataforma que, como acaba de verse en un apartado anterior, ha sido desarrollada con bibliotecas externas que son distribuidas con licencias de software libre.

4.1 Interfaz gráfica

Como se comentó en el capítulo introductorio, se ha diseñado la interfaz gráfica haciendo especial hincapié en la simplicidad y la usabilidad. Para llevarla a cabo, se ha utilizado la biblioteca *Swing* de Java. Además, se ha decidido que esta GUI esté íntegramente en inglés, con la posibilidad de añadir soporte para más idiomas como trabajo futuro.

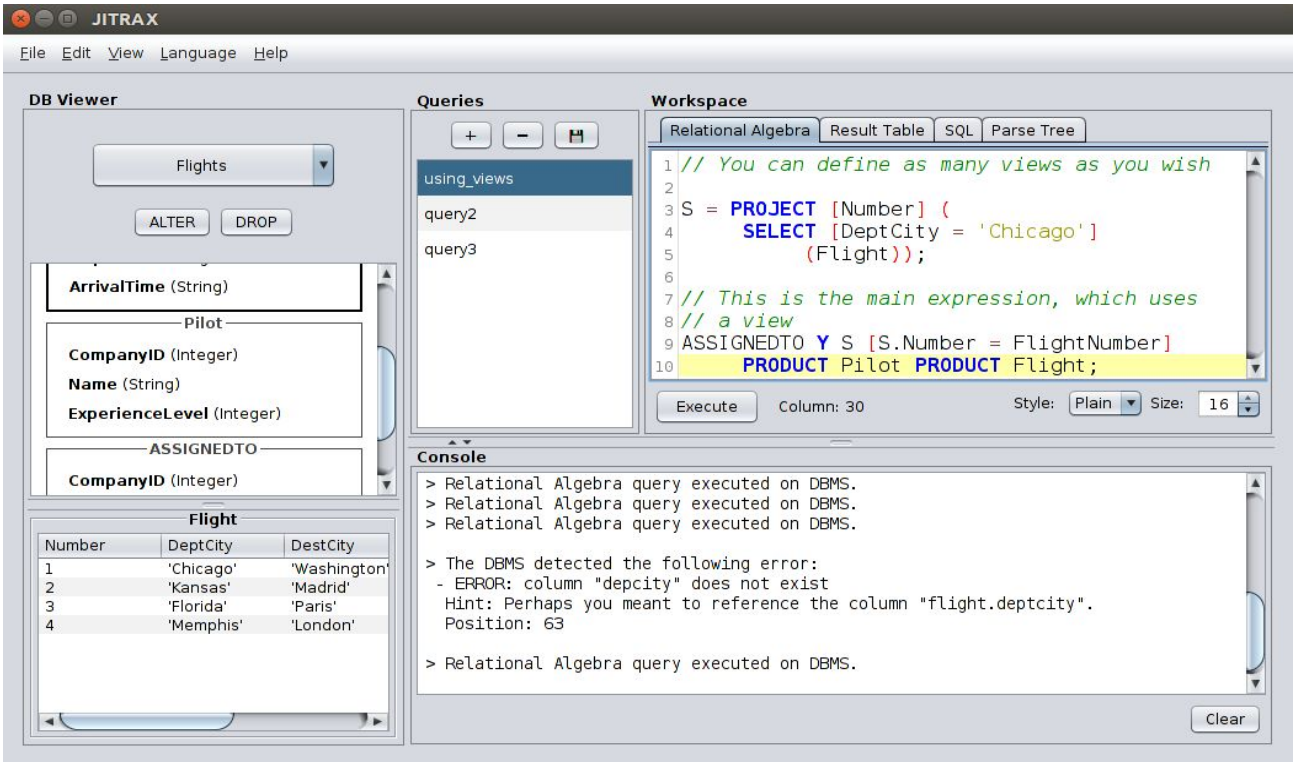


Figura 4.1: Vista principal de la aplicación

En la figura 4.1 se muestra la vista principal de *JITRAX*, en la que destacan cuatro partes diferenciadas: el visor de la base de datos, la consola

de retroalimentación, la lista de consultas y el espacio de trabajo.

4.1.1 Visor de la base de datos

El visor de la base de datos (*DB Viewer*) muestra información acerca de la base de datos definida y replicada en el SGBD sobre la que el usuario ejecutará sus consultas en Álgebra Relacional. Así, se muestra el listado de relaciones que componen la base de datos y una vista más pequeña situada en la parte inferior de esta sección para visualizar el contenido de cada una de ellas (ver de qué tuplas se componen haciendo click sobre la relación que se desee). Además, es posible ver de forma inmediata el esquema de atributos de cada tabla, así como el dominio de cada uno de ellos.

