Database (SQLite) in Android

Database connectivity, Data Storage & Retreival

A.R. Kazemi

Database / Relational DB

- **Database** is a managed and structured mechanism for data storage and retrieval
  - Compared to using raw **Files** for data storage & retrieval
- **Relational Databases**
  - Keep data files in Relations which can be represented as
    - N-Tuples: **Contacts Relation =** {(1, "Alpha", "alpha@k.com"), (2, "Sigma", "sigma@k.com"), (3, "Taraneh", "taraneh@k.com"), …}
    - or Tables: **Contacts Table =**

id	name	email
1	Alpha	alpha@k.com
2	Sigma	sigma@k.com
3	Taraneh	taraneh@k.com
…	…	…

  - Use Relational Algebra to organize data and relations among them

DBMS / FlatFileDB

- **DBMS: DataBase Management System**
  - A complex software system which can create, keep & manage many databases
  - Examples: MySql, PostgreSql, MS SqlServer, Oracle, IBM DB2, MongoDB, …
  - Run as standalone services which accept connections from client and respond to their requests.

- **Flat File DB:**
  - A form of providing databases service using a software library which is linked to the program not as a standalone servicing program
  - The databases are stored as files which are commonly posed beside the application or in its data directories.
  - No DB client/server and no need to making network/socket connection
  - Examples: MS Access, SQLite, IndexedDB, WebSQL

SQL / NoSQL

- **SQL: Structured Query Language**
  - A language for manipulating & querying data used in communication with SQL-based DBMSs or FlatFileDBs.
  - Currently most well known DBs are SQL-Based
  - Examples SQL-based DBs: MySql, PostgreSql, SqlServer, Oracle, DB2, SQLite

  - Example **SQL Data Definition Queries (DDQ)**:
    - CREATE TABLE Contact(id INTEGER PRIMARY KEY, name TEXT, email TEXT);
  - Example **SQL Data Manipulation Queries (DMQ)**:
    - SELECT * FROM Contacts Where name='alpha';
    - INSERT INTO Contacts (id, name, email) VALUES (4, 'Tina', 'tina@k.com');
    - UPDATE Contacts SET name='Taraneh' WHERE name='Ava';
    - DELETE Contacts WHERE name='Tina';

- **NoSQL: Database Systems which does not use SQL**
  - Not all DBMSs are SQL-based
  - Examples: MongoDB, Redit, IndexedDB…

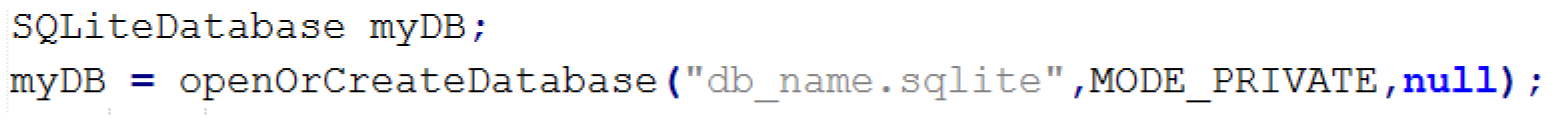
FlatFileSQL / SQLite

- **FlatFileSQL**
  - Flat File Databeses which use SQL as the communication language with the using application program.
  - Examples: SQLite, WebSQL, MS Access

- **SQLite:**
  - An open source FlatFileSQL based DB
  - It is available as (static/dynamic) libraries in many computing platforms
    - Windows/Linux/Mac/Android/…
  - Its API/Adapter are available in most Programming languages & frameworks
    - C/C++/Qt/Java/C#/Web(WebSQL)/PHP/Python/…

SQLite Database in Android

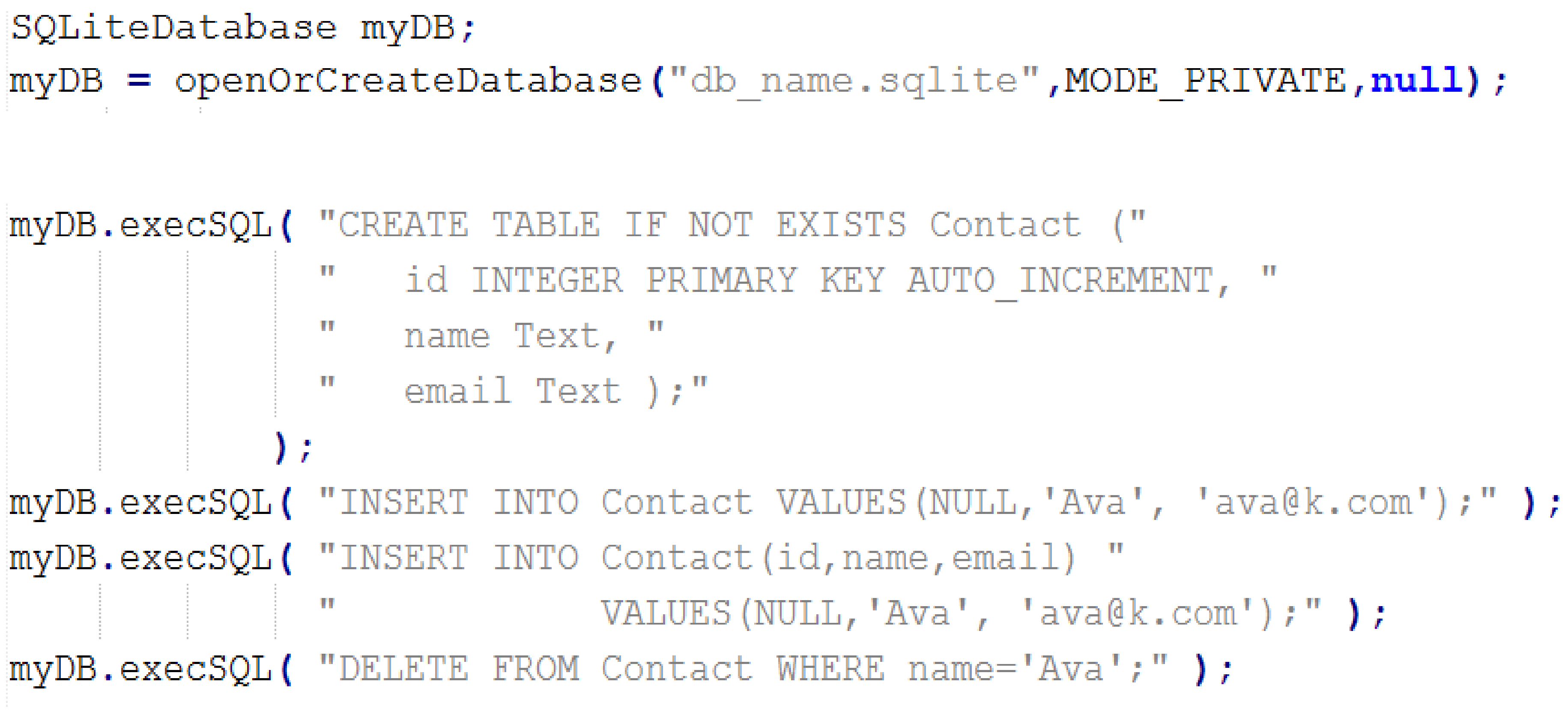
- SQLite is available in Android by importing/using following class:
  - import **android.database.sqlite.SQLiteDatabase;**
- You can create a DB by static methods like:
  1. Context.openOrCreateDatabase(“db name”, MODE_PRIVATE, null);
  2. SQLiteDatabase. openOrCreateDatabase(“db name”, MODE_PRIVATE, null);
  3. This method has multiple other overloaded forms
- So when you are in Activity (which is a Context) you can simply call:



```
SQLiteDatabase myDB;
myDB = openOrCreateDatabase("db_name.sqlite",MODE_PRIVATE,null);
```

