01

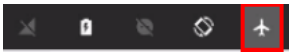
Adicionando o campo

Transcrição

A nossa aplicação está funcionando muito bem, estamos conseguindo enviar, receber e deletar informações. Porém, tudo o que estamos fazendo é no cenário perfeito, onde a nossa aplicação e servidor sempre estarão online e conectados.

Mas não é o que acontece no dia a dia. É bem comum que os celulares percam a conexão, por isso, precisamos estar preparados para trabalhar com esse cenário.

Em um primeiro momento, selecionaremos o modo avião do celular para simular que estamos sem conexão.



Agora que já não estamos mais conectados, podemos tentar adicionar um novo aluno e verificar o comportamento. Conseguiremos adicionar normalmente o novo aluno. Se verificarmos, o novo aluno ainda não consta no servidor, justamente por não estarmos conectados. Mas e quando conectarmos novamente, o que vai acontecer?

Vamos retirar o celular do modo avião, para que a conexão volte a ser estabelecida. Porém, mesmo conectado, o novo aluno ainda não consta no servidor. Se fizermos um *swipe* para atualizar, ainda assim o aluno não foi enviado para o servidor.

Isso acontece porque, atualmente, o nosso processo adiciona o aluno no banco de dados local e faz uma requisição para o servidor enviar o novo aluno. Porém, se essa requisição falhar como nos casos em que estamos offline, não será feita uma nova posteriormente, ainda quando a conexão for restabelecida. Desta forma, qualquer aluno que for adicionado offline, seus dados não serão enviados para o servidor.

Vamos implementar a função de quando a conexão for estabelecida ou fazer algum movimento - tipo um *swipe* - e as informações serão sincronizadas com o servidor. Para que a implementação funcione, nós colocaremos algum tipo de marcação no dado informando que ele ainda não foi sincronizado com o servidor, semelhante a forma do *soft delete*.

Na classe `Aluno`, adicionaremos mais um atributo na classe para indicar se a sincronização com o servidor ocorreu ou não. Vamos adicionar o campos `private int sincronizado`. Não esquecendo de colocar os ***Getters*** e ***Setters***

```
// ...

private String id;
private String nome;
private String endereco;
private String telefone;
private String site;
private double nota;
private String caminhoFoto;
private int desativado;
private int sincronizado;

public void setSincronizado(ind sincronizado){
    this.sincronizado = sincronizado;
```

```
}

public int setSincronizado(){
    return this.sincronizado;
}

// ...
```

Com o novo campo adicionado, precisamos ir agora no `AlunoDAO` e criar uma nova *migration* que incluirá o novo campo na tabela. Dentro do `AlunoDAO`, no método `onUpgrade()`, adicionaremos uma nova condição para o `switch case`. Iremos colocar um `case 4`, que terá a *query* responsável por adicionar a nova coluna na tabela de alunos.

```
// ...

case 4:
    String adicionaCampoSicronizado = "ALTER TABLE Alunos ADD COLUMN sincronizado";

// ...
```

O problema dessa *query* é que nós estamos adicionando uma nova tabela mas sem nenhum valor padrão. Ou seja, ele vai criar nova coluna e colocar o valor como `null`, isto não é o que queremos. Logo, definiremos um valor padrão como **0**, que significa "não sincronizado". Colocamos na query `DEFAULT 0`.

Com a *query* pronta basta executá-la.

```
// ...

case 4:
    String adicionaCampoSicronizado = "ALTER TABLE Alunos ADD COLUMN sincronizado DEFAULT 0";
    bd.execSQL(adicionaCampoSicronizado);

// ...
```

Como estamos adicionando uma nova coluna na tabela, replicaremos essa instrução na criação da tabela. Caso o usuário desinstale a aplicação ou se trate da primeira instalação, ela já será criada com a nova coluna.

