Dispositivos Móveis

Aula 11 - SQLite

Apresentação

Nesta aula, complementaremos o estudo de armazenamento e manipulação de dados através do uso do SQLite no Android.

Vídeo 01 - Apresentação

Objetivos

Ao final desta aula, você deverá ser capaz de:

- Entender o uso de banco de dados no Android.
- Acessar e manipular dados na aplicação.

Introdução ao SQLite

Em essência, o SQLite é um pacote de software de domínio público que provê um sistema de gerenciamento de bancos de dados relacionais que são utilizados para armazenar registros, definidos pelo usuário em grandes tabelas.

Mais do que apenas armazenar os dados, um sistema de gerenciamento de bancos deve definir uma estrutura que permita a execução de consultas complexas sobre os dados armazenados, combinando diversas tabelas, por exemplo.

O uso do nome "lite" não faz referência às capacidades disponibilizada pelo pacote, mas sim à facilidade existente no processo de preparação, administração e uso de recursos. Para melhor definir a tecnologia, vamos conferir algumas de suas principais características:

- **Arquitetura sem Servidor** - Não é necessário ter um servidor ou processo rodando separadamente para o funcionamento do SQLite, visto que seu processamento é feito diretamente em arquivos.
- **Sem necessidade de Configuração** - Por não possuir um servidor, não é necessário fazer nenhum tipo de preparação para utilizar um banco SQLite.
- **Multiplataforma** - Seu uso é independente da plataforma, retirando necessidade de gerenciamento do ambiente de execução.
- **Tamanho Reduzido** - O tamanho total da biblioteca não chega a 1 megabyte e seu consumo de memória também é bem baixo. Dependendo da necessidade de uso, é possível fazer alguns ajustes a fim de melhorar seu uso de memória e de espaço.

Para melhor entender o funcionamento do SQLite, vamos comparar as **Figuras 1 e 2**.

Figura 01 - Funcionamento de Sistemas de Banco de Dados tradicionais

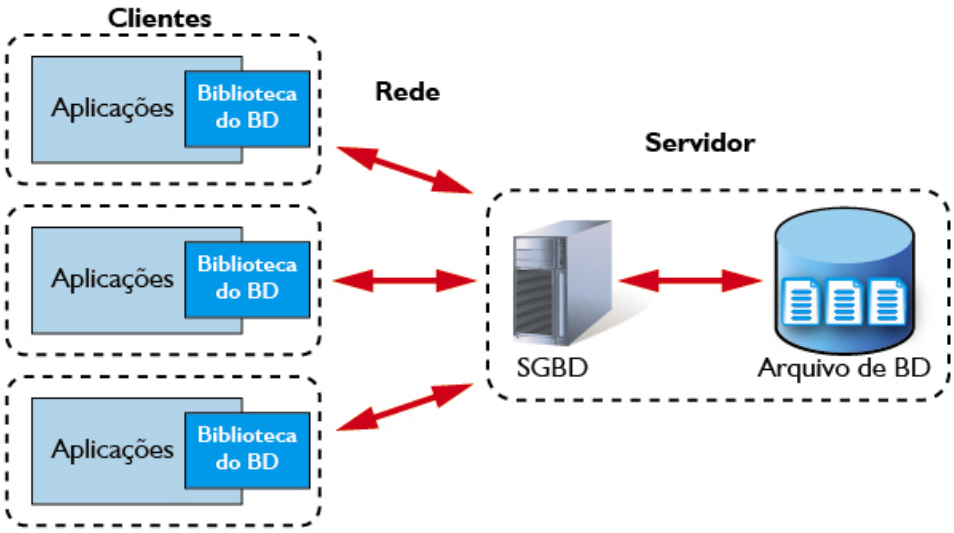
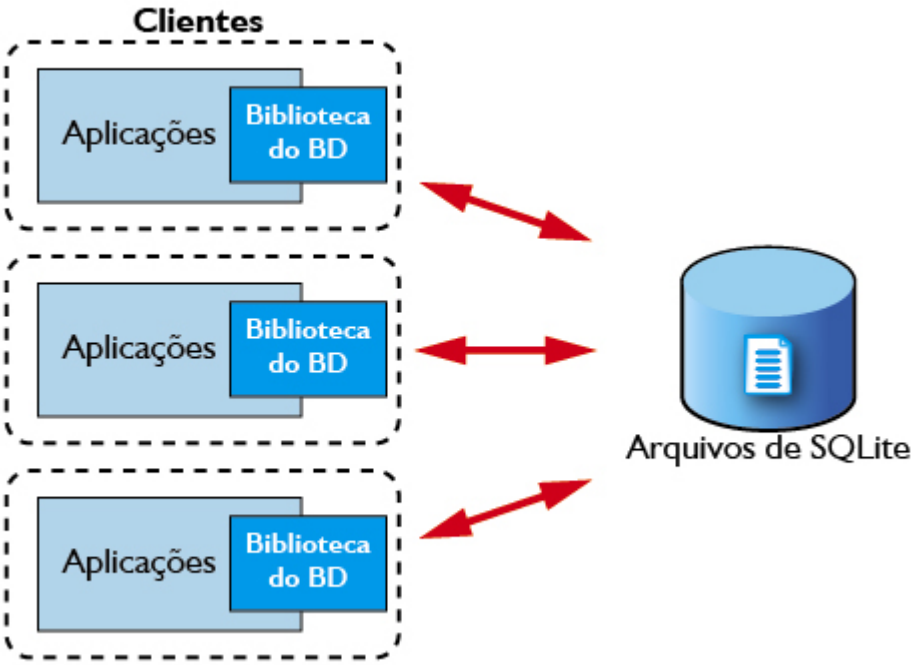


