

QuickDB

Autores:

Diego Sarmentero

e-mail: diego.sarmentero@gmail.com

Emilio Ramirez

e-mail: emilioramirez04@gmail.com

Universidad Tecnológica Nacional

Facultad Regional Córdoba

Abstract

"QuickDB es un framework de persistencia que permite al desarrollador escribir sólo el modelo de datos, y el framework se encargara de todas las operaciones entre las entidades y la base de datos, sin tener que escribir ni una línea de código para conexión, SQL, etc. Esta herramienta de Mapeo Objeto-Relacional pretende no solo simplificar las tareas de mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, sino a su vez, lograr que el uso de la librería sea totalmente natural para el desarrollador, dejando de lado tediosas tareas de configuración. Donde cada operación implica sólo lo que se quiere hacer (guardar, modificar, etc) y a qué se lo va a aplicar (el objeto)."

Palabras Clave: quickdb, framework, persistencia, orm, librería, reflexión, bases de datos

1. Introducción

QuickDB, como otras herramientas de mapeo objeto-relacional, busca solucionar las diferencias entre los dos modelos de datos coexistentes: Modelo Orientado a Objetos y el Modelo Relacional. Para hacer frente a este problema, QuickDB toma un enfoque completamente orientado a objetos, donde estructuras como "objetos compuestos por otros objetos", "herencia", "colecciones" (relaciones uno-a-muchos y muchos-a-muchos) son soportadas por defecto como entidades comunes, y se agregan otras características como creación de tablas automáticamente y modificación de las tablas de forma dinámica si la estructura del Objeto cambiara con el tiempo (inclusión de nuevos atributos). QuickDB no requiere la implementación de ninguna interfaz, ni la utilización de herencia por parte del modelo de datos

a persistir, simplemente se basa en ciertas convenciones del nombrado de atributos para inferir la información relevante sobre el objeto. No obstante, es posible la utilización de Anotaciones para explicitar determinadas características del objeto, lo cual brinda la seguridad de que todo lo que QuickDB reconoce por defecto, puede ser a su vez, completamente controlado por el desarrollador. Para realizar consultas, QuickDB pretende mantener este enfoque donde el desarrollador trabaja con su modelo de datos de forma totalmente orientada a objetos, y es por ello, que las sentencias SQL (aunque están permitidas) no son necesarias, y se puede utilizar por defecto un sistema de Query donde se especifica la condición a evaluar haciendo referencia simplemente a los atributos de los objetos en el modelo de datos.

QuickDB esta basado principalmente en la idea de que una aplicación o librería altamente funcional no tiene porque ser sinónimo de una alta complejidad, creando de esta forma los algoritmos apropiados para poder eliminar esta brecha entre el modelo Orientado a Objetos y el modelo Relacional, sin la necesidad de que el usuario deba invertir tiempo en tediosas tareas de configuración o aprendiendo a manejar la librería, ya que la curva de aprendizaje de la misma es muy rápida gracias al diseño con el que fue desarrollada para ser intuitiva ante cualquier desarrollador.

2. Elementos del Trabajo y Metodología

Este proyecto aborda el problema de los distintos modelos de datos existentes centrándose en un enfoque que permita a los usuarios de esta librería ser capaces de expresar todas las actividades del manejo de datos a través de un único lenguaje.

El problema tradicional al trabajar con Bases de Datos se plantea al tener que convertir los Objetos con los

que se trabaja a un formato que sea compatible con el DBMS siendo empleado u obtener datos desde la Base de Datos para manipularlos muchas veces como colecciones de arrays o correr determinadas rutinas que permitan convertir esos datos a Objetos propios del modelo del programa que se esta utilizando. Para un análisis más profundo de estas problemáticas, se plantearan los siguientes casos:

Convertir Objetos desde un Modelo Orientado a Objetos a un Modelo Relacional:

En este caso se plantean determinados conflictos como ser:

- Los tipos de datos del Lenguaje son distintos a los tipos de datos de la Base de Datos.
- El código total de la aplicación se encuentra fragmentado, es decir, al hacer alguna modificación en el tipo o cantidad de datos que se desean persistir, suele ser necesario modificar el Modelo de Datos de la aplicación, las rutinas encargadas de enviar esos valores a la Base de Datos, modificar las Tablas en la Base de Datos, etc.
- Al tener que mantener en sincronía una aplicación fragmentada en cuanto a lo que se refiere a su proceso de persistencia, esta aplicación es mas propensa a errores al ser necesarias modificaciones en el modelo de datos, ya que estas distintas áreas encargadas de lograr la persistencia de datos de la aplicación pueden ser mantenidas por distintas personas, presentarse un error humano al tener que realizar tediosas modificaciones repetitivas en distintas partes del programa, siendo posible olvidar de modificar alguna de estas áreas, o por el contrario realizar modificaciones defectuosas y luego tener que invertir tiempo buscando la solución del error al presentarse la incompatibilidad.
- Los desarrolladores deben ser conscientes de los 2 modelos de datos, operar con uno y otro para lograr los objetivos de la aplicación y hacerlos coexistir como uno privándose de centrarse en un enfoque completamente orientado a objetos.

