

Install the latest Eclipse in Ubuntu

This article shows you the way to install the latest version of Eclipse in Ubuntu. If you do not have Java in your system, follow this [link](#) and install the Java first.



Step 1:

Download the desired version of Eclipse from this [link](#).

Step 2:

Open the Terminal (*Ctrl + Alt + T*) and enter the following command to change the directory.

```
cd /opt
```

Step 3:

Enter the command given below to extract the Eclipse from `~/Downloads` directory. If your downloaded file is in any other directory, replace the last parameter by the actual file path.

```
sudo tar -xvzf ~/Downloads/eclipse-jee-mars-R-linux-gtk-x86_64.tar.gz
```

Step 4:

Open another Terminal (*Ctrl + Alt + T*) and enter the following command to create a shortcut file for eclipse.

```
gedit eclipse.desktop
```

Step 5:

In the opened gedit, copy and paste the following text.

```
[Desktop Entry]
Name=Eclipse
Type=Application
Exec=/opt/eclipse/eclipse
Terminal=false
Icon=/opt/eclipse/icon.xpm
Comment=Integrated Development Environment
NoDisplay=false
Categories=Development;IDE;
Name[en]=Eclipse
Name[en_US]=Eclipse
```

Step 6:

Save and close the gedit.

Step 7:

Enter the following command in the terminal to install the shortcut.

```
sudo desktop-file-install eclipse.desktop
```

Now search for Eclipse in the dashboard and open it.

Fix for unreadable tooltips

In Ubuntu, the default tooltip background is black, which makes it unreadable in Eclipse. To fix this problem, follow these steps.

Step 1:

Enter the following commands one by one, in a Terminal

```
sudo gedit /usr/share/themes/Ambiance/gtk-2.0/gtkrc
```

```
sudo gedit /usr/share/themes/Ambiance/gtk-3.0/settings.ini
```

```
sudo gedit /usr/share/themes/Ambiance/gtk-3.0/gtk-main.css
```

Change the values of *tooltip_fg_color* and *tooltip_bg_color* in all the files, to the following values and save the changes.

```
tooltip_fg_color:#000000
tooltip_bg_color:#f5f5c5
```

Step 2:

Restart the Eclipse.

- [Twitter](#)
- [Facebook](#)
- [Google](#)
- [Tumblr](#)

- [Pinterest](#)

Related Posts

- [Add MySQL JDBC Driver to Eclipse](#)



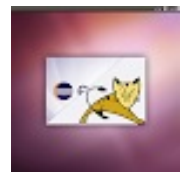
- [Sublime Text](#)



- [Install MySQL with phpMyAdmin on Ubuntu](#)



- [Install Apache Tomcat on Ubuntu](#)



- [Setup Apache Axis2 on Ubuntu](#)



- [Singleton Template For Eclipse](#)



Basic Android Database

Android has a built in SQLite database, which is used to store local information. This article shows you, the quick and dirty way to use database in your application.



Step 1:

Download and open the Android project provided at this [link](#).

This application is a partially completed application, where all the GUI and other GUI related functionalities are already developed. Only the database related Create, Read, Update and Delete (CRUD) operations are left uncompleted.

Step 2: Create Database

Goto the MainActivity.java, *createDatabase* method. This is the place to create a database and a table if not exist. Modify the method as shown below.

```
private void createDatabase() {  
    SQLiteDatabase database = openOrCreateDatabase("contacts.db", Context.MODE_PRIVATE,  
    null);  
    database.execSQL("CREATE TABLE IF NOT EXISTS Contact(first_name TEXT, last_name  
    TEXT, phone TEXT PRIMARY KEY, email TEXT);");  
    database.close();  
}
```

ContextWrapper.openOrCreateDatabase method is used to create a new database if it is not exists, or to open the existing database at the provided location. Context.MODE_PRIVATE is used to declare that the created database file must be local to the application only. It is recommended to keep the database file as a private file of the application. The third parameter is a reference for CursorFactory object. This object will be used to create custom Cursor objects. This value is passed as null reference, since there are no needs for a custom Cursor object (We will receive a default Cursor object when reading from database).

SQLiteDatabase.execSQL method is used to execute an SQL query on the database. The SQLite database queries are much similar to MySQL database queries. However, there are some features which are not supported by SQLite database. For more details about SQLite database, visit to this official [link](#).

The most important thing is, closing the opened connection. Whenever, a resource is opened, it is recommended to close. In Android, if a database connection is not closed, you will get a runtime error.

Step 3: Insert

Open the `ViewActivity.java` and goto the `insertContact` method. In this method, we need to write the code to insert a Contact detail into the database.

Modify the `insertContact` method as shown here.

```
private void insertContact() {
    SQLiteDatabase database = openOrCreateDatabase("contacts.db", Context.MODE_PRIVATE,
null);
    ContentValues values = new ContentValues();
    values.put("first_name", etFirstName.getText().toString());
    values.put("last_name", etLastName.getText().toString());
    values.put("phone", etPhone.getText().toString());
    values.put("email", etEmail.getText().toString());
    database.insert("Contact", null, values);
    database.close();
    this.finish();
}
```

`ContentValues` is a wrapper object, used to keep a set of key-value pairs. Here the key must be a valid column name of the database table and the value can be any valid data to be stored.