Figura 02 - Funcionamento do SQLite e sua arquitetura sem servidor



Como podemos observar nas **Figuras 1 e 2**, enquanto nos sistemas tradicionais existe uma camada de acesso aos arquivos do banco (servidor), no SQLite esse acesso é feito de forma direta, eliminando a necessidade de uma camada de comunicação entre as aplicações do usuário e os arquivos de banco.

Vídeo 02 - Usuando o SQLite Manager

Uso do SQLite no Android

Agora que entendemos a teoria por trás do SQLite, veremos como implementar um banco de dados utilizando essa tecnologia e as classes de suporte disponibilizadas pelo Android. Antes de começar qualquer implementação, é importante saber que existem outras formas de fazer uso do banco de dados, mas mostraremos aqui a forma mais simples e recomendada pelo Android. Seguindo, então, a estrutura recomendada, devemos criar uma classe filha de SQLiteOpenHelper, que cuidará da criação e gerência de versão de dados. Após criar a subclasse, implementamos os métodos onCreate, onUpgrade e, opcionalmente, onOpen. Todo o tratamento de criação, modificação e abertura de conexão serão feitas pelo Android, com o comportamento sendo definido através dos métodos de callback, assim como ocorre no fluxo de vida de uma Activity.

Para melhor exemplificar o uso do banco de dados, iremos trabalhar com um exemplo bem simples, mas bem estruturado. Faremos toda a estrutura necessária para a criação do banco, inserção e gerenciamento dos dados de uma classe de domínio que chamaremos de Comment, uma classe para representar um comentário qualquer de um usuário em um website. A implementação dessa classe é bem simples e segue o modelo mostrado na **Listagem 1.**

```
1  package PACOTE.DE.SUA.APLICACAO;
2
3  public class Comment {
4      private long id;
5      private String comment;
6      public Comment(long id, String comment) {
7          this.id = id;
8          this.comment = comment;
9      }
10      public long getId() {
11          return id;
12     }
13     public void setId(long id) {
14          this.id = id;
15     }
16     public String getComment() {
17          return comment;
18     }
19     public void setComment(String comment) {
20          this.comment = comment;
21     }
22     @Override
23     public String toString() {
24          return comment;
25     }
26 }
```

Listagem 1 - Implementação da classe representativa do modelo

É importante ressaltar alguns pontos na implementação da classe Comments. Começando pelos atributos definidos, temos somente dois, o id e o comment. O atributo id será utilizado como chave primária da nossa tabela de comentários no banco de dados, enquanto que o comment será a representação textual daquele comentário. Logo após essas definições, temos os respectivos *getters* e *setters*, seguidos de um *override* do método toString. Esse *override* serve para facilitar nosso trabalho quando formos representar nosso objeto comentário em formato de String.

