Universidad del Valle de Guatemala. Semestre 1, 2018. Departamento en Ingeniería en Ciencias de la Computación y Tecnologías de la Información.

Construcción de un DBMS

MENDOZA, E.; CUSTODIO, J.; BROLO, G.
Carnets: 15002, 15220, 15105

Documentación presentada para el curso de BASES DE DATOS, S. 10, Universidad del Valle de Guatemala.

Abstract

Este proyecto abarca la implementación de un DBMS para un subconjunto de la gramática de PostgreSQL, incluyendo la implementación de los sistemas de DDL y de DML. Se utilizó Java como lenguaje principal para la elaboración de este proyecto, Vaadin como herramienta para la realización del IDE del DBMS y ANTLR para la generación del Lexer y Parser de la gramática elaborada. Para el desarrollo del DML se construyó un árbol B+ para alojar los datos ingresados por el usuario y para poder manejar las instrucciones de INSERT, SELECT, UPDATE y DELETE. Se hizo uso de archivos JSON para la generación de archivos de metadatos para las bases de datos generadas y sus respectivas tablas.

Objetivos

Construir un manejador de bases de datos para un subconjunto de la gramática de PostgreSQL, que implemente las instrucciones de un DDL y un DML, proporcionando además un IDE para la manipulación y presentación de los datos.

Metodología

Se empleó ANTLR 4 para la generación del Lexer y Parser de la gramática elaborada. Se puede consultar el archivo *Sql.g4* para analizar la gramática. Con ello, se estructuró dentro del proyecto la carpeta *src/main/java/com/cusbromen/antlr*, que contiene los archivos de Lexer y Parser, así como un *listener* para los errores de lexeo encontrados. Se generó también una interfaz de Visitante para poder ir parseando el input del usuario conforme se encontrara una cabeza de la producción de la gramática. En la carpeta *src/main/java/com/cusbromen//semanticControl* se encuentra la implementación del Visitante (*src/main/java/com/cusbromen//semanticControl/Visitor.java*).

Dentro de esta ultima carpeta, se encuentra también la clase *SymbolTableHashMap*, la cual contiene los métodos que el visitante utiliza para manipular la información dependiendo de la producción que se esté visitando en un determinado momento.

Para las instrucciones implementadas se generan archivos de metadatos en formato JSON. Se genera un archivo *dbs.json* que contiene la información general de todas las bases de datos en el DBMS. La estructura de este archivo es la siguiente:

```
{"nombre_db":{"noTables":n},"nombre_db_2":{"noTables":n}, ... }
```

En donde se guarda el objeto para la base de datos con su nombre y una propiedad *noTables*, que corresponde al número *n* de tablas de la base de datos. Luego, para cada base de datos se crea un directorio con el nombre de la base y un archivo JSON con el nombre de la base también. La estructura de este archivo es la siguiente:

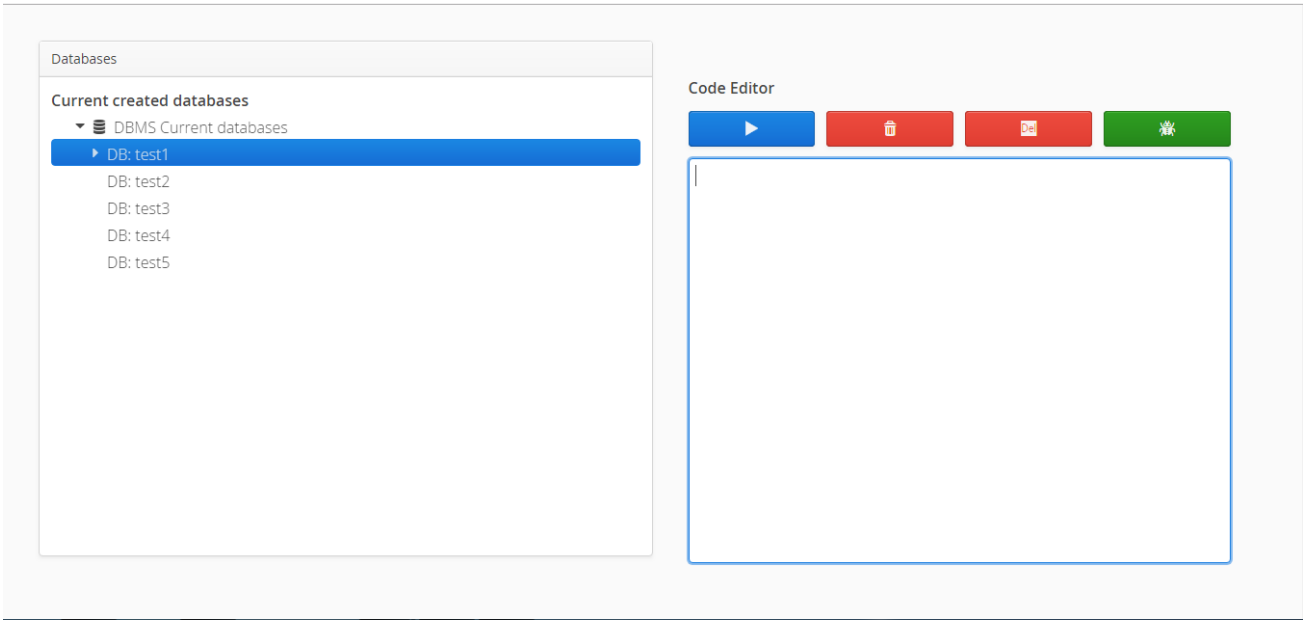
```
{"nombre_tabla":{"noRecords":0,"columns":{"nombre_col":{"type":"data_type"}},
                                "constraints":{}},...}, ...
```

En donde por cada tabla en la base de datos se guarda un objeto con el nombre de la tabla *nombre_tabla*, con propiedades *noRecords*, que corresponde al número de registros en la tabla; *columns*, que corresponde a las columnas dentro de la tabla, en dónde cada columna tiene propiedades *nombre_col*, el nombre de la columna, y *constraints*, las restricciones para la columna. Para la propiedad *nombre_col* se tiene la propiedad *type* para el tipo de dato de la columna.

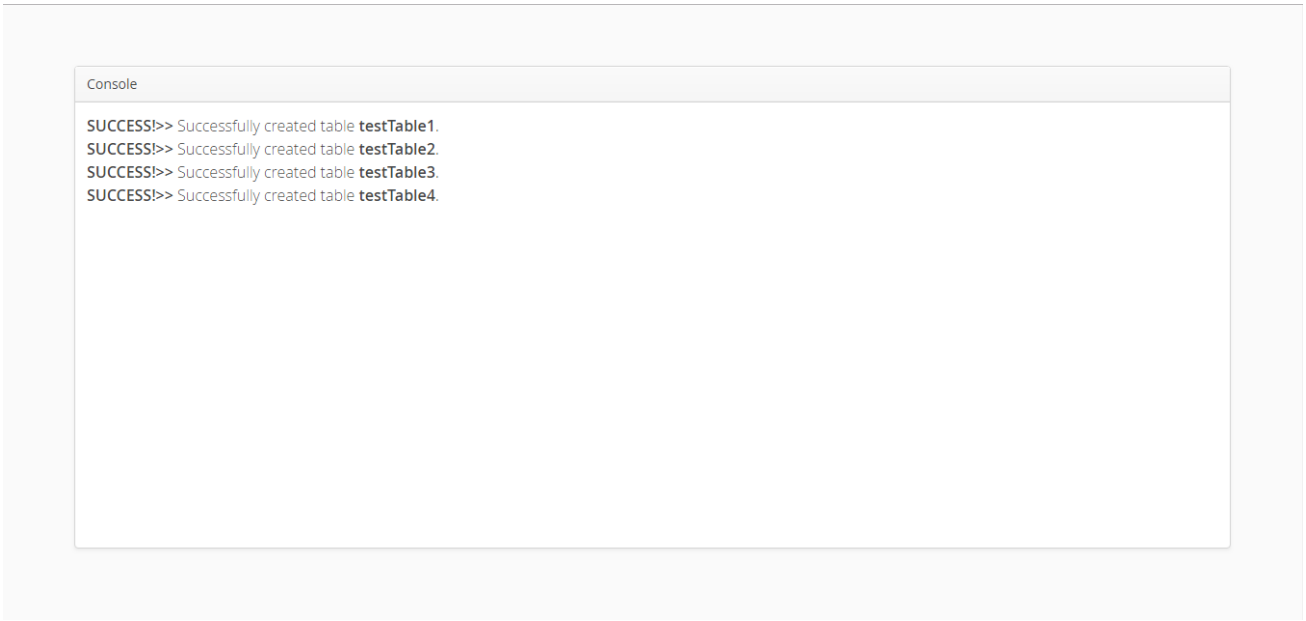
Asimismo, para cada tabla se crea un archivo con el nombre de esa tabla, el cual contiene el **árbol B+** que aloja los datos de esta, indexados según la primary key establecida. En la carpeta *src/main/java/com/cusbromen//bptree*, se encuentran los archivos fuente para la creación del árbol. Para más información sobre como funciona un árbol B+ se puede consultar la literatura de árboles B+[1].

