ISSN 2316-2872
T.I.S. São Carlos, v. 4, n. 1, p. 20-30, jan-abr 2015

Comparativo entre gerenciadores de banco de dados para aplicação Android

Drielli Peres Costa, Marilde Terezinha Prado Santos

Resumo: Este artigo apresenta um estudo prático e comparativo sobre a utilização de gerenciadores de banco de dados em aplicações Android. O artigo evidencia as principais diferenças na utilização de dois tipos de gerenciadores, permitindo o funcionamento de maneira offline. O estudo de caso desenvolvido utiliza os gerenciadores SQLite e DB4O para um mesmo aplicativo desenvolvido para gerenciar os treinamentos do usuário, ambos os gerenciadores atenderam as necessidades de forma transparente ao usuário.

Palavras-Chave: Aplicações Android, SQLite, DB4O.

Comparative of Databases Manangers for Android Applications

Abstract: *This paper presents a practical and comparative study about the use of Data Base Management in Android applications. This paper shows the major differences in using two types of Data Base managers, allowing offline operation. The case study developed uses SQLite and DB4O for the same application developed to manage the user training. Both Data Base managers clearly met the needs.*

Keywords:*Android applications, SQLite, DB4O.*

I. Introdução

O número de usuários de dispositivos móveis aumenta a cada dia, isto pode estar diretamente associado ao avanço da tecnologia e ao baixo custo desses dispositivos, o que torna este mercado muito competitivo e com grandes oportunidades de negócio.

Os aparelhos celulares deixaram de ser visto somente como um telefone portátil e assumiram um espaço na vida do usuário, com o objetivo de facilitar e auxiliar as necessidades do usuário. Em consequência, o desenvolvimento de aplicativos também evoluiu para atender esta demanda. Em agosto de 2014, já existiam 1,3 milhões de aplicativos Android (para 84,70% dos dispositivos no mercado) e 1,2 milhões de aplicativos iOS (para 11, 70% dos dispositivos no mercado) (TECMUNDO, 2014).

Na maioria dos sistemas ou aplicativos desenvolvidos existe a necessidade de se armazenar algum tipo de informação permanente que será utilizada em algum momento posterior. A utilização de um sistema de banco de dados como forma de armazenamento para informações de aplicativos móveis permite a opção de uso desses aplicativos em modo offline, ou seja, mesmo sem internet o uso da aplicação não é comprometido.

Diante deste cenário, o presente trabalho tem como objetivo principal comparar o potencial de duas soluções de banco de dados disponíveis para aplicativos móveis Android: SQLite e Database for objects (DB4O). A opção pela plataforma Android deve-se ao fato de ser uma plataforma open-source, baseada na linguagem Java e por ter alta inserção no mercado de dispositivos e aplicativos.

Este artigo está organizado da seguinte forma: na seção 2 é apresentada uma introdução ao sistema operacional Android; na seção 3 é apresentada uma visão geral sobre o armazenamento de dados no Android e ferramentas para gerenciamento de banco de dados disponíveis nesta plataforma. Na seção 4 é apresentado o aplicativo MeusTreinamentos em duas versões, uma utilizando o gerenciador de banco de dados relacional SQLite e a outra utilizando o gerenciador orientado a objetos DB4O. Na seção 5 são apresentados os trabalhos correlatos a este artigo e na seção 6 é apresentada a conclusão deste artigo com os trabalhos futuros que poderão ser realizados.

II. Android

Android é sistema operacional livre disponibilizado pela Google, baseado em Linux, voltado para dispositivos móveis, possui seu código aberto e funciona em diversos aparelhos como telefones, tablets e até mesmo em alguns televisores (MONTEIRO, 2012; LECHETA, 2013).

O Android representa uma grande oportunidade para construir novas aplicações inovadoras que funcionem em uma quantidade cada vez maior de dispositivos pelo mundo.

O Android fornece uma plataforma de desenvolvimento rica, permitindo fácil acesso aos recursos de hardware como, por exemplo, Wi-Fi (permite a conexão entre dispositivos sem fio) e GPS (Global Positioning System), seu desenvolvimento é baseado na linguagem Java o que facilita a criação de aplicativos que só aumentam a cada dia,

Departamento de Computação - Universidade Federal de São Carlos (UFSCar)
Caixa Postal 676 – 13.565-905 – São Carlos – SP – Brasil
Autor para correspondência: driellipéresc@ig.com.br, marilde@dc.ufscar.br

popularizando ainda mais a plataforma (MONTEIRO, 2012).

A) A plataforma Android

Antes do surgimento do Android, o mercado de desenvolvimento era fechado aos fabricantes e operadoras que controlavam quais aplicativos eram criados e incluídos em seus aparelhos. Com a evolução do mercado, os fabricantes liberaram um kit de desenvolvimento de software (SDK) para suas plataformas e criaram lojas de distribuição de aplicativos, o que permitiu que qualquer empresa ou desenvolvedor construísse aplicativos para os dispositivos móveis, aumentando desta forma as oportunidades de negócio (MONTEIRO, 2012).

A plataforma Android é composta de um sistema operacional baseado em Linux, middlewares (responsáveis pela comunicação do software com a aplicação) e um conjunto de aplicativos principais como gerenciador de contatos, navegador de internet, gerenciador de chamadas, entre outros. O desenvolvimento do Android se iniciou em 2003 com a empresa Android Incorporation, que foi comprada pela Google em 2005. Atualmente é líder no desenvolvimento da plataforma (MONTEIRO, 2012).

Como o sistema operacional é baseado em Linux, ele mesmo se encarrega de gerenciar a memória e os processos que são executados, isso permite que várias aplicações possam ser executadas ao mesmo tempo (LECHETA, 2013). Com os diferenciais e facilidades oferecidos por esta plataforma única de desenvolvimento é possível realizar a integração das novas aplicações desenvolvidas e aplicações nativas de uma forma simples, criando assim aplicações mais dinâmicas.

B) Google Play

Para facilitar a distribuição e acesso às aplicações Android, foi criado inicialmente o site Android Market. Atualmente a loja de aplicativos é chamada de Google Play, seu nome foi alterado para que os serviços Google como músicas, filmes e livros fossem unificados e assim o conteúdo digital fosse acessado em um único lugar (LECHETA, 2013).

