Mobile Programming

Dr. Nader Mahmoud

Lecturer at Computer Science department

Course Content

Agenda

- **Chapter 7** - **Data Persistence**
  - Introduction
  - Saving and Loading User Preferences
    - Accessing Preferences Using an Activity
    - Programmatically Retrieving and Modifying the Preferences Values
  - Persisting Data to Files
    - Saving to Internal Storage
    - Saving to External Storage (SD Card)
    - Choosing the Best Storage Option
  - Creating and Using Databases
    - Creating the DBAdapter Helper Class
    - Using the Database Programmatically

Creating and Using Databases

- For saving **relational data**, using a **database** is much more efficient.
  - For example, if you want to store the test results of all the students in a school, it is much more efficient to use a database to represent them because you can use database querying to retrieve the results of specific students.
- Moreover, using databases enables you to enforce **data integrity** by specifying the relationships between different sets of data.

Creating and Using Databases

- Android uses the **SQLite database system**. The database that you create for an application is **only accessible to itself**; other applications will not be able to access it.
- In this section, you find out how to programmatically create a SQLite database in your Android application.
  - For Android, the SQLite database that you create programmatically in an application is always stored in the **/data/data/<package_name>/databases** folder.

Creating the DBAdapter Helper Class

- A good practice for dealing with databases is to create a **helper class** to encapsulate all the complexities of accessing the data so that it is transparent to the calling code.
  - You will create a helper class called DBAdapter, which creates, opens, closes, and uses a SQLite database.

- In the next example, you are going to create a database named MyDB containing one table named contacts with three columns: *_id*, *name*, and *email*.

Creating the DBAdapter Helper Class

- The SQLLiteDatabase class contains the following methods:

Method	Parameters	Return value
execSQL	String SQL	void
insert	String table	long
	String nullColumnHack	
	ContentValues values	
delete	String table	Long
	String whereClause	
	String[] whereArgs	
update	String table	long
	ContentValues values	
	String whereClause	
	String[] whereArgs	

Creating the DBAdapter Helper Class

- The SQLLiteDatabase class contains the following methods:

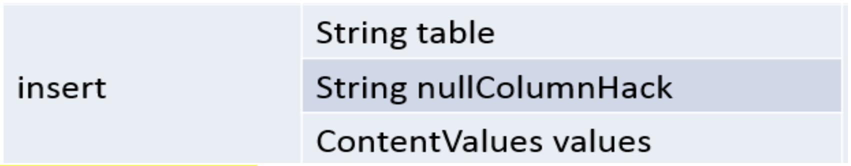
Method	Parameters	Return value
query	Boolean distinct	Cursor
	String table	
	String[] cloumns	
	String selection	
	String[] selectionArgs	
	String groupBy	
	String having	
	String orderBy	
	String Limit	

DBAdapter.java

```
package fci.third.dbtest;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

public class DBAdapter {
    static final String KEY_ROWID = "_id";
    static final String KEY_NAME = "name";
    static final String KEY_EMAIL = "email";
    static final String TAG = "DBAdapter";
    static final String DATABASE_NAME = "MyDB";
    static final String DATABASE_TABLE = "contacts";
    static final int DATABASE_VERSION = 1;
static final String DATABASE_CREATE =
    "create table contacts (_id integer primary key autoincrement, "
        + "name text not null, email text not null);";
final Context context;
DatabaseHelper DBHelper;
SQLiteDatabase db;

public DBAdapter(Context ctx) {
  this.context = ctx;
  DBHelper = new DatabaseHelper(context);
}

private static class DatabaseHelper extends SQLiteOpenHelper {
  DatabaseHelper(Context context) {
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
   }
```

```
    @Override
    public void onCreate(SQLiteDatabase db) {
        try {
            db.execSQL(DATABASE_CREATE);   // create table contacts
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.w(TAG, "Upgrading database from version " + oldVersion + " to "
                + newVersion + ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS contacts");      // delete table
        onCreate(db);     // create table contacts
    }
}
```

```
//---opens the database---
public DBAdapter open() throws SQLException {
    db = DBHelper.getWritableDatabase();
    return this;
}

//---closes the database---
public void close() {
    DBHelper.close();
}

//insert a contact into the database - returns the ID of the inserted row or -1 if error
public long insertContact(String name, String email) {
    ContentValues initialValues = new ContentValues();
    initialValues.put(KEY_NAME, name);
    initialValues.put(KEY_EMAIL, email);
    return db.insert(DATABASE_TABLE, null, initialValues);
}
```



