04

Desativando o aluno internamente

Transcrição

Vamos implementar o *Soft Delete* no código. Porém, verificaremos se o aluno tem a capacidade de receber a informação na qual indica se ele está ativo ou inativo.

Com o atalho "Shift + Shift", e preencheremos "Aluno" no campo de busca que aparecerá, os dados serão filtrados, e selecionaremos a classe `Aluno` .

Analisando a classe, vamos notar que a classe possui os atributos de um aluno e também tem *flags* como `private int desativado` e `private int sincronizado` . Ou seja, ela suporta o conceito de *Soft Delete*.

A aplicação precisa ser capaz de salvar as informações internamente. Vamos fazer alguns ajustes no *SQLite* para que isso seja possível.

Vamos acessar a classe `AlunoDAO` e verificar a *query* no método `onCreate()` , notamos que não possuímos uma forma de armazenar a informação de que o aluno está ativado ou desativado. Iremos atualizar o nosso banco de dados.

No método `onUpgrade()` , vamos criar um nova *migration* para adicionar o campo "desativado", que vai trabalhar com o valor `1` para desativado e `0` para ativado. Não podemos esquecer de executar a *query*.

```
// ...

case 5:
    String adicionaCampoDesativado =
        "ALTER TABLE Alunos ADD COLUMN desativado INT DEFAULT 0";
    db.execSQL(adicionaCampoDesativado);

// ...
```

Antes de executar, voltaremos ao método `onCreate()` para adicionar o campo na *query*, além de mudar no construtor para a versão 6.

```
// ...

public AlunoDAO(Context context) {
    super(context, "Agenda", null, 6);
}

@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," +
                "desativado INT DEFAULT 0);";
        db.execSQL(sql);
    }

    // ...
```

Faltam mais alguns detalhes. Sempre que colocarmos um campo novo, precisaremos colocar no `pegaDadosDoAluno()` e no `populaAlunos()`. Vamos começar colocando o comando `dados.put("desativado", aluno.getDesativado);` no `pegaDadosDoAluno()`.

```
    // ...

    @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());
        dados.put("desativado", aluno.getDesativado());
        return dados;
    }
    // ...
```

Agora no `populaAlunos()`, adicionaremos a linha `aluno.setDesativado(c.getInt(c.getColumnIndex("desativado")));`.

```
    // ...

    @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")));
            aluno.setDesativado(c.getInt(c.getColumnIndex("desativado")));

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

    // ...
```

Se executarmos, veremos que tudo está funcionando normalmente, porém, ainda não fizemos nenhuma alteração no comportamento de remoção dos alunos em nossa aplicação.

Analisando o método `deleta()` da classe `AlunoDAO`, veremos que atualmente, estamos pegando uma instancia para poder escrever no banco de dados. Em seguida, vamos enviar os parâmetros na *query* e, logo, removeremos o aluno. Não é feita nenhuma verificação se o aluno está desativado.

```
// ...

public void deleta(Aluno aluno) {
    SQLiteDatabase db = getWritableDatabase();

    String[] params = {aluno.getId().toString()};
    db.delete("Alunos", "id = ?", params);
}

// ...
```

Faremos a verificação com um `if(aluno.estaDesativado())`. O método `estaDesativado()` já tínha sido implementado anteriormente. Caso seja verdadeiro, nós removeremos fisicamente o alunos.

```
// ...

public void deleta(Aluno aluno) {
    SQLiteDatabase db = getWritableDatabase();

    String[] params = {aluno.getId().toString()};
    if(aluno.estaDesativado()){
        db.delete("Alunos", "id = ?", params);
    }

}

// ...
```

Mas e se o aluno não estiver desativado? Caso ele não esteja, não vamos conseguir removê-lo fisicamente, por isso, iremos desativá-lo. Desta forma, a próxima remoção terá sucesso.

Como uma boa prática de orientação a objetos, delegaremos a função para a própria classe `Aluno` chamando no `else` o `aluno.desativa()`.

```
// ...

public void deleta(Aluno aluno) {
    SQLiteDatabase db = getWritableDatabase();

    String[] params = {aluno.getId().toString()};
    if(aluno.estaDesativado()){
        db.delete("Alunos", "id = ?", params);
    } else {
        aluno.desativa()
```

```
        }

    }

    // ...
```

Como o método `desativa()` ainda não existe, criaremos dentro da classe `Aluno`:

```
    // ...

    public void desativa() {
        this.desativado = 1;
        desincroniza();
    }

    // ...
```

Ainda é necessário salvar as informações, portanto, para avisar o banco de dados precisaremos usar o `altera(aluno)` no `else` do `aluno.desativa()`.

```
    // ...

    public void deleta(Aluno aluno) {
        SQLiteDatabase db = getWritableDatabase();

        String[] params = {aluno.getId().toString()};
        if(aluno.estaDesativado()){
            db.delete("Alunos", "id = ?", params);
        } else {
            aluno.desativa()
            altera(aluno);
        }
    }

    // ...
```

Vamos executar o aplicação e tentar remover um aluno em modo offline. Ao removermos o aluno, a lista será mantida, o que está errado?

Se olharmos a forma como buscamos os alunos no método `buscaAlunos()`, a *query* pedirá todos os alunos sem exceção. Deveríamos pegar apenas os alunos ativos.

```
    // ...

    public List<Aluno> buscaAlunos() {
        String sql = "SELECT * FROM Alunos WHERE desativado = 0;";
        SQLiteDatabase db = getReadableDatabase();
        Cursor c = db.rawQuery(sql, null);

        List<Aluno> alunos = populaAlunos(c);
        c.close();
```

```
        return alunos;
    }

    // ...
```

Executando novamente a aplicação, é possível perceber que o aluno removido anteriormente não aparece mais na lista. O próximo passo será, no momento em que conectarmos com o servidor, enviarmos o aluno desativado para que ele remova e replique a todos os clientes.