



Máster Universitario En Investigación En Ingeniería De Software Y Sistemas Informáticos

Generación Automática de Código

Autor

César Hugo Bárzano Cruz



TRABAJO 1 DE EVALUACIÓN CONTINUA

—
2017/2018

Índice general

1. Introducción	7
2. Desarrollo de la Práctica	9
2.1. Tecnologías Utilizadas	9
2.2. Especificación de la Solución	9
2.3. Análisis de la solución	12
2.4. Pruebas	12
2.4.1. Pruebas de Funcionalidad	12
2.4.2. Pruebas de Error	24
3. Entrega	25
3.1. Generador	25
3.2. Makefile	25
3.3. Directorio software specification	25
3.4. Trabajo1_CesarHugoBarzanoCruz.pdf	25
3.5. Directorio DOC	26
3.6. Directorio input	26
3.7. Directorio Output	26
4. Anexo	29
4.1. generator.py	29
4.2. makefile	31

Índice de figuras

2.1. Mysql status	13
2.2. Test Case 1	14
2.3. Test Case 1 - HTML Output	14
2.4. Test Case 1 - XML Output	15
2.5. Postgres status	15
2.6. Test Case 2	16
2.7. Test Case 3	16
2.8. Test Case 3	17
2.9. Test Case 4 - HTML Output	18
2.10. Test Case 4 - XML Output	18
2.11. Test Case 5	19

Capítulo 1

Introducción

El presente documento representa la memoria formal para evaluación continua de la segunda práctica de la asignatura Generación Automática de Código. Dicha práctica consiste en un generador de datos en formatos (CSV , HTML , XML, JSON) como salida a partir de un sistema de gestión de bases de datos (SGBD) relacional como entrada.

En los siguientes capítulos se detallará la solución propuesta, las tecnologías utilizadas, se analizarán las ventajas e inconvenientes de la solución propuesta, se explicarán los casos de uso o pruebas a los que se ha sometido el generador, se explicará como utilizar correctamente el generador así como las dependencias necesarias para su correcto uso.

Finalmente se analizarán las tareas y conocimientos adquiridos en esta práctica junto con un anexo de los documentos necesarios.

Capítulo 2

Desarrollo de la Práctica

2.1. Tecnologías Utilizadas

Como tecnología base, se ha decidido utilizar el lenguaje de programación interpretado Python[3], versión 2.7.15 debido a la sencillez y capacidad del mismo para proporcionar una solución.

Como se indicó en la introducción, la entrada del generador a impletar es uno o varios sistemas de gestión de bases de datos(SGBD). Los SGBDs que este generador soporta son los siguientes:

1. **mysql**
2. **postgres**
3. **sqlite3**

El sistema operativo base sobre el que se han realizado las pruebas es Ubuntu 18.04 LTS

Adicionalmente para la confección de esta memoria se ha utilizado LaTeX con el paquete de librerías texlive-full y el editor texmaker. Los siguientes enlaces muestran como instalar y utilizar correctamente LaTeX, han sido utilizados como referencia para el presente documento.

1. Instalar LaTeX
2. Usar LaTeX

2.2. Especificación de la Solución

Las posibles especificaciones e implementaciones para un generador como el propuesto por el enunciado pueden ser muy variadas. Dichas especificaciones pueden variar significativamente en función de las tecnologías utilizadas,

las aserciones o comprobaciones de robustez que se quieran realizar a las entradas, al procesamiento o a las salidas, los posibles casos que se quieran tener en cuenta así como el grado de modularidad que se quiera alcanzar de cara a futuras actualizaciones.

El generador utiliza a modo de configuración un fichero Json en el que vienen especificados los parametros necesarios para establecer la conexión con los distintos SGBD soportados. Un Ejemplo de entrada para este fichero de configuración es el siguiente:

Listing 2.1: Entrada Unitaria Configuración Generador

```
1 {  
2 ...  
3  
4 "mysql_1": {  
5     "user": "user",  
6     "password": "pass",  
7     "host": "host",  
8     "database": "database"  
9 },  
10 ...  
11 }
```

Las entradas usan como clave el identificador del tipo de SGBD, esto permite al generador diferenciar el tipo de conector a usar para poder extraer los datos, es decir, `mysql_n` establece la informacion necesaria para conectarse al SGBD número N basado en mysql. Esto permite configurar tantos SGBD basados en mysql como se quiera. El siguiente ejemplo muestra un ejemplo de configuración más completo en el que se incluyen todos los SGBD soportados por el generador:

Listing 2.2: Entrada Completa Configuración Generador

```
1 {  
2  
3     "mysql_1": {  
4         "user": "user",  
5         "password": "pass",  
6         "host": "host",  
7         "database": "database"  
8     },  
9     "mysql_2": {  
10         "user": "user2",  
11         "password": "pass2",  
12         "host": "host2",  
13         "database": "database2"  
14     },  
15     "postgres_1": {  
16         "user": "postgres",  
17         "password": "123456",  
18         "host": "host",  
19         "database": "database"  
20     },  
21     "sqlite3_1": {  
22         "db_path": "path/to/database/file"
```

```

23     }
24 }

```

Como podemos observar en el ejemplo anterior, el fichero de configuración para el generador nos permite definir tantos SGBD como necesitemos indicando siempre el indicador de tecnología: `mysql_N`, `postgres_N` o `sqlite3_N`.

