DigitalHouse >

Bases de dados

Bases de dados

Um banco de dados é uma coleção de dados relacionados. Com dados, queremos dizer fatos conhecidos que podem ser registrados.

DigitalHouse >

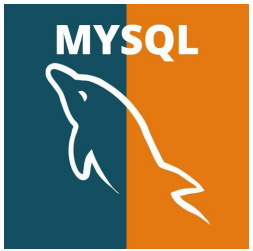
O que é uma **base de dados**?

- Coleção de **dados organizados e relacionados.**
- Representa **aspectos** da **realidade**.

DigitalHouse >

Motores de base de dados

- Coleção de **programas** que permitem **criar e manter uma Base de Dados.**
- São responsáveis por armazenar e organizar dados para acesso rápido.

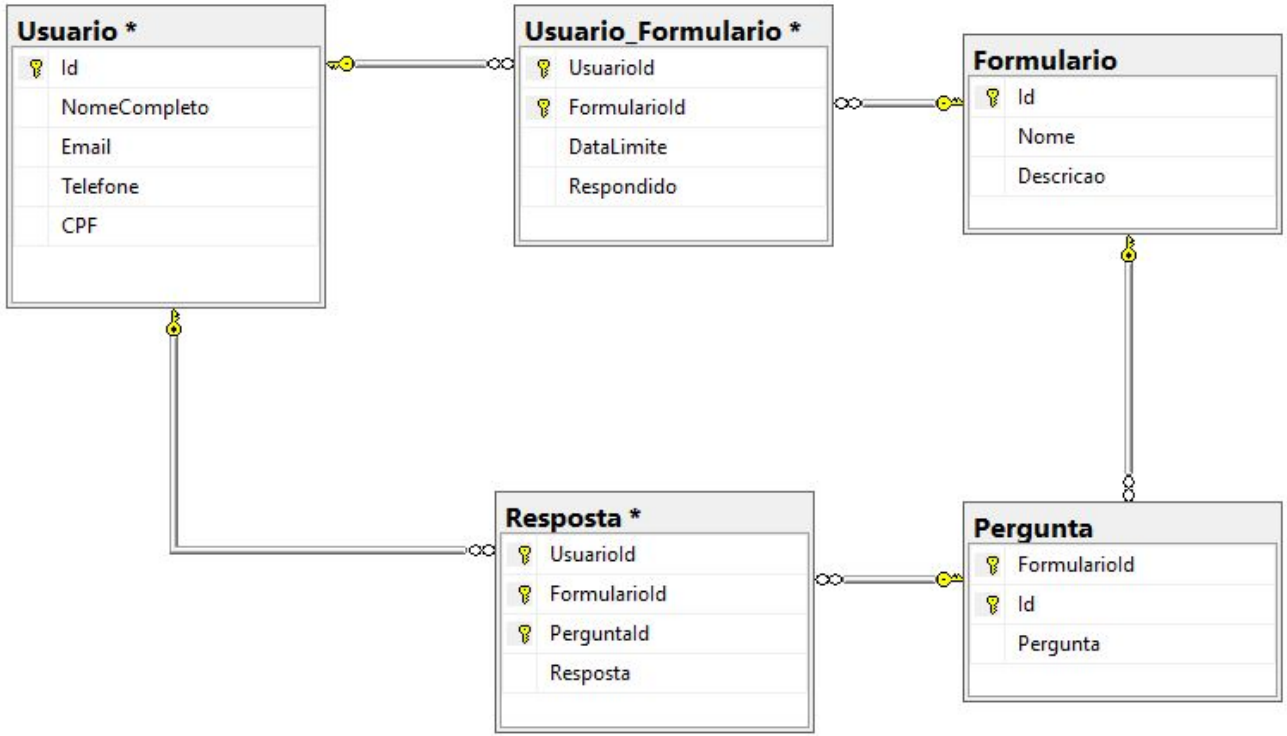


DigitalHouse >

Metodologia

- **Identificar entidades:**
  - Definir objetos do mundo a representar.
- **Identificar atributos:**
  - Definir as “propriedades” de cada entidade.

