|  |
| --- |
| space |
|  |
| next>><<prev  **Android SQLite Tutorial**  **SQLite** is an **open-source relational database** i.e. used to perform database operations on android devices such as storing, manipulating or retrieving persistent data from the database.  It is embedded in android bydefault. So, there is no need to perform any database setup or administration task.  Here, we are going to see the example of sqlite to store and fetch the data. Data is displayed in the logcat. For displaying data on the spinner or listview, move to the next page.  **SQLiteOpenHelper** class provides the functionality to use the SQLite database.  **SQLiteOpenHelper class**  The android.database.sqlite.SQLiteOpenHelper class is used for database creation and version management. For performing any database operation, you have to provide the implementation of **onCreate()** and **onUpgrade()** methods of SQLiteOpenHelper class.  **Constructors of SQLiteOpenHelper class**  There are two constructors of SQLiteOpenHelper class.   |  |  | | --- | --- | | **Constructor** | **Description** | | **SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version)** | creates an object for creating, opening and managing the database. | | **SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version, DatabaseErrorHandler errorHandler)** | creates an object for creating, opening and managing the database. It specifies the error handler. |   **Methods of SQLiteOpenHelper class**  There are many methods in SQLiteOpenHelper class. Some of them are as follows:   |  |  | | --- | --- | | **Method** | **Description** | | **public abstract void onCreate(SQLiteDatabase db)** | called only once when database is created for the first time. | | **public abstract void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)** | called when database needs to be upgraded. | | **public synchronized void close ()** | closes the database object. | | **public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion)** | called when database needs to be downgraded. |   **SQLiteDatabase class**  It contains methods to be performed on sqlite database such as create, update, delete, select etc.  **Methods of SQLiteDatabase class**  There are many methods in SQLiteDatabase class. Some of them are as follows:   |  |  | | --- | --- | | **Method** | **Description** | | **void execSQL(String sql)** | executes the sql query not select query. | | **long insert(String table, String nullColumnHack, ContentValues values)** | inserts a record on the database. The table specifies the table name, nullColumnHack doesn't allow completely null values. If second argument is null, android will store null values if values are empty. The third argument specifies the values to be stored. | | **int update(String table, ContentValues values, String whereClause, String[] whereArgs)** | updates a row. | | **Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)** | returns a cursor over the resultset. |   **Example of android SQLite database**  Let's see the simple example of android sqlite database.  File: Contact.java   1. package com.example.sqlite; 2. public class Contact { 3. int \_id; 4. String \_name; 5. String \_phone\_number; 6. public Contact(){   } 7. public Contact(int id, String name, String \_phone\_number){ 8. this.\_id = id; 9. this.\_name = name; 10. this.\_phone\_number = \_phone\_number; 11. } 13. public Contact(String name, String \_phone\_number){ 14. this.\_name = name; 15. this.\_phone\_number = \_phone\_number; 16. } 17. public int getID(){ 18. return this.\_id; 19. } 21. public void setID(int id){ 22. this.\_id = id; 23. } 25. public String getName(){ 26. return this.\_name; 27. } 29. public void setName(String name){ 30. this.\_name = name; 31. } 33. public String getPhoneNumber(){ 34. return this.\_phone\_number; 35. } 37. public void setPhoneNumber(String phone\_number){ 38. this.\_phone\_number = phone\_number; 39. } 40. }   File: DatabaseHandler.java  Now, let's create the database handler class that extends SQLiteOpenHelper class and provides the implementation of its methods.   1. package com.example.sqlite; 2. import java.util.ArrayList; 3. import java.util.List; 5. import android.content.ContentValues; 6. import android.content.Context; 7. import android.database.Cursor; 8. import android.database.sqlite.SQLiteDatabase; 9. import android.database.sqlite.SQLiteOpenHelper; 11. public class DatabaseHandler extends SQLiteOpenHelper { 12. private static final int DATABASE\_VERSION = 1; 13. private static final String DATABASE\_NAME = "contactsManager"; 14. private static final String TABLE\_CONTACTS = "contacts"; 15. private static final String KEY\_ID = "id"; 16. private static final String KEY\_NAME = "name"; 17. private static final String KEY\_PH\_NO = "phone\_number"; 19. public DatabaseHandler(Context context) { 20. super(context, DATABASE\_NAME, null, DATABASE\_VERSION); 21. //3rd argument to be passed is CursorFactory instance 22. } 24. // Creating Tables 25. @Override 26. public void onCreate(SQLiteDatabase db) { 27. String CREATE\_CONTACTS\_TABLE = "CREATE TABLE " + TABLE\_CONTACTS + "(" 28. + KEY\_ID + " INTEGER PRIMARY KEY," + KEY\_NAME + " TEXT," 29. + KEY\_PH\_NO + " TEXT" + ")"; 30. db.execSQL(CREATE\_CONTACTS\_TABLE); 31. } 33. // Upgrading database 34. @Override 35. public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) { 36. // Drop older table if existed 37. db.execSQL("DROP TABLE