4.1.2 Consola

La consola (*Console*) se utiliza como medio para retroalimentar al usuario acerca de la actividad que éste genera mientras tiene lugar la ejecución de la aplicación. Entre otros, cuando el usuario escribe consultas en Álgebra Relacional sintácticamente incorrectas o, cuando se producen errores de carácter semántico durante la ejecución de estas sobre la base de datos, el usuario es notificado de manera conveniente a través de esta vista de consola.

4.1.3 Lista de consultas

La lista de consultas (*Queries*) permite el manejo simultáneo de consultas, de manera que el usuario no pierda aquellas que ha escrito anteriormente si desea escribir una nueva.

4.1.4 Espacio de trabajo

El espacio de trabajo (*Workspace*) está conformado por el editor de consultas en Álgebra Relacional, el visor de la tabla de resultados (2), el editor de código SQL (3) y el visor del árbol de análisis sintáctico (*Parse Tree*).

- **Editor de consultas en Álgebra Relacional:** permite al usuario insertar sus consultas en este lenguaje teórico para su posterior ejecución sobre la base de datos seleccionada. Incorpora resaltador sintáctico para las palabras clave del Álgebra Relacional (añadido gracias a una funcionalidad de la herramienta *RSyntaxTextArea* comentada en el apartado 3.3.2) y reconocimiento de comentarios al estilo Java (multilínea, mediante la estructura */*...*/*; y de una línea, a través de la secuencia de caracteres ‘//’).
- **Tabla de resultados (*Result Table*)**: muestra al usuario la tabla que se

genera como resultado de la ejecución de su consulta en Álgebra Relacional sobre la base de datos seleccionada. Puede ser vacía.

- **Vista de la traducción a SQL (*SQL*)**: se da al usuario la posibilidad de estudiar la traducción a SQL que genera JITRAX. Además, se incluye la posibilidad de mostrarlo con formato. Cabe destacar también que no existe la posibilidad de modificar este código.
- **Árbol de análisis sintáctico (*Parse Tree*)**: contiene una imagen gráfica -que posteriormente se puede exportar- del árbol de parse que genera la herramienta utilizada para el reconocimiento del lenguajes ya comentada (ANTLR). Si bien es una vista incluida para facilitar las labores de desarrollo, puede resultar relevante también para el usuario con interés en el proceso de interpretación.

4.1.5 Otros aspectos de la interfaz

La vista principal de la aplicación mostrada en la figura 4.1 no es la primera que entra en escena. Inicialmente, el entorno gráfico se muestra vacío, siendo necesario establecer una conexión con un SGBD de PostgreSQL para comenzar a trabajar. Este establecimiento de la conexión puede apreciarse en la siguiente figura, donde son necesarios la dirección IP y el puerto de escucha del SGBD, así como el nombre de usuario y la contraseña de la cuenta con los que iniciar sesión:

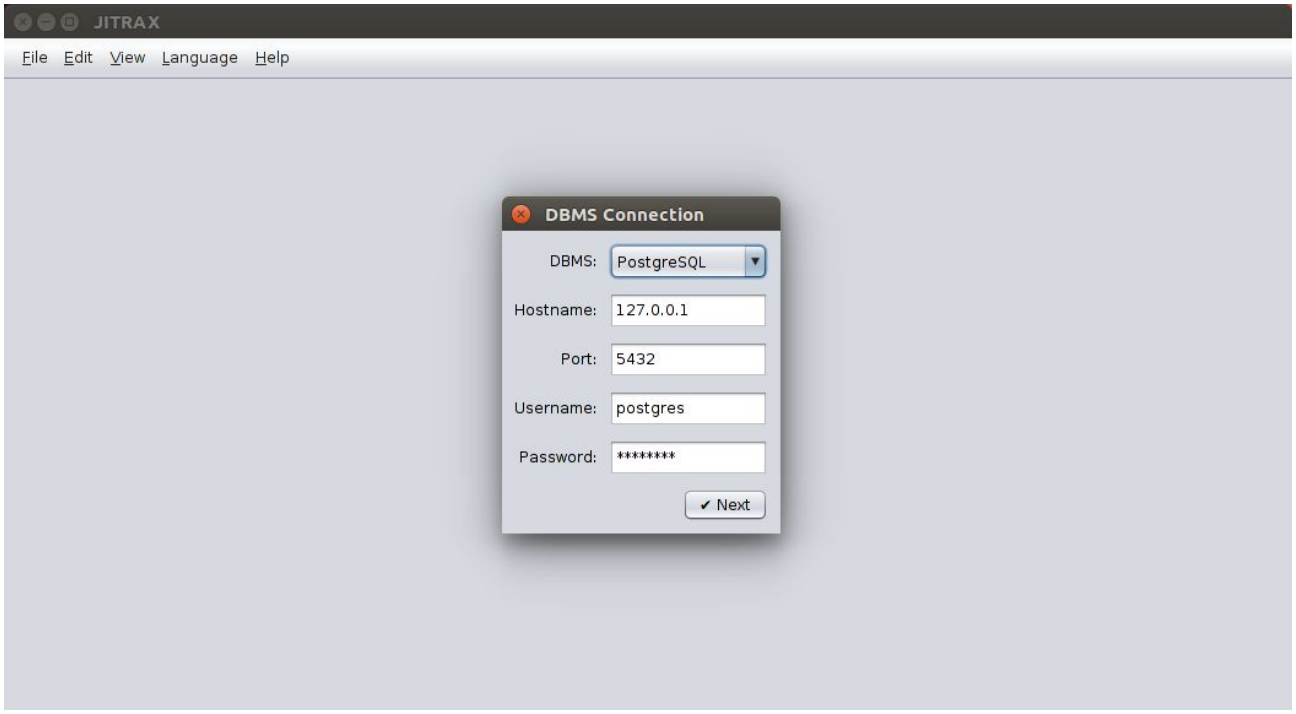


Figura 4.2: Establecimiento de la conexión con PostgreSQL

Otro punto que cabe reseñar acerca de la interfaz desarrollada es la posibilidad de modificar y actualizar el esquema de la base de datos desde el

propio entorno gráfico. Uno de estos casos se ilustra en la figura 4.3, en la que se muestra la vista desde la cual es posible modificar el esquema y los contenidos de una determinada tabla.

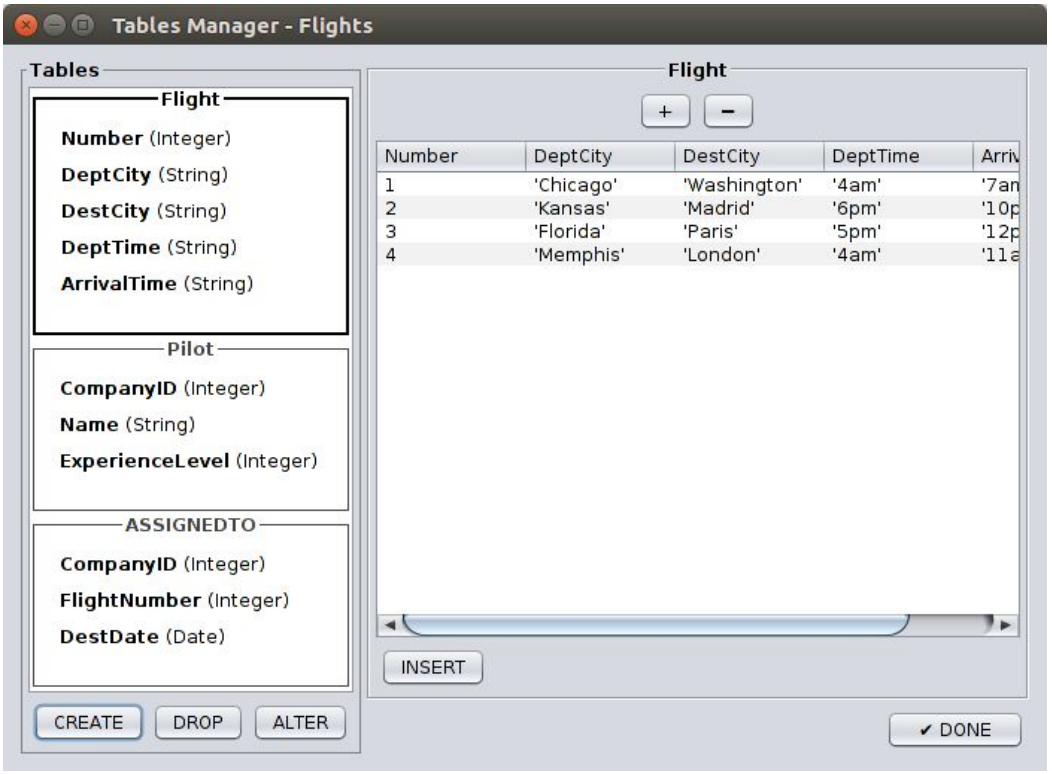


Figura 4.3: Vista de modificación de tablas

4.2 Características

En este TFG no se ha desarrollado una herramienta que se limite únicamente a realizar las funciones que propiciaron su propuesta -intérprete de consultas del Álgebra Relacional capaz de ejecutar sentencias de dicho lenguaje sobre bases de datos reales-. Además de esto, se han incorporado una serie de características que permiten dar solución a algunos de los problemas encontrados durante el desarrollo, o bien simplemente se han añadido porque resultan interesantes y útiles para el usuario.

