

PERSISTÊNCIA DE DADOS NO ANDROID



“Muitas vezes, as pessoas não sabem o que elas querem até que você mostre a elas.” (Steve Jobs)

CONTEÚDO

- **Internal Storage**
- **Shared Preferences**
- **SQLite Databases**



PERSISTÊNCIA DE DADOS

- Existem 3 formas principais de armazenar dados no dispositivo
 - **Arquivos comuns (formato texto ou binário)**
 - Classes FileInputStream e FileOutputStream
 - Armazenados em /data/data/<pacote>/files
 - **Arquivos de preferências**
 - Classe SharedPreferences
 - **Bancos de Dados usando SQLite**
 - Classes SQLiteDatabase, SQLiteOpenHelper
 - Armazenados em /data/data/<pacote>/databases



CLASSE ANDROID.OS.ENVIRONMENT

MÉTODO	FUNÇÃO
<code>static File getDataDirectory()</code>	Retorna o diretório de dados do Android.
<code>static File getDownloadCacheDirectory()</code>	Retorna o diretório onde o Android salva os downloads e cache.
<code>static File getExternalStorageDirectory()</code>	Retorna o diretório de armazenamento externo (p.ex. cartão de memória)
<code>static File getExternalStoragePublicDirectory(tipo)</code>	Retorna o diretório público de armazenamento externo para um tipo de arquivo. Tipo pode ser: <code>DIRECTORY_DCIM</code> , <code>DIRECTORY_DOWNLOADS</code> , <code>DIRECTORY_MOVIES</code> , <code>DIRECTORY_MUSIC</code> , <code>DIRECTORY_PICTURES</code> , etc.
<code>static String getExternalStorageState()</code>	Retorna o estado do dispositivo de armazenamento externo principal. Retorno pode ser: <code>MEDIA_MOUNTED</code> (disponível para uso leitura e escrita), <code>MEDIA_MOUNTED_READ_ONLY</code> (disponível apenas para leitura), <code>MEDIA_REMOVED</code> (dispositivo não presente), <code>MEDIA_SHARED</code> (dispositivo presente, não montado e compartilhado via USB). Outros: <code>MEDIA_UNMOUNTABLE</code> , <code>MEDIA_UNMOUNTED</code> , <code>MEDIA_BAD_REMOVAL</code> , <code>MEDIA_CHECKING</code>
<code>static boolean isExternalStorageRemovable()</code>	Se o dispositivo externo de armazenamento é removível ou não.

ARQUIVOS COMUNS

- Métodos da classe Context

FileInputStream
openFileInput(String name)

Abre um arquivo privado associado a aplicação para leitura. Argumento name não pode conter '/'.

FileOutputStream
openFileOutput(String name,
int mode)

Abre um arquivo privado associado a aplicação para escrita. Argumento mode pode ser Context.MODE_PRIVATE ou Context.MODE_APPEND.

- **FileInputStream**: contém métodos para **ler bytes**
- **FileOutputStream**: contém métodos para **escrever bytes**
- **InputStreamReader**: pode ser construído a partir de um **FileInputStream** e contém métodos para **ler caracteres**
- **OutputStreamWriter**: pode ser construído a partir de um **FileOuputStream** e contém métodos para **escrever caracteres**