Uma das vantagens de se desenvolver aplicativos Android é a liberdade com que se pode publicar e distribuir sua aplicação da maneira como desejar, a maneira mais comum é a distribuição de aplicativos na loja online Google Play, mas é possível também escolher outras maneiras de distribuição como, por exemplo, um site pessoal (MEIER, 2012). Essa facilidade permite que cada usuário personalize seu dispositivo de acordo com seu interesse. Para os aplicativos que são vendidos cerca de 70% dos lucros são repassados para o desenvolvedor (MONTEIRO, 2012; LECHETA, 2013).

Em março de 2012, já existiam mais de 450 mil aplicações na Google Play, com mais de dez bilhões de downloads de aplicativos em 130 países, possuindo uma taxa de crescimento de mais de um bilhão de downloads por mês (MEIER, 2012).

III. Armazenamento de dados no Android

O android oferece várias opções de armazenamento de dados, a decisão sobre qual das opções deve ser utilizada depende da necessidade do aplicativo em compartilhar informações com outros aplicativos e a quantidade de espaço que será necessário para os dados (ANDROID, 2014). Algumas das formas de armazenamento que podem ser utilizadas nos aplicativos são as seguintes:

Preferências Compartilhadas (Shared Preferences): permite salvar e recuperar dados através de um mapeamento de chave e valor de informações de tipos primitivos (booleans, strings, floats, ints e longs) (DEVMEDIA, 2014). Os dados são persistidos na sessão do usuário, ou seja, podem ser utilizados por todos os componentes de uma mesma aplicação, porém não pode ser compartilhado por outras aplicações (MEIER, 2012). Estes dados não serão perdidos quando a aplicação for finalizada. Um exemplo de uso é armazenar o usuário logado no aplicativo (MONTEIRO, 2012).

Armazenamento Interno (Internal Storage): permite salvar arquivos diretamente na memória interna do dispositivo, por padrão esses arquivos são privados, ou seja, não podem ser acessados por outras aplicações, que não seja aquela que o criou. Quando o aplicativo é desinstalado o arquivo gerado por ele também é removido (ANDROID, 2014). Pode-se optar por utilizar Cache ao invés de memória interna, neste caso, os arquivos são gravados em diretórios temporários. Caso o dispositivo esteja com pouca memória o Android poderá apagar os arquivos para disponibilizar espaço, no entanto é recomendado que o gerenciamento desses arquivos ocorra durante o ciclo de vida do aplicativo (MEIER, 2012; ANDROID, 2014). Um uso comum é o armazenamento de objetos complexos e também objetos serializáveis (DEVMEDIA, 2014).

Armazenamento externo (External Storage): qualquer dispositivo compatível com o Android possui suporte para o armazenamento externo, onde a memória externa utilizada pode ser uma mídia de memória removível, como um cartão de memória SD (Secure Digital Card) ou a memória interna (não removível), todos os arquivos salvos utilizando o modo de memória externa podem ser visualizados por qualquer outro dispositivo ou aplicação e podem ser modificados pelos usuários (ANDROID, 2014).

Banco de dados (Databases) - permite armazenar os dados estruturados em um banco de dados privado. O uso de banco de dados é muito útil para grandes ou pequenas aplicações, desde que haja necessidade de armazenar e recuperar qualquer tipo de informação.

Conexão de rede (Network Connection) - permite utilizar um serviço Web para armazenar ou recuperar dados que estejam disponíveis em outros serviços remotos, para isto é necessário possuir acesso à rede (ANDROID, 2014). Este tipo de serviço permite a centralização e integração de dados. Muitas empresas disponibilizam acesso aos seus serviços remotos via API (Application Programming Interface), de forma a permitir sua utilização para enriquecer as aplicações, exemplo de empresas que disponibilizam seus serviços são Facebook, Twitter, Google e Yahoo (MONTEIRO, 2012).

A) Integrações de banco de dados ao android

Muitos aplicativos necessitam do armazenamento de dados para seu funcionamento, sejam eles informados pelo próprio usuário, ou consultados de uma base ou organização externa. Para armazenamento de informações permanentes existem duas principais opções, o uso de web services através do qual é possível recuperar e armazenar os dados através de serviços na web, ou o uso de banco de dados embarcados na plataforma Android (MONTEIRO, 2012). A escolha do método que será utilizado para o armazenamento das informações vai depender da necessidade do aplicativo e de quanto espaço os dados irão precisar para serem armazenados. O armazenamento e recuperação de dados eficiente e rápida são essenciais para um dispositivo que possui capacidade de armazenamento relativamente limitada.

SQLite

A plataforma Android oferece suporte completo à biblioteca para criação e acesso a banco de dados SQLite, que é open-source e conhecido pelo uso em aplicações populares, como o Mozilla Firefox. O Android fornece a possibilidade de acesso a um banco de dados relacional para cada aplicação via o SQLite, o armazenamento é seguro e eficiente, uma vez que somente a aplicação que criou o banco de dados possui acesso às suas informações (MEIER, 2012; VOGEL, 2012).

A biblioteca SQLite, é um gerenciador de banco de dados independente de servidor e sem necessidade de configurações. Esta biblioteca utiliza o modelo relacional e tem suporte a transações. Seu uso é facilitado pelo amplo suporte da plataforma Android, além do suporte à linguagem SQL (SQLITE, 2014).

O SQLite armazena tudo que é preciso em um único arquivo, nele está contido toda a estrutura do banco de dados e os dados nele armazenados indiferentemente do número de tabelas e índices utilizados. É considerado um gerenciador de banco de dados leve, mas isto não está relacionado à sua capacidade de armazenamento de dados e sim a questões de complexidade de instalação e administração de recursos. É independente de servidor, ou seja, os arquivos são acessados diretamente de sua biblioteca, sem necessidades de configurações extras, necessita de pouco menos de 4MB de memória para seu funcionamento, realizando o mínimo de configuração (eliminando recursos avançados) pode ser alcançado um funcionamento com menos de 256KB de memória (KREIBICH, 2010).