4.2.1 DSL de especificación de BBDD

Un lenguaje de dominio específico (*DSL, Domain-Specific Language*) es un lenguaje de programación o especificación dedicado a resolver un problema en particular, representar un problema específico y proveer una técnica para solucionar una situación particular [2].

En el contexto de este TFG, se ha hecho uso de la misma herramienta utilizada para el reconocimiento de expresiones del Álgebra Relacional (ANTLR) para diseñar un DSL que permita la definición y posterior carga de

una base de datos en la aplicación a través de un fichero script. La gramática independiente del contexto confeccionada para el reconocimiento de este lenguaje puede encontrarse en el apéndice A.2.

Se trata de un lenguaje de dominio específico basado en la notación original del trabajo de Edgar F. Codd en el que propone el ya comentado *Modelo Relacional* de datos (véase el apartado 2.1). Así, mediante el uso de un fichero que contenga la definición de una base de datos en este lenguaje, el usuario será capaz de cargar en JITRAX dicha base de datos sobre la que poder ejecutar sus consultas -y, de forma transparente para él, la herramienta la crearía sobre el SGBD de PostgreSQL con el que se está conectado-. La utilización de este DSL presenta dos ventajas:

- **Sintaxis sencilla**: la sencillez de la notación que utiliza este DSL hace que sea fácil de aprender por el usuario, de manera que constituya una forma cómoda y rápida de cargar en JITRAX bases de datos sobre las que operar.
- **Independiente del SGBD**: muchos SGBD de la actualidad ya proveen una notación que permite la definición de una base de datos desde un fichero *script*. No obstante, se trata de una notación no estandarizada que puede variar de un SGBD a otro. Como se verá en el capítulo de líneas de trabajo futuras, se pretende incrementar la lista de SGBD soportados por la aplicación, y este DSL proporcionaría una forma de definir base de datos mediante ficheros que sería independiente del dialecto SQL utilizado por los sistemas gestores.

4.2.2 Sistema de importación/exportación de ficheros

Se ha considerado también de utilidad incorporar un sistema de importación/exportación de ficheros que permita no sólo la carga de información a la aplicación -como los *scripts* de definición de bases de datos anteriormente comentados-, sino que ofrezca también la posibilidad de guardar trabajo generado por parte del usuario.

Por un lado, la información que puede ser importada hacia la aplicación incluye la siguiente:

- Definición de base de datos mediante el citado DSL.
- Consultas en Álgebra Relacional.

Por otro lado, el usuario será capaz de exportar a ficheros elementos como los siguientes:

- Definiciones de las bases de datos con las que se ha estado

trabajando, mediante el DSL de especificación comentado.

- Consultas realizadas en Álgebra Relacional.
- Traducciones generadas por la herramienta en lenguaje SQL.
- Tablas de resultado en formato *csv*.
- Imágenes de los árboles de análisis sintáctico en formato *png*.

4.2.3 Mecanismo de sincronización con el SGBD

Cuando en la aplicación se muestra al usuario la definición de la base de datos seleccionada para trabajar (listado de tablas, filas de datos, esquemas, etc) tiene que ocurrir que esa información debe encontrarse replicada -sin que se dé ningún tipo de discrepancia- en la base de datos que está alojada en el SGBD con el que la aplicación está conectada. En otras palabras, la base de datos que se muestra de forma gráfica al usuario en la aplicación debe ser la misma que la que le corresponde en el SGBD.

Para asegurar en la medida de lo posible que esa hipotética discrepancia de contenidos no tiene lugar, se ha implementado un "mecanismo de sincronización". La aplicación de dicho mecanismo se resume en el siguiente diagrama de actividades de UML:

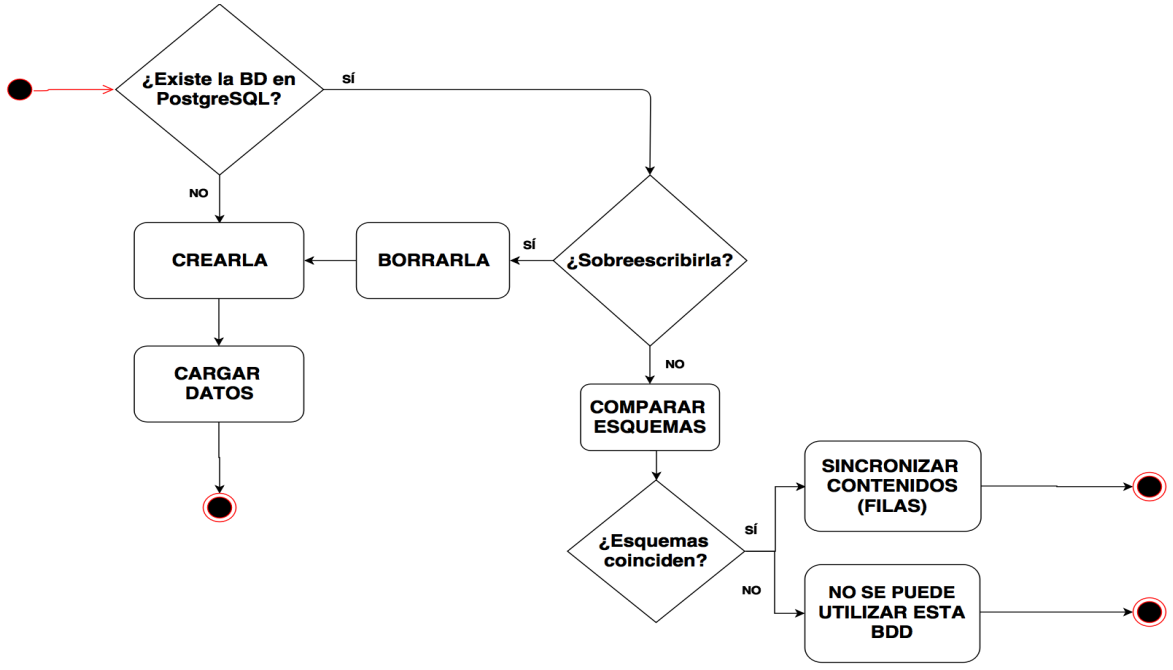


Figura 4.4: Diagrama de actividades UML para el mecanismo de sincronización

Primeramente, la aplicación pregunta al SGBD si ya existe una base de datos con el mismo nombre que la que el usuario ha definido para comenzar a trabajar. Si no existe, simplemente se crea (mediante sentencias

CREATE TABLE e *INSERT INTO* que se ejecutan sin que el usuario se percate de ello).

En el caso de que dicha base de datos ya exista sobre el SGBD, se pregunta al usuario si desea sobreescribirla, lo cual supone eliminar todo el contenido de la base de datos en el SGBD (mediante sentencias *DROP*) y volver a crearla con la nueva definición que ha llegado a la aplicación.

Por último, si el usuario no desea sobrescribir la base de datos en el SGBD, se pasa a comparar los esquemas de ambas bases de datos -esto es, comprobar que todas las tablas de la base de datos de la aplicación existen en el SGDB, que las tablas correspondientes tienen los mismos esquemas de atributos y que, además, para los atributos correspondientes se tiene el mismo dominio (tipo de dato)-. En caso de que los esquemas discrepen, no se puede utilizar la nueva definición de la base de datos -ya que se han propuesto al usuario alternativas para utilizarla sin tener que perder información existente en la base de datos del SGBD-. Por contra, si los esquemas coinciden, se pasa a sincronizar los contenidos -las filas de ambas bases de datos se fusionan-.

4.2.4 Traducción avanzada de consultas

Uno de los aspectos de más relevancia incluidos en la herramienta desarrollada y sobre el que enfocar esfuerzos de cara a continuar ofreciéndole soporte es la *traducción avanzada de consultas*. Ésta consiste en, teniendo en cuenta el concepto de *optimización de consultas* visto en el apartado 2.4, realizar las traducciones a SQL de aquella expresión óptima de la original en Álgebra Relacional.

En particular, la traducción avanzada de consultas entra en escena cuando se introducen consultas en Álgebra Relacional que contienen **cascadas de proyecciones o selecciones**.

4.2.4.1 Cascada de proyecciones

Un ejemplo de **cascada de proyecciones** es la que a continuación se muestra:

```
PROJECT [A] (PROJECT [A, B] (PROJECT [A, B, C] (R)));
```

A efectos prácticos, la consulta anterior genera el mismo resultado que realizar una única proyección sobre el atributo A de la relación R:

```
PROJECT [A] (R);
```

No obstante, esta última expresión resulta mucho más eficiente que la

primera -puesto que se ha reducido el número de proyecciones que computar-.

Para poder aplicar la simplificación anterior, debe comprobarse que la lista de atributos que se especifica en una proyección contiene a los atributos que se definen en la lista de la proyección externa. Por ejemplo, en la cascada de proyecciones

```
PROJECT [A] (PROJECT [A, B] (R));
```

vemos cómo en la lista de atributos de la proyección interna ([A, B]) está contenido el atributo de la lista de la proyección externa ([A]).