En los casos de `mysql` y `postgres` es necesario establecer la configuración necesaria para la conexión indicando usuario, contraseña, host (url o ip) y base de datos a la que conectar. De manera adicional el generador soporta `sqlite3` pero debido a ser esta una base de datos para pruebas generalmente en local, la información necesaria es el path al fichero en disco que actúa como base de datos. Para este último caso es necesario tener en cuenta que dicho fichero tiene que tener los permisos adecuados (lectura al menos) para que el generador sea capaz de leer la información de dicha base de datos.

El generador cuenta con la siguiente ayuda con el objetivo de facilitar la comprensión de los argumentos que necesita para un uso nominal.

Listing 2.3: `python generadorP2.py -help`

```

1
2 Usage: python generador.py [options]
3
4 Options
5 -v, --version          Show the version of this script
6 -h, --help             Show this help.
7 -t <table_name>, --table <table_name>  Input database table
8 -w <"column = 'value'">, --where <"column = 'value'">  Optional Where
   Clause
9 -d, --debug            Debug Mode

```

Como podemos observar en la ayuda del generador, es necesario el flag `-t` o `--table` para indicar la tabla de la cual el generador va a extraer la información dentro de los distintos SGBD establecidos en la configuración.

De manera adicional, se puede añadir el flag `-w` o `--where` para filtrar utilizando la cláusula relacional `WHERE`.

Con el objetivo de facilitar la comprensión de que hace el generador en cada etapa de su ejecución, se ha incluido el flag `-d` o `--debug` el cual permite ver por la salida estándar que acción está realizando el generador.

Tras una ejecución, el generador creará en el directorio `OUTPUT` 4 ficheros por cada uno de los SGBD especificados con la siguiente convención de nombres:

1. `GAC_mysql_1_20180820_203859.csv`
2. `GAC_mysql_1_20180820_203859.html`
3. `GAC_mysql_1_20180820_203859.json`
4. `GAC_mysql_1_20180820_203859.xml`

2.3. Análisis de la solución

Se considera que la solución propuesta es valida, ya que cumple con lo solicitado, realiza las adecuadas comprobaciones para una correcta ejecución, genera los distintos formatos esperados y además se ha alcanzado de una manera sencilla, en un solo script python junto con los conectores necesarios para cada uno de los SGBD tratados. La modularidad de la solución propuesta favorece posibles actualizaciones del generador, incluyendo en el bucle de iteración principal el conector necesario para soportar nuevas bases de datos. El generador carga su configuración mediante variables de entorno (`GENERATOR_CONFIG`) lo que permite cambiar rapidamente el tipo de configuración a utilizar en cada ejecución.

La solución alcanzada no es completa debido a los conectores con las bases de datos, ya que como se ha explicado antes solo soporta 3 SGBD por lo que quizas esta seria su principal defecto.

2.4. Pruebas

Con el Objetivo de mostrar la potencia del generador, se van a realizar una serie de ejecuciones de prueba para mostrar todos los posibles escenarios del generador. Para ello es necesario disponer de los 3 tipos de SGBD soportadas por el generador con datos significativos que exportar. Para ello se incluye el fichero `init.py` cuyo objetivo es el de añadir entradas a las bases de datos instaladas localmente en mi equipo.

Para facilitar los distintos escenarios de pruebas a los que vamos a someter al generador, se va a utilizar el usuario `root` con password `root` para la base de datos `mysql`, una base de datos `postgres` cuyo usuario y contraseña es `postgres:123456` y como detallamos anteriormente una base de datos `sqlite3` representada por un fichero en disco.

Adicionalmente se va a mostrar ciertos casos de error tenidos en cuenta en la implementación de la solución, por lo que se propone dividir esta sección en las siguientes sub-secciones. Para automatizar este proceso se ha creado el `Makefile` que se adjunta en el anexo donde se especifica cada uno de los `testCases_N.sh` que serán ejecutados, tomando como entrada los ficheros alojados en el directorio `input` y dejando el resultado en el directorio `output`. En las capturas que muestran cada una de las pruebas se ve todo lo necesario para su ejecución.

2.4.1. Pruebas de Funcionalidad

Para mostrar la correcta funcionalidad del generador se va a comenzar exportando datos de cada uno de los SGBD de manera independiente. Tras esto, se ejemplificará el uso de la clausula `WHERE` para filtrar los datos a exportar y finalmente se mostrarán casos de pruebas mas complejos en los

que se atará con una sola ejecución a todos los SGBD soportados por el generador.

