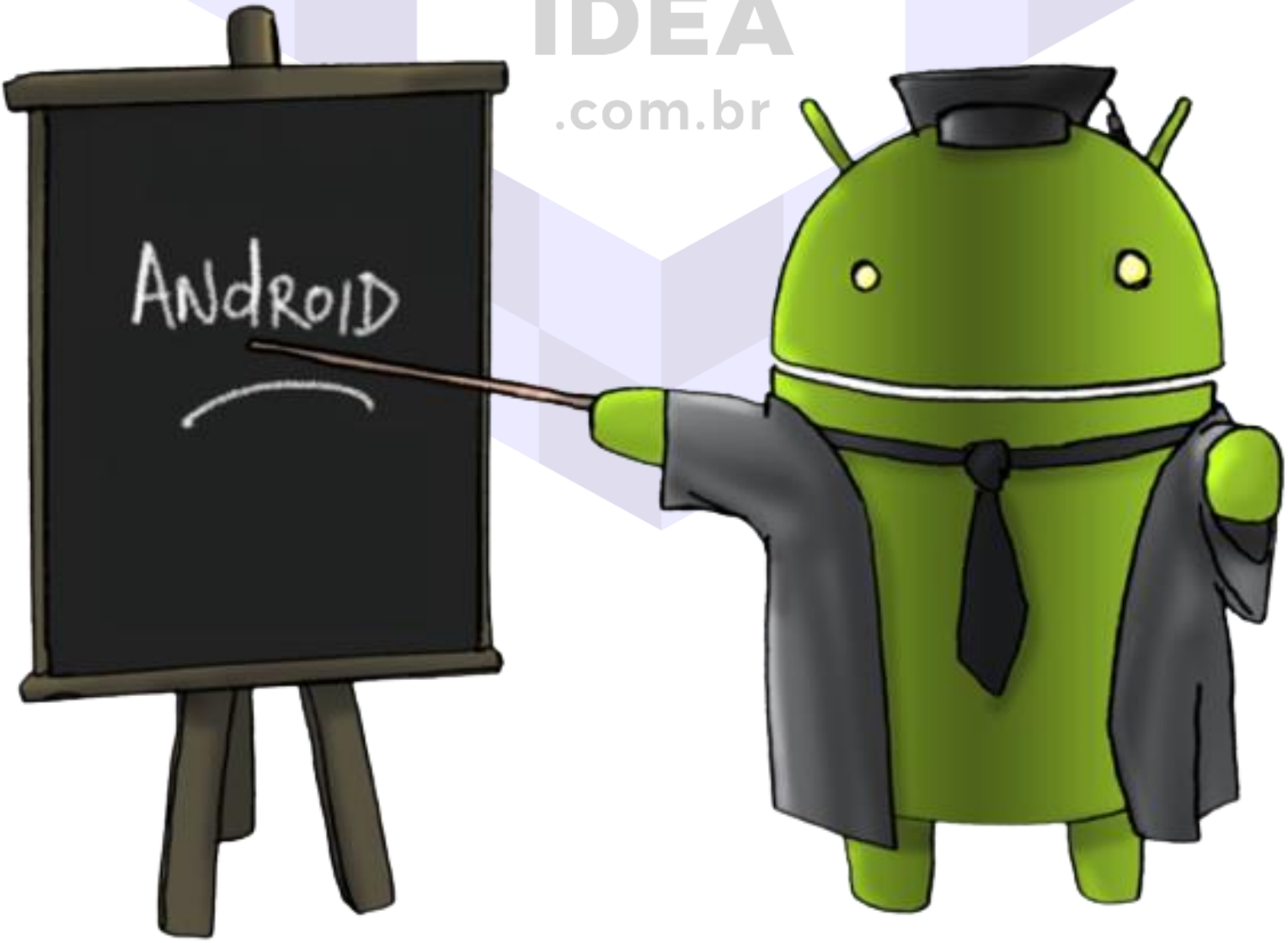
Programação Android



Acesso a Dados com SQLite no Android

A gestão de dados é uma parte essencial de muitos aplicativos móveis, permitindo que os usuários armazenem, recuperem e manipulem informações diretamente em seus dispositivos. No Android, uma das maneiras mais comuns de gerenciar dados localmente é usando SQLite, um sistema de gerenciamento de banco de dados relacional leve e eficiente.

O que é o SQLite?

SQLite é um sistema de gerenciamento de banco de dados relacional (RDBMS) embutido. Ao contrário de outros RDBMS tradicionais, o SQLite não requer um servidor separado para operar, mas integra-se diretamente ao aplicativo. Suas principais características incluem:

- Leveza: O SQLite tem uma pegada mínima, tornando-o ideal para dispositivos móveis com espaço e recursos limitados.

- Autônomo: Ele não depende de servidores externos ou configurações adicionais.

- Transacional: O SQLite suporta transações completas, garantindo a integridade dos dados.

Criando e Acessando um Banco de Dados

No Android, o acesso ao SQLite é facilitado pela classe `SQLiteOpenHelper`.
Para criar e acessar um banco de dados:

1. Extendendo `SQLiteOpenHelper`: Você precisa criar uma subclasse de `SQLiteOpenHelper`, onde definirá o nome do banco de dados, a versão e os métodos para criar e atualizar o esquema do banco de dados.

java Copy code

```java
public class DatabaseHelper extends SQLiteOpenHelper {
    public DatabaseHelper(Context context) {
        super(context, "nome_do_banco.db", null, 1);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        // Criar tabelas e estruturas iniciais aqui
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
        // Atualizar esquema do banco de dados, se necessário
    }
}
```

2. Acessando o Banco de Dados: Para acessar o banco de dados, instancie sua classe `DatabaseHelper`.

java Copy code

```java
DatabaseHelper dbHelper = new DatabaseHelper(context);
SQLiteDatabase db = dbHelper.getWritableDatabase();
```

CRUD (Criar, Ler, Atualizar e Deletar) com SQLite

O SQLite fornece operações CRUD básicas para gerenciar dados:

1. Criar (Insert):

```java
ContentValues values = new ContentValues();
values.put("nome_coluna", "valor");
db.insert("nome_tabela", null, values);
```

2. Ler (Query):

```java
cursor = db.query("nome_tabela", null, "coluna=?", new String[]{"valor"}, nul
```

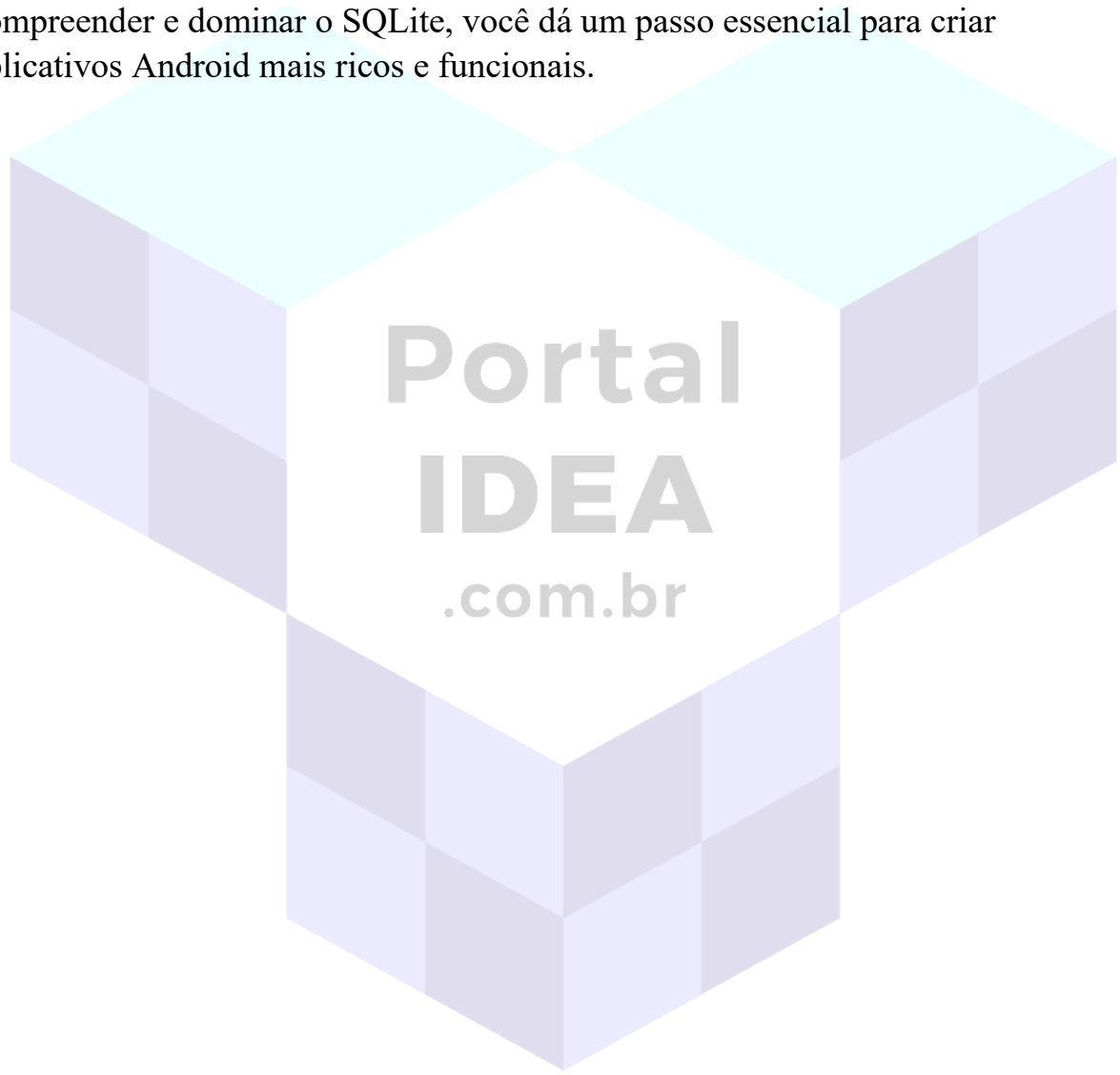

3. Atualizar (Update):

```java
ContentValues values = new ContentValues();
values.put("nome_coluna", "novo_valor");
db.update("nome_tabela", values, "coluna=?", new String[]{"valor"});
```

4. Deletar (Delete):

```java
db.delete("nome_tabela", "coluna=?", new String[]{"valor"});
```

SQLite oferece uma solução embutida, leve e robusta para gerenciar dados localmente em aplicativos Android. Com uma API intuitiva, os desenvolvedores podem implementar funcionalidades essenciais de banco de dados com facilidade e eficiência, permitindo que os usuários armazenem e recuperem informações diretamente em seus dispositivos sem depender de conexões externas. Ao compreender e dominar o SQLite, você dá um passo essencial para criar aplicativos Android mais ricos e funcionais.



Persistência de Dados
com SharedPreferences no Android

Em muitos aplicativos Android, surge a necessidade de armazenar pequenas quantidades de informações de forma persistente, mas sem a complexidade de um sistema completo de banco de dados. Para situações como estas, o Android oferece um mecanismo chamado `SharedPreferences`.

O que são SharedPreferences?

`SharedPreferences` é uma API do Android para armazenar e recuperar dados na forma de pares chave-valor. É especialmente útil para armazenar preferências do usuário, configurações do aplicativo ou mesmo dados simples, como pontuações altas de um jogo. Algumas características notáveis são:

- Simplicidade: `SharedPreferences` oferece uma abordagem simples e direta para armazenar dados primitivos, como booleanos, floats, ints, longs e strings.

- Privacidade: Os dados armazenados via `SharedPreferences` são privados ao aplicativo, garantindo que outras aplicações não tenham acesso a essas informações.

- Persistência: Mesmo após o aplicativo ser fechado ou o dispositivo reiniciado, os dados salvos permanecerão intactos, tornando-os ideais para armazenar configurações ou estados que precisam ser lembrados entre as sessões.

Armazenando e Recuperando Dados

Utilizar `SharedPreferences` é bastante direto:

1. Obtendo uma Instância de SharedPreferences: Você pode obter uma instância usando dois métodos, dependendo de sua necessidade:

- **`getSharedPreferences(String name, int mode)`:** Este método permite que você acesse um arquivo de preferências nomeado, o que é útil se você tiver diferentes conjuntos de preferências no seu aplicativo.
- **`getDefaultSharedPreferences(Context context)`:** Este método é frequentemente usado para acessar as preferências que são administradas pelo framework Android, como aquelas definidas usando uma `PreferenceActivity`.


```java
SharedPreferences prefs = getSharedPreferences("meuApp", MODE_PRIVATE);
```

