ANDROID - Banco de Dados

SQLLite

É um banco de dados opensource que suporta as características de um banco de dados relacional e comandos SQL, transações e funções.
Os tipos de dados aceitos são: TEXT, INTEGER e REAL. Para maiores detalhes http://www.sqlite.org
O SQLite vem embarcado no Android, logo você não precisa de nenhuma instalação para fazer funcionar.
Os dados são normalmente armazenados na pasta DATA/data/APP_NAME/databases/FILENAME

SQLiteOpenHelper

Para criar ou atualizar uma base de dados na sua aplicação Android, você deve criar uma subclasse de SQLiteOpenHelper. No construtor, você deve chamar o método super especificando o nome da base de dados e a versão corrente.
Nesta classe você deverá sobrescrever os métodos onCreate (chamado quando a base de dados é acessada mas não está criada) e onUpgrade (chamada quando a versão da base de dados é alterada, você pode fazer alterações na tabela ou simplesmente apagar a tabela e recriá-la. CUIDADO !!!!)
A classe SQLiteOpenHelper nos proporciona os métodos getReadableDatabase() e getWriteableDatabase(), que dão acesso de leitura e escrita ao objeto.
As tabelas da base de dados utilizam um identificador (_id) que representa a chave primária da tabela.

SQLiteDatabase

SQLiteDatabase é a classe case para se trabalhar com base de dados SQLite no Android e provê os métodos para abrir, consultar, modificar e fechar a base de dados.
Mais especificamente ele provê os métodos: insert(), update() e delete().
Além disso fornece o método execSQL() que possibilita o envio de comandos SQL diretamente para o banco.
O objeto ContentValues permite que se defina chave/valor. A chave representa o nome da coluna da tabela e o valor é o que será colocado neste campo. ContentValues podem ser usados para inserção e modificação dos dados.
As consultas podem ser criadas pelos métodos rawQuery() ou query() ou via classe SQLiteQueryBuilder.

rawQuery() – Aceita comando SQL select diretamente.

```
Cursor cursor = getReadableDatabase().
  rawQuery("select * from todo where _id = ?", new String[] { id });
```

query() - Fornece um modo estrutural para especificar o SQL

```
return database.query(DATABASE_TABLE,
  new String[] { KEY_ROWID, KEY_CATEGORY, KEY_SUMMARY, KEY_DESCRIPTION },
  null, null, null, null, null);
```

Parameter	Comment
String dbName	The table name to compile the query against.
String[] columnNames	A list of which table columns to return. Passing "null" will return all columns.
String whereClause	Where-clause, i.e. filter for the selection of data, null will select all data.
String[] selectionArgs	You may include ?s in the "whereClause"". These placeholders will get replaced by the values from the selectionArgs array.
String[] groupBy	A filter declaring how to group rows, null will cause the rows to not be grouped.
String[] having	Filter for the groups, null means no filter.
String[] orderBy	Table columns which will be used to order the data, null means no ordering.

```java
public class MySQLiteHelper extends SQLiteOpenHelper {

    private static final int VERSAO_BD = 1;
    private static final String NOME_BD = "faculdade";

    // Tabelas
    private static final String TABELA_ALUNO = "alunos";
     // Books Table Columns names
    private static final String CAMPO_RA = "ra";
    private static final String CAMPO_NOME = "nome";

    private static final String[] COLUNAS = {CAMPO_RA,CAMPO_NOME};


    public MySQLiteHelper(Context context) {
        super(context, NOME_BD, null, VERSAO_BD);
    }

    @Override
    public void onCreate(SQLiteDatabase bd) {
        // Criando as Tabelas
        String CRIA_TABELA_ALUNOS = "CREATE TABLE alunos (ra integer primary key, nome text)";
        bd.execSQL(CRIA_TABELA_ALUNOS);

    }

    @Override
    public void onUpgrade(SQLiteDatabase bd, int versaoAnterior, int novaVersao) {
        // Atualizando as Tabelas
        bd.execSQL("DROP TABLE alunos");
        this.onCreate(bd);
    }
```

```java
public void insereAluno(Aluno aluno) {
    SQLiteDatabase bd = this.getWritableDatabase();
    //ContentValues - Classe usada para armazenar valores
    ContentValues cv = new ContentValues();
    cv.put(CAMPO_RA, aluno.getRa());
    cv.put(CAMPO_NOME, aluno.getNome());

    bd.insert(TABELA_ALUNO, null, cv);

    bd.close();
}
```

```java
public void alteraAluno(Aluno aluno) {
    SQLiteDatabase bd = this.getWritableDatabase();
    ContentValues cv = new ContentValues();
    cv.put(CAMPO_RA, aluno.getRa());
    cv.put(CAMPO_NOME, aluno.getNome());

    int i = bd.update(TABELA_ALUNO, cv, "ra=?", new String[] {String.valueOf(CAMPO_RA)});

    bd.close();
}
```

```
public void apagaAluno(Aluno aluno) {
    SQLiteDatabase bd = this.getWritableDatabase();

    bd.delete(TABELA_ALUNO, "ra=?", new String[] {String.valueOf(aluno.getRa()) });

    bd.close();
}



public Aluno getAluno(int ra) {
    SQLiteDatabase bd = this.getReadableDatabase();
    Cursor c = bd.query(TABELA_ALUNO,COLUNAS," ra=?",new String[] { String.valueOf(ra) }
                        ,null,null,null,null);

    if (c != null)
        c.moveToFirst();
    Aluno aluno = new Aluno();
    aluno.setRa(c.getInt(0));
    aluno.setNome(c.getString(1));

    return aluno;
}



public List<Aluno> getTodosAlunos() {
    List<Aluno> alunos = new LinkedList<Aluno>();
    String consulta = "SELECT ra,nome FROM "+TABELA_ALUNO;

    SQLiteDatabase bd = this.getReadableDatabase();

    Cursor cursor = bd.rawQuery(consulta, null);

    Aluno aluno = null;

    if (cursor.moveToFirst()) {
        do {
            aluno = new Aluno();
            aluno.setRa(cursor.getInt(0));
            aluno.setNome(cursor.getString(1));

            alunos.add(aluno);

        } while(cursor.moveToNext());
    }

    return alunos;

}
```