ARQUIVOS COMUNS

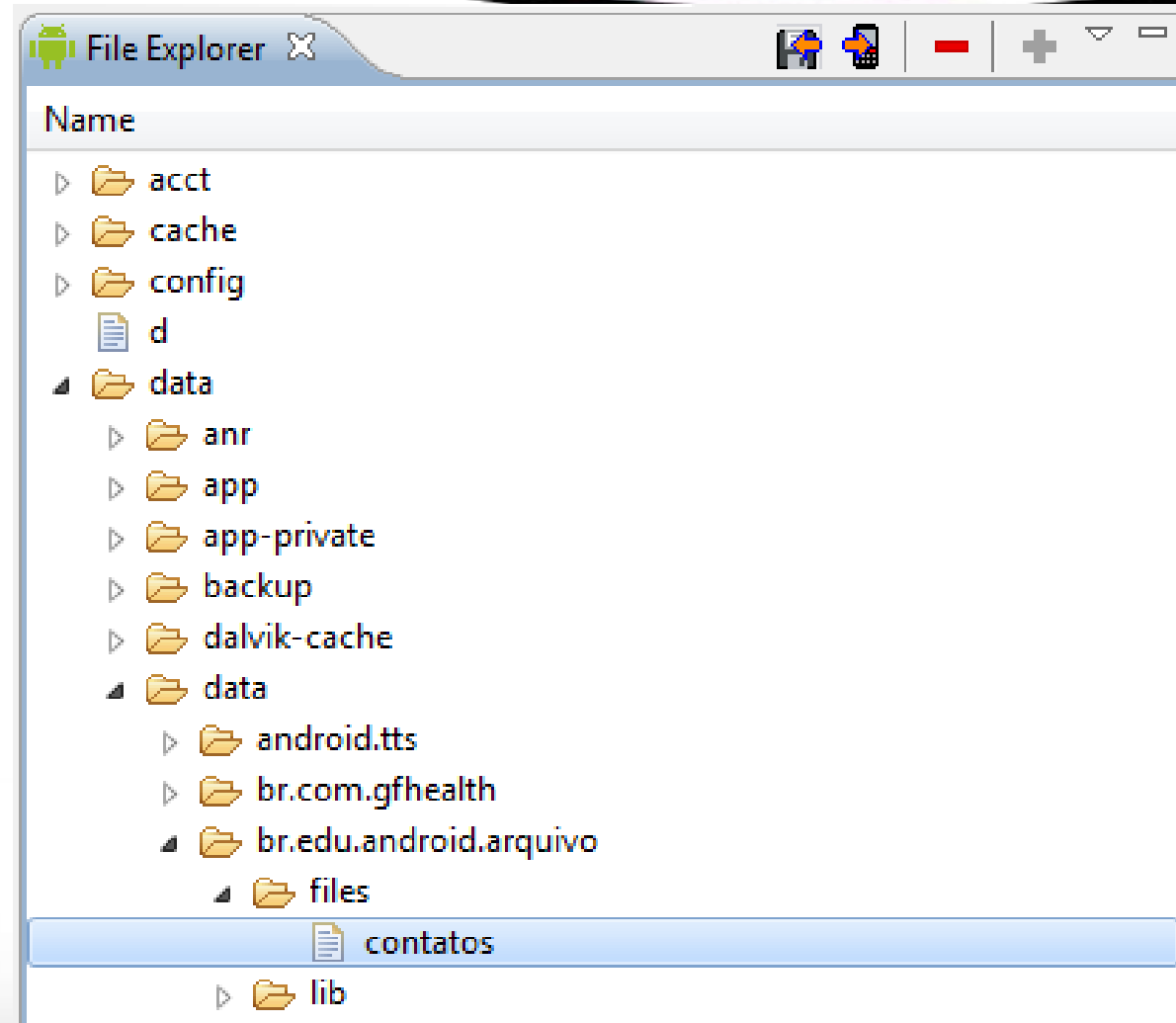
- Exemplo

```
try {  
    // abrindo o arquivo dados para escrita  
    FileOutputStream fos = openFileOutput(ARQUIVO, MODE_PRIVATE);  
    OutputStreamWriter osw = new OutputStreamWriter(fos);  
  
    osw.write("Nome: Roberson Alves\n");  
    osw.write("email: roberson.alves@unoesc.edu.br\n");  
    osw.close();  
  
    // abrindo o arquivo de dados para leitura  
    FileInputStream fis = openFileInput(ARQUIVO);  
    InputStreamReader isr = new InputStreamReader(fis);  
    StringBuilder sb = new StringBuilder();  
    int ch;  
    while ((ch = isr.read()) != -1) {  
        sb.append((char)ch);  
    }  
    TextView txtConteudo = (TextView) findViewById(R.id.txtConteudo);  
    txtConteudo.setText(sb.toString());  
    isr.close();  
  
} catch (FileNotFoundException e) {  
    Log.e("ExemploArquivoActivity.class", e.getMessage());  
} catch (IOException e) {  
    Log.e("ExemploArquivoActivity.class", e.getMessage());  
}
```



ARQUIVOS COMUNS – LOCALIZAÇÃO

- Exemplo



ANDROID.CONTENT.SHAREDREFERENCES

- Classe para armazenar preferências ou configurações da aplicação
- É possível usar quantas quiser, desde que associadas a um nome de arquivo
- Guarda informações na forma de pares: **chave-valor**
- Para obter a instância desta classe associada ao nome de arquivo “preferencias”:

```
SharedPreferences pref =  
    Context.getSharedPreferences("preferencias", 0);
```

