ISBN 978 -987-633-574-4

Un Análisis Experimental de Sistemas de Gestión de Bases de Datos para Dispositivos Móviles

Fernando Tesone, Pablo Thomas, Luciano Marrero, Verena Olsowy, Patricia Pesado

Instituto de Investigación en Informática LIDI,
Facultad de Informática, Universidad Nacional de La Plata
La Plata, Argentina
{ftesone,pthomas,lmarrero,volsowy,ppesado}@lidi.info.unlp.edu.ar

Resumen Con el crecimiento en el alcance y uso de internet, de los smartphones, y de las redes sociales, se está produciendo un aumento exponencial en el volumen de datos administrados, pudiendo ser éstos estructurados, semiestructurados, o sin estructura. En este contexto surgen las bases de datos NoSQL, que facilitan el almacenamiento de datos semiestructurados o sin estructura.
Por otra parte, las mejoras en las prestaciones de hardware de los dispositivos móviles conducen a que éstos administren cada vez más información, y que surjan nuevos sistemas de gestión de bases de datos que se instalan en dichos dispositivos. Este trabajo tiene por objetivo realizar un relevamiento de los sistemas de gestión de bases de datos para dispositivos móviles, y realizar un análisis experimental de los sistemas más representativos de los modelos de bases de datos más utilizados.

Palabras clave: Bases de Datos para Dispositivos Móviles, DBMS Relacional, DBMS NoSQL

1. Introducción

Los sistemas de gestión de bases de datos (DBMS, por su sigla en inglés correspondiente a *Database Management System*) jugaron un rol fundamental en el desarrollo de software desde su surgimiento en la década de 1960, ya que proveían una forma eficiente de generar aplicaciones complejas, al eliminar la necesidad de programar la persistencia y el acceso a los datos [1,2].

En 1970 Edgar Codd desarrolla el modelo de base de datos relacional, que a partir de entonces, y hasta la actualidad, se volvió el modelo dominante [1,3].

Se presentan en 2007 el *Apple iPhone* y en 2008 el sistema operativo *Android*, hechos que cambiarían radicalmente la industria de los *smartphones*, ya que incrementaron la popularidad del uso de los dispositivos móviles, llegando a ser usados por el 80 % de la población en algunos países [4,5,6]. Este crecimiento en el uso traería acoplado una diversificación de plataformas. Para maximizar la presencia en el mercado, las *apps* deben estar disponibles en múltiples plataformas o sistemas operativos, por lo que los desarrolladores de software deben

ISBN 978 -987-633-574-4

optar por realizar desarrollos nativos, específicos de cada plataforma, o desarrollos multiplataforma [7,8].

Con el crecimiento en el alcance y uso de internet y de los dispositivos móviles, sumado a la aparición de las redes sociales, se está produciendo un crecimiento exponencial en el volúmen de datos administrados [9], pudiendo ser éstos estructurados, semi–estructurados o sin estructura. Ante esta situación de crecimiento en el volúmen de información, surgen las bases de datos no relacionales o NoSQL (Not-only SQL) como alternativa a las bases de datos relacionales, que facilitan el almacenamiento masivo de datos semi–estructurados o no estructurados.

Con la aparición de estas tecnologías, los desarrolladores de software deben analizar cuáles DBMSs son adecuados para las necesidades del problema a resolver.

El objetivo de este trabajo es (1) realizar un relevamiento de los DBMSs existentes, tanto relacionales como no relacionales, para dispositivos móviles —es decir, que pueden ser embebidos en aplicaciones para dispositivos móviles—, (2) seleccionar DBMSs que se consideren representativos del conjunto relevado, a partir de la definición de una serie de criterios, y (3) realizar un experimento que permita análizar, para cada DBMS seleccionado, características específicas, ventajas y desventajas, desde el punto de vista de la experiencia del ingeniero de software.

Este trabajo se organiza del siguiente modo: en la Sección 2 se discuten los trabajos relacionados; en la Sección 3 se presenta un relevamiento de sistemas de gestión de bases de datos para dispositivos móviles; en la Sección 4 se seleccionan DBMSs representativos, y se realiza una experimentación que permite analizar características específicas, y ventajas y desventajas de cada DBMS seleccionado. Posteriormente, en la Sección 5 se analizan los resultados obtenidos a partir de la experimentación realizada. Finalmente, en la Sección 6 se presentan las conclusiones y se definen posibles líneas de investigación como trabajo futuro.

2. Trabajos Relacionados

En esta sección se describen los trabajos relacionados encontrados, vinculados al tema presentado.

En [10] se expone un listado de características que los sistemas de gestión de bases de datos móviles y embebibles deben tener: ser distribuídos junto con la aplicación; minimizar el uso de memoria principal y secundaria; permitir incluir sólo los componentes del DBMS necesarios; soportar el almacenamiento en memoria principal; ser portables; ejecutarse en dispositivos móviles, y sincronizar datos con DBMSs de backend.

En [11] se realiza un relevamiento de diferentes opciones de almacenamiento en dispositivos móviles, siendo éstas: HTML5, a partir de la utilización del framework WebKit, posibilitando el uso de la API *localStorage*; SQLite, una librería que encapsula funcionalidad SQL y almacena la información en un archivo local; almacenamiento en la nube, utilizando servicios como *Apple iCloud*, *Google Drive*, *Dropbox*, *Amazon S3*; almacenamiento específico del dispositivo, como son *Shared Preferences* en Android y *Core Data* en iOS, sumados a las opciones anteriores, presentes en ambos sistemas.

ISBN 978 -987-633-574-4

En [12] se realiza una descripción de la arquitectura de la plataforma Android y se analiza, entre otras cuestiones, la arquitectura y la forma en que las aplicaciones se ejecutan. Finalmente, se hace una introducción al uso de SQLite en Android.

En [13] se analizan ventajas y desventajas del uso de computación en la nube de forma integral con las aplicaciones móviles, mencionando como ventajas el almacenamiento, los respaldos (*backups*), y la redundancia de datos, entre otras.

En los trabajos previamente descriptos, si bien se analizan temas relacionados al almacenamiento de datos en dispositivos móviles, no se establece un análisis de los distintos sistemas de gestión de bases para dispositivos móviles que permita al ingeniero de software seleccionar el DBMS más adecuado a utilizar para resolver un problema determinado, motivo por el cual se pretende cubrir este aspecto con la presentación de este artículo.

3. Bases de Datos para Dispositivos Móviles

Se presenta un relevamiento de DBMSs relacionales (RDBMSs, por su sigla en inglés correspondiente a *Relational Database Management System*) y DBMSs NoSQL que pueden utilizarse embebidos en aplicaciones móviles. La búsqueda de DBMSs a analizar se realizó a través de los buscadores Google y Google Scholar, utilizando los términos "mobile database", "mobile dbms", entre otros. También se realizó una búsqueda en Google con el término "mobile site:db-engines.com/en/system", ya que el sitio DB-Engines utiliza ese *path* para las páginas en las que se describen características de cada DBMS relevado por el sitio.

DB-Engines se trata de un proyecto que busca recopilar y presentar información sobre DBMSs, creado y mantenido por solidIT, una compañía austríaca especializada en el desarrollo de software, consultoría y formación para aplicaciones centradas en datos. El sitio elabora un ranking mensual a partir de la puntuación que obtenga cada DBMS relevado según su popularidad, definida por diferentes parámetros [3].

3.1. DBMSs Relacionales

Los DBMSs relacionales o RDBMSs existentes para dispositivos móviles, ordenados de acuerdo al ranking elaborado por DB-Engines [3] son:

1. SQLite
2. Interbase
3. SAP SQL Anywhere
4. SQLBase

SQLite es una librería que implementa un motor de base de datos autocontenido (embebido). Tiene licencia Public Domain, y puede utilizarse tanto en el desarrollo nativo de aplicaciones, ya sea en Android o en iOS, como también en el desarrollo de aplicaciones multiplataforma en los enfoques híbrido, interpretado o de compilación cruzada [14].

Interbase, un RDBMS embebido, con licencia comercial, y conforme al estándar SQL. En cuanto a su uso en el desarrollo de aplicaciones móviles, se puede utilizar en el desarrollo nativo en Android y iOS [15].

ISBN 978 -987-633-574-4

SAP SQL Anywhere integra un paquete de DBMSs relacionales y tecnologías de sincronización para servidores, en entornos de escritorio y móviles [16]. Se encuentra disponible para utilizar en el desarrollo de aplicaciones móviles nativas, tanto en Android como en iOS.

SQLBase es un RDBMS desarrollado por la empresa Opentext. Tiene licencia de uso comercial. Se encuentra disponible para el desarrollo de aplicaciones móviles nativas, en Android y iOS [17].

3.2. DBMSs NoSQL

El término NoSQL se utiliza para denominar a bases de datos de distintos modelos, diferentes al modelo relacional. Dentro de los distintos modelos de bases de datos NoSQL existentes, el modelo documental representa al más utilizado en la actualidad [3]. De los DBMSs NoSQL existentes para dispositivos móviles, la mayoría corresponde al modelo documental.

Los DBMSs NoSQL para dispositivos móviles existentes, ordenados de acuerdo al ranking DB-Engines son:

1. Couchbase Lite (Documental)
2. Firebase Realtime Database (Documental)
3. Realm (Documental)
4. Google Cloud Firestore (Documental)
5. Oracle Berkeley DB (Clave-valor)
6. PouchDB (Documental)
7. LiteDB (Documental)
8. ObjectBox (Orientada a objetos)
9. Sparksee (Grafos)

DBMSs Documentales

Couchbase Lite es un DBMS embebido, que utiliza JSON como formato de los documentos. Al formar parte del paquete Couchbase, es posible utilizar el sistema *Sync Gateway* para sincronizar datos con bases de datos remotas [18]. Tiene licencia dual, y es posible utilizarlo para el desarrollo de aplicaciones móviles nativas Android y iOS, y multiplataforma en los enfoques híbrido, interpretado, y de compilación cruzada [3].

Firebase Realtime Database es un DBMS alojado en la nube, que almacena los datos en un único JSON, y cuenta con sincronización de datos en tiempo real con todos los clientes conectados, manteniéndose disponibles aún sin conexión. Se encuentra disponible para su uso en el desarrollo de aplicaciones nativas en Android y en iOS, y en el desarrollo de aplicaciones móviles multiplataforma en enfoques híbridos, interpretados, y de compilación cruzada [19].

Realm es un DBMS embebido que utiliza un modelo de datos orientado a objetos. Es posible utilizarlo de forma autónoma, o de forma sincronizada con una base de datos *backend* MongoDB. Se encuentra disponible para el desarrollo de aplicaciones nativas Android y iOS, y el desarrollo de aplicaciones móviles multiplataforma en enfoques interpretado y de compilación cruzada [3,20].

Google Cloud Firestore es un DBMS alojado en la nube, que almacena los datos en documentos JSON y cuenta con sincronización de datos en tiempo

ISBN 978 -987-633-574-4

real con los clientes, manteniéndo los datos disponibles aún sin conexión. Es posible utilizarlo en el desarrollo de aplicaciones móviles nativas Android y iOS, y en el desarrollo de aplicaciones móviles multiplataforma con enfoques híbrido, interpretado, y de compilación cruzada [21].

PouchDB es una librería javascript que implementa un DBMS inspirado en CouchDB, y que utiliza su protocolo de sincronización. Es distribuido bajo licencia Apache 2.0 y se encuentra disponible para el desarrollo de aplicaciones móviles multiplataforma bajo los enfoques híbrido e interpretado [22].

LiteDB es una librería distribuida como un único archivo DLL bajo licencia MIT. Utiliza documentos BSON para almacenar la información. Se encuentra disponible para el desarrollo de aplicaciones móviles multiplataforma en Xamarin (compilación cruzada) [23].

DBMSs de otros tipos

Oracle Berkeley DB es una familia de bases de datos de clave-valor embebidas. Tiene licencia open source y se encuentra disponible para el desarrollo de aplicaciones nativas Android y iOS [3,24].

ObjectBox es un DBMS orientado a objetos para dispositivos móviles e IoT. Tiene licencia Apache 2.0, y se encuentra disponible para el desarrollo de aplicaciones móviles nativas en Android y iOS, y multiplataforma en Flutter. Cuenta con un servicio de sincronización y almacenamiento en la nube [3,25].

Sparksee es un DBMS de grafos. Se distribuye bajo licencia comercial y cuenta con licencias gratuitas para educación e investigación. Se puede utilizar en el desarrollo de aplicaciones nativas en Android y iOS [3,26].

4. Experimentación

Para la realización de la experimentación se seleccionan DBMSs bajo las siguientes condiciones, definidas para este trabajo: poder ser utilizado sin una licencia comercial; poder ser utilizado de forma autónoma, completamente offline; contar con herramientas para realizar sincronización con bases de datos remotas; poder ser utilizado en aplicaciones móviles desarrolladas de forma nativa en las principales plataformas (Android y iOS); poder ser utilizado en aplicaciones móviles desarrolladas con diferentes enfoques multiplataforma en frameworks de desarrollo conocidos (React Native, NativeScript, Ionic Framework, Xamarin, Flutter); que esté en las primeras tres posiciones de acuerdo al ranking por modelo de DBMS elaborado por DB-Engines.

A partir de las características que presenta cada DBMS, y de los criterios enunciados previamente, se seleccionan para la realización del análisis experimental comparativo SQLite, Couchbase Lite y Realm.

Para llevar a cabo el análisis se desarrollaron parcialmente tres aplicaciones móviles en la plataforma Android, cada una utilizando los DBMSs seleccionados. La aplicación consiste en una agenda de contactos que cumple con los siguientes requerimientos:

Requerimientos funcionales:

R1 La aplicación debe permitir crear un nuevo contacto, con los siguientes datos: Apellido (requerido); Nombre (requerido); Fecha de nacimiento (opcional);

ISBN 978 -987-633-574-4

Emails (cero o más); Teléfonos (cero o más; para cada Teléfono se almacena: Número —requerido— y Tipo —requerido, se debe seleccionar entre las siguientes opciones: Móvil, Casa, Trabajo, Otro—)

R2 La aplicación debe listar los contactos almacenados ordenados por apellido y nombre de forma ascendente.

R3 La aplicación debe permitir filtrar los contactos almacenados, a partir de un único término de búsqueda, listando los contactos que coincidan parcial o totalmente con el término de búsqueda en algunos de sus campos. Los contactos filtrados se mostrarán ordenados por Apellido y Nombre ascendentemente.

Requerimientos no funcionales:

R4 Toda la información debe almacenarse de forma local en la base de datos.

Para el desarrollo de la aplicación se define un único diagrama correspondiente al modelo conceptual de base de datos, utilizando el Modelo Entidad-Relación, que luego se deriva a los modelos lógico y/o físico correspondientes según cada modelo de base de datos y DBMS.

El análisis se realizará desde el punto de vista de la experiencia del ingeniero de software en el desarrollo de cada aplicación, considerando la complejidad de implementación para la utilización del DBMS correspondiente, en cuanto a factores como instalación/configuración, implementación de componentes requeridos —por ejemplo, estructuras de datos, clases—, definición/ejecución de consultas, entre otros.

Teniendo en cuenta el objetivo del análisis no se considera relevante que el esquema de datos definido se adapte mejor a un modelo de base de datos específico, ya que el interés está focalizado en analizar la implementación de las diferentes características específicas de cada modelo de base de datos.

Los requerimientos y esquemas de datos definidos tienen por objetivo la implementación de las características específicas de los modelos de bases de datos seleccionados: *relaciones* en el modelo relacional; *documentos embebidos*, *arrays de tipos escalares*, y *arrays de documentos embebidos* en el modelo NoSQL documental.

4.1. SQLite

Para la experimentación utilizando SQLite como DBMS, se deriva el diagrama correspondiente al modelo conceptual al modelo físico (Figura 1)

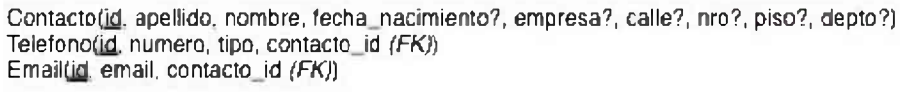

Figura 1: Diagrama correspondiente al modelo físico para SQLite

Para la utilización de SQLite en el desarrollo de aplicaciones móviles, Android provee dos formas de gestionar bases de datos en dicho DBMS. La primera

ISBN 978 -987-633-574-4

de ellas (recomendada por la documentación oficial) es utilizando Room, una biblioteca de persistencia que funciona a modo de capa de abstracción de SQLite; la segunda es utilizando la API de SQLite directamente [27]. La forma elegida para la experimentación es la primera.

Instalación/Configuración La instalación de Room se realiza agregando las dependencias necesarias en el archivo *gradle*.

Definición del esquema de datos El uso de Room se lleva a cabo definiendo clases e interfaces de objetos a los que se les deben definir determinadas anotaciones.

La biblioteca cuenta con tres componentes principales: *entidades*, que son clases que representan a las entidades del modelo, y que representan las tablas del modelo relacional. Estas clases deben anotarse con `@Entity`.

El segundo componente principal de Room son los *DAO*s, definidos a partir de interfaces con la anotación `@Dao`, que son utilizados para realizar consultas sobre la tabla que representa cada entidad, permitiendo obtener entidades, modificarlas, crearlas y eliminarlas.

El tercer componente es la *base de datos*, que sirve como punto de acceso principal para la conexión subyacente a la base de datos relacional. Se debe definir una clase abstracta que extienda de la clase `RoomDatabase` y que esté anotada con `@Database`.

En las relaciones entre las diferentes entidades deben anotarse las restricciones de clave foránea, indicando para cada una la entidad y propiedad/es que referencia, y la propiedad sobre la que aplica.

Debido a la imposibilidad de referenciar objetos en Room, las propiedades cuyo tipo no correspondan a un tipo de dato escalar (tipos numéricos, *strings*, y booleano) deben ser convertidas para poder ser almacenadas en la base de datos. Para ello deben definirse dos métodos de conversión por cada tipo de dato que desee persistirse.

Inserción de datos Para satisfacer el requerimiento funcional *R1* deben insertarse las tuplas correspondientes al contacto a crear, y a los teléfonos y emails del contacto. Para ello se deben definir métodos en las interfaces correspondientes a los *DAOs* anotados con `@Insert`.

Recuperación de datos La aplicación desarrollada en el marco de la experimentación debe listar todos los contactos almacenados (*R2*), y los contactos que coincidan parcial o totalmente con un término de búsqueda (*R3*). Para consultas de recuperación de tuplas deben definirse métodos en la interfaz correspondiente al *DAO* anotadas con `@Query`, cuyo valor de la anotación es la consulta SQL. Para filtrar a partir de un término de búsqueda es necesario parametrizar éste, lo que se logra definiendo un parámetro en el método, cuyo valor es posible referenciar en la consulta prefijando con dos puntos el nombre del parámetro.

En los casos que sea necesario recuperar información de múltiples tablas interrelacionadas, se debe implementar una clase con propiedades cuyo tipo corresponda a las entidades involucradas.

Código fuente El código fuente de la experimentación se encuentra en `https://github.com/ftesone/tesina-room/tree/clei-2021`.

ISBN 978 -987-633-574-4

4.2. Couchbase Lite

Una característica particular de las bases de datos documentales es que ofrecen la posibilidad de almacenar datos sin estructura o semiestructurados. Asimismo, es posible definir parcial o totalmente un esquema para los documentos.

Para la implementación no se define un esquema para la base de datos, pero sí se estructuran los documentos según el modelo físico de la Figura 2.

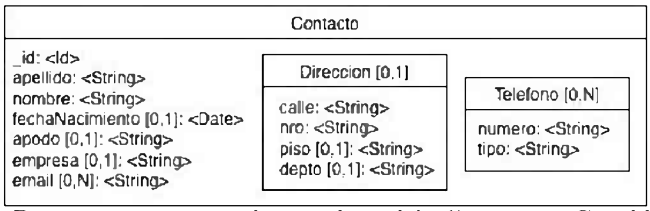

Figura 2: Diagrama correspondiente al modelo físico para Couchbase Lite

Instalación/Configuración La instalación de Couchbase Lite se realiza agregando las dependencias necesarias en el archivo *gradle*.

Definición del esquema de datos La administración de datos se realiza utilizando clases ya definidas por Couchbase Lite, entre las que se encuentran `MutableDocument`, `MutableArray`, `MutableDictionary`, entre otras. Objetos de estas clases se utilizan para definir la estructura de cada documento a almacenar en la base de datos; cada documento puede tener una estructura única.

Si bien es posible definir una estructura de clases para realizar una implementación orientada a objetos, debe también implementarse la hidratación de los objetos.

Inserción de datos Para realizar altas de documentos se deben crear instancias de la clase `MutableDocument` y definir los campos del documento (los campos opcionales que no tienen un valor no se definen).

Para embeber documentos se utiliza la clase `MutableDictionary`, ya sean éstos correspondientes a un único documento o a un array de documentos.

En el caso de los arrays, ya sea de documentos embebidos o de tipos de datos escalares, para la definición de éstos se debe utilizar la clase `MutableArray`. Por lo tanto, un array de documentos embebidos se define como un `MutableArray` cuyos valores son instancias de `MutableDictionary`.

Recuperación de datos Para realizar consultas de recuperación de información se debe utilizar un objeto de la clase `QueryBuilder`, al que deben enviarse mensajes para definir expresiones que permitan establecer condiciones sobre propiedades de los documentos, criterios de orden, entre otros. Los tipos de datos devueltos por la ejecución de consultas son variantes inmutables de los utilizados para inserciones (`Dictionary`, `Array`; los documentos se encapsulan en objetos de clase `Result`).

Código fuente El código fuente de la experimentación se encuentra en `https://github.com/ftesone/tesina-couchbase/tree/clei-2021`.

ISBN 978 -987-633-574-4

4.3. Realm

Para la implementación de Realm se utiliza el mismo modelo físico definido para Couchbase Lite, presentado en la Figura 2, ya que, si bien hay diferencias entre los dos DBMS, las estructuras definidas en el modelo también aplican para Realm.

Instalación/Configuración La instalación de Realm se realiza agregando las dependencias necesarias en el archivo *gradle*.

Definición del esquema de datos La utilización de Realm se lleva a cabo definiendo clases que representen los documentos del modelo, que deben definirse como subclases de `RealmObject` o que implementen la interfaz `RealmModel`, ya sea que se traten de documentos embebidos o no.

Para los documentos *top-level* se debe definir una propiedad con tipo `ObjectId` y anotarse con `@PrimaryKey`. Las clases correspodientes a documentos que se utilicen embebidos en otros documentos, ya sea directamente o como un array de documentos, deben anotarse con `@RealmClass(embedded = true)`.

En los casos que se definan documentos con arrays de valores de tipos de datos escalares o arrays de documentos, los tipos de las propiedades deben definirse como `RealmList<T>`, donde `T` es el tipo de dato escalar o la clase correspondiente al documento embebido.

Inserción de datos Las operaciones de escritura en la base de datos en Realm se realiza a través de transacciones, las cuales se definen invocando al método `executeTransaction` de la instancia de la base de datos, en donde se pasa como parámetro una expresión lambda en la que se define el comportamiento. En el cuerpo de la expresión debe obtenerse una instancia de la clase correspondiente al documento que se insertará y deben definirse los valores de las propiedades a persistir.

Recuperación de datos Para realizar consultas para recuperar documentos almacenados se debe obtener una instancia de `RealmQuery<T>` —donde `T` corresponde a la clase de documento que se desea recuperar—, invocando al método `where` de la instancia de la base de datos, pasando como parámetro el nombre de la clase `T`.

La clase `RealmQuery` permite invocar distintos métodos para filtrar y ordenar, entre otras, y obtener los documentos encapsulados en una instancia de `RealmResults<T>`.

Código fuente El código fuente de la experimentación se encuentra en `https://github.com/ftesone/android-realm/tree/clei-2021`.

5. Análisis de Resultados

5.1. SQLite

En la experimentación utilizando SQLite como DBMS para persistir la información se decidió utilizar la biblioteca Room, que representa una capa de abstracción sobre SQLite. En la implementación realizada se destacan como ventajas: (1) la estructura de clases que se requiere definir lleva a trabajar de una forma ordenada y sistemática; (2) la definición del esquema de la BD a partir de las clases que conforman las entidades del modelo, por lo que no es necesario

ISBN 978 -987-633-574-4

definirlo con sentencias SQL; (3) la flexibilidad que se provee para definir consultas para obtener información, ya que sólo se debe definir la consulta como valor de una anotación, incluídas las consultas parametrizadas; (4) la facilidad con la que se definen conversiones de tipos que no se pueden almacenar directamente en la base de datos; (5) la facilidad con la que se insertan tuplas.

Asimismo, se enumeran algunas desventajas en la utilización de Room: (1) la necesidad de utilizar estructuras auxiliares para obtener tuplas resultado de productos (*joins*) entre tablas; (2) la necesidad de definir un conversor para un tipo de dato ampliamente utilizado como es el tipo de dato *Date*.

5.2. Couchbase Lite

En la implementación realizada con Couchbase Lite, considerando la experimentación realizada, pueden observarse como ventajas: (1) la flexibilidad en la definición de la estructura de documentos, ya que ésta puede ser definida de forma completamente dinámica; (2) la utilización de estructuras definidas por Couchbase Lite para la persistencia y recuperación de información.

Pueden considerarse como desventajas: (1) no es posible agrupar los documentos en colecciones; (2) la implementación de consultas complejas requiere definir gran cantidad de código, lo que dificulta su comprensión; (3) la implementación de consultas que referencien valores en arrays es compleja debido a la necesidad de definir variables que hagan referencia a cada valor del array; (4) la necesidad de implementar la hidratación de objetos en caso de desarrollar la aplicación utilizando el paradigma de programación orientada a objetos;

5.3. Realm

La implementación realizada con Realm permite enumerar las siguientes ventajas: (1) la definición de la estructura de los documentos a través de la definición de clases; (2) la posibilidad de agrupar documentos en colecciones; (3) la implementación de consultas resulta concisa y clara, especialmente porque no es necesario definir variables para referenciar a campos en documentos embebidos o en arrays de documentos embebidos.

Asimismo, se encontró como desventaja la imposibilidad de referenciar a valores correspondientes a arrays de tipos escalares en las consultas.

El Cuadro 1 resume los aspectos evaluados en SQLite, Couchbase Lite y Realm. Cada aspecto fue calificado en una escala de cinco valores: *muy bajo*, *bajo*, *medio*, *alto*, *muy alto*. Los aspectos evaluados son:

- Complejidad: expresa el nivel de dificultad para realizar la tarea;
- Flexibilidad: aplica sólo a la definición del esquema de datos. Se refiere a cuán flexible es el esquema para adaptarse a cambios;
- Legibilidad del código: expresa el nivel de dificultad para comprender el código fuente;
- Integración con POO (Programación Orientada a Objetos): nivel de interacción de los tipos de datos del DBMS con los objetos definidos en la aplicación;
- Soporte de tipos de datos no escalares: nivel de dificultad en la persistencia de tipos de datos no escalares, es decir, tipos de datos distintos a tipos numéricos, *strings*, y booleanos.

ISBN 978 -987-633-574-4

La mejor calificación posible es *muy alto* para todos los aspectos analizados, a excepción de Complejidad, cuya mejor calificación es *muy baja*.

Tarea	Aspectos	SQLite	Couchbase Lite	Realm
Instalación/Configuración	Complejidad	Muy baja	Muy baja	Muy baja
Definición del esquema de datos	Complejidad	Baja	Muy baja	Muy baja
	Flexibilidad	Baja	Muy alta	Media
	Legibilidad del código	Alta	Muy baja	Muy alta
	Integración con POO	Alta	Muy baja	Alta
	Soporte de tipos de datos no escalares	Muy baja	Baja	Baja
Inserción de datos	Complejidad	Baja	Media	Baja
	Legibilidad de código	Muy alta	Alta	Alta
	Integración con POO	Muy alta	Baja	Alta
Recuperación de datos	Complejidad	Baja	Muy alta	Muy baja
	Legibilidad de código	Muy alta	Muy baja	Muy alta
	Integración con POO	Alta	Muy baja	Muy alta

Cuadro 1: Aspectos evaluados en los DBMSs

6. Conclusiones y Trabajo Futuro

Este trabajo intenta abordar la problemática que representa la elección de un DBMS adecuado que pueda ser embebido en una aplicación móvil, acorde al problema a resolver.

El aumento exponencial en el volúmen de datos administrados, incluyendo datos estructurados, semi-estructurados, y no estructurados, provocó el surgimiento de nuevos modelos de bases de datos, denominados bases de datos NoSQL, para facilitar el almacenamiento masivo de datos semi-estructurados y no estructurados.

Por otra parte, las mejoras en las prestaciones de hardware de los dispositivos móviles conducen a que éstos administren cada vez más información, y que surjan nuevos sistemas de gestión de bases de datos que se instalan en dichos dispositivos.

A raíz de esto, este trabajo analiza distintos aspectos referidos a la instalación y/o configuración, implementación de componentes requeridos, definición y/o ejecución de consultas de modificación y recuperación de datos, para asistir al ingeniero de software en la selección de un DBMS adecuado para ser embebido en aplicaciones móviles acorde al problema a resolver.

A partir de la experimentación y análisis realizados puede considerarse que el problema a resolver resulta determinante en la elección de un DBMS. SQLite y Realm serían los más adecuados dado que presentan una baja complejidad, alta legibilidad de código y alta integración con estructuras del paradigma de programación orientada a objetos, en las tareas correspondientes a definición del esquema de datos, inserción de datos, y recuperación de datos. Sin embargo, presentan una baja flexibilidad en la definición del esquema de datos, aspecto en el que Couchbase Lite sería más adecuado.

ISBN 978 -987-633-574-4

En situaciones en las que no se requiera una gran flexibilidad en el esquema de datos, SQLite y Realm serían los más adecuados. Por el contrario en situaciones en las cuales se requiera una gran flexibilidad en el esquema de datos, Couchbase Lite representaría una mejor opción. En la experimentación planteada, que no requiere un esquema de datos flexible, SQLite y Realm resultan más adecuados, considerándose el primero como la mejor opción, debido a que el problema planteado se ajusta mejor al modelo relacional.

Finalmente, se proponen varias líneas de investigación como posible trabajo futuro:

1. extender la experimentación realizada agregando el requerimiento no funcional de sincronización de la base de datos de la aplicación móvil con una base de datos *backend*;
2. analizar el impacto que produce la utilización de SQLite, Couchbase Lite o Realm en requerimientos no funcionales que resultan determinantes en el éxito de la aplicación, como el uso de memoria principal y de memoria secundaria, el rendimiento en la ejecución de consultas, y el consumo de energía;
3. extender el análisis realizado utilizando otros DBMSs, resultando particularmente interesante aquellos DBMSs que se encuentren disponibles para su uso tanto en Android como en iOS, y en frameworks conocidos para el desarrollo de aplicaciones móviles multiplataforma.

Referencias

1. K. L. Berg, T. Seymour, and R. Goel, "History of databases," *International Journal of Management & Information Systems (IJMIS)*, vol. 17, no. 1, pp. 29–36, 2013.
2. B. Grad and T. J. Bergin, "Guest editors' introduction: History of database management systems," *IEEE Annals of the History of Computing*, vol. 31, no. 4, pp. 3–5, 2009.
3. "Db-engines ranking - popularity ranking of database management systems." https://db-engines.com/en/ranking. Accedido por última vez: 17/02/2021.
4. M. Campbell-Kelly and D. D. Garcia-Swartz, *From mainframes to smartphones: a history of the international computer industry*, vol. 1. Harvard University Press, 2015.
5. "List of countries by smartphone penetration - wikipedia." https://en.wikipedia.org/wiki/List_of_countries_by_smartphone_penetration#2013_rankings. Accedido por última vez: 17/02/2021.
6. "• cell phone sales worldwide 2007-2020 | statista." https://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/. Accedido por última vez: 17/02/2021.
7. L. Delia, N. Galdamez, P. Thomas, L. Corbalan, and P. Pesado, "Multi-platform mobile application development analysis," in *2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS)*, pp. 181–186, IEEE, 2015.
8. S. Xanthopoulos and S. Xinogalos, "A comparative analysis of cross-platform development approaches for mobile applications," in *Proceedings of the 6th Balkan Conference in Informatics*, pp. 213–220, 2013.

ISBN 978 -987-633-574-4

9. L. Marrero, V. Olsowy, P. J. Thomas, L. N. Delía, F. Tesone, J. Fernández Sosa, and P. M. Pesado, "Un estudio comparativo de bases de datos relacionales y bases de datos nosql," in *XXV Congreso Argentino de Ciencias de la Computación (CACIC)(Universidad Nacional de Río Cuarto, Córdoba, 14 al 18 de octubre de 2019)*, 2019.
10. A. Nori, "Mobile and embedded databases," in *Proceedings of the 2007 ACM SIGMOD International Conference on Management of data*, pp. 1175–1177, 2007.
11. Q. H. Mahmoud, S. Zanin, and T. Ngo, "Integrating mobile storage into database systems courses," in *Proceedings of the 13th annual conference on Information technology education*, pp. 165–170, 2012.
12. S. Lee, "Creating and using databases for android applications," *International Journal of Database Theory and Application*, vol. 5, no. 2, 2012.
13. A. Alzahrani, N. Alalwan, and M. Sarrab, "Mobile cloud computing: advantage, disadvantage and open challenge," in *Proceedings of the 7th Euro American Conference on Telematics and Information Systems*, pp. 1–4, 2014.
14. "About sqlite." https://www.sqlite.org/about.html. Accedido por última vez: 21/02/2021.
15. "Interbase - embarcadero website." https://www.embarcadero.com/es/products/interbase. Accedido por última vez: 03/03/2021.
16. "Sap sql anywhere | rdbms for iot & data-intensive apps | technical information." https://www.sap.com/products/sql-anywhere/technical-information.html. Accedido por última vez: 03/03/2021.
17. "Opentext gupta sqlbase." https://www.opentext.com/products-and-solutions/products/specialty-technologies/opentext-gupta-development-tools-databases/opentext-gupta-sqlbase. Accedido por última vez: 03/03/2021.
18. "Lite | couchbase." https://www.couchbase.com/products/lite. Accedido por última vez: 21/02/2021.
19. "Firebase realtime database | firebase realtime database." https://firebase.google.com/docs/database. Accedido por última vez: 21/02/2021.
20. "Home | realm.io." https://realm.io/. Accedido por última vez: 22/02/2021.
21. "Cloud firestore | firebase." https://firebase.google.com/docs/firestore/. Accedido por última vez: 22/02/2021.
22. "Pouchdb, the javascript database that syncs!." https://pouchdb.com/. Accedido por última vez: 21/02/2021.
23. "Litedb :: A .net embedded nosql database." http://www.litedb.org/. Accedido por última vez: 03/03/2021.
24. "Oracle berkeley db." https://www.oracle.com/database/technologies/related/berkeleydb.html. Accedido por última vez: 03/03/2021.
25. "Mobile database | android database | ios database | flutter database." https://objectbox.io/mobile-database/. Accedido por última vez: 03/03/2021.
26. "Sparsity-technologies: Sparksee high-performance graph database." http://sparsity-technologies.com/#sparksee. Accedido por última vez: 03/03/2021.
27. "Descripción general del almacenamiento de archivos y datos." https://developer.android.com/training/data-storage. Accedido por última vez: 21/02/2021.