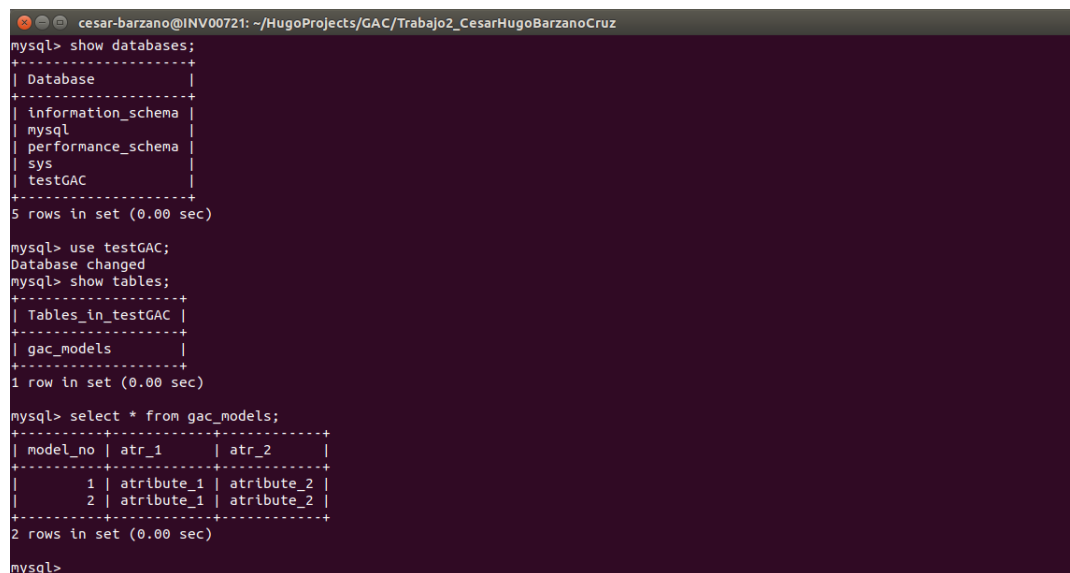
Test Case 1 - mysql

A continuación se muestra la configuración utilizada por el generador para el primer caso de prueba.

Listing 2.4: GAC_GENERATOR_CONFIG_1.json

```
1 {
2     "mysql_1": {
3         "user": "root",
4         "password": "root",
5         "host": "localhost",
6         "database": "testGAC"
7     }
8 }
```

Utilizando la línea de órdenes para mysql, podemos observar cual es el estado interno del SGBD



```
cesar-barzano@INV00721: ~/HugoProjects/GAC/Trabajo2_CesarHugoBarzanoCruz
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
| testGAC |
+-----+
5 rows in set (0.00 sec)

mysql> use testGAC;
Database changed
mysql> show tables;
+-----+
| Tables_in_testGAC |
+-----+
| gac_models |
+-----+
1 row in set (0.00 sec)

mysql> select * from gac_models;
+-----+
| model_no | atr_1 | atr_2 |
+-----+
| 1 | attribute_1 | attribute_2 |
| 2 | attribute_1 | attribute_2 |
+-----+
2 rows in set (0.00 sec)

mysql>
```

Figura 2.1: Mysql status

Ejecutando el primer caso de prueba mediante `make testCase1`, el generador mostrará por pantalla lo siguiente:

```
cesar-barzano@INV00721: ~/HugoProjects/GAC/Trabajo2_CesarHugoBarzanoCruz
cesar-barzano@INV00721:~/HugoProjects/GAC/Trabajo2_CesarHugoBarzanoCruz$ make testCase1
sh TestCase_1.sh
GAC: P2
Mysql: mysql_1
Writing output file in ./output/GAC_mysql_1_20180829_202227.xml
Writing output file in ./output/GAC_mysql_1_20180829_202227.csv
Writing output file in ./output/GAC_mysql_1_20180829_202227.json
Writing output file in ./output/GAC_mysql_1_20180829_202227.html
cesar-barzano@INV00721:~/HugoProjects/GAC/Trabajo2_CesarHugoBarzanoCruz$
```

Figura 2.2: Test Case 1

La ejecución del primer caso de prueba produce las siguientes salidas dentro del directorio output

Listing 2.5: Formato CSV

```
1 1,atribute_1,atribute_2
2 2,atribute_1,atribute_2
```

Listing 2.6: Formato HTML

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>mysql_1</title>
5 </head>
6 <body>
7 <h1>mysql_1</h1><hr><br><table style="width:100%"><tr><td>2</td><td>atribute_1</td><td>atribute_2</td></tr></table></body></html>
```

El resultado del generador para formato HTML queda mas claro si es interpretado por el navegador:

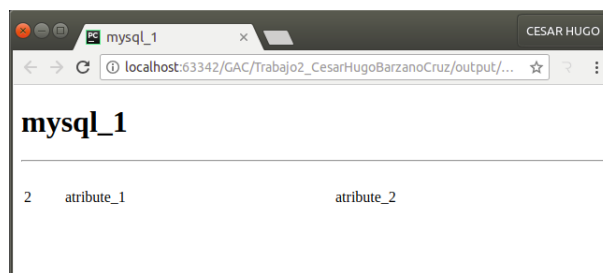


Figura 2.3: Test Case 1 - HTML Output

Listing 2.7: Formato Json

```
1 [[1, "atribute_1", "atribute_2"], [2, "atribute_1", "atribute_2"]]
```

Listing 2.8: Formato XML