- Se o arquivo ainda não existir será criado
- Para editar usar a classe **SharedPreferences.Editor**

```
SharedPreferences.Editor editor = prefs.edit();
```


ANDROID.CONTENT.SHAREDREFERENCES

• Métodos

boolean contains(String key)	Retorna true caso a chave key exista.
boolean getBoolean(String key, boolean) float getFloat(String key, float) int getInt(String key, int) long getLong(String key, long) String getString(String key, String)	Retorna o valor de uma chave. O segundo parâmetro indica o valor default que deve ser retornado caso a chave não exista. Note que é necessário que o desenvolvedor saiba o tipo do dado armazenado na chave. Caso a chave exista mas o tipo seja diferente do especificado no método será lançado um ClassCastException .
SharedPreferences.Editor	Retorna um editor para as preferências que permite editar e salvar informações.



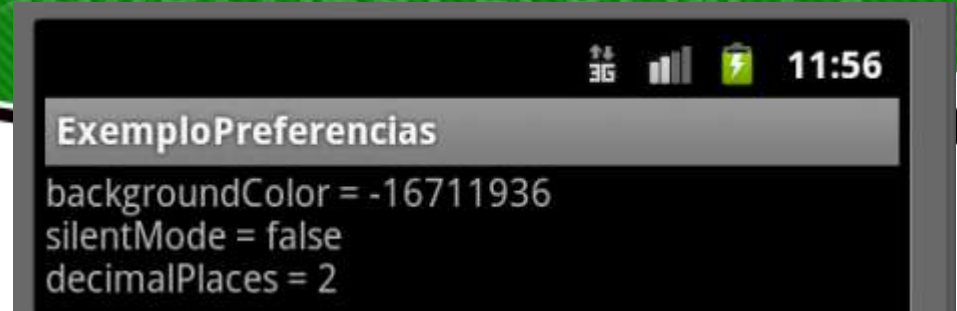
ANDROID.CONTENT.SHAREDREFERENCES.EDITOR

- Usada para editar as preferências

<code>boolean commit()</code>	Salva as preferências no objeto <code>SharedPreferences</code> associado e em disco. Operação síncrona, ou seja, só retorna após ter salvo em disco retornando <code>true</code> em caso de sucesso, ou <code>false</code> senão.
<code>void apply()</code>	Aplica os novos valores das preferências ao objeto em memória e retorna. O armazenamento em disco será feito de forma assíncrona, ou seja, não é possível saber se houve algum error durante o armazenamento.
<code>clear()</code>	Remove todos os valores de preferências do objeto.
<code>putBoolean(String key, boolean)</code> <code>putFloat(String key, float)</code> <code>putInt(String key, int)</code> <code>putLong(String key, long)</code> <code>putString(String key, String)</code>	Adiciona ou altera o valor de uma chave. O nome do método e o segundo parâmetro indicam o tipo do valor da chave.
<code>remove(String key)</code>	Remove a chave definida por <code>key</code> e consequentemente o seu valor.