Convertir datos desde un Modelo Relacional a un Modelo Orientado a Objetos:

Los conflictos planteados al obtener datos desde la Base de Datos suelen ser los siguientes:

- El formato en que los datos desde la Base de Datos son recibidos suelen ser representados en forma de Tabla, las distintas APIs que brindan los DBMS para acceder a estos datos permiten explorar estas tablas accediendo a los valores como si los datos fueran leídos a través de un puntero que se mueve entre las filas y columnas, esto obliga al programador a conocer la estructura en que esa Tabla (o Matriz) sera obtenida desde la Base de Datos para poder leer el dato específico que se requiera.
- Al trabajar con esta representación en Tabla, se pierde la noción de Objeto como una entidad que puede operar sobre sus atributos para lograr un resultado deseado, y por el contrario, es necesario operar campo por campo con los valores de la Tabla.
- Al no estar contenidos en el formato del modelo de datos definido por la aplicación los valores retornados por la Base de Datos, es necesario incurrir en la escritura adicional de código para manejar estos datos si es necesario que los mismos sean comunicados a distintas partes de la aplicación.

2.1 Análisis de la Problemática

Teniendo en cuenta los conflictos analizados anteriormente que surgen al tratar de convivir con los 2 modelos de datos planteados, es que se comenzó a utilizar esta técnica de programación denominada ORM, en donde se intenta crear un nexo entre ambos modelos y se procura interactuar con uno y otro de forma indistinta.

Un ORM, cuyas siglas significan Object-Relational Mapping (Mapeo Objeto-Relacional), consiste en un software capaz de interpretar tanto los datos del modelo orientado a objetos y del modelo relacional y ser capaz de realizar las transformaciones o traducciones necesarias para brindarle a cada modelo los datos en el respectivo formato que sea soportado por cada uno. De esta forma, la utilización de un ORM reduce de gran manera el código que debe ser generado por el programador reduciendo el tiempo de desarrollo en un 30% aproximadamente.

Ejemplificando el problema:

En la programación orientada a objetos, las tareas de manejo de datos son implementadas generalmente por la manipulación de objetos, los cuales son casi siempre valores no escalares. Para ilustrarlo, considere el ejemplo de una entrada en una libreta de direcciones, que representa a una sola persona con

cero o más números telefónicos y cero o más direcciones. En una implementación orientada a objetos, esto puede ser modelado por un "objeto persona" con "campos" que almacenan los datos de dicha entrada: el nombre de la persona, una lista de números telefónicos y una lista de direcciones. La lista de números telefónicos estaría compuesta por "objetos de números telefónicos" y así sucesivamente. La entrada de la libreta de direcciones es tratada como un valor único por el lenguaje de programación (puede ser referenciada por una sola variable, una instancia). Varios métodos pueden asociarse con el objeto, como uno que devuelva el número telefónico preferido, la dirección de su casa, etc.

Sin embargo, muchos productos populares de base de datos, como los productos SQL DBMS, solamente puede almacenar y manipular valores escalares como enteros y cadenas, organizados en tablas.

El programador debe convertir los valores de los objetos en grupos de valores simples para almacenarlos en la base de datos (y volverlos a convertir luego de recuperarlos de la base de datos), o solo usar valores escalares simples en el programa. El mapeo relacional de objetos es utilizado para implementar esta primera aproximación.

El viacrucis del problema reside en traducir estos objetos a formas en las cuales puedan ser almacenadas en la base de datos, y los cuáles pueden ser recuperados más tarde fácilmente, mientras se preserven las propiedades de los objetos y sus relaciones; estos objetos se dice entonces que son persistentes.

2.1.1 Ventajas ORM

- Rapidez en el desarrollo. La mayoría de las herramientas actuales permiten la creación del modelo por medio del esquema de la base de datos, leyendo el esquema, nos crea el modelo adecuado.
- Abstracción de la base de datos. Al utilizar un sistema ORM, lo que conseguimos es separarnos totalmente del sistema de Base de datos que utilizamos, y así si en un futuro debemos de cambiar de motor de bases de datos, tendremos la seguridad de que este cambio no nos afectará a nuestro sistema, siendo el cambio mas sencillo.
- Reutilización. Nos permite utilizar los métodos de un objeto de datos desde distintas zonas de la aplicación, incluso desde aplicaciones distintas.
- Seguridad. Los ORM suelen implementar sistemas para evitar tipos de ataques como pueden ser los SQL injections.

- Mantenimiento del código. Nos facilita el mantenimiento del código debido a la correcta ordenación de la capa de datos, haciendo que el mantenimiento del código sea mucho mas sencillo.
- Lenguaje propio para realizar las consultas. Estos sistemas de mapeo traen su propio lenguaje para hacer las consultas, lo que hace que los usuarios dejen de utilizar la sentencias SQL para que pasen a utilizar el lenguaje propio de cada herramienta.