`SQLiteDatabase.insert` method receives the table name as the first parameter and the `ContentValues` as the third parameter. The second parameter `nullColumnHack`, is used to provide a nullable column name in case of empty row insertion. More details about `nullColumnHack` is taken from the API documentation and provided below.

SQL doesn't allow inserting a completely empty row without naming at least one column name. If your provided 'values' is empty, no column names are known and an empty row can't be inserted. If not set to null, the “`nullColumnHack`” parameter provides the name of nullable column name to explicitly insert a NULL into in the case where your 'values' is empty.

Step 4: Read

Open the `MainActivity.java`'s `getContacts` method and modify it as shown below.

```

private List<Contact> getContacts() {
    List<Contact> list = new ArrayList<>();
    SQLiteDatabase database = openOrCreateDatabase("contacts.db", Context.MODE_PRIVATE,
null);
    Cursor cursor = database.rawQuery("SELECT * FROM Contact", null);
    cursor.moveToFirst();
    while(!cursor.isAfterLast()) {
        Contact contact = new Contact();
        contact.setFirstName(cursor.getString(0));
        contact.setLastName(cursor.getString(1));
        contact.setPhone(cursor.getString(2));
        contact.setEmail(cursor.getString(3));
        list.add(contact);
        cursor.moveToNext();
    }
    cursor.close();
    database.close();
    return list;
}

```

The rawQuery method is same as execSQL, used to execute an SQL query on the database, but a result is expected from the rawQuery method. The returned result is a Cursor object, which is a wrapper object of the returned rows. This code iterates through the cursor and creates new contacts using the returned details.

rawQuery method has two parameters. The first one is the SQL query as String and the second one is the parameters for SQL query as an array of String. To avoid SQL injection, it is recommended to use prepared statements. The second parameter is used for prepared statement's parameters. (*updateContact* method uses this feature)

Step 5: Update

Open the *ViewActivity.java*'s *updateContact* method and modify it as shown below.

```

private void updateContact() {
    SQLiteDatabase database = openOrCreateDatabase("contacts.db", Context.MODE_PRIVATE,
null);
    ContentValues values = new ContentValues();
    values.put("first_name", etFirstName.getText().toString());
    values.put("last_name", etLastName.getText().toString());
    values.put("phone", etPhone.getText().toString());
    values.put("email", etEmail.getText().toString());
    database.update("Contact", values, "phone = ?", new String[]{contact.getPhone()});
    database.close();
    this.finish();
}

```

In this code, the first parameter of SQLiteDatabase.update method is the table name. The second parameter is the ContentValues. The third parameter is the 'where clause' and the last parameter is an array of parameters for prepared statement. In this code, the returned value of *contact.getPhone()* is used as the where clause parameter.

Step 6: Delete

Modify the `deleteContact` method of the `ViewActivity.java` as provided here.

```
private void deleteContact() {  
    SQLiteDatabase database = openOrCreateDatabase("contacts.db", Context.MODE_PRIVATE,  
null);  
    database.delete("Contact", "phone = ?", new String[]{contact.getPhone()});  
    database.close();  
    this.finish();  
}
```

The `SQLiteDatabase.delete` method needs the table name, where clause and the parameters for prepared statement respectively.

Step 7:

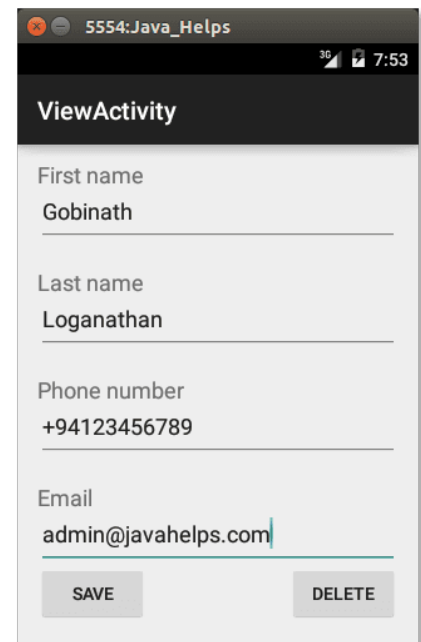
Save all the changes and run the application.

Note:

Since this is a quick and dirty tip for beginners, all the database related codes are written inside the activity classes. The recommended design will be discussed in another tutorial.

Find the projects at [Git Hub](#).

- [Twitter](#)
- [Facebook](#)
- [Google](#)
- [Tumblr](#)
- [Pinterest](#)



5554:Java_Helps

ViewActivity

First name
Gobinath

Last name
Loganathan

Phone number
+94123456789

Email
admin@javahelps.com

SAVE DELETE

Related Posts
