Universidad Nacional de Rosario
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Licenciatura en Ciencias de la Computación
Análisis de Lenguajes de Programación

Generador de Reportes de Bases de Datos

Alumna:
BORRERO, Paula (P-4415/6)

Docente:
JASKELIOFF, Mauro

Contents

Introducción

Una base de datos es una colección organizada de datos. Las bases de datos forman parte esencial de casi todas las empresas actuales. Les permite mantener un registro de todos los datos que consideran relevantes. No solo sirven para mantener datos que les permiten cumplir sus funciones diarias a nivel operativo, la disponibilidad de grandes cantidades de información que pueden relacionar y visualizar con relativa facilidad juega un papel importante en la toma de decisiones.
También son usadas en investigación e informática para almacenar y gestionar grandes cantidades de datos, así como también para obtener información a partir de ellos.
Un generador de reportes de base de datos permite elegir parte de los datos contenidos en una base para representarlos en un formato más amigable al ojo humano. Esto permite a las personas que tengan que utilizarlos, visualizarlos de un modo prolijo y tan fácil de comprender como los mismos datos lo permitan.
El presente trabajo es un generador de reportes hecho en Haskell para bases de datos SQLite3 que presenta los datos en tablas en formato PDF. Diversos aspectos de la presentación final pueden modificarse antes de generarlo, por ejemplo el tamaño de la página, el título, el formato de la tabla, etc.

Manual

Instalación

Primero se debe instalar:

- SQLite3
- La librería readline (editar líneas de comandos)
- La librería ncurses5 (manejo de la terminal)

Si se cuenta con el sistema de gestión de paquetes apt, todo esto se puede hacer mediante el siguiente comando:

```
sudo apt-get install sqlite3 libsqlite3-dev libreadline-dev libncurses5-dev
```

Luego hay que posicionarse en la carpeta raíz del proyecto (DBR) y ejecutar:

```
cabal install
```

Uso

El generador de reportes funciona como un intérprete de comandos en una terminal.

Inicio

./DBR *database* donde database es el path a una base de datos SQLite 3.

Comandos disponibles

- Exclusivos del intérprete:
  - **load *filename***
    Carga un archivo de configuración, el mismo sólo puede tener comandos de modificación (los que estan en el punto *De configuración*)
  - **show *item***
    Muestra la configuración actual del elemento *item* donde *item* puede ser:
    - **query**: consulta actual.
    - **table layout**: diseño de los bordes de la tabla.
    - **table header font**: fuente de los títulos de las columnas.
    - **table body font**: fuente del resto de las celdas de las columnas.
    - **paper type**: tamaño del papel.
    - **paper landscape status**: determinar si el papel está apaisado o no.
    - **paper title status**: determinar si se muestra el título o no.
    - **title**: contenido del título.
    - **title font**: fuente del título del reporte.
  - **generate**
    Genera el pdf con el nombre pasado como título. Si no se le pasó ninguno toma el título por defecto.

- De configuración:
  Modifican la configuración de un aspecto particular del reporte actual.
  - **query *selectquery***: Cambia el contenido de la tabla. *selectquery* debe ser una consulta (SELECT) escrita en SQL.
  - **title**
    Modifica el título del reporte en los siguientes aspectos:
    * ***titletext***: contenido del título. *titletext* debe ser una cadena.
    * **pos [center, left, right]**: alineación del texto.
    * **font [roman, serif, mono] *tamaño***: fuente del título, donde *tamaño* es un natural.
    * **decor *d1 d2 ... dn***: decoraciones del título.
  - **table**
    Cambia el diseño de la tabla en los siguientes aspectos:
    * **header**: títulos de las columnas.
      · **font [roman, serif, mono] *tamaño***: fuente, donde *tamaño* es un natural.
      · **decor *d1 d2 ... dn***: decoraciones.
    * **body**: cuerpo de la tabla. Continúa igual que **header**.
    * **rows *fst rst***: número de filas en la tabla de la primer página (*fst*) y en las del resto de las páginas (*rst*). *fst* y *rst* son números enteros.
    * **layout**: contorno de la tabla.
      · ***v1 h1 v2 h2*** donde:
      v1 son las líneas verticales externas.
      h1 son las líneas horizontales externas.
      v2 son las líneas verticales internas.
      h2 son las líneas horizontales internas.
      Además *vi* es una de:
      none: sin líneas.
      single: líneas simples.
      double: líneas dobles.
      Y *hi* es una de:
      false: sin líneas.
      true: líneas simples.
  - **paper**
    Cambia el diseño de la página de la siguiente forma:
    * **set *label***: activa banderas de configuración. *label* puede ser:
      · ***landscape***: colocar la hoja en modo apaisado.
      · ***title***: mostrar título en la primer página.
    * **unset**: desactiva las banderas que se pueden activar con set.
    * ***size***: cambiar el tamaño de la hoja. *size* puede ser: **a0, a1, ..., a6, b0, b1,.., b5, letter** o **legal**

Observaciones:

1. *decor* es una de:
   - **normal**: sin decoraciones.
   - **bold**: negrita.
   - **italics**: itálica.
   - **smallcaps**: versalita.
   - **slanted**: inclinada.
   - **upright**: vertical.
   - **underline**: subrayada.
2. Las enumeraciones entre corchetes se usan cuando se debe tomar uno sólo de dichos elementos.