Tendo definida nossa classe de modelo e, portanto, nossa tabela no banco, podemos agora implementar nossa classe filha de SQLiteOpenHelper, para criar e gerenciar o banco, além de abrir nossas conexões com o mesmo. Para essa classe, daremos o nome de MySQLiteHelper, e sua implementação é mostrada logo a seguir na **Listagem 2**.

```
package PACOTE.DE.SUA.APLICACAO;

public class MySQLiteHelper extends SQLiteOpenHelper {

    public static final String TABLE_COMMENTS = "comments";
    public static final String COLUMN_ID = "_id";
    public static final String COLUMN_COMMENT = "comment";

    private static final String DATABASE_NAME = "commments.db";
    private static final int DATABASE_VERSION = 1;

    private static final String DATABASE_CREATE = "create table "
        + TABLE_COMMENTS + "( " + COLUMN_ID
        + " integer primary key autoincrement, " + COLUMN_COMMENT
        + " text not null);";

    public MySQLiteHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase database) {
        database.execSQL(DATABASE_CREATE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.w(MySQLiteHelper.class.getName(),
            "Upgrading database from version " + oldVersion + " to "
            + newVersion + ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_COMMENTS);
        onCreate(db);
    }

}
```

Listagem 2 - Implementação do SQLiteOpenHelper da aplicação

Em relação ao código da **Listagem 2**, iniciamos sua implementação criando algumas constantes que serão usadas na definição dos *scripts* SQL para criação e gerenciamento da nossa tabela de comentários, são elas: *TABLE_COMMENTS*, *COLUMN_ID* e *COLUMN_COMMENT* que representam, respectivamente, o nome da nossa tabela, a coluna referente a seu id e a coluna do comentário propriamente dito. Logo após, temos duas constantes de definição e gerenciamento do banco de dados como um todo, *DATABASE_NAME,* que armazena o nome que usaremos para a nossa base de dados SQLite e a *DATABASE_VERSION*, que será utilizada no controle de versão do nosso banco. Por fim, definimos a *String* que representa o *script* SQL de criação do banco de dados, a *DATABASE_CREATE*.

Para melhorar a compreensão, mostraremos na **Listagem 3** o *script* resultante na constante.

```
create table comments (
        _id integer primary key autoincrement,
        comment text not null
);
```

Listagem 3 - *Script* resultante para criação da estrutura do banco de dados

Como podemos observar na **Listagem 3**, o *script* de criação define uma tabela com a estrutura similar à classe *Comments* mostrada na **Listagem 1**. Importante notar que, no caso do id, por se tratar da chave primária, foi indicado como tal, além de ser definido como *autoincrement*, ou seja, essa informação será atribuída automaticamente ao seu registro na tabela no momento de sua criação.

Dessa forma, se precisarmos inserir um objeto do tipo *Comments* na base de dados, basta criar o objeto, definir o texto do comentário e deixar que o próprio banco de dados gere o valor para o atributo id. Logo após as definições dessas constantes, temos definidos três métodos essenciais para o correto funcionamento da nossa base de dados. São eles: o construtor da classe e os métodos onCreate e onUpgrade.

No construtor, apenas invocamos um construtor de nossa classe pai, passando algumas informações como o contexto, nome do banco e sua versão (é importante salientar que de acordo com essa versão, o método onUpgrade será invocado, para que o banco possa ser recriado de acordo com a nova estrutura). Ele é responsável por preparar a estrutura necessária para criação da nossa base de dados e abertura de conexões.

Logo após, temos a definição do método onCreate. Aqui será feita a criação do nosso banco de dados propriamente dito, e devemos, também, popular o banco com as informações necessárias para seu correto funcionamento. No nosso exemplo, estamos apenas criando a tabela, executando o *script* mostrado na **Listagem 3**.

Por fim, definimos o método onUpgrade. Por definição, aqui devemos fazer todo o procedimento necessário para as possíveis mudanças de versão, removendo tabelas, adicionando tabelas, colunas etc.

