

# ANDROID

Persistência



## Persistência

- Shared Preferences
  - Armazena pares CHAVE/VALOR
  - Usado para pequenas informações
- Internal Storage
  - Armazenamento de arquivos na área interna do dispositivo
  - Visível somente para a aplicação
  - `java.io.InputStream` e `java.io.OutputStream`
- External Storage
  - Armazenamento de arquivos no cartão SD, interno ou externo no dispositivo
  - Visíveis para todos
  - `java.io.InputStream` e `java.io.OutputStream`
- SQLite
  - Banco de Dados interno





## SQLite

- Banco de Dados simples
- Opensource, baseado em SQL como linguagem de manipulação
- Vem nativamente com o Android
- Cada aplicação pode criar quantos bancos precisar
  - Os BDs são visíveis somente para a aplicação que os criou
- Link: <http://www.sqlite.org>
- Link: <https://developer.android.com/training/basics/data-storage/databases.html?hl=pt-br>
- Link: <https://www.sqlite.org/datatype3.html>
- Link: <http://www.sqlitetutorial.net/sqlite-date/>



# SQLite – Acesso ao Banco

- Classes que controlam a criação do BD herdam de **SQLiteOpenHelper**
  - Construtor deve informar: contexto, nome do banco de dados, versão
  - Basicamente controla a criação do BD e sua atualização
- Métodos:
  - **getReadableDatabase()** : cria ou abre o BD, retornando um objeto do tipo **SQLiteDatabase**, que é a conexão, retorna um BD somente de leitura em caso de problema (ex, sem espaço)
  - **getWritableDatabase()** : cria ou abre o BD, retornando um objeto do tipo **SQLiteDatabase**, que é a conexão, usado para leitura/escrita.
    - Se o BD não existe, chama **onCreate()**
    - Se o número de versão no **onCreate()** difere do BD, chama **onUpgrade()**



# SQLite – Acesso ao Banco

```
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class Banco extends SQLiteOpenHelper {
    public Banco(Context context) {
        super(context, "banco.db" ,null, 1);
    }
    public void onCreate(SQLiteDatabase db) {
        String sql = "CREATE TABLE tb_boi ( "
                    + "boi_id integer primary key autoincrement, "
                    + "boi_nm text, "
                    + "boi_raca text )";
        db.execSQL(sql);
    }
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS tb_boi");
        onCreate(db);
    }
}
```



# SQLite – Manipulação de Dados

- Classes de acesso ao BD:
  - Criam uma classe controladora do banco (Banco)
  - Obtém uma conexão invocando `getWritableDatabase()`
  - Usam métodos de SQLiteDatabase para manipulação dos dados
- Métodos de SQLiteDatabase
  - `query()` : busca registros do banco; retorna um Cursor ou exceção se erro
  - `insert()` : insere registros no banco; retorna o ID do elemento inserido ou -1 se erro
  - `delete()` : remove registros do banco; retorna o número de linhas afetadas se uma cláusula where for passada, ou 0 caso contrário.
  - `update()` : atualiza registros do banco; retorna o número de linhas afetadas



# SQLite – Manipulação de Dados

- Para passar dados para as queries, como em um `insert`, usa-se um objeto de `ContentValues`
  - É um mapa do tipo <CHAVE>/<OBJETO>
  - <CHAVE> é o nome do campo
  - <OBJETO> é o dado passado
- Exemplo

```
ContentValues values = new ContentValues();
values.put("boi_nm", "Mimosa");
values.put("boi_raca", "Holandesa");

long res = db.insert("tb_boi", null, values);
```



## SQLite – Manipulação de Dados

- Pode-se usar um *design pattern* para encapsular o acesso aos dados
  - Exemplo: DAO
  - Classe DAO que executa as chamadas ao SQLite
  - Outros pontos do projeto só acessam o DAO
  - Necessária a criação de uma classe de representação de dados (bean)
  - Mantém uma instância de uma filha de `SQLiteOpenHelper` (no exemplo aqui, classe `Banco`)
  - Mantém uma instância de `SQLiteDatabase`, usada para manipular o BD



## SQLite - Bean

```
public class Boi {  
    private int id;  
    private String nome;  
    private String raca;  
  
    // setters/getters  
}
```



## SQLite - DAO

```
public class BoiDAO {
    private SQLiteDatabase db = null;
    private Banco banco = null;
    public BoiDAO(Context context) {
        banco = new Banco(context);
    }
    public void open() {
        if (db == null)
            db = banco.getWritableDatabase();
    }
    public void close() {
        if (db != null) {
            db.close();
            db = null;
        }
    }
    // métodos de CRUD aqui
}
```



## SQLite – DAO - insert()

```
public class BoiDAO {
    // ...
    public long insert(Boi boi) {

        open();
        ContentValues values = new ContentValues();
        values.put("boi_nm", boi.getNome());
        values.put("boi_raca", boi.getRaca());
        long res = db.insert("tb_boi", null, values);
        close();
        return res; // -1 = erro; senão ID inserido
    }
    // ...
}
```



## SQLite – DAO - update()

```
public class BoiDAO {
    // ...
    public int update(Boi boi) {

        open();
        String where = "boi_id=" + boi.getId();
        ContentValues values = new ContentValues();
        values.put("boi_nm", boi.getNome());
        values.put("boi_raca", boi.getRaca());
        int res = db.update("tb_boi", values, where, null);
        close();
        return res; // número de linhas afetadas
    }
    // ...
}
```



## SQLite – DAO - delete()

```
public class BoiDAO {
    // ...
    public int delete(Boi boi) {

        open();
        String where = "boi_id=" + boi.getId();
        int res = db.delete("tb_boi", where, null);
        close();
        return res; // número de linhas afetadas
    }
    // ...
}
```



## SQLite – DAO - findById()

```
public class BoiDAO {
    // ...
    public Boi findById(int id) {
        String[] colunas = { "boi_id", "boi_nm", "boi_raca" };

        open();
        String where = "boi_id=" + id;
        Cursor res = db.query("tb_boi", colunas, where, null, null, null, null);
        Boi boi = null;
        if (res.moveToFirst()) {
            boi = new Boi();
            boi.setId( res.getInt(0) );
            boi.setNome( res.getString(1) );
            boi.setRaca( res.getString(2) );
        }
        close();
        return boi; // objeto ou null
    }
    // ...
}
```



## SQLite – DAO - findAll()

```
public class BoiDAO {
    // ...
    public List<Boi> findAll() {
        String[] colunas = { "boi_id", "boi_nm", "boi_raca" };
        List<Boi> lista = new ArrayList<Boi>();

        open();
        Cursor res = db.query("tb_boi", colunas, null, null, null, null, null);
        if (res.moveToFirst()) {
            do {
                Boi boi = new Boi();
                boi.setId( res.getInt(0) );
                boi.setNome( res.getString(1) );
                boi.setRaca( res.getString(2) );
                lista.add(boi);
            } while (res.moveToNext());
        }
        close();
        return lista;
    }
    // ...
}
```



# Uso do DAO

```
public void testarBD(View view) {
    BoiDAO dao = new BoiDAO(getApplicationContext());

    Boi b = new Boi();
    b.setId(10);
    b.setNome("Mimoso");
    b.setRaca("Nelore");
    dao.insert(b);

    b = new Boi();
    b.setId(20);
    b.setNome("Mimosa");
    b.setRaca("Holandesa");
    dao.insert(b);

    List<Boi> lista = dao.findAll();

    for (Boi x : lista) {
        Toast.makeText(getApplicationContext(),
                "Boi: " + x.getNome(), Toast.LENGTH_SHORT)
                .show();
    }
}
```