2.1.2 Desventajas ORM Tradicionales

- Tiempo utilizado en el aprendizaje. Este tipo de herramientas suelen ser complejas por lo que su correcta utilización lleva un tiempo que hay que emplear en ver el funcionamiento correcto y ver todo el partido que se le puede sacar.
- Suelen requerir complejas configuraciones, siendo necesario a veces la intervención de una persona experta que sepa plasmar el modelo en las configuraciones de la herramienta.
- Aplicaciones algo mas lentas. Esto es debido a que todas las consultas que se hagan sobre la base de datos, el sistema primero deberá de transformarlas al lenguaje propio de la herramienta, luego leer los registros y por último crear los objetos.

2.1.3 Enfoque QuickDB

QuickDB surge de la necesidad de cubrir las 2 trabas fundamentales en el uso de los ORM:

- Tiempo utilizado en el aprendizaje.
- Complejas y tediosas configuraciones.

El objetivo de la aparición de los ORMs fue justamente intentar facilitar y agilizar el trabajo del desarrollador, pero estas herramientas para realizar el mapeo entre el modelo orientado a objetos y el modelo relacional suelen depender de que el usuario cree configuraciones adicionales para explicitar de una forma más simple como estos mapeos entre los 2 modelos va a llevarse a cabo. De esta forma, las herramientas actuales de ORM simplifican bastante la tarea del desarrollador en lo relacionado a la capa de persistencia y a la vez se aseguran de centralizar el manejo de la capa de una forma más centralizada, no obstante al comenzar a trabajar con cualquiera de estas herramientas se debe invertir mucho tiempo para comprender como configurar y utilizar la misma, ya que la curva de aprendizaje de estas herramientas suele ser bastante grande y a medida que la estructura del modelo de datos en el programa se va haciendo más complejo se empiezan a requerir configuraciones

más y más complejas de la herramienta. Por estos motivos principalmente, es que se comenzó con el desarrollo de QuickDB, donde desde el primer momento se asume que el modelo de datos es Orientado a Objetos y el desarrollador desea trabajar de forma directa con ese paradigma sin la necesidad de comprender o invertir tiempo en preocuparse como sus objetos van a convertirse al modelo relacional.

Los algoritmos desarrollados para QuickDB eliminan la necesidad de toda configuración en lo relacionado a la Base de Datos gracias a su habilidad para *comprender* los Objetos del modelo de datos, de esta forma, el desarrollador posee un modelo de datos independiente de toda librería externa y puede operar con las estructuras de datos más complejas que se puedan generar de una forma totalmente natural. Para QuickDB las unidades de trabajo son realmente los Objetos, es por ello que Objetos compuestos de otros Objetos, Objetos con Herencia, Objetos compuestos por Colecciones, o cualquier combinación de estos, son reconocidos por defecto como elementos totalmente validos sin tener que explicitar los tipos de los atributos o ninguna de estas características.

El enfoque planteado por el proyecto, consiste en ser capaz de detectar, por ejemplo, cuando un Objeto posee una referencia a otro Objeto, para reconocer que ese segundo Objeto sera tratado en el modelo relacional como otra tabla distinta, y la Clave Primaria de esta tabla corresponderá a una Clave Foránea en la Tabla del primer Objeto con el que se estaba trabajando. Básicamente el aporte que hace QuickDB al mundo de los ORMs, esta vinculado a su capacidad de poder entender realmente la estructura de los Objetos propios del lenguaje y permitirle al desarrollador trabajar a este nivel sin necesidad de conocer cual es el DBMS que esta funcionando por debajo.

Por ejemplo, la siguiente Estructura de Datos:



Figura 1: Ejemplo Modelo de Datos

La cual solo consiste en la definición de las clases, con sus respectivos atributos y sus métodos de "get" y "set" siguiendo las convenciones propias del lenguajes (en este caso Java), para ser persistida utilizando QuickDB solo seria necesario crear una instancia del Objeto a persistir completando los atributos que se deseen y utilizar el *Administrador* de QuickDB para

llevar a cabo el proceso de almacenado. El código resultante para el caso propuesto seria el siguiente:

```

//Crea instancia de AdminBase para MySQL
AdminBase admin = new AdminBase(AdminBase.DATABASE.MYSQL, "localhost",
"3306", "exampleQuickDB", "root", "");

//Crea Objeto Address
Address a = new Address();
a.setNumber(123);
a.setStreet("unnamed street");

//Crea Colección de Objetos Phone
Phone p1 = new Phone();
p1.setAreaCode("351");
p1.setNumber("123456");
Phone p2 = new Phone();
p2.setAreaCode("351");
p2.setNumber("4567890");
ArrayList<Phone> phones = new ArrayList<Phone>();
phones.add(p1);
phones.add(p2);

//Crea Objeto Employee
Employee e = new Employee();
e.setCode(555);
e.setRolDescription("play ping pong");
e.setName("Diego Sarmentero");
e.setBirth(new java.sql.Date(100, 4, 20));
e.setAddress(a);
e.setPhone(phones);