```

1 <data>&lt;data&gt;&lt;i&gt;&lt;i&gt;1&lt;/i&gt;&lt;i&gt;atribute_1&lt;
  ;/i&gt;&lt;i&gt;atribute_2&lt;/i&gt;&lt;/i&gt;&lt;i&gt;&lt;i&gt;2&
  lt;/i&gt;&lt;i&gt;atribute_1&lt;/i&gt;&lt;i&gt;atribute_2&lt;/i&gt
  ;&lt;/i&gt;&lt;/data&gt;</data>

```

El resultado del generador para formato xml queda mas claro si es interpretado por el navegador:



Figura 2.4: Test Case 1 - XML Output

Test Case 2 - Postgres

A continuación se muestra la configuración utilizada por el generador para el segundo caso de prueba.

Listing 2.9: GAC_GENERATOR_CONFIG_2.json

```

1 {
2   "postgres_1": {
3     "user": "postgres",
4     "password": "123456",
5     "host": "127.0.0.1",
6     "database": "testgac"
7   }
8 }

```

Utilizando la linea de ordenes para Postgres, podemos observar cual es el estado interno del SGBD. Por simplicidad la estructura de la base de datos es similiar a mysql.

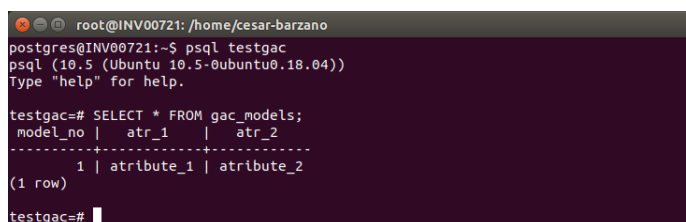


Figura 2.5: Postgres status

El segundo caso de prueba puede ejecutarse mediante make testCase2, el resultado es similiar al caso de prueba 1.

```

cesar-barzano@INV00721: ~/HugoProjects/GAC/Trabajo2_CesarHugoBarzanoCruz
cesar-barzano@INV00721:~/HugoProjects/GAC/Trabajo2_CesarHugoBarzanoCruz$ make testCase2
sh TestCase_2.sh
GAC: P2
Postgrespostgres_1
Writing output file in ./output/GAC_postgres_1_20180831_180830.xml
Writing output file in ./output/GAC_postgres_1_20180831_180830.csv
Writing output file in ./output/GAC_postgres_1_20180831_180830.json
Writing output file in ./output/GAC_postgres_1_20180831_180830.html
cesar-barzano@INV00721:~/HugoProjects/GAC/Trabajo2_CesarHugoBarzanoCruz$

```

Figura 2.6: Test Case 2

Test Case 3 - Sqlite3

A continuación se muestra la configuración utilizada por el generador para el tercer caso de prueba.

Listing 2.10: GAC_GENERATOR_CONFIG_3.json

```

1 {
2   "sqlite3_1": {
3     "db_path": "./testGAC.db"
4   }
5 }

```

El tercer caso de prueba puede ejecutarse mediante `make testCase3`, el resultado es similar al caso de prueba 1 y 2.

```

cesar-barzano@INV00721: ~/HugoProjects/GAC/Trabajo2_CesarHugoBarzanoCruz
cesar-barzano@INV00721:~/HugoProjects/GAC/Trabajo2_CesarHugoBarzanoCruz$ make testCase3
sh TestCase_3.sh
GAC: P2
Sqlite3: ./testGAC.db
Writing output file in ./output/GAC_sqlite3_1_20180831_181354.xml
Writing output file in ./output/GAC_sqlite3_1_20180831_181354.csv
Writing output file in ./output/GAC_sqlite3_1_20180831_181354.json
Writing output file in ./output/GAC_sqlite3_1_20180831_181354.html
cesar-barzano@INV00721:~/HugoProjects/GAC/Trabajo2_CesarHugoBarzanoCruz$

```

Figura 2.7: Test Case 3

Test Case 4 - WHERE

El cuarto caso de prueba muestra el uso de la cláusula WHERE para filtrar los resultados del generador. De manera adicional el caso de prueba número 4 trata de mostrar la capacidad del generador para utilizar varios SGBD establecidos en la misma configuración. Por simplicidad se va a utilizar la misma base de datos mysql pero a efectos prácticos de configuración, podrían ser 2 SGBD totalmente independientes.

Listing 2.11: GAC_GENERATOR_CONFIG_4.json

```

1 {
2   "mysql_1": {
3     "user": "root",
4     "password": "root",

```



```

5         "host": "localhost",
6         "database": "testGAC"
7     },
8     "mysql_2": {
9         "user": "root",
10        "password": "root",
11        "host": "127.0.0.1",
12        "database": "testGAC"
13    }
14 }

```

Para mostrar la capacidad de filtrado mediante clausula WHERE, se van a filtrar todos los modelos cuyo número sea 1 es, es decir, `-where "model_no = '1'"`. El cuarto caso de prueba puede ser ejecutado mediante `make testCase4`