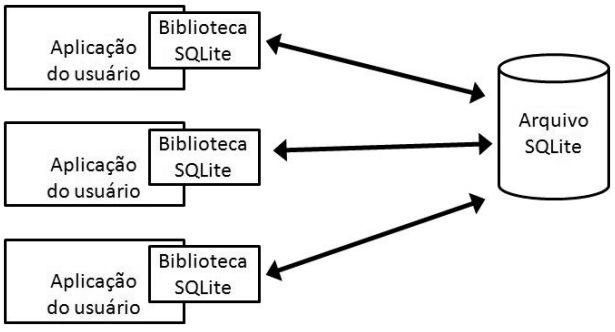


Figura 1. Acesso da Biblioteca SQLite ao arquivo de banco de dados (KREIBICH, 2010).

A Figura 1 representa a arquitetura da biblioteca SQLite, independente de servidor, ou seja, o acesso de leitura e escrita é feito diretamente da biblioteca ao arquivo de dados, sem a intervenção de um servidor de banco de dados.

Além do SQLite existem outros meios para o armazenamento de dados em banco de dados para o android que devem ser adicionados por plug-in, como por exemplo, o DB4O (Database For Objects) e NeoDatis ODB (NeoDatis Object Data Base), que são orientados a objetos.

DB4O

O DB4O é open-source, nele os objetos são armazenados nativamente, evitando assim a complexidade extra e perda de desempenho ao converter os dados para outro formato qualquer. É muito utilizado em aplicações embarcadas, sejam elas usadas em redes, aplicativos para dispositivos móveis, desktop ou servidores. O DB4O pode ser utilizado em sistemas escritos em JAVA ou .Net, explorando o fato dessas linguagens também serem orientadas a objetos (DB4O, 2014; MACORATTI, 2014). O DB4O não utiliza a linguagem SQL (Structured Query Language), mas possui suporte a uma tecnologia chamada de Native Query's, que permite que as consultas sejam realizadas utilizando uma semântica semelhante à linguagem de programação utilizada.

O DB4O pode ser utilizado sem muitos problemas no desenvolvimento de aplicativos Android oferecendo vantagens como desempenho, administração zero, transações seguras, implantação automática e processamento de memória compartilhada (DB4O, 2014).

NeoDatis ODB

O NeoDatis ODB é um banco de dados orientado a objetos open-source, é indicado para projetos de pequeno porte, atualmente possui suporte para o desenvolvimento de aplicações nas plataformas JAVA, .Net, Google Android, Groovy e Scala. É considerado nativo e transparente, pois persiste o objeto diretamente, como são utilizados na aplicação, sem a necessidade de alguma conversão (NEODATIS, 2014).

A Tabela 1 apresenta o comparativo de características das ferramentas SQLite, DB4O e NeoDatis, que podem ser utilizados nas soluções para aplicativos Android.

Tabela 1. Comparativo das ferramentas para persistência em banco de dados (SQLite, 2014; DB4O, 2014; NEODATIS, 2014)

Características	SQLite	DB4O	NeoDatis ODB
Fabricante	Domínio Público (patrocinado por SQLite Consortium)	Versant Corporation	NeoDatis Team
Tipo	Relacional	Orientado a Objetos	Orientado a Objetos
Plataformas suportadas	Java, .Net, iOS, Google Android	Java, .Net e Google Android	Java, .Net, Google Android, Groovy e Scala
Open Source	Sim	Sim	Sim
Suporte a SQL	Sim	Não	Não
Native Queries	Não	Sim	Sim
Tamanho da biblioteca	Aprox. 500 KB	Aprox. 400 KB	Aprox. 550 KB
Tamanho máximo (limitado a capacidade do dispositivo)	140 TB	254 GB	Informação não localizada

IV. Aplicação do estudo de caso

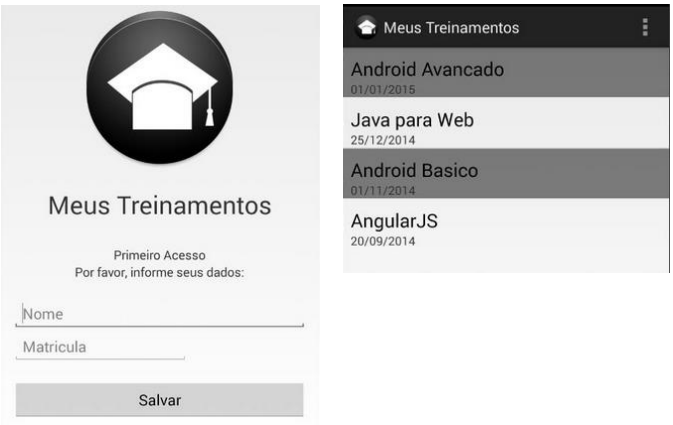
Para o desenvolvimento deste trabalho foi utilizada a biblioteca SQLite que utiliza o modelo do banco de dados relacional e o DB4O com o modelo de banco de dados orientado a objetos, com o objetivo de ressaltar os diferentes usos de armazenamento de dados para uma mesma solução. Para que seja aplicado o uso de ambas as ferramentas citadas anteriormente, foi criada uma aplicação na plataforma Android que complementa as funcionalidades do projeto Gestão de Qualificação.

O projeto Gestão de Qualificação foi desenvolvido como trabalho de conclusão do curso de Pós Graduação em Desenvolvimento de Software para Web da turma do ano de 2013. O objetivo do sistema é minimizar as deficiências de qualificação dos funcionários e permitir um maior gerenciamento dos treinamentos aplicados aos colaboradores.

O sistema pode ser dividido em três partes: o gerenciamento dos centros de custos e funções (associados às áreas da empresa e cargos), gerenciamento dos funcionários e gerenciamento dos treinamentos aplicados na empresa para cada centro de custo e função. O sistema foi desenvolvido para ser utilizado pela área de recursos humanos visando garantir um maior controle da qualificação de seus funcionários.

