Saving Data in SQL Databases

[Previous](https://developer.android.com/training/basics/data-storage/files.html)[Next](https://developer.android.com/training/basics/data-storage/databases.html)

This lesson teaches you to

1. [Define a Schema and Contract](https://developer.android.com/training/basics/data-storage/databases.html#DefineContract)
2. [Create a Database Using a SQL Helper](https://developer.android.com/training/basics/data-storage/databases.html#DbHelper)
3. [Put Information into a Database](https://developer.android.com/training/basics/data-storage/databases.html#WriteDbRow)
4. [Read Information from a Database](https://developer.android.com/training/basics/data-storage/databases.html#ReadDbRow)
5. [Delete Information from a Database](https://developer.android.com/training/basics/data-storage/databases.html#DeleteDbRow)
6. [Update a Database](https://developer.android.com/training/basics/data-storage/databases.html#UpdateDbRow)

You should also read

* [Using Databases](https://developer.android.com/guide/topics/data/data-storage.html#db)

Saving data to a database is ideal for repeating or structured data, such as contact information. This class assumes that you are familiar with SQL databases in general and helps you get started with SQLite databases on Android. The APIs you'll need to use a database on Android are available in the [android.database.sqlite](https://developer.android.com/reference/android/database/sqlite/package-summary.html) package.

Define a Schema and Contract

One of the main principles of SQL databases is the schema: a formal declaration of how the database is organized. The schema is reflected in the SQL statements that you use to create your database. You may find it helpful to create a companion class, known as a *contract* class, which explicitly specifies the layout of your schema in a systematic and self-documenting way.

A contract class is a container for constants that define names for URIs, tables, and columns. The contract class allows you to use the same constants across all the other classes in the same package. This lets you change a column name in one place and have it propagate throughout your code.

A good way to organize a contract class is to put definitions that are global to your whole database in the root level of the class. Then create an inner class for each table that enumerates its columns.

**Note:** By implementing the [BaseColumns](https://developer.android.com/reference/android/provider/BaseColumns.html) interface, your inner class can inherit a primary key field called \_ID that some Android classes such as cursor adaptors will expect it to have. It's not required, but this can help your database work harmoniously with the Android framework.

For example, this snippet defines the table name and column names for a single table:

public final class FeedReaderContract {  
    // To prevent someone from accidentally instantiating the contract class,  
    // make the constructor private.  
    private FeedReaderContract() {}  
  
    /\* Inner class that defines the table contents \*/  
    public static class FeedEntry implements BaseColumns {  
        public static final String 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 you have defined how your database looks, you should implement methods that create and maintain the database and tables. Here are some typical statements that create and delete a table:

private static final String SQL\_CREATE\_ENTRIES =  
    "CREATE TABLE " + FeedEntry.TABLE\_NAME + " (" +  
    FeedEntry.\_ID + " INTEGER PRIMARY KEY," +  
    FeedEntry.COLUMN\_NAME\_TITLE + " TEXT," +  
    FeedEntry.COLUMN\_NAME\_SUBTITLE + " TEXT)";  
  
private static final String SQL\_DELETE\_ENTRIES =  
    "DROP TABLE IF EXISTS " + FeedEntry.TABLE\_NAME;

Just like files that you save on the device's [internal storage](https://developer.android.com/guide/topics/data/data-storage.html#filesInternal), Android stores your database in private disk space that's associated application. Your data is secure, because by default this area is not accessible to other applications.

A useful set of APIs is available in the [SQLiteOpenHelper](https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html) class. When you use this class to obtain references to your database, the system performs the potentially long-running operations of creating and updating the database only when needed and *not during app startup*. All you need to do is call[getWritableDatabase()](https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html#getWritableDatabase()) or [getReadableDatabase()](https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html#getReadableDatabase()).

**Note:** Because they can be long-running, be sure that you call [getWritableDatabase()](https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html#getWritableDatabase()) or [getReadableDatabase()](https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html#getReadableDatabase()) in a background thread, such as with [AsyncTask](https://developer.android.com/reference/android/os/AsyncTask.html) or [IntentService](https://developer.android.com/reference/android/app/IntentService.html).

