Mumuki Query Learning

Un Runner de SQL para el Proyecto Mumuki

8 de julio de 2017

Trabajo de Inserción Profesional
Tecnicatura Universitaria en Programación

Leandro Di Lorenzo

Coordinador: Ing. Fernando Dodino

Universidad Nacional de Quilmes
Roque Sáenz Peña 352, Bernal
Buenos Aires, Argentina (B1876BXD)
Tel. (+54 11) 4365 7100 | Fax (+54 11) 4365 7101
http://www.unq.edu.ar/ | info@unq.edu.ar

Índice

1. Introducción

Hay nuevas formas de aprender a programar, de enseñar a programar y también nuevas formas de programar. No tiene que ver con gustos or criterios sino con una realidad. La programación como motor de la computación y las comunicaciones cambiaron paradigmáticamente la interacción de las personas y las sociedades a nivel mundial. Hoy existe una fuerte necesidad de instantaneidad en las respuestas. En cierto punto se sabe que la respuesta está a la vuelta *de una googleada*, sólo es necesario indicar las palabras de búsqueda mágicas.

Esta velocidad de la información repercute en las prácticas de la enseñanza formal, la cual necesita moverse por terreno firme antes que veloz. Cada momento didáctico con un grupo de alumnos es una oportunidad única de transmisión de conocimiento; y de la calidad del conocimiento transmitido. Pero el conocimiento no puede ser transmitido sólo por la calidad del docente. El alumno tiene que estar dispuesto a aceptar ese conocimiento.

Aprender programación es un desafío en sí mismo. Es necesario poder manejar herramientas tecnológicas como lenguajes de programación, entornos de desarrollo, compiladores, intérpretes, etc. Pero también conceptos como abstracción, refactorización, parametrización. Los flujos y los protocolos de comunicación, las estructuras de datos: árboles, listas, pilas, colas, etc. Y todo sin dejar de tener que cuenta que es necesario ser ordenado y declarativo en el diseño de programas, pues será leído por otras personas que tienen que poder entenderlo.

Es fundamental entonces organizar estos conceptos y presentarlos de forma que no sea abrumadora para el alumno. Es importante poder minimizar los problemas de una herramienta tecnológica cuando se busca enseñar un concepto o una estructura. Y es importante que el alumno logre sentirse atraído por ese conocimiento.

Gran parte de los profesores y alumnos están muy naturalizado con el uso de ciertas tecnologías (computadoras, celulares, internet, etc.), las cuales fueron construidas por docentes y profesionales de la programación. La próxima generación ya usa activamente las herramientas que se busca enseñarles a construir. Es posible entonces utilizar estas mismas herramientas para enseñar a construirlas.

2. Motivación

La materia *Bases de Datos* contiene mayormente alumnos en su segundo o tercer semestre en la Tecnicatura en Programación dictada en la Universidad Nacional de Quilmes. Idealmente son alumnos que vienen de cursar las materias *Introducción a la Programación* y *Organización de Computadoras*, materias antagónicas desde el *alto* y *bajo nivel* de los conceptos transmitidos. En *Intro* se aprenden conceptos de **alto nivel** usando Gobstones y en *Orga* conceptos de código de máquina utilizando QSIM[1]. Y al mismo tiempo que Bases de Datos muchos cursan *Programación con Objetos I*.

Desde *BBDD* se busca transmitir el concepto de relación de información. Se enseñan conceptos como *Modelo de Entidad-Relacion*, *Normalización* y *Álgebra Relacional*. El puente entre estos conceptos y el mundo de la programación es el **lenguaje SQL**.

Al querer enseñar los conceptos necesarios para que el alumno pueda manejar SQL, aparecen los problemas recurrentes asociados a las tecnologías. Los motores utilizados en la industria como Oracle, Postgres, SQL Server, MySQL, etc., son piezas de software tan robustas como complejas, y pueden ser una demora cuando se quieren trasmitir otras ideas fundacionales.

[1] `http://orga.blog.unq.edu.ar/qsim/`

Mumuki cuenta con soporte para Gobstones [2] [2] y para QSIM[3], y ambos son utilizados en Intro y Orga respectivamente. Además de la ventaja de la familiarización de la tecnología, Mumuki permite incorporar los componentes necesarios para obtener una adaptación a la necesidad puntual.