Foi criado o aplicativo Android MeusTreinamentos, que permite ao usuário organizar seus treinamentos realizados e controlar seu desempenho. Para este aplicativo foram criadas duas versões, uma utilizando a persistência de dados com o SQLite e a outra utilizando o DB4O, mas ambas com as mesmas funcionalidades, que consistem em um cadastro de usuários, apresentação de todos seus treinamentos cadastrados, consulta dos treinamentos que ainda não foram realizados, cadastro, alteração e exclusão de treinamentos.

A Figura 2 representa a tela inicial do sistema para o primeiro acesso do usuário (a) e a tela com listagem dos treinamentos existentes (b).



(a) (b)

Figura 2. Telas do aplicativo. (a) - Primeiro acesso do usuário, (b) – Lista de treinamentos cadastrados.

A) SQLite

Para o desenvolvimento do aplicativo MeusTreinamentos utilizando o armazenamento de dados no SQLite, foi criada uma aplicação Android dividida em camadas. Na Tabela 2 são apresentados os pacotes criados e as funcionalidades associadas.

Tabela 2. Pacotes do Aplicativo MeusTreinamentos – versão SQLite

Pacote	Funcionalidade
vSqlite.app	armazena as *Activities*, que representam a interface gráfica que permite a interação com o usuário, ou seja, elas fornecem o conjunto de funcionalidades ao usuário.
vSqlite.app.util	armazena as classes utilitárias do aplicativo, ou seja, contém classes que podem ser reutilizadas em outras partes do nosso aplicativo, evitando assim a duplicidade de código por exemplo.
vSqlite.dao	armazena as classes correspondentes a parte de acesso aos dados do aplicativo.
vSqlite.dao.mapping	armazena as classes correspondentes aos mapeamentos das entidades do aplicativo, para serem utilizadas nas consultas e manipulação dos dados.
vSqlite.entidade	armazena as classes correspondentes as entidades manipuladas pelo aplicativo.

A utilização da ferramenta SQLite está centralizada nas camadas vSqlite.dao e vSqlite.dao.mapping. A classe BaseDao (Data Access Object), é uma classe que herda da classe SQLiteOpenHelper que facilita a criação, manutenção e versionamento do banco de dados, ou seja, a classe BaseDao é responsável pela comunicação com o banco de dados. Esta classe contém duas propriedades utilizadas para o controle de criação e atualização do banco de dados, a DB_NAME que é o nome e DB_VERSAO que é a versão do banco de dados utilizado pela aplicação.

O construtor dessa classe deve ser desenvolvido informando o contexto que está sendo utilizado, nele é chamado o construtor da classe herdada informando o contexto, o nome do banco de dados e a versão atual.

O método onCreate é responsável por criar a estrutura necessária que é utilizada pela aplicação. Neste caso existem duas tabelas: usuário e treinamento, este método é executado quando o banco de dados for criado pela primeira vez. Para a criação das tabelas, é necessário executar o comando SQL utilizando o método execSQL da classe SQLiteDatabase, este método espera uma string com um comando SQL. Na exclusão das tabelas, também é utilizado o execSQL, passando o comando SQL a ser executado. A Figura 3 apresenta a criação das tabelas do aplicativo Meus Treinamentos.

Existem cinco tipos de classes de armazenamento disponíveis no SQLite, sendo o NULL que permite o armazenamento de um valor nulo, INTEGER que permite o armazenamento de um valor numérico inteiro de até oito bytes, REAL que permite o armazenamento de um valor numérico com ponto flutuante de até oito bytes, TEXT permite o armazenamento de textos utilizando a codificação de dados nos formatos UTF-8, UTF-16BE ou UTF-16LE e BLOB que permite o armazenamento de dados exatamente como eles são, este tipo de armazenamento converte o valor inserido em uma coleção de dados binário, permite o armazenamento de até dois gigabytes (SQLITE, 2014; KREIBICH, 2010).

```
@Override
public void onCreate(SQLiteDatabase db) {
    String createTableUsuario = "CREATE TABLE usuario (_id INTEGER PRIMARY KEY,"
            + "nome TEXT NOT NULL);";
    db.execSQL(createTableUsuario);

    String createTableTreinamento = "CREATE TABLE treinamento ("
            + "_id INTEGER PRIMARY KEY, descricao TEXT NOT NULL,"
            + "sigla TEXT NOT NULL, data_execucao TEXT NOT NULL,"
            + "validade INTEGER NOT NULL, nota REAL NULL,"
            + "is_aprovado INTEGER NULL, matricula_usuario INTEGER,"
            + "FOREIGN KEY(matricula_usuario) REFERENCES usuario(_id));";
    db.execSQL(createTableTreinamento);
}
```

Figura 3. Método onCreate da classe BaseDao

O SQLite possui um tipo de armazenamento dinâmico, o que permite que qualquer coluna com exceção da coluna PRIMARY KEY, possa receber qualquer tipo de valor das classes de armazenamento, ou seja, durante a execução do comando SQL, o valor inserido é convertido no tipo da classe de armazenamento utilizada. SQLite não possui suporte aos tipos Boolean, Date e Time (SQLITE, 2014).

O método onUpgrade é responsável por atualizar a versão do banco de dados da aplicação quando uma nova versão for disponibilizada, ou seja, é possível disponibilizar a uma primeira versão somente com a tabela usuário e depois disponibilizar uma atualização com as tabelas de usuário e treinamento, neste método é aplicada a lógica para esta atualização. A Figura 4 apresenta o método para atualização de versão, na aplicação MeusTreinamentos, em caso de atualização todas as tabelas são apagadas e então o método onCreate é chamado para a criação das tabelas necessárias.

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    String dropTableUsuario = "DROP TABLE IF EXISTS usuario;";
    db.execSQL(dropTableUsuario);

    String dropTableTreinamento = "DROP TABLE IF EXISTS treinamento;";
    db.execSQL(dropTableTreinamento);

    this.onCreate(db);
}
```

Figura 4. Método onUpgrade da classe BaseDao

O método onDowngrade segue o mesmo princípio do onUpgrade, mas é utilizado caso a versão seja regredida.

Todas as classes que necessitem de acesso ou leitura aos dados do banco de dados do aplicativo herdam de BaseDao, assim o acesso ao SQLiteDatabase é centralizado. Para as operações de leitura do banco de dados é necessário possuir uma instância apenas de leitura, recuperada através do método getReadableDatabase e para operações de escrita no banco de dados é utilizada uma instância de escrita, recuperada pelo método getWritableDatabase.

Para a realização de consultas é possível utilizar comandos SQL executando-os diretamente através do método rawQuery, conforme ilustrado na Figura 5, ou utilizando o método query disponível no SQLiteDatabase, informando os parâmetros necessários, conforme ilustrado na Figura 6.