To use [SQLiteOpenHelper](https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html), create a subclass that overrides the [onCreate()](https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html#onCreate(android.database.sqlite.SQLiteDatabase)), [onUpgrade()](https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html#onUpgrade(android.database.sqlite.SQLiteDatabase, int, int)) and [onOpen()](https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html#onOpen(android.database.sqlite.SQLiteDatabase)) callback methods. You may also want to implement [onDowngrade()](https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html#onDowngrade(android.database.sqlite.SQLiteDatabase, int, int)), but it's not required.

For example, here's an implementation of [SQLiteOpenHelper](https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html) that uses some of the commands shown above:

public class FeedReaderDbHelper extends SQLiteOpenHelper {  
    // If you change the database schema, you must increment the database version.  
    public static final int DATABASE\_VERSION = 1;  
    public static final String DATABASE\_NAME = "FeedReader.db";  
  
    public FeedReaderDbHelper(Context context) {  
        super(context, DATABASE\_NAME, null, DATABASE\_VERSION);  
    }  
    public void onCreate(SQLiteDatabase db) {  
        db.execSQL(SQL\_CREATE\_ENTRIES);  
    }  
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
        // This database is only a cache for online data, so its upgrade policy is  
        // to simply to discard the data and start over  
        db.execSQL(SQL\_DELETE\_ENTRIES);  
        onCreate(db);  
    }  
    public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
        onUpgrade(db, oldVersion, newVersion);  
    }  
}

To access your database, instantiate your subclass of [SQLiteOpenHelper](https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html):

FeedReaderDbHelper mDbHelper = new FeedReaderDbHelper(getContext());

Put Information into a Database

Insert data into the database by passing a [ContentValues](https://developer.android.com/reference/android/content/ContentValues.html) object to the [insert()](https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html#insert(java.lang.String, java.lang.String, android.content.ContentValues)) method:

// Gets the data repository in write mode  
SQLiteDatabase db = mDbHelper.getWritableDatabase();  
  
// Create a new map of values, where column names are the keys  
ContentValues values = new ContentValues();  
values.put(FeedEntry.COLUMN\_NAME\_TITLE, title);  
values.put(FeedEntry.COLUMN\_NAME\_SUBTITLE, subtitle);  
  
// Insert the new row, returning the primary key value of the new row  
long newRowId = db.insert(FeedEntry.TABLE\_NAME, null, values);

The first argument for [insert()](https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html#insert(java.lang.String, java.lang.String, android.content.ContentValues)) is simply the table name.

The second argument tells the framework what to do in the event that the [ContentValues](https://developer.android.com/reference/android/content/ContentValues.html) is empty (i.e., you did not [put](https://developer.android.com/reference/android/content/ContentValues.html#put(java.lang.String, byte[])) any values). If you specify the name of a column, the framework inserts a row and sets the value of that column to null. If you specify null, like in this code sample, the framework does not insert a row when there are no values.

Read Information from a Database

To read from a database, use the [query()](https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html#query(boolean, java.lang.String, java.lang.String[], java.lang.String, java.lang.String[], java.lang.String, java.lang.String, java.lang.String, java.lang.String)) method, passing it your selection criteria and desired columns. The method combines elements of [insert()](https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html#insert(java.lang.String, java.lang.String, android.content.ContentValues))and [update()](https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html#update(java.lang.String, android.content.ContentValues, java.lang.String, java.lang.String[])), except the column list defines the data you want to fetch, rather than the data to insert. The results of the query are returned to you in a [Cursor](https://developer.android.com/reference/android/database/Cursor.html) object.

SQLiteDatabase db = mDbHelper.getReadableDatabase();  
  
// Define a *projection* that specifies which columns from the database  
// you will actually use after this query.  
String[] projection = {  
    FeedEntry.\_ID,  
    FeedEntry.COLUMN\_NAME\_TITLE,  
    FeedEntry.COLUMN\_NAME\_SUBTITLE  
    };  
  
// Filter results WHERE "title" = 'My Title'  
String selection = FeedEntry.COLUMN\_NAME\_TITLE + " = ?";  
String[] selectionArgs = { "My Title" };  
  
// How you want the results sorted in the resulting Cursor  
String sortOrder =  
    FeedEntry.COLUMN\_NAME\_SUBTITLE + " DESC";