2.1. Proyecto Mumuki

> "Mumuki es un software educativo para aprender a programar a partir de la resolución de problemas; plantea enseñar conceptos de programación, en un proceso conducido por guías prácticas en las que la teoría surge a medida que se avanza. Esta herramienta se presenta al estudiante como una aplicación Web interactiva, en la que se articulan explicaciones y ejemplos con la opción de que cada uno realice su propia solución y la plataforma la pruebe y corrija instantáneamente, orientando acerca de los aciertos y errores." ([3])

Mumuki es el resultado del trabajo de muchas personas que creen que es necesario plantear nuevas formas de enseñar a programar. Lo que se plantea es un complemento a la clase en el aula, no un reemplazo, pero haciendo foco en que la plataforma pueda ayudar a resolver el mayor número de *problemas técnicos* que el alumno pueda tener con la tecnología y permitir al docente enfocarse en las problemáticas inherentes a la enseñanza de los conceptos.

Es común en cursos introductorios de programación orientada a objetos que los alumnos estén una o dos clases (y el tiempo entre semana) tratando de hacer funcionar el entorno de Java con sus bibliotecas y el IDE, en lugar de aprovechar ese tiempo para practicar conceptos elementales como mensajes entre objetos, comportamiento, herencia, etc.

Al usar una plataforma como Mumuki el docente puede saltear esta problemática y enforcarse en lo conceptual. Por supuesto que todo estudiante de programación debe lograr aprender a instalar y configurar entornos de desarrollo, pero en ciertos niveles esta nueva forma de enseñar evita muchas frustaciones puramente inherentes a la tecnología, las que sino terminan por ser asociadas a la programación como un todo.

2.2. Gobstones como hilo conductor

Si bien este trabajo no se relaciona directamente con Gobstones [4], gran parte de las ideas tienen orígen en las planteadas en el libro «Las bases conceptuales de la Programación: Una nueva forma de aprender a programar» [1], del cual obtenemos su definición:

> "Gobstones es un lenguaje conciso de sintaxis razonablemente simple, orientado a personas que no tienen conocimientos previos en programación. El lenguaje maneja distintos componentes propios, ideados con el fin de aprender a resolver problemas en programación, pero al mismo tiempo intentando volver atractivo el aprendizaje."

Gobstones busca facilitar la enseñanza de conceptos iniciales de programación tratando de mantener distancia de las problemáticas técnicas y la comprensión de errores complejos ante fallos de programas. Cuenta con el entorno *PyGobstones* [5] programado en python pero actualmente se está desarrollando una versión web [6] que no requiere instalación.

[2] `https://github.com/mumuki/mumuki-gobstones-runner`
[3] `https://github.com/mumuki/mumuki-qsim-runner`
[4] `http://www.gobstones.org/`
[5] `http://inpr.web.unq.edu.ar/instalacion-de-pygobstones/`
[6] `https://gobstones.github.io/editor-beta/`

3. Presentación

Para obtener una herramienta que pueda ser utilizadas desde Mumuki es necesario cumplir con ciertas pautas a la hora de interactuar con la plataforma.

3.1. Requerimientos de la Plataforma

La Plataforma Mumuki se puede entender desde estos cuatro componentes:

- **Laboratory:** el entorno web en donde los estudiantes resuelven ejercicios y reciben *feedback*.
- **Classroom:** herramienta para que el docente pueda generar seguimiento de sus alumnos.
- **Bibliotheca:** repositorio de guías y ejercicios.
- **Runners:** componentes que se encargan de ejecutar y verificar los programas enviados por los alumnos.

Existen muchos *Runners* en Mumuki, cada uno se encarga de trabajar sobre una tecnología puntual. Existe un Runner de Javascript, otro de Ruby, de Gobstones, de C++, de Wollok, de QSIM, de Prolog, de Java, etc...

A cada ejercicio (o guía) se le asocia un *Runner*, el cual es el encargado de procesar el ejercicio y retornar un buen feedback. Para que todos ellos puedan convivir deben respetar ciertas reglas de comportamiento. Cada Runner debe utilizar el framework *Mumukit* [7] y establecer los métodos necesarios para que el flujo del proceso de evaluación del ejercicio sea satisfactorio.

Como el *Runner* debe ejecutar código, debe hacerlo en un lugar aislado y seguro. Mumukit espera que cada Runner tenga asociado un *container de Docker*[8].

En la figura 1 se muestra el flujo de interacción desde que el alumno envía la solución de un problema hasta que obtiene la corrección.

Cada *Runner*, al usar Mumukit, provee una interfaz HTTP con 3 rutas posibles

- `GET /info`: retorna metada del runner
- `POST /query`: ejecuta una sentencia en particular
- `POST /test`: ejecuta todo el ejercicio

Cuando el alumno envía la solución, se ejecuta una petición `POST /test` con todo el ejercicio completo (la solución del alumno y lo preestablecido por el docente). Mumukit recibe la petición y ejecuta al *Runner* correspondiente, el cual debe cumplir con ciertos métodos. El *Runner* adapta el ejercicio e interactúa con *Docker* para ejecutar el ejercicio y poder responder con la evaluación.