Así, la traducción SQL que se propone para la cascada de tres proyecciones anteriormente descrita se corresponde con realizar una única operación SELECT del SQL, en lugar de generar una traducción con tres subconsultas anidadas (lo que se obtendría si no se aplicase ningún proceso de simplificación):

```
SELECT A
FROM R;
```

4.2.4.2 Cascada de selecciones

De manera análoga al caso anterior, se produce el de las cascadas de selecciones, como la que se observa en este ejemplo:

```
SELECT [A = 'a'] (SELECT [B = 'b'] (SELECT [C = 'c'] (R)));
```

La expresión anterior tiene una equivalente más eficiente en la que se computa una única selección del Álgebra Relacional -cuya condición lógica asociada consiste en la conjunción de las tres condiciones presentes en la cascada-. Así, se obtiene la siguiente expresión más eficiente:

```
SELECT [A = 'a' and B = 'b' and C = 'c'] (R);
```

A partir de ésta, se genera la traducción a SQL -como en el caso de la traducción de una cascada de proyecciones, se evita también la escritura de subconsultas anidadas-:

```
SELECT *
FROM R
WHERE A = 'a' and B = 'b' and C = 'c';
```

4.2.4.2 Operador cociente

La traducción propuesta del operador cociente -ya comentada en otro

capítulo- merece también especial mención en este apartado. Se trata de uno de los operadores más complejos del Álgebra Relacional, no sólo en cuanto a implementación, sino también de cara a ser asimilado por parte de los estudiantes.

Inicialmente, se barajó la posibilidad de realizar la traducción de este operador a partir de forma en función de operadores básicos del Álgebra Relacional:

```
R / S = PROJECT [r-s] (R) -
  - PROJECT [r-s] (PROJECT [r-s] (R) x S - R);
```

Sin embargo, la expresión anterior resulta muy ineficiente y compleja -debido principalmente a la presencia de diferencias y productos cartesianos entre relaciones-. Es por ello que la idea de traducir este operador en función de la expresión anterior fue deshecha en detrimento de utilizar la traducción ya vista en el apartado 2.1.4.9, la cual consiste en hacer uso de funciones avanzadas que incorpora el SQL, como lo son las sentencias *GROUP BY* y *HAVING*.

4.3 Acceso al código fuente y compilación

El repositorio del código fuente del proyecto en *GitHub* puede ser accedido a través de este enlace. En él, puede encontrarse una opción para la descarga del proyecto y un fichero *README.md* con información complementaria acerca del uso y ejecución de la aplicación.

4.4 Contribución al software libre

La herramienta desarrollada como parte de este Trabajo Fin de Grado ha sido creada haciendo uso de las herramientas software vistas en el capítulo 3, las cuales son completamente libres.

Además, se ha presentado recientemente este trabajo al Concurso Universitario de Software Libre (CUSL) que organiza cada año la Universidad de Sevilla a nivel estatal. En este evento, celebrado los días 11 y 12 de mayo de 2017, se le ha otorgado el ***Premio al Mejor Proyecto Educativo***, el cual confiere a este trabajo un reconocimiento destacable como herramienta que

favorece el aprendizaje de Álgebra Relacional.

Figura 4.5: Logo de la XI edición del CUSL

Puede consultarse información adicional sobre el reconocimiento otorgado a través de este enlace.

Asimismo, el día 20 de abril de 2017 se le concedió el 2º premio al mejor proyecto en una edición del concurso celebrada a nivel local por parte de la Universidad de La Laguna.

Capítulo 5.
Conclusiones y líneas futuras

5.1 Conclusiones

La aplicación desarrollada como parte de este TFG es una herramienta de carácter académico y metodológico, y que presenta unos rasgos que la convierten en un potente medio para alcanzar el fin al cual está destinado: facilitar la comprensión del lenguaje abstracto de consultas del Álgebra Relacional. Además, se han visto implicados en su desarrollo campos de diferente índole: bases de datos relacionales, construcción de interfaces gráficas, reconocimiento de lenguajes, etc.

Sin embargo, se debe hacer especial mención a que a este enfoque didáctico se le suma uno más práctico: incluso los propios profesionales de las BBDD también podrían hacer uso de la herramienta de manera que puedan aprovechar este tratamiento más práctico del Álgebra Relacional para utilizarlo como si fuese un lenguaje comercial de consultas más del mercado. Así, las consultas que se escriban en Álgebra Relacional serían ejecutadas sobre una base de datos que se encuentra alojada sobre un Sistema Gestor real. La gran ventaja que esto proporcionaría sería la de utilizar un lenguaje de consultas que provee un mayor nivel de abstracción que el que está presente en los lenguajes de naturaleza más comercial, como el SQL.

Nos encontramos ahora a la espera de ver cuál es la acogida que la herramienta recibe el próximo curso por parte de los estudiantes que se inician en el estudio del Álgebra Relacional para conocer hasta qué punto ésta ayuda a que el proceso de aprendizaje del mencionado lenguaje se desarrolle de forma más eficaz.