DigitalHouse >



Agora como o Android utiliza tudo isso?

Android + SQLite

Utilizando armazenamento interno do android para funcionalidades offline.

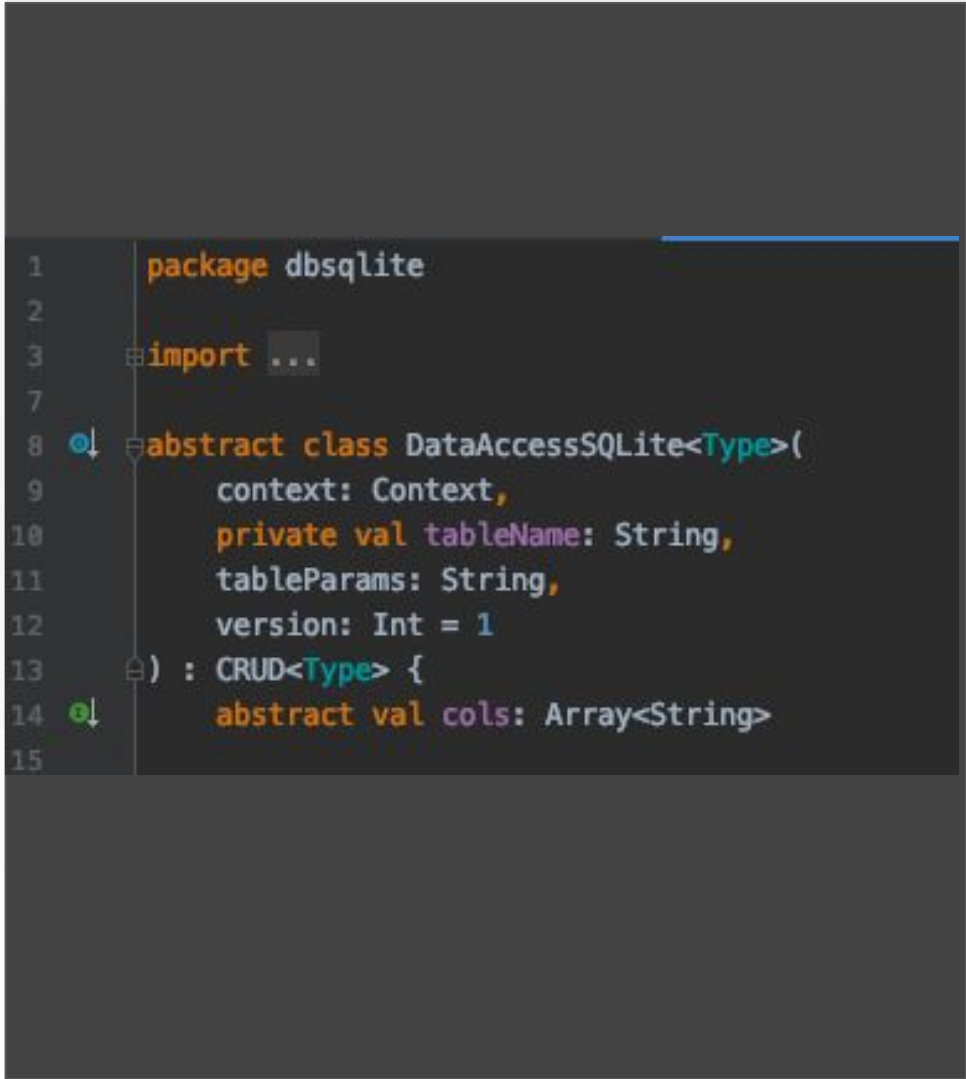
Entidade, Banco de Dados e Queries no Android

DigitalHouse > Coding School

Criando nosso CRUD..

```
package dbsqlite

interface CRUD<Type> {

    fun create(objekt: Type)
    fun read(value: Any? = null, col: String = "id"): List<Type>
    fun update(id: Int, objekt: Type)
    fun destroy(id: Int)
}
```

