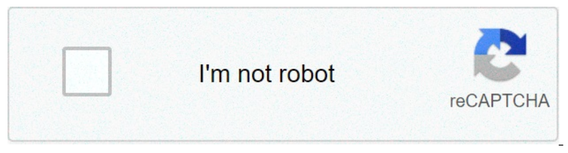

Continue

Android db example

Room db android example. Db.query android example. Room db android example github. Db.rawquery android example. Room db android example kotlin. Realm db android example. Paper db android example. Android room db insert example.

Saving data in a database is ideal for repetition or structured data, such as contact information. This page presupposes that you have familiarity with SQL databases in general and help you start with SQLite databases on Android. The APIs you will need to use a database on Android are available in the Android.database.sqlite package. Attention: Although these APIs are powerful, they are quite low-level and require great quantity of time and effort for use: there is no verification when compiling the prime SQL queries. As the graphics data changes, you need to update the SQL queries hit manually. This process can take a long time and subject to errors. You need a lot of standard code for conversion between SQL queries and data objects. For these reasons, it is strongly recommended to use the Library Room Persistence as an abstraction level for data access in the SQLite databases of your app. Defining a schema and the contract One of the main principles of SQL database is the scheme: a formal statement of how the database is organized. The scheme is reflected in the SQL statements that you use to create the database. You can find useful to create a companion class, known as a contract class, which explicitly specifies the layout of your scheme in a systematic way and self-documentation. A contract class is a container for the constants that define the names for Uris, tables and columns. The contract class allows you to use the same constants on all other classes in the same package. This allows you to change a column name in a place and make it look for during your code. A good way to organize a contract class is to put global definitions for all your database in the class root level. Then create an internal class for each table. Each interior class enumerate the columns of the corresponding table. Note: By implementing the BaseColumns interface, the inner class can inherit a primary key field called _ID that some Android classes like the cursoratadadapter expect to have. It is not required, but this can help your database work harmoniously with the Android picture. For example, the following contract defines the name of the table and column names for a single table that represents a RSS feed: FeedReadRercontract object {// The contents of the table are grouped together in an anonymous object. Object food: basolumns {cont val table_name = "entrance" const val column_name_title = "title" const val column_name_subtitle = "subtitle"}} public final class fedreadercontract {// to prevent anyone of instantization accidentally the class of the contract, / / Make the private manufacturer. PRIVATE FEEDREADCONTRACT () {} / * Internal class that defines the contents of the table * / feedny of class Statistics public basic implements of the basocolumns {String final static public table_name = "entry"; Public Static Final String Column_Name_Title = "Title"; Public Static Final String Column_Name_Subtitle = "Subtitle"; }} Create a database using a SQL helper Once defined as the database appearance, you need to implement methods that create and maintain the database and tables. Here are some typical statements that create and delete a table: Private Const Val sql_create_entries = "Create table $ {} feedentry.table_name (" + "$ {} basolumns._id integer primary key," + "$ {} feedentry.column_name_title Text, "+" $ {} feedentry.column_name_subtitle text) "Private Const Val sql_delete_entries =" Drop Table IF Exists $ {} Feedentry.Table_Name "Private Static Finale String SQL_CREATE_ENTRIES =" CREATE TABLE "+ FEEDTRY.TABLE_NAME +" ("+ + Feedentry ._ID + "Integer Primary Key," + feedentry.column_name_title + + FEEDENTRY.COLUMN_NAME_SUBTITLE + "TEXT)"; Private Static Final String SQL_DELETE_ENTRIES = "Drop Table if it exists" + feedentry.table_Name; Just like the files saved in the internal memory of the device, Android stores your database in your app's private folder. Your data is safe, because by default this area is not accessible to other apps or the user. The sqliteopenhelper sqliteopenhelper Contains a useful Set of APIs for database management. When using this class to get database references, the system performs the operations that are potentially long execution of creating and updating the database only when it is necessary and not when starting the application. All you have to do is call GetwritableDatabase () or GetReadEdatBase (). Note: Because they can have long running, be sure that it's called GetwritableDatabase () or GetReadEdatBase () in a background thread. See Threading on Android for more information. To use SQLiteOpenHelper, create a subclass that replaces ogncreate () and onupgrade () callback methods. It is also advisable to implement the ondowngrade () or Onopen () methods, but they are not necessary. For example, here is an implementation of sqliteopenhelper that uses some of the commands indicated above: Class FeedReaderDBhelper (Context: Context): SQLiteOpenhelper (context, database_name, null, database_version) {exclusion fun owl (db: sqlitatatabase) {db.execsql (sql_create_entries)} Override of fun Onupgrade (DB: SQLiteDatabase, OldVersion: int, NewVersion: int) {// This database is only a cache for online data, then your update policy is / / simply discard I data and restarting db.execsql (sql_delete_entries) Oncreate (db)} Override of fun ondowngrade (db: sqlitatatabase, Oldversion: int, newversion: int) {onupgrade (db, oldversion, newversion)} Associated object {// if you change it Database diagram, you need to increase the database version. CONST VAL DATABASE_VERSION = 1 CONST VAL DATABASE_NAME = "FEEDREADER.DB"}} Public Class FeedReaderDBHELPER extends SQLiteOpenHelper {// If you change the database schema, you need to increase the database version. Static public final database_version = 1; Public Static Finale String Database_Name = "FeedReader.db"; Public FeedReaderDbhelper (context context) {super (context, database_name, null, database_version); } Public void Oncreate (SQLiteDatabase Ter) {db.Execsql (SQL_CREATE_ENTRIES); Onupgrade blank} Public (sqliteDatabase db, int oldversion, int newversion) {// This database is only a cache for online data, so its update policy is / / simply discard the data and restarted db. Execsql (sql_delete_entries); Oncreate (DB); } Public void ondowngrade (sqliteDatabase db, int oldversion, int newversion) {onupgrade (db, oldversion, newversion); }} To access the database, an instance your SQLiteopenhelper subclass: Val = DBHelper FeedReaderDBHELPER (context) FEEDReaderDBHELPER DBHELPER = New FeedReaderDBHELPER (GetContext ()); Enter the information in a data entry database in the database by passing a contentValues â €