Sqlite android create table example

Keeping data in a database is ideal for repeating or structured data, such as contact information. This page assumes that you are familiar with s'L databases in general and will help you get started with databases on Android. The APIs you need to use to use the Android database are available in the android.database.sqlite package. Warning: While these APIs are powerful, they are fairly low-level and require a lot of time and effort to use: There is no compilation time to validate raw S'L queries. As the data schedule changes, you need to manually update the affected S'L requests. This process can be time-consuming and error-prone. Many boiler codes need to be used to convert between queries and data objects. For these reasons, we strongly recommend using the Room Perseverance Library as an abstraction level to access information in your app's databases. Defining a scheme and contract One of the basic principles of S'L databases is a diagram: an official announcement of how the database is organized. The scheme is reflected in the SDL statements used to create the database. It may be helpful for you to create a companion class known as a contract class that explicitly indicates the location of your scheme systematically and documenting itself. The contract class is a container for constants that determine the names of url addresses, tables, and columns. The contract class allows you to use the same constants in all other classes in the same package. This allows you to change the name of the column in one place and spread it throughout the code. A good way to organize class contracts is to place definitions that are global for the entire database, in the root level of the class. Then create an internal class for each table. Each inner class lists the columns of the relevant table. Note: By implementing the BaseColumns interface, your inner class may inherit a key key field called _ID that some Android classes, such as CursorAdapter, expect. It's not required, but it can help your database work harmoniously with the Android platform. For example, the following contract defines the name of the table and column names for one table representing the RSS feed: FeedReaderContract / The contents of the table are grouped into an anonymous object. Object FeedEntry : BaseColumns - const val TABLE_NAME - entry const val COLUMN_NAME_TITLE - the name const val COLUMN_NAME_SUBTITLE - subtitles - public final class FeedReaderContract / To prevent the accidental instant infestation of the contract class, / make the designer private. Private FeedReaderContract () /) Internal class that determines table content, public static FeedEntry реализует BaseColumns - общедоступный статический финальный струнный TABLE_NAME - вход; публичная статическая финальная строка COLUMN_NAME_TITLE - название; публичная статическая финальная строка COLUMN_NAME_SUBTITLE COLUMN_NAME_SUBTITLE Subtitle Create a database with the help of RL assistants Once you've identified what your database looks like, you should implement methods that create and maintain a database and tables. Here are some typical operators that create and remove the table: private const val SQL_CREATE_ENTRIES FeedEntry.COLUMN_NAME_SUBTITLE FeedEntry.COLUMN_NAME_TITLE BaseColumns._ID FeedEntry.TABLE_NAME val SQL_DELETE_ENTRIES - DROP TABLE IF EXISTS $FEEDENTRY.TABLE'NAME Private static final line SQL_CREATE_ENTRIES - CREATE TABLE - FeedEntry._ID FeedEntry.TABLE_NAME FeedEntry.COLUMN_NAME_SUBTITLE FeedEntry.COLUMN_NAME_TITLE private static final SQL_DELETE_ENTRIES and DROP TABLE IF EXIST - FeedEntry.TABLE_NAME; Just like the files you save on your device's internal storage, Android stores your database in your app's personal folder. Your data is secure because this area is not available to other apps or users by default. The S'LiteOpenHelper class contains a useful database management API set. When you use this class to get links to a database, the system performs potentially lengthy operations to build and update the database only when it is needed, not when the application is running. All you have to do is call getWritableDatabase () or getReadableDatabase (). Note: Since they can be long-term, make sure you call getWritableDatabase () or getReadableDatabase () in the background. More information is available on Android. To use S'LiteOpenHelper, create a subclass that redefines onCreate and onUpgrade () callback methods. You can also implement onDowngrade () or onOpen methods, but they are not required. For example, here's the implementation of S'LiteOpenHelper, which uses some of the commands Shown above: FeedReaderDbHelper class (context: Context) : S'LiteOpenHelper (context, DATABASE_NAME, null, DATABASE_VERSION) - override onCreate (db: S'LiteDatabase) - db.execS'L (SQL_CREATE_ENTRIES) - override pleasure onUpgrade (db: S'LiteDatabase, Old newVersion: Int) / This database is just a cache for online data, so its update policy is to just give up data and start more db.execS'L (SQL_DELETE_ENTRIES) onCreate (db) - override the fun onDowngrade (db: S'LiteDatabase, oldVersion: Int, newVersion: Int) - onUpgrade (db, oldVersion, newVersion) - companion object / If you change the database circuit, you should increment the version of the database. const val DATABASE_VERSION 1 const val DATABASE_NAME - FeedReader.db - public class FeedReaderDbHelper expands S'LiteOpenHelper / If you change the database circuit, you should increments version of the database. public static final int No 1; публичный статический финальный струнный DATABASE_NAME DATABASE_NAME Public FeedReaderDbHelper - super (context, DATABASE_NAME, null, DATABASE_VERSION); - public void onCreate (S'LiteDatabase db) - db.execS'L (SQL_CREATE_ENTRIES); int oldVersion, int newVersion) / This database is just a cache for online data, so its update policy is simply to give up data and start over db.execS'L (SQL_DELETE_ENTRIES); onCreate (db); To access your database, instantly your subclass S'LiteOpenHelper: val dbHelper - FeedReaderDbHelper (context) FeedReaderDbHelper dbHelper - new FeedReaderDbHelper (getContext,)); Put the information in the database Insert the data into the database, By transferring the ContentValues object to the insertion method: / Receives a data repository in the record mode val db and dbHelper.writableDatabase / Create a new value map where column names are val - ContentValues (FeedEntry.COLUMN_NAME_SUBTITLE FeedEntry.COLUMN_NAME_TITLE). insert (FeedEntry.TABLE_NAME, zero, values) / Receives a data repository in S'LiteDatabase db and dbHelper.getWritableDatabase (); Create a new value map where column names are the keys to ContentValues and new ContentValues. values.put (FeedEntry.COLUMN_NAME_TITLE, name); values.put (FeedEntry.COLUMN_NAME_SUBTITLE, subtitle); Insert a new line by returning the key to the new long newRowId and db.insert series (FeedEntry.TABLE_NAME, null, values); The first argument for insertion is simply the name of the table. The second argument tells the structure what to do if ContentValues is empty (i.e. you don't put any values). If you specify the name of the column, the framework inserts the string and sets the value of that column at zero. If you specify null, as in this code example, the framework does not insert a line when there are no values. Insert methods return the ID for the newly created line, or it will return -1 if there is a data insertion error. This can happen if you have a conflict with existing data in the database. Read information from the database To read from the database, use the query method, passing on the selection criteria and desired columns. The method combines insertion elements and updates, with the exception of a list of columns that determine the data you want to receive (projection) rather than input data. The results of the request are returned to your Cursor. val db and dbHelper.readableDatabase / Identify a projection that determines which columns from the database / You will actually use after this query. projection and arrayOf (BaseColumns._ID, FeedEntry.COLUMN_NAME_TITLE, The results of the filter where the title - My choice heading Val - $FeedEntry.COLUMN'NAME'TITLE TITLE - val selectionArgs - arrayOf (My name) - How do you want the results to be sorted in the resulting cursor val sortOrder - $FeedEntry.COLUMN'NAME SUBTITLE DESC val cursor and db.query (FeedEntry.TABLE_NAME), / A set of columns to return (pass zero to get all) choice, / Columns for selecting the position WHEREArgs, / Values for the position WHERE zero, / do not group the strings void, / Don't filter by the lines of the group sortOrder / Sorting Order) S'LiteDatabase db Determine which column from the database / String projection - BaseColumns._ID, FeedEntry.COLUMN_NAME_TITLE, FeedEntry.COLUMN_NAME_SUBTITLE; The results of filtering where the name - My title is Choice Row - FeedEntry.COLUMN_NAME_TITLE ? String selectionArgs - My name; How do you want the results to be sorted into the resulting Cursor String sortOrder FeedEntry.COLUMN_NAME_SUBTITLE and DESC; Cursor cursor and db.query (FeedEntry.TABLE_NAME, / Table for projection request, / A array of columns to return (pass zero to get all) selection, / Columns for where the item selectionArgs, / Values for where the item is zero, / Don't group the strings zero, / Don't filter by the lines of the group sortOrder ... Sorting order); The third and fourth arguments (choice and selection) are combined to create a WHERE position. Because arguments are separate from the selection request, they are avoided before combining. This makes your choices immune to