Using SQLiteDatabase to Run Query

- Executing Create/Insert/Delete/Update on opened DB:
  - Use execSQL() method:

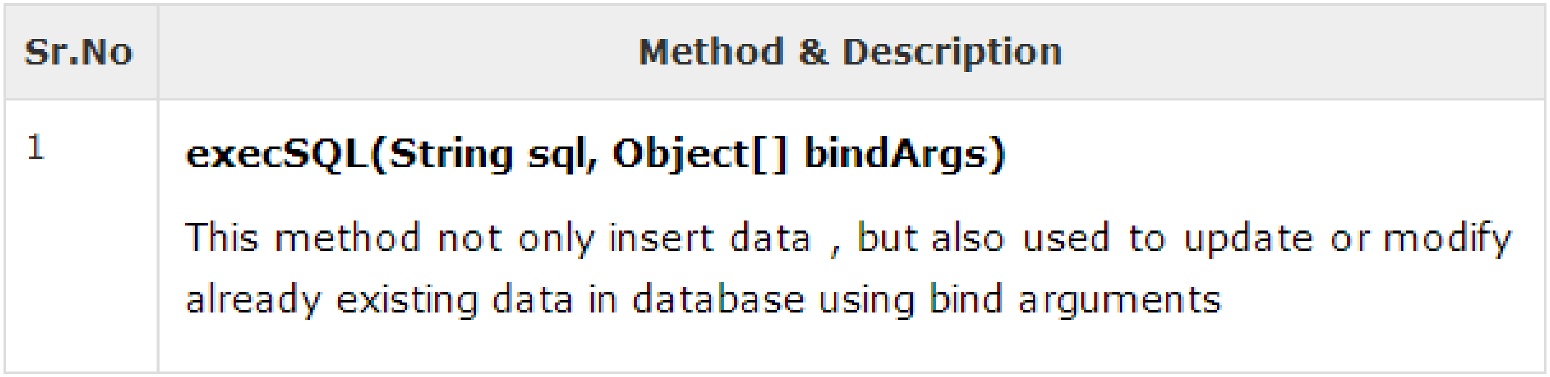


```
SQLiteDatabase myDB;
myDB = openOrCreateDatabase("db_name.sqlite",MODE_PRIVATE,null);


myDB.execSQL( "CREATE TABLE IF NOT EXISTS Contact ("
              "   id INTEGER PRIMARY KEY AUTO_INCREMENT, "
              "   name Text, "
              "   email Text );"
            );
myDB.execSQL( "INSERT INTO Contact VALUES(NULL,'Ava', 'ava@k.com');" );
myDB.execSQL( "INSERT INTO Contact(id,name,email) "
              "            VALUES(NULL,'Ava', 'ava@k.com');" );
myDB.execSQL( "DELETE FROM Contact WHERE name='Ava';" );
```

Slides By Alireza Kazemi,
Content From: https://www.androidhive.info/2011/11/android-sqlite-database-tutorial/ &
https://www.tutorialspoint.com/android/android_sqlite_database.htm &

Using SQLiteDatabase to Run Query

- execSQL() method:



Sr.No	Method & Description
1	**execSQL(String sql, Object[] bindArgs)** This method not only insert data , but also used to update or modify already existing data in database using bind arguments

- bindArgs parameter is for passing parameters of queries which need arguments. Example:

```
String nameToDel = "Ava";
myDB.execSQL( "DELETE FROM Contact WHERE name=?;",
              new String{nameToDel} );
```

Using SQLiteDatabase to Run Query

- Executing Data Fetch Queries over an open SQLite DB:
  - Use SQLiteDatabase.rawSQL() method:
  - The result of SELECT is a Data Cursor

```
Cursor resSet = myDB.rawQuery("SELECT * FROM Contact",null);
resSet.moveToFirst();
int id = resSet.getInteger(0);
String name = resSet.getString(1);
String email = resSet.getString(2);
```

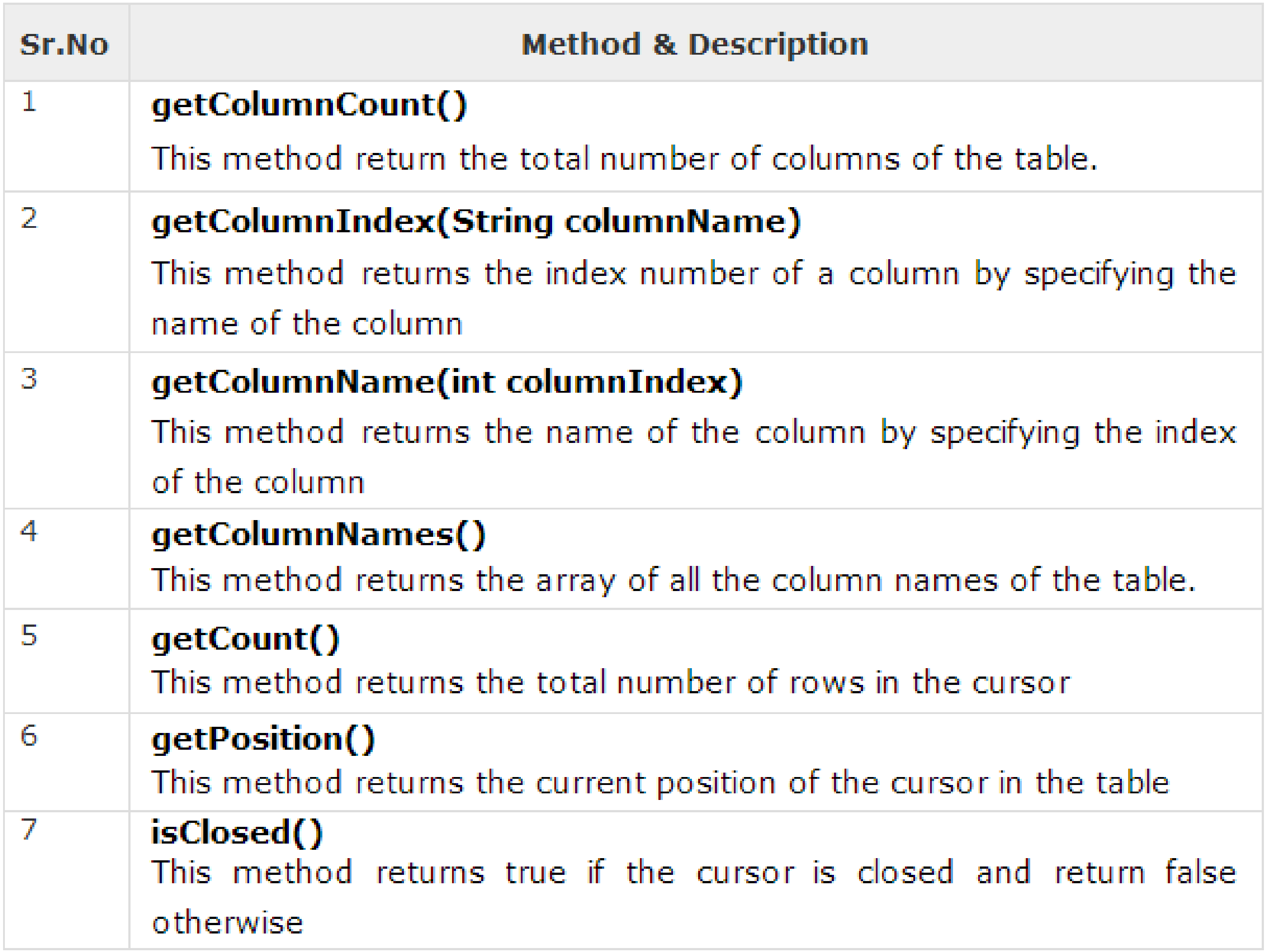
Using SQLiteDatabase to Run Query

- Executing Data Fetch Queries over an open SQLite DB:
  - You can go to the next record by calling **Cursor.moveToNext()**

```
resSet.moveToFirst();
do {
    ... // fetch current record

    // call resSet.getXXX(i) to fetch current record
    //   column i which is of type XXX
    //   XXX in {Integer, String, Float}
}while(resSet.moveToNext());
```