No método `onCreate()`, adicionaremos a *string* `sql` à instrução `sincronizado INT DEFAULT 0`. Nosso método vai ficar assim.

```
// ...

@Override
public void onCreate(SQLiteDatabase db){
    String sql = "CREATE TABLE Alunos (id CHAR(36) PRIMARY KEY, " +
    "nome TEXT NOT NULL, " +
    "endereco TEXT, " +
    "telefone TEXT, " +
    "site TEXT, " +
    "nota REAL, " +
    "caminhoFoto TEXT, " +
    "sincronizado INT DEFAULT 0);";
```

```
        db.execSQL(sql);
    }
    // ...
```

Temos que mudar a versão do construtor de **4** para **5**.

```
    // ...

    public AlunoDAO(Context context) {
        super(context, "Agenda", null, 5);
    }

    // ...
```

No *SQLite*, nós temos os métodos que pegam os dados e o que popula. Depois, adicionaremos `dados.put("sincronizado", aluno.getSincronizado());` , no método `pegaDadosDoAluno()` :

Com as alterações o código ficou assim:

```
    // ...

    @NonNull
    private ContentValues pegaDadosDoAluno(Aluno aluno) {
        ContentValues dados = new ContentValues();
        dados.put("id", aluno.getId());
        dados.put("nome", aluno.getNome());
        dados.put("endereco", aluno.getEndereco());
        dados.put("telefone", aluno.getTelefone());
        dados.put("site", aluno.getSite());
        dados.put("nota", aluno.getNota());
        dados.put("caminhoFoto", aluno.getCaminhoFoto());
        dados.put("sincronizado", aluno.getSincronizado());
        return dados;
    }

    // ...
```

No método `populaAlunos()` , nós vamos *settar* o valor do atributo `sincronizado` utilizando o comando `aluno.setSincronizado(c.getInt(c.getColumnIndex("sincronizado")));` .

```
    @NonNull
    private List<Aluno> populaAlunos(Cursor c) {
        List<Aluno> alunos = new ArrayList<Aluno>();
        while (c.moveToNext()) {
            Aluno aluno = new Aluno();
            aluno.setId(c.getString(c.getColumnIndex("id")));
            aluno.setNome(c.getString(c.getColumnIndex("nome")));
            aluno.setEndereco(c.getString(c.getColumnIndex("endereco")));
            aluno.setTelefone(c.getString(c.getColumnIndex("telefone")));
            aluno.setSite(c.getString(c.getColumnIndex("site")));
            aluno.setNota(c.getDouble(c.getColumnIndex("nota")));
            aluno.setCaminhoFoto(c.getString(c.getColumnIndex("caminhoFoto")));
```

```
            aluno.setSincronizado(c.getInt(c.getColumnIndex("sincronizado")));

            alunos.add(aluno);
        }
        return alunos;
    }
```

Com isso já conseguimos colocar a marcação no aluno, armazenar e pegar esse valor do banco de dados. Mas como podemos ter certeza se a marcação está funcionando? Simples, podemos ir na classe `ListaAlunosActivity`, na qual temos o método `carregaLista()`. Nele, fazemos um `forEach` para imprimir informações do aluno.

Vamos colocar o ***Log*** para verificar o valor do atributo `sincronizado`. Como ainda não implementamos toda a funcionalidade, todos os alunos vão estar com o valor **0**. O `carregaLista()` deve ficar da seguinte maneira:

```
// ...

private void carregaLista() {
    AlunoDAO dao = new AlunoDAO(this);
    List<Aluno> alunos = dao.buscaAlunos();

    for (Aluno aluno :
            alunos) {
        Log.i("aluno sincronizado", String.valueOf(aluno.getSincronizado()));
    }

    dao.close();

    AlunosAdapter adapter = new AlunosAdapter(this, alunos);
    listaAlunos.setAdapter(adapter);
}

// ...
```

Analisando o Android Monitor veremos que temos cinco alunos com valor sincronizado **0**.

```
I/aluno sincronizado: 0
I/aluno sincronizado: 0
I/aluno sincronizado: 0
I/aluno sincronizado: 0
I/aluno sincronizado: 0
```

Tudo funcionou, já demos o primeiro passo e implementamos a marcação nos alunos.