```

cesar-barzano@INV00721: ~/HugoProjects/GAC/Trabajo2_CesarHugoBarzanoCruz
cesar-barzano@INV00721:~/HugoProjects/GAC/Trabajo2_CesarHugoBarzanoCruz$ make testCase4
sh TestCase_4.sh
GAC: P2
Mysql: mysql_1
Writing output file in ./output/GAC_mysql_1_20180831_184552.xml
Writing output file in ./output/GAC_mysql_1_20180831_184552.csv
Writing output file in ./output/GAC_mysql_1_20180831_184552.json
Writing output file in ./output/GAC_mysql_1_20180831_184552.html
Mysql: mysql_2
Writing output file in ./output/GAC_mysql_2_20180831_184552.xml
Writing output file in ./output/GAC_mysql_2_20180831_184552.csv
Writing output file in ./output/GAC_mysql_2_20180831_184552.json
Writing output file in ./output/GAC_mysql_2_20180831_184552.html
cesar-barzano@INV00721:~/HugoProjects/GAC/Trabajo2_CesarHugoBarzanoCruz$

```

Figura 2.8: Test Case 3

La ejecución del cuarto caso de prueba produce las siguientes salidas dentro del directorio output, 4 ficheros con los correspondientes formatos para `mysql_1` junto con otros 4 ficheros con los correspondientes formatos para `mysql_2`. Como indicamos anteriormente las bases de datos son similares por lo que a continuación solo se va a mostrar los ficheros generados para `mysql_2`.

Listing 2.12: Formato CSV

```

1 1,attribute_1,attribute_2

```

Listing 2.13: Formato HTML

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>mysql_2</title>
5 </head>
6 <body>
7 <h1>mysql_2</h1><hr><br><table style="width:100%"><tr><td>1</td><td>attribute_1</td><td>attribute_2</td></tr></table></body></html>

```

El resultado del generador para formato HTML queda mas claro si es interpretado por el navegador:

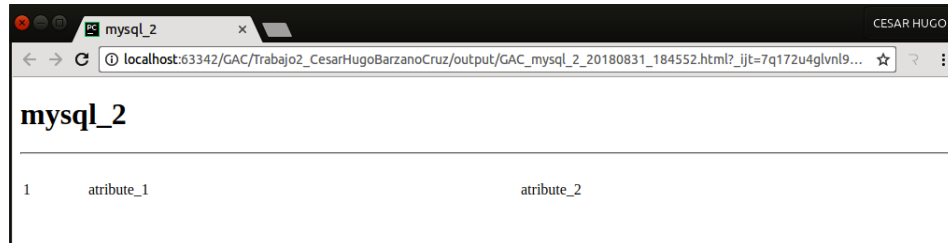


Figura 2.9: Test Case 4 - HTML Output

Listing 2.14: Formato Json

```
1 [[1, "atribute_1", "atribute_2"]]
```

Listing 2.15: Formato XML

```
1 <data>&lt;i>data&gt;&lt;i>i&gt;&lt;i>i&gt;1&lt;/i&gt;&lt;i>i&gt;atribute_1&lt;/i&gt;&lt;i>i&gt;atribute_2&lt;/i&gt;&lt;/i&gt;&lt;/data&gt;</data>
```

El resultado del generador para formato xml queda mas claro si es interpretado por el navegador:

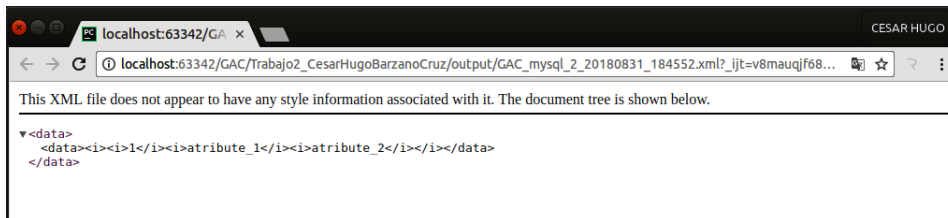


Figura 2.10: Test Case 4 - XML Output

Test Case 5 - All SGBD

El quinto caso de prueba muestra la capacidad del generador para producir salida involucrando a todos los SGBD soportados por el mismo. Para ello, a continuación se muestra la configuración aplicada al generador.

Listing 2.16: GAC_GENERATOR_CONFIG_5.json

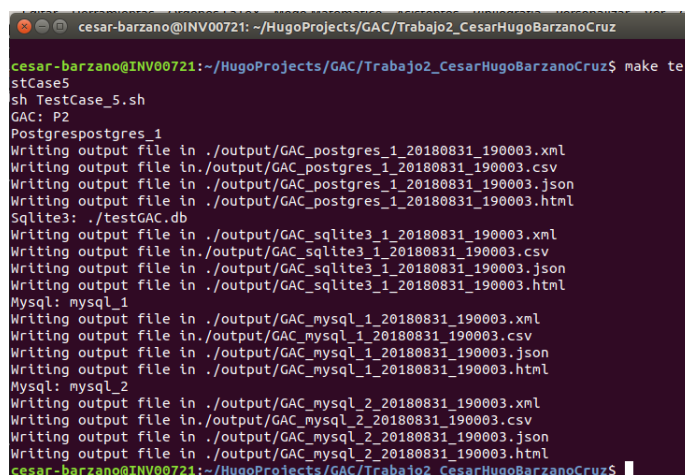
```
1 {
2   "mysql_1": {
3     "user": "root",
4     "password": "root",
```

```

5         "host": "localhost",
6         "database": "testGAC"
7     },
8     "mysql_2": {
9         "user": "root",
10        "password": "root",
11        "host": "127.0.0.1",
12        "database": "testGAC"
13    },
14    "postgres_1": {
15        "user": "postgres",
16        "password": "123456",
17        "host": "127.0.0.1",
18        "database": "testgac"
19    },
20    "sqlite3_1": {
21        "db_path": "./testGAC.db"
22    }
23 }