Criando nossa classe abstrata...



```
1   package dbsqlite
2
3   import ...
7
8   abstract class DataAccessSQLite<Type>(
9       context: Context,
10      private val tableName: String,
11      tableParams: String,
12      version: Int = 1
13  ) : CRUD<Type> {
14      abstract val cols: Array<String>
15
```

Criando o banco de dados

```
private val banco: SQLiteDatabase by lazy {
    FactoryDataBase(context, tableName, tableParams, version).writableDatabase
}
```

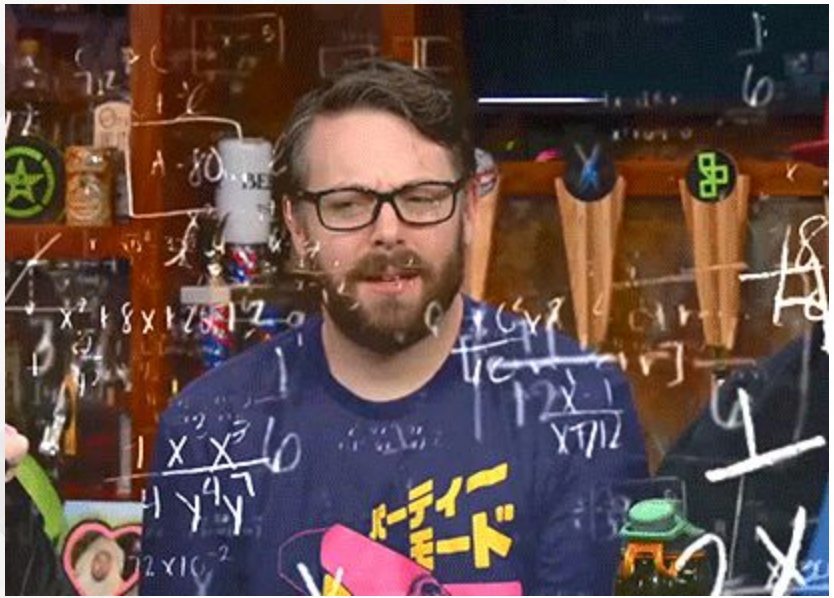
Criando o insert, delete e update...

```
fun insert(values: ContentValues.() -> Unit) {
    banco.insert(tableName,  nullColumnHack: null, ContentValues().apply(values))
}

fun update(whereClause: String, id: Int, values: ContentValues.() -> Unit) {
    banco.update(tableName, ContentValues().apply(values), whereClause, arrayOf(id.toString()))
}

fun delete(whereClause: String, id: Int) {
    banco.delete(tableName, whereClause, arrayOf(id.toString()))
}
```

Criando uma query...

```
fun query(
    selection: String? = null,
    args: Array<String>? = null,
    lambda: Cursor.() -> Any
) {
    val cursor : Cursor! = banco.query(
        tableName,
        cols,
        selection,
        args,
         groupBy: null,
         having: null,
         orderBy: null
    )
    cursor.lambda()
    cursor.close()
}
```

MEU DEUS!



Utilizando SQLite dessa forma sem nenhuma ajuda, deixa a nossa experiência muito trabalhosa e ocupa muito tempo de desenvolvimento.

Android Room

Para isso foi criado a biblioteca chamada Android Room (dentro do conjunto do JetPack), essa biblioteca permite trabalharmos com anotações e métodos que facilita muito a interação com o SQLite.