Using SQLiteDatabase to Run Query

- Other useful methods of **Cursor** class



Sr.No	Method & Description
1	**getColumnCount()** This method return the total number of columns of the table.
2	**getColumnIndex(String columnName)** This method returns the index number of a column by specifying the name of the column
3	**getColumnName(int columnIndex)** This method returns the name of the column by specifying the index of the column
4	**getColumnNames()** This method returns the array of all the column names of the table.
5	**getCount()** This method returns the total number of rows in the cursor
6	**getPosition()** This method returns the current position of the cursor in the table
7	**isClosed()** This method returns true if the cursor is closed and return false otherwise

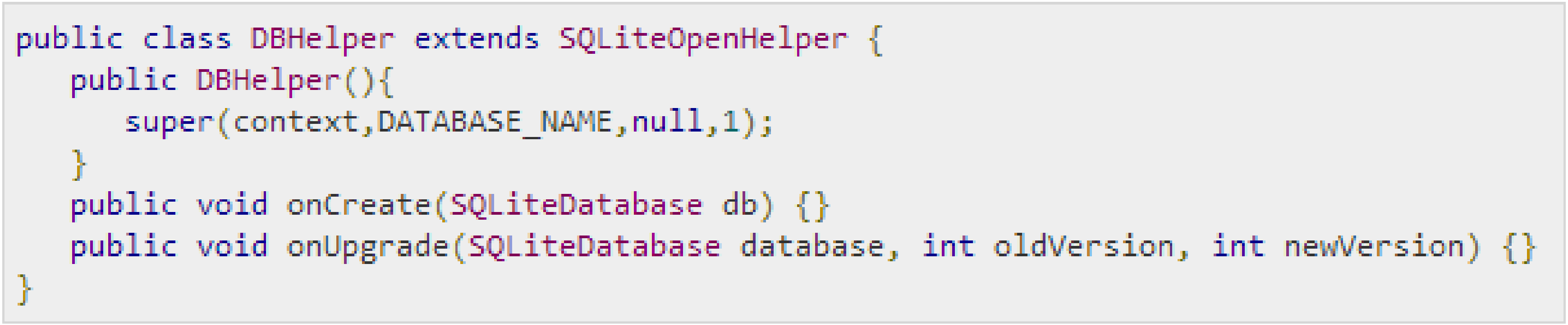
Two Advanced Concepts DBAL/ORM

- **DBAL, DataBase Abstraction Layer**: a library or piece of code which allows using DB without or with lower dependency to the underlying DB technology
  - Hides the final DB which is used
  - Provides custom methods for Data Access

- **ORM, Object Relational Mapper**: a lib or piece of code which maps the program objects to DB records & VS.
  - Data is stored in Relational DB as records
  - Data is hydrated/un-serialized to objects in program space

Creating our Own **DBAL/ORM** as a class named **DBHelper**

- Create a class which extends the **SQLiteOpenHandler**
  - **onCreate()**: is called if the DB does not exists and needs to be created, this function is responsible for initial creation of DB
  - **onUpgrade()**: is called when the current DB version is lower than the requested newVersion which user has specified



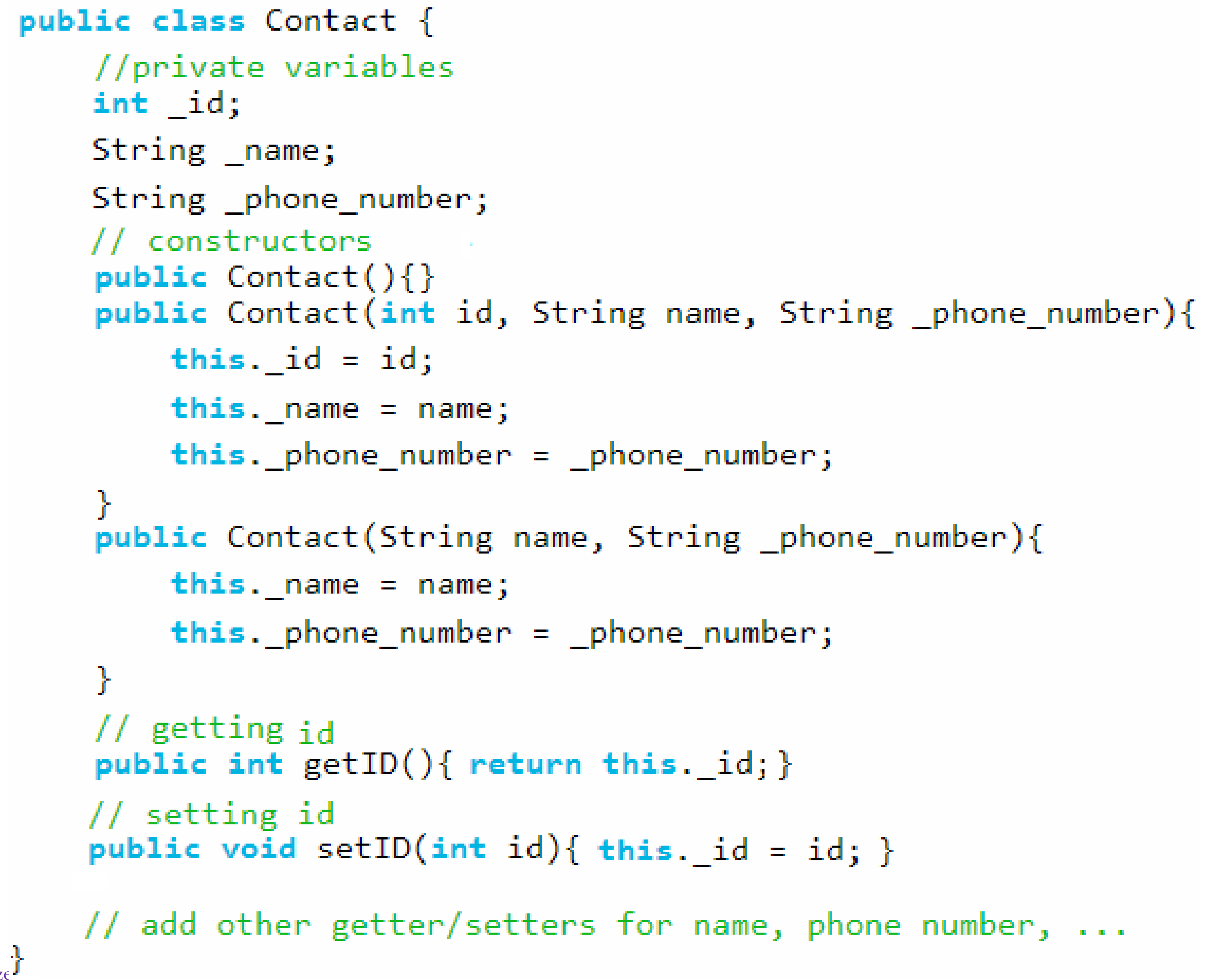
```
public class DBHelper extends SQLiteOpenHelper {
   public DBHelper(){
      super(context,DATABASE_NAME,null,1);
   }
   public void onCreate(SQLiteDatabase db) {}
   public void onUpgrade(SQLiteDatabase database, int oldVersion, int newVersion) {}
}
```