```

Figura 2: Código instanciación de AdminBase y Modelo de Datos

Una vez creado el Objeto con el que se desea operar y una Instancia de AdminBase, el cual es el Administrador de operaciones de QuickDB, solo es necesario decirle al Administrador la operación que se desea realizar y cual Objeto se vera involucrado en dicha operación:

```

//Si la tabla no existe, se crea automaticamente
admin.save(e); //Salvar Employee

```

Figura 3: Almacenar Objeto en Base de Datos

De esta forma QuickDB se encargara de procesar el Objeto, entender la Estructura de Datos que representa, y mapear esa Estructura de Datos a una representación que sea entendible por el modelo relacional. Otra característica de QuickDB, es que no precisa de que las Tablas para los respectivos Objetos del modelo de datos se encuentren creadas previas a una inserción, si al intentar realizar una inserción de datos en la Base de Datos descubre que la Tabla donde deberían guardarse los datos no existe, esta es creada de forma automática, ya que QuickDB al conocer la estructura del Objeto posee todos los datos necesarios para determinar como debería ser esta Tabla.

2.2 Standarización de APIs de Acceso a Datos

Se han definido APIs genéricas para fomentar que los Drivers (Conectores, Módulos, etc. Su nombre cambia según el lenguaje de programación) creados para dar soporte a los distintos DBMS que van a interactuar con un determinado lenguaje sigan ciertos

lineamientos que faciliten la integración con el mismo y se adapten a una estandarización.

Básicamente se hace uso de ODBC (Open DataBase Connectivity), el cual es un estándar de acceso a Bases de datos, el objetivo de ODBC es hacer posible el acceder a cualquier dato desde cualquier aplicación, sin importar qué Sistema Gestor de Bases de Datos (DBMS por sus siglas en inglés) almacene los datos, ODBC logra esto al insertar una capa intermedia llamada manejador de Bases de Datos, entre la aplicación y el DBMS, el propósito de esta capa es traducir las consultas de datos de la aplicación en comandos que el DBMS entienda. Para que esto funcione tanto la aplicación como el DBMS deben ser compatibles con ODBC, esto es que la aplicación debe ser capaz de producir comandos ODBC y el DBMS debe ser capaz de responder a ellos. De esta forma, QuickDB hace uso de este estándar de acceso a Bases de Datos y generaliza su capa de interacción con el DBMS, permitiendo a su vez que el desarrollador solo deba depender de QuickDB y la librería que implemente el Driver del determinado DBMS con el que desee trabajar, reduciendo el tamaño de la aplicación y evitando tener que incluir en QuickDB librerías externas que posiblemente no sean necesarias, y esta característica se logra debido a la posibilidad de cargar dinámicamente el Driver que sea apropiado según el DBMS elegido por el usuario.

A pesar de las APIs o los estándares, cada Base de Datos suele poseer sus propias funciones para manejo de fecha, variaciones de SQL para la creación de tablas, etc. A través de la utilización de QuickDB estos temas dejan de ser de preocupación, ya que QuickDB se encarga de adaptar cada una de estas operaciones para que funcionen con la Base de Datos adecuada haciendo que el uso de estas operaciones sea totalmente transparente al usuario.

2.3 Introspección del Modelo

La reflexión, la cual es la propiedad que permite hacer introspección de los componentes en tiempo de ejecución, es uno de los pilares fundamentales de QuickDB, y es en lo que se basa para brindar la facilidad de uso inherente que posee.

QuickDB no busca solo permitir realizar el mapeo objeto-relacional, sino que además esta actividad sea de una gran sencillez para el programador, reduciendo lo más posible el tiempo que se suele invertir para definir como se interactuara con la BD, y esta característica es indudablemente posible gracias a la reflexión.

La facilidad de uso de QuickDB nace de su capacidad para obtener toda la información relevante de las

estructuras de datos que se pretenden persistir, sin que para ello el programador deba invertir tiempo en tediosas configuraciones.

Al trabajar con Reflexión en QuickDB, las siguientes operaciones son las que toman mayor importancia:

- Obtener el nombre de la Clase: de esta forma se puede conocer el nombre de la Tabla a la que se mapeara el objeto.
- Obtener instancia de la Clase Padre: obtener la instancia de la clase padre del objeto que se intenta persistir es de suma importancia cuando el modelo de datos esta planteado de esa forma, para representar la estructura de datos de la forma más precisa posible. Además la reflexión suele permitir solo acceder a los datos del objeto con el que se esta trabajando, y algunas veces se pierden los datos heredados si no se accede a una instancia del padre. A su vez, es de suma importancia saber identificar cuando acceder a los datos de la Clase Padre y cuando no, ya que la Clase del modelo de datos puede estar heredando de alguna otra Clase, pero no por ello significar que esa Clase Padre debe ser persistida también (el ejemplo más simple es darse cuenta que si no se establecen ciertas condiciones, se podría llegar a persistir hasta la Clase Object, en el caso de Java)
- Obtener Anotaciones/Atributos/Decoradores (el nombre varia según el lenguaje): básicamente son metadatos que agregamos a la Clase y que creando las "Anotaciones" de la forma correcta, esta información persiste aun en tiempo de ejecución y puede ser accedida por QuickDB para los casos en que el desarrollador desea exponer alguna información adicional acerca de algún Objeto.
- Obtener el tipo de dato de los atributos: esta operación es de importancia cuando se intenta crear una Tabla en la BD que represente precisamente al objeto, o conocer más en profundidad la estructura del Objeto con el que se esta trabajando, ya que los tipos datos nativos del lenguaje se suelen mapear directamente a los tipos de datos de la Base de Datos, pero analizando el tipo de dato de los atributos del Objeto podríamos descubrir que cierto determinado atributo no es un tipo de dato nativo, sino que es una colección o una referencia a otro Objeto, por lo que requeriría un tratamiento especial.
- Obtener los Valores de los Atributos: Obviamente al ser la funcionalidad principal persistir los datos de los objetos, debe haber alguna forma de poder obtener el valor de las

variables que componen el Objeto para volcarlos en la Base de Datos.

- Colocar Valores a los Atributos: Así como en el punto anterior se habla de la importancia de poder obtener el valor de los atributos para volcarlos en la Base de Datos, es de igual importancia que los datos obtenidos desde la Base de Datos puedan ser insertados en un Objeto para restablecer su estado.

Ni el componente de acceso a los datos, ni la Base de Datos entiende las operaciones a nivel de objetos, es por ello que QuickDB nace a cubrir esa necesidad.

Para ello QuickDB al trabajar con los objetos utiliza la potencia de la Reflexión para hacer introspección del Objeto y poder crear una representación del mismo que sea utilizable por los distintos algoritmos de QuickDB para enviar y recibir datos desde y hacia el motor de persistencia.

2.4 Independencia del Modelo de Datos

A diferencia de los ORMs más conocidos actualmente, QuickDB no requiere depender de ningún archivo de configuración, ni de que el desarrollador adapte su Modelo de Datos para que este extienda de alguna Clase de QuickDB o implemente determinada interfaz, por el contrario, un Modelo de Datos escrito puramente en el lenguaje que se esta utilizando es más que suficiente para poder interactuar con QuickDB y que este se encargue de orquestar todas las operaciones entre los Objetos y la Base de Datos.

Para soportar esta independencia entre el Modelo de Datos y QuickDB solo se requiere que el código escrito por el desarrollador siga las convenciones propias del lenguaje con el que esta trabajando y de esta forma QuickDB sera capaz de navegar entre los Objetos y crear las representaciones apropiadas de los mismos. Esta habilidad de QuickDB para trabajar con convenciones es lo que posibilita eliminar toda tarea de configuración innecesaria.

Las convenciones se utilizan para eliminar las tareas de configuración dentro de QuickDB, hay muchas propiedades que pueden ser deducidas automáticamente sin necesidad de configuración, y a continuación se explicarán cuales son los detalles a tener en cuenta en el Modelo de Datos para que sea soportado por QuickDB sin necesidad de realizar mas que la escritura de las Entidades. Estas convenciones son necesarias al no trabajar con Anotaciones (el cual es un recurso de QuickDB que se analizara en breve), de lo contrario, pueden escribirse las Entidades de cualquier otra forma siempre que se especifiquen dichas características mediante las anotaciones.

- Para claves primarias: debe existir una variable entera de nombre "id" declarada como primer atributo de la Clase.
- Para Herencia: Para que QuickDB pueda reconocer la herencia automáticamente, es necesario que la Clase de la que se extienda se encuentre dentro del mismo paquete que la Clase hija, esto posibilita poder determinar cuando dejar de analizar la herencia de forma recursiva y no llegar hasta el nivel de Object.
- Para las Colecciones: Al trabajar con colecciones, solo son soportadas aquellas que implementan la interfaz "java.util.List" o "java.util.Collection" (en el caso de Java), y cuando se trabaja sin anotaciones además el nombre del atributo debe ser igual al nombre de la clase de los objetos que compondrán dicha colección (con la primer letra en minúscula si se desea).
- Getters y Setters: Al no trabajar con anotaciones, QuickDB debe determinar automáticamente cuales serán los métodos desde los que se obtendrán los valores de los atributos, y cuales serán los métodos a través de los cuales se le podrán dar valores a los atributos. Para hacer esto se sigue la convención por defecto de escribir "get" o "set" y luego el nombre del atributo con CamelCase.