Android Room

Devs Android

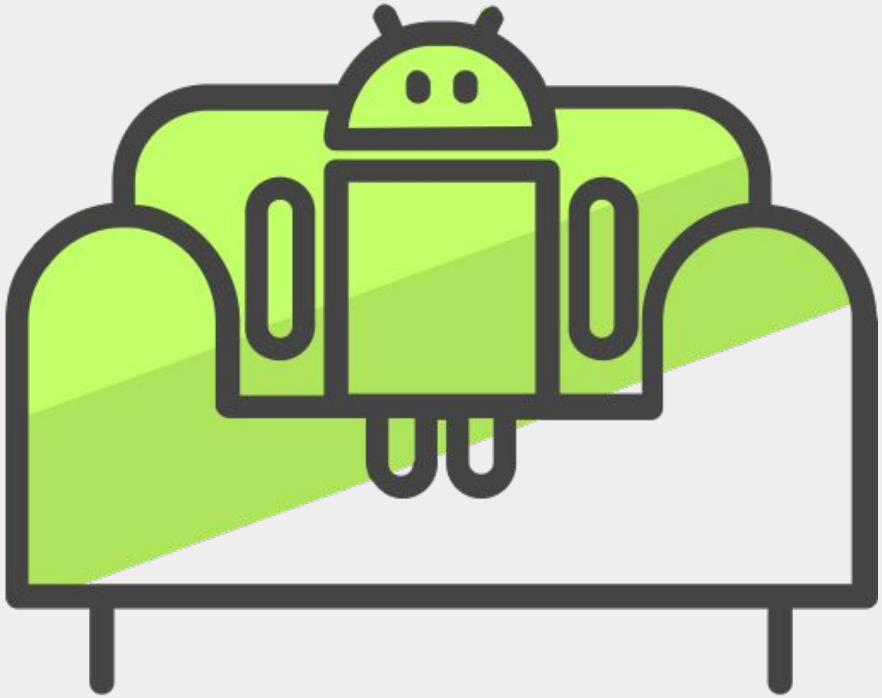
“ANDROID ROOM”

DigitalHouse >
Coding School

BD + Entidade + Android = Android Room

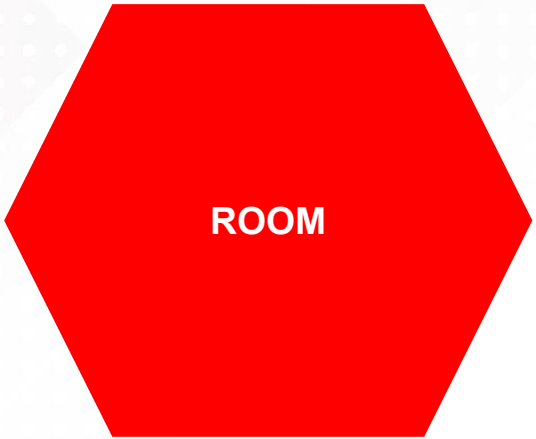
Utilizamos o Android Room como nosso ORM.

ORM = Object-Relational Mapping



ORM - Android ROOM

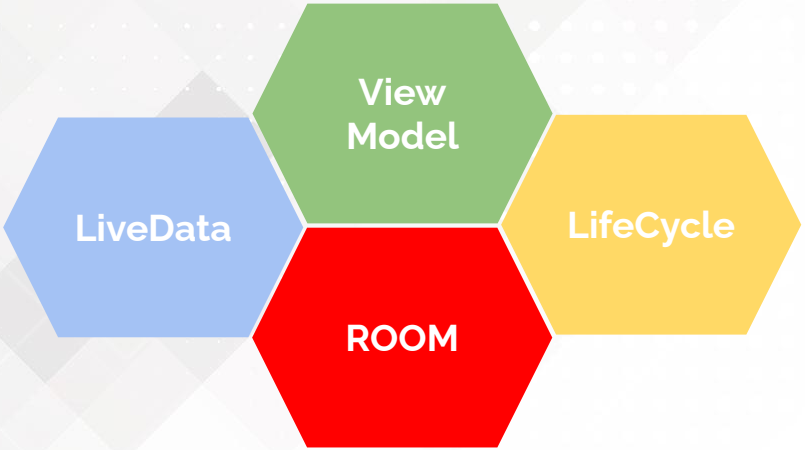
- O ORM permite gerenciar instâncias de objetos
- Armazenamento **baseado no Modelo Relacional.**



ORM - Android ROOM

Permite gerenciar um componente durante o ciclo de vida

Observer, notificações



Permite criar coisas que estejam conscientes de um ciclo de vida

Persistência ORM

DigitalHouse > Coding School

Por hoje é isso...

DigitalHouse > Coding School