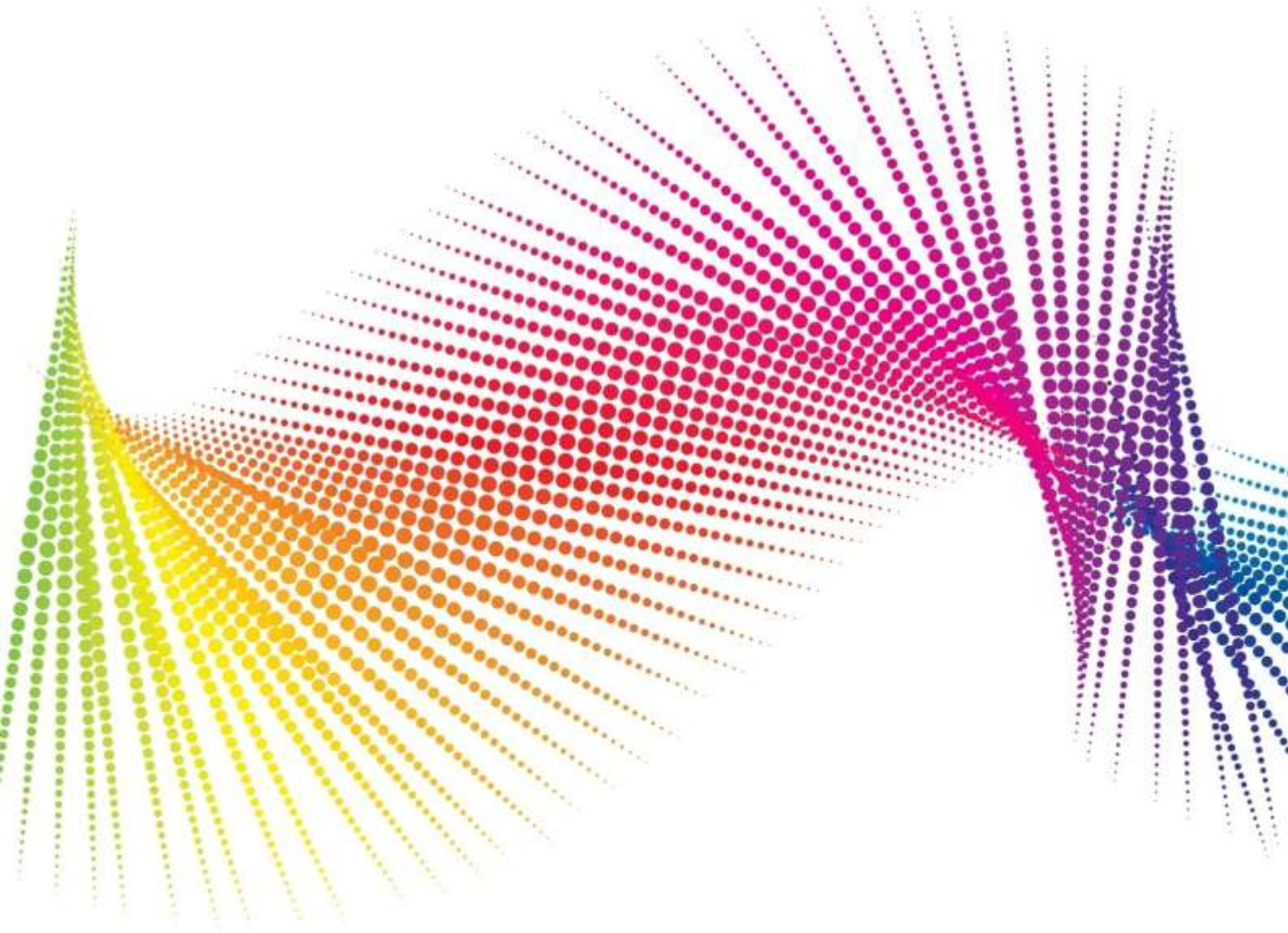
Computação Móvel

Aula 15



Aula 15: Armazenamento de dados

Objetivo: Introduzir a utilização de bancos de dados no android, utilizando o SQLite, tendo como foco principal a manutenção dos dados, ou seja, inclusão, alteração, exclusão e pesquisa em banco de dados.

Utilizando bancos de dados SQL no android

O android (assim como o iPhone) usa um programa independente incorporado a sua estrutura, o SQLite, que pode ser usado para: criar uma base de dados, definir tabelas SQL, criar índices, realizar pesquisas, criar visualizações, definir triggers, inserir registros, deletar registros, alterar registros e administrar uma base de dados SQLite.

Algumas das características do SQLite são:

- Autocontido.
- Sem servidor.
- Zero configuração.

O SQLite possui o mesmo motor de banco de dados transacional SQL. Segundo seu site, esse é o motor de banco de dados SQL mais amplamente implantado no mundo, sendo que seu código fonte é de domínio público.

Usando SQLite

Exemplo 1

```
package com.exemplo1.sqldatabases;
public class SQLDemo1 extends Activity {
SQLiteDatabase db;
@Override
public void onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
// this provides the 'real' path name to theexternal SD card
String SDcardPath = Environment.getExternalStorageDirectory().getPath();
// For internal memory: "data/data/cis470.matos.sqldatabases/myfriendsDB"
TextView txtMsg = (TextView)findViewById(R.id.txtMsg);
String myDbPath = SDcardPath + "/" + "myfriends";
txtMsg.setText("DB Path: " + myDbPath);
try {
db = SQLiteDatabase.openDatabase(myDbPath,
null,
SQLiteDatabase.CREATE_IF_NECESSARY);
// here you do something with your database ...
db.close();
txtMsg.append("\nAll done!");
}
catch (SQLiteException e) {
txtMsg.append( e.getMessage() );
}
}//onCreate
}//class
```

Antes de começarmos a utilizar o SQLite, precisamos saber um pouco mais sobre seu funcionamento, conforme segue.

- Implementa a maior parte do padrão SQL-92 para SQL.
- Possui uma estrutura voltada para utilização de triggers, permitindo consultas mais complexas.
- Não implementa restrições de integridade referencial por meio do modelo de restrição de chave estrangeira.
- Utiliza um modelo de digitação livre dados.

- Não atribui um tipo a uma coluna inteira, os tipos são atribuídos aos valores individuais (isto é semelhante ao tipo de variante em Visual Basic).
- Não possui nenhuma verificação de tipo de dados, logo, é possível a inserção de uma cadeia de caracteres em coluna numérica e assim por diante.

Como criar um banco de dados SQLite?

Existem duas maneiras de criar um banco de dados com o SQLite como esta descrito a seguir.

Método 1

public static SQLiteDatabase.openDatabase (String path, SQLiteDatabase.CursorFactory factory, int flags)

Abre a base de dados de acordo com os flags OPEN_READWRITE, OPEN_READONLY, CREATE_IF_NECESSARY. Definindo o local do banco de dados como sendo a localidade atual do sistema.

Parâmetros

- Path – caminho utilizado para abrir e/ ou criar o banco de dados
- CursorFactory – classe opcional que é chamada para instanciar um cursor quando consulta é chamada, ou os flags padrões para controlar o modo de acesso de banco de dados são nulos.

Este comando retorna uma banco de dados recém criado ou lança SQLiteException se o banco de dados não pode ser aberto.

Exemplo 1: criar um banco de dados SQLite

Uma forma alternativa de abertura ou criação de um banco de dados SQLite em seu espaço local de dados do android é dado a seguir.

SQLiteDatabase db = this.openOrCreateDatabase ("myfriendsDB", MODE_PRIVATE, null)

Se o aplicativo é criado em um espaço chamado "cis493.sql1", por exemplo, o nome completo do arquivo de banco de dados recém criado será:

/data/data/cis493.sql1/databases/myfriendsDB

Em que "MyFriendsDB" é o caminho do arquivo abreviado. O prefixo é atribuído pelo android, como: /data/data/<appnamespace>/databases/myFriendsDB.

MODE poderia ser: MODE_PRIVATE, MODE_WORLD_READABLE e MODE_WORLD_WRITEABLE.

Execução de comandos SQL no banco de dados

Uma vez criado, o banco de dados está pronto para realizar operações normais, tais como: criar, alterar recursos (tabelas, índices, triggers, views, consultas, etc) ou administrar recursos de banco de dados (contentores, usuários, ...). As tarefas de manutenção inclusão, alteração, exclusão e consultas de dados representam as operações mais comuns em banco de dados.

Criando/ preenchendo uma tabela

Vamos usar o método execSQL(...) para manipular as ações SQL que desejamos executar.

O exemplo a seguir cria uma nova tabela chamada tblAmigo.

A tabela possui três campos: um identificador numérico exclusivo chamado RecID, e dois campos string, representando o nome e telefone do nosso amigo. Em seguida, incluiremos três registros desta, utilizando as instruções a seguir.

db.execSQL("create table tblAMIGO ("RecID integer PRIMARY KEY autoincrement, nome text, telefone text); ");

db.execSQL("insert into tblAMIGO(nome, telefone) values ('AAA', '555');");
db.execSQL("insert into tblAMIGO(nome, telefone) values ('BBB', '777');");
db.execSQL("insert into tblAMIGO(nome, telefone) values ('CCC', '999');");

Comentários

RecID campo é definido como chave primária da tabela. O "autoincrement" é a característica a qual garante que cada novo registro será dado um número de série único (0,1,2, ...).

Os tipos de dados do banco de dados são muito simples, por exemplo, vamos utilizar: text, varchar, integer, float, number, date, time, date and time, blob, boolean, e assim por diante.

- Deve-se fazer a chamada para execSQL dentro de um bloco try-catch-finally. Esteja ciente de situações potenciais retornadas pelo método SQLiteException.

Usando RawQuery

Considere o seguinte fragmento de código:

Cursor c1 = db.RawQuery ("select count(*) as total from tblAMIGO", null);

O comando RawQuery anterior contém uma declaração de seleção, que conta com as linhas da tabela tblAMIGO, o resultado desta contagem é realizada em uma tabela com apenas uma linha e uma coluna. A coluna é chamada "total".

Usando RawQuery parametrizado

Usando argumentos: suponha que queremos contar quantos amigos estão lá, cujo nome é 'BBB' e RecId > 1. Poderíamos usar a seguinte construção:

```
String mySQL = "select count(*) as Total from tblAmigo where RecId > ?  and name = ? ";
String[] args = {"1", "BBB"};
Cursor c1 = db.rawQuery(mySQL, args);
```

Usando argumentos

Suponha que queremos contar quantos amigos estão na base de dados, cujo nome é 'BBB' e RecId > 1. Poderíamos concatenar pedaços de strings. Cuidado especial ao redor (único) de strings entre aspas.

```
String[] args = {"1", "BBB"};
String mySQL = " select count(*) as Total from tblAmigo where RecID > " + args[0] + " and name = '" + args[1] + "'";
Cursor c1 = db.rawQuery(mySQL, null);
```

Consultas simples

As consultas simples utilizam um esquema modelo que tem por objeto "ajudar" os desenvolvedores em seu processo de consulta a um banco de dados. O modelo expõe uma versão paramétrica da instrução select do SQL. Consultas simples só podem recuperar dados a partir de uma única tabela.

O comando utilizado para realização de consultas simples no android é:

query (String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)

Em que os campos representam:

- table – nome da tabela,
- columns – colunas a serem recuperadas,
- selection – condição de pesquisa,
- selectionArgs – parâmetros para a cláusula WHERE,

- groupBy – campos de agrupamento,
- having – cláusula having, e
- orderBy – campos de ordenação.

Exemplo 2

Consultar o EmployeeTable, encontrar o salário médio dos trabalhadores do sexo feminino supervisionadas por 123456789. Apuração de resultados por Dno. Primeira lista a maior média, e assim por diante, não incluem depts. ter menos de dois funcionários.

```
String[] columns =
{"Dno", "Avg(Salary) as AVG"};
String[] conditionArgs =
{"F", "123456789"};
Cursor c = db.query (
"EmployeeTable",                    ← table name
columns,                            ← columns
"sex = ? And superSsn = ? " ,       ← condition
conditionArgs,                      ← condition args
"Dno",                              ← group by
"Count(*) > 2",                     ← having
"AVG Desc "                         ← order by
);
```

Cursores

Os cursores android são usados para ganhar o acesso (sequencial ou aleatório) a tabelas produzidas por instruções de seleção SQL. Eles fornecem principalmente dados armazenados em uma tabela, em tempo de execução e incluem vários tipos de operadores, entre eles:

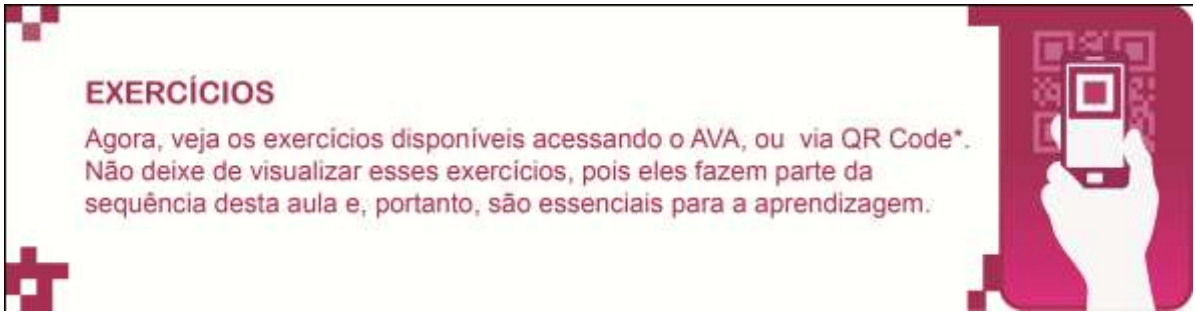
- Operadores para verificação de posicionamento (isFirst(), isLast(), isBeforeFirst(), isAfterLast())
- Navegação de registros (moveToFirst(), moveToLast(), MoveToNext(), moveToPrevious())
- Recuperação de registros (getInt, getString, getFloat, getBlob, getDate, etc)

Exercício

Para termos certeza de que entendemos a aula, vamos realizar uma atividade sobre armazenamento de dados no sistema android.

Próxima aula

Uma vez que já temos o conhecimento sobre o armazenamento de dados utilizados na arquitetura android, podemos dar o próximo passo em nosso aprendizado. Na próxima aula – **Aula 16** – aprenderemos a resgatar dados dos provedores de dados no sistema android.



***** O QR Code é um código de barras que armazena links às páginas da web. Utilize o leitor de QR Code de sua preferência para acessar esses links de um celular, tablet ou outro dispositivo com o plugin Flash instalado.

REFERÊNCIAS

LECHETA, Ricard R. *Android*: aprenda a criar aplicações para dispositivos móveis com o android SDK. 2. ed. São Paulo: Novatec, 2010.

ROGERS, Rick; et al. *Desenvolvimento de aplicações Android*. São Paulo: Novatec, 2009.