ANDROID.CONTENT.SHAREDREFERENCES

- Exemplo



```
// recuperando a instância das preferências, se não existir cria ...
SharedPreferences prefs = getSharedPreferences(ARQUIVO_PREFERENCIAS,
    Context.MODE_PRIVATE);

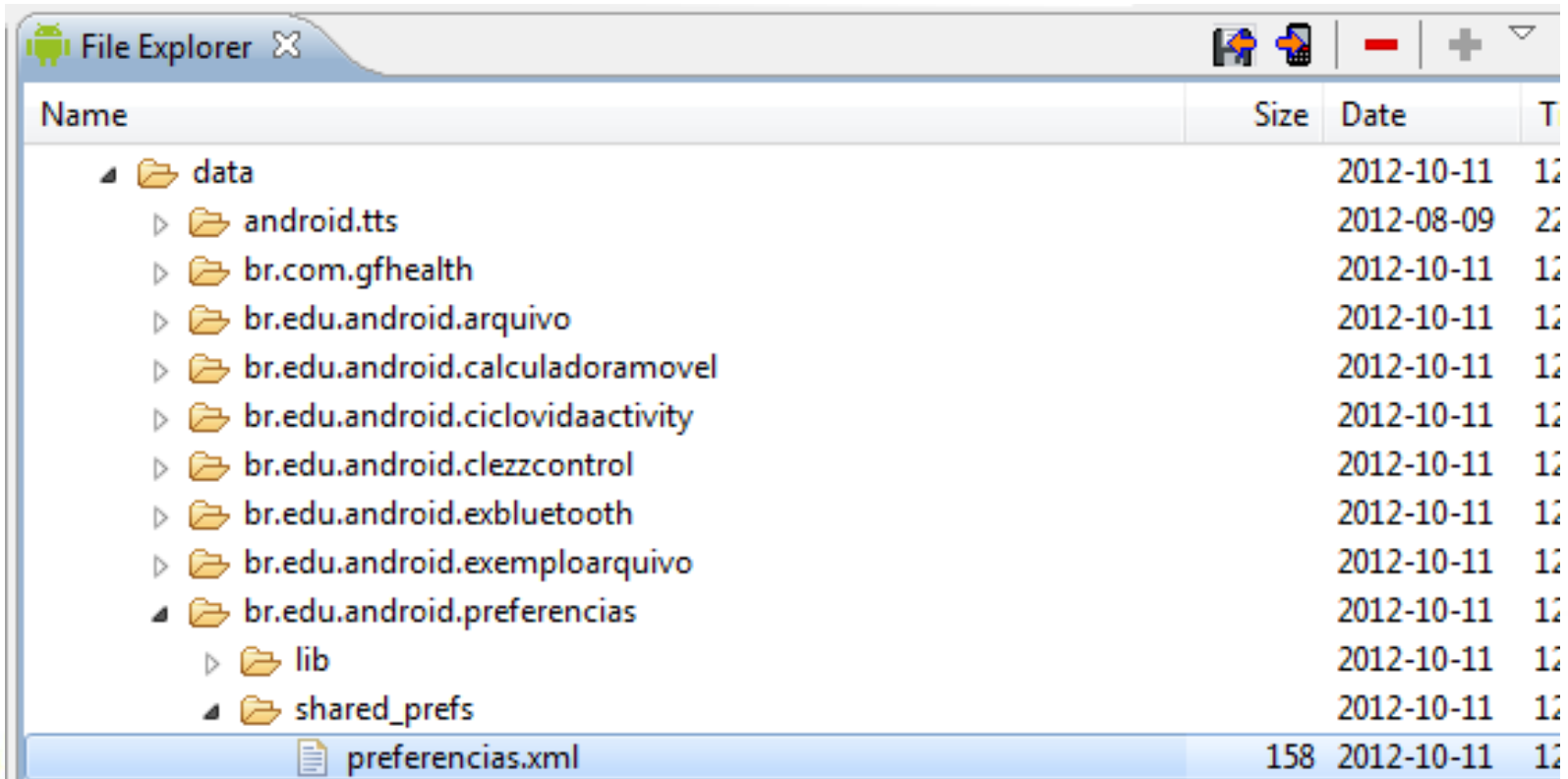
// criando um editor para alterar as preferências ...
SharedPreferences.Editor editor = prefs.edit();
editor.putInt("backgroundColor", Color.GREEN);
editor.putInt("decimalPlaces", 2);
// confirma as alterações
editor.commit();

TextView txtPreferencias = (TextView) findViewById(R.id.txtPreferencias);
// lê as configurações e exibe na tela
StringBuilder sb = new StringBuilder();
for (Entry<String, ?> entry : prefs.getAll().entrySet()) {
    sb.append(entry.getKey() + " = "
        + entry.getValue().toString() + "\n");
}
txtPreferencias.setText(sb.toString());
```



ANDROID.CONTENT.SHAREDREFERENCES

- Exemplo



File Explorer

Name	Size	Date	T
data		2012-10-11	12
android.tts		2012-08-09	22
br.com.gfhealth		2012-10-11	12
br.edu.android.arquivo		2012-10-11	12
br.edu.android.calculadoramovel		2012-10-11	12
br.edu.android.ciclovidaactivity		2012-10-11	12
br.edu.android.clezzcontrol		2012-10-11	12
br.edu.android.exbluetooth		2012-10-11	12
br.edu.android.exemploarquivo		2012-10-11	12
br.edu.android.preferencias		2012-10-11	12
lib		2012-10-11	12
shared_prefs		2012-10-11	12
preferencias.xml	158	2012-10-11	12



BANCO DE DADOS - SQLITE

- Consiste em tabelas onde dados podem inseridos, modificados ou excluídos
- O Android suporta um o modelo lógico relacional para armazenamento de dados
- No Android, é utilizado o SQLite para gerenciar as tabelas de bancos de dados

— www.sqlite.org

- Is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed SQL database engine in the world.