```
public List<Treinamento> listar() throws SQLException {
    try {
        SQLiteDatabase db = super.getReadableDatabase();
        Cursor cursor = db.rawQuery(
                "SELECT _id, descricao, sigla, data_execucao, "
                        + "validade, nota, is_aprovado, matricula_usuario "
                        + "FROM treinamento order by data_execucao desc",
                null);
        List<Treinamento> lstTreinamento = TreinamentoMap
                .preencherListaTreinamentos(cursor);

        return lstTreinamento;
    } catch (Exception ex) {
        Log.e(TAG, ex.getMessage());
        throw ex;
    } finally {
        super.close();
    }
}
```

Figura 5. Consulta utilizando comando SQL

Independente da forma utilizada para a realização da consulta, o retorno dos métodos é um Cursor que retorna uma lista de registros encontrados, composto pelas colunas consultadas, este cursor deve ser percorrido linha a linha para que o objeto desejado seja preenchido com as informações desejadas. Para um melhor reuso do código fonte, existem as classes de mapeamento das entidades, que contém o mapeamento da tabela e suas colunas correspondentes.

```
public List<Treinamento> listar() throws SQLException {
    try {
        SQLiteDatabase db = super.getReadableDatabase();

        String ordenacaoData = String.format("%s %s",
                TreinamentoMap.COLUNA_DATA_EXECUCAO, "DESC");

        Cursor cursor = db.query(TreinamentoMap.TABELA,
                TreinamentoMap.COLUNAS, null, null, null, null,
                ordenacaoData);

        List<Treinamento> lstTreinamento = TreinamentoMap
                .preencherListaTreinamentos(cursor);

        return lstTreinamento;
    } catch (Exception ex) {
        Log.e(TAG, ex.getMessage());
        throw ex;
    } finally {
        super.close();
    }
}
```

Figura 6. Consulta utilizado no método query do SQLiteDatabase

As classes de mapeamento também contêm os métodos que efetuam a leitura de um cursor retornado pelas consultas e realiza a conversão de cada item deste cursor em um objeto ou em uma lista de objetos da entidade correspondente. A leitura do cursor para a conversão na entidade Treinamento é representada na Figura 7.

```
if (!cursor.isBeforeFirst() || cursor.moveToNext()) {
    treinamento = new Treinamento();

    treinamento.setCodigo(cursor.getInt(cursor
            .getColumnIndex(COLUNA_CODIGO)));
    treinamento.setDescricao(cursor.getString(cursor
            .getColumnIndex(COLUNA_DESCRICAO)));
```

Figura 7. Conversão de um cursor para a entidade Treinamento

Para a inserção de dados é possível utilizar a instrução SQL INSERT INTO, ou utilizar o ContentValues que permite mapear o conjunto de dados através de chave e valor, onde a chave é o nome da coluna e o valor é o dado a ser inserido. Para que a inserção seja realizada é utilizado o método insert acessado pela instância de escrita do banco de dados, informando o nome da tabela e os valores já mapeados no ContentValues. Para as operações que efetuam modificações na base de dados, é recomendada a utilização de transações, que permite que dados inconsistentes não sejam armazenados, pois em caso de falha, a modificação não será efetivada na base de dados, isto é controlado através dos métodos setTransactionSuccessful quando acionado ele efetiva as operações na base de dados e o endTransaction finaliza a transação, caso a transação seja finalizada antes do setTransactionSuccessful, as operações realizadas não serão efetivadas no banco de dados. A Figura 8 apresenta uma inserção de um treinamento.

Para a atualização de um registro a forma utilizada é bem similar ao de inserção utilizando o ContentValues, para este caso é utilizado o método update, através de uma instância de escrita do banco de dados, informando o nome da tabela, os valores, a condição e os argumentos da restrição utilizada para identificar quais registros serão atualizados, conforme apresentado na Figura 9.

```
public void salvar(Treinamento treinamento) throws SQLException {
    SQLiteDatabase db = super.getWritableDatabase();
    db.beginTransaction();
    try {
        ContentValues valores = new ContentValues();
        valores.put(TreinamentoMap.COLUNA_DESCRICAO,
                treinamento.getDescricao());
        valores.put(TreinamentoMap.COLUNA_SIGLA, treinamento.getSigla());
        valores.put(TreinamentoMap.COLUNA_DATA_EXECUCAO, treinamento
                .getDataExecucao().getTime());
        valores.put(TreinamentoMap.COLUNA_VALIDADE,
                treinamento.getValidade());
        db.insert(TreinamentoMap.TABELA, null, valores);
        db.setTransactionSuccessful();

    } catch (SQLException ex) {
        Log.e(TAG, ex.getMessage());
        throw ex;
    } finally {
        db.endTransaction();
        super.close();
    }
}
```

Figura 8. Inserção de um treinamento

```
String[] argumentosWhere = new String[] { treinamento.getCodigo()
        .toString() };
db.update(TreinamentoMap.TABELA, valores, CONDICAO_WHERE,
        argumentosWhere);
```

Figura 9. Atualização de um treinamento

Para a exclusão de registros está disponível através da classe SQLiteDatabase o método delete, para sua utilização é preciso informar o nome da tabela, uma condição e os argumentos da restrição. Caso a restrição não for informada, todos os dados da tabela serão excluídos, ou seja, a utilização deste método deve ser feita com muita atenção e cuidado. A Figura 10 representa a exclusão de um treinamento utilizando a condição de restrição com o identificador da entidade.