insert	String table
	String nullColumnHack
	ContentValues values

```
//---deletes a particular contact---
public boolean deleteContact(long rowId) {
    return db.delete(DATABASE_TABLE, KEY_ROWID + "=" + rowId, null) > 0;
}
```

delete	String table
	String whereClause
	String[] whereArgs

```
//---retrieves all the contacts---
public Cursor getAllContacts() {
    return db.query(DATABASE_TABLE,
    new String[]{KEY_ROWID, KEY_NAME,
            KEY_EMAIL}, null, null, null, null, null);
}
```

query	String table	Cursor
	String[] cloumns	
	String selection	
	String[] selectionArgs	
	String groupBy	
	String having	
	String orderBy	

```
 //---updates a contact---
public boolean updateContact(long rowId, String name, String email) {
    ContentValues args = new ContentValues();
    args.put(KEY_NAME, name);
    args.put(KEY_EMAIL, email);
    return db.update(DATABASE_TABLE, args, KEY_ROWID + "=" + rowId, null) > 0;
}
```

update	String table
	ContentValues values
	String whereClause
	String[] whereArgs

```
  //---retrieves a particular contact---
  public Cursor getContact(long rowId) throws SQLException {
    Cursor mCursor =
        db.query(true, DATABASE_TABLE, new String[]{KEY_ROWID, KEY_NAME,
                 KEY_EMAIL}, KEY_ROWID + "=" + rowId, null,  null, null, null, null);
    if (mCursor != null) {
      mCursor.moveToFirst();
    }
    return mCursor;
  }
}
```

query	Boolean distinct	Cursor
	String table	
	String[] cloumns	
	String selection	
	String[] selectionArgs	
	String groupBy	
	String having	
	String orderBy	

Creating the DBAdapter Helper Class

- You first define several constants to contain the various fields for the table that you are going to create in your database.
  - In particular, the DATABASE_CREATE constant contains the SQL statement for creating the contacts table within the MyDB database.
- Within the DBAdapter class, you also add a private class that extends the **SQLiteOpenHelper** class. SQLiteOpenHelper is a helper class in Android to manage database creation and version management.
  - In particular, you must override the onCreate() and onUpgrade() methods.

Using the Database Programmatically

- With the DBAdapter helper class created, you are now ready to use the database.

- Example: Use DBAdapter class to insert two records.

MainActivity.java

```java
package fci.third.dbtest;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        DBAdapter db = new DBAdapter(this);
        //---add a contact---
        db.open();
        long id = db.insertContact("Mohamed Malhat", "m.gmalhat@yahoo.com");
        id = db.insertContact("Ahmed Said", "ahmed88@yahoo.com");
        db.close();
    }
}
```

```java
//---opens the database---
public DBAdapter open() throws SQLException {
    db = DBHelper.getWritableDatabase();
    return this;
}
```

```java
//---closes the database---
public void close() {
    DBHelper.close();
}
```

```java
public long insertContact(String name, String email) {
    ContentValues initialValues = new ContentValues();
    initialValues.put(KEY_NAME, name);
    initialValues.put(KEY_EMAIL, email);
    return db.insert(DATABASE_TABLE, null, initialValues);
}
```

insert

Adding Contacts

- In this example, you create an instance of the DBAdapter class:

```
DBAdapter db = new DBAdapter(this);
```

- The insertContact() method returns the ID of the inserted row. If an error occurs during the operation, it returns –1.

Retrieving All the Contacts

- To retrieve all the contacts in the contacts table, use the *getAllContacts()* method of the DBAdapter class.