BANCO DE DADOS - SQLITE

- **SQLite implementa a maior parte do padrão SQL-92;**
- **Suporte parcial para triggers e queries complexas;**
- **SQLITE não implementa integridade referencial seguindo o tradicional modelo de restrições com chave estrangeira.**

BANCO DE DADOS - SQLITE

- **API do Android (executar SQL)**
 - Adotado aqui pois permite criar o BD pela aplicação
- **Cliente gráfico do SQLite**
 - SQLite Expert Personal (gratuito)
 - <http://www.sqliteexpert.com/index.html>
- **Usando SQL via linha de comando**
 - Programa na pasta do SDK do Android
 - `<android_sdk>\tools\sqlite3.exe`
- **Nos 2 últimos, após criar, é preciso mover o arquivo para a pasta /data/data/<pacote>/databases do emulador**
 - No eclipse (com emulador aberto):
 - `Window > Show view > Other... > FileExplorer`

ANDROID.DATABASE.SQLITE.SQLITEOPENHELPER

- Auxilia abertura e criação de um banco de dados**

SQLiteOpenHelper(Context, String name, SQLiteDatabase.CursorFactory, int version)	Cria um objeto para auxiliar no gerenciamento da base de dados.
SQLiteDatabase getReadableDatabase()	Cria ou abre um banco de dados apenas para leitura.
SQLiteDatabase getWritableDatabase()	Cria ou abre um banco de dados para leitura e escrita.
void onCreate(SQLiteDatabase db)	Chamado quando o banco de dados precisa ser criado, ou seja, não existe.
void onOpen(SQLiteDatabase db)	Chamado quando o banco de dados é aberto.
void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)	Chamado quando a versão do banco de dados sendo aberto é diferente da versão existente.

ANDROID.DATABASE.SQLITE.SQLITEDATABASE

- Representa o banco de dados e executa operações de consulta, inclusão, alteração e exclusão de registros

static SQLiteDatabase openDatabase(String path, CursorFactory factory, flags)	Abre banco de dados de acordo com os flags: OPEN_READWRITE, OPEN_READONLY, CREATE_IF_NECESSARY, NO_LOCALIZED_COLLATORS.
static SQLiteDatabase openOrCreateDatabase(String path, CursorFactory factory) static SQLiteDatabase openOrCreateDatabase(File file, CursorFactory factory)	Abre ou cria banco de dados. Equivalente a usar openDatabase(...) com flags = CREATE_IF_NECESSARY
boolean isOpen()	Verifica se o banco de dados está aberto.
void close()	Fecha o banco de dados.
void execSQL(String sql)	Executa script SQL que não seja SELECT. Exemplo: CREATE TABLE, INSERT, UPDATE, etc.

ANDROID.DATABASE.SQLITE.SQLITEDATABASE

Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)
Cursor query(table, columns, selection, selectionArgs, groupBy, having, orderBy, limit)
Cursor query(boolean distinct, table, columns, selection, selectionArgs, groupBy, having, orderBy, String limit)

Mostra e executa um comando SQL.

long insert(table, null, ContentValues values)

Insere um registro e retorna o id.

int update(table, ContentValues values, whereClause, whereArgs)

Altera registro(s) e retorna quantidade de linhas modificadas.

int delete(table, whereClause, whereArgs)

Deleta registro(s) e retorna quantidade de linhas modificadas.