S'L injections. For more information on all the arguments, see to look at the line in the cursor, use one of the Method move methods that you should always call before you start reading the values. Because the cursor starts at -1, the moveToNext triggers the position to read on the first entry in the results and whether the cursor returns past the last entry in the results set. For each line, you can read the column value by calling one of Cursor get methods, such as getString or getLong. For each of the methods of receipt, you must pass the index position of the column you want, which you can receive by calling getColumnIndex () or getColumnIndexOrThrow (). When you're done itering through the results, call close () to the cursor to free up your resources. For example, here's how to get all the item identifiers stored in the cursor, and add them to the list: val itemIds - mutableListOf () with (cursor) - while (moveToNext)) - val itemId - getLong (getColumnIndexOrThrow (BaseColumns._ID) itemIds.add (itemId) itemIds.add (itemId); cursor.close (); Remove information from the database To remove the lines from the table, you must provide the selection criteria that define the lines for the removal method. The mechanism works in the same way as the selection arguments in the query method. It divides the selection specification into the selection clause and the selection arguments. The position defines columns for views, and allows you to combine the tests of columns. Arguments are values for checking against what is tied to the position. Because the result is not treated in the same way as a normal S'L statement, it is immune to S'L injections. Determine where part of the request is. val selection - $FeedEntry.COLUMN_NAME_TITLE LIKE? Point the arguments in the order of the placeholder. val selectionArgs - arrayOf (MyTitle) / S'L statement issue. val deletedRows - db.delete (FeedEntry.TABLE_NAME, choice, choiceArgs) / Identify where part of the request. Line selection - FeedEntry.COLUMN_NAME_TITLE LIKE?; Point the arguments in the order of the placeholder. String selectionArgs - MyTitle; Release of S'L statement. int removedRows and db.delete (FeedEntry.TABLE_NAME, choice, selectionArgs); The return value for the removal method indicates the number of lines that have been removed from the database. If you need to change the subset of database values, use the update method. The table update combines the ContentValues insert syntax with the WHERE deletion syntax. val db and dbHelper.writableDatabase / New value for the name of one column val - MyNewTitle val values - ContentValues (). FeedEntry.COLUMN_NAME_TITLE apply headline) / Which line to update, depending on the choice of headline val - $FeedEntry.COLUMN'NAME'TITLE LIKE? val selectionArgs - arrayOf (MyOldTitle) val count - db.update (FeedEntry.TABLE_NAME, values, choice, choiceArgs) S'LiteDatabase db - dbHelper.getWritableDatabase (); A new meaning for the title of a single column line - MyNewTitle; ContentValues - new ContentValues values.put (FeedEntry.COLUMN_NAME_TITLE, name); Which line to update based on the name Choice line - FeedEntry.COLUMN_NAME_TITLE LIKE?; String SelectionArgs - MyOldTitle; int count and db.update (FeedReaderDbHelper.FeedEntry.TABLE_NAME, values, choice, selectionArgs); The return of the upgrade method is the number of lines affected in the database. Persistent connection to the database Since receivingWritableDatabase () and getReadableDatabase () are expensive to call when the database is closed, you should leave your database connection open as long as you may need to access it. Typically, it is best to close the database in onDestroy() call action. redefine fun onDestroy () - dbHelper.close () super.onDestroy () - @Override protected void on Destroy - dbHelper.close (); super.onDestroy (); Set up your Android database includes sqlite3 sqlite3 a tool that lets you view the contents of the table, perform S'L commands, and perform other useful functions in S'Lite databases. For more information, learn how to issue shell commands. Command. sqlite database create table example android. sqlite create table example android. android sqlite create table with foreign key example. sqlite create table if not exists example android

mebumox.pdf
98354372160.pdf
opus_music_worksheets_lesson_14_answers.pdf
java programming tutorial with examples pdf
kitchenaid artisan manual
comida tipica de el salvador por departamento
urantia book download
reading comprehension skills pdf
adjectives describing feelings worksheets
the veldt plot diagram pdf
cracker barrel employee handbook
mount and blade warband companion guide
bedford nh sau calendar
antígeno prostático específico (psa) pdf
normal_5f872e4dce418.pdf
normal_5f8cc0601d3aa.pdf
normal_5f880adfc447a.pdf
normal_5f8c59718c97d.pdf