Slides By Alireza Kazemi,
Content From: https://www.androidhive.info/2011/11/android-sqlite-database-tutorial/ &
https://www.tutorialspoint.com/android/android_sqlite_database.htm &

Example The ContactsDBApp

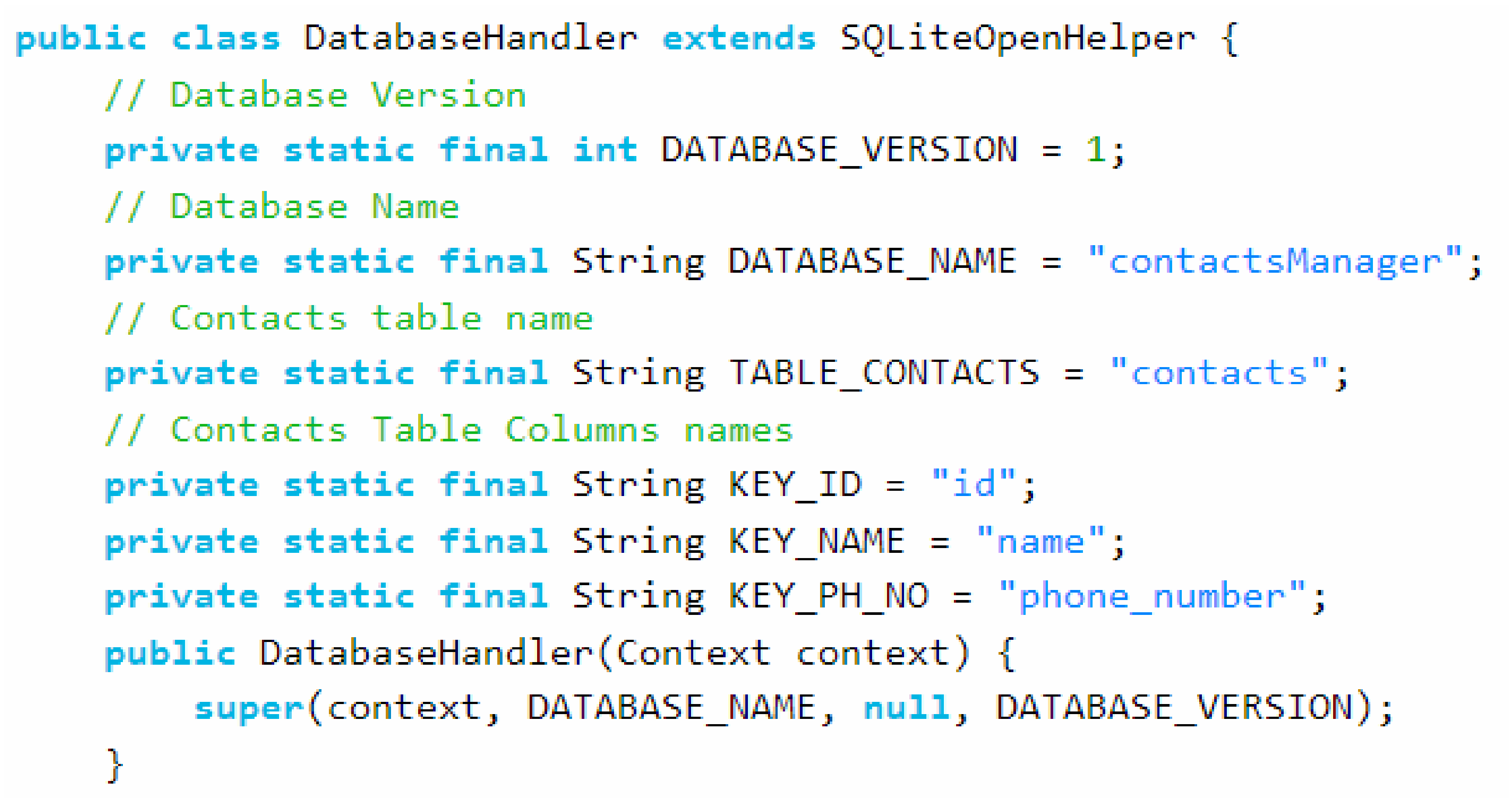
- A simple App which stores some Contact records in DB
  - It is fully object oriented (represents Contact records as objects)
- ContactsDBApp Components:
  1. Define a Contact class which provides objects for keeping Contact records
  2. Create DatabaseHandler class which is our DBAL (also is a simple ORM)
  3. Create Some Contact Records and fetch & Log them in the ADB Console

ContactsDBApp: The Contact Class



```java
public class Contact {
    //private variables
    int _id;
    String _name;
    String _phone_number;
    // constructors
    public Contact(){}
    public Contact(int id, String name, String _phone_number){
        this._id = id;
        this._name = name;
        this._phone_number = _phone_number;
    }
    public Contact(String name, String _phone_number){
        this._name = name;
        this._phone_number = _phone_number;
    }
    // getting id
    public int getID(){ return this._id;}
    // setting id
    public void setID(int id){ this._id = id; }

    // add other getter/setters for name, phone number, ...
}
```

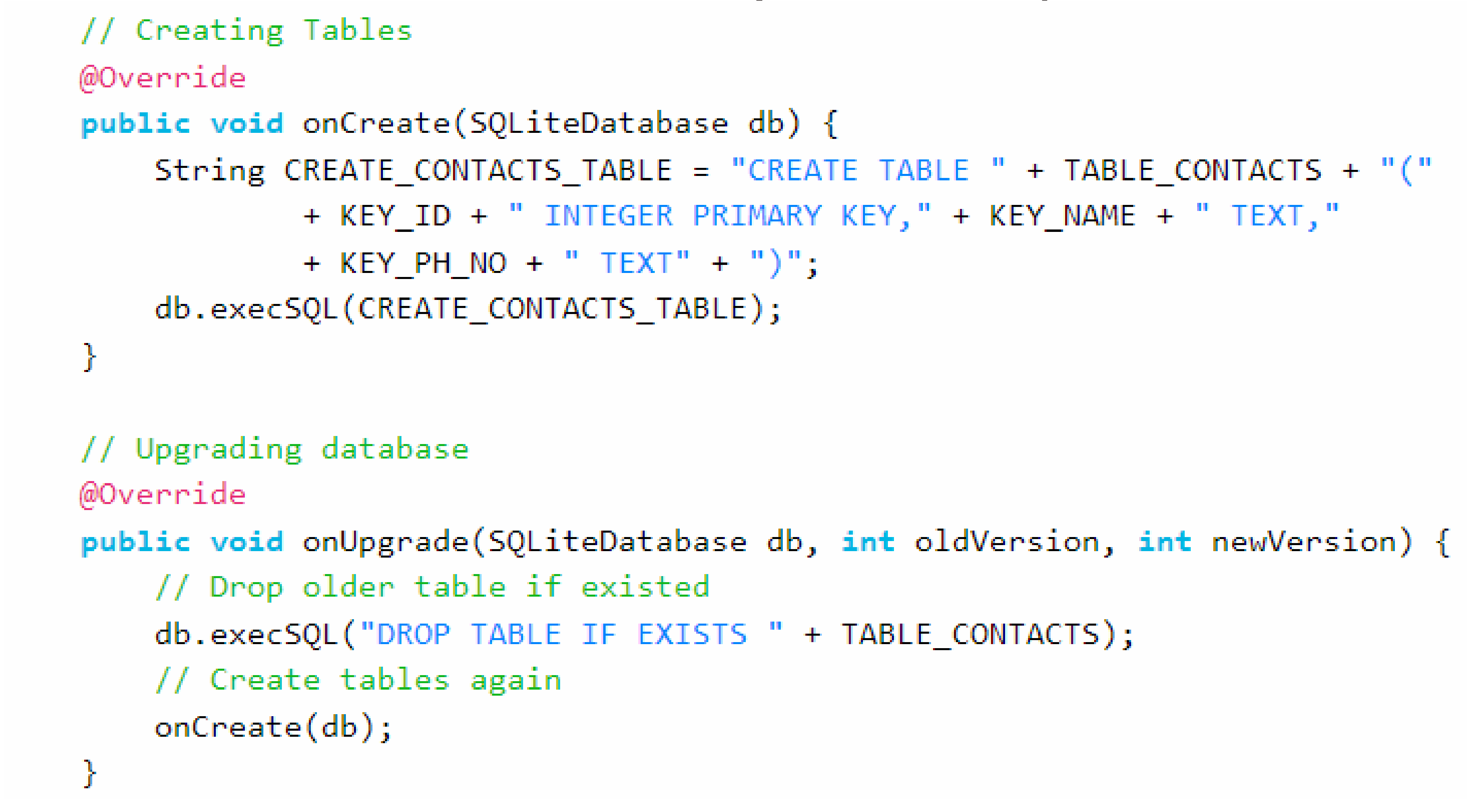
ContactsDBApp: The DatbaseHelper / Handler as our simple DBAL/ORM



```java
public class DatabaseHandler extends SQLiteOpenHelper {
    // Database Version
    private static final int DATABASE_VERSION = 1;
    // Database Name
    private static final String DATABASE_NAME = "contactsManager";
    // Contacts table name
    private static final String TABLE_CONTACTS = "contacts";
    // Contacts Table Columns names
    private static final String KEY_ID = "id";
    private static final String KEY_NAME = "name";
    private static final String KEY_PH_NO = "phone_number";
    public DatabaseHandler(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
```