```

El quinto caso de prueba puede ser ejecutado mediante `make testCase5` el cual genera en el directorio `output` 4 ficheros con los correspondientes formatos para cada uno de los SGBDs configurados.



```

cesar-barzano@INV00721: ~/HugoProjects/GAC/Trabajo2_CesarHugoBarzanoCruz
cesar-barzano@INV00721:~/HugoProjects/GAC/Trabajo2_CesarHugoBarzanoCruz$ make testCase5
sh TestCase_5.sh
GAC: P2
Postgres: postgres_1
Writing output file in ./output/GAC_postgres_1_20180831_190003.xml
Writing output file in ./output/GAC_postgres_1_20180831_190003.csv
Writing output file in ./output/GAC_postgres_1_20180831_190003.json
Writing output file in ./output/GAC_postgres_1_20180831_190003.html
Sqlite3: ./testGAC.db
Writing output file in ./output/GAC_sqlite3_1_20180831_190003.xml
Writing output file in ./output/GAC_sqlite3_1_20180831_190003.csv
Writing output file in ./output/GAC_sqlite3_1_20180831_190003.json
Writing output file in ./output/GAC_sqlite3_1_20180831_190003.html
Mysql: mysql_1
Writing output file in ./output/GAC_mysql_1_20180831_190003.xml
Writing output file in ./output/GAC_mysql_1_20180831_190003.csv
Writing output file in ./output/GAC_mysql_1_20180831_190003.json
Writing output file in ./output/GAC_mysql_1_20180831_190003.html
Mysql: mysql_2
Writing output file in ./output/GAC_mysql_2_20180831_190003.xml
Writing output file in ./output/GAC_mysql_2_20180831_190003.csv
Writing output file in ./output/GAC_mysql_2_20180831_190003.json
Writing output file in ./output/GAC_mysql_2_20180831_190003.html
cesar-barzano@INV00721:~/HugoProjects/GAC/Trabajo2_CesarHugoBarzanoCruz$

```

Figura 2.11: Test Case 5

No se va a mostrar el contenido de los ficheros generados debido a que el estado de las bases de datos no ha cambiado con respecto a los casos de prueba anteriores.

2.4.2. Pruebas de Error

Para realizar las pruebas de error, se han definido 2 escenarios.

Test Case Error - 1

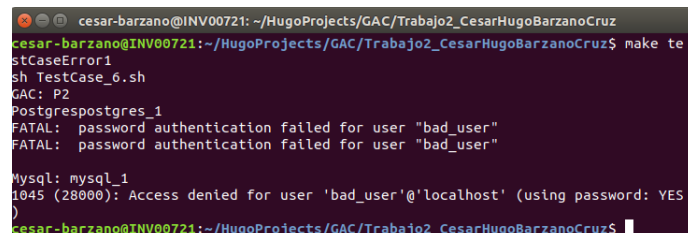
En el primer escenario de error, las credenciales para conectar con el SGBD son erróneas.

Listing 2.17: GAC_GENERATOR_CONFIG_error.json

```

1 {
2     "mysql_1": {
3         "user": "bad_user",
4         "password": "bad_pass",
5         "host": "localhost",
6         "database": "testGAC"
7     },
8     "postgres_1": {
9         "user": "bad_user",
10        "password": "bad_pass",
11        "host": "127.0.0.1",
12        "database": "testgac"
13    }
14 }
```

El primer caso de error puede ser ejecutado mediante `make testCaseError1`. Como podemos observar, el generador continua su ejecución intentando conectarse a los distintos SGBD configurados.



```

cesar-barzano@INV00721: ~/HugoProjects/GAC/Trabajo2_CesarHugoBarzanoCruz
cesar-barzano@INV00721:~/HugoProjects/GAC/Trabajo2_CesarHugoBarzanoCruz$ make testCaseError1
sh TestCase_6.sh
GAC: P2
Postgrespostgres_1
FATAL: password authentication failed for user "bad_user"
FATAL: password authentication failed for user "bad_user"

Mysql: mysql_1
1045 (28000): Access denied for user 'bad_user'@'localhost' (using password: YES)
cesar-barzano@INV00721:~/HugoProjects/GAC/Trabajo2_CesarHugoBarzanoCruz$
```

Figura 2.12: Test Case Error - 1

Test Case Error - 2

Para el segundo escenario de error se va a aplicar la misma configuración y ejecución. La diferencia está en que los servicios locales que representas los SGBD configurados estarán detenidos, de esta manera podemos simular que tanto los host como los SGBD no están disponibles para su conexión.

