



**UNIVERSIDAD DE TALCA
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA CIVIL EN COMPUTACIÓN**

**Desarrollo de sistema para migrar esquemas relacionales
a distintas variaciones de SQL**

RODRIGO EDUARDO VALENZUELA GUZMÁN

Profesor Guía: MATTHEW BARDEEN

Memoria para optar al título de
Ingeniero Civil en Computación

Curicó – Chile
Julio, 2018

CONSTANCIA

La Dirección del Sistema de Bibliotecas a través de su encargado Biblioteca Campus Curicó certifica que el autor del siguiente trabajo de titulación ha firmado su autorización para la reproducción en forma total o parcial e ilimitada del mismo.



Two circular stamps and signatures are present. The left stamp is blue and contains the text "UNIVERSIDAD DE TALCA", "DIRECCIÓN", "SISTEMA DE BIBLIOTECAS", and "BIBLIOTECAS". A signature is written over it. The right stamp is grey and contains the text "UNIVERSIDAD DE TALCA", "SISTEMA DE BIBLIOTECAS", "CAMPUS CURICO", and "BIBLIOTECAS". A signature is written over it.

Curicó, 2022

∴ ကိစ္စအား အားသာစွာ နှင့် အားသာစွာ အားသာစွာ အားသာစွာ အားသာစွာ
နှင့် အားသာစွာ အားသာစွာ အားသာစွာ အားသာစွာ အားသာစွာ ∴

Dedicado a los que no quieren programar las Bases de Datos.

AGRADECIMIENTOS

Agradecimientos en primer lugar a mi Mamá por todo, a Roemil Jorquera por ser un referente y ayudarme con los pagos de las primeras matriculas, a mis amigos de la vida por toda la buena onda, los juegos, el carrete y los conciertos, y a mi polola por revisar la redacción.

Por otro lado, a los que conocí en la Universidad, a la Comarka con mención especial a mi amigo Iván y a su moto, a los Hanssens por todos los malos ensayos y buenos momentos, y a Syrakh, por ser la mejor banda del mundo.

... Y a StackOverflow, porque lo necesario para hacer este proyecto no está en este documento, está ahí.

TABLA DE CONTENIDOS

	página
Dedicatoria	I
Agradecimientos	II
Tabla de Contenidos	III
Índice de Figuras	VI
Índice de Tablas	VII
Resumen	VIII
1. Introducción	9
1.1. Planteamiento del problema	9
1.2. Descripción del proyecto	9
1.3. Objetivo general	10
1.4. Objetivos Específicos	10
1.5. Alcances	10
1.6. Descripción del contenido	11
2. Marco teórico	13
2.1. Bases teóricas	13
2.1.1. Modelo relacional	13
2.1.2. Base de datos relacional	14
2.1.3. SQL	15
2.1.4. Sistema de gestión de bases de datos	15
2.2. Trabajo relacionado	16
2.3. Herramientas utilizadas	18
3. Metodología	20
3.1. Metodología de desarrollo aplicada	20
3.2. Metodología de evaluación del sistema	21
3.2.1. Evaluación de la interfaz	22

3.2.2.	Test unitarios	22
3.2.3.	Evaluación del funcionamiento	22
4.	Implementación de la Solución	24
4.1.	Diseño	24
4.1.1.	Arquitectura Física	24
4.1.2.	Arquitectura lógica	25
4.1.3.	Diagrama de clases	27
4.1.4.	Modelo de datos	32
4.1.5.	Interfaz de usuario	34
4.2.	Planificación	36
4.2.1.	Estándares de codificación	36
4.2.2.	Historias de usuario	37
4.2.3.	Funcionalidades y requisitos	37
4.2.4.	Planificación de iteraciones	38
4.3.	Proceso de desarrollo	38
4.3.1.	Requerimientos y Planificación - Iteración 0	39
4.3.2.	Iteración 1	40
4.3.3.	Iteración 2	40
4.3.4.	Iteración 3	41
4.3.5.	Iteración 4	41
4.3.6.	Iteración 5	42
4.3.7.	Iteración 6	43
5.	Evaluación del sistema	44
5.1.	Interfaz	44
5.2.	Funcionamiento	46
5.2.1.	MySQL	47
5.2.2.	SQLite	49
5.2.3.	Generación de scripts	51
6.	Conclusiones	52
6.1.	De la Metodología	52
6.2.	Del producto	52
6.3.	Objetivos	53

6.4. Trabajo futuro	54
Glosario	55
Bibliografía	57
Anexos	
A: Lista de funcionalidades	61
B: Tareas planificadas	64
C: Evaluación del funcionamiento	65
C.1. MySQL	65
C.2. SQLite	70
C.3. Generación de scripts	73

ÍNDICE DE FIGURAS

	página
4.1. Arquitectura física del sistema.	25
4.2. Arquitectura del sistema basada en el modelo MVC.	26
4.3. Esquema relacional (fragmento diagrama de clases).	28
4.4. Migraciones (fragmento diagrama de clases).	31
4.5. Generador de datos (fragmento diagrama de clases).	32
4.6. Modelo de datos de la solución.	33
4.7. Vista de una tabla en el sistema.	34
4.8. Vista de una base de datos en el sistema.	35
4.9. Vista de descarga de una base de datos en el sistema.	35
4.10. Vista principal de un usuario en el sistema.	36
5.1. Base de datos de evaluación ingresada en el sistema.	47
5.2. DB puesta en marcha en MySQL Workbench.	48
5.3. DB puesta en marcha en SQLiteStudio.	50
B.1. Fragmento de planificación de tareas del proyecto.	64
C.1. Diagrama EER para la evaluación del sistema.	65
C.2. DB creada en MySQL.	66
C.3. Consulta de selección en múltiples tablas en MySQL.	67
C.4. Consulta de inserción en MySQL.	68
C.5. Consulta de actualización en MySQL.	68
C.6. Consulta de selección en MySQL.	69
C.7. Consulta de eliminación en MySQL.	69
C.8. DB creada en SQLite.	70
C.9. Consulta de selección en múltiples tablas en SQLite.	71
C.10. Consulta de inserción y actualización en SQLite.	71
C.11. Consulta de selección en SQLite.	72
C.12. Consulta de eliminación y su demostración en SQLite.	72
C.13. Archivos de scripts SQL con números para orden en la ejecución.	73

ÍNDICE DE TABLAS

	página
5.1. Tiempo que tomo a los usuarios generar scripts utilizando el sistema. . . .	45
5.2. Consultas SQL programadas sin usar el sistema en 30 minutos.	45

RESUMEN

SQL (del inglés *Structured Query Language*) es un lenguaje de consultas utilizado para manipular bases de datos, que a pesar de ser un estándar, sus códigos no son siempre portables entre distintos sistemas gestores de bases de datos, además, la programación de consultas básicas en SQL es un proceso monótono, el cual puede ser automatizado.

En esta memoria se detalla todo lo que involucra al proceso de desarrollo de una plataforma web llamada Migra, cuyo objetivo es disminuir el tiempo de desarrollo de las bases de datos relacionales, permitiendo a los usuarios ingresar un esquema relacional de una manera rápida y simple, seleccionar el motor de bases de datos a utilizar y generar todas las consultas necesarias para dejar una base de datos operativa. Además de poder guardar los esquemas para ser reutilizados posteriormente.

El proyecto consiste en una herramienta que no actúa como una capa más que pueda mermar el rendimiento de la aplicación, sino que es un sistema totalmente aparte e independiente del entorno de trabajo o lenguaje de programación, que cumple con la función de acelerar el proceso de desarrollo sin tener que invertir tiempo en instalación, capacitación ni codificación.

Para llevar esto a cabo, se empleó una variación de la metodología de desarrollo *Extreme Programming*, la cual está adaptada para desarrollos de una sola persona, y finalmente se hizo un protocolo de evaluación para poner a prueba tanto de funcionamiento del sistema como de la interfaz. Donde tras mediciones del tiempo empleado por varios usuarios para crear bases de datos y verificar el correcto funcionamiento de las salidas entregadas por el sistema se concluyó que el producto cumplió con los objetivos establecidos.

1. Introducción

1.1. Planteamiento del problema

Basándose en que la codificación en SQL (del inglés *Structured Query Language*) de las bases de datos relacionales requiere de mucho tiempo de desarrollo, sabiendo que este proceso puede ser automatizado, y también en que la labor de poblar una base de datos para ejecutar pruebas puede tornarse complicada al momento de mantener la consistencia de los datos, nace la idea de desarrollar un sistema que permita automatizar estos procesos y así reducir el tiempo invertido en ellos.

Las herramientas disponibles para esto requieren la instalación de *software* o uso de ciertos *frameworks*, los cuales en algunos casos pueden actuar como una capa que merma el rendimiento de los sistemas. Por otra parte, el proceso de poblado de las bases de datos no está soportado por la mayoría de estos sistemas, lo que conlleva a tener que invertir tiempo en el desarrollo de generadores de datos, o el uso de otras herramientas.

1.2. Descripción del proyecto

Teniendo lo ya visto en cuenta, el proyecto se define como un sistema web para exportar esquemas relacionales a distintas variaciones de SQL mediante la generación de *scripts*. Lo cual asegura un fácil acceso a la plataforma y flexibilidad para los usuarios a la hora de elegir o migrar entre distintos motores de bases de datos.

1.3. Objetivo general

Disminuir el tiempo de desarrollo en la etapa de construcción de las bases de datos de un sistema, mediante una plataforma web para generar *scripts* de poblado, definición y manipulación de datos para distintos sistemas de gestión de bases de datos (en adelante DBMS por sus siglas en inglés *Database Management System*), a partir de esquemas relacionales.

1.4. Objetivos Específicos

1. Automatizar la generación de *scripts* de definición, inserción y manipulación de datos, a partir de un esquema relacional.

Para esto, el sistema debe poder:

- Exportar *scripts* con todas las consultas necesarias para dejar operativa una base de datos.
- Permitir gestionar esquemas relacionales en una cuenta de usuario, mediante una interfaz gráfica.

2. Permitir al desarrollador abstraerse del motor de bases de datos utilizado, sin perder las características que este entrega.

Para cumplir este objetivo se debe poder exportar los esquemas guardados a distintas variaciones de SQL según un DBMS escogido por el usuario.

3. Disminuir tiempo invertido en poblar las bases de datos

Esto se conseguirá mediante la:

- Generación de datos de prueba consistentes a los esquemas.
- Generación de *scripts* de inserción de datos.

1.5. Alcances

- En este trabajo se espera implementar un prototipo funcional de la idea a desarrollar.
- Para ingresar esquemas relacionales se utilizarán formularios HTML y mostrar una representación gráfica del esquema ingresado.

- Implementar un manejador de sesiones y cuentas de usuario básico.
- En este trabajo solo se implementará la migración para MySQL y SQLite.
- Para efectos de este trabajo, las migraciones implementadas se ceñirán a estándares personalizados, y no estrictamente a los impuestos por los sistemas de administración de bases de datos.
- Los datos se generarán correspondiendo al tipo de dato y restricciones especificadas en cada columna, y no a formatos personalizados, por ejemplo: RUT, rango de números, coordenadas geográficas, etc.
- El sistema no será puesto en marcha en un servidor.
- Este trabajo se limita a generar *scripts* en formato SQL para:
 - Creación de esquema.
 - Creación de tablas con tipos de datos y restricciones en las columnas, y la agregación de *primary keys*.
 - Creación de *foreign keys*.
 - Creación de consultas o procedimientos almacenados para crear, leer, modificar y eliminar filas en cada tabla.
 - Creación de consultas básicas (o procedimientos almacenados) para recuperar datos basándose en las relaciones dadas por las *foreign keys*.
 - Creación de archivos con datos generados aleatoriamente para poblar la base de datos.

1.6. Descripción del contenido

Capítulo 2: Toda la teoría y los conceptos necesarios para el entendimiento y el desarrollo del proyecto son explicados en este capítulo, además se analizan los trabajos relacionados al proyecto y las herramientas utilizadas para el desarrollo de este.

Capítulo 3: Se describe la metodología de desarrollo empleada y como esta es llevada a cabo, también se hace un seguimiento de los avances a lo largo de todo el proceso de

desarrollo y, se concluye describiendo la metodología de evaluación que contrasta el funcionamiento del sistema con sus objetivos.

Capítulo 4: El proceso de desarrollo e implementación es detallado a través de sus etapas y de la metodología empleada, se abordan temas de diseño y la planificación del proyecto.

Capítulo 5: Se exponen y analizan los resultados obtenidos tras aplicar la metodología de evaluación en el sistema.

Capítulo 6: En este capítulo se muestran las conclusiones extraídas del proyecto y se comenta sobre el trabajo futuro de este.

2. Marco teórico

Para la comprensión de este documento es necesario explicar algunos conceptos básicos concernientes a las bases de datos relacionales, abordando temas como el modelo relacional en el que gran parte de las DB (del inglés *Database*) se basan, y los lenguajes de consulta empleados para manipular la información almacenada. También se revisa el estado del arte y las herramientas empleadas para el desarrollo de este proyecto.

2.1. Bases teóricas

2.1.1. Modelo relacional

Las bases de datos relacionales, tal como su nombre indica, están basadas en el modelo relacional, el que a su vez tiene sus bases en la teoría de conjuntos y la lógica de predicados utilizado para el modelamiento y la gestión de las bases de datos. Se basa en el uso de relaciones, donde cada relación se compone de un conjunto de registros y campos. En una visión simplificada del modelo se puede visualizar una relación como una tabla con filas (registros) y columnas (campos), y se considera una base de datos como, un conjunto de relaciones.

Los datos se almacenan en las relaciones a modo de conjunto, por lo cual el orden de los registros es irrelevante, y tanto para almacenar como para recuperar los datos se utilizan expresiones lógicas llamadas consultas, las cuales son llevadas a cabo utilizando un conjunto de operadores definidos en el álgebra relacional [9, 10].

Dentro de lo que es un modelo relacional, se llama esquema a los metadatos que definen la estructura de una relación, mediante un nombre, el cual lo identifica de las demás

relaciones pertenecientes a la base de datos, y mediante nombres y dominios para cada campo de la relación, los cuales definirán los valores permitidos para cada campo [13].

2.1.2. Base de datos relacional

Teniendo en cuenta lo anterior, se llama base de datos relacional (RDB del inglés *Relational Database*) a toda colección digitalizada de datos que se estructure según el modelo relacional, es decir, en tablas con filas y columnas [13].

Dominio

Los tipos de datos que puede contener una columna vienen dados por el dominio de esta, el cual se encarga de acotar y definir precisamente que datos pueden ser almacenados en la columna, por ejemplo: el tipo de dato de una columna es *INT(32)*, lo que indica que el dominio de dicha columna son todos los enteros de *32 bits*. También existen tipos de datos que hacen referencia a formatos de strings como por ejemplo las fechas [13].

Restricciones

Las restricciones o *constraints* son usualmente operaciones que retornan un valor booleano el cual indica si un dato puede o no pertenecer a una columna, se utilizan para acotar el dominio de cada columna o para agregar reglas a esta, como por ejemplo evitar que se repitan valores dentro de la columna, o asegurar que no se ingresen datos nulos, también se pueden crear *constraints* más avanzadas, un ejemplo de esto es caso de verificar si un número es un RUT válido o no [13].

Primary key

Para identificar las filas dentro de una tabla se suele utilizar una *primary key* o llave primaria (en adelante PK), estas puede ser formada por una o más columnas (concatenando los datos para cada registro) y tiene la principal propiedad de ser única dentro de la tabla por lo tanto, también actúa como una restricción [13].

Foreign key

Una *foreign key* o llave foránea, en adelante FK, es usada para indicar que una columna de una tabla hace referencia a otra columna con el mismo dominio, perteneciente a otra

tabla, de esta manera se establecen vínculos que relacionan una tabla con otra. Se le llama *child column* a la columna que tiene la FK, y *referenced column* a la columna a la cual se hace referencia (también aplica para las tablas, *child* y *referenced table*), la *referenced column* suele ser una PK [13].

2.1.3. SQL

El canal de comunicación del desarrollador con la DB es el lenguaje SQL, por sus siglas en inglés *Structured Query Language*, es un lenguaje de consultas estándar, declarativo, de alto nivel y basado en el álgebra relacional. Es utilizado para manipular las RDB mediante consultas, ya sea para crear y modificar la estructura de esta, como para acceder y recuperar los datos almacenados. Todo esto mediante comandos que se separan en definición y manipulación de datos [13].

SQL es un estándar ANSI, pero tiene varias versiones y dialectos que agregan características y sintaxis distintas, pero siempre incorporando los comandos elementales definidos por SQL [13].

Definición de datos

El lenguaje de definición de datos (DDL por sus siglas en inglés *Data Description Language*) es usado esencialmente para la creación de esquemas de bases de datos, y sus operaciones básicas son *CREATE*, *ALTER*, *DROP* y *TRUNCATE* [13].

Manipulación de datos

El lenguaje de manipulación de datos (DML por sus siglas en inglés *Data Manipulation Language*) es usado para realizar consultas a una base de datos. SQL es un DML, por lo tanto implementa operaciones como *SELECT*, *INSERT*, *UPDATE*, *DELETE* y varias más [13].

2.1.4. Sistema de gestión de bases de datos

La manera de operar y poner en marcha una base de datos es a través de un DBMS, que es un *software* que se encarga de establecer una comunicación entre el usuario final y el conjunto de herramientas y programas necesarios para el funcionamiento de una DB.

Permiten crear esquemas, ejecutar consultas en SQL, analizar datos, y varias funciones más dependiendo del motor que se esté utilizando [11].

2.2. Trabajo relacionado

Para realizar migraciones de código SQL existen varios *softwares* disponibles donde el más destacable es DbSchema, que es un sistema de escritorio muy completo pero con tarifas elevadas (63 – 197 USD), este sistema está orientado a grandes empresas de desarrollo y su gran cantidad de herramientas lo hacen un *software* complejo el cual requiere de tutoriales y un basto conocimiento sobre bases de datos para ser utilizado, además que no automatiza la creación de procedimientos almacenados (o equivalentes), que es uno de los puntos importantes de este trabajo [3].

Por otro lado, en el proceso de poblado de bases de datos existen muchos sistemas *online*, pero el más completo es Mockaroo, el cual ofrece muchas posibilidades y flexibilidad en la creación de modelos de datos, tiene una basta cantidad de tipos de datos que pueden ser aleatorios, basados en diccionarios, en expresiones, en datos proporcionados por el usuario, secuencias, etc. También consta con un sistema de filtros para adecuar los datos a diversos contextos, sumado a esto cuenta con una interfaz muy simple e intuitiva [5].

Teniendo en cuenta las dos herramientas mencionadas y a las muchas mas que ofrecen funcionalidades similares, se puede apreciar que cubren a cabalidad las funcionalidades que se desean implementar en Migra, pero no logran empaquetar y transparentar todas estas funcionalidades para el usuario.

También existe el concepto ORM (del inglés *Object-Relational Mapping*) que consiste en mapear objetos de una RDB a un lenguaje de programación orientado a objetos, mediante una base de datos virtual de objetos. Esta técnica simplifica mucho el proceso de programación pero con un costo de rendimiento asociado, ya que el ORM se comporta como una capa extra entre la DB y el lenguaje de programación [18].

La mayoría de ORM permiten migrar fácilmente entre distintos motores de bases de datos, además de ofrecer funcionalidades como la de complementar las bases de datos

principales de un sistema con bases de datos en memoria para optimizar el rendimiento de las aplicaciones como por ejemplo Redis o Memcached. [18] [21]

Por lo tanto, Migra apunta al desarrollo de sistemas que utilicen una RDB y, en los que sea necesario evitar la capa extra que supone el uso de un ORM, ya sea por concepto de rendimiento o por la necesidad de acceder a características específicas de un motor de BD.

Como resumen, entre los sistemas existentes más destacados, podemos destacar algunos aspectos a modo de ventajas y desventajas. Cabe destacar que no se intenta hacer una comparación ya que son sistemas con propósitos diferentes, sino que resaltar aspectos a tener en consideración para el desarrollo de Migra.

DbSchema

- Precio elevado (versión de prueba gratuita, muy limitada).
- Difícil de utilizar.
- Soporta una gran cantidad de lenguajes y sintaxis.
- Es fácil migrar un proyecto de un lenguaje a otro.
- Soporta la creación de Índices y Foreign keys
- Se pueden modelar las bases de datos mediante un editor de diagramas.
- Los proyectos de DbSchema pueden ser guardados en un archivo.

Mockaroo

- Se puede utilizar de manera gratuita.
- Interfaz simple y bien lograda, se pueden crear esquemas sin la necesidad de un editor de diagramas.
- Se pueden guardar los datos del usuario en una cuenta, registrándose en la página.
- Gran cantidad de contextos y tipos de datos.
- Se pueden usar secuencias personalizadas con lógica.

- No permite exportar los datos en formatos directamente compatibles con los motores de bases de datos.
- La cantidad de filas a exportar en los archivos generados es limitada.

2.3. Herramientas utilizadas

Dado que el desarrollo será llevado a cabo solo por una persona, para la selección de herramientas se tomó en cuenta, además de los factores obvios, otros como el criterio y la experiencia en ciertas herramientas y procedimientos del autor de este documento, en pro de acotar el tiempo y de evitar estancamientos que puedan aplazar el tiempo de desarrollo.

Ya que el sistema consiste en un sitio web, se optó por utilizar el modelo MVC, ya que permite desacoplar completamente la vista, el modelo y los controladores, haciendo muy fácil la mantención y la escalabilidad del sistema, lo que es esencial en este caso. Algunos de los más populares *frameworks* MVC para desarrollo web son: Laravel, Ruby on rails, Django, ASP.NET, Yii, Symphony, etc. [17].

En este caso se optó por ASP.NET, ya que para cualquier entorno soportado por .NET se puede utilizar el IDE Visual Studio y el set de herramientas Community para el desarrollo en .NET, los cuales ofrecen un gran número de características y funcionalidades que facilitan en gran medida el proceso de construcción en comparación a otros IDEs, y también porque se puede utilizar el motor de vistas Razor, el cual agiliza bastante el proceso de desarrollo de vistas [2][17].

En cuanto a la base de datos se eligió MySQL Community Edition, porque es totalmente gratuita para sistemas en desarrollo (*server-only*) o bajo licencia GPL, el precio de *hosting* con bases de datos MS Sql Server es muy alto y porque la mayoría de los servicios de *hosting* incorporan al menos una base de datos MySQL. Además, se puede utilizar MySQL Workbench, que permite gestionar la base de datos fácilmente y de manera gráfica y diseñar bases de datos mediante diagramas EER [20].

Para las bases de datos también se utilizó Apache ANT, para cargar los *scripts* SQL de manera fácil y rápida a la base de datos, mediante un conjunto de configuraciones especi-

ficadas en un archivo XML [1].

Finalmente, para las vistas se utilizó Bootstrap 3.3.7, porque es un *framework* para CSS y JS sencillo y liviano, además es la versión con más documentación y componentes previamente creados en la web, y se decidió no utilizar plantillas ya hechas, ya que se requiere de un diseño muy específico [17].

3. Metodología

En este capítulo se explicará el porqué de la metodología escogida y el cómo esta se aplicó, abarcando aspectos como los estándares de programación, historias de usuarios y se mostrara un seguimiento del avance del trabajo realizado. También se definirán procedimientos para evaluar que el sistema cumpla con los objetivos.

3.1. Metodología de desarrollo aplicada

Ya que este proyecto debió ser llevado a cabo por una sola persona, la metodología de desarrollo utilizada fue una variación de *Extreme Programming* llamada PXP (por sus siglas en inglés *Personal Extreme Programming*), la cual adecua la mayoría de las etapas del XP tradicional para poder ser realizadas solo por una persona [8][12].

En PXP se definen varias actividades necesarias para llevar a cabo la aplicación de la metodología, a continuación, serán mencionadas en orden y se explicarán brevemente.

- **Requerimientos:** Captura inicial de requerimientos, los cuales se mantendrán relativamente estables a lo largo del proyecto, en este caso en particular, se elaborara un documento con requisitos funcionales y no funcionales [12].
- **Planificación:** Se elabora un conjunto de tareas y subtareas separados en categorías, para luego estimar un coste en tiempo para la implementación de cada una de estas tareas. En el caso de las sub tareas, la suma de sus tiempos debe representar el tiempo tomado en completar la tarea padre de la cual derivan [12].

- **Ietraciones:** En este caso se definió una “Iteración 0”, con el fin de adaptar la metodología de evaluación del presente trabajo con la de desarrollo. A continuación las etapas de cada iteración:
 - **Inicialización de la iteración:** En esta etapa se seleccionaran las tareas a realizar, las cuales serán el foco de la iteración. La cantidad de tareas tomadas debe ser acorde al tiempo asignado a la iteración, por ejemplo: una semana [12].
 - **Diseño:** En esta etapa se diseñan los nuevos módulos y clases que serán implementados en la iteración en curso [12].
 - **Implementación:** Aquí se elaboran los test unitarios y se genera todo el código necesario para la implementación de las tareas asignadas a la iteración. En caso de errores, mejoras, o fallas en los tests unitarios, se procede a la refactorización del código. Los test deben ejecutarse correctamente y el código debe compilar sin errores [12].
 - **Evaluación del sistema:** Se debe probar el sistema completo con todo el incremento acumulado de la iteración, verificar su correcto funcionamiento y que lo implementado corresponda a los objetivos del proyecto [12].
 - **Retrospectiva:** Se hace un análisis de los datos obtenidos en las etapas anteriores de la iteración, se debe reajustar el criterio de estimación para las tareas de las siguientes iteraciones y se debe prever posibles problemas que puedan afectar el avance del proyecto. También se analizan propuestas, mejoras e impedimentos. Esta fase puede dar paso a una siguiente iteración, así como también puede significar la finalización del proyecto, si todos los requerimientos están completados y no quedan correcciones por realizar [12].

3.2. Metodología de evaluación del sistema

Se diseñó un plan de pruebas que consiste en tres secciones: Evaluación de interfaz, funcionamiento y test unitarios, para así tener una cobertura mas amplia que va desde el correcto funcionamiento del código hasta métricas del comportamiento de usuarios de prueba.

3.2.1. Evaluación de la interfaz

Ya que el objetivo del sistema es reducir el tiempo de desarrollo de bases de datos, la interfaz será puesta a prueba con usuarios reales, los cuales seguirán el siguiente protocolo de evaluación, para determinar si el uso del sistema desarrollado agiliza o no el trabajo.

1. Se entregará al usuario un diseño de base de datos, el cual constará con todo lo necesario para poner a prueba el sistema.
2. El usuario deberá ingresar el diseño entregado (o un equivalente) a través del sistema, y luego exportar los *scripts* para el DBMS que estime conveniente. Este proceso será cronometrado.
3. El usuario deberá generar los *scripts* que representen el diseño entregado (o un equivalente) utilizando métodos tradicionales, con un límite de tiempo mayor al promedio de los resultados del ítem anterior, de esta manera se obtendrá información suficiente para evaluar si la interfaz del sistema cumple con la premisa de disminuir el tiempo de desarrollo de una base de datos.

3.2.2. Test unitarios

Consiste en la implementación de tests unitarios para los métodos del modelo, En este caso en particular solo se testearán las funciones del que interactúan directamente con la base de datos, probando así su correcta interacción y que se mantenga la consistencia de la base de datos.

Los test deben ser ejecutados después de cada integración del sistema y sin obtener ningún resultado erróneo, en caso contrario se deberá revisar el código o actualizar los test para poder proseguir con el proceso de integración.

3.2.3. Evaluación del funcionamiento

El sistema genera conjuntos de *scripts* para ser usados en un DBMS (en este caso MySQL y SQLite), para evaluar el correcto funcionamiento de estos, se debe crear un usuario y luego una base de datos, basándose en un diagrama de bases de datos.

El esquema debe ser exportado en cada una de las variantes de SQL permitidas, y luego ejecutar los *scripts* directamente en los DBMS correspondientes. Se debe obtener la base de datos creada, con datos cargados y lista para ser utilizada. Luego deben ejecutarse las siguientes consultas para cada DBMS, y compararlas con sus respectivas salidas.

Generación de scripts

Si el conjunto de *scripts* es generado en un solo archivo, este debe ser descargado automáticamente en formato SQL. Por otro lado, si el conjunto de *scripts* es generado como archivos separados cada archivo debe tener 4 dígitos al inicio de su nombre, donde el primero simboliza que tipo de *script* contiene, y los siguientes tres, el número correlativo de cada archivo, y estos deben ser descargados en un solo archivo comprimido.

- 1xxx = definición de esquema.
- 2xxx = definición de tablas y PK.
- 3xxx = definición de FK.
- 4xxx = definición de SP.
- 9xxx = inserción de datos.

4. Implementación de la Solución

4.1. Diseño

Antes de comenzar la etapa de implementación se diseñó una solución al sistema, la cual fue plasmada en múltiples diagramas para tener una visión general del sistema que sirvió como base para el proceso de programación. En este capítulo se explicarán las decisiones de diseño tomadas, la arquitectura elegida, la manera en la que se abstraigo el problema para ser documentado en diagramas, las interfaces gráficas y, algunos detalles de implementación.

4.1.1. Arquitectura Física

En la arquitectura utilizada para este sistema, el cliente se conecta con el servidor mediante el peticiones y respuestas vía HTTP, manejadas por los controladores de ASP.NET. A su vez el modelo de datos de ASP.NET se conecta mediante el conector de MySQL (Connector/NET) con la base de datos alojada en el servidor de base de datos. Para efectos de este proyecto se desplegaron los servidores en un mismo computador [6] [17].

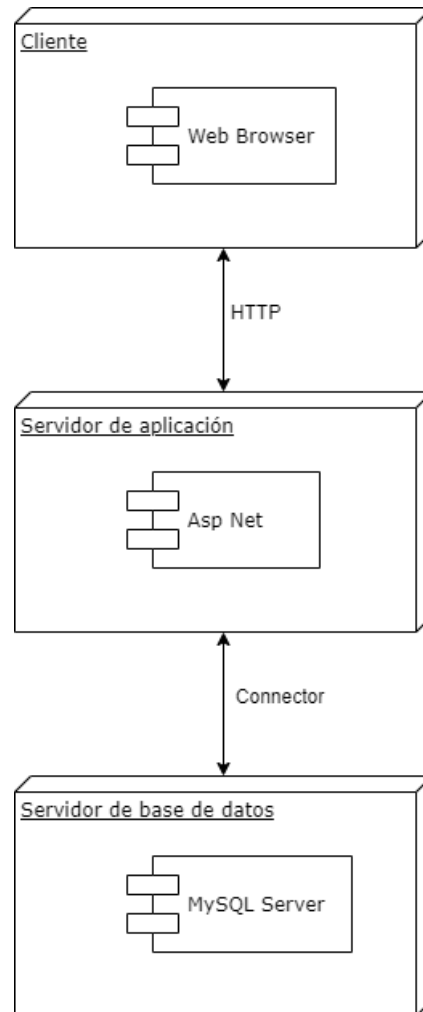


Figura 4.1: Arquitectura física del sistema.

4.1.2. Arquitectura lógica

En cuanto a la arquitectura de software, el modelo MVC es el que mejor se acomoda a las características de este proyecto, ya que permite desacoplar el sistema en tres capas principales, facilitando así la mantención y escalabilidad de este. En la Figura 4.2 se muestra un diagrama de la arquitectura y la interacción entre las capas.

Las capas que se describen para el modelo son:

- **Migraciones:** Consiste en clases que implementan la interfaz que se usa para definir la sintaxis de un lenguaje (SQL), aquí se pueden agregar lenguajes soportados, o modificar los ya implementados.

- **Generador de datos:** Es un modulo encargado solamente de generar datos para poblar las bases de datos, aquí se pueden agregar algoritmos para que los datos generados sean cada vez más files a la semántica de cada caso.
- **Diccionarios:** Es un conjunto de archivos con datos de distinta índole, utilizados por los algoritmos del generador de datos, en su mayoría utilizados para generar textos grandes y datos pseudo aleatorios.
- **Modelo:** Es un conjunto de clases que representa parcialmente una base de datos y que se usa para modelar los esquemas ingresados al sistema, aquí se pueden agregar nuevas funcionalidades como por ejemplo, que el sistema genere triggers, índices, vistas, etc.

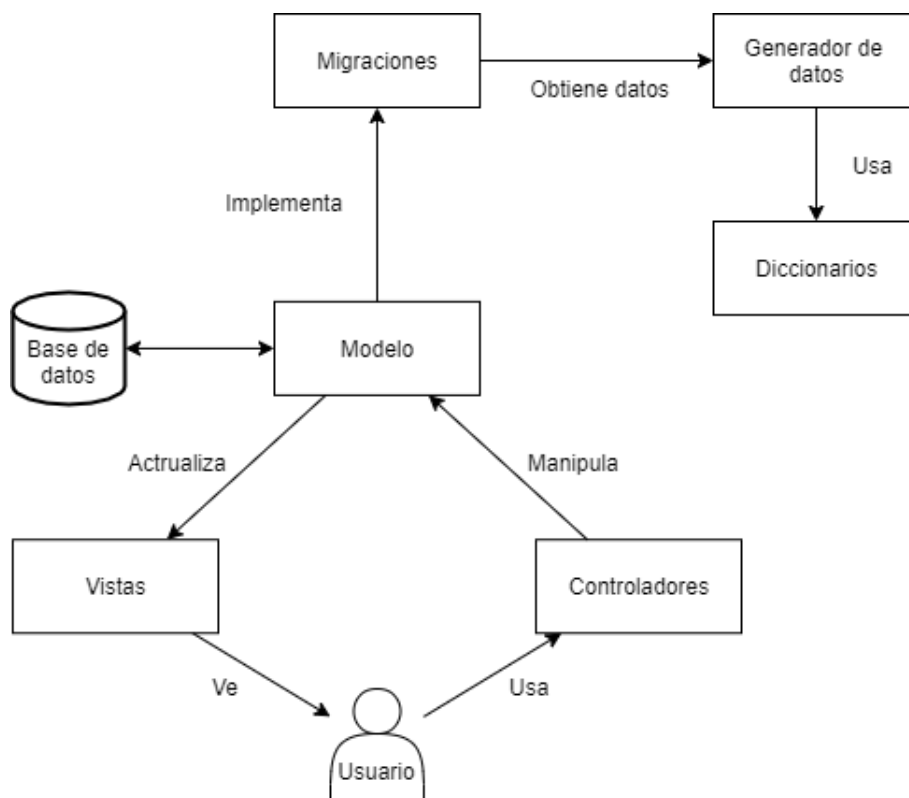


Figura 4.2: Arquitectura del sistema basada en el modelo MVC.

4.1.3. Diagrama de clases

A continuación en la Figura 4.3 se muestra un diseño de clases simplificado, donde se representa la abstracción de una base de datos con sus componentes básicos, omitiendo características como índices, vistas, *triggers*, etc. También se añade una clase *Script*, la cual se encarga de identificar cada *script* generado con su nombre y tipo, para después ser exportados de manera conjunta o separada. Otra clase extra es *User*, la cual representa al usuario que utiliza el sistema.

Para el entendimiento del diagrama es necesario referirse a los métodos CRUD(), presentes en muchas clases, que corresponden a *Create()*, *Read()*, *Update()* y *Delete()*, exceptuando el caso de *User* el cual cuenta con *ChangePassword()* en vez *Update()*, y con *Login()* en vez de *Read()*.

El enum *DataGroup* cumple la función de agrupar los tipos de datos para identificar qué tipos de entradas deben mostrarse al usuario en la interfaz del sistema al momento de ingresar los datos de una columna, para así evitar problemas como por ejemplo ingresar letras o datos alfanuméricos en un campo que debe ser llenado con números, como una columna de tipo *INT*.

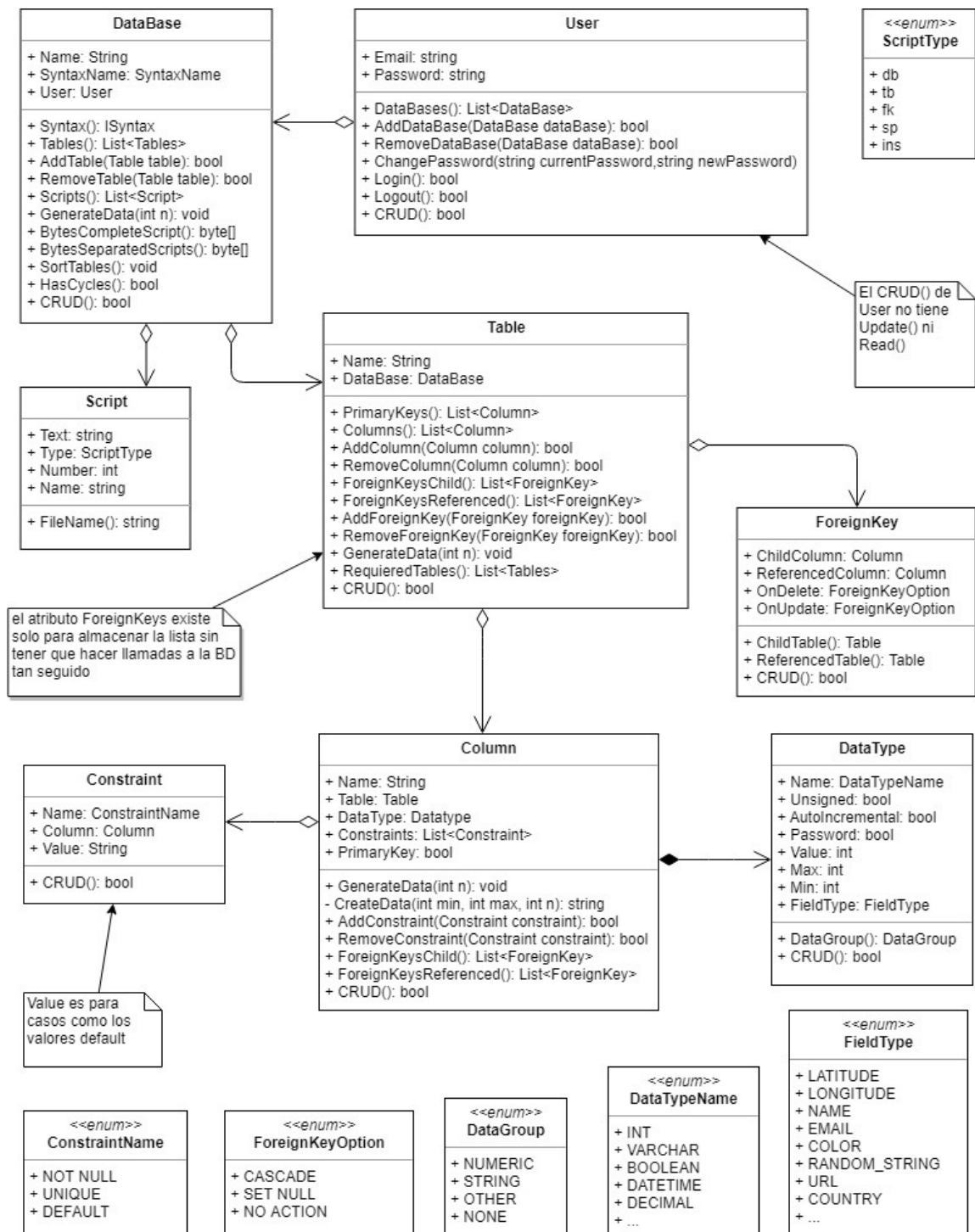


Figura 4.3: Diseño de clases que simula un esquema relacional básico (fragmento diagrama de clases).

Es importante mencionar el método `HasCycles()`, de la clase `DataBase`, el cual identifica si existen referencias circulares dentro del esquema, el algoritmo busca recursivamente ciclos donde todas las FK involucradas no permitan valores nulos en las columnas a las que hacen referencia, ya que en este caso no se podrían ingresar datos de forma normal en la DB. A continuación, se muestra un pseudo código del algoritmo.

```
bool HasCycles(DataBase DB) {
    foreach (table in DB) {
        history = new List<Table>();
        if (HasCycles(table, table, history))
            return true;
    }
    return false;
}

bool HasCycles(Table root, Table table, List<Table> history) {
    history.Add(table);
    fks = table.ForeignKeysChild();
    fksnn = fks.Where(x =>
        x no hace referencia a columna que permita null);

    foreach (fk in fksnn) {
        if (fk.ReferencedTable() == root) {
            if (fk.ChildColumn tampoco permite datos null)
                return true;
            continue;
        }
        if (HasCycles(root, foreignKey.ReferencedTable(), history))
            return true;
    }
    return false;
}
```

Por otra parte, para poder hacer las migraciones a múltiples sintaxis, se diseñó una interfaz llamada ISyntax (ver Figura 4.4) que es implementada por las clases del conjunto de migraciones, las cuales actúan bajo el patrón Singleton, para así evitar que se creen múltiples instancias para cada DBMS. La interfaz ISyntax se encarga de definir todos los métodos necesarios para generar el conjunto de *scripts* que cumplen con las funcionalidades básicas abarcadas en este proyecto. Para esto se definió una gramática de libre contexto, simplificada y sin símbolos terminales, que representa a grandes rasgos una idea inicial de que métodos debían ser definidos en la interfaz para luego ser reescritos para cada motor. Finalmente, para la creación de la clase ISyntax, se tomaron en cuenta tanto la gramática definida como otros aspectos de utilidad, comunes entre varios tipos de sintaxis analizados.

Schema → *DB* . *Table** . *FK** . *CRUD** . *QueryFK** . *InsertData**

Table → *Column**

Column → *DataType* . *Constraint**

CRUD → *Create* . (*Delete* | ε) . ((*Read* . *Update*) | ε)

CRUD → *Create* . (*Delete* | ε) . ((*Authenticate* . *ChangePassword*) | ε)

Create → *Parameters* . *Insert* . *LastInsertId*

Read → *Parameters* . *Select* . *From* . *Where*

Update → *Parameters* . *Set* . *Where*

Delete → *Parameters* . *From* . *Where*

Authenticate → *Parameters* . *Select* . *From* . *WhereAuth*

ChangePassword → *ParametersPass* . *Set* . *Where*

QueryFK → *Parameters* . *Select* . *From* . *Where*

El enum `SyntaxName` es utilizado en la clase `DataBase` para indicar cual es el DBMS que se desea utilizar en determinado momento, y el método `Syntax()` retorna la referencia a la migración correspondiente según el `SyntaxName` especificado. Cabe mencionar que dicha propiedad no es persistida en la base de datos del sistema, ya que solo es indicada al momento de exportar un esquema.

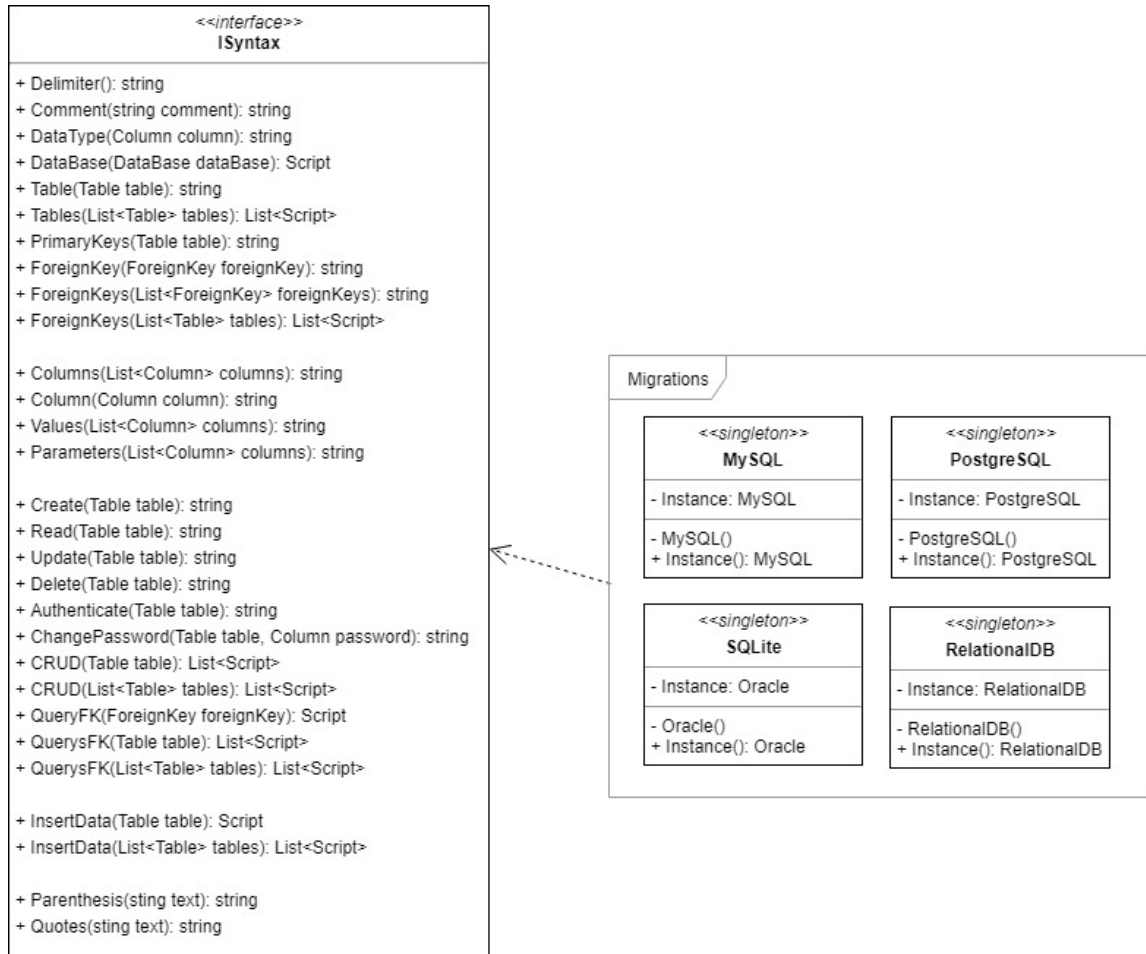


Figura 4.4: Implementación de la interfaz ISyntax por las clases que representan los distintos motores de DB (fragmento diagrama de clases).

Por otro lado, está la clase estática RandomDataGenerator (ver Figura 4.5) se encarga de proveer métodos para generar datos aleatorios según el tipo de dato de cada columna, y es utilizados para generar los *scripts* de inserción de datos de las tablas. Varios métodos de esta clase hacen uso de los diccionarios contenidos en la clase Dictionaries, la cual actúa bajo el patrón Singleton. Se utilizó esta manera de almacenar los diccionarios con el fin de evitar hacer tantas lecturas al disco en caso de que se almacenarán en un archivo, y se utilizó el patrón de diseño para reducir el uso de memoria al crear múltiples instancias.

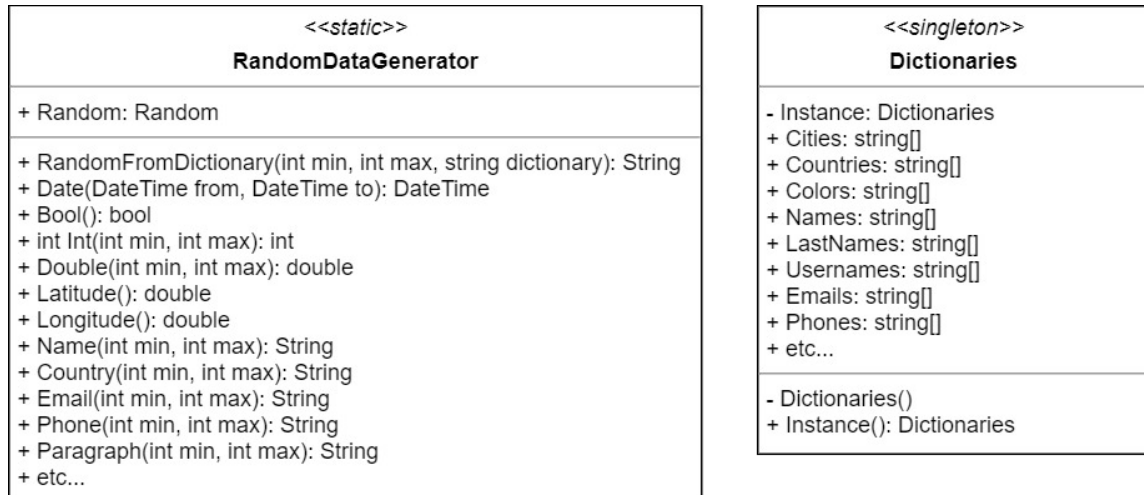


Figura 4.5: Clases utilizadas para generar datos para poblar las DB (fragmento diagrama de clases).

4.1.4. Modelo de datos

En cuanto a la persistencia de los datos en el sistema, se diseñó una base de datos capaz de almacenar todos los elementos necesarios para una representación básica de un esquema relacional y la información de los usuarios registrados, los cuales a su vez tienen varios esquemas.

Como se puede ver en el diagrama EER de la Figura 4.6, cada tabla está compuesta de columnas, las cuales a su vez tienen restricciones (*constraints*) y un tipo de dato donde es necesario explicar algunos campos.

En la tabla “datatype” se manejan los atributos “unsigned” y “autoincrement” de manera separada, ya que estos actúan como una extensión del tipo de dato, también existe la columna “password” la cual indica si en las consultas de inserción se debe encriptar la contraseña y si se deben hacer consultas especiales como en el caso de las cuentas de usuario como por ejemplo la autenticación de una cuenta. También se tiene la columna “fieldtype” la cual indica el tipo semántico del campo, por ejemplo, si es un nombre, apellido, país, color, etc. Y las columnas “min” y “max” las cuales indican el rango de valores en los cuales se pueden generar datos para posteriormente poblar la DB, donde para el caso de cadenas de texto estos valores restringen el largo de la cadena generada.

Otro campo importante es “value” que indica el parámetro que requieren algunos tipos de datos como por ejemplo: *varchar(255)*.

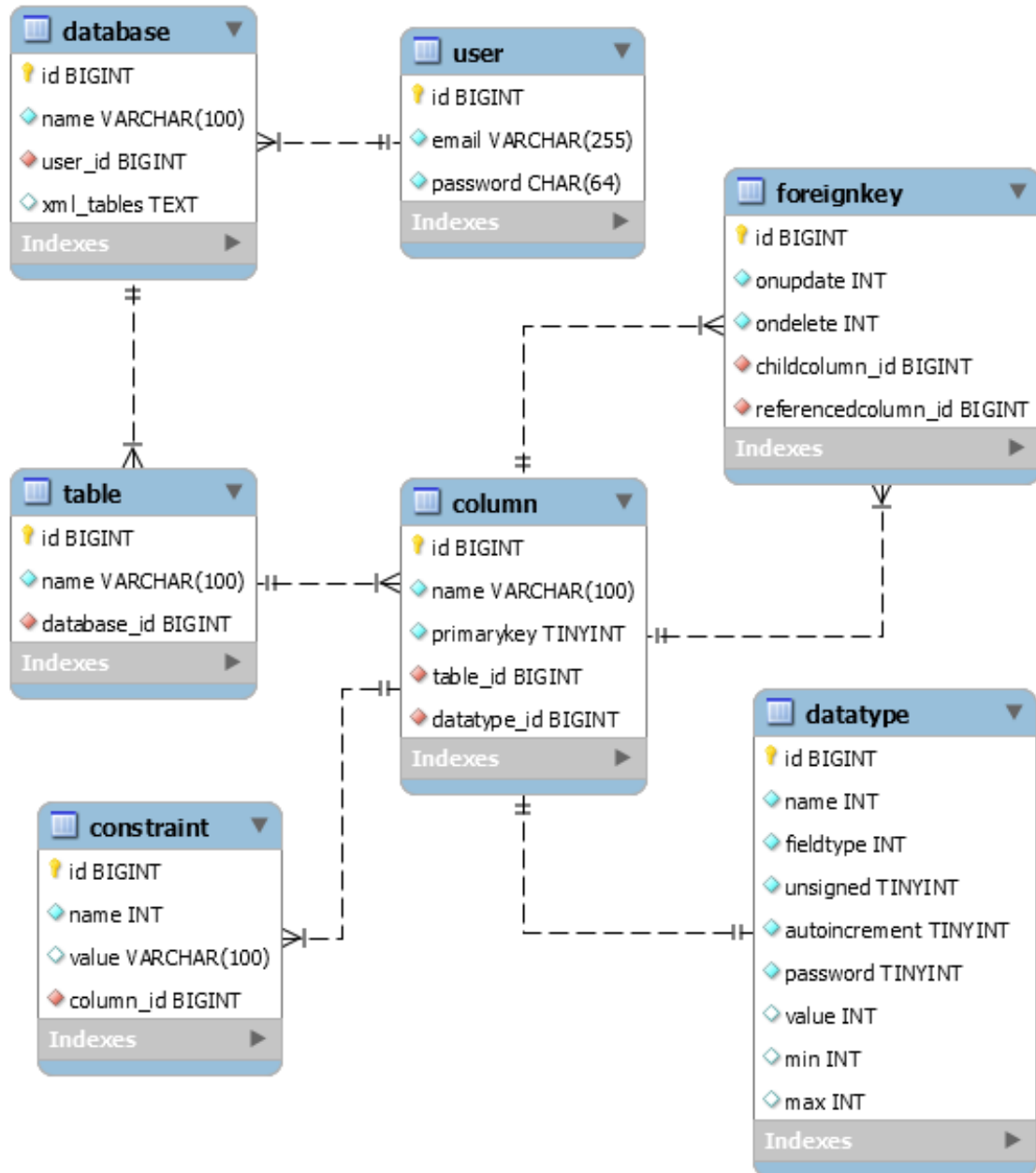


Figura 4.6: Modelo de datos de la solución.

4.1.5. Interfaz de usuario

Se comenzó el diseño de la interfaz mediante la confección de *wireframes* para mostrar a los usuarios una idea inicial de que es lo que se quería lograr y cómo se iba a ver el sistema, luego se creó un *mockup* utilizando HTML5, CSS y JavaScript, con el fin de ser un prototipo evolutivo, el cual se basó en las ideas expuestas en los *wireframes* y, fue utilizado como base para la creación de la interfaz final del sistema.

Se puso énfasis en el diseño de una interfaz que permitiera ver, agregar, modificar y eliminar columnas de una tabla, sin tener que navegar entre una y otra página, y minimizando la cantidad de *clicks* necesarios y tiempos de espera con llamadas a la base de datos y cargas de páginas. Finalmente se llegó al diseño mostrado en la Figura 4.7, el cual muestra la información de las columnas a modo de acordeón, donde se permite realizar todas las acciones necesarias sin necesidad de ir a otra página.

Databases / Guitar_catalogue / Store Change name [+ Add Column](#)

Id	Primary key	✓	🗑️	▼
name		✓	🗑️	▼
web page		✓	🗑️	▼

address **Foreign key** 🗑️ ^

Column name:

Data type: Unsigned Autoincrement

Constraints: Primary key Unique Not Null Foreign key Default value:

Fill type:

Foreign Key: Table - Column: On delete: On update:

[Save changes](#)

Figura 4.7: Vista de una tabla en el sistema, con un acordeón que despliega la información de las columnas.

Ya que es usual agregar identificadores a las tablas se automatizó la creación de una columna Id al momento de crear cada tabla, la cual es un número auto incremental, único y, además es una PK, esto se puede ver reflejado en las Figuras 4.7 y 4.8.

En cuanto a la vista parcial de las tablas de una DB (Figura 4.8), se diseñó una interfaz sencilla que da al usuario una vista general de las PK y relaciones entre tablas, mediante iconos situados antes de los nombres, donde el verde indica que esa columna compone una PK y el azul que es una FK. Además se agregó un botón Download que permite descargar la DB actual (Figura 4.9).



Figura 4.8: Vista de una base de datos en el sistema.

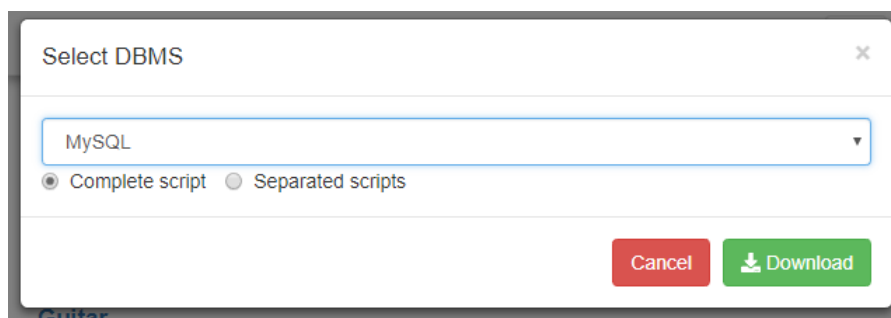


Figura 4.9: Vista de descarga de una base de datos en el sistema.

Finalmente, para la vista que muestra los esquemas de un usuario se agregó un indicador para notificar al usuario si un esquema tiene o no ciclos, ya que las referencias circulares no pueden ser manejadas de manera estándar en la mayoría de los DBMS. Adicionalmente se puede ver en la Figura 4.10 el diseño de la barra de navegación del sistema.

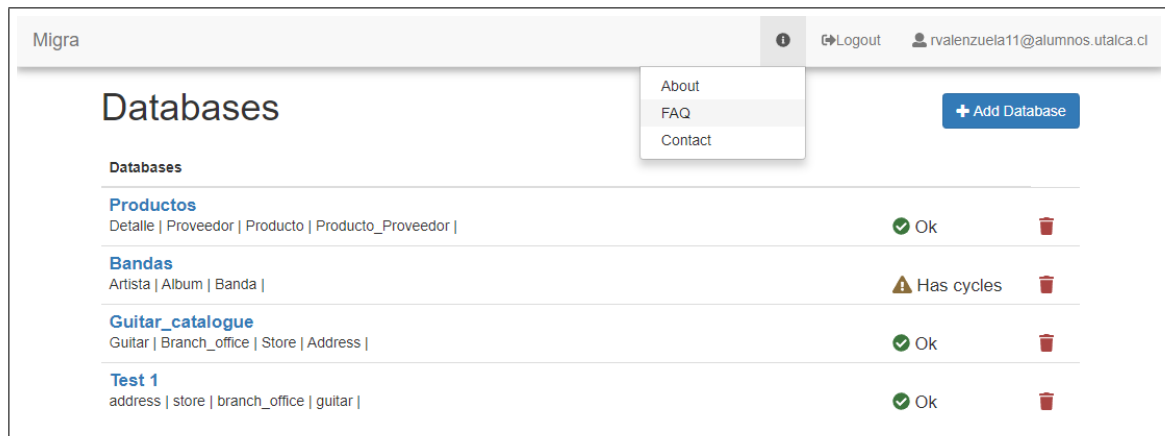


Figura 4.10: Vista principal de un usuario en el sistema, con la lista de sus esquemas.

4.2. Planificación

En esta sección se expone todo lo que se planificó a lo largo del proyecto, basándose en la metodología PXP. La mayor parte de esto se documentó en las primeras etapas del proyecto y se abordan aspectos como la definición de estándares, creación y recopilación y desglose de historias de usuarios y la planificación de las iteraciones.

4.2.1. Estándares de codificación

Una vez claros los objetivos, alcances del proyecto y las herramientas a utilizar (ver Capítulos 1 y 2), se definieron los estándares y convenciones de codificación para ser utilizados a lo largo de todo el desarrollo. Se fijaron estándares para las siguientes tecnologías:

- **C#:** Se siguió la guía de codificación y convenciones de C# publicada en el sitio web oficial de .NET [19].
- **Razor:** Se utilizó la sintaxis definida en el sitio web de w3schools.com para los archivos CSHTML [2].
- **JS:** Para JavaScript se siguió la guía de convenciones y estándares de codificación disponible en el sitio web de w3schools.com [4].
- **CSS:** En el caso de CSS se utilizó el estándar definido por un usuario de GitHub [15].

- **MySQL:** Se utilizaron los estándares y convenciones definidos en las guías para desarrolladores puestas a disposición en el sitio web de MySQL [6].

4.2.2. Historias de usuario

Las historias de usuarios fueron tanto creadas directamente por quien desarrolla el sistema, como capturadas en reuniones con los interesados en el proyecto, y de la retroalimentación recibida de quienes participaron en los procesos de pruebas y validaciones. A continuación, se listan las historias de usuarios utilizadas.

- Que el sistema permita ingresar esquemas relacionales de tablas.
- Que en cada tabla pueda poner restricciones, llaves, tipos de datos, etc.
- Que pueda seleccionar un motor de base de datos y exportar el esquema a varios *scripts* de ese motor.
- Que los *scripts* se puedan generar con números (en los nombres de los archivos) para configurar un XML y ejecutarlos en orden, o poder generar un solo archivo.
- Que se generen *scripts* con procedimientos almacenados o sus equivalentes con las consultas básicas.
- Que se generen *scripts* para poblar la base de datos, sin que queden inconsistencias.
- Poder crear una cuenta de usuario para poder guardar las tablas y esquemas ingresados, para posteriormente reutilizarlos.
- Que el sitio web sea fácil de utilizar y que tenga una sección de ayuda.
- Que se muestre una vista previa de las relaciones entre las tablas para poder validar los datos ingresados.

4.2.3. Funcionalidades y requisitos

Las historias de usuarios fueron descompuestas en características que debe poseer el sistema, que para efectos de este documento serán llamadas funcionalidades o requisitos. Las funcionalidades fueron agrupadas y sub agrupadas según sus tipos, de los cuales de identificaron tres grupos importantes:

- **Vistas:** son todas las interfaces gráficas y controladores. Permiten al usuario ver y crear esquemas de bases de datos.
- **Generación de scripts:** hace referencia al núcleo del sistema, es la parte que se encarga de recibir un esquema de BD, persistirlo, procesarlo, y generar los scripts con su correspondiente migración. Aquí es donde esta casi toda la lógica.
- **Migraciones:** Son clases a parte, que se comunican con el resto del modelo a través de una interfaz, cada migración hace referencia a un motor de base de datos o a una versión de este, y definen su sintaxis para cada instrucción, por ejemplo: crear tabla, crear FK, crear consultas, etc.

La lista completa de funcionalidades está disponible en el *Anexo A. Lista de funcionalidades*.

4.2.4. Planificación de iteraciones

El proyecto completo se dividió anticipadamente en siete iteraciones (para calzar con la planificación del módulo de este proyecto de título), comenzando por la iteración cero, cuya duración será mucho más larga en comparación a las siguientes seis, las cuales deben ser planificadas cada cuatro semanas.

La *Iteración 0* contempla los procesos de captura y elaboración de requerimientos y planificación general del proyecto, además de una fase de investigación y desarrollos experimentales para probar la factibilidad de los futuros desarrollos.

Las tareas de cada iteración se irán priorizando acorde a las peticiones de los interesados para así poder presentar avances, aun así, se priorizaron las del grupo *Generación de scripts* y luego en paralelo se trabajó en las del grupo *Migraciones* y *Vistas*.

4.3. Proceso de desarrollo

En este capítulo se detalla el avance en el proceso de desarrollo a través de cada iteración, tomando como referencia las fases de la metodología PXP. Cabe destacar que el primer sprint fue de una duración extendida, y no se implementó nada, solo se planificó

y elaboro la propuesta de proyecto, por otro lado, el último sprint involucra el proceso de evaluación del sistema en su totalidad. Las tareas planificadas pueden ser consultadas en el *Anexo B. Tareas planificadas*.

4.3.1. Requerimientos y Planificación - Iteración 0

Esta iteración contempla (en la práctica) las fases de Requerimientos y Planificación, pero se agregaron dentro de una iteración “Especial” ya que al tratarse de un proyecto propio se necesitaba obtener retroalimentación de profesores e interesados, hacer diseños preliminares y probar la factibilidad de desarrollar ciertas cosas en el tiempo previamente establecido para el desarrollo del proyecto. Cabe mencionar que se tomaron tareas dentro de esta misma iteración.

- **Inicialización de la iteración:** las tareas abordadas en esta iteración fueron en general ligadas al diseño general del sistema:
 - Comparación de las sintaxis de los distintos motores para la creación de scripts.
 - Investigar sobre algoritmos o métodos para generar datos.
 - Diseño de mock-up en HTML.
 - Estudio y análisis del problema a solucionar e investigación de los distintos motores.
 - Modelo de una base de datos relacional.
 - Modelo del sistema de migraciones.
- **Diseño:** Se creo la primera versión del diagrama de clases del sistema a y de diseñaron los primeros *wireframes* para el futuro desarrollo de las interfaces. Se decidió hacer esto en la primera iteración para poder tener una visión más clara al momento de fijar plazos y duraciones de las siguientes actividades y de cómo abordar el proyecto.
- **Planificación:** Se crearon conjuntos de funcionalidades a partir de las funciones y características extraídas de las historias de usuarios (*Migraciones, Vistas y Generación de scripts*). También se planifico el orden y prioridad de cada funcionalidad del sistema. También se hicieron estudios y análisis de los distintos DBMS.

- **Testing:** Se escribieron las pruebas de aceptación para cada conjunto de funcionalidades.
- **Retrospectiva:** Se tuvo retroalimentación principalmente sobre el diseño del módulo de migraciones, por lo que este será corregido en el futuro. Y se decidió sacar ciertas funcionalidades del alcance del proyecto.

4.3.2. Iteración 1

- **Inicialización de la iteración:** las tareas abordadas en esta iteración corresponden a todo lo relacionado con la biblioteca de clases principal del sistema:
 - Diseño de base de datos mediante diagramas EER.
 - Construcción del modelo.
 - Construcción de base de datos.
- **Diseño:** Se realizó el diseño del modelo de datos el cual, a su vez, simula una base de datos. También se hicieron modificaciones menores al diseño de clases creado en la Iteración anterior.
- **Implementación:** Se creó la biblioteca de clases correspondiente al modelo del sistema. También se implementaron todos los test unitarios para el modelo, los cuales interactúan con la BD.
- **Testing:** El conjunto de tests unitarios no se ejecutó ya que aún no había una conexión a la base de datos.
- **Retrospectiva:** Se siguió recibiendo retroalimentación acerca del diseño de clases y de los patrones de diseño utilizados para las migraciones. Y se decidió acelerar el proceso de implementación de migraciones para MySQL, para así generar los scripts de la BD del propio sistema.

4.3.3. Iteración 2

- **Inicialización de la iteración:** Se comenzó la construcción e integración de las vistas con el modelo, además se hicieron correcciones de la iteración anterior:
 - Construcción de la vista.

- Construcción de los controladores.
- **Diseño:** Se actualizo el diseño de clases, del modulo de migraciones, y se retocaron los *wireframes* para proceder con la implementación de las vistas.
- **Implementación:** Se hicieron correcciones menores al modelo y se actualizaron los tests. También se comenzó con la construcción de las vistas y controladores del sistema, así como también la implementación de las migraciones para MySQL en la biblioteca de clases.
- **Testing:** Nuevamente el conjunto de tests unitarios no se ejecutó ya que aún no había una conexión a la base de datos.
- **Retrospectiva:** Se recibieron recomendaciones sobre la usabilidad de las interfaces graficas.

4.3.4. Iteración 3

- **Inicialización de la iteración:** Se continuo con las tareas de la iteración anterior.
- **Diseño:** Se hicieron arreglos menores en el diseño de clases.
- **Implementación:** Se continuó con la construcción de las vistas, controladores y la migración de MySQL. Tambien se replicaron los cambios del diseño de clases inmediatamente.
- **Testing:** Nuevamente el conjunto de tests unitarios no se ejecutó ya que aún no había una conexión a la base de datos. Pero se documento detalladamente el plan de pruebas para test unitarios de funcionamiento del sistema, basado en el avance de la migración de MySQL.
- **Retrospectiva:** No hubieron cambios en requisitos, ni retroalimentaciones.

4.3.5. Iteración 4

- **Inicialización de la iteración:** Se continuo con las tareas de la iteración anterior (Construcción de vista y controladores) y se creó una tarea para terminar de programar la migracion de MySQL.

- **Diseño:** En esta iteración no hubo diseño.
- **Implementación:** Se continuó con la construcción de las vistas y controladores y se finalizó la implementación de la migración de MySQL. Además se generaron los scripts de la BD utilizando la misma migración de MySQL ya desarrollada, se dejó la BD operativa y se estableció la conexión con el modelo de clases. Se terminó con el desarrollo del conjunto de funcionalidades de *Generación de scripts*.
- **Testing:** Se pudo ejecutar el conjunto de tests unitarios con éxito y se revisó que se estuvieran cumpliendo los criterios de aceptación basándose en los scripts generados.
- **Retrospectiva:** Se decidió cambiar la siguiente migración a implementar de PostgreSQL a SQLite, ya que la implementación de la primera hubiese sido muy similar a la de MySQL. Además los resultados de los test unitarios dejaron ver ciertas fallencias que pueden afectar el rendimiento del sistema, las cuales serán revisadas en la siguiente iteración.

4.3.6. Iteración 5

- **Inicialización de la iteración:** Se continuo con la creacion de vista y controladores y se creo una tarea para implementar la migracion de SQLite.
- **Diseño:** Cambios menores en el diseño de la interfaz que implementan las migraciones.
- **Implementación:** Se continuó con la construcción de las vistas y controladores y se hicieron cambios en los métodos para crear migraciones. Además, se comenzó a desarrollar la migración de SQLite.
- **Testing:** Se ejecutó el conjunto de tests unitarios con éxito, También se empezó a validar con usuarios las vistas que terminadas y las que estaban aun en construcción.
- **Retrospectiva:** Se solicitaron cambios menores en tipografías y palabras empleadas en las interfaces, y sugerencias para trabajos futuros.

4.3.7. Iteración 6

- **Inicialización de la iteración:** Se terminaron las funcionalidades y se comenzó la etapa de pruebas generales:
 - Construcción de migración SQLite.
 - Pruebas generales del sistemas.
- **Diseño:** Retoques finales a los planes de prueba, para la evaluación del sistema.
- **Implementación:** Se finalizó la construcción de las interfaces de usuario y la migración de SQLite concluyendo así con los grupos de funcionalidades de *Migraciones* y *Vistas*. Se dejó el sistema listo para ser puesto a prueba.
- **Testing:** Se llevo a cabo todo el proceso de evaluación del sistema, detallado mas adelante en el Capítulo 5.
- **Retrospectiva:** Se obtuvieron opiniones de usuarios y resultados de las evaluaciones las cuales fueron utilizadas para la redacción de conclusiones acerca del trabajo realizado.

Se finalizó la construcción de las interfaces de usuario y la migración de SQLite concluyendo así con los grupos de funcionalidades de *Migraciones* y *Vistas*.

5. Evaluación del sistema

Con el desarrollo finalizado, se procedió con la etapa de pruebas y evaluaciones para determinar en qué medida fueron cumplidos los objetivos del proyecto. A continuación, se describirá la metodología de evaluación empleada junto a los resultados obtenidos de esta. Todos los procedimientos mostrados a continuación siguen la metodología de evaluación descrita en la Sección 3.2 de este documento.

5.1. Interfaz

Para evaluar la interfaz se midió el tiempo que tardan los sujetos de prueba en realizar una tarea determinada (la cual requiere conocimientos muy básicos sobre bases de datos), sin tener una experiencia previa utilizando el sistema, de esta manera se pretende determinar si la interfaz ayudó a acelerar el proceso o no, y si los tiempos de uso fueron o no similares entre los distintos usuarios. Seguido de esto, se midió el tiempo que tardaron los mismos usuarios en crear *scripts* SQL utilizando otros métodos, sin el fin de comparar los tiempos medidos, sino que de hacerse una idea de lo demoroso que puede ser este proceso, para esta parte de la evaluación se asignó un tiempo límite ya que de antemano era sabido que podía extenderse demasiado.

La evaluación se realizó con cinco usuarios voluntarios, con conocimientos previos en bases de datos y sin haber utilizado el prototipo del *software* anteriormente. Los usuarios uno, dos y tres tenían varios años de experiencia con las RDB, no así los usuarios cuatro y cinco, los cuales solo han trabajado con bases de datos en el ambiente académico.

A continuación, se muestra en el Cuadro 5.1, el tiempo que tardo cada usuario en

traspasar el diagrama de la Figura C.1 al sistema y luego generar los *scripts* (para cualquier DBMS).

Cuadro 5.1: Tiempo que tomo a los usuarios generar scripts utilizando el sistema.

	Tiempo (minutos)
Usuario 1	10.7
Usuario 2	14.1
Usuario 3	12.8
Usuario 4	12.5
Usuario 5	15.6

Como se puede apreciar en el Cuadro 5.1, el promedio de tiempo necesario fue de 13.14 minutos, por lo tanto, se dio un tiempo de 30 minutos a los usuarios, para generar *scripts* equivalentes con métodos tradicionales.

Tras esto se obtuvo la información disponible en el Cuadro 5.2, donde se marca con una «x» los códigos o tareas que se completaron satisfactoriamente por los usuarios.

Cuadro 5.2: Consultas SQL programadas sin usar el sistema en 30 minutos.

	Comentarios	Tablas	Drops	PK	FK	SP	Poblado
Usuario 1		X		X	X		X
Usuario 2	X	X	X	X	X		X
Usuario 3		X	X	X			
Usuario 4	X	X		X	X		
Usuario 5	X	X		X	X		

Complementariamente a los resultados observados en el Cuadro 5.2 es necesario mencionar la situación particular de cada usuario.

- **Usuario 1:** Se encontraba haciendo los SP al término del tiempo y generó datos en formato CSV con ayuda de otro sistema.
- **Usuario 2:** Pasó por alto los SP y utilizó otro sistema para generar las consultas de inserción y así poblar la base de datos.

- **Usuario 3:** No logró concluir el desarrollo de los SP y se retiró a los 20 minutos (aprox.), ya que considero que era un tiempo muy reducido para completar más tareas.
- **Usuario 4:** Se encontraba haciendo los SP al término del tiempo.
- **Usuario 5:** Se encontraba haciendo los SP al término del tiempo y logro avanzar bastante en esta tarea, ya que hizo varias consultas utilizando otros sistemas.

En cuanto a la retroalimentación recibida por los usuarios, el aspecto más destacable fue las sugerencias de mejoras a la interfaz gráfica del sistema, estas acotaciones apuntaban principalmente al eventual caso de crear esquemas mucho más grandes que el utilizado en el proceso de evaluación, ya que en dicho caso se tornaría difícil manejar las referencias correctamente, por lo cual sería una buena opción el poder representar los esquemas mediante diagramas que sean más familiares al usuario y que ayuden a validar visualmente las referencias.

5.2. Funcionamiento

Una vez ingresado el esquema de la Figura C.1 en el sistema, como se puede ver en la Figura 5.1, se procedió a exportarlo a la sintaxis de MySQL y SQLite para luego ejecutar las consultas respectivas para cada motor, especificadas más adelante, con el fin de poner a prueba el correcto funcionamiento de los *scripts* generados.

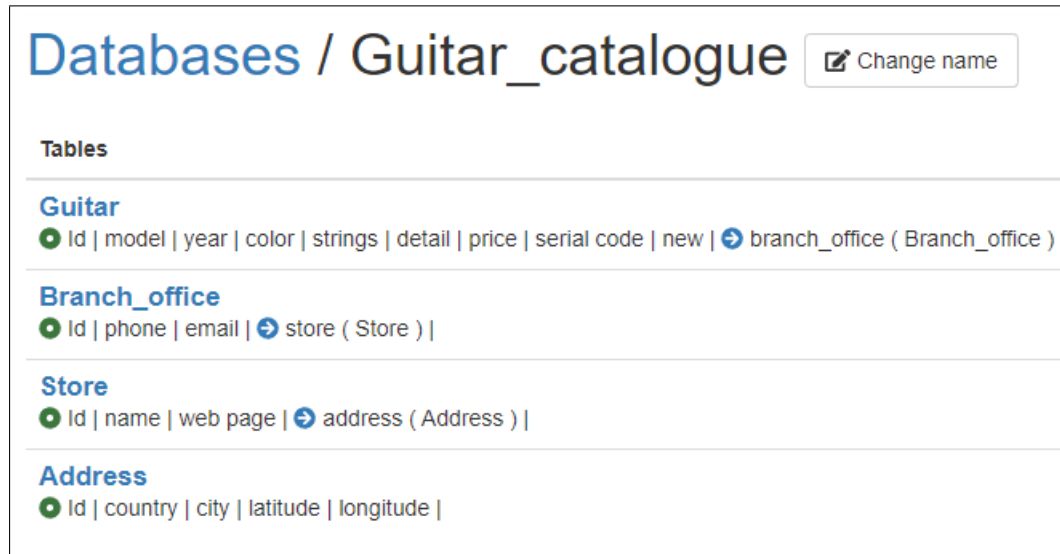


Figura 5.1: Base de datos de evaluación ingresada en el sistema basándose en un diagrama.

5.2.1. MySQL

Conjunto de pruebas MySQL

- La salida obtenida para la siguiente consulta debe ser **100**.

```
SELECT address.Id
FROM address, branch_office, guitar, store
WHERE guitar.Id = 100 AND
guitar.branch_office = branch_office.Id AND
branch_office.store = store.Id AND
store.address = address.Id;
```

- La salida obtenida para la siguiente consulta debe ser **101**, ya que el SP selecciona el valor del último ID ingresado.

```
CALL sp_create_Store("name", "www.example.com", 100);
```

- La siguiente consulta debe funcionar sin errores.

```
CALL sp_update_Store(101, "Name2", "www.example.com", 50);
```

- La salida obtenida para la siguiente consulta debe ser:
101, "Name2", "www.example.com", 50.

```
CALL sp_read_Store(101);
```

- La primera consulta debe ejecutarse sin errores y la salida obtenida para la segunda debe ser **100**.

```
CALL sp_delete_Store(101);  
SELECT COUNT(store.Id) FROM store;
```

Ejecución de las pruebas MySQL

Para ejecutar las pruebas en MySQL se utilizó *MySQL Workbench 8.0 CE* y como se puede ver en las imágenes del Anexo C.1, se obtuvieron resultados satisfactorios en todo el conjunto de pruebas. A continuación, en la Figura 5.2 se muestra una captura de la DB correctamente cargada en el DBMS, y una consulta para revisar que los datos fueron satisfactoriamente ingresados.

The screenshot shows the MySQL Workbench interface. On the left, the 'Navigator' pane displays the 'guitar_catalogue' database schema with tables (address, branch_office, guitar, store) and stored procedures. The main window shows a query: `SELECT * FROM guitar_catalogue.address;` and a 'Result Grid' with 23 rows of data.

Id	country	city	latitude	longitude
1	Israel	North Las Vegas	13.3844673425818	81.6112395755999
2	Latvia	Port St. Lucie	7.86832958826252	37.5482534172704
3	Colombia	Lansing	-32.0092574665366	-76.0196653031836
4	Algeria	Columbia	-24.9189733331646	26.2855736689109
5	El Salvador	Tucson	49.4698636385006	-166.924420679419
6	Burundi	Columbia	-84.482439616454	111.256350993298
7	Italy	Coral Springs	1.56560697106906	-18.6434168311038
8	Taiikistan	Columbia	38.2260844084556	-53.3790261328123
9	Tunisia	Marvsville	-12.1390235135979	21.3305675961685
10	Singapore	Murrieta	23.5678154335207	-99.5277350505477
11	Vietnam	Vallejo	2.39487427118927	151.404800872507
12	Ivorv Coast	North Charleston	-85.75207949977	-76.7470888564117
13	Rwanda	Hollywood	-54.4248800940928	21.9041344518327
14	Liberia	Monterev	-51.3012181279721	-57.5967165127381
15	Svria	Columbia	-46.9730363688306	98.3572382304618
16	Korea North	Tallahassee	-54.9179839691697	-77.7451468607155
17	Guvana	Little Rock	-44.601356569492	-168.341583017885
18	Belaium	Tuscaloosa	-73.5830248536463	12.3984425460912
19	Bahamas	Roanoke	50.0581953232448	104.868879803395
20	Turkmenis	Westminster	31.3413124561037	143.494071126214
21	Israel	New London	-57.0218065036562	-164.937624266808
22	Anoola	Havre de Grace	7.73769324120958	112.973203656717
23	Iceland	Johnson Citv	-0.673010304883593	152.468473385772

Figura 5.2: DB puesta en marcha en MySQL Workbench.

5.2.2. SQLite

Conjunto de pruebas SQLite

En el caso de SQLite se deben copiar las consultas generadas ya que no existen procedimientos almacenados para este motor.

- La salida obtenida para la siguiente consulta debe ser **100**.

```
SELECT Address.Id
FROM Address, Branch_office, Guitar, Store
WHERE Guitar.Id = 100 AND
Guitar.branch_office = Branch_office.Id AND
Branch_office.store = Store.Id AND
Store.address = Address.Id;
```

- Las siguientes consultas deben ejecutarse sin errores.

```
INSERT INTO "Store" ( "name", "web page", "address" )
VALUES ( "name", "www.example.com", 100 );
```

```
UPDATE "Store"
SET
"name" = "Name2",
"web page" = "www.example.com",
"address" = 50
WHERE
101 = "Store"."Id";
```

- La salida obtenida para la siguiente consulta debe ser: **101, "Name2", "www.example.com", 50**.

```
SELECT "Id", "name", "web page", "address"
FROM "Store"
WHERE 101 = "Store"."Id";
```

- La primera consulta debe ejecutarse sin errores y la salida obtenida para la segunda debe ser **100**.

```
DELETE FROM "Store"
WHERE 101 = "Store"."Id";

SELECT COUNT(store.Id) FROM store;
```

Ejecución de las pruebas SQLite

Para ejecutar las pruebas en SQLite se utilizó *SQLiteStudio 3.1.1* y como se puede ver en las imágenes del Anexo C.2, se obtuvieron resultados satisfactorios en todo el conjunto de pruebas. A continuación, en la Figura 5.3 se muestra una captura de la DB correctamente cargada en el DBMS, y una consulta para revisar que los datos fueron satisfactoriamente ingresados.

A diferencia de MySQL, aquí no se crearon SP, ya que en su lugar se decidió generar consultas equivalentes a estos.

	Id	country	city	latitude	longitude
1	1	Ecuador	Flint	23.8854150343153	-4.35294022008448
2	2	Russian Federation	Bethlehem	19.5443613392042	55.7301601892012
3	3	Mauritius	New Haven	-58.2559953389019	-0.877751477937098
4	4	Tuvalu	Rockford	59.4137932399352	-171.673662761074
5	5	Monaco	Port Arthur	87.8684991048037	144.000525804702
6	6	Andorra	Fort Smith	-36.0148516981932	-2.10617213654619
7	7	Lebanon	Palm Bay	42.3140298660444	-156.996873580849
8	8	Switzerland	Fairfield	-83.4778699597706	-121.205571306034
9	9	Sweden	Antioch	70.4504018111389	69.4917747911493
10	10	Estonia	Newport News	48.521578348019	-117.858816060638
11	11	Kuwait	San Antonio	82.2408128475029	-149.100110354414
12	12	Myanmar, {Burma}	Chesapeake	10.9212168841256	-94.3035056778712
13	13	Ecuador	Newark	-39.7103043723434	68.8804156961294
14	14	Mauritius	Newburgh	-12.3630860030479	-16.5593736463037
15	15	Kuwait	Leominster	-44.9488265434973	143.042023362146
16	16	Mali	St. Paul	73.2175974642009	-64.9088809121907
17	17	Bolivia	Huntsville	-84.0850666496367	-114.822634763467
18	18	Israel	Mesquite	29.4691077160971	-97.1300017466443
19	19	Latvia	High Point	85.0197095354179	12.4843728637716
20	20	Liechtenstein	Elk Grove	85.9573155906784	-80.2310959669906
21	21	Belgium	Visalia	-72.8285713614098	-127.394784310551
22	22	Burundi	Pueblo	-87.3444340603167	-27.1405651635214
23	23	Lebanon	Mesquite	50.842220047881	-148.141144249654
24	24	Laos	Brighton	-32.634537544397	162.048468114831

Figura 5.3: DB puesta en marcha en SQLiteStudio.

5.2.3. Generación de scripts

Tras realizar las pruebas se obtuvieron resultados satisfactorios en cada una de ellas (disponibles en el *Anexo C*), aun así esto no descarta la posibilidad de encontrar errores en el futuro ya que la enorme cantidad de casos posibles hace inviable una evaluación tan profunda. De todos modos, la evaluación realizada basta para asegurar el correcto funcionamiento del prototipo en la mayoría de los casos.

6. Conclusiones

En este capítulo se exponen las conclusiones obtenidas a lo largo del proceso de elaboración de esta memoria. Se concluye acerca del sistema desarrollado, los problemas que se presentaron durante las etapas y como estos fueron solucionados, también se contrarrestan los resultados obtenidos en la evaluación del sistema con los objetivos del proyecto. Adicionalmente se habla sobre el posible trabajo futuro con relación al sistema, como mejoras y extensiones.

6.1. De la Metodología

En cuanto a la metodología de desarrollo utilizada, Personal Extreme Programming, fue una buena elección, tomando en cuenta principalmente que el desarrollo fue llevado a cabo por una sola persona y a que el tiempo de desarrollo era acotado. En cuanto a los requisitos se tuvieron usuarios con visiones muy distintas del sistema, pero, aun así, no hubo mayores cambios, ya que la formulación del proyecto fue bastante sólida, por lo tanto, esto no impacto en el proceso de desarrollo. Finalmente se puede decir que a pesar de no existir experiencia previa con PXP, la metodología pudo ser llevada a cabo sin ningún tipo de problemas y entrego buenos resultados.

6.2. Del producto

El prototipo de Migra cumple su función a cabalidad, pero aun así es un sistema que puede madurar en muchos aspectos y agregar funcionalidades nuevas, pero contrarrestando el producto obtenido con los objetivos planteados al inicio del proyecto, se puede afirmar que cumple con el cometido principal de disminuir los tiempos de desarrollo de las

bases de datos entregando una solución alternativa a los métodos, sistemas y tecnologías ya existentes.

Cabe indicar que las pruebas fueron llevadas a cabo con una DB pequeña, pero se intuye que la usabilidad de la interfaz, y por ende la eficiencia del sistema, pueden verse mermadas con una DB mucho más grande.

6.3. Objetivos

Como se pudo ver en la evaluación del sistema (Capítulo 5.1), a pesar de los distintos niveles de conocimiento de los usuarios, todos pudieron utilizar el *software* sin problema alguno y, logrando una notable disminución del tiempo de desarrollo en comparación al empleado utilizando otros métodos, por lo tanto, se dan por cumplidos los objetivos de disminuir el tiempo invertido en programar y poblar bases de datos mediante la generación de *scripts*. Por consiguiente, y como se demostró el correcto funcionamiento del sistema (Capítulo 5.2), se da por cumplido el objetivo de permitir al desarrollador abstraerse del motor de base de datos utilizado. Por lo tanto, se da por cumplido el objetivo general.

A continuación un detalle de los objetivos específicos planteados en este proyecto y el como fueron cumplidos.

1. **Automatizar la generación de *scripts* de definición, inserción y manipulación de datos, a partir de un esquema relacional.**

El usuario puede generar un esquema y en base a este generar de manera automática todos los *scripts* necesarios (según el DBMS seleccionado) para poner en marcha la base de datos. Esto va desde la creación de la misma hasta el poblado de las tablas y la generación de consultas y procedimientos almacenados. Además, el usuario puede guardar sus esquemas en su cuenta para reutilizarlos posteriormente.

2. **Permitir al desarrollador abstraerse del motor de bases de datos utilizado, sin perder las características que este entrega.**

Actualmente el sistema puede exportar un mismo esquema a más de un motor sin necesidad de hacer ningún cambio por parte del usuario, independientemente de las diferencias entre un lenguaje y otro, o las características soportadas por los distintos DBMS.

3. Disminuir tiempo invertido en poblar las bases de datos

El sistema permite generar scripts de poblado con datos generados automáticamente, asegurando la consistencia y el correcto funcionamiento de la base de datos.

6.4. Trabajo futuro

En cuanto al trabajo posterior, el proyecto deja muchas puertas abiertas para seguir mejorando y extendiendo sus funcionalidades, algunas de estas son:

- Ampliar y mejorar el módulo de generación de datos para incluir más dominios y, trabajar más la semántica de los datos.
- Añadir migraciones para más motores de DB y permitir elegir entre distintas versiones de un mismo motor.
- Ejecutar pruebas y evaluaciones avanzadas de rendimiento y usabilidad, para realizar mejoras en algoritmos e interfaces.
- Puesta en marcha del sistema en un servidor *online*.
- Visualización de diagramas correspondientes a los esquemas, que sean mas familiares a los usuarios y permitan validar las referencias entre tablas.

Glosario

DB: Base de Datos, (del inglés, *Data Base*).

RDB: Base de Datos Relacional, (del inglés, *Relational Data Base*).

DBMS: Sistema de gestión de bases de datos (del inglés *Database Management System*), es un conjunto de programas que gestionan las bases de datos y permiten al usuario comunicarse con estas mediante un lenguaje de consultas.

RDBMS: Sistema de gestión de bases de datos relacionales (del inglés *Relational Database Management System*).

CSV: Formato de archivos donde se almacenan tuplas de datos separados por comas «,» (del inglés *Comma-Separated Values*).

SP: Procedimiento almacenado (del inglés *Stored Procedure*), es una consulta o conjunto de consultas que se guardan para ser utilizadas recurrentemente.

PK: Llave primaria (del inglés *Primary Key*).

FK: Llave foránea (del inglés *Foreign Key*).

SQL: Lenguaje de consulta estructurada (del inglés *Structured Query Language*).

Script: Para efectos de este documento solo se hace referencia a Scripts SQL, que son archivos con conjuntos de consultas.

CRUD: Métodos básicos para el mantenedor de un objeto, Crear, Leer, Modificar, Eliminar (del inglés *Create Read Update Delete*).

Framework: Entorno de trabajo conceptual y tecnológico, que puede contar con programas, bibliotecas o lenguajes interpretados.

XP: Programación extrema (del inglés *Extreme Programming*), es una metodología de desarrollo de *software*.

ORM: Mapeo de objeto-relacional (del inglés *Object-Relational Mapping*), es una herramienta para guardar los datos de los objetos de un sistema directamente a una base de datos.

EER: Diagrama entidad relación extendido (del inglés *Enhanced Entity-Relationship* ó *Extended Entity-Relationship*).

Bibliografía

- [1] The apache ant proyect. <https://ant.apache.org/>. Consultado: 2018-06-2.
- [2] Asp.net razor - c# and vb code syntax. https://www.w3schools.com/asp/razor_syntax.asp. Consultado: 2018-05-15.
- [3] Dbschema - diagram designer and query tool. <http://www.dbschema.com/>. Consultado: 2017-08-31.
- [4] Javascript style guide and coding conventions. https://www.w3schools.com/js/js_conventions.asp. Consultado: 2018-05-15.
- [5] Mockaroo - a realistic data generator. <https://www.mockaroo.com/>. Consultado: 2017-08-31.
- [6] Mysql: Test case content-formatting guidelines. https://dev.mysql.com/doc/dev/mysql-server/latest/PAGE_CONTENT_FORMATTING_GUIDELINES.html. Consultado: 2018-05-15.
- [7] Spikes in scrum: The exception, not the rule. <https://www.scrumalliance.org/learn-about-scrum/agile-atlas/agile-atlas-commentaries/may-2014/spikes-in-scrum-the-exception,-not-the-rule,2014>. Consultado: 2018-10-21.
- [8] R. Agarwal and D. Umphress. Extreme programming for a single person team. In *Proceedings of the 46th Annual Southeast Regional Conference on XX*, ACM-SE 46, pages 82–87, New York, NY, USA, 2008. ACM.
- [9] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

- [10] E. F. Codd. Derivability, redundancy and consistency of relations stored in large data banks. *ACM SIGMOD Record*, 38(1):17, 2009.
- [11] K. R. Dittrich, S. Gatzju, and A. Geppert. The active database management system manifesto: A rulebase of adbms features. In T. Sellis, editor, *Rules in Database Systems*, pages 1–17, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.
- [12] Yani Dzhurov, Iva Krasteva, and Sylvia Ilieva. Personal extreme programming—an agile process for autonomous developers. 2009.
- [13] R. Elmasri and S. Navathe. *Fundamentals of Database Systems*. Addison-Wesley Publishing Company, USA, 6th edition, 2010.
- [14] Martin Fowler and Matthew Foemmel. Continuous integration, 2006. URL <http://martinfowler.com/articles/continuousIntegration.html>, 2012.
- [15] K. Lubos. xfiveco/css-coding-standards. <https://github.com/xfiveco/css-coding-standards>, Apr 2018. Consultado: 2018-05-15.
- [16] Robert C Martin. *Agile software development: principles, patterns, and practices*. Prentice Hall, 2002.
- [17] Jamie Munro. *ASP.NET MVC 5 with Bootstrap and Knockout.js: Building Dynamic, Responsive Web Applications*. O’Reilly Media, Inc., 2015.
- [18] P. Van Zyl, D. G. Kourie, and A. Boake. Comparing the performance of object databases and orm tools. In *Proceedings of the 2006 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, pages 1–11. South African Institute for Computer Scientists and Information Technologists, 2006.
- [19] B. Wagner. C# coding conventions (c# programming guide). <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions>, Jul 2015. Consultado: 2018-05-15.
- [20] Li Yang and Li Cao. The effect of mysql workbench in teaching entity-relationship diagram (erd) to relational schema mapping. *International Journal of Modern Education and Computer Science*, 8(7):1, 2016.

- [21] Hao Zhang, Bogdan Marius Tudor, Gang Chen, and Beng Chin Ooi. Efficient in-memory data management: An analysis. *Proceedings of the VLDB Endowment*, 7(10):833–836, 2014.

ANEXOS

A. Lista de funcionalidades

Se asigno un identificador a cada funcionalidad correspondiente a los números de la lista que se muestra a continuación.

1. Generación de scripts.
 - 1.1. Comentarios.
 - 1.2. Crear BD.
 - 1.3. Crear tablas.
 - 1.4. Crear PKs.
 - 1.5. Crear FKs.
 - 1.6. Crear SPs.
 - 1.6.1. CRUD.
 - 1.6.2. Autenticación de usuario.
 - 1.6.3. Cambio de contraseña.
 - 1.6.4. Que el SP para crear devuelva el ID si es que es auto incremental (en los DBMS que así lo permitan).
 - 1.6.5. Encriptar passwords.
 - 1.6.6. Que los inputs de los SP sean solo los campos necesarios.
 - 1.6.7. Consultas derivadas de las FKs.
 - 1.7. Poblar DB.
 - 1.7.1. Generar datos.
 - 1.7.2. Generar scripts para poblar (insert).

- 1.7.3. Que no se generen datos inconsistentes.
- 1.7.4. Generar los scripts en un orden consistente con las FK para evitar errores.
- 1.8. Poder generar un solo script grande.
- 1.9. Poder generar varios archivos separados y con números (para ejecutar con Apache ANT (.zip)).
- 1.10. Poder seleccionar el DBMS.
- 2. Migraciones.
 - 2.1. Migración de MySQL.
 - 2.2. Migración de SQLite.
- 3. Vistas.
 - 3.1. Ingresar esquema mediante formularios.
 - 3.1.1. Nombre de columna.
 - 3.1.2. Tipo de dato (int, char, bool, etc).
 - 3.1.3. Restricciones (unique, not null, etc).
 - 3.1.4. Tipo de campo (id, password, nombre, edad).
 - 3.1.5. Limites (min y max).
 - 3.1.6. Ingresar cada tabla por separado.
 - 3.1.7. Indicar a que tabla apuntas las FK.
 - 3.2. Vista Home.
 - 3.2.1. Información acerca de cómo utilizar la aplicación y para qué sirve.
 - 3.3. Navbar.
 - 3.3.1. Gestión de cuentas de usuarios (Login, Logout, Sign In, User).
 - 3.3.2. Ayuda dropdown (about, contact, etc.).
 - 3.4. Vista User.
 - 3.4.1. Ver datos.
 - 3.4.2. Cambiar password.
 - 3.5. Vista Tabla.

- 3.5.1. Crear nueva tabla.
- 3.5.2. Cambiar nombre.
- 3.5.3. Poder eliminar tabla.
- 3.5.4. Agregar, modificar, eliminar columnas.
- 3.5.5. Poder guardar cambios.
- 3.5.6. Poder guardar como una tabla nueva.
- 3.6. Crear nuevo esquema.
 - 3.6.1. Todos los esquemas creados por el usuario.
 - 3.6.2. Que se pueda buscar.
- 3.7. Vista Esquema.
 - 3.7.1. Cambiar nombre.
 - 3.7.2. Poder eliminar esquema.
 - 3.7.3. Agregar, modificar y eliminar tablas.
 - 3.7.4. Ayudas visuales para facilitar la validación del diagrama ingresado.
 - 3.7.5. Poder guardar cambios.
 - 3.7.6. Poder guardar como un esquema nuevo.
 - 3.7.7. Botón para seleccionar script único o formato ANT.
 - 3.7.8. Botón para descargar scripts.
- 3.8. Que la aplicación valide los campos llenados en los formularios.

B. Tareas planificadas

Lista de Tareas planificadas para las iteraciones uno a seis con sus respectivos pesos en semanas.

Diseño de arquitectura del sistema.	7 sem.
Estudio y análisis el problema a solucionar e investigación de los distintos motores.	2 sem.
Modelo la solución mediante un diagrama de clases.	5 sem.
Modelo de una base de datos relacional.	2 sem.
Modelo del sistema de migraciones para distintas sintaxis, creación de scripts, poblado de datos y gestion de usuarios..	3 sem.
Diseño de base de datos mediante diagrama EER.	2 sem.
Diseño de interfaz web.	3 sem.
Diseño de wireframes para definir layouts preliminares de la interfaz.	1 sem
Diseño de mock-up en HTML, para ser reutilizados en la construcción del software	2 sem.
Construcción del sistema usando ASP.NET y MySQL.	25 sem.
Comparación de las sintaxis de los distintos motores para la creación de scripts.	1 sem
Investigación sobre algoritmos o métodos para generar datos.	1 sem
Construcción de base de datos.	2 sem.
Construcción del modelo.	6 sem.
Construcción de la vista.	16 sem.
Construcción de los controladores.	10 sem.
Pruebas del sistema	4 sem.

Figura B.1: Fragmento de planificación de tareas del proyecto.

C. Evaluación del funcionamiento

C.1. MySQL

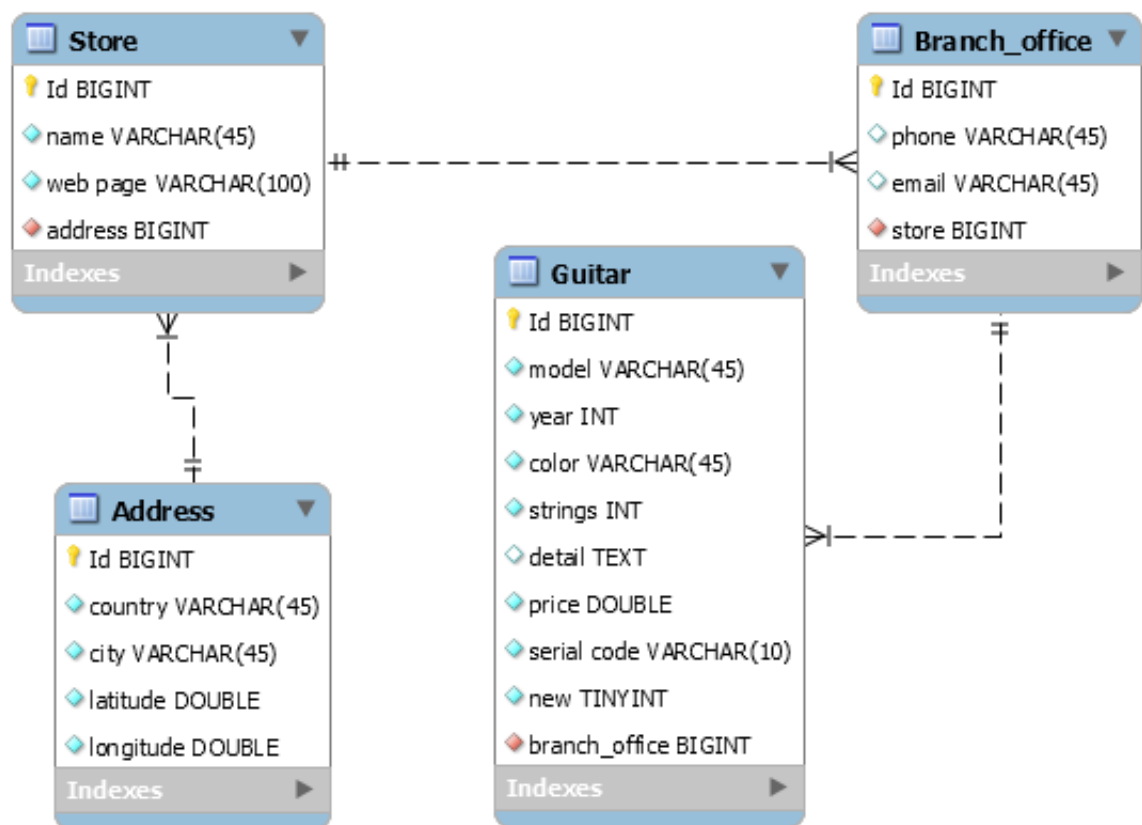


Figura C.1: Diagrama EER utilizado como referencia para llevar a cabo la evaluación del sistema.

The screenshot displays the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows a tree view for the 'guitar_catalogue' database, including tables (address, branch_office, guitar, store), views, and stored procedures. The main window shows a SQL script with 60 rows of SQL commands. The 'Output' pane at the bottom displays the execution results for these commands.

#	Time	Action	Message
45	23:05:59	CREATE PROCEDURE 'sp_delete_Branch_office'('Id' BIGINT UNSIGNED) BEGIN DELE...	0 row(s) affected
46	23:05:59	DROP PROCEDURE IF EXISTS 'sp_create_Guitar'	0 row(s) affected,
47	23:05:59	CREATE PROCEDURE 'sp_create_Guitar'('model' VARCHAR(45), 'year' INT, 'color' VAR...	0 row(s) affected,
48	23:05:59	DROP PROCEDURE IF EXISTS 'sp_read_Guitar'	0 row(s) affected,
49	23:05:59	CREATE PROCEDURE 'sp_read_Guitar'('Id' BIGINT UNSIGNED) BEGIN SELECT 'Id', '...	0 row(s) affected
50	23:05:59	DROP PROCEDURE IF EXISTS 'sp_update_Guitar'	0 row(s) affected,
51	23:05:59	CREATE PROCEDURE 'sp_update_Guitar'('Id' BIGINT UNSIGNED, 'model' VARCHAR(45)...	0 row(s) affected
52	23:05:59	DROP PROCEDURE IF EXISTS 'sp_delete_Guitar'	0 row(s) affected,
53	23:05:59	CREATE PROCEDURE 'sp_delete_Guitar'('Id' BIGINT UNSIGNED) BEGIN DELETE FRO...	0 row(s) affected,
54	23:05:59	DROP PROCEDURE IF EXISTS 'sp_Address_Stores_address'	0 row(s) affected,
55	23:05:59	CREATE PROCEDURE 'sp_Address_Stores_address'('address' BIGINT UNSIGNED) BEGI...	0 row(s) affected
56	23:06:00	DROP PROCEDURE IF EXISTS 'sp_Store_Branch_offices_store'	0 row(s) affected,
57	23:06:00	CREATE PROCEDURE 'sp_Store_Branch_offices_store'('store' BIGINT UNSIGNED) BEGI...	0 row(s) affected
58	23:06:00	DROP PROCEDURE IF EXISTS 'sp_Branch_office_Guitars_branch_office'	0 row(s) affected,
59	23:06:00	CREATE PROCEDURE 'sp_Branch_office_Guitars_branch_office'('branch_office' BIGINT U...	0 row(s) affected
60	23:06:00	INSERT INTO 'Address' VALUES (0, 'Israel', 'North Las Vegas', 13.3844673425818, 81.6...	100 row(s) affecte

Figura C.2: Captura de pantalla de la DB creada en MySQL utilizando Workbench.

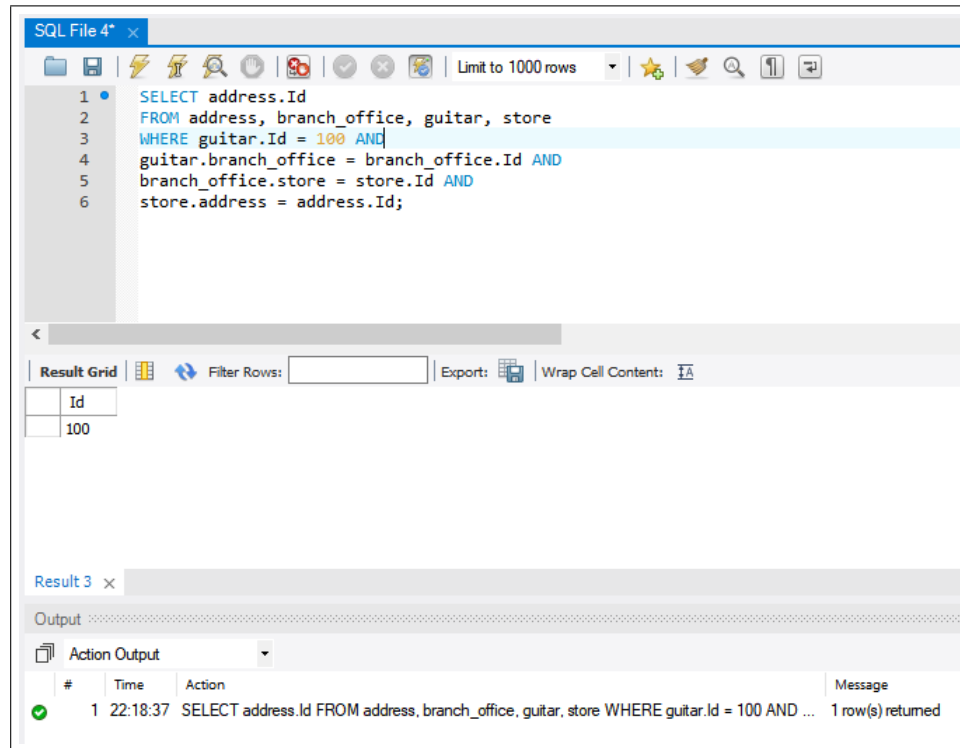


Figura C.3: Captura de pantalla de consulta de selección en múltiples tablas en MySQL.

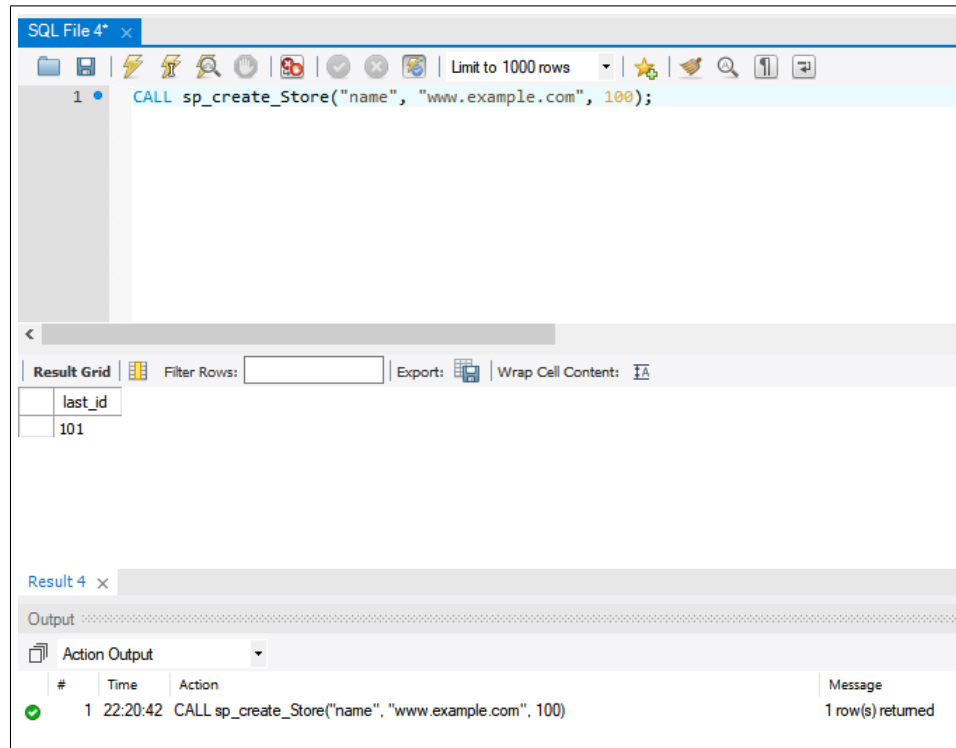


Figura C.4: Captura de pantalla de consulta de inserción en MySQL.

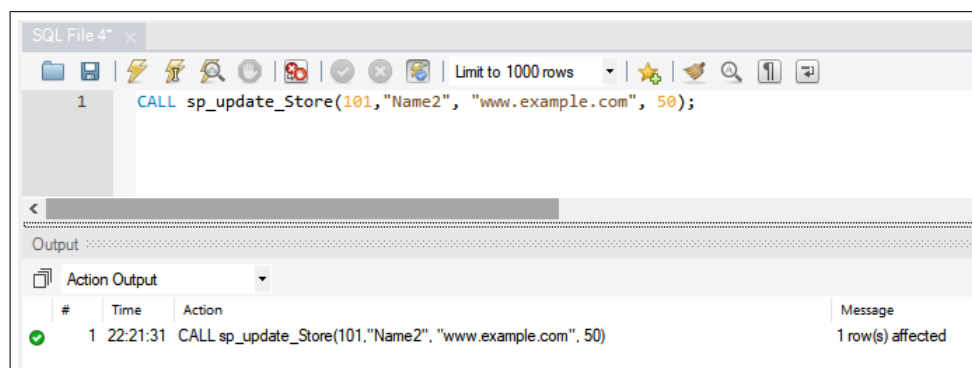


Figura C.5: Captura de pantalla de consulta de actualización en MySQL.

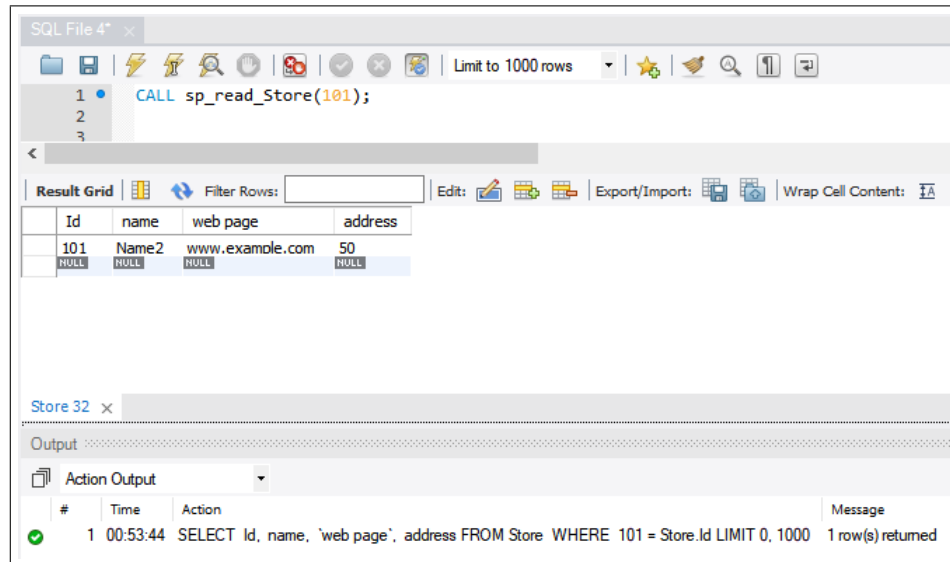


Figura C.6: Captura de pantalla de consulta de selección en MySQL.

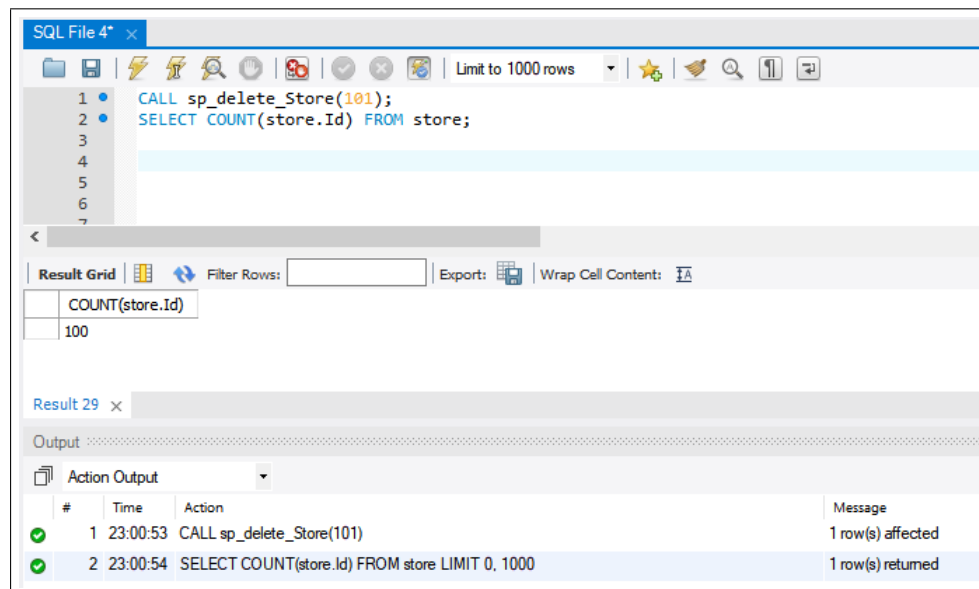


Figura C.7: Captura de pantalla de consulta de eliminación y su demostración en MySQL.

C.2. SQLite

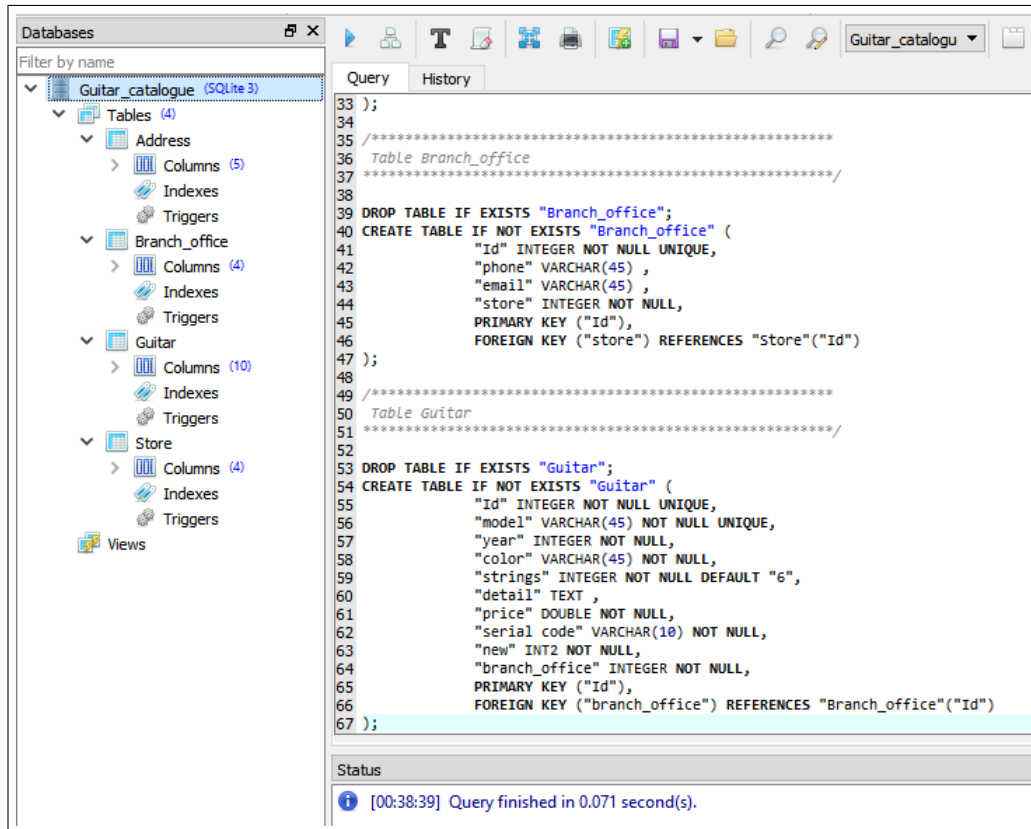


Figura C.8: Captura de pantalla de la DB creada en SQLite utilizando SQLiteStudio.

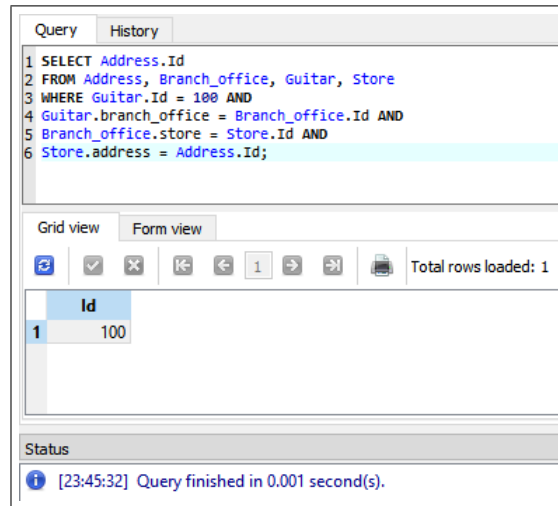


Figura C.9: Captura de pantalla de consulta de selección en múltiples tablas en SQLite.

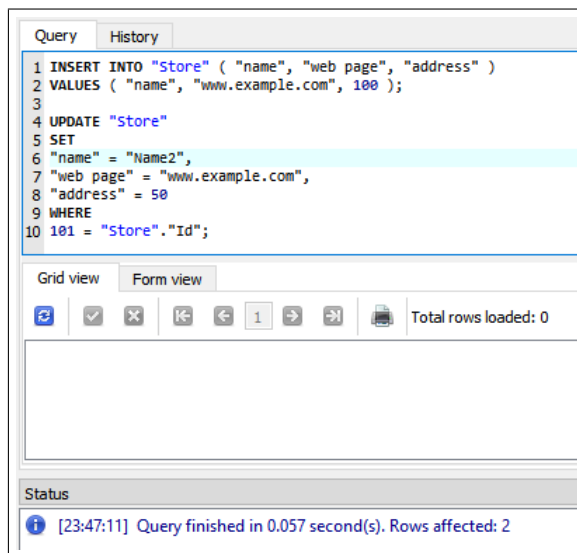
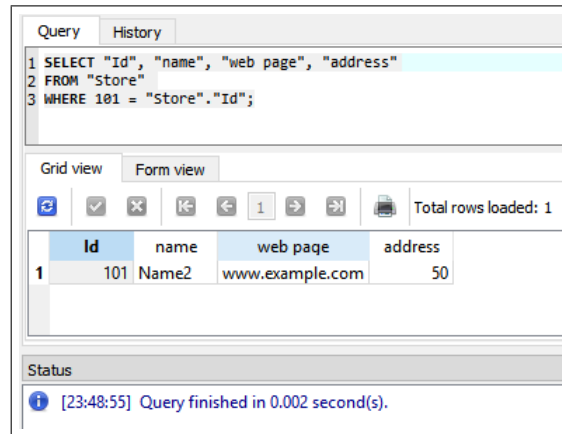


Figura C.10: Captura de pantalla de consulta de inserción y actualización en SQLite.



The screenshot shows a SQLite query tool interface. The 'Query' tab is active, displaying the following SQL query:

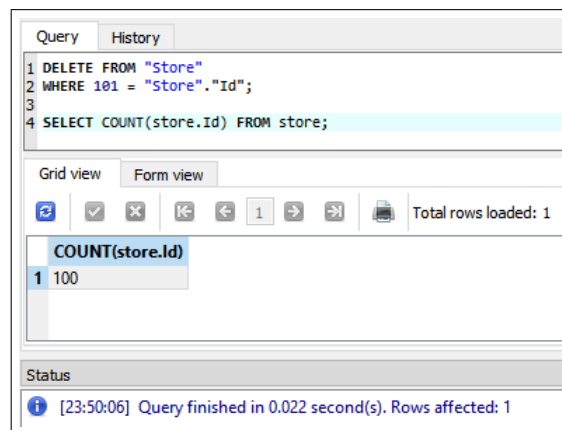
```
1 SELECT "Id", "name", "web page", "address"  
2 FROM "store"  
3 WHERE 101 = "Store"."Id";
```

Below the query, the 'Grid view' is selected, showing a table with the following data:

	Id	name	web page	address
1	101	Name2	www.example.com	50

The status bar at the bottom indicates: [23:48:55] Query finished in 0.002 second(s).

Figura C.11: Captura de pantalla de consulta de selección en SQLite.



The screenshot shows a SQLite query tool interface. The 'Query' tab is active, displaying the following SQL queries:

```
1 DELETE FROM "Store"  
2 WHERE 101 = "Store"."Id";  
3  
4 SELECT COUNT(store.Id) FROM store;
```

Below the queries, the 'Grid view' is selected, showing a table with the following data:

	COUNT(store.Id)
1	100

The status bar at the bottom indicates: [23:50:06] Query finished in 0.022 second(s). Rows affected: 1

Figura C.12: Captura de pantalla de consulta de eliminación y su demostración en SQLite.

C.3. Generación de scripts

1000_db_Guitar_catalogue.sql	285	116	Archivo SQL
2000_tb_Address.sql	587	228	Archivo SQL
2001_tb_Store.sql	563	223	Archivo SQL
2002_tb_Branch_office.sql	575	220	Archivo SQL
2003_tb_Guitar.sql	741	297	Archivo SQL
3000_fk_Store.sql	341	158	Archivo SQL
3001_fk_Branch_office.sql	355	165	Archivo SQL
3002_fk_Guitar.sql	374	166	Archivo SQL
4000_sp_create_Address.sql	528	225	Archivo SQL
4001_sp_read_Address.sql	381	182	Archivo SQL
4002_sp_update_Address.sql	585	241	Archivo SQL
4003_sp_delete_Address.sql	324	151	Archivo SQL
4004_sp_create_Store.sql	474	221	Archivo SQL
4005_sp_read_Store.sql	355	172	Archivo SQL
4006_sp_update_Store.sql	513	224	Archivo SQL
4007_sp_delete_Store.sql	314	150	Archivo SQL
4008_sp_create_Branch_office.sql	493	222	Archivo SQL
4009_sp_read_Branch_office.sql	391	179	Archivo SQL
4010_sp_update_Branch_office.sql	572	225	Archivo SQL
4011_sp_delete_Branch_office.sql	354	159	Archivo SQL
4012_sp_create_Guitar.sql	763	306	Archivo SQL
4013_sp_read_Guitar.sql	442	220	Archivo SQL
4014_sp_update_Guitar.sql	849	330	Archivo SQL
4015_sp_delete_Guitar.sql	319	152	Archivo SQL
4016_sp_Address_Stores_address.sql	413	185	Archivo SQL
4017_sp_Store_Branch_offices_store.sql	431	192	Archivo SQL
4018_sp_Branch_office_Guitars_branch_office.sql	554	233	Archivo SQL
9000_ins_Address.sql	6.904	3.511	Archivo SQL
9001_ins_Store.sql	7.804	3.006	Archivo SQL
9002_ins_Branch_office.sql	5.914	2.833	Archivo SQL
9003_ins_Guitar.sql	20.365	3.668	Archivo SQL

Figura C.13: Archivos de scripts SQL con números iniciales en sus nombres para procurar orden en la ejecución de estos.