[7] `https://github.com/mumuki/mumukit`
[8] `https://www.docker.com/what-docker`

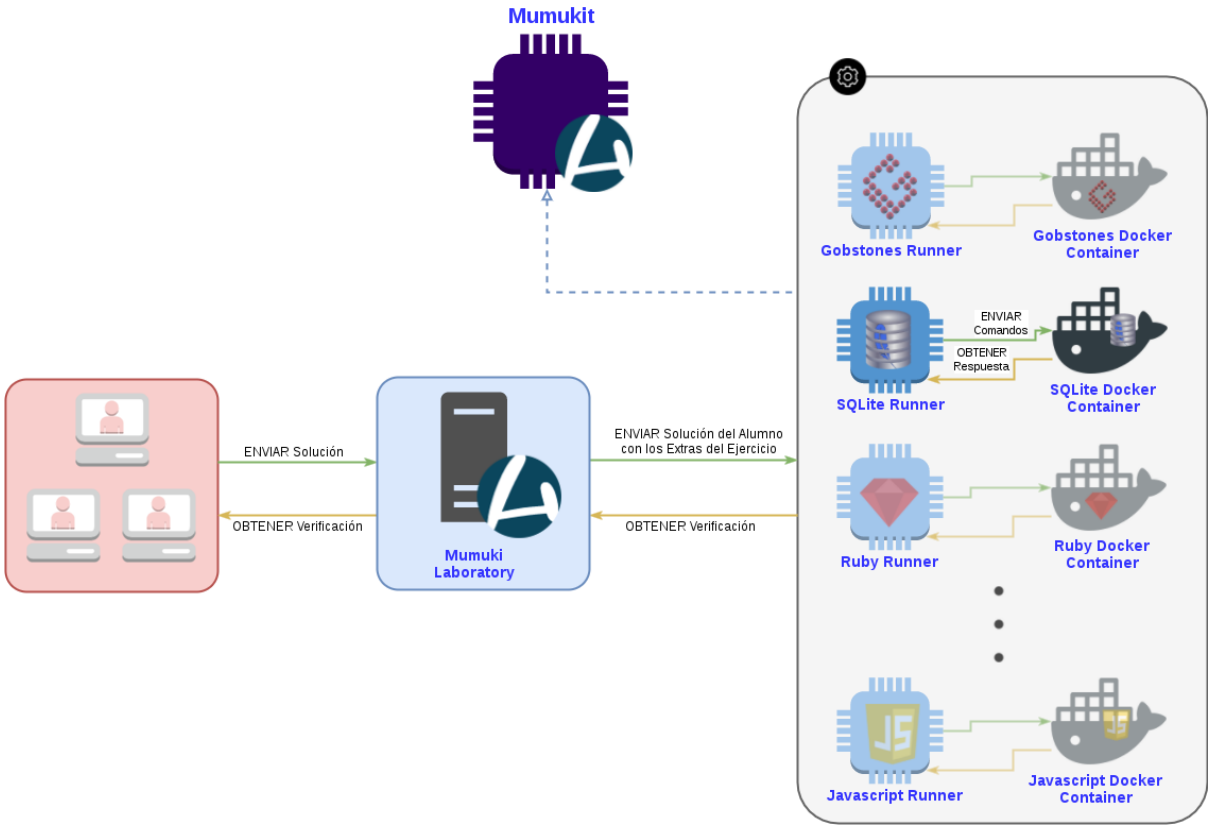

Figura 1: Arquitectura MQL

3.2. Trabajo Realizado y Stack Tecnológico

La elección del stack estuvo mayormente definida por las necesidades de Mumukit. El *Runner* debe está escrito en *ruby* y se creó un *container Docker* con un motor SQL instalado. Se optó por utilizar *SQLite* como motor SQL por ser mucho más veloz que los motores utilizados en la industria. Se optó por una herramienta versatil que permita rápidamente concentrarse en los conceptos.

Docker funciona creando y destruyendo máquinas virtuales de forma muy rápida y económica. Este aspecto es muy útil ya que permite garantizar que cada solución enviada por cada alumno a un mismo *Runner* sea ejecutada de forma aislada y en un ambiente limpio.

Esta interfaz permite que el docente pueda generar de manera *ad-hoc* y personalizable a cada ejercicio la configuración de base para que un alumno pueda trabajar enfocándose en el aprendizaje del concepto.

