04

Iniciando processo de migration

Caso não tenha o projeto com as alterações da aula passada, você pode baixá-lo neste link (https://github.com/alura-cursos/android-sync/archive/7dd1aedc90896fb63a3dfbdcd55698d370901b0b.zip)

Agora que sabemos o que precisamos fazer para adaptar a nossa App para que funcione com UUID vamos colocar a mão no código! Como vimos, temos que realizar os seguintes passos:

- Realizar a migration no SQLite.
  - Adicionar novo campo para suportar UUID.
  - Gerar UUID para cada aluno.
- Alterar a classe `Aluno` para que o id suporte o UUID
- Modificar todos os pontos do sistema que trabalhavam com ids numéricos para agora atuar como `String` s (que é justamente o tipo de dado que usaremos para armazenar os UUIDs)

Realizando a migration

Para esse processo de refatoração do projeto vamos iniciar pela migration do SQLite. Portanto, vá até a classe `AlunoDAO` e dentro do método `onUpgrade()` (responsável por realizar as migrations) adicione um novo `case` com valor 2 que será o escopo da nova versão do nosso banco de dados. O nosso primeiro passo na migration é fazer com que o id, que atualmente está com o tipo `INTEGER` , seja do tipo compatível com o UUID, porém o SQLite não permite alterar o tipo de dado de uma coluna!

Embora não consigámos alterar o tipo de dado de uma coluna, podemos realizar alguns passos que permitirá a modificação do tipo da coluna, esses passos são:

- Criar uma tabela nova.
- Enviar todos os dados da tabela antiga para a nova.
- Apagar tabela antiga.
- Renomeando tabela nova para o nome da antiga.

Bora começar? Vamos lá

Criando nova tabela

Inicialmente vamos criar uma tabela nova, portanto, dentro do `case 2` , crie uma `String` com o nome `criandoTabelaNova` . Em seguida, atribua o valor da `String` abaixo para essa variável:

```
"CREATE TABLE Alunos_novo " +
    "(id CHAR(36) PRIMARY KEY," +
    "nome TEXT NOT NULL, " +
    "endereco TEXT, " +
    "telefone TEXT, " +
    "site TEXT, " +
    "nota REAL, " +
    "caminhoFoto TEXT);"
```

Repara que essa `String` representa de fato a tabela nova que contém as mesmas colunas da anterior, porém, com o tipo de dado `CHAR(36)` para o id que é justamente a quantidade de caracteres que forma um UUID. Em seguida, execute esse script com o método `execSQL()` enviando como pârametro a `String` `criandoTabelaNova` a partir do objeto `db` .

Enviando todos os dados da tabela antiga para nova

Agora que a tabela nova foi criada, precisamos de fato pegar todos dados da tabela antiga e enviar para a tabela nova. Para isso adicione a `String` com o nome `inserindoAlunosNaTabelaNova` e atribua o seguinte valor:

```
"INSERT INTO Alunos_novo " +
    "(id, nome, endereco, telefone, site, nota, caminhoFoto) " +
    "SELECT id, nome, endereco, telefone, site, nota, caminhoFoto " +
    "FROM Alunos"
```

Repara que neste script estamos fazendo uma inserção de todos os campos da tabela nova, para cada um dos registros da tabela antiga. Da mesma forma como foi feito no script de criação de tabela execute esse script com o método `execSQL()` passando a `String inserindoAlunosNaTabelaNova` como parâmetro .

Removendo tabela antiga

Agora que temos uma tabela nova com todos os dados da tabela antiga, basta apenas removermos a tabela antiga. Para isso adione a `String removendoTabelaAntiga` com o seguinte valor:

```
"DROP TABLE Alunos"
```

Então execute o script da mesma forma como os anteriores.

Alterando nome da tabela nova para o nome da tabela antiga

Por fim, precisamos de fato renomear a tabela nova com o mesmo nome da tabela antiga. Portanto adicione a `String alterandoNomeDaTabelaNova` com o seguinte valor:

```
"ALTER TABLE Alunos_novo " +
    "RENAME TO Alunos"
```

Da mesma forma como fizemos com os outros scripts, execute esse script também!

Mudando a versão do banco

Agora que finalizamos os passos necessários vamos mudar a versão do banco para que o SQLite invoque `onUpgrade()` com a nova versão que adicionamos. Para isso basta apenas incrementar 1 na versão atual do SQLite, ou seja, modifique o valor do construtor que atualmente é 2 para 3 conforme o código abaixo:

```
public class AlunoDAO extends SQLiteOpenHelper {
    public AlunoDAO(Context context) {
        super(context, "Agenda", null, 3);
    }
```

Mantendo o banco mais atualizado

Embora tenhamos indicado a versão do banco, o método `onUpgrade` só é acionado apenas quando mudamos de versão do banco com a qual já existe no celular. Em outras palavras, se o usuário instalar a nossa App pela primeira vez, ou então, desinstalar e instalar novamente, ele terá a tabela que está escrita no `onCreate()` do SQLite. Sendo assim, precisamos também modificar a tabela do `onCreate()` para que contenha os mesmos dados das nossas últimas atualizações do banco. Para isso modifique o conteúdo da `String sql` para:

```
"CREATE TABLE Alunos (id CHAR(36) PRIMARY KEY, " +
    "nome TEXT NOT NULL, " +
    "endereco TEXT, " +
    "telefone TEXT, " +
    "site TEXT, " +
    "nota REAL, " +
    "caminhoFoto TEXT);"
```

Testando a App

Agora que preparamos todo o ambiente da migration do SQLite vamos testar nossa App. Para verificar como está o id de cada aluno adicione um faça um *foreach* para cada aluno que é recuperado dentro do método `carregaLista()` da `ListaAlunosActivity` , e então, adicione um log.