Organización de los Archivos

La carpeta raíz del trabajo contiene en 3 subcarpetas:

- **src**: Contiene los archivos con código fuente.
  - **Main.hs**: Módulo principal, en él se implementa el intérprete.
  - **TTree.hs**: Módulo con el tipo principal de los reportes.
  - **DBReport.hs**: Módulo con las implementaciones de construcción, manipulación y ejecución del DSL, además de funciones para examinar el contenido de un elemento del tipo principal.
  - **Parser.hs**: Módulo que contiene el parser de comandos de modificación.
- **docs**: Contiene la documentación.
- **tests**: Contiene los casos de prueba. Los mismos se hicieron usando una base de datos de muestra que se encuentra en el archivo *chinook.db*. También hay un archivo llamado *chinook diagram.pdf* en el que se puede ver el diagrama de la base de datos.

Decisiones de Diseño

1. **¿Qué tipo de base de datos usar?**
   En un principio probé con MySQL pero finalmente opté por usar SQLite3 ya que era más portable. De todos modos, por lo que pude probar, no sería demasiado difícil usar lo ya hecho para trabajar con MySQL.
2. **¿Qué formato de salida usar?**
   En todo momento quise que fuera PDF ya que es un formato portable y se obtienen resultados prolijos.

   Esta decisión trajo algunos problemas al principio pues estaba usando la biblioteca HPDF para generarlos. La documentación no me ayudó en algunos puntos como

generar tablas que se vieran bien y tuve problemas con las unidades de medida y las dimensiones de las cosas. Llegué a imprimir texto alineado pero que no se veía bien. Esto me llevó a usar HaTeX para generar, en un paso previo, un archivo LaTeX, ejecutar desde dentro de Haskell pdflatex para convertirlo a pdf y borrar los intermedios. En este paso pdflatex me tiraba el resultado de su ejecución a la terminal en donde estaba corriendo el interprete. Esto no quedaba muy bien y lo pude evitar usando la función *silence* de la biblioteca *System.IO.Silently*.

HaTeX no tiene todos los paquetes disponibles para LaTeX entonces esta decisión también afectó el aspecto estético del trabajo ya que el formato final es más restringido del que podría hacer escribiendo directamente el código LaTeX.

3. **¿Por qué un intérprete?**
   Hice un interprete en vez de hacer una biblioteca para Haskell porque quería que pudiera ser usada con relativa facilidad por cualquier persona. Sólo hay que aprenderse unos cuantos comandos para usarlo. De todos modos esto se restringe un poco al pedir que las consultas sean en SQL.

4. **¿Por qué las consultas son en SQL?**
   En un principio pensé en hacer las consultas de forma que no fuera necesario usar SQL, el programa sería el que lo transformara en una consulta. Sin embargo, me pareció que era un trabajo no tan importante ya que las consultas SELECT de SQL no son tan complejas de aprender y me pareció mejor dejar que se pueda usar todo su potencial.

5. **¿Por qué usar un parser?**
   Hice un parser para permitir archivos de configuración que pudieran cargarse. De este modo se pueden guardar las configuraciones que nos gusten y podemos utilizarlas fácilmente en el futuro. Para esto hice un parser usando la biblioteca Parsec.

Sobre el Lenguaje de Dominio Específico

Se definió un DSL que manipula el tipo Repo definido en TTree.hs. La implementación las operaciones del DSL se encuentran en DBRepo.hs y es la siguiente:

- **Constructor**:
  Construye un elemento con el formato por defecto para la conexión dada.

  ```
  initRepo :: Connection -> Repo
  ```

- **Manipulación**:

  ```
  titleNew :: String -> Repo -> Repo
  titlePos :: HPos -> Repo -> Repo
  titleFont :: FontFamily -> Integer -> Repo -> Repo
  titleDecor :: [FontStyle] -> Repo -> Repo
  query :: String -> Repo -> Repo
  paperSize :: PaperType -> Repo -> Repo
  paperLands :: Bool -> Repo -> Repo
  ```

```
paperTitle :: Bool -> Repo -> Repo
tableLayout :: Vert -> Bool -> Vert -> Bool -> Repo -> Repo
tableRows :: Integer -> Integer -> Repo -> Repo
tableBFont :: FontFamily -> Integer -> Repo -> Repo
tableHFont :: FontFamily -> Integer -> Repo -> Repo
tableBDecor :: [FontStyle] -> Repo -> Repo
tableHDecor :: [FontStyle] -> Repo -> Repo
```

- **Ejecución**
  Genera el PDF final

  ```
  generate :: Repo -> IO ()
  ```

Modificaciones posibles

Este trabajo tiene muchísimas cosas que se pueden agregar, estas son algunas particularmente interesantes:

1. Extenderlo para otros tipos de bases de datos (por ejemplo MySQL).
2. Agregar comandos para obtener más información de la base de datos (nombres de tablas/columnas, etc).
3. Agregarle más características modificables (hay muchas para elegir).
4. Hacerlo como una biblioteca de Haskell (posiblemente usando una mónada de estado).
5. Hacer una versión en la que en vez de consultas SQL se usen opciones sencillas (columnas a incluir, restricciones, etc.) que luego puedan ser traducidas a SQL.

Bibliografía y código de terceros

- `http://book.realworldhaskell.org/read/using-databases.html`
- `https://en.wikibooks.org/wiki/LaTeX/Tables`
- `http://hackage.haskell.org/package/HaTeX-3.17.0.2/docs`
- `https://wiki.haskell.org/Tutorials/Programming_Haskell/Argument_handling`
- Trabajos prácticos de la materia, sobre todo para el parser.