[1] B+ tree. Disponible en: https://en.wikipedia.org/wiki/B%2B_tree
B+ trees. Disponible en: https://www.sci.unich.it/~acciaro/bpiutrees.pdf

Para la interfaz gráfica se utilizó el framework para Java en la Web, Vaadin 8. El IDE está compuesto por un panel que muestra las bases de datos en el DBMS, con la opción de seleccionar qué base de datos se utilizará para las consultas, así como una opción para mostrar la jerarquía de las tablas por cada base. A la derecha de este panel, se encuentra el editor, donde se pueden ingresar las consultas siguiendo el correspondiente lenguaje SQL, y además se encuentran 4 botones: ejecutar, despejar la consola, despejar el editor y verbose. El botón para ejecutar, corre las consultas escritas, los botones para despejar la consola y el editor eliminan cualquier texto o tabla mostrada y las dejan limpias; el botón de verbose activa la funcionalidad para mostrar los logs de las instrucciones realizadas, como función de debug.



En la parte inferior se encuentra la consola, que muestra los resultados, errores y cualquier otra información.



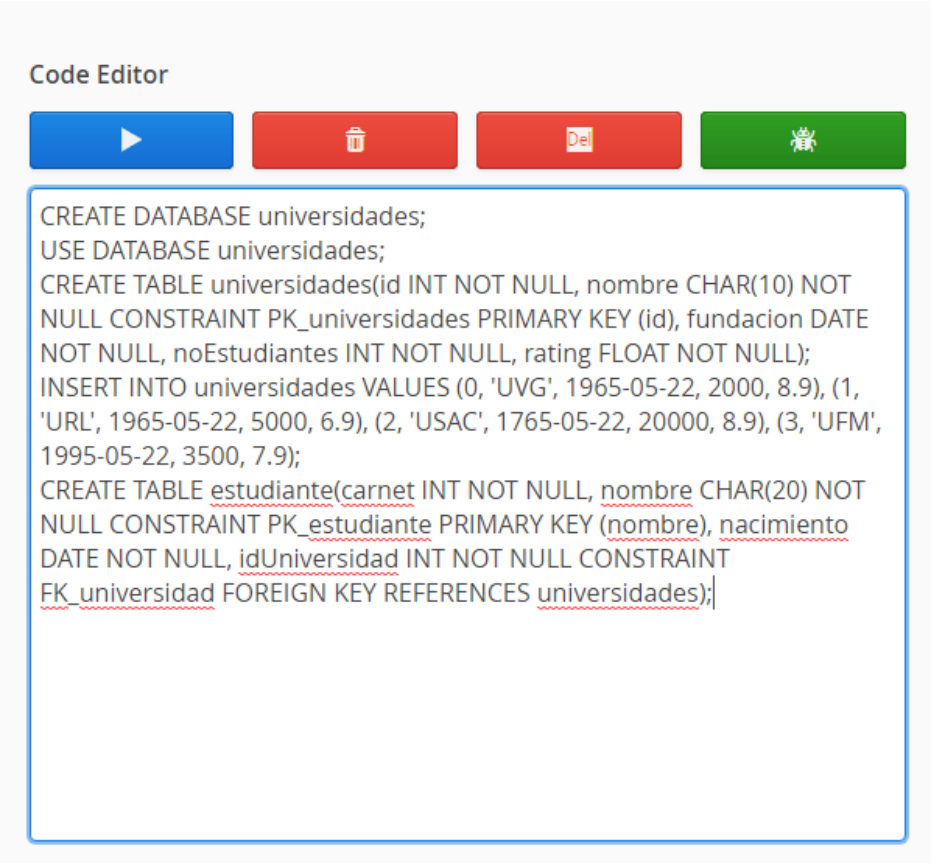
Pruebas

Prueba No. 1

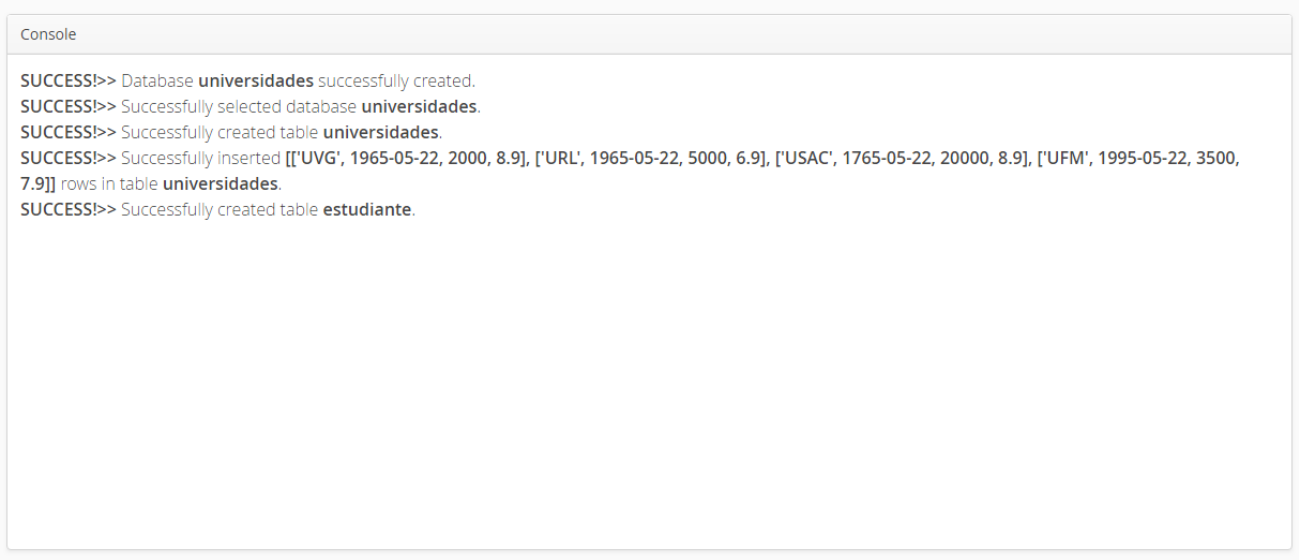
Considere la siguiente secuencia de scripts y resultados obtenidos:

```
CREATE DATABASE universidades;
USE DATABASE universidades;
CREATE TABLE universidades(id INT NOT NULL, nombre CHAR(10) NOT NULL CONSTRAINT PK_universidades PRIMARY KEY (id), fundacion DATE NOT NULL, noEstudiantes INT NOT NULL, rating FLOAT NOT NULL);
INSERT INTO universidades VALUES (0, 'UVG', 1965-05-22, 2000, 8.9), (1, 'URL', 1965-05-22, 5000, 6.9), (2, 'USAC', 1765-05-22, 20000, 8.9), (3, 'UFM', 1995-05-22, 3500, 7.9);
CREATE TABLE estudiante(carnet INT NOT NULL, nombre CHAR(20) NOT NULL CONSTRAINT PK_estudiante PRIMARY KEY (nombre), nacimiento DATE NOT NULL, idUniversidad INT NOT NULL CONSTRAINT FK_universidad FOREIGN KEY REFERENCES universidades);
```