Vídeo 03 - Acessando o Banco de Dados no Emulador

Na nossa implementação, primeiramente cadastramos um *log* de *warning* indicando as versões antiga e nova da base de dados que está sendo atualizada, e em seguidas executamos o *script* mostrado na **Listagem 4.**

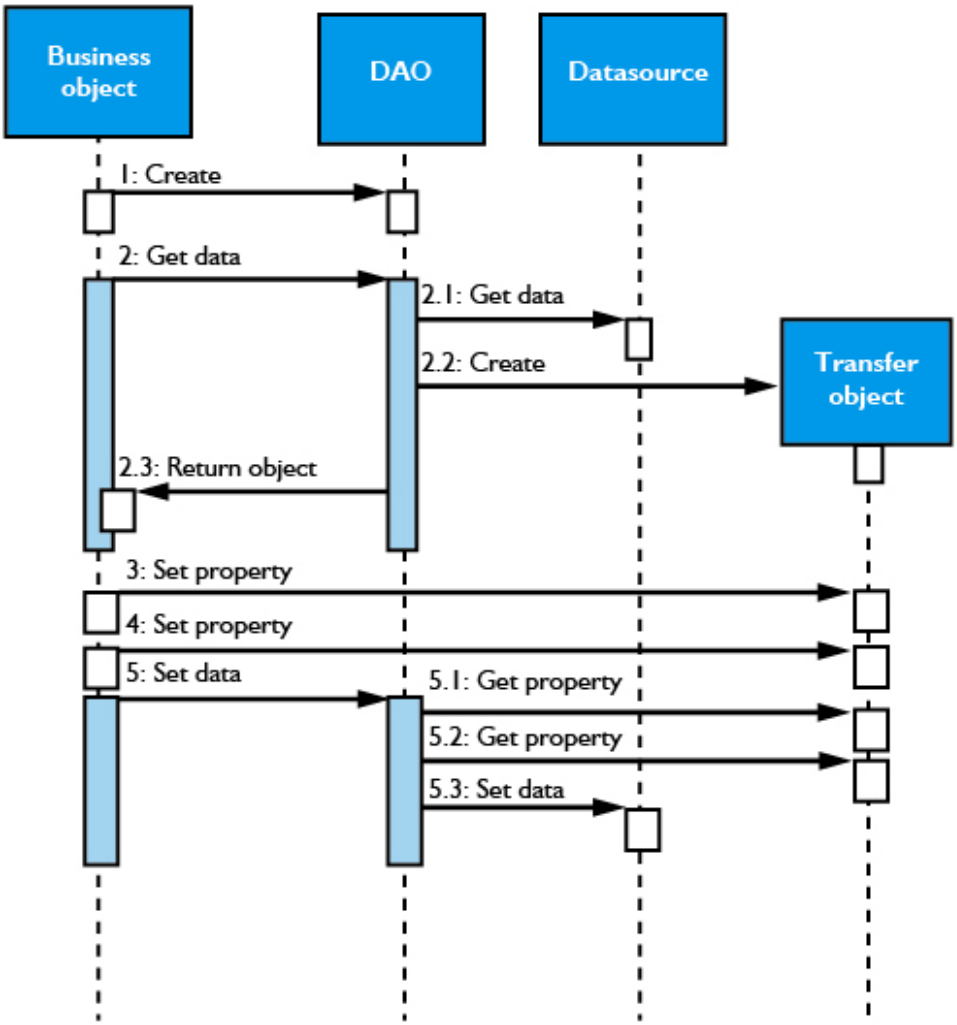
```
1 DROP TABLE IF EXISTS comments
```

Listagem 4 - *Script* de remoção da tabela do banco de dados

O *script* apresentado na **Listagem 4** irá remover, caso exista, a tabela cujo nome seja "comments", que nesse caso representa toda a estrutura da nossa base de dados. Logo após limpar nossa base de dados, garantimos que o processo de criação será executado completamente, a partir da chamada direta ao método onCreate.

Com o código até agora mostrado, temos a parte estrutural do nosso banco de dados, mas ainda necessitamos de um canal de comunicação para que possamos trabalhar com os dados armazenados. Para tal comportamento, utilizaremos o padrão de projeto Data Access Object (DAO), explicado no diagrama de sequência na **Figura 3**.