5.2 Líneas futuras

A pesar de que la herramienta desarrollada cuenta ya con un alto grado de funcionalidad y, en su mayoría, las expectativas previas al desarrollo están cubiertas, existen una serie de líneas a las que poder dedicarnos para continuar ofreciendo soporte a esta aplicación.

En primer lugar, se desea acometer el desarrollo de una versión en formato web que contribuya a favorecer el acceso a la herramienta.

En segundo lugar, existe la posibilidad de incorporar un motor de traducción para el Cálculo Relacional, otro de los grandes lenguajes teóricos de consulta para bases de datos relacionales. En particular, existen el Cálculo Relacional de Tuplas y el de Dominios, pero se abordaría el soporte para el primero dado que el SQL incorpora elementos que toma directamente a partir de este, lo cual facilitaría las labores de traducción.

En tercer lugar, se pretende mejorar el apartado de optimización de consultas, dado que el manejo de las cascadas de proyecciones y selecciones que se han incluido hasta el momento constituyen una mínima parte de lo que realmente se puede lograr en lo relativo a este punto. Esto ayudaría también a que el estudiante fuese capaz de mejorar su capacidad para la escritura eficiente de consultas a través del estudio del código SQL que se genera a partir de las expresiones que resultan de aplicar este proceso de optimización.

Por último, se abordarían otra serie de tareas orientadas a la mejora de la aplicación de escritorio: incrementar la lista de operadores soportados del Álgebra Relacional y la de SGBD con los que poder trabajar (MariaDB, MySQL, etc), entre otros.

Capítulo 6. Conclusions and future work

6.1 Conclusions

To sum up, the application which has been developed as a result of this work is a tool with an academical and methodological character and presents features that makes it a powerful vehicle to achieve its intended purpose: to facilitate the comprehension of the Relational Algebra as an abstract query language. Furthermore, fields of different nature have been involved in its development: relational databases, designing of graphical user interfaces, language recognition, etc.

However, particular reference should be made to a more practical approach: even databases professionals would be able to use this tool in a manner that the Relational Algebra could be used as a commercial query language. Thus, the queries written in Relational Algebra would be executed on a database which is stored on a real Database Management System. This would afford the big advantage of using a query language with a level of abstraction which is higher than the one found in languages with a more commercial nature, such as SQL.

Now, we look forward to seeing what is the acceptance this tool will receive from those students who are beginning in the study of the Relational Algebra query language the next academic year. This would allow us to know how much the learning process of this language becomes more effective.

6.2 Future work

Despite the developed tool has a high degree of functionality and meets, for the most part, the expectations prior to the development, there are several work lines which can be addressed in order to continue to support this application.

Firstly, it is desired to accomplish the development of a web version of this tool, which would facilitate the access to it.

Secondly, we have the chance to incorporate a translation engine for the Relational Calculus, which is also another big theoretical query language for relational databases. In particular, there are two types of Relational Calculus: Tuple (1) and Domain (2) Relational Calculus. However, the

support for the first one would be performed as the SQL language includes elements which are directly taken from this one, and that would facilitate the translation tasks.

Thirdly, it is our aim to enhance the query optimization issue, since the included handling of projections and selections' cascades are just a small part of what we can really achieve on this point. This would also help students to improve their query writing capacity by studying the SQL code generated from the expressions which have been obtained through the use of this optimization process.

Lastly, another kind of tasks would be addressed, such as the enhancement of the desktop application by extending the list of the supported Relational Algebra operators and that of the DBMS which the user can work with.

Capítulo 7. Presupuesto

7.1 Presupuesto

Recursos	Presupuesto
Nº de horas de trabajo: 250 (40€/hora)	10.000€
Equipo de desarrollo	500€
TOTAL	10.500€

Tabla 7.1. Tabla de presupuestos.

Apéndice A. Gramáticas independientes del contexto

Las gramáticas independientes del contexto (CFG, Context-Free Grammar) que se incluyen en este apéndice siguen la notación BNF extendida (Backus-Naur Form).