```
String[] argumentosWhere = new String[] { treinamento.getCodigo()
        .toString() };
db.delete(TreinamentoMap.TABELA, CONDICAO_WHERE, argumentosWhere);
```

Figura 10. Exclusão de um treinamento

Uma boa prática aplicada para a leitura e escrita no banco de dados é a utilização de blocos try/catch/finally, que permite o tratamento de erros inesperados, neste aplicativo nesses casos um log é registrado e a transação e a conexão com o banco de dados são encerradas corretamente.

B) DB4O

Para a implementação do aplicativo MeusTreinamentos utilizando o armazenamento de dados no DB4O, foi criada uma aplicação Android dividida em camadas semelhantes a estrutura da aplicação com SQLite da seção 4.1. Na Tabela 3 são apresentados os pacotes criados e as funcionalidades associadas.

Tabela 3. Pacotes do Aplicativo MeusTreinamentos – versão DB4O

Pacote	Funcionalidade
vdb4o.app	armazena as *Activities*, que representam a interface gráfica que permite a interação com o usuário, ou seja, elas fornecem o conjunto de funcionalidades ao usuário.
vdb4o.app.util	armazena as classes utilitárias do aplicativo, ou seja, contém classes que podem ser reutilizadas em outras partes do nosso aplicativo.
vdb4o.dao	armazena as classes correspondentes a parte de acesso aos dados do aplicativo.
vdb4o.entidade	armazena as classes correspondentes as entidades manipuladas pelo aplicativo.

A utilização do DB4O está centralizada na camada vdb4o.dao. Como utiliza um banco de dados orientado a objetos, não há nesta aplicação uma camada para mapeamentos dos dados do banco com os dados da entidade, pois o objeto é inserido na base de dados sem nenhuma conversão de dados. A utilização do DB4O na aplicação é realizada através de uma biblioteca que precisa ser adicionada ao projeto, para o desenvolvimento deste trabalho foi utilizado o arquivo db4o-8.0.249.16098-core-java5.jar, que foi adicionado na pasta libs do projeto.

A classe BaseDao (Data Access Object), é a responsável por criar e configurar o banco de dados. Esta classe também é responsável por centralizar e disponibilizar o acesso ao ObjectContainer que representa o banco de dados DB4O e permite que as operações de leitura e escrita sejam realizadas. Esta classe contém os métodos config que é responsável por criar uma nova configuração do banco de dados e getCaminho responsável por recuperar o caminho onde o arquivo é armazenado, conforme Figura 11.

```
private EmbeddedConfiguration config() {
    EmbeddedConfiguration configuration = Db4oEmbedded.newConfiguration();
    return configuration;
}

private String getCaminho(Context contexto) {
    return contexto.getDir("data", 0) + "/" + DATABASE_NAME;
}
```

Figura 11. Classe BaseDao – métodos config e getCaminho

A classe BaseDao também possui o método getDatabase que retorna o banco de dados da aplicação, nele é realizado o gerenciamento das instâncias do banco de dados baseado no contexto que está sendo utilizado, conforme a Figura 12 apresenta.

```
public ObjectContainer getDatabase() {
    try {
        if (database == null || database.ext().isClosed()) {
            database = Db4oEmbedded
                    .openFile(config(), getCaminho(contexto));
        }
    } catch (Exception ie) {
        Log.e(BaseDao.class.getName(), ie.toString());
    }
    return database;
}
```

Figura 12. Classe BaseDao – método getDataBase

Os métodos responsáveis por efetivar as operações na base de dados (commit), por reverter operações em andamento (rollBack) e por liberar o uso da base de dados (close) também devem ser implementados na classe BaseDao. As demais classes criadas para o acesso aos objetos devem herdar a BaseDao, para o reaproveitamento e centralização do uso do banco de dados da aplicação.

Para a realização das operações no banco de dados DB4O não há uma linguagem como o SQL, as operações são realizadas utilizando a própria sintaxe da linguagem de programação. Para recuperar objetos existem três formas que podem ser realizadas utilizando QBE (Query By Example), NQ (Native Queries) e SODA (Simple Object Data Access).

Para realizar a consulta de dados, não é necessário que haja nenhuma dependência ou conversão dos dados para a consulta, basta informar o tipo da classe pesquisada no método query do objeto de banco de dados, este tipo de consulta utiliza a forma QBE, que é representada na Figura 13.

```
private List<Treinamento> listarTodos() {
    try {

        List<Treinamento> lstTreinamento = getDatabase().query(
                Treinamento.class);
        return lstTreinamento;
    } catch (Exception ex) {
        Log.e(TAG, ex.toString());
        throw ex;
    }
}
```

Figura 13. Consulta de treinamentos – QBE

A Figura 14 representa uma consulta de treinamentos que é ordenada pela data de execução utilizando a forma de consulta SODA. Para esta consulta com ordenação são utilizados os métodos constrain para indicar qual a entidade que será utilizada, descend para indicar qual a propriedade da entidade será utilizada e o método do tipo de ordenação desejada, orderAscending para ordenação ascendente ou orderDescending para ordenação descendente, após realizar as configurações da query, basta que ela seja executada através da chamada do método execute.

```
Query consulta = getDatabase().query();
consulta.constrain(Treinamento.class);
consulta.descend("nrDataExecucao").orderDescending();

lstTreinamento = consulta.execute();
```

Figura 14. Consulta com ordenação de dados – SODA

No exemplo de ordenação por data, a propriedade nrDataExecucao do tipo Long foi criada para que a ordenação por um campo do tipo Date pudesse ser realizada, pois com a utilização deste tipo a ordenação não foi eficiente. A propriedade nrDataExecucao recebe a data de execução convertida para um tipo numérico, a ordenação este tipo de dado funciona corretamente. Outro ponto de atenção para o uso de ordenação, é que ao utilizar o método descend é preciso que existam registros para retorno na consulta, caso contrário uma exceção de referência nula é lançada, para tratar este problema deve ser adicionada a verificação de quantidade de registros existentes antes de ser realizada a ordenação.

Para a realização de consultas mais complexas é possível utilizar recursos da própria linguagem de programação para a construção das comparações utilizando as Natives Queries, o método de comparação pode ser utilizado com qualquer código que seja necessário, a única exigência é que seja retornado um valor Boolean. Na Figura 15 é apresentada uma consulta onde só devem ser listados os treinamentos com a data de execução superior a data atual.