Figura 03 - Funcionamento do padrão DAO



Como podemos perceber no diagrama, o padrão DAO se responsabiliza pela comunicação dos nossos objetos de negócio com a nossa fonte de dados, separando nosso processamento da estrutura do banco. Seguindo um pouco a ideia proposta por esse padrão, criaremos uma classe responsável pelo tratamento das operações básicas com nossa classe Comment. A Listagem 5 apresenta a implementação da classe CommentsDao.

```
package PACOTE.DE.SUA.APLICACAO;

public class CommentsDao {

    private MySQLiteHelper dbHelper;

    private SQLiteDatabase database;

    private String[] allColumns = { MySQLiteHelper.COLUMN_ID, MySQLiteHelper.COLUMN_COMME

    public CommentsDao(Context context) {
        dbHelper = new MySQLiteHelper(context);
    }

    public void open() throws SQLException {
        database = dbHelper.getWritableDatabase();
    }

    public void close() {
        database.close();
        dbHelper.close();
    }

    public Comment createComment(String comment) {
        ContentValues values = new ContentValues();
        values.put(MySQLiteHelper.COLUMN_COMMENT, comment);
        long insertId = database.insert(MySQLiteHelper.TABLE_COMMENTS, null, values);
        Comment newComment = new Comment(insertId,comment);
        return newComment;
    }

    public void deleteComment(Comment comment) {
        long id = comment.getId();
        database.delete(MySQLiteHelper.TABLE_COMMENTS, MySQLiteHelper.COLUMN_ID + " = " + i
    }

    public List<Comment> getAllComments() {
        List<Comment> comments = new ArrayList<Comment>();
        Cursor cursor = database.query(MySQLiteHelper.TABLE_COMMENTS, allColumns, null, null, r
        cursor.moveToFirst();

        while (!cursor.isAfterLast()) {
            Comment comment = cursorToComment(cursor);
            comments.add(comment);
            cursor.moveToNext();
        }

        cursor.close();

        return comments;
    }
```

```
52
53      private Comment cursorToComment(Cursor cursor) {
54          Comment comment = new Comment(cursor.getLong(0), cursor.getString(1));;
55
56          return comment;
57      }
58  }
```

Listagem 5 - Implementação da classe CommentsDao

Iniciamos a implementação definindo duas variáveis que irão nortear nossa conexão e trabalho com o banco de dados: dbHelper e database. O objeto dbHelper será uma instância da nossa classe MySQLiteHelper, mostrada na **Listagem 2**, e será responsável por preparar o banco, assim como abrir e fechar nossa conexão com ele. Trabalhando em conjunto com o dbHelper, temos o objeto database, que como veremos mais a frente, será a instância do banco de dados, e é através dela que iremos fazer todas as operações que dependerem do banco.

Logo após essas duas variáveis, definimos um vetor de *Strings*, chamado allColumns, que armazena todas as colunas presentes na nossa tabela de comentários e será utilizado mais a frente para realizar consultas e inserir dados na nossa tabela.

Em seguida, definimos um construtor para nosso DAO, que recebe o contexto da aplicação e o utiliza para instanciar o objeto dbHelper. Após criado o nosso DAO, toda a estrutura de conexão já está pronta, mas o banco ainda não será criado e nem será possível realizar nenhum tipo de consulta. Para que isso seja possível, teremos que invocar o método open, que na sua definição abre a conexão com a nossa base de dados, armazenando a instância retornada pela chamada de dbHelper.getWritableDatabase() no objeto database. A partir da chamada desse método será realizada a criação do banco e será possível a realização de consultas nele. Depois dessa definição, temos o método close, que fecha as conexões existentes com o banco e com o dbHelper.

Dando continuidade, temos mais três métodos públicos que fazem parte das funcionalidades básicas disponibilizadas pelo DAO a outras classes que desejam manipular objetos do tipo Comments. Esses métodos são:

1. createComment – armazena um novo comentário no banco de dados.
2. deleteComment – remove um comentário do banco de dados.

3. getAllComments – recupera todos os comentários armazenados no banco de dados.

O método createComment recebe uma String representando o texto do comentário a ser criado. Sua implementação é iniciada com a criação de um objeto do tipo ContentValues. Esse objeto é um conjunto de pares chave-valor, que representam o nome e conteúdo de cada campo de uma tabela. No exemplo, um único par é definido, referente à coluna de comentário. A inserção desse registro no banco é feita através do método database.insert, que retorna o valor do id do novo registro na tabela. Após o seu armazenamento, criamos e retornamos o objeto que representa o comentário recém-criado, finalizando o processo de inserção da informação na base de dados.