EXEMPLO 01 – CRIANDO/CONECTANDO AO BANCO DE DADOS

Classe estende SQLiteOpenHelper e permite gerenciar SQLiteDatabase's.

```
public final class DataBaseHelper extends SQLiteOpenHelper {
    private static final String DATABASE_NAME = "rango.db";
    private static final int DATABASE_VERSION = 4;

    private static DataBaseHelper instance;

    public static synchronized DataBaseHelper getHelper(Context context) {

        * Helper para auxiliar na manipulação da base de dados
        private DataBaseHelper(Context context) {
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }

        @Override
        public void onCreate(SQLiteDatabase db) {
```



EXEMPLO 01 – CRIANDO/CONECTANDO AO BANCO DE DADOS

Implementação do método onCreate() do banco de dados.

```
@Override
public void onCreate(SQLiteDatabase db) {
    // lendo arquivo de script para criação da base de dados
    String scriptCreate = new FileUtil()
        .getFileContent("/br/com/rango/scripts/create.sql");
    String[] instrucoes = scriptCreate.split(";");
    for (int i = 0; i < instrucoes.length; i++) {
        if (instrucoes[i] != null && !instrucoes[i].trim().equals("")) {
            db.execSQL(instrucoes[i]);
        }
    }
}
```

Implementação do método onUpgrade() do banco de dados.

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    String scriptDrop = new FileUtil()
        .getFileContent("/br/com/rango/scripts/drop.sql");
    String[] instrucoes = scriptDrop.split(";");
    for (int i = 0; i < instrucoes.length; i++) {
        if (instrucoes[i] != null && !instrucoes[i].trim().equals("")) {
            try {
                db.execSQL(instrucoes[i]);
            } catch (SQLException ex) {
            }
        }
    }
}
```



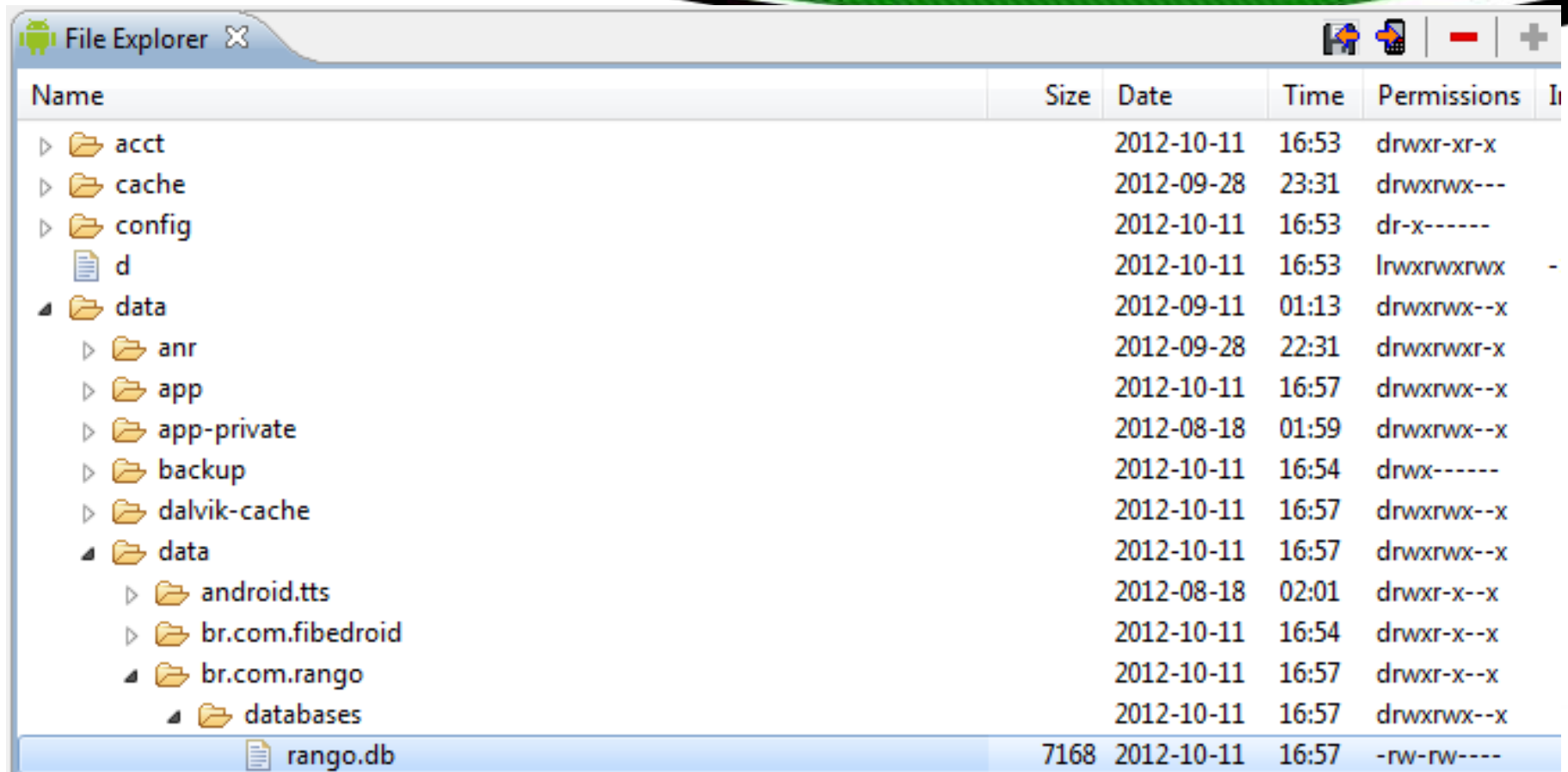
EXEMPLO 01 – CRIANDO/CONECTANDO AO BANCO DE DADOS

Recuperando uma instância do banco de dados para escrita.