```
List<Treinamento> lstTreinamento = getDatabase().query(
        new Predicate<Treinamento>() {

            private static final long serialVersionUID = 1L;

            public boolean match(Treinamento treinamento) {
                return treinamento.getDataExecucao().getTime() > new Date()
                        .getTime();
            }
        });
```

Figura 15. Consulta de treinamentos – NQ

Na inserção de registros é utilizado o método store, acessado via ObjectContainer da aplicação. Para realizar o armazenamento de dados a entidade que será armazenada deve ser informada como parâmetro do método store e após isso é preciso realizar o commit da operação, conforme apresentado na Figura 16.

A mesma lógica utilizada para a operação de inserção é aplicada para a atualização dos registros, porém para a atualização primeiramente é preciso recuperar o objeto atual da base de dados e atualizar os dados necessários, após esta atualização o mesmo processo da inserção é realizado, conforme apresentado na Figura 17.

```
public void salvar(Treinamento treinamento) {
    try {
        getDatabase().store(treinamento);
        commit();
    } catch (Exception ex) {
        Log.e(TAG, ex.getMessage());
        throw ex;
    }
}
```

Figura 16. Inserção de treinamento

```
Treinamento treinamentoAlterar = recuperar(treinamento.getCodigo());

treinamentoAlterar.setNota(treinamento.getNota());
treinamentoAlterar.setIsAprovado(treinamento.getIsAprovado());

getDatabase().store(treinamentoAlterar);
commit();
```

Figura 17. Atualização de treinamento

Na operação de exclusão de registros o método delete é utilizado informando o objeto da entidade em questão que será removido. A Figura 18 representa a exclusão de um treinamento, para que a exclusão ocorra corretamente é necessário recuperar o objeto atual da base de dados e então efetuar a remoção deste objeto.

Em todas as operações que efetuam modificações no banco de dados, ou seja, para todo acesso de escrita realizado no banco de dados é preciso realizar o commit desta operação para que ela seja realmente efetivada.

```
Treinamento treinamentoExcluir = recuperar(treinamento.getCodigo());
getDatabase().delete(treinamentoExcluir);
commit();
```

Figura 18. Exclusão de treinamento

C) Considerações

Após finalizar o desenvolvimento das aplicações utilizando formas de armazenamento diferentes foi possível concluir pontos positivos e negativos em ambas as formas de desenvolvimento.

Houve uma maior facilidade para o desenvolvimento da aplicação utilizando o SQLite, devido ao conhecimento já existente da linguagem SQL, devido a isto a implementação das operações foram mais produtivas, outro ponto positivo é que a biblioteca possibilita o uso de comandos SQL ou de recursos da própria biblioteca. A maior dificuldade encontrada no uso do SQLite foi com a manipulação de datas, conforme informações presentes na seção 4.1, os tipos de armazenamento são limitados, para o armazenamento de datas foi utilizado o tipo de armazenamento TEXT, sendo assim ao armazenar foi preciso converter a data com o método getTime (retorna a data convertida em milissegundos) para armazenar no banco de dados, ao recuperar as informações foi necessário uma nova conversão da data em milissegundos para a data real. Esta necessidade foi utilizada para que o filtro dos treinamentos ainda não executados fosse realizado, neste filtro somente são listados os treinamentos com data de execução superior a data atual para a realização desta consulta foram utilizadas as funções date e substr disponíveis no SQLite, a Figura 19 ilustra a estrutura da condição de data.