Los SGBD usados para este escenario pueden ser detenidos mediante:

1. **mysql:** `sudo /etc/init.d/mysql stop`
2. **postgres:** `sudo service postgresql stop`



```
cesar-barzano@INV00721: ~/HugoProjects/GAC/Trabajo2_CesarHugoBarzanoCruz
cesar-barzano@INV00721:~/HugoProjects/GAC/Trabajo2_CesarHugoBarzanoCruz$ make testCaseError1
sh TestCase_6.sh
GAC: P2
Postgrespostgres_1
could not connect to server: Connection refused
        Is the server running on host "127.0.0.1" and accepting
        TCP/IP connections on port 5432?

Mysql: mysql_1
2003: Can't connect to MySQL server on 'localhost:3306' (111 Connection refused)
cesar-barzano@INV00721:~/HugoProjects/GAC/Trabajo2_CesarHugoBarzanoCruz$
```

Figura 2.13: Test Case Error - 2

Capítulo 3

Entrega

En este capítulo se detallan cada uno de los ficheros/directorios que forman parte de la entrega.

3.1. Generador

El generador propuesto por el enunciado se compone de un único fichero denominado `generadorP2.py` el cual se encuentra alojado en la raíz del fichero comprimido que se envía. El generador ha sido desarrollado sobre Ubuntu18 y hace uso de la versión python 2.7.15. Si alguna de ellas causará error, podría ser instalada fácilmente descargando el paquete de las URLs indicadas en la bibliografía y ejecutando `sudo pip install nombre_paquete`

3.2. Makefile

El fichero "makefile" establece las distintas ejecuciones de pruebas para el generador. También dispone de la entrada "make doc" para generar la especificación software mediante pydoc. Alojado en la raíz del fichero comprimido para la entrega.

3.3. Directorio software specification

Contiene la especificación del generador en formato HTML. Dicha especificación ha sido generada utilizando la librería pydoc.

3.4. Trabajo2_CesarHugoBarzanoCruz.pdf

Memoria de la práctica, referencia a este documento en si mismo, alojado en el directorio raíz de la entrega.

3.5. Directorio DOC

Directorio donde se almacenan todos los fuentes usados para generar esta documentación utilizando LaTeX. Incluye tambien las imagenes usadas en la memoria.

3.6. Directorio input

Directorio donde se almacenan todos los ficheros usados como configuracion para el generador.

3.7. Directorio Output

Directorio donde se almacenan todos los ficheros resultados de la ejecución de la pruebas.

Bibliografía

- [1] XML, *XML Format* <https://www.w3.org/XML/>
- [2] JSON, *JSON Format* <https://www.json.org/>
- [3] PYTHON, *Python 2.7.6 language* <https://www.python.org/download/releases/2.7.6/>
- [4] IMPORT JSON, *JSON encoder and decoder* <https://docs.python.org/2/library/json.html>
- [5] IMPORT XMLTODIC, *Makes working with XML feel like you are working with JSON* <https://pypi.python.org/pypi/xmltodict>
- [6] IMPORT SYS, *System-specific parameters and functions* <https://docs.python.org/2/library/sys.html>
- [7] IMPORT OS, *Miscellaneous operating system interfaces* <https://docs.python.org/2/library/os.html>
- [8] IMPORT GETOPT, *C-style parser for command line options* <https://docs.python.org/2/library/getopt.html>

Capítulo 4

Anexo

4.1. generator.py

Listing 4.1: generatorP2.py

```
1 #!/usr/bin/python
2
3
4 __author__ = 'Hugo Barzano'
5 __date__ = '2017/2018'
6 __version__ = 'v1.0'
7 __credits__ = 'GAC'
8 __file__ = 'generatorP2.py'
9
10
11 import sys
12 import os
13 import getopt
14 import time
15 import mysql.connector
16 import json
17 import psycopg2
18 import sqlite3
19 import lxml.etree as et
20 import csv
21 import io
22
23
24 #INPUT and OUTPUT DATA FOLDER
25 OUTPUT_FOLDER="./output/"
26 INPUT_FOLDER="./input/"
27
28 """Generator configuration file: Databases interfaces definition"""
29 #CONFIG=INPUT_FOLDER+"GENERATOR_CONFIG.json"
30 CONFIG=os.environ['GENERATOR_CONFIG']
31 def data2xml(data, name='data'):
32     root = et.Element(name)
33     return et.tostring(buildxml(root, data))
34
35 def buildxml(root, data):
36     if isinstance(data, dict):
37         for key, value in data.iteritems():
```

```

38         sub = et.SubElement(root, key)
39         buildxml(sub, value)
40     elif isinstance(data, tuple) or isinstance(data, list):
41         for value in data:
42             sub = et.SubElement(root, 'i')
43             buildxml(sub, value)
44     elif isinstance(data, basestring):
45         root.text = data
46     else:
47         root.text = str(data)
48     return root
49
50 def buildCsv(data,sgbd):
51     with open(OUTPUT_FOLDER+"GAC_"+sgbd+"_"+time.strftime("%Y%m%d_%H%M%S")+".csv", "w") as output:
52         writer = csv.writer(output, lineterminator='\n')
53         writer.writerows(data)
54     print "Writing output file in"+OUTPUT_FOLDER+"GAC_"+sgbd+"_"+time.strftime("%Y%m%d_%H%M%S")+".csv"
55
56 def buildHtml(data,sgbd):
57     html_table_1="<table style='width:100%'>"
58     html_table_2=""
59     for tupla in data:
60         html_table_2="<tr>"
61         for t in tupla:
62             html_table_2=html_table_2+"<td>"+str(t)+"</td>"
63         html_table_2=html_table_2+"</tr>"
64     html_table=html_table_1+html_table_2+"</table>"
65     html_doc="<!DOCTYPE html>"
66     <html>
67     <head>
68     <title>"+sgbd+"</title>
69     </head>
70     <body>
71     <h1>"+sgbd+"</h1><hr><br>"+html_table+"</body></html> "
72     return html_doc
73
74 def writeOutput(output_file,output_string):
75     """Function to write output files.
76         :param output_file: output file path
77         :param output_string: string to write in the output file"""
78     print "Writing output file in "+output_file
79     try:
80         with open(output_file, 'w') as f:
81             f.write(output_string)
82     except EnvironmentError:
83         print("Oops! Error writing output file...")
84         exit()
85
86 def configurePostgres(config_json):
87     config_str="host='"+config_json["host"]+"', dbname='"+config_json["database"]+"', user='"+config_json["user"]+"', password='"+config_json["password"]+"'"
88     return config_str
89
90 def main(argv):
91     #python generadorP2.py -t gac_models -w "atr_1 = 'atribute_1'"
92     """Main method to execute the generator.
93         :param argv: value from 0 to n-1 to ident"""
94     table=None
95     where=None