```
public ContatoHelper(Context context) {  
    this.db = DataBaseHelper.getHelper(context).getWritableDatabase();  
}
```



LOCALIZAÇÃO DA BASE DE DADOS



File Explorer

Name	Size	Date	Time	Permissions	
acct		2012-10-11	16:53	drwxr-xr-x	
cache		2012-09-28	23:31	drwxrwx---	
config		2012-10-11	16:53	dr-x-----	
d		2012-10-11	16:53	lrwxrwxrwx	-
data		2012-09-11	01:13	drwxrwx--x	
anr		2012-09-28	22:31	drwxrwxr-x	
app		2012-10-11	16:57	drwxrwx--x	
app-private		2012-08-18	01:59	drwxrwx--x	
backup		2012-10-11	16:54	drwx-----	
dalvik-cache		2012-10-11	16:57	drwxrwx--x	
data		2012-10-11	16:57	drwxrwx--x	
android.tts		2012-08-18	02:01	drwxr-x--x	
br.com.fibedroid		2012-10-11	16:54	drwxr-x--x	
br.com.rango		2012-10-11	16:57	drwxr-x--x	
databases		2012-10-11	16:57	drwxrwx--x	
rango.db	7168	2012-10-11	16:57	-rw-rw----	



ANDROID.CONTENT.CONTENTVALUES

- **ContentValues**

- Usado para armazenar o conteúdo de um registro para uma operação
- Equivalente a um HashMap
 - put(String key, valor)

```
ContentValues values = new ContentValues();
values.put("nomcon", contato.getNome());
if (contato.getTelRes() != null)
    values.put("telrescon", contato.getTelRes());
if (contato.getTelCel() != null)
    values.put("telcelcon", contato.getTelCel());
if (contato.getTelTra() != null)
    values.put("teltracon", contato.getTelTra());
if (contato.getEmail() != null)
    values.put("emacon", contato.getEmail());
if (contato.getDataNascimento() != null)
    values.put("datnascon", contato.getDataNascimento().getTime());
values.put("sexcon", contato.getSexo());
values.put("fotcon", contato.getFoto());
values.put("latcon", contato.getLatitude());
values.put("loncon", contato.getLongitude());
```



CURSORES

- Permite diversas operações sobre um conjunto de dados retornados
 - Posicionamento (*isFirst()*, *isLast()*, *isBeforeFirst()*, *isAfterLast()*);
 - Navegação(*moveToFirst()*, *moveToLast()*, *moveToNext()*, *moveToPrevious()*, *move(n)*);
 - Extração de campos(*getInt*, *getString*, *getFloat*, *getBlob*, *getDate*, etc.);
 - Inspeção de esquema(*getColumnName*, *getColumnNames*, *getColumnIndex*, *getColumnCount*, *getCount*).



CURSORES

- **Cursor**
 - Implementado por **ANDROID.DATABASE.CURSOR**
 - Equivalente a um **ResultSet** no **JDBC**
 - Lista de resultados de uma consulta no banco



EXEMPLO 02 – MANIPULAÇÃO DA BASE DE DADOS – CURSORES