Obviamente se reconoce que en determinados casos el desarrollador podría llegar a querer forzar cierta configuración especifica sobre su modelo de datos, de esta forma, todo lo que QuickDB reconoce por defecto puede a la vez ser configurado por el desarrollador, teniendo cada una de estas configuraciones valores por defecto, lo cual significa que al recurrir al uso de configuraciones tampoco se debe caer en largas y complejas configuraciones, sino que solo sera necesario especificar los atributos que se deseen manipular. Otra característica de QuickDB, es que estas configuraciones se realizan agregando metadatos sobre la Clase escrita por el desarrollador, centralizando las configuraciones en el modelo de datos mismo y no dependiendo de un archivo de configuración externo. A su vez, vale aclarar que el uso de configuraciones solo afecta al atributo de la Clase que se desea configurar, logrando de esta forma que los demás atributos si se desea sean identificados por QuickDB de forma automática.

2.5 Lenguaje de Consultas

En las aplicaciones tradicionales se suele tener código SQL inmerso en la aplicación, esto presenta ciertas complicaciones al tener que mantener códigos distintos situados dentro de la misma aplicación,

puede ser que se realicen cambios en el modelo de datos y el código SQL no quede en sincronía, a su vez, este código SQL contenido en la aplicación suele estar representado mediante cadenas de texto, las cuales no pueden ser validadas hasta el momento en que se hacen pruebas del programa en ejecución para ver si las distintas consultas fueron bien escritas sin errores sintácticos, y además estas consultas deben expresarse en el formato en que se maneja el modelo relacional apareciendo nuevamente el problema de la convivencia de los 2 modelos de datos.

Para resolver esta problemática planteada, QuickDB posee su propio lenguaje de consultas, el cual fue creado siguiendo los lineamientos y representaciones de un lenguaje orientado a objetos, de esta forma, el desarrollador puede hacer consultas sobre sus objetos planteando dicha consulta de una forma 100% Orientado a Objetos, logrando una mejor comprensión en la consulta y reduciendo la cantidad de código generado. QuickDB en realidad hace uso de 2 lenguajes de consultas creados para esta librería, el primero permite expresar operaciones de estructuras de datos medianamente simples con un enfoque orientado a objetos, y de forma bien compacta mediante una cadena de texto, de manera tal que teniendo en cuenta el Modelo de Datos planteado en la sección "Enfoque de QuickDB", si se quisiera realizar una búsqueda sobre una entidad almacenada en la Base de Datos del Objeto *Employee* solo se necesitaría realizar la consulta de la siguiente forma haciendo uso de este sistema de consultas planteado por QuickDB:

```
Employee e = new Employee();
admin.obtain(e, "address.street = 'unnamed street'");
```

Figura 4: Consulta Básica de QuickDB

Cuando en el caso de estar usando consultas en SQL embebidas en el código, para obtener el mismo resultado sería necesario escribir:

```
SELECT Employee.id, Employee.code, Employee.rolDescription, Employee.parent_id
FROM Employee
JOIN Person ON Employee.parent_id = Person.id
JOIN Address ON Person.address = Address.id
WHERE Address.street = 'unnamed street'
```

Figura 5: Consulta tradicional en SQL

Y luego crear la rutina necesaria de completar un Objeto con los datos recibidos, en cambio, el sistema de consultas de QuickDB devuelve automáticamente el resultado de la consulta expresado con los Objetos que constituyen el modelo de datos de la aplicación.

Con cualquier de estos 2 enfoques aun se sigue teniendo el problema de no poder validar si la consulta se encuentra bien expresada hasta el

momento de tener la aplicación en ejecución, para lo cual se genero un nuevo sistema de consultas en QuickDB mucho mas potente, el cual soporta cualquier tipo de estructura de datos y se encarga de mapear cualquier consulta, comprobación, subconsulta, etc. a una sintaxis que se asemeja a como se realizaría la misma en un lenguaje Orientado a Objetos, y la consulta expresada anteriormente ahora podría escribirse de la forma:

```
Employee e = new Employee();
admin.obtain(e).If("name").equal("son name").find();
//Para obtener una colección de Objetos Employee
ArrayList array = admin.obtain(e).If("name").equal("son name").findAll();
```

Figura 6: Sistema de Consultas de QuickDB

Y de esta forma a la vez, los algoritmos y las estructuras de datos involucrados en la confección de este tipo de consulta aseguran durante el desarrollo de que no se pueden escribir consultas mal formadas debido a que evalúa si las funciones que se están utilizando pueden ser llamadas en ese orden, etc.

2.6 Capacidades de QuickDB

QuickDB posee la capacidad de brindarle al desarrollador la habilidad de trabajar de manera indistinta con diferentes Bases de Datos, de esta forma puede cambiarse la Base de Datos de trabajo en cualquier momento sin afectar a la aplicación en lo más mínimo.