MainActivity.java

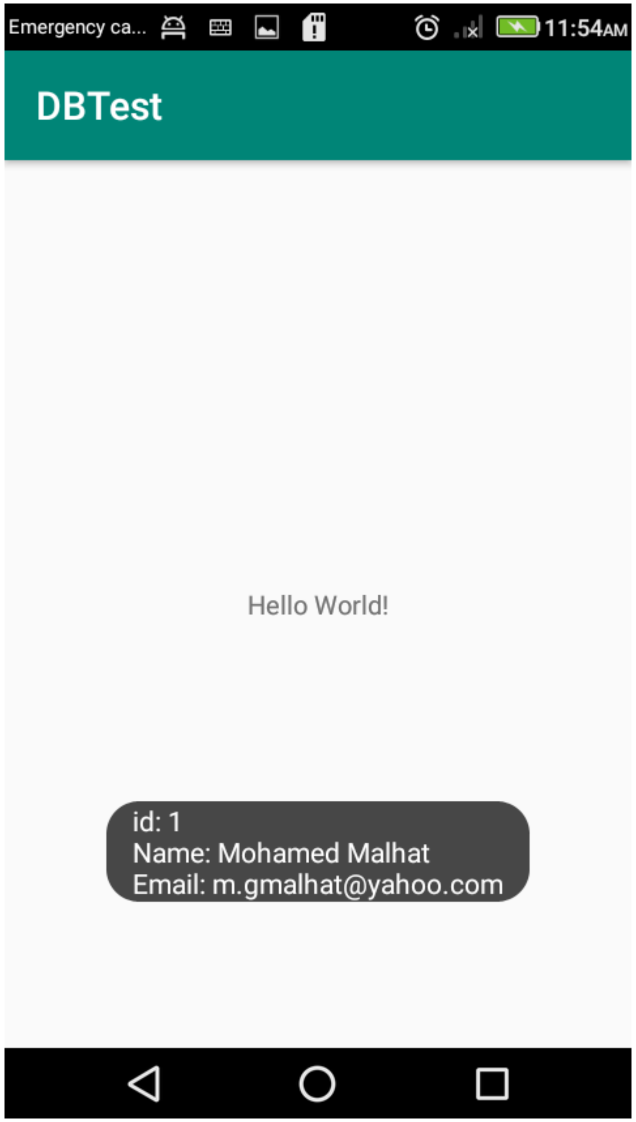
```
package fci.third.dbtest;

import androidx.appcompat.app.AppCompatActivity;
import android.database.Cursor;
import android.os.Bundle;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        DBAdapter db = new DBAdapter(this);
```

MainActivity.java

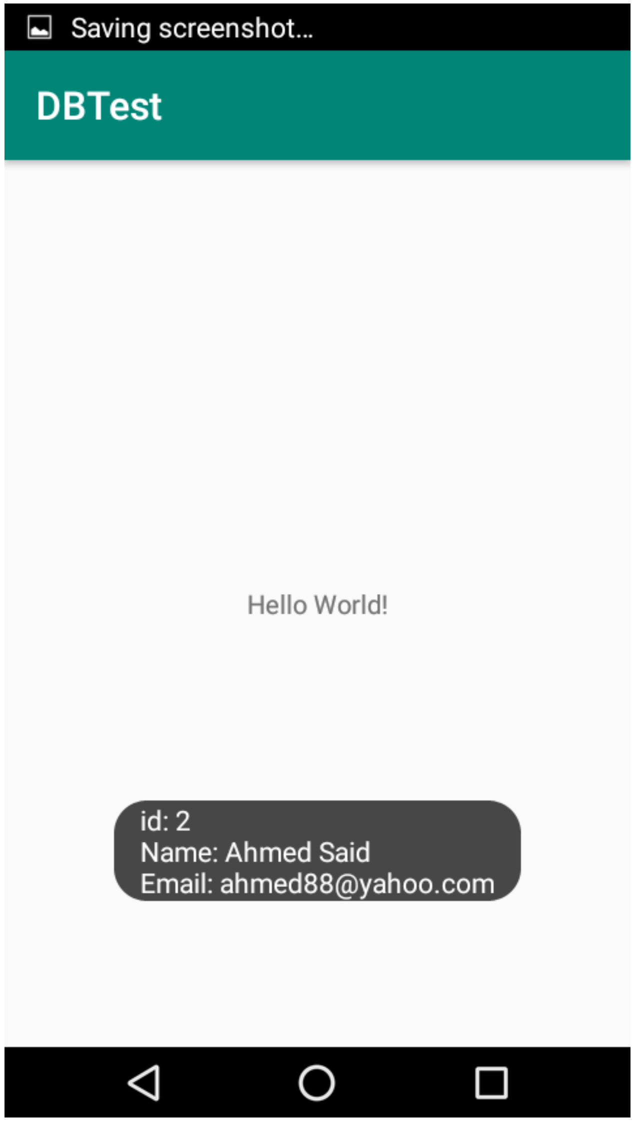
```
        // --- retrieve all contacts
        db.open();
        Cursor c = db.getAllContacts();
        if (c.moveToFirst()) {
            do {
                DisplayContact(c);
            } while (c.moveToNext());
        }
        db.close();
    }
    public void DisplayContact(Cursor c) {
        Toast.makeText(this,  "id: " + c.getString(0) + "\n" + "Name: " + c.getString(1) +
                              "\n" + "Email: " + c.getString(2), Toast.LENGTH_LONG).show();
    }
}
```

```
//---retrieves all the contacts---
public Cursor getAllContacts() {
  return db.query(DATABASE_TABLE,
  new String[]{KEY_ROWID, KEY_NAME,
      KEY_EMAIL}, null, null, null, null, null);
}
```