Las peticiones entre el *Runner* y el *worker* se dan mediante datos con formato *json*. Mumukit permite configurar que la petición hacia el *worker* sea mediante un comando que recibe un archivo como parámetro. El *Runner SQLite* transforma el ejercicio en una estructura *json* como se observe en el listing 1.

El *worker* utiliza un script en *python* para ejecutar el código del alumno y el del docente en el DDL planteado para cada set de datos. La respuesta esperada por *Mumukit* es un `buffer` de datos en el `stdout` junto al `exit code`.

Cuando la ejecución no contiene errores de sintaxis y puede ser procesada por el motor,

```
{
    "init": "CREATE TABLE ...; CREATE ...;",
    "solution": "SELECT ... FROM ...;",
    "student": "SELECT * FROM table;",
    "datasets": ["INSERT INTO ....; INSERT INTO ...", "INSERT ....;"]
}
```

Listing 1: **JSON** enviado al *worker*

genera un *dump* en formato *json* con las claves `solutions` y `results`, que contienen los *rows* obtenidos como resultado de ejecutar cada respectiva *query* en cada set de datos provisto, tal como se muestra en el listing 2. Esos resultados son procesados por el *Runner* que evalúa a posteriori la correctitud de la solución del alumno y retorna un *feedback* acorde.

```
{
    "solutions": [
        "name\nTest 1.1\nTest 1.2\nTest 1.3",
        "name\nTest 2.1\nTest 2.2\nTest 2.3",

    ],
    "results": [
        "id|name\n1|Test 1.1\n2|Test 1.2\n3|Test 1.3",
        "id|name\n1|Test 2.1\n2|Test 2.2\n3|Test 2.3",
    ]
}
```

Listing 2: **JSON** devuelto por *worker* en caso de éxito

Figura 2: Solución incorrecta

3.3. Retrospectiva

Este trabajo se realizó mediante la modalidad *Taller de Trabajo de Inserción Profesional*. Esta modalidad tiene un semestre de tiempo de vida y es coordinada por un profesor de la

carrera. El siguiente es un detalle de las entregas acordadas como plan de trabajo a lo largo de la cursada.

3.3.1. Presentación del Proyecto

Para la presentación del Proyecto MQL se generó cierto trabajo de base que consistió en definir un backlog de tareas, priorizarlas y estimarlas, generar el repositorio con el stack tecnológico funcionando y hacer la presentación.

3.3.2. Prueba de concepto

- Investigación sobre Docker y armado inicial del *worker*
- Integración con Travis CI
- Integración con CodeClimate
- Primer borrador de este documento
- Ejercicio base con sintaxis inválida, detectando el fallo y mostrando la respuesta correcta.
- Ejercicio base correcto, verificando que los resultados retornados sean los deseados

3.3.3. Entrega 1

- Modificación del script del *worker* agregando soporte para ambas soluciones: la del alumno y la provista por el docente.
- Investigación sobre el funcionanmiento de los *Hooks* de Mumukit.
- Modificación de ejercicios: se establece que el código inicial sea el obtenido de la variable asociada al campo `extra_code` utilizado por el docente para la configuración inicial.

3.3.4. Entrega 2

- Refactorización de *worker* y *Runner*. Se mejora la comunicación y se establecen mejores condiciones para el postprocesamiento de la solución.
- Se generan fixtures para tests de verificación correcta.
- Se generan fixtures para tests de casos de error.
- Generación de Test de Integración con la plataforma.
- Instalación de Mumuki Laboratory en forma local.
- Generación de un Capítulo SQL con ejercicios en Mumuki Laboratory local

3.3.5. Entrega 3

- Refactorización del *worker*. Se cambia la estructura de datos por formato *json* en ambos sentidos de las peticiones. Se establecen mejoras en el procesamiento de datos.
- Refactorización del *Runner* para que utilice el *Checker* de *Mumukit*. Esto permite desacoplar y delegar el análisis y la correción a otro componente que cumple esa única tarea.
- Refactorización completa de la comparación de los resultados del *worker*.
- Codificación del *Render HTML* para un *feedback* adecuado.
- Definición de un ejercicio con mayor complejidad y múltiples datasets.

3.3.6. Entrega 4

- Coloreo de sintaxis.
- Pull Request y correcta integración con Proyecto Mumuki en producción.
- Corrección de error que no permitía el encoding correcto.
- Corrección de error que no permitía que un resultado sea vacío.
- Refactorización de la estructura utilizada por docente al cargar un ejercicio. Se establece formato YAML para brindar mayor flexibilidad en el ejercicio.
- Se agrega al formato YAML la posibilidad de establecer el dataset esperado en lugar de la query dada como solución.
- Se agrega parseo mediante `diff`.
- Refactorización *Render HTML* para colorear el `diff`.
- Versión final de este documento.
- Armado de presentación y demo.