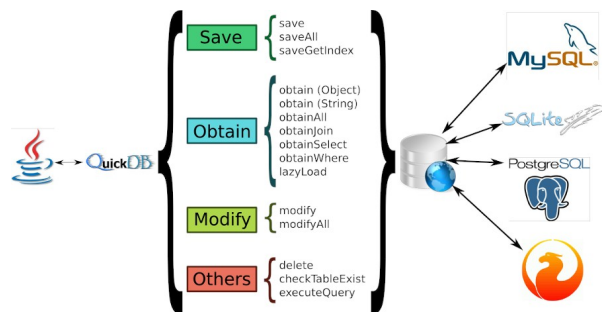


Figura 7: Capacidad de QuickDB

Logrando de esta forma que el desarrollador no solo pueda abstraerse de los mecanismos de interacción y mapeo entre los Objetos y la Base de Datos, sino también de la existencia de la Base de Datos misma.

2.7 Operaciones de Negocio (pre y post operaciones)

QuickDB implementa la posibilidad de poder realizar validaciones de forma automática sobre un Objeto en el momento previo a ser enviado hacia la Base de Datos, de esta forma, las clásicas operaciones de negocio para determinar la validez del Objeto pueden expresarse con propiedades de QuickDB, las cuales se

especifican a nivel de cada atributo de una Clase y de esta forma se elimina cierta complejidad en las operaciones involucradas previas a la persistencia del Objeto al centralizar las comprobaciones de estado del Objeto dentro del mismo Objeto y siendo estas ejecutadas de forma automática.

A su vez, QuickDB brinda los mecanismos necesarios para poder especificar algún método propio del Objeto o ajeno a este que se requiera ejecutar previa o posteriormente a las operaciones de persistencia del mismo.

2.8 Operaciones desde el Modelo de Datos

QuickDB también brinda la posibilidad de permitirle al desarrollador que el Modelo de Datos con el que se va a trabajar extienda de la clase AdminBinding (aunque no es necesaria esta forma de uso, solo se tiene en cuenta como un recurso más disponible), la cual como su nombre lo indica, cumple la función de hacer de "vínculo" entre las entidades y AdminBase (el Administrador de operaciones de QuickDB), para permitir que las operaciones de interacción con la Base de Datos sean ejecutadas desde el objeto mismo y no utilizar directamente como intermediario a AdminBase.

AdminBinding no cubre todas las funcionalidades que brinda AdminBase, ya que solo maneja aquellas operaciones que se refieren específicamente al Objeto que las contiene, dejando de lado a aquellas que se refieren a colecciones u operaciones independientes del objeto.

3. Resultados

El proyecto lleva ya casi un año de desarrollo desde su inicio, y a lo largo de este tiempo se han ido agregando mejoras tanto en cuestiones de performance como correcciones en los diversos algoritmos que posee. El estado actual del proyecto provee como resultado una versión estable de la librería de QuickDB, la cual actualmente soporta la interacción con 4 tipos de Bases de Datos distintas a través del lenguaje Java. Los objetivos que se perseguían al comenzar el proyecto pudieron ser logrados permitiendo de esta forma lograr desarrollar un software altamente intuitivo y con una simplicidad de uso notable, sin que esto signifique tener que sacrificar ningún tipo de funcionalidad. Ante las distintas situaciones que se presentaron durante el desarrollo de la librería, siempre se busco llegar a la solución que hiciera el uso de la librería lo más simple e intuitivo posible para la determinada acción que se pretendía desarrollar. Se tuvo en cuenta que para cada acción o funcionalidad que brinde la librería, siempre hay una determinada configuración que suele ser la mas frecuente, por lo que se estudiaron estas

situaciones y cada funcionalidad se programo tomando estas configuraciones como el caso por defecto con el que se trabajaría, dejando siempre abierta la posibilidad de que todas las características puedan ser detalladamente configuradas para modificar la ejecución de las operaciones a gusto del desarrollador.

El proyecto se encuentra hosteado en Google Code, haciendo uso del repositorio de versionado brindado por este servicio, y desde la creación del proyecto se han recibido más de 3000 visitas al día de la fecha distribuidas geográficamente de la siguiente forma:

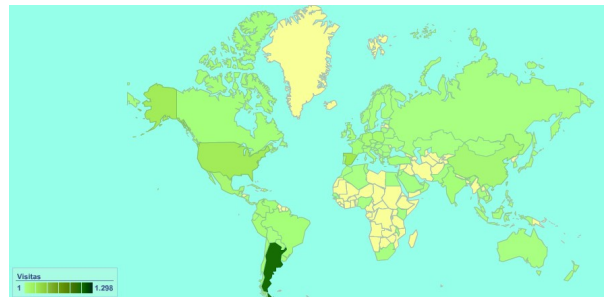


Figura 8: Distribución geográfica de acceso al Proyecto

Al ser un proyecto de Licencia Libre, pudo contar con aportes de algunas personas que se mostraron interesadas en el proyecto y han sugerido desde nuevas funcionalidades que se deberían implementar, hasta otras personas que se han sumado al proyecto de forma activa para comenzar a portar la librería a otros lenguajes como Python, PHP, .NET, etc.