```

```

96     debug=False
97
98     # process arguments
99     try:
100         opts, args = getopt.getopt(argv, "hvt:w:d", ["help", "version"
101             , "table=", "where=", "debug"])
102     except getopt.GetoptError as err:
103         print str(err)
104         usage()
105         sys.exit(2)
106     for opt, arg in opts:
107         if opt in ("-h", "--help"):
108             usage()
109             sys.exit()
110         elif opt in ("-v", "--version"):
111             version()
112             sys.exit()
113         elif opt in ("-t", "--table"):
114             table = arg
115         elif opt in ("-w", "--where"):
116             where=arg
117         elif opt in ("-d", "--debug"):
118             debug = True
119             print "Debug: TRUE"
120         else:
121             usage()
122             sys.exit()
123
124     print "GAC: P2"
125     with open(CONFIG) as config_file:
126         SGBD_config = json.load(config_file)
127
128     if where==None:
129         query = ("SELECT * FROM "+table)
130     else:
131         query = ("SELECT * FROM "+table+ " WHERE "+where)
132     connector=None
133     cursor=None
134     data=None
135     for SGBD in SGBD_config:
136         if "mysql" in SGBD:
137             print "Mysql: "+ SGBD
138             try:
139                 connector = mysql.connector.connect(**SGBD_config[SGBD
140                     ])
141             except mysql.connector.Error as err:
142                 print(err)
143                 continue
144             elif "postgres" in SGBD:
145                 print "Postgres"+ SGBD
146                 try:
147                     connector = psycopg2.connect(configurePostgres(
148                         SGBD_config[SGBD]))
149                 except psycopg2.OperationalError as err:
150                     print (err)
151                     continue
152             elif "sqlite3" in SGBD:
153                 print "Sqlite3: "+SGBD_config[SGBD]["db_path"]
154                 try:
155                     connector=sqlite3.connect(SGBD_config[SGBD]["db_path"
156                         ])
157                 except Exception as err:

```

```

154         print "no sql3"
155         print (err)
156         continue
157     else:
158         print SGBD+" Not SGBD supported"
159
160     cursor = connector.cursor()
161     cursor.execute(query)
162     data=cursor.fetchall()
163     if data != None:
164         """XML FILE"""
165         xml_data=data2xml(data2xml(data, name='data'))
166         writeOutput(OUTPUT_FOLDER+"GAC_"+SGBD+"_"+time.strftime("%Y%m%d_%H%M%S")+ ".xml",xml_data)
167
168         """CSV FILE"""
169         buildCsv(data,SGBD)
170
171         """HTML FILE"""
172         jsonObj = json.dumps(data)
173         writeOutput(OUTPUT_FOLDER+"GAC_"+SGBD+"_"+time.strftime("%Y%m%d_%H%M%S")+ ".json",jsonObj)
174
175         """HTML FILE"""
176         html_doc=buildHtml(data,SGBD)
177         writeOutput(OUTPUT_FOLDER+"GAC_"+SGBD+"_"+time.strftime("%Y%m%d_%H%M%S")+ ".html",html_doc)
178     else:
179         print "No data"
180     #isinstance(object, classinfo)
181
182 #
183
184 # Main exec
185 #
186
187 def usage():
188     """Method to display generator usage. """
189     print """
190 Usage: python generador.py [options]
191
192 Options
193 -v, --version          Show the version of this script
194 -h, --help             Show this help.
195 -t <table_name>, --table <table_name>   Input database table
196 -w <"column = 'value'">, --where <"column = 'value'">   Optional Where
197                               Clause
198 -d, --debug            Debug Mode
199 """
200 def version():
201     """Function to display software version"""
202     print "version 1.0"
203
204 if __name__ == "__main__":
205     if len(sys.argv) > 1:
206         main(sys.argv[1:])
207     else:

```

```
208     usage()
209     sys.exit()
```

4.2. makefile

Listing 4.2: makefile