```
String CONDICAO_WHERE = "date(substr(TreinamentoMap.COLUNA_DATA_EXECUCAO "
        + ",1,10), 'unixepoch', 'localtime')" + "> date('now')";
```

Figura 19. Condição de restrição por data

No desenvolvimento da aplicação com o uso do DB4O foi encontrado um número maior de dificuldades, neste modelo de banco de dados orientado a objetos houve a necessidade de um estudo aprofundado no aprendizado deste conceito. O ponto positivo da utilização do DB4O é o uso dos próprios objetos, não sendo necessárias conversões ou adaptações. A maior dificuldade encontrada foi para realizar a ordenação dos treinamentos pela data de execução, pois utilizando a propriedade do tipo Date, a ordenação não foi realizada corretamente, sendo preciso utilizar uma forma alternativa com a data em formato numérico, ainda na ordenação são apresentados problemas quando a lista ordenada não possui registros, com isso é preciso ser realizada uma validação para que a ordenação seja acionada.

Como o DB4O trabalha diretamente com os objetos utilizados no mundo real, não há uma forma de gerar as chaves identificadoras automaticamente para este objeto quando ele é armazenado, o DB4O possui um controlador interno conhecido como OID, mas este não é exposto ao usuário do banco de dados. Para esta aplicação foi necessário

que o treinamento possuísse um identificador, então foi desenvolvida uma lógica que disponibiliza o código para que seja este utilizado durante a inserção do objeto.

Ao utilizar o DB4O é necessário um controle do acesso das Activities (contexto para a manipulação do banco de dados) ao arquivo do banco de dados, ou seja, quando uma Activity utilizar o arquivo, ao terminar é preciso realizar o encerramento deste arquivo através do método close, caso contrário outra Activity não poderá acessá-lo, pois ele está bloqueado para uso de outro contexto.

V. Trabalhos Relacionados

Os estudos para o desenvolvimento deste trabalho foram baseados em pesquisas de trabalhos com o tema correlacionado ao que foi proposto por este artigo, que é o desenvolvimento de aplicações Android com persistência de dados locais, sem a necessidade de uma conexão de rede. Também foram realizados estudos utilizando a documentação disponibilizada pelos websites oficiais das ferramentas SQLite e DB4O.

O trabalho de Silva e Lopes (2011) apresenta um comparativo entre algumas ferramentas de persistência de dados para dispositivos móveis: Berkeley DB XML, DB4O, SQL Anywhere e SQLite. Neste trabalho foi realizado o levantamento das principais características de cada ferramenta, porém não foram apresentadas as formas de utilização dentro de uma aplicação, diferente do artigo presente que tem o intuito de comparar as formas de utilização das ferramentas em aplicativos móveis.

No trabalho de Imperial et al. (2014), é apresentada a ferramenta Litebase que é um gerenciador de banco de dados para dispositivos móveis, o seu uso depende diretamente do TotalCross que é uma plataforma de desenvolvimento de aplicações para PDAs (Personal digital assistants) e smartphones. Neste trabalho também são apresentadas as características do Litebase, assim como suas limitações e comparações com o banco de dados SQLite.

O livro de Monteiro (2012) apresenta todo o processo de desenvolvimento de uma aplicação Android contemplando a criação do projeto, integração com o SQLite e publicação do aplicativo na Google Play. Este trabalho auxiliou em grande parte do desenvolvimento do aplicativo utilizado no estudo de caso da ferramenta SQLite apresentado no presente artigo.

Para o desenvolvimento do estudo de caso utilizando o DB4O foram utilizados os materiais da fabricante do produto e também os trabalhos disponibilizados por Devmedia (2014). O material que foi utilizado da fabricante do DB4O ficou indisponível no final do mês de setembro do ano de 2014. Segundo informações apresentadas ao tentar acessar conteúdo, esta indisponibilidade é devido a uma reestruturação nos produtos do fabricante Versant.

Este trabalho se diferencia dos demais relacionados, pois apresenta um comparativo exploratório sobre o uso prático do armazenamento de dados com o uso de duas ferramentas que possuem conceitos diferentes quanto ao modelo de dados adotado.

VI. Conclusão e Trabalhos Futuros

O presente trabalho apresentou uma abordagem comparativa do uso de dois diferentes gerenciadores de banco de dados para uma aplicação Android. Os estudos de caso desenvolvidos e apresentados demonstraram que ambas as ferramentas utilizadas atendem às necessidades da aplicação e podem ser utilizadas com eficiência, mas é importante salientar que no uso das ferramentas foram apresentadas dificuldades em partes do desenvolvimento que foram solucionadas, porém necessitaram de um tempo extra de estudo. Com o desenvolvimento realizado, observou-se que as necessidades da aplicação proposta foram atendidas independentes da forma que foi utilizada para o armazenamento de dados. Foram desenvolvidos dois aplicativos para o estudo de caso e ambos apresentam o mesmo comportamento visual e funcional ao usuário final, ficando o armazenamento de dados utilizado totalmente encapsulado e transparente ao usuário do aplicativo.

Nos estudos de caso, pode ser analisado o desenvolvimento das principais funcionalidades existentes como consulta, inserção, alteração e exclusão de registros, com exemplos reais da utilização dos gerenciadores de banco de dados SQLite e DB4O, é possível concluir que independente do modelo de banco de dados utilizado sendo ele relacional ou orientado a objetos o desenvolvimento de uma aplicação Android com armazenamento de dados local é possível. A decisão de qual método utilizar vai depender diretamente da necessidade de cada aplicativo, do tempo disponível para desenvolvimento e do conhecimento do desenvolvedor.

Para os trabalhos futuros, pretende-se realizar um estudo do desempenho de cada uma das ferramentas em dispositivos com configurações diversas, para que seja possível apontar o uso do gerenciador mais indicado com relação ao dispositivo utilizado. Seria interessante também a realização de um estudo do uso do gerenciador NeoDatis citado neste trabalho, com o objetivo de ampliar as opções de uso nos aplicativos. As aplicações já existentes também podem ser melhoradas com o desenvolvimento de uma integração com webservices que podem sincronizar o dado periodicamente com o sistema web existente.

Referências

ANDROID. Disponível em: < http://developer.android.com>. Acesso em: 03 set. 2014.

DEVMEDIA. (a) Disponível em: < http://www.devmedia.com.br/persistencia-de-dados-no-android/26947>. Acesso em: 02 set. 2014.

DEVMEDIA. (b) Disponível em: <http://www.devmedia.com.br/db4objects-na-terra-de-gigantes-do-bd-relacional-com-java-parte-i/4121>. Acesso em: 20 out. 2014.

DIMARZIO, J. Android: A programmer's guider. Nova Iorque: McGraw-Hill, 2008.

KREIBICH, J. A. Using SQLite. Sebastopol: O'Reilly Media, 2010.

LECHETA, R. R. Google Android: Aprenda a criar aplicações para dispositivos móveis com o Android SDK. 3 ed. São

Paulo: Novatec Editora, 2013.

MEIER, R. Professional Android 4 Application Development. Indianapolis: John Wiley & Sons, 2012.

MONTEIRO, J. B. Google Android - Crie aplicações para celulares e tablet. São Paulo: Casa do Código, 2012.

NEODATIS. Disponível em: <http://neodatis.wikidot.com>. Acesso em: 10 set. 2014.

SIMON, J. Head First Android Development. Sebastopol: O'Reilly Media, 2011.

TECMUNDO. Disponível em: <http://www.tecmundo.com.br/android/32016-com-700-mil-aplicativos-google-play-alcanca-app-store-na-quantidade-de-programas-ofertados.htm>. Acesso em: 22 out. 2014.

VOGEL, L. Android SQLite Database and ContentProvider – Tutorial. s.l: s. ed, 2014. Disponível em <http://www.vogella.com/tutorials/AndroidSQLite/article.html>. Acesso em: 28 ago. 2014.

IMPERIAL, J. C. et.al. Litebase: A Database System for Mobile Devices. In: SBBD - Simpósio Brasileiro de Banco de Dados, 29, 2014, Curitiba. Anais. Curitiba: SBBD 2014, p. 215-220.

SILVA, C. P. R; LOPES, F. S. Análise de opções de persistência para aplicações móveis Corporativas. Revista Eletrônica de Tecnologia e Cultura, Jundiaí, v. 3, n. 3, p. 1-8, set./nov. 2011.

DB4O. Disponível em: < http://www.db4o.com>. Acesso em: 03 set. 2014.

MACORATTI. Disponível em: <http://www.macoratti.net/09/08/net_db4o.htm>. Acesso em: 05 set. 2014.

SQLITE. Disponível em: <http://www.sqlite.org>. Acesso em: 07 out. 2014.