La página web donde se encuentra ubicado el proyecto (<http://quickdb.googlecode.com>) cuenta con un Tutorial bien completo paso por paso que explica el uso de la librería, y gracias a la contribución de miembros que se fueron sumando, este Tutorial al día de la fecha se encuentra también disponible en Ingles, contando con ambos tutoriales tanto en formato HTML para ser consultado desde la página, como en formato PDF para ser descargado y consultado offline.

Debido a los resultados positivos obtenidos, se ha establecido un cronograma en el proyecto para ir liberando nuevas versiones cada 6 meses agregando muchas nuevas funcionalidades en cada una de estas versiones, siempre respetando la filosofía que persigue el proyecto donde: "La funcionalidad no es sinónimo de complejidad. Ser capaz de hacer diversas cosas no tiene que implicar perder mucho tiempo en tareas de configuración".

4. Discusión

QuickDB nace como una implementación con gran simplicidad de uso de un ORM en el lenguaje Java, ámbito donde ya existen algunas herramientas de gran renombre como Hibernate, el cual es el más utilizado actualmente. A diferencia de otras herramientas, QuickDB intenta que la simplicidad sea sumamente importante y que el programador nunca deje de pensar en sus objetos, ni tenga que preocuparse en como van a ser persistidos, y este framework de persistencia hace frente a esos objetivos logrando a la vez una gran facilidad de uso y una curva de aprendizaje muy rápida. QuickDB es Software Libre, lo que significa que todo el código fuente de la librería está disponible para quien lo necesite, pudiendo cualquier persona modificar y adaptar la funcionalidad a sus necesidades puntuales. También cualquier persona puede colaborar con la comunidad de QuickDB en su desarrollo y corrección de errores u optimizaciones para ampliar la calidad de la misma. Esto representa una gran ventaja para los usuarios, ya que el código fuente puede ser controlado, mejorado y optimizado por una gran cantidad de personas, en donde todas aprovechan estos aportes y todas son beneficiadas por este desarrollo en conjunto. Es interesante notar como esto facilita e incentiva el desarrollo y la investigación en Universidades, las cuales pueden participar de este proyecto ya sea usando la librería o aportando código que en definitiva va a ser un bien común, haciendo que la comunidad educativa forme parte de este desarrollo para su crecimiento. Esto no significa que QuickDB no se pueda adaptar a un modelo de negocios, ya que no se prohíbe el uso comercial de la herramienta, ni del soporte que se brinde a la misma mientras las modificaciones y correcciones sigan siendo Software Libre.

5. Conclusión

Este proyecto busca que la tarea de hacer persistente los datos sea sencilla y rápida, sin una curva de aprendizaje muy elevada, simplificando los procesos para que el programador nunca deje de pensar al nivel de objetos y pueda salvar el estado de los mismos en cualquier Base de Datos e incluso interactuar con distintas Bases de Datos sin tener que cambiar su forma de ver los objetos o la configuración de la aplicación. Otro punto que se busca con esta implementación, es disminuir las complicadas y tediosas configuraciones, ya que obligan al desarrollador a invertir un tiempo innecesario, y este framework pretende solucionar este problema con el

uso de las convenciones del lenguaje o simples anotaciones en la definición de cada Clase.

QuickDB logra estos objetivos haciendo uso de características como la reflexión y la recursividad, las cuales se encuentran presente en la mayoría de los lenguajes de programación modernos, teniendo en cuenta también las características especiales y forma de trabajar de cada uno de estos, para que el programador pueda utilizar sus conocimientos y las ventajas que ofrece el lenguaje siendo utilizado sin mucho esfuerzo ni tener que recurrir a la adaptación de su código.

Todo esto apoyado sobre una comunidad de desarrolladores y licencia de Software Libre en donde todos son beneficiados por las correcciones y nuevas características del framework, motiva al desarrollo de este proyecto, siendo además posible que las instituciones educativas puedan formar parte del desarrollo como una experiencia de aprendizaje mientras se obtienen conocimientos sobre una potente herramienta.

6. Referencias

- <http://es.wikipedia.org/wiki/QuickDB>
- Java Reflection In Action (Ira R. Forman – Nate Forman)
- <http://code.google.com/p/quickdb/>
- <http://code.google.com/p/quickdb/wiki/TutorialSpanish>
- http://es.wikipedia.org/wiki/Mapeo_objeto-relacional
- http://es.wikipedia.org/wiki/Open_Database_Connectivity
- <http://java.sun.com/docs/books/tutorial/reflect/index.html>
- <http://tutorials.jenkov.com/java-reflection/index.html>
- <http://sites.google.com/site/quickdbdeveloper>

Datos de Contacto

Nombre: *Diego Sarmentero*
Teléfono: 0351-152779700
e-mail: diego.sarmentero@gmail.com

Nombre: *Emilio Ramirez*
Teléfono: 0351-155949975
e-mail: emilioramirez04@gmail.com