A implementação do método de remoção do registro é mais simples, e consiste apenas em chamar o método database.delete. Esse método recebe como parâmetro o nome da tabela e a condição de remoção, que nesse caso é a tabela comment e a condição é que o registro tenha o id igual ao passado como parâmetro para o nosso método.

Por fim, o método getAllComments é responsável por listar todos os comentários armazenados em banco. Para tal, é feita uma consulta semelhante através de uma chamada ao método database.query. Esse método possui vários parâmetros que representam as partes de uma *query* SQL, são eles:

1. O nome da tabela;
2. A lista das colunas a serem recuperadas;
3. A cláusula WHERE (opcionalmente incluindo parâmetros);
4. A lista dos valores para substituir os parâmetros;
5. A cláusula GROUP BY;
6. A cláusula HAVING;
7. A cláusula ORDER BY.

No nosso exemplo, só passamos os dois primeiros parâmetros, indicando que a consulta deve retornar todos os registros existentes no banco de dados. O resultado desse método é um objeto do tipo Cursor, que representa o conjunto de resultados

da consulta. Para recuperar os resultados é necessário navegar (iterar) pelo Cursor, obtendo um resultado por vez. No código do exemplo, essa navegação no cursor é feita através dos métodos moveToFirst, moveToNext e isAfterLast. Esses métodos, respectivamente, posicionam o cursor no primeiro registro, movem o cursor para o próximo registro e verifica se o cursor chegou ao fim dos resultados.

Logo após iterarmos sobre o Cursor, adicionamos cada comentário encontrado na lista criada no início do método, para por fim fechar o *Cursor* e retorná-la. A recuperação dos campos de cada registro do resultado da consulta e a criação do objeto do tipo Comment é feita através de um método auxiliar, chamado cursorToComment, que foi criado especificamente para o exemplo.

Seguindo esse processo, temos toda a estrutura para trabalhar com um banco de dados em Android. Mais detalhes sobre as classes e métodos relacionados ao uso de SQLite no Android podem ser obtidos no material indicado na Leitura Complementar.

Vídeo 04 - Frameworks de ORM

Atividade 01

1. Acrescente um novo método ao DAO, para permitir a atualização de um comentário já existente.
2. Desenvolva uma Activity para testar os códigos mostrados nesta aula. Deve ser possível criar, remover e listar os comentários na base de dados, além de atualizar um comentário já existente.

Leitura Complementar

- Data Storage. Disponível em: <http://developer.android.com/guide/topics/data/data-storage.html> [Acessado em 26 mai. 2015.]

- Core J2ee Patterns - Data Access Object. Disponível em: <http://www.oracle.com/technetwork/java/dataaccessobject-138824.html> [Acessado em 26 mai. 2015.]

Resumo

Nesta aula, estudamos como o Android trabalha com bancos de dados, através do SQLite. Mostramos, também, como criar uma estrutura de trabalho com uma base de dados utilizando as classes de suporte disponibilizadas pelo próprio Android.

Autoavaliação

1. Porque o Android optou por disponibilizar uma estrutura de banco de dados SQLite no lugar de outras tecnologias de banco de dados?

2. Mostre quais modificações teríamos que fazer para adicionar uma nova tabela no banco de dados mostrado durante a nossa aula.

3. Busque na documentação indicada na seção de Leitura Complementar os demais métodos de movimentação do Cursor e explique como eles poderiam ser usados no exemplo apresentado na aula.

Referências

ANDROID DEVELOPERS. 2012. Disponível em: <http://developer.android.com/index.html> [Acessado em 26 mai. 2015].

DIMARZIO, J. **Android: A Programmer'sGuide. New York:McGraw-Hill, 2008.** Disponível em: <http://books.google.com.br/books?id=hoFI5pxjGesC> [Acessado em 14 set. 2012].

HASEMAN, C. **Android Essentials. Berkeley, CA. USA: Apress, 2008.** Disponível em: <http://books.google.com.br/books?id=ZthJlG4o-2wC> [Acessado em 14 set. 2012].

MEIER, R. **Professional Android 2 Application Development.** John Wiley e Sons, 2010.