IF EXISTS " + TABLE\_CONTACTS); 39. // Create tables again 40. onCreate(db); 41. } 43. // code to add the new contact 44. void addContact(Contact contact) { 45. SQLiteDatabase db = this.getWritableDatabase(); 47. ContentValues values = new ContentValues(); 48. values.put(KEY\_NAME, contact.getName()); // Contact Name 49. values.put(KEY\_PH\_NO, contact.getPhoneNumber()); // Contact Phone 51. // Inserting Row 52. db.insert(TABLE\_CONTACTS, null, values); 53. //2nd argument is String containing nullColumnHack 54. db.close(); // Closing database connection 55. } 57. // code to get the single contact 58. Contact getContact(int id) { 59. SQLiteDatabase db = this.getReadableDatabase(); 61. Cursor cursor = db.query(TABLE\_CONTACTS, new String[] { KEY\_ID, 62. KEY\_NAME, KEY\_PH\_NO }, KEY\_ID + "=?", 63. new String[] { String.valueOf(id) }, null, null, null, null); 64. if (cursor != null) 65. cursor.moveToFirst(); 67. Contact contact = new Contact(Integer.parseInt(cursor.getString(0)), 68. cursor.getString(1), cursor.getString(2)); 69. // return contact 70. return contact; 71. } 73. // code to get all contacts in a list view 74. public List<Contact> getAllContacts() { 75. List<Contact> contactList = new ArrayList<Contact>(); 76. // Select All Query 77. String selectQuery = "SELECT  \* FROM " + TABLE\_CONTACTS; 79. SQLiteDatabase db = this.getWritableDatabase(); 80. Cursor cursor = db.rawQuery(selectQuery, null); 82. // looping through all rows and adding to list 83. if (cursor.moveToFirst()) { 84. do { 85. Contact contact = new Contact(); 86. contact.setID(Integer.parseInt(cursor.getString(0))); 87. contact.setName(cursor.getString(1)); 88. contact.setPhoneNumber(cursor.getString(2)); 89. // Adding contact to list 90. contactList.add(contact); 91. } while (cursor.moveToNext()); 92. } 94. // return contact list 95. return contactList; 96. } 98. // code to update the single contact 99. public int updateContact(Contact contact) { 100. SQLiteDatabase db = this.getWritableDatabase(); 102. ContentValues values = new ContentValues(); 103. values.put(KEY\_NAME, contact.getName()); 104. values.put(KEY\_PH\_NO, contact.getPhoneNumber()); 106. // updating row 107. return db.update(TABLE\_CONTACTS, values, KEY\_ID + " = ?", 108. new String[] { String.valueOf(contact.getID()) }); 109. } 111. // Deleting single contact 112. public void deleteContact(Contact contact) { 113. SQLiteDatabase db = this.getWritableDatabase(); 114. db.delete(TABLE\_CONTACTS, KEY\_ID + " = ?", 115. new String[] { String.valueOf(contact.getID()) }); 116. db.close(); 117. } 119. // Getting contacts Count 120. public int getContactsCount() { 121. String countQuery = "SELECT  \* FROM " + TABLE\_CONTACTS; 122. SQLiteDatabase db = this.getReadableDatabase(); 123. Cursor cursor = db.rawQuery(countQuery, null); 124. cursor.close(); 126. // return count 127. return cursor.getCount(); 128. } 130. }   File: MainActivity.java   1. package com.example.sqlite; 3. import java.util.List; 5. import android.os.Bundle; 6. import android.app.Activity; 7. import android.util.Log; 8. import android.view.Menu; 10. public class MainActivity extends Activity { 12. @Override 13. protected void onCreate(Bundle savedInstanceState) { 14. super.onCreate(savedInstanceState); 15. setContentView(R.layout.activity\_main); 17. DatabaseHandler db = new DatabaseHandler(this); 19. // Inserting Contacts 20. Log.d("Insert: ", "Inserting .."); 21. db.addContact(new Contact("Ravi", "9100000000")); 22. db.addContact(new Contact("Srinivas", "9199999999")); 23. db.addContact(new Contact("Tommy", "9522222222")); 24. db.addContact(new Contact("Karthik", "9533333333")); 26. // Reading all contacts 27. Log.d("Reading: ", "Reading all contacts.."); 28. List<Contact> contacts = db.getAllContacts(); 30. for (Contact cn : contacts) { 31. String log = "Id: "+cn.getID()+" ,Name: " + cn.getName() + " ,Phone: " + 32. cn.getPhoneNumber(); 33. // Writing Contacts to log 34. Log.d("Name: ", log); 35. } 36. } 38. @Override 39. public boolean onCreateOptionsMenu(Menu menu) { 40. // Inflate the menu; this adds items to the action bar if it is present. 41. getMenuInflater().inflate(R.menu.activity\_main, menu); 42. return true; 43. } 45. }   [download this sqlite example](http://www.javatpoint.com/src/android/sqlite.zip)  **Output:**  Open Logcat and see the output. It is the basic example of android sqlite without any GUI.  For GUI application with android SQLite, visit next page.  **Output:**  android simple sqlite example output 1  Next Topic[Android Sqlite Example With Spinner](http://www.javatpoint.com/android-sqlite-example-with-spinner)  [<<prev](http://www.javatpoint.com/android-external-storage-example) [next>>](http://www.javatpoint.com/android-sqlite-example-with-spinner) | |

Like/Subscribe us for latest updates or newsletter [RSS Feed](http://feeds.feedburner.com/javatpointsonoo)[Subscribe to Get Email Alerts](http://feedburner.google.com/fb/a/mailverify?uri=javatpointsonoo)[Facebook Page](https://www.facebook.com/javatpoint)[Google Page](https://plus.google.com/u/0/113973042385691981314/posts)[Twitter Page](https://twitter.com/pagejavatpoint)[Blog Page](http://javatpoint.blogspot.com/)[**↑Top**](http://www.javatpoint.com/android-sqlite-tutorial)

**Tutorials**

* [» Core Java Tutorial](http://www.javatpoint.com/java-tutorial)
* [» Servlet Tutorial](http://www.javatpoint.com/servlet-tutorial)
* [» JSP Tutorial](http://www.javatpoint.com/jsp-tutorial)