Foreign key syntax in sqlite android

Before you start, let's have a brief introduction to S'Lite. S'Lite is a lightweight transactional database engine that takes up a small amount of disk and memory storage. These features make it a good choice to create databases on many mobile operating systems such as Android. As we all know, Android is the default engine lite database, making it ideal for Android. Note: Before we get started, let's take a look at a few things to consider when working with S'Lite. They are: First, the integrity of the data type is not supported in S'Lite, so in this case we can put a certain type of data value in a column of a different type of data. Reference integrity is then not supported in S'Lite, as there are no RESTRICTIONS on FOREIGN KEY or JOIN statements. Finally, support for a full Unicode is optional and is not set by default. Now we'll start by building the S'Lite database. In doing so, we'll create a class that handles all the operations you need to work with the database, such as building a database, creating tables, inserting and deleting records, and so on. So the first step is to create a class that inherits from the S'LiteOpenHelper class. This created class will provide us with two ways to redefine how to work with databases. They are: onCreate: It is called when creating a database, here we can create tables and columns in addition to views or triggers. onUpgrade: It is called when changes to the database are made, such as changing, deleting, and creating new tables. List 1: Shows code to include members of the public class DatabaseHelper expands S'LiteOpenHelper - static final string dbNamedemoDB; static final string employeeTableEmployees; static final String colIDEmployeeID; static final String colNameEmployeeName; Static Final String ColAgeAge; static final String colDeptDept Now let's look at the parameters of the superclass designer. These are: Context con: This is the context attached to the database. dataBaseName: This is the name of the database. CursorFactory: Sometimes we can use a class that expands the Cursor class to implement some additional reconciliation or operations on requests running in the database. In this case, we go through the CursorFactory instance to return the link to our derivative class, which will be used instead of the default cursor. Version: This is a version of the database diagram. The designer creates a new empty database with a given name and version. List 2: Shows coding for the designer of the public databaseHelper (context context) - super (context, dbName, null,33); Now we're going to create a database with onCreate. The method creates tables with Kind and trigger. The method is called when the database is created. So we create our table and specify the columns. Listing 3: Shows coding to create a class of public void onCreate (S'LiteDatabase db) / TODO Auto-generated stub method db.execS'L (CREATE TABLE (deptTable) (ColDeptID INTEGER PRIMARY KEY, colDeptName TEXT)); db.execS'L (CREATE TABLE (EMPLOYEETABLE (COLID INTEGER PRIMARY KEY AUTOINCREMENT, ColName TEXT, ColAge INTEger, ColDept INTEGER NOT NULL,FOREIGN KEY (ColDept) db.execS'L (CREATE TRIGGER fk_empdept_deptid - DO INSERT ON EmployeeTable FOR EACH ROW START ColDept) IS NULL) THEN RAISE (ABORT,'Foreign Key Violation') END; END;); db.execS'L (CREATE VIEW ViewEmps AS SELECT EmployeeTable. «ColID» AS _id», ««EmployeeTable». « colName, colAge, DeptTabl. colDeptName from EmployeeTable JOIN deptTable on EmployeeTable. colDept» («DeptTable»)». CalDeptide); InsertDepts (db) insertions; Sometimes you need to update the database by changing the diagram, adding new tables, or changing the types of column data. Thus, this is done by overriding onUpdate. This method is called when you change the number of the version listed in the class designer. List 4: Shows coding to update the public void of the database onUpgrade (S'LiteDatabase db, int oldVersion, int newVersion) -/ TODO Auto-generated stub method db.execs'L (DROP IF TABLE EXISTS db.execS'L (DROP TABLE IF EXIST db.execS'L (DROP TRIGGER PRODUCT DEPT_ID_TRIGGER); db.execs'L (DROP TRIGGER IF EXISTS DEPT_ID_TRIGGER22); db.execs'L (DROP TRIGGER IF EXISTS FK_EMPDEPT_DEPTID); db.execS'L (DROP VIEW IF EXISTS ONCreate (db); Note: When a user wants to make changes to the database, he must change the version number in the class designer. that S'Lite 3 by default does not support foreign key restrictions, however we can force such a restriction by triggers. : Shows the code to create the CREATE TRIGGER trigger fk_empdept_deptid before INSERT ON Employees FOR EACH ROW BEGIN SELECT CASE WHEN ((SELECT