Slides By Alireza Kazemi,
Content From: https://www.androidhive.info/2011/11/android-sqlite-database-tutorial/ &
https://www.tutorialspoint.com/android/android_sqlite_database.htm &

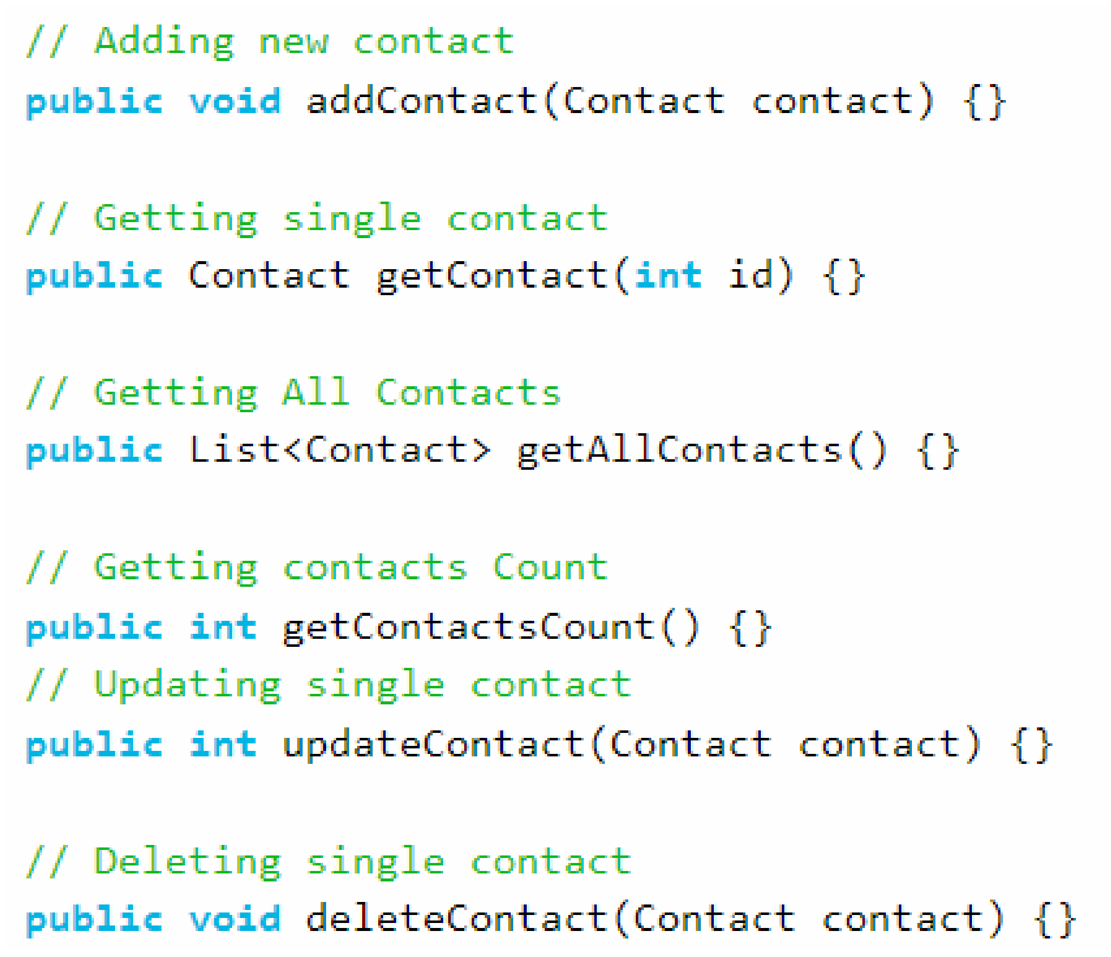
ContactsDBApp: The DatbaseHelper / Handler as our simple DBAL/ORM



```java
// Creating Tables
@Override
public void onCreate(SQLiteDatabase db) {
    String CREATE_CONTACTS_TABLE = "CREATE TABLE " + TABLE_CONTACTS + "("
            + KEY_ID + " INTEGER PRIMARY KEY," + KEY_NAME + " TEXT,"
            + KEY_PH_NO + " TEXT" + ")";
    db.execSQL(CREATE_CONTACTS_TABLE);
}

// Upgrading database
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    // Drop older table if existed
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_CONTACTS);
    // Create tables again
    onCreate(db);
}
```