```
public List<Contato> getAll() {  
    List<Contato> contatos = new ArrayList<Contato>();  
    Cursor c = this.db.query(this.getTableName(), null, null, null, null,  
        null, "nomcon COLLATE NOCASE ASC");  
    if (c != null) {  
        c.moveToFirst();  
        while (!c.isAfterLast()) {  
            contatos.add(this.fillContato(c));  
            c.moveToNext();  
        }  
        c.close();  
    }  
    return contatos;  
}
```



EXEMPLO 03 – MANIPULAÇÃO DA BASE DE DADOS – INSERT

```
public void insert(Contato contato) {  
    ContentValues values = new ContentValues();  
    values.put("nomcon", contato.getNome());  
    if (contato.getTelRes() != null)  
        values.put("telrescon", contato.getTelRes());  
    if (contato.getTelCel() != null)  
        values.put("telcelcon", contato.getTelCel());  
    if (contato.getTelTra() != null)  
        values.put("teltracon", contato.getTelTra());  
    if (contato.getEmail() != null)  
        values.put("emacon", contato.getEmail());  
    if (contato.getDataNascimento() != null)  
        values.put("datnascon", contato.getDataNascimento().getTime());  
    values.put("sexcon", contato.getSexo());  
    values.put("fotcon", contato.getFoto());  
    values.put("latcon", contato.getLatitude());  
    values.put("loncon", contato.getLongitude());  
    this.db.insert("contato", null, values);  
}
```



EXEMPLO 04 – MANIPULAÇÃO DA BASE DE DADOS – UPDATE

```
public void update(Contato contato) {
    ContentValues values = new ContentValues();
    values.put("nomcon", contato.getNome());
    if (contato.getTelRes() != null)
        values.put("telrescon", contato.getTelRes());
    if (contato.getTelCel() != null)
        values.put("telcelcon", contato.getTelCel());
    if (contato.getTelTra() != null)
        values.put("teltracon", contato.getTelTra());
    if (contato.getEmail() != null)
        values.put("emacon", contato.getEmail());
    if (contato.getDataNascimento() != null)
        values.put("datnascon", contato.getDataNascimento().getTime());
    values.put("sexcon", contato.getSexo());
    values.put("fotcon", contato.getFoto());
    values.put("latcon", contato.getLatitude());
    values.put("loncon", contato.getLongitude());
    this.db.update("contato", values, "codcon = ?", new String[] { contato.getCodigo()
        .toString() });
}
```



EXEMPLO 05 – MANIPULAÇÃO DA BASE DE DADOS – DELETE

```
public void delete() {  
    this.db.delete("contato", "codcon = ?", new String[]{"1"});  
}
```




SQLITE DATATYPES

- **Suportados na versão 3**
 - **NULL**. The value is a NULL value.
 - **INTEGER**. The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.
 - **REAL**. The value is a floating point value, stored as an 8-byte IEEE floating point number.
 - **TEXT**. The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE).
 - **BLOB**. The value is a blob of data, stored exactly as it was input.



SQLITE TRASACTIONS

- Controle transacional é desejável para garantir a consistência ou evitar perdas de dados;
- Normalmente uma transação segue a premissa: **“Sucesso completo ou falha total!”**;
-  O Android/SQLite possuem suporte para controle transacional.

SQLITE TRASACTIONS - EXEMPLO

Caso não seja realizado um `setTransactionSuccessful()` = `commit`, será realizado automaticamente um `rollback`.

```
db.beginTransaction();  
try {  
    //perform your database operations here ...  
    db.setTransactionSuccessful(); //commit your changes  
}  
catch (SQLException e) {  
    //report problem  
}  
finally {  
    db.endTransaction();  
}
```



OrmLite - Lightweight Java ORM

Supports Android and SQLite

- Persistência de objetos java para bases SQL;
- Suporte a várias bases de dados SQL incluindo SQLite;
- Suporte a anotações;
- Mais informações em:
http://ormlite.com/sqlite_java_android_orm.shtml



REFERÊNCIAS

- Data Storage. 2012. Disponível em: < <http://developer.android.com/guide/topics/data/data-storage.html> >. Acesso em: 20 ago. 2012.
- LECHETA, Ricardo R. **Google android:** aprenda a criar aplicações para dispositivos móveis com o android SDK. 2. ed., rev. e ampl. São Paulo: Novatec, 2010. 608 p. ISBN 9788575222447.
- MURPHY, Mark L. **Beginning android.** New York, USA: Apress, 2009. xxii, 361 p. ISBN 9781430224198.