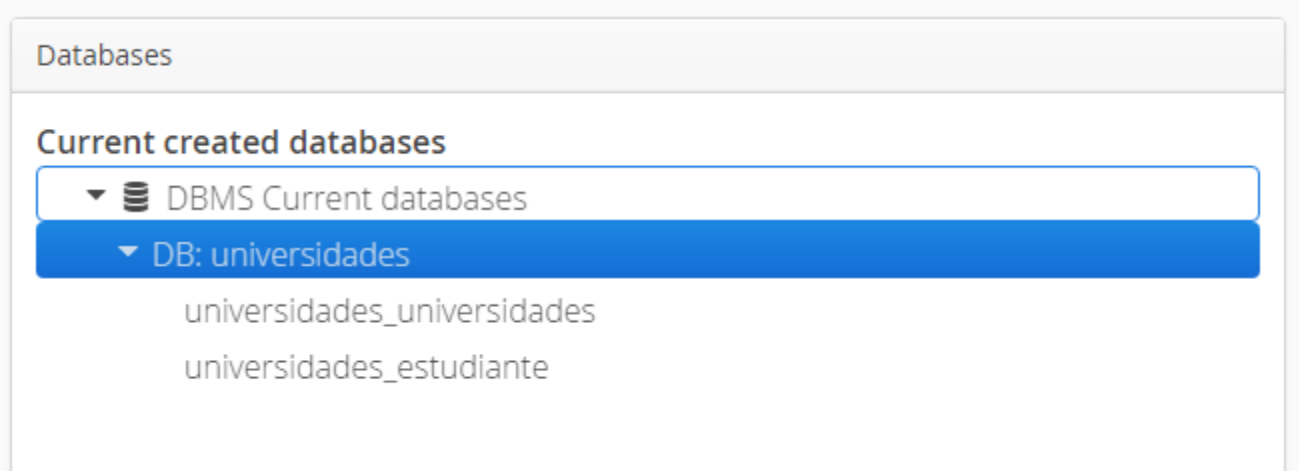
A continuación, el ingreso en el IDE elaborado:



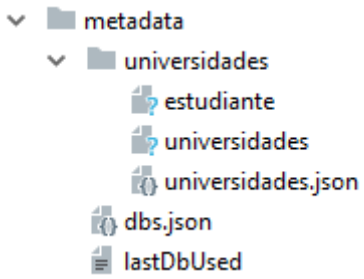
La ejecución del anterior script da como resultado el siguiente output en la consola:



Se puede notar en el panel de las bases de datos, la adición de la nueva base, con sus respectivas tablas.



Asimismo, nótese que los archivos de metadatos y los que contienen los árboles con los datos, son creados:



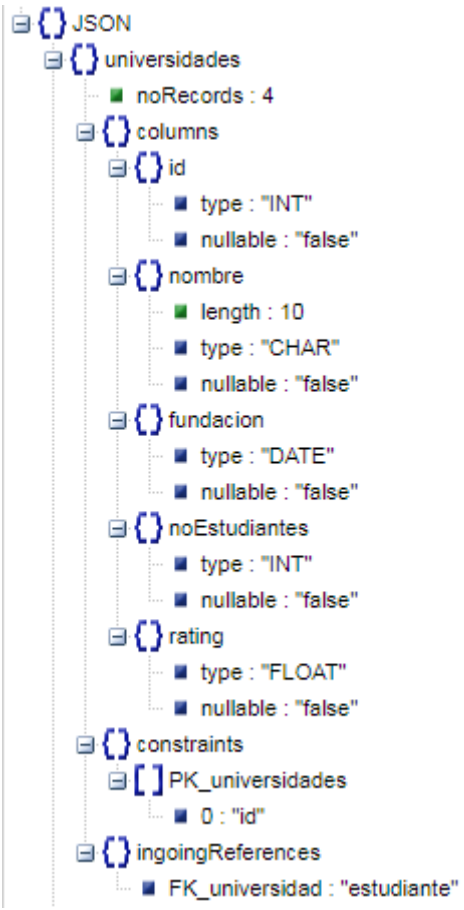
dbs.json

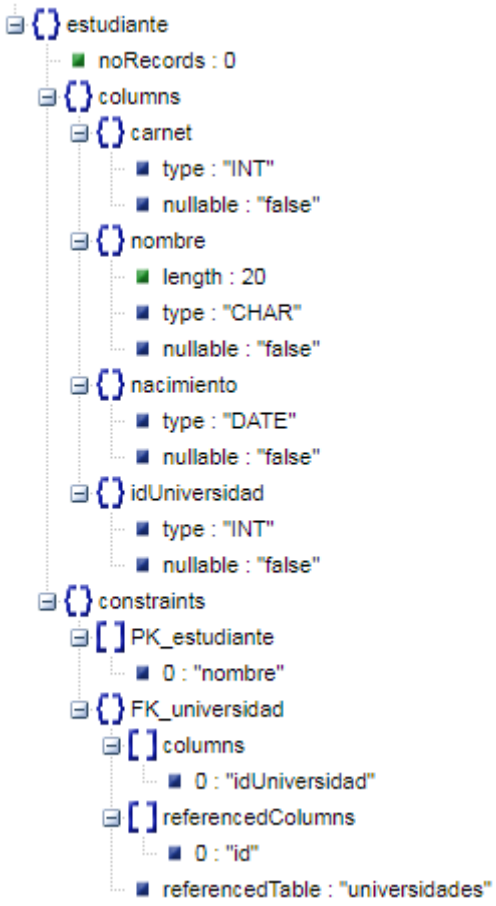
```
{"universidades":{"noTables":2}}
```

universidades.json

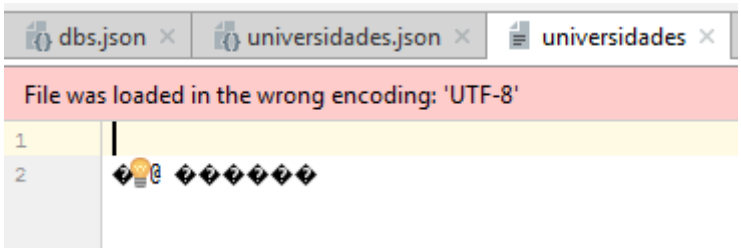

```
{"universidades":{"noRecords":4,"columns":{"id":{"type":"INT","nullable":
"false"},"nombre":{"length":10,"type":"CHAR","nullable":"false"},"fundacion
":{"type":"DATE","nullable":"false"},"noEstudiantes":{"type":"INT","nullabl
e":"false"},"rating":{"type":"FLOAT","nullable":"false"}},"constraints":{"P
K_universidades":["id"]},"ingoingReferences":{"FK_universidad":"estudiante"
}},"estudiante":{"noRecords":0,"columns":{"carnet":{"type":"INT","nullable"
:"false"},"nombre":{"length":20,"type":"CHAR","nullable":"false"},"nacimien
to":{"type":"DATE","nullable":"false"},"idUniversidad":{"type":"INT","nulla
ble":"false"}},"constraints":{"PK_estudiante":["nombre"],"FK_universidad":{
"columns":["idUniversidad"],"referencedColumns":["id"],"referencedTable":"u
niversidades"}}}}}
```


La estructura del JSON creado para los metadatos es la siguiente:



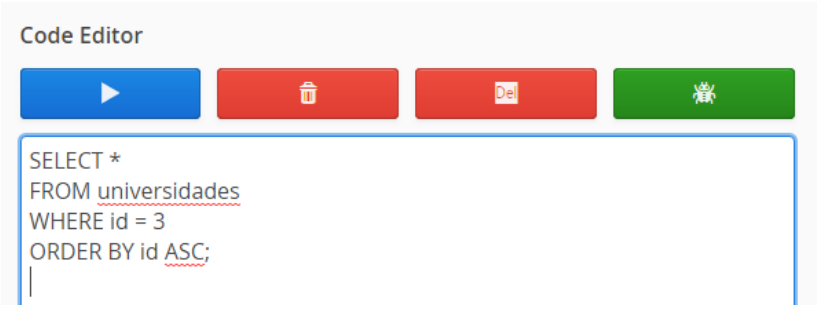


universidades.bin



Luego se ejecuta el siguiente script:

```
SELECT *
FROM universidades
WHERE id = 3
ORDER BY id ASC;
```



Obteniendo así el resultado de la consulta: