Comparação de desempenho de imagens no formato base64 e bit array utilizando banco de dados SQLite

Jhony Santos, Pedro Henrique B. Guaraldo, Kelton Costa

Curso de Tecnologia em Banco de Dados – Faculdade de Tecnologia de Bauru (FATEC)

Rua Manoel Bento Cruz, n° 30, Quadra 3 – Centro – 17.015-171 – Bauru, SP – Brasil

jhony.s.s@hotmail.com,pedroguaraldo@gmail.com,kelton.costa@gmail.com

Abstract. *In an increasingly technological world, smartphones are playing a key role in the lives of the users. Thanks to a strong interest in mobile phones, demands for new applications are increasing, requiring a quick delivery and better processing of the data generated by the applications. The Android platform has integrated in its native configuration the SQLite database, not requiring its installation. The purpose of this work is to perform a general performance monitoring of the SQLite DBMS, thus measuring the speed and behavior of some of the data manipulation processes, such as insert, delete, update, among others.*

Resumo. *Em um mundo cada vez mais tecnológico, os smartphones passaram a ter um papel fundamental na vida dos usuários. Graças a esse crescente interesse pelos aparelhos mobiles, a demanda por novos aplicativos está cada dia maior, sendo necessário uma rápida entrega e um melhor processamento dos dados gerados pelas aplicações. A plataforma Android possui integrada em sua configuração nativa o banco de dados SQLite, não necessitando a instalação do mesmo. O objetivo deste trabalho foi realizar um monitoramento de desempenho geral do Sistema Gerenciador de Banco de Dados (SGBD) SQLite, resultando em análises comparativas de resultados de desempenho.*

1. Introdução

Devido ao crescimento dos usuários de smartphones, os aplicativos estão sendo mais demandados, e passaram a ter um papel fundamental na vida dos usuários. Eles oferecem diversas ferramentas para diferentes tipos de necessidades de forma ágil e prática. Atualmente várias aplicações precisam salvar e recuperar informações, desde aplicações simples até as mais robustas, que necessitam apresentar as informações de forma rápida. Assim, evitar o constante consumo de dados dentro de uma aplicação é algo fundamental devido ao limite de dados de um aparelho mobile.

Trocas de informações entre os aplicativos e o servidor são constantemente usadas, e necessitam de uma conexão com a rede. Visando isso, é importante que exista uma base de dados dentro das aplicações, para que os aplicativos consigam trabalhar sem total dependência da internet. Com um banco de dados dentro da aplicação, é possível trabalhar de forma off-line, facilitando seu uso.

Um banco de dados é uma coleção de registros: “Banco de dados é uma coleção de dados ou informações relacionadas entre si, que representam aspectos do mundo real com significado próprio e que desejamos armazenar” [Guimarães 2003, p. 19], e ainda, pode ser relacional ou não relacional. O Sistema de Gerenciamento de Banco de Dados (SGBD) consiste em um conjunto de programas de computador (softwares) responsáveis pelo gerenciamento de um banco de dados. O objetivo principal é retirar da camada cliente a responsabilidade de gerenciar acessos, manipular acessos e organizar dados.

No contexto de aplicações móveis nem todos os benefícios de SGBD se aplicam devido à escassez de recursos computacionais como memória, capacidade de armazenamento, inerentes a dispositivos móveis (smartphones e tablets). Em contrapartida a plataforma Android trouxe suporte nativo ao SQLite que é uma poderosa biblioteca de banco de dados baseado em Structured Query Language (SQL) que atua como um “mini-SGBD” fazendo o controle de diversos bancos de dados que podem conter diversas tabelas.

O SQLite é uma biblioteca de software que fornece um sistema de gerenciamento de banco de dados relacional. O lite no SQLite significa peso leve em termos de configuração, administração de banco de dados e recursos necessários. O SQLite possui os seguintes recursos notáveis: autocontido, sem servidor, configuração zero, transacional [SQLite 2007].

Este trabalho tem por objetivo monitorar o desempenho do banco de dados SQLite ao realizar trocas de dados entre o aplicativo e o smartphone, gerando assim, um registro relatando todos os processos e execuções realizados, assim como o tempo decorrido para realizar cada operação.

2. Conceitos e Definições

O SQLite é uma biblioteca em processo que implementa o mecanismo de banco de dados SQL transacional autônomo, sem servidor e sem a necessidade de configuração. O código para o SQLite é de domínio público e, portanto, livre para uso para qualquer finalidade, comercial ou privada [SQLite 2007].

O SQLite é um mecanismo de banco de dados SQL incorporado. Ao contrário da maioria dos outros bancos de dados SQL, o SQLite não possui um processo de servidor separado. O SQLite lê e grava diretamente em arquivos de disco comuns. O banco de dados SQL completo com várias tabelas, índices, triggers e views estão contidos em um único arquivo de disco [SQLite 2007].

SQL é uma linguagem padrão universal para manejar bancos de dados relacionais através dos SGBDs. A "Linguagem Estruturada de Consultas" (SQL, traduzida para o português) é utilizada para interagir com o SGBD e realizar muitas tarefas, tais como: acrescentar e alterar registros, criar objetos no banco de dados, gerenciar usuário, consultar informações, controlar transações, etc. A totalidade das operações realizadas no banco de dados podem ser solicitadas ao SGBD utilizando esta linguagem [Furtado 2013].

Apesar da maioria dos SGBDs usarem a linguagem SQL, existem algumas diferenças entre eles, seja na sintaxe, nos recursos, etc. O SQLite entende a maior parte da linguagem SQL padrão. Mas omite alguns recursos e, ao mesmo tempo, adiciona alguns recursos próprios [SQLite 2007].

SQLite Data Type é um atributo que especifica o tipo de dados de qualquer objeto. Cada coluna, variável e expressão tem um tipo de dado relacionado no SQLite. No SQLite, o tipo de dados de um valor é associado ao próprio valor, não ao seu contêiner [Tutorialspoint 2013].

Cada valor armazenado em um banco de dados SQLite possui uma das seguintes classes de armazenamento:

NULL	O valor é um valor Nulo
INTEGER	O valor é um número inteiro, armazenado em 1, 2, 3, 4, 6 ou 8 bytes, dependendo da magnitude do valor.
REAL	O valor é um valor de ponto flutuante, armazenado como um número de ponto flutuante IEEE de 8 bytes.
TEXT	O valor é uma cadeia de texto armazenada usando a codificação do banco de dados (UTF-8, UTF-16BE ou UTF-16LE)
BLOB	O valor é um blob de dados, armazenado exatamente como foi inserido.

Quadro 1. Classes de Armazenamento.
Fonte: Adaptado de Tutorialspoint (2013).

O SQLite suporta o conceito de afinidade de tipos nas colunas. Qualquer coluna ainda pode armazenar qualquer tipo de dados, mas a classe de armazenamento preferencial de uma coluna é chamada de afinidade. Cada coluna da tabela em um banco de dados SQLite é designada a um dos seguintes tipos de afinidades:

TEXT	Esta coluna armazena todos os dados usando as classes de armazenamento NULL, TEXT ou CLOB. Tipos: Character(20), Varchar(255), Varying Character(255), Nchar(55), Native Character(70), Nvarchar(100), Text;

NUMERIC	Esta coluna pode conter valores usando todas as cinco classes de armazenamento. Tipos: Numeric, Decimal(10,5), Boolean, Date, Datetime;
INTEGER	Funciona da mesma forma que uma coluna com afinidade NUMERIC, com uma exceção em uma expressão CAST. Tipos: Int, Integer, Tinyint, Smallint, Mediumint, Bigint, Unsigned Big Int, Int2, Int8;
REAL	Comporta-se como uma coluna com afinidade NUMÉRICA, exceto que força valores inteiros em representação de ponto flutuante. Tipos: Real, Double, Double Precision, Float;
NONE	Uma coluna com afinidade NONE não prefere uma classe de armazenamento sobre outra e nenhuma tentativa é feita para coagir dados de uma classe de armazenamento para outra. Tipos: Blob

Quadro 2. Afinidade SQLite.
Fonte: Adaptado de Tutorialspoint (2013).

O SQLite é o banco de dados interno e oficial da plataforma Android, no qual é possível modelar uma estrutura de tabelas relacionadas entre si para representar os dados do mundo real. O mesmo é incorporado em todos os dispositivos Android, não sendo necessária configuração adicional. Somente é necessário definir as estruturas das tabelas e as operações que serão feitas utilizando os dados armazenados. Posteriormente, o banco de dados é gerenciado automaticamente pela plataforma Android [Cordeiro 2017].

Foi utilizado dois formatos de imagens como parâmetro de comparação de desempenho do SQLite trabalhando com imagem. O primeiro formato foi o Base64, que é um método para codificação de dados e transferência de dados no qual é constituído por 64 caracteres, para se transmitir dados binários por meios que lidam apenas com texto. O segundo formato foi o BitArray, que é um arranjo no qual armazena bits compactamente. Ele é um mapeamento para valores no conjunto {0, 1} contendo apenas duas possibilidades de valores que serão armazenados em um bit.

3. Materiais e Métodos

O foco desse trabalho foi analisar o desempenho de todas as execuções que o banco SQLite irá realizar trabalhando com o processamento de imagens, avaliando o tempo e seus tamanhos com diferentes quantidades de registros de todos os processos realizados pelo banco de dados como inserção, atualização e busca, conforme mostra o fluxograma abaixo:

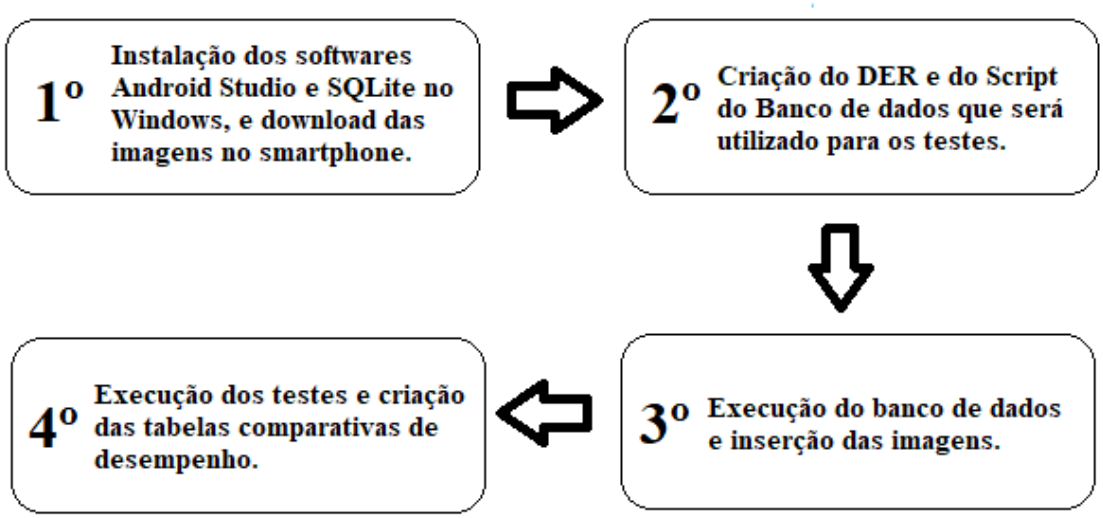

Figura 1. Descrição metodológica. Fonte: Elaborado pelos próprios autores.

Para iniciar o primeiro passo mostrado na figura 1, foi necessário instalar o software Android Studio, juntamente com o software SQLite a partir de um computador com sistema operacional Windows 10, da Microsoft.

O Android é um sistema operacional para dispositivos móveis desenvolvido pelo Google, e é o sistema operacional móvel mais utilizado no mundo graças ao seu baixo custo, e sua fácil configuração. Nesse experimento, será utilizado um smartphone com a versão Android 8.0.0.

O software Android Studio, é um ambiente de desenvolvimento integrado para desenvolver a plataforma Android, que providencia ferramentas e suportes totalmente especializados para esse sistema operacional, tornando essencial sua utilização nessa pesquisa. Não foi necessário a instalação da biblioteca SQLite no smartphone, pois a plataforma Android já possui a mesma instalada nas configurações de fábrica.

Foi baixada uma imagem no formato JPG com o tamanho de 70,4 KB e dimensões de 800 x 1120, ela foi armazenada no dispositivo para posteriormente ter seu formato modificado, e ser inserida na base de dados.

Conforme a segunda etapa da figura 1, foi desenvolvido um diagrama entidade relacionamento (DER) utilizando o software MySQL Workbench para facilitar o desenvolvimento do banco.

O MySQL Workbench é uma ferramenta visual unificada para arquitetos de banco de dados, desenvolvedores e administradores de banco de dados (*database administrator* -DBA), fornecendo modelagem de dados, desenvolvimento de SQL e ferramentas de administração abrangentes para configuração de servidor, administração de usuários, backup e muito mais [MySQL Workbench 2014].

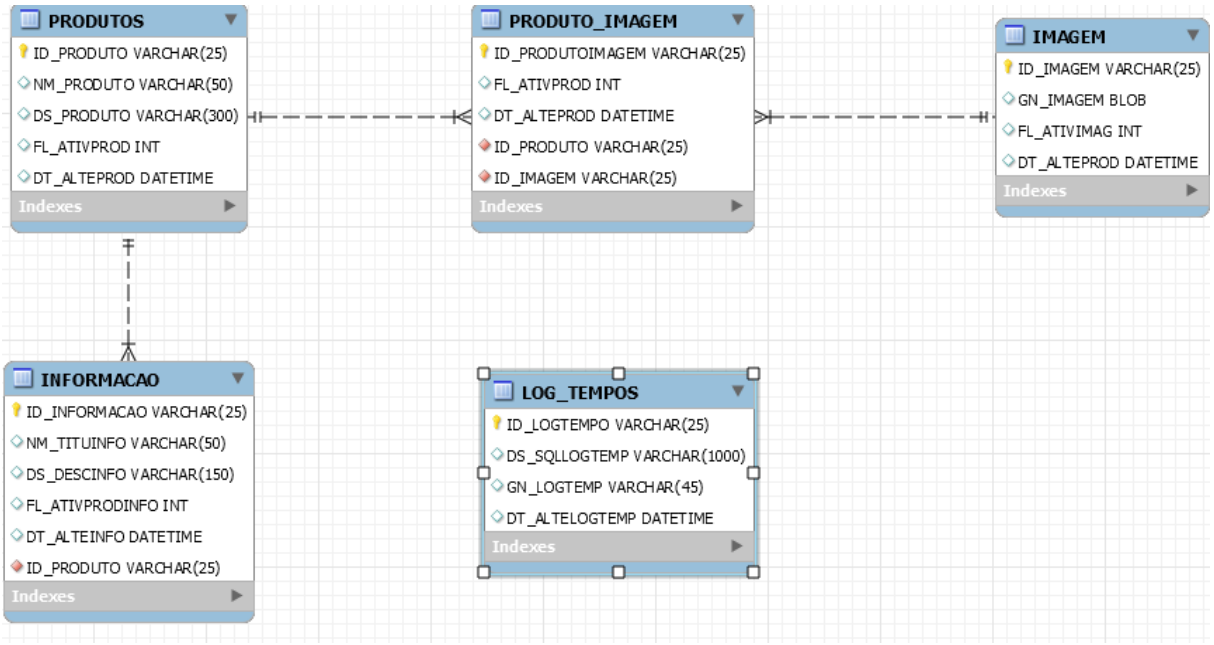

Figura 2. Diagrama Entidade Relacionamento (DER). Fonte: Elaborado pelos próprios autores.

Após a criação do modelo mostrado na figura 2, foi realizado a criação do script do banco de dados, nessa parte não há uma interface, portanto, o script foi criado a mão, na linguagem SQL respeitando as particularidades do banco SQLite.

Por padrão as chaves estrangeiras do banco SQLite vem desabilitadas, antes de se criar o banco foi necessário habilitá-las, para isso deve-se rodar o comando "PROGMA foreign_keys". Esse comando retorna ao número 1 caso a operação seja bem sucedida.

Os scripts do banco ficam em uma classe que estende da classe SQLiteOpenHelper. Essa classe contém dois métodos principais, o onCreate e o onUpgrade. O onCreate é responsável pela criação do script inicial do banco de dados no android, já o onUpgrade é onde os scripts de modificação ficarão depois que o banco já foi criado, qualquer alteração no banco depois de criado deve estar nesse método, pois toda vez que o banco é instanciado é feita uma verificação de versão, portanto a cada nova alteração os scripts posteriores a versão em que o banco estava serão executados.

A figura 3 a seguir, mostra todo o conteúdo do script criado para o banco de dados, com todos os atributos necessários para a realização dos testes com as imagens:

```
1.
2.      CREATE TABLE IF NOT EXISTS PRODUTO_IMAGEM (
3.         ID_PRODUTOIMAGEM VARCHAR(25) PRIMARY KEY NOT NULL,
4.         FL_ATIVPROD INT NOT NULL,
5.         DT_ALTEPROD DATETIME NOT NULL,
6.         ID_PRODUTO VARCHAR(25) NOT NULL,
7.         ID_IMAGEM VARCHAR(25) NOT NULL
8.      )
9.
10.     CREATE TABLE IF NOT EXISTS PRODUTOS(
11.        ID_PRODUTO VARCHAR(25) PRIMARY KEY NOT NULL,
12.        NM_PRODUTO VARCHAR(50) NOT NULL,
13.        DS_PRODUTO VARCHAR(300) NOT NULL,
14.        FL_ATIVPROD INT NOT NULL,
15.        DT_ALTEPROD DATETIME NOT NULL
16.     )
17.
18.     CREATE TABLE IF NOT EXISTS INFORMACAO(
19.        ID_INFORMACAO VARCHAR(25) PRIMARY KEY NOT NULL,
20.        NM_TITUINFO VARCHAR(50) NOT NULL,
21.        DS_DESCINFO VARCHAR(150) NOT NULL,
22.        FL_ATIVPRODINFO INT NOT NULL,
23.        DT_ALTEINFO DATETIME NOT NULL,
24.        ID_PRODUTO VARCHAR(25) NOT NULL
25.     )
26.
27.     CREATE TABLE IF NOT EXISTS IMAGEM(
28.        ID_IMAGEM VARCHAR(25) PRIMARY KEY NOT NULL,
29.        GN_IMAGEM BLOB NOT NULL,
30.        FL_ATIVIMAG INT NOT NULL,
31.        DT_ALTEPROD DATETIME NOT NULL
32.     )
33.
34.     CREATE TABLE IF NOT EXISTS LOG_TEMPOS(
35.        ID_LOGTEMPO VARCHAR(25) PRIMARY KEY NOT NULL,
36.        DN_SQLLOGTEMP VARCHAR(1000) NOT NULL,
37.        GN_LOGTEMP VARCHAR(45) NOT NULL,
38.        GT_ALTELOGTEMP DATETIME NOT NULL,
39.     )
40.    }
```

Figura 3. Scripts SQL. Fonte: Elaborado pelos próprios autores.

Conforme a figura 1, a terceira etapa do estudo é iniciada após a criação dos scripts, fazendo com que o arquivo do banco de dados seja gerado no primeiro acesso a ele, criando o arquivo com o formato .DB3 dentro de uma pasta chamada database que se localiza dentro da pasta da aplicação. Cada aplicação possui uma pasta de acesso restrito, pois somente a aplicação e o desenvolvedor têm acesso a ela. O Android Studio possui uma ferramenta que permite a visualização do arquivo de banco diretamente, uma vez que a aplicação já foi compilada no dispositivo.

Para os campos Ids foi usado um campo do tipo TEXT, pois quando se trabalha no SQLite com Android em vários dispositivos, não há como via banco garantir a

unicidade dos registros gerados em diferentes dispositivos, para isso a Google recomenda o uso de uma classe chamada UUID no Android que irá gerar um número único independentemente do aparelho, garantindo assim a unicidade do registro no banco de dados em diferentes aparelhos.

Para armazenar as imagens no banco de dados, foi necessário utilizar o campo BLOB (Binary Large Object) pois ele armazena uma cadeia de objetos binários, as informações armazenadas nessa cadeia ficam exatamente da forma como ela foi inserida, isentando a informação de qualquer tipo de formatação ou codificação.

Para monitorar o tempo de cada operação foi criada uma tabela de log que monitora o tempo de execução de todo o processo, o tempo de execução da query, a query que foi executada e a data da execução. Como o SQLite no Android não possui um recurso para monitoramento de tempo de execução de consultas, o tempo é monitorado via programação no Android, antes da execução do processo é obtido o tempo através de uma função chamada Current Time Millis, essa função possui precisão de nanossegundos, e após o término da operação um novo tempo é obtido e subtraído do primeiro, gerando assim o tempo de execução da operação. Esse processo também é feito antes de cada execução de cada query.

Com a imagem armazenada no smartphone, foi feita uma programação no Android para buscá-la e convertê-la no primeiro formato a ser monitorado, que foi a Base64, o sistema busca a imagem no armazenamento interno do dispositivo, converte para o formato acima e insere no banco. Todo esse processo é monitorado e gravado na tabela de Log.

Na quarta e última etapa mostrada na figura 1, foram executados testes com 100, 1.000 e 10.000 registros com o intuito de monitorar o tempo e o tamanho do banco após a inserção dos registros.

Após esse experimento, um novo banco foi criado, para isso apenas foi trocado o nome do banco no construtor da classe SQLite Open Helper, isso fez com que um novo banco fosse gerado sem perder o antigo, porém com as mesmas configurações do banco do experimento anterior. O segundo experimento segue os mesmos passos do anterior a única diferença é o formato da imagem gravada no banco, que agora passou a ser em BitArray, gerando o Log com os tempos e a query utilizada na inserção.

A inserção é feita por um comando no SQlite android que é o Insert With on Conflict, ele é equivalente a um insert normal porém com ele é possível determinar o que o banco fará em casos de duplicação de chave primária, é possível substituir o registro pelo novo ou ignorar o novo registro.

5. Resultados

Para demonstração prática do desempenho do SQLite trabalhando com imagens, foi realizado um experimento no qual foram inseridas 100 imagens em dois formatos diferentes, Base64 e BitArray. Após a inserção, foi realizada uma query de busca dessas mesmas imagens, utilizando o seguinte script:

```
"SELECT I.ID_IMAGEM, I.GN_IMAGEM, I.DT_ALTEPROD FROM IMAGEM I
INNER JOIN PRODUTO_IMAGEM PI ON I.ID_IMAGEM = PI.ID_IMAGEM
INNER JOIN PRODUTOS P ON PI.ID_PRODUTO = P.ID_PRODUTO
INNER JOIN INFORMACAO INFO ON INFO.ID_PRODUTO = P.ID_PRODUTO
AND I.FL_ATIVIMAG = 1 AND PI.FL_ATIVPROD = 1 AND P.FL_ATIVPROD =
1 AND INFO.FL_ATIVPRODINFO = 1"
```

Para a atualização, foi realizada a troca de imagens, alterando a imagem original por uma outra com mesmo tamanho (70,4 kb) e dimensões idênticas (800 x 1120). Foi utilizado o seguinte script:

```
"UPDATE IMAGEM SET GN_IMAGEM = ?, DT_ALTEPROD = ? WHERE
FL_ATIVIMAG = 1"
```

Quando executado o script acima, o banco compreende que o ponto de interrogação é um parâmetro que será fornecido pela linguagem, no caso do primeiro ponto é a imagem convertida, e o segundo é a data e hora atual do sistema.

Após a execução das queries, foram criados quadros comparando o tempo de execução de cada formato, assim como o tamanho ocupado pelo banco:

100 IMAGENS	INSERÇÃO	BUSCA	ATUALIZAÇÃO
Base64	2.333 milissegundos	1.787 milissegundos	1.158 milissegundos
BitArray	4.885 milissegundos	12.703 milissegundos	6.007 milissegundos

Quadro 3. Resultados com 100 imagens em milissegundos
Fonte: Elaborado pelos próprios autores.

O próximo experimento foi realizado com 1.000 imagens, porém o aparelho Android suporta retornar apenas 200 registros em BitArray, e 1.000 registros em Base64. Para a realização desse teste, foi utilizado um recurso no SQLite chamado "LIMIT", com ele se torna possível limitar a quantidade de registros que será retornada numa query. Nos testes com BitArray o LIMIT foi definido como 200 por consulta, dessa forma a consulta foi paginada, gerando 5 consultas para realizar a busca inteira das 1.000 imagens, sendo necessária a criação de outra query para juntar o tempo de cada consulta para se obter o tempo total da mesma:

```
"SELECT ID_LOGTEMPO, DS_SQLLOGTEMP, SUM(CAST(GN_LOGTEMP AS
INT)), DT_ALTELOGTEMP FROM LOG_TEMPOS WHERE ROWID > 1 AND ROWID
< 7"
```

1.000 IMAGENS	INSERÇÃO	BUSCA	ATUALIZAÇÃO
Base64	59.520 milissegundos	107.301 milissegundos	86.614 milissegundos
BitArray	165.970 milissegundos	478.608 milissegundos	62.813 milissegundos

Quadro 4. Resultados com 1.000 imagens em milissegundos
Fonte: Elaborado pelos próprios autores.

No segundo experimento, mostrado pelo quadro 4, Base64 continua sendo o formato com melhor desempenho comparado com o BitArray, com exceção da atualização, na qual o BitArray executou essa tarefa um pouco mais rápido do que o formato Base64.

Nesse próximo quadro, foram inseridas 10.000 imagens no banco, sendo necessário paginar também as queries de busca do formato Base64, já que o limite é de apenas 1.000 imagens. Com isso, tem-se 10 queries de busca com Base64, e 50 utilizando o formato BitArray.

10.000 IMAGENS	INSERÇÃO	BUSCA	ATUALIZAÇÃO
Base64	233.253 milissegundos	759.060 milissegundos	134.870 milissegundos
BitArray	510.572 milissegundos	2.262.213 milissegundos	635.737 milissegundos

Quadro 5. Resultados com 10.000 imagens em milissegundos
Fonte: Elaborado pelos próprios autores.

De acordo com o quadro 5, o desempenho do formato Base64, trabalhando diretamente com imagens no SQLite se mostrou superior comparado com o BitArray. O motivo disso, é devido ao seu peso de codificação, pois como o Base64 é composto por 64 caracteres, BitArray possui apenas 2, se tornando uma codificação mais pesada, portanto, mais demorada de ser modelada no banco de dados. O quadro 6 a seguir mostra o peso em megabyte do Banco de Dados após a inserção das imagens em ambos os formatos:

TAMANHO	100 IMAGENS	1.000 IMAGENS	10.000 IMAGENS
Base64	43,712 MB	436,64 MB	4.366,044 MB
BitArray	149,840 MB	1.497,936 MB	14.980,016 MB

Quadro 6. Tamanho do Banco de Dados em megabyte após a inserção das imagens
Fonte: Elaborado pelos próprios autores.

Foi realizado um quarto experimento, envolvendo 100.000 imagens utilizando o formato Base64. O tamanho total do banco foi de 21.829,760 megabytes, sendo necessário que o aparelho mobile possuísse pelo menos o dobro dessa quantidade para a realização de todos os testes. Com isso, não seria possível a reprodução desse mesmo teste utilizando o formato BitArray, pois seu tamanho total seria superior a 100.000 megabytes, quantidade a qual o smartphone não suportaria.

Enquanto esta pesquisa foi realizada, outras ideias surgiram e que poderão ser utilizadas em trabalhos futuros. Uma delas é a de refazer as pesquisas realizadas nesse trabalho, porém envolvendo outros tipos de arquivos diferentes de imagens, para verificar se a diferença entre BitArray e Base64 continua.

6. Conclusão

Com a execução dos experimentos, foi possível trabalhar diretamente com imagens no SQLite. O banco consegue suportar mais imagens do que os números utilizados nos testes, porém o Android não consegue carregá-las de uma única vez, além de ocupar um espaço imenso no aparelho, no qual o mesmo não suporta. Além disso, quando se trata de trabalhar com imagens, o formato Base64 trará um desempenho muito superior para o aparelho e para o banco quando comparado com o BitArray, pois irá custar menos tempo para a realização das pesquisas, e menos espaço no aparelho para armazenar as informações.

Por se tratar de um banco reduzido, trabalhar com imagens no SQLite se mostrou mais difícil e menos performático do que um banco com mais funcionalidades, porém devido ao aumento dos usuários da plataforma Android, é necessário que o SQLite trabalhe com diversos tipos de arquivos diferentes, como vídeos, áudios, imagens, entre outros, sendo valioso a otimização dos mesmos. A capacidade total do SQLite está diretamente ligada ao aparelho smartphone, e a versão do Android instalada no mesmo, logo, com a crescente evolução dos aparelhos mobiles, a manipulação de dados utilizando essa biblioteca serão mais performáticas, suportando ainda mais imagens.

7. Referências

Cordeiro, F. (2017) “Guardando Dados com SQLite”. https://www.androidpro.com.br/blog/armazenamento-de-dados/sqlite/, Novembro

Furtado, G. (2013). “Você precisa saber o que é SQL!”. http://dicasdeprogramacao.com.Br/o-que-e-sql/, Outubro

Gomes, E. H. (2013). “SGBD”. http://ehgomes.com.br/disciplinas/bdd/sgbd.php, Outubro

Guimarães, C. C. (2003). “Fundamentos de bancos de dados. Modelagem, projeto e linguagem SQL”. Campinas: Unicamp, Outubro

Moraes, A. (2015). “O que é um SGBD?”. https://www.mysqlbox.com.br/o-que-e-um-sgbd/, Outubro

MySQL Workbench (2014). “Products”. https://www.mysql.com/products/workbench/, Setembro

SQLite (2007). “About SQLite”. https://www.sqlite.org/about.html, Setembro

SQLite Tutorial (2015) “SQLite Create View”. http://www.sqlitetutorial.net/sqlite-create-view/, Novembro

Tutorialspoint (2013). “SQL – Data Type”. https://www.tutorialspoint.com/sqlite/sqlite_data_types.htm, Novembro