Retrieving All the Contacts



id: 1
Name: Mohamed Malhat
Email: m.gmalhat@yahoo.com



id: 2
Name: Ahmed Said
Email: ahmed88@yahoo.com

Retrieving All the Contacts

- The result of getAllContacts() method is returned as a Cursor object.
- To display all the contacts, you first need to call the *moveToFirst()* method of the Cursor object. If it succeeds (which means at least one row is available), then you display the details of the contact using the DisplayContact() method.
- To move to the next contact, call the moveToNext() method of the Cursor object.

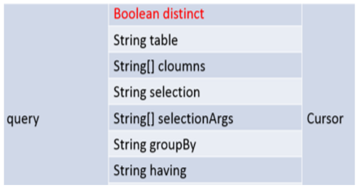
Retrieving a Single Contact

- To retrieve a single contact using its ID, call the getContact() method of the DBAdapter class.

MainActivity.java

```
package fci.third.dbtest;
import android.database.Cursor;
import android.os.Bundle;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        DBAdapter db = new DBAdapter(this);
        db.open();
        Cursor c = db.getContact(2);
        if (c.moveToFirst()){
            DisplayContact(c);
        }else{
            Toast.makeText(this, "No contact found", Toast.LENGTH_LONG).show();
        }
```

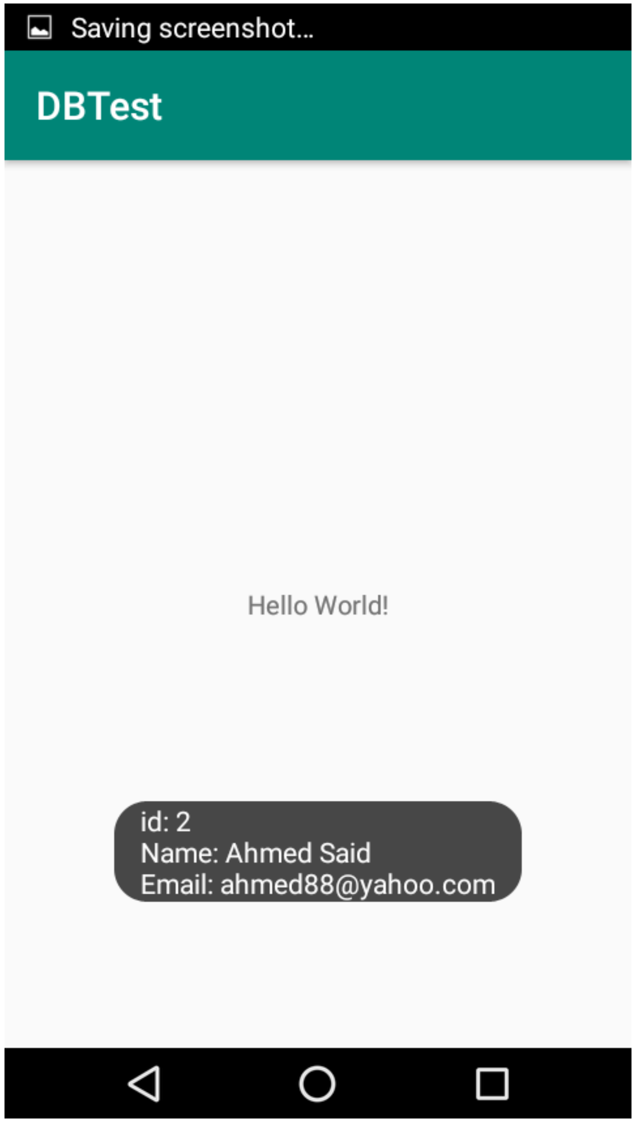
```
public Cursor getContact(long rowId) throws SQLException {
    Cursor mCursor =
            db.query(true, DATABASE_TABLE, new String[]{KEY_ROWID, KEY_NAME,
                    KEY_EMAIL}, KEY_ROWID + "=" + rowId, null,  null, null, null, null)
    if (mCursor != null) {
        mCursor.moveToFirst();
    }
    return mCursor;
}
```