4. Alcance y Trabajo futuro

Inicialmente este trabajo busca lograr establecer guías de estudio de SQL como parte de los contenido de la materia *Bases de Datos*. Pero al ser parte de un proyecto *Open Source*, se espera que pueda servir como mejora en todo ámbito que sea posible la enseñanza de SQL y de Bases de Datos Relacionales.

Al mismo tiempo queda planteado el desafío de lograr herramientas que cubran una superficie mayor de los conceptos de Bases de Datos. Un *Runner de Álgebra Relacional* sería un complemento ideal. Al ser *SQLite* poco estricto con algunas restricciones de datos, podría ser útil contrar con un *Runner* que permita establecer **transacciones** y eventualmente interrumpirlas para analizar comportamiento. Las Bases de Datos No Relaciones son un buen desafío también para enseñar *mongoDB*, *Neo4j*, etc.

5. Conclusiones

El trabajo produjo desafíos en varios aspectos. El hecho de trabajar en un proyecto *Open Source* establece de por sí el obecer ciertas prácticas de desarrollo para lograr convivir armoniosamienta con otras colaboraciones. A la vez que era imperioso hacer primar las necesidades de la cátedra en cuanto a la flexibilidad de los ejercicios.

Se logró aprender sobre una nueva tecnología como Docker, la cual plantea nuevas formar de desarrollar software a través del *microservicios*.

La adaptación a las reglas establecidad por *Mumukit* fue también una tarea a destacar. La flexibilidad y la potencia de la *metaprogramación* brindada por ruby trae como contrapartida la dificultad en la correcta comprensión de los objetos y en el segumiento del flujo del programa.

Este trabajo, como cierre de carrera, busca también ser un aporte a la materia Bases de Datos, a la Universidad de Nacional de Quilmes y a toda la comunidad de Programación.

Referencias

[1] Pablo E. Martínez López. *Las bases conceptuales de la Programación: Una nueva forma de aprender a programar*. EBook, La Plata, Buenos Aires, Argentina, 2013. ISBN: 978-987-33-4081-9. URL: http://www.gobstones.org/bibliografia/Libros/BasesConceptualesProg.pdf.

[2] Federico Aloi. «Implementación de un runner de Gobstones para la plataforma Mumuki». En: (2015). URL: https://github.com/faloi/tip-mumuki-gobstones/releases/download/v1.1/TIP-Aloi-RunnerGobstonesMumuki.pdf.

[3] Federico Aloi, Franco Bulgarelli y Lucas Spigariol. «Mumuki, una plataforma libre para aprender a programar». En: (2015). URL: https://www.academia.edu/25374997/Mumuki_una_plataforma_libre_para_aprender_a_programar.

Apéndices

A. Ejemplo un ejercicio con solución vía *Query*

```sql
CREATE TABLE motores (
    id integer primary key,
    nombre varchar(200) NOT NULL
);
```

Listing 3: **Extra Code:** SQL

```yaml
solution_type: "query"
solution_query: "select * from motores";
examples:
    - data: |
        INSERT INTO motores (nombre) values ('MySQL');
        INSERT INTO motores (nombre) values ('PostgreSQL');
        INSERT INTO motores (nombre) values ('Oracle');
        INSERT INTO motores (nombre) values ('Microsoft SQL Server');
        INSERT INTO motores (nombre) values ('SQLite');
```

Listing 4: **Test Code:** YAML

B. Ejemplo un ejercicio con solución vía *Datasets*

```
CREATE TABLE `bolitas` (
    id integer primary key,
    color varchar(200) NOT NULL
);
```

Listing 5: **Extra Code:** SQL

```
solution_type: "datasets"
examples:
-   data: |
        INSERT INTO bolitas (color) values ('Verde');
        INSERT INTO bolitas (color) values ('Rojo');
        INSERT INTO bolitas (color) values ('Azul');
        INSERT INTO bolitas (color) values ('Negro');
    solution_dataset: |
        id|color
        1|Verde
        2|Rojo
        3|Azul
        4|Negro
-   data: |
        INSERT INTO bolitas (color) values ('Violeta');
        INSERT INTO bolitas (color) values ('Amarillo');
    solution_dataset: |
        id|color
        2|Violeta
        3|Amarillo
```

Listing 6: **Test Code:** YAML