DeptID FROM DeptID, WHERE DeptID . Dept) IS NULL) THEN RAISE (ABORT, 'Foreign Key Violation') END; END Now, if a user wants to create code using theCreate method, there will be a change in coding. List 6: Shows the code to create a trigger using onCreate db.execS'L (CREATE TRIGGER fk_empdept_deptid - BEFORE INSERT ON SELECT CASE WHEN ((SELECT 'colDeptID) from DeptTable - WHERE ColDeptID. ColDept) IS NULL) THEN RAISE (ABORT,'Foreign Key Violation') END; END;); After that, we need to do the main S'L operators. You can make any S'L statement that is not a request (such as insertion, deletion or update) using db.execS'L. List 7: Shows code to make statements with S'L db.execS'L (CREATE TABLE (DEPTTable) (COLDeptID INTE PRIMARYGER KEY, colDeptName TEXT); The aforementioned execution is done in the same way as when creating database tables. Now let's put records in the databases. List 8: Shows the code to insert records into the S'LiteDatabase db'this.getWritableDatabase database; ContentValues cv'new ContentValues (); cv.put (colDeptID, 1); cv.put (colDeptName, Sales); db.insert (deptTable, colDeptID, cv); cv.put (colDeptID, 2); cv.put (colDeptName, IT); db.insert (deptTable, colDeptID, cv); db.close(); Now, if we want to update the statements, we can do so in two ways. They are: To perform db.execS'L to perform the db.update method has the following parameters: Table line: This is a table to update the value of the inch ContentValues cv: This is an object of content values that has new values. Line where the position is: This is where the position is to indicate which entry to update. Line Args: These are the arguments where the situation is. List 9: Shows code to update values in the database of public int UpdateEmp (Employee emp) - S'LiteDatabase db'this.getWritableDatabase (); ContentValues cv'new ContentValues (); cv.put (colName, emp.getName()); cv.put(colAge, emp.getAge()); cv.put (colDept, emp.getDept()); db.update (employeeTable, cv, colID-?, new string.valueOf (emp.getID)))); Now let's check the method of deleting lines. There are two ways to do this. They are: To perform db.execS'L To perform db.delete method Listing 10: Shows code to remove the lines of public void DeleteEmp (Employee emp) Finally, you need to fulfill requests. To do this, we can also use two methods. They are: Perform db.raw'y method Run db.query method Listing 11: Shows code to fulfill the requests Of The Curlor getAllDepts () - S'LiteDatabase db'this.getableDatabase (); Cur'db.raw'query (SELECT ColDeptID as _id, ColDeptName from deptTable, new line); The return of chickens; Finally, in this article, we learned about the creation of s'Lite databases in Android. Hope you enjoy reading this! Frequently asked questions on foreign keys from S'Lite: Can you show me how to identify foreign clues in the design of the S'Lite database table? The S'Lite database supports foreign keys, and its foreign key syntax is similar to other databases. A quick foreign key example of S'Lite. A foreign key example to show how this works, first identify two database tables that have no foreign keys: -- -- sellers - CREATE TABLE Sellers (ID INTEGER PRIMARY KEY, first_name TEXT NOT NULL, last_name TEXT NOT NULL, commission_rate REAL NOT NULL); -- -- CUSTOMERS -- CREATE TABLE CUSTOMERS (ID INTEGER PRIMARY KEY, COMPANY_NAME TEXT NOT NULL, STREET_ADDRESS TEXT NOT NULL, CITY TEXT NOT NULL, STATE TEXT NOT NULL, ZIP TEXT NOT NULL); Next, identify the S'Lite table, which has two foreign keys, one that takes the new order table back to the customer table, and the second foreign key that takes the order table back to the sellers' table: -- -- orders -- CREATE TABLE ORDERS (ID INTEGER PRIMARY KEY, customer_id INTEGER, salesperson_id INTEGER, KEY FOREIGN (customer_id salesperson_id) As you can see, the syntax of S'Lite foreign keys is very similar to other databases. Example of foreign key data S'Lite If you want to test this foreign key example of S'Lite in your own S'Lite database, here are some sample data for each of these tables: -- -- of sample data sellers -- INSERT INTO SELLERS VALUES (null, 'Fred', 'Flinstone', 10.0); INSERT INTO sellers VALUES (null, 'Barney', 'Rubble', 10.0); -- -- of selective data customers - INSERT INTO Customers VALUES (null, 'ACME, INC.', '101 Main Street', 'Anchorage', 'AK', '99501'); INSERT INTO values customers (null, 'FOOBAR', '200 Foo Way', 'Louisville', 'KY', '40207'); -- -- of sample data - INSERT INTO ORDERS VALUES (null, 1, 1); INSERT INTO orders VALUES (null, 2, 2); I just tested these foreign key S'Lite examples in my system using the S'Lite 3.4.0 version and they all work fine. Well done.