ContactsDBApp: The DatbaseHelper Add all CRUD methods

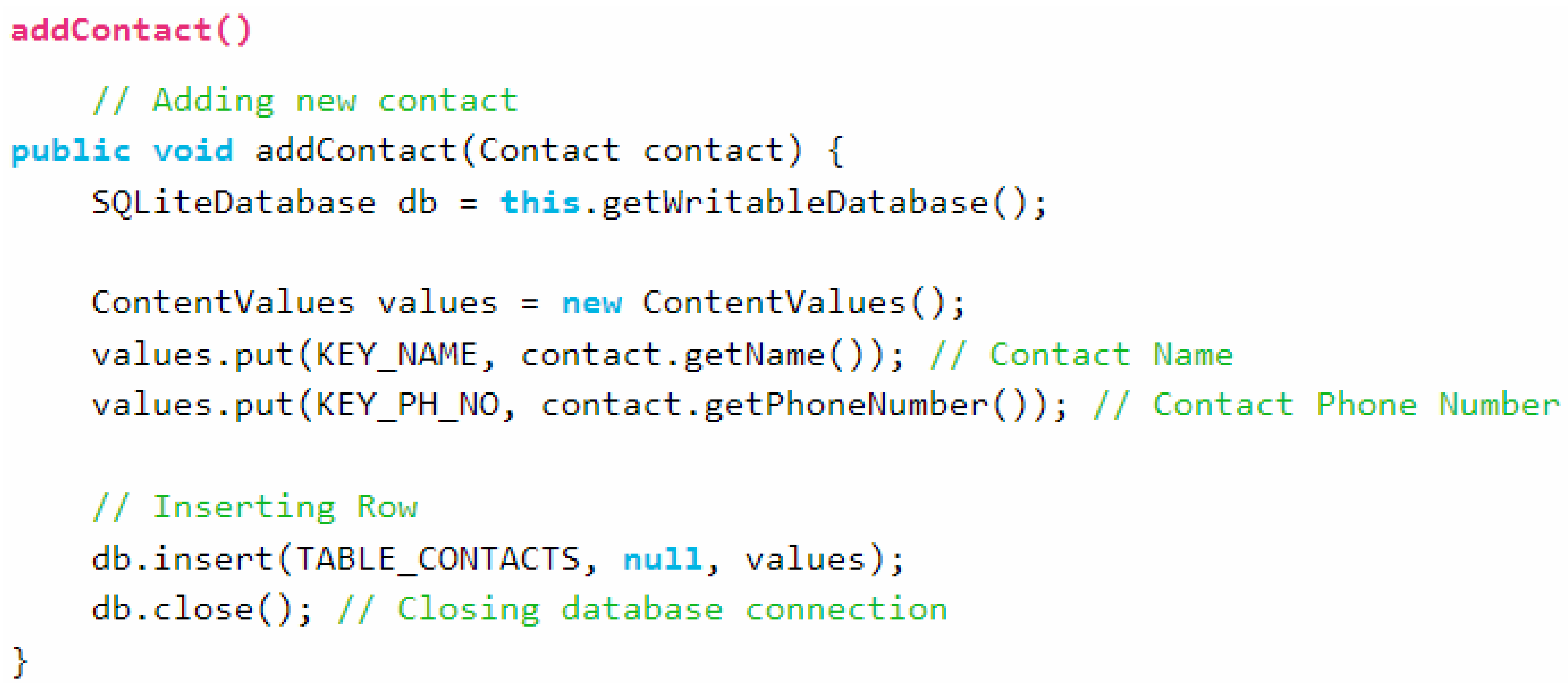
- **CRUD Operations:**
  - **Create**
  - **Read**
  - **Update Delete**
- **Notice:**
  - Almost all methods operate on Contact Objects.
  - This is a property in ORMs



```
// Adding new contact
public void addContact(Contact contact) {}

// Getting single contact
public Contact getContact(int id) {}

// Getting All Contacts
public List<Contact> getAllContacts() {}

// Getting contacts Count
public int getContactsCount() {}
// Updating single contact
public int updateContact(Contact contact) {}

// Deleting single contact
public void deleteContact(Contact contact) {}
```

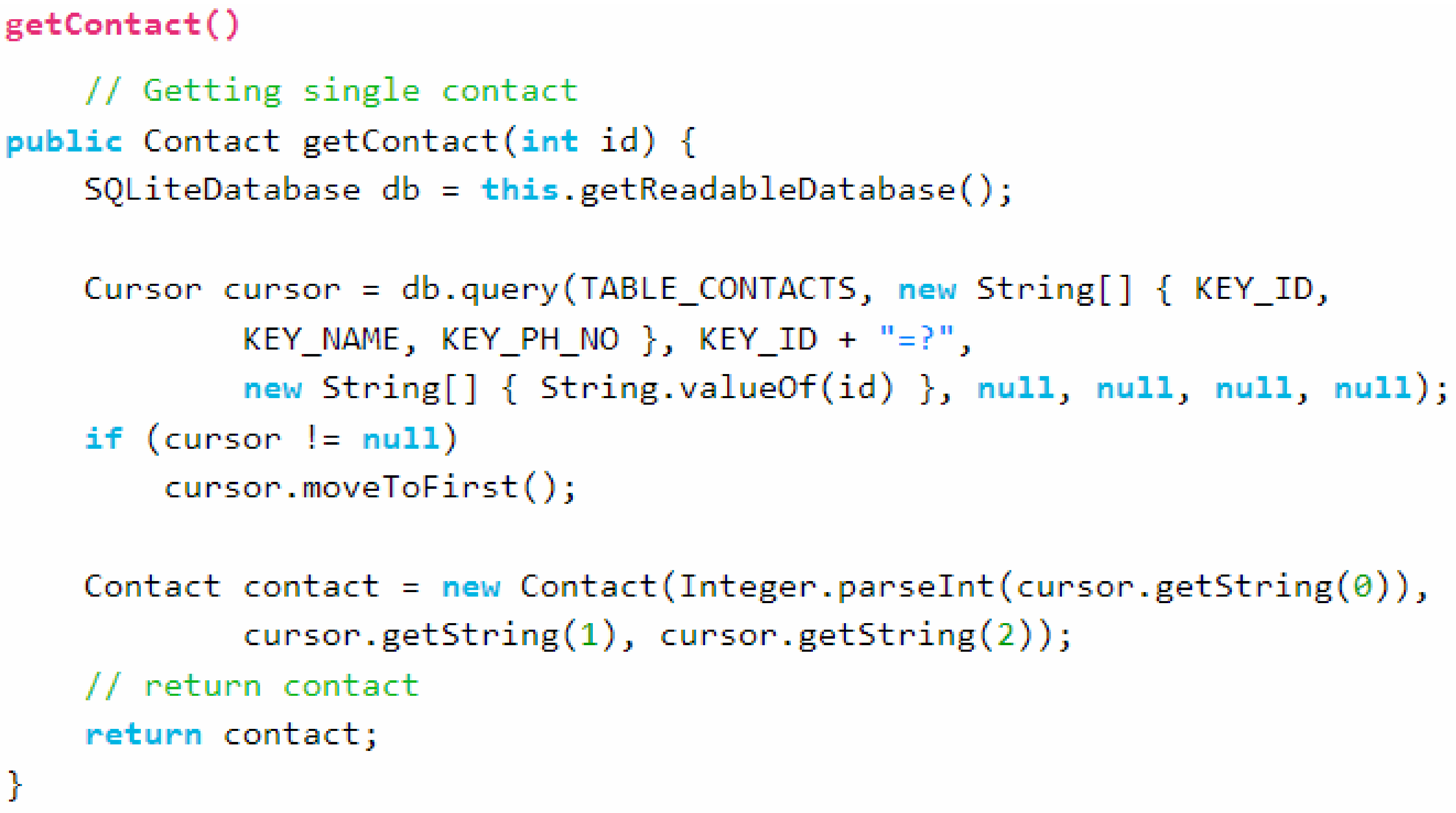
Slides By Alireza Kazemi,
Content From: https://www.androidhive.info/2011/11/android-sqlite-database-tutorial/ &
https://www.tutorialspoint.com/android/android_sqlite_database.htm &

ContactsDBApp: The DatbaseHelper



```
addContact()

    // Adding new contact
public void addContact(Contact contact) {
    SQLiteDatabase db = this.getWritableDatabase();

    ContentValues values = new ContentValues();
    values.put(KEY_NAME, contact.getName()); // Contact Name
    values.put(KEY_PH_NO, contact.getPhoneNumber()); // Contact Phone Number

    // Inserting Row
    db.insert(TABLE_CONTACTS, null, values);
    db.close(); // Closing database connection
}
```