query	Boolean distinct	Cursor
	String table	
	String[] cloumns	
	String selection	
	String[] selectionArgs	
	String groupBy	
	String having	

MainActivity.java

```
    db.close();
  }

  public void DisplayContact(Cursor c) {
    Toast.makeText(this,  "id: " + c.getString(0) + "\n"
                   + "Name: " + c.getString(1) +
                   "\n" + "Email: " + c.getString(2),
                    Toast.LENGTH_LONG).show();
  }
}
```



id: 2
Name: Ahmed Said
Email: ahmed88@yahoo.com

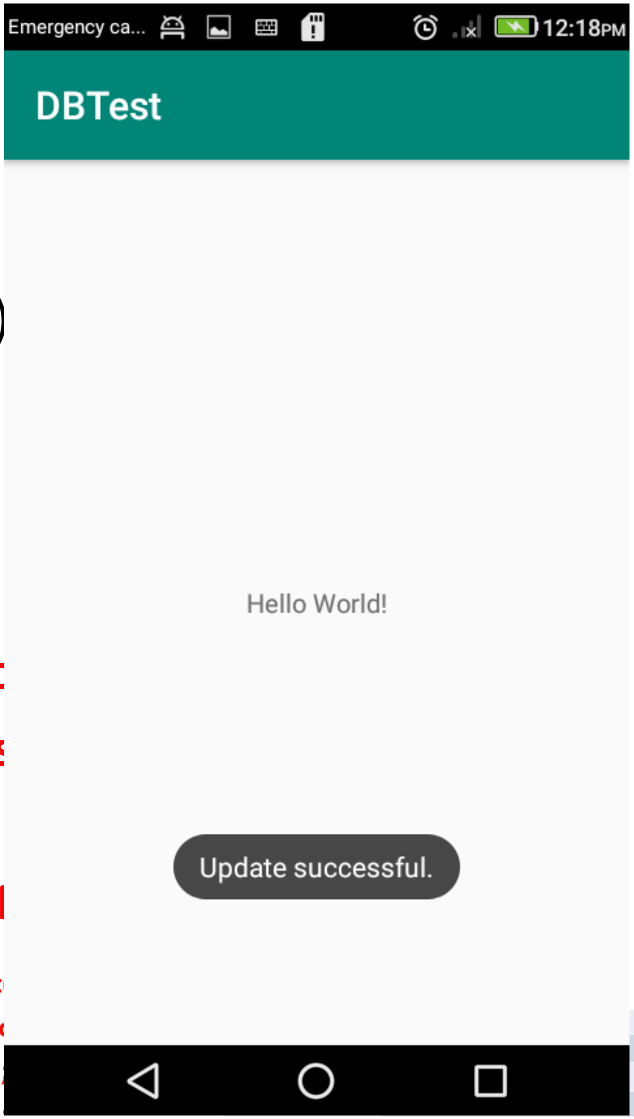
Updating a Contact

- To update a particular contact, call the *updateContact()* method in the DBAdapter class by passing the ID of the contact you want to update and new attributes values.