A.1. Álgebra Relacional

```
<start> :== (<view> ‘;’)* <expr> ‘;’

<view> :== IDENTIFIER ‘=’

<expr> :== <relation>
          | ‘(‘ <expr> ‘)’
          | PROJECTION ‘[‘ <attrlist>’]’ ‘(‘ expr ‘)’
          | SELECTION ‘[‘ <condlist> ‘]’ ‘(‘ <expr> ‘)
          | <expr> UNION <expr>
          | <expr> CARTESIAN_PRODUCT <expr>
          | <expr> DIFFERENCE <expr>
          | <expr> INTERSECTION <expr>
          | <expr> JOIN <expr>
          | <expr> NATURAL_JOIN <expr>
          | <expr> DIVISION <expr>

<attrlist> :== <attribute> | <attribute> ‘,’ <attrlist>

<condlist> :== <condlist> BOOLEAN_OR <condlist>
               | <condlist> BOOLEAN_AND <condlist>
               | BOOLEAN_NOT <condlist>
               | ‘(‘ <condlist> ‘)’
               | <compared> <comparator> <compared>
```

```
<comparator> :== ‘=’
                 | ‘!=’
                 | ‘<>’
                 | ‘>’
                 | ‘>=’
                 | ‘<’
                 | ‘<=’

<compared> :== <attribute>
               | STRING
               | NUMBER

<relation> :== IDENTIFIER

<attribute> :== IDENTIFIER
```

A.2. DSL para la especificación de BBDD

```
  <start> :== <database>

  <database> :== ‘DATABASE’ ID ‘;’ <table>+

  <table> :== ‘TABLE’ ID ‘(‘ <attrlist> ‘)’
                  (‘=>’ (‘(‘ datalist ‘)’ ‘;’)+)?

  <attrlist> :== <attribute> | <attribute> ‘,’ <attrlist>

  <datalist> :== <datum> | <datum> ‘,’ <datum>

  <attribute> :== ID ‘:’ <datatype>

  <datum> :== STRING | NUMBER

  <datatype> :== ‘string’| ‘char’ | ‘int’ | ‘float’ | ‘date’
```

Apéndice B. Traducciones

Se presentan en este apéndice algunas traducciones de consultas en Álgebra Relacional que involucran a varios operadores -a la izquierda- a SQL -a la derecha- que la herramienta ha sido capaz de generar. Las consultas que aquí se muestran se han realizado sobre el siguiente esquema de relaciones:

- R1 (A: Char, B: Char, C: Char)
- R2 (A: Char, B: Char, C: Char, D: Char)
- S1 (D: Char, E: Char, F: Char)
- S2 (C: Char, D: Char)
- R3 (A: Integer, B: Integer, C: Integer)
- S3 (D: Integer, E: Integer)

B.1. Producto cartesiano de una relación y un cociente

```
(R2 / S2) PRODUCT S1;      ⇨      SELECT * FROM (
                                        SELECT A, B
                                        FROM R2 NATURAL JOIN S2
                                        GROUP BY A, B
                                        HAVING COUNT (*) = (SELECT COUNT(*)
                                                            FROM S2)
                                  ) AS q1, S1;
```

B.2. Producto cartesiano de una proyección y una yunción natural

```
PROJECT [A, B]                          SELECT *
      (SELECT [A = 'a'] (R1))     ⇨     FROM (SELECT A, B
            PRODUCT                           FROM R1
      (R3 * S3);                              WHERE (A = 'a')
                                        ) AS q1, (
                                              (SELECT *
                                              FROM R3 NATURAL JOIN S3)
                                              ) AS q2;
```

Bibliografía

[1] Codd, E. (1983). A relational model of data for large shared data banks. *Communications of the ACM*, 26(1), pp.66-69.

[2] Wikipedia contributors. “Domain-specific language”. Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 23 May 2017. Web. 26 May 2017.

[3] Parr, Terence. *The definitive ANTLR 4 reference*. Dallas, Texas: The Pragmatic Bookshelf, 2013.

[4] Cornelio, Enrique; Martínez, Luis y Alonso, Israel. *Bases de datos relacionales: fundamentos y diseño lógico*. Madrid: Universidad Pontificia Comillas, 2005.

[5] Millán, José A. *Compiladores y procesadores de lenguajes*. Cádiz: Universidad de Cádiz, Servicio de Publicaciones, 2004.

[6] Cornelio, Enrique. *Bases de datos relacionales*. Madrid: Paraninfo, 1992.

[7] Piattini, Mario G. *Tecnología y diseño de bases de datos*. Paracuellos de Jarama, Madrid: Ra-Ma, 2006.

[8] Molková, L. (2009). *Relational Algebra Expression Evaluation*. Licenciatura. Masaryk University.

[9] Wikipedia contributors. “Semantic analysis (compilers)”. *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 24 Sep. 2016. Web. 26 May 2017.

[10] Wikipedia contributors. “Swing (Java)”. Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 28 Feb 2017. Web. 3 Jun 2017.

[11] Wikipedia contributors. “Java Database Connectivity”. Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 25 May 2017. Web. 3 Jun 2017.

[12] Chavarría-Báez, Lorena, José Antonio Hijar-Miranda and Dario Emmanuel Vázquez. *“Estudio comparativo sobre herramientas de software para la enseñanza del diseño conceptual de bases de datos”*.