2. Armazenando Dados:

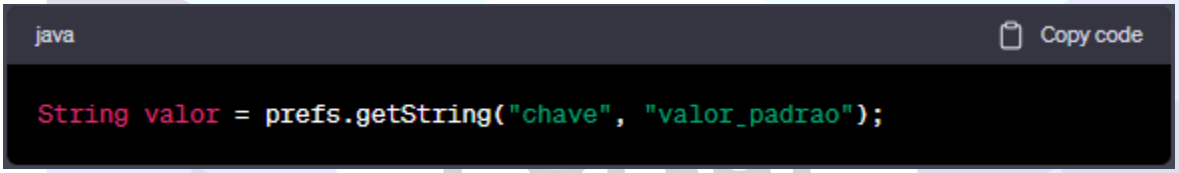
Para armazenar dados, você primeiro precisa obter um `SharedPreferences.Editor`, fazer suas modificações e então aplicar ou confirmar as mudanças.

```java
SharedPreferences.Editor editor = prefs.edit();
editor.putString("chave", "valor");
editor.apply();  // ou editor.commit();
```

A diferença entre `apply()` e `commit()` é que `apply()` faz a gravação de forma assíncrona e `commit()` faz de forma síncrona retornando um booleano indicando o sucesso ou falha.

3. Recuperando Dados:

Recuperar dados é simples, você só precisa especificar a chave e um valor padrão (que será retornado caso a chave não exista).



`SharedPreferences` fornece um meio fácil e eficiente para persistir pequenas quantidades de dados no Android. Seja para salvar as configurações do usuário, o progresso em um jogo ou qualquer outra informação leve que precisa ser lembrada entre sessões, é uma ferramenta indispensável para muitos desenvolvedores. Contudo, é importante notar que `SharedPreferences` não é destinado para o armazenamento de grandes quantidades de dados ou dados complexos; para tais necessidades, um banco de dados ou outra solução de armazenamento pode ser mais apropriado.

Boas Práticas no Desenvolvimento Android

Desenvolver aplicativos Android vai além de simplesmente escrever código que funcione. Para garantir uma boa experiência ao usuário e a manutenibilidade do projeto, é essencial adotar boas práticas durante todo o processo de desenvolvimento.

Conceito de Material Design

O Material Design é uma linguagem visual desenvolvida pelo Google para unificar a experiência do usuário em vários dispositivos e plataformas. Ele introduz diretrizes específicas para design de interface, animações e interações:

- Elementos UI Tangíveis: O Material Design trata os elementos da UI como objetos reais, ou seja, eles têm profundidade, sombras e se comportam de maneira previsível.

- Animações Significativas: Animações não são apenas estéticas, mas ajudam a guiar o usuário, mostrando a relação entre os elementos.

- Consistência: Seja através de cores, tipografia ou disposição dos elementos, o Material Design busca uma interface consistente e intuitiva.

Incorporar os princípios do Material Design no seu aplicativo não apenas melhora a estética, mas também a usabilidade e a experiência geral do usuário.

Performance e Otimização

Um aplicativo eficiente não consome mais recursos do que o necessário. Algumas dicas para otimização incluem:

- Evite Vazamentos de Memória: Utilize ferramentas como o LeakCanary para identificar e corrigir vazamentos no seu aplicativo.

- Otimização de Imagens: Imagens grandes podem consumir muito da memória. Utilize formatos adequados e ferramentas de compressão.

- Reduza a Complexidade de Layouts: Layouts excessivamente aninhados podem afetar o desempenho. O uso de `ConstraintLayout` pode ajudar a criar interfaces complexas com menos aninhamentos.

- Uso Eficiente de Threads: Utilize `AsyncTask`, `Handler`, ou bibliotecas como RxJava para gerenciar tarefas em segundo plano, evitando travamentos na UI.

Dicas para Manter o Código Organizado

Um código bem estruturado não é apenas mais fácil de entender, mas também mais fácil de manter e expandir:

- Arquitetura Limpa: Adote padrões arquitetônicos como MVP, MVVM ou MVI para separar responsabilidades no seu código.

- Reutilização e Modularização: Crie componentes reutilizáveis e evite duplicação de código.

- Comente e Documente: Comentários claros e uma boa documentação ajudam qualquer pessoa (inclusive você, no futuro) a entender o propósito e funcionamento do código.

- Controle de Versão: Utilize sistemas como Git para rastrear mudanças, colaborar e manter um histórico das alterações no código.

- Testes: Escreva testes unitários e de UI para garantir que seu código funcione como esperado e evitar regressões.

Adotar boas práticas é fundamental para o sucesso de um aplicativo Android. Desde a estética com o Material Design até a eficiência do código e a organização, cada aspecto contribui para criar um aplicativo de alta qualidade que atende às expectativas do usuário e é fácil de manter e expandir.