MainActivity.java

```
package fci.third.dbtest;
import android.database.Cursor;
import android.os.Bundle;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
public class MainActivity extends AppCompatActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState)
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
     DBAdapter db = new DBAdapter(this);
     db.open();
     if (db.updateContact(1, "CS Dept", "CS@yahoo.co
        Toast.makeText(this, "Update successful.", Toas
     }else{
        Toast.makeText(this, "Update failed.", Toast.LE
     }
    db.close();
  }
}
```

```
public boolean updateContact
   ContentValues args = new C
   args.put(KEY_NAME, name);
   args.put(KEY_EMAIL, email);
   return db.update(DATABASE_TABLE, args, KEY_ROWID + "=" + rowId, null) > 0;
}
```



DBTest

Hello World!

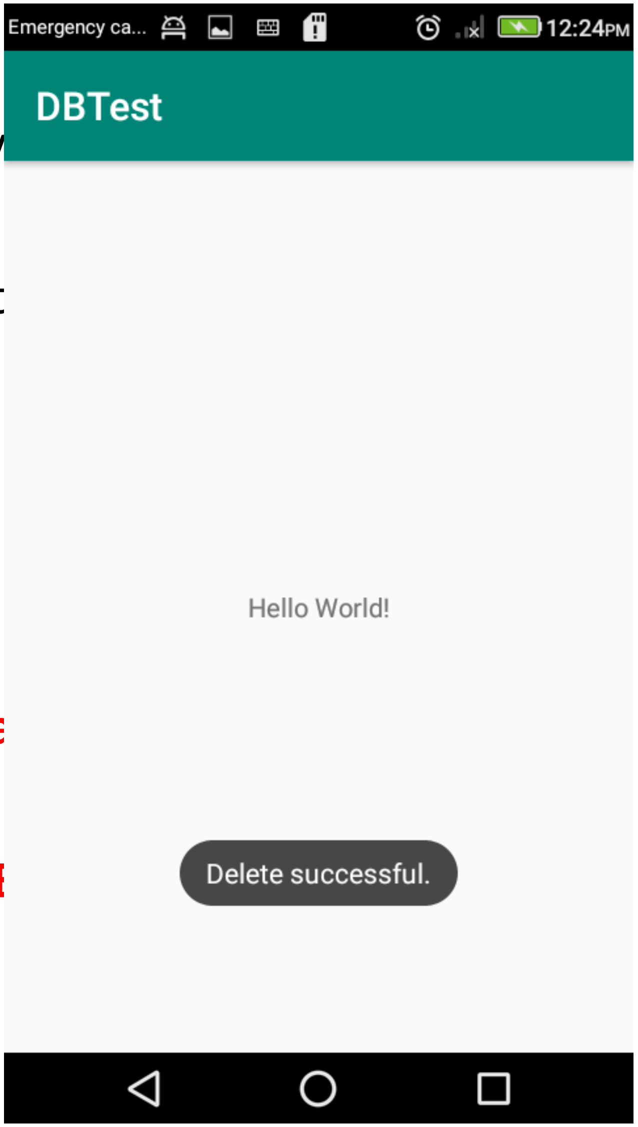
Update successful.

Deleting a Contact

- To delete a contact, use the deleteContact() method in the DBAdapter class by passing the ID of the contact you want to delete.

MainActivity.java

```
package fci.third.dbtest;
import android.database.Cursor;
import android.os.Bundle;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
public class MainActivity extends AppCompatActivity
    @Override
    protected void onCreate(Bundle savedInstanceStat
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
       DBAdapter db = new DBAdapter(this);
       db.open();
       if (db.deleteContact(1)) {
          Toast.makeText(this, "Delete successful.", Toa
       } else{
          Toast.makeText(this, "Delete failed.", Toast.LE
       }
       db.close();
    }
}
```



Delete successful.

Upgrading a Database

- Sometimes, after creating and using the database, you might need to add additional tables, change the schema of the database, or add columns to your tables.
- In this case, you need to migrate your existing data from the old database to a newer one.
- To upgrade the database, change the DATABASE_VERSION constant to a value higher than the previous one.

Upgrading a Database

- For example, if its previous value was 1, change it to 2:

```
public class DBAdapter {
    static final int DATABASE_VERSION = 2;
```

- When you run the application one more time, you see the following message in the logcat window of Android Studio:

```
DBAdapter(8705): Upgrading database from version 1 to 2,
which will destroy all old data
```

End of Lecture