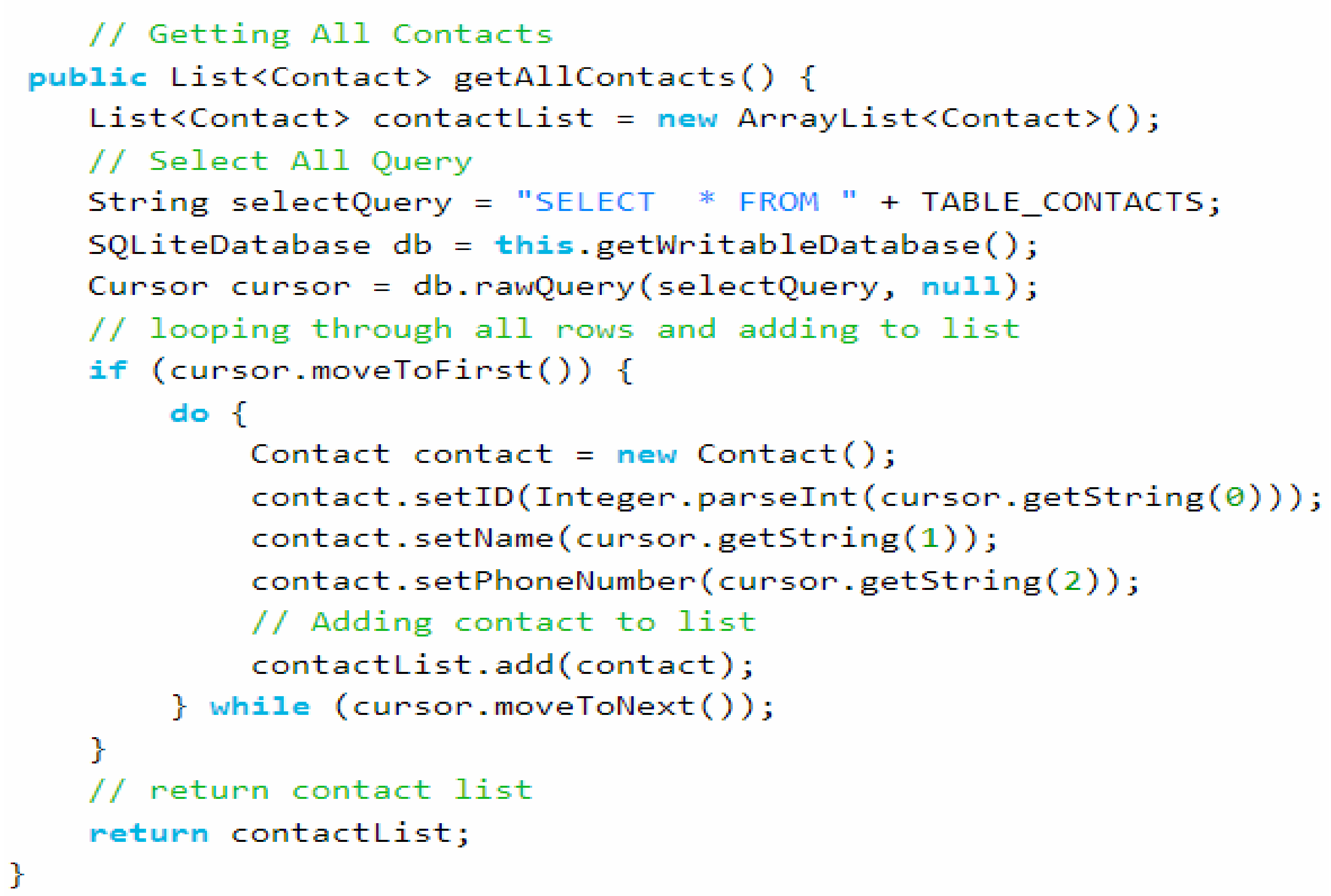
ContactsDBApp: The DatbaseHelper



```
getContact()

    // Getting single contact
public Contact getContact(int id) {
    SQLiteDatabase db = this.getReadableDatabase();

    Cursor cursor = db.query(TABLE_CONTACTS, new String[] { KEY_ID,
            KEY_NAME, KEY_PH_NO }, KEY_ID + "=?",
            new String[] { String.valueOf(id) }, null, null, null, null);
    if (cursor != null)
        cursor.moveToFirst();

    Contact contact = new Contact(Integer.parseInt(cursor.getString(0)),
            cursor.getString(1), cursor.getString(2));
    // return contact
    return contact;
}
```

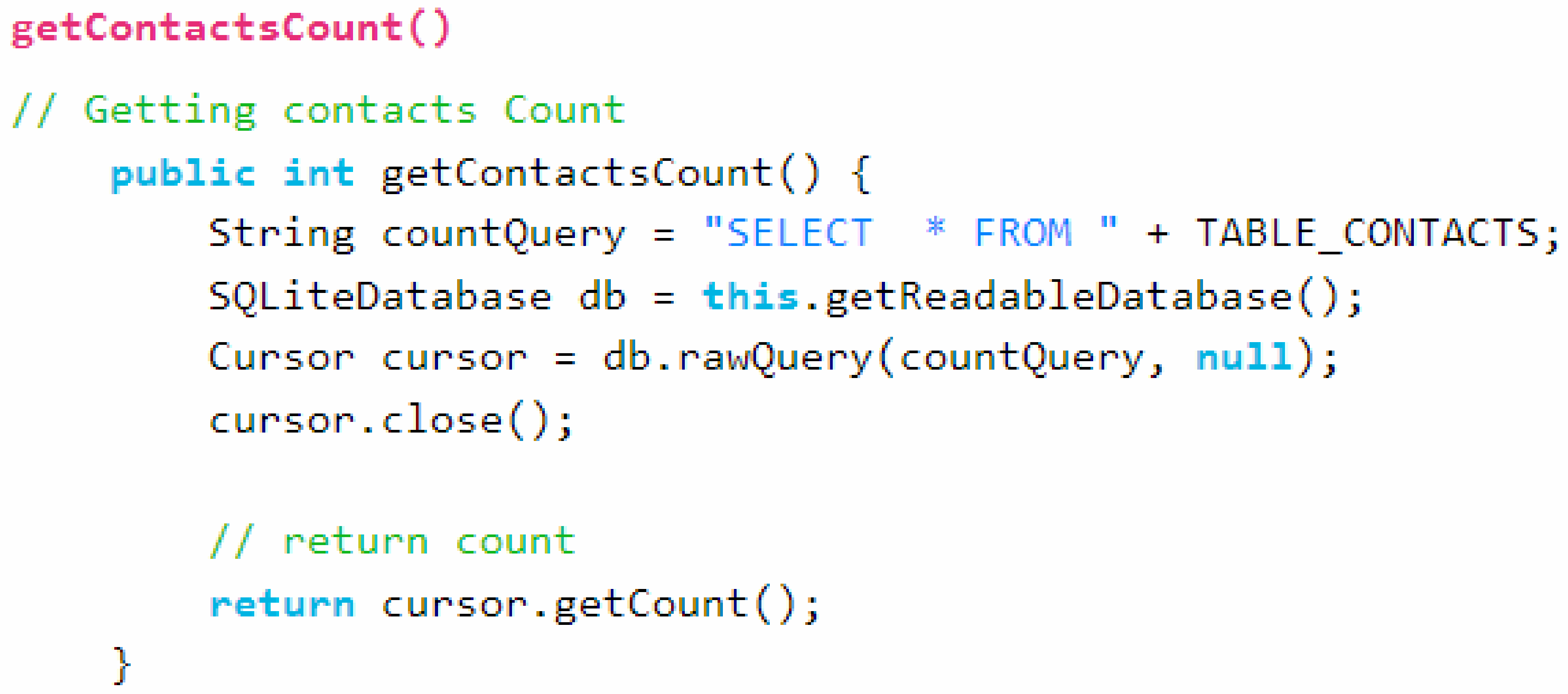
ContactsDBApp: The DatbaseHelper

getAllContacts()



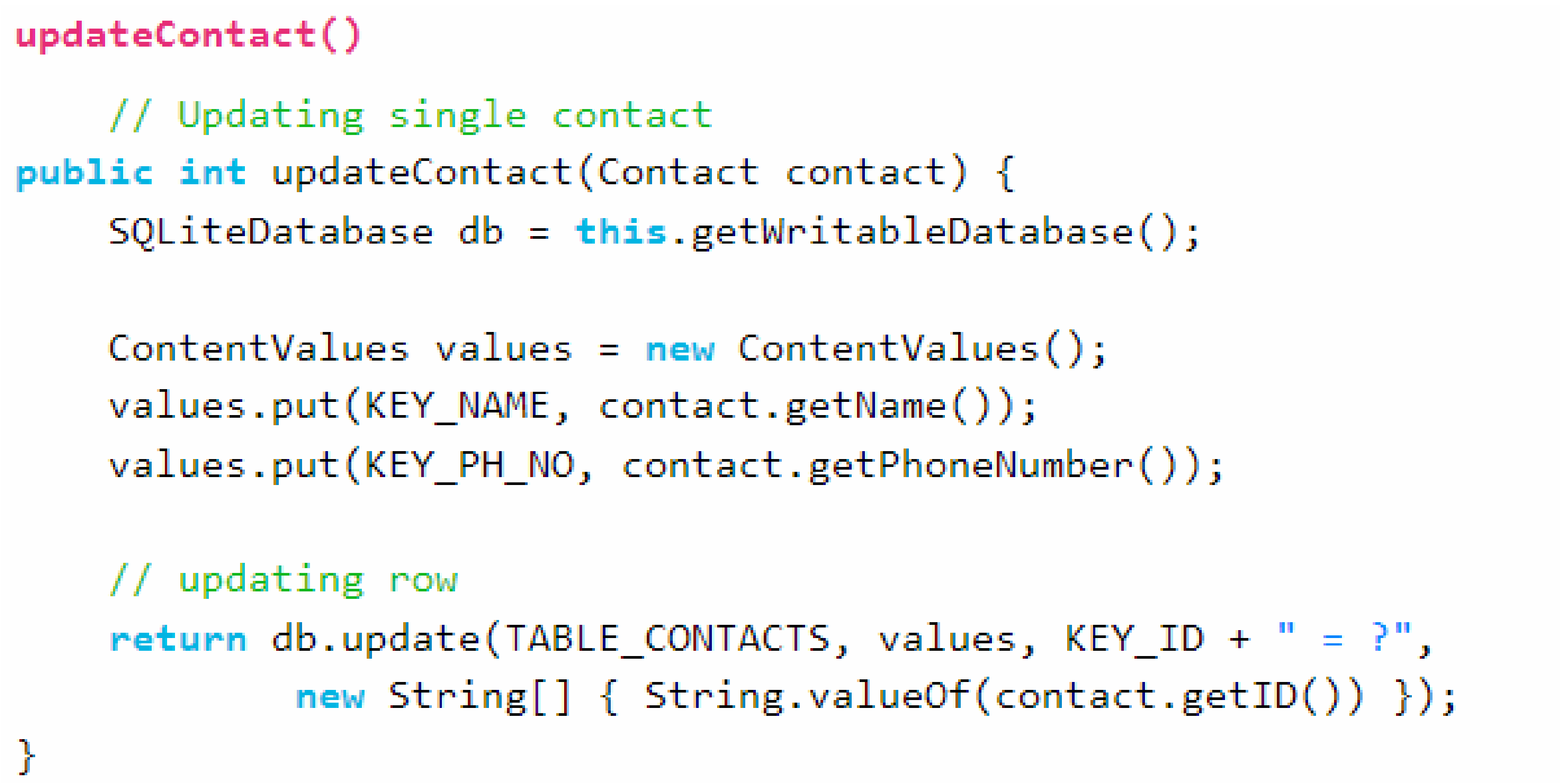
```java
    // Getting All Contacts
 public List<Contact> getAllContacts() {
    List<Contact> contactList = new ArrayList<Contact>();
    // Select All Query
    String selectQuery = "SELECT  * FROM " + TABLE_CONTACTS;
    SQLiteDatabase db = this.getWritableDatabase();
    Cursor cursor = db.rawQuery(selectQuery, null);
    // looping through all rows and adding to list
    if (cursor.moveToFirst()) {
        do {
            Contact contact = new Contact();
            contact.setID(Integer.parseInt(cursor.getString(0)));
            contact.setName(cursor.getString(1));
            contact.setPhoneNumber(cursor.getString(2));
            // Adding contact to list
            contactList.add(contact);
        } while (cursor.moveToNext());
    }
    // return contact list
    return contactList;
}
```

ContactsDBApp: The DatbaseHelper



```
getContactsCount()

// Getting contacts Count
    public int getContactsCount() {
        String countQuery = "SELECT  * FROM " + TABLE_CONTACTS;
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(countQuery, null);
        cursor.close();

        // return count
        return cursor.getCount();
    }
```

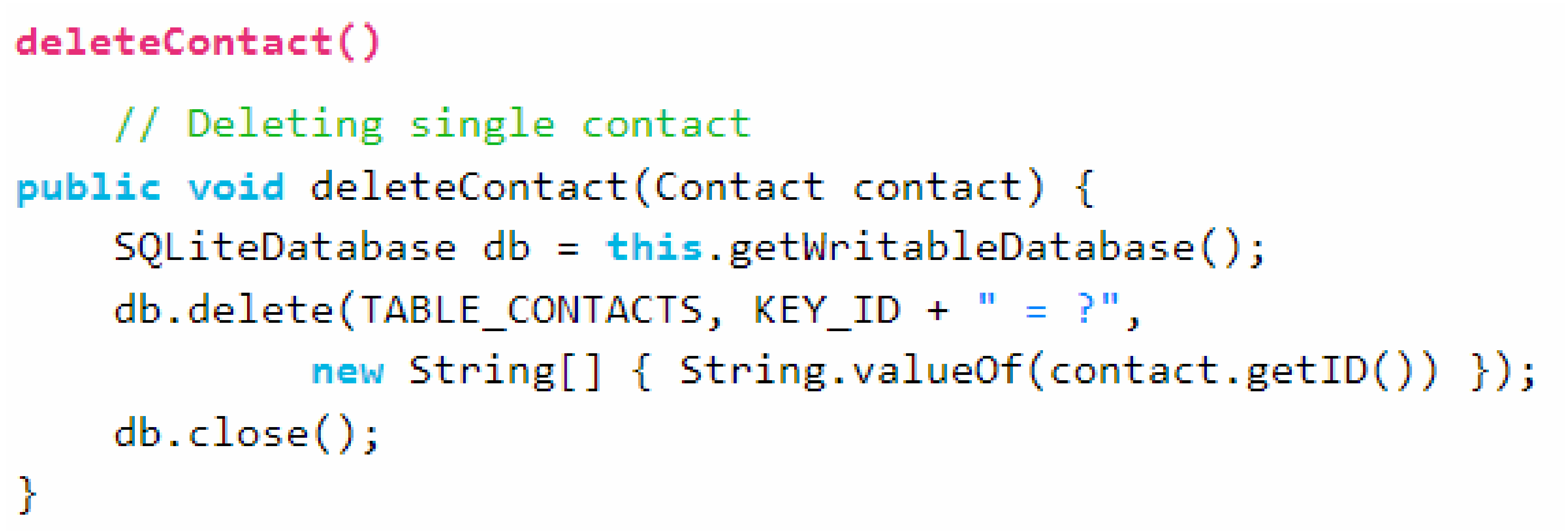
ContactsDBApp: The DatbaseHelper



updateContact()

```
    // Updating single contact
public int updateContact(Contact contact) {
    SQLiteDatabase db = this.getWritableDatabase();

    ContentValues values = new ContentValues();
    values.put(KEY_NAME, contact.getName());
    values.put(KEY_PH_NO, contact.getPhoneNumber());

    // updating row
    return db.update(TABLE_CONTACTS, values, KEY_ID + " = ?",
            new String[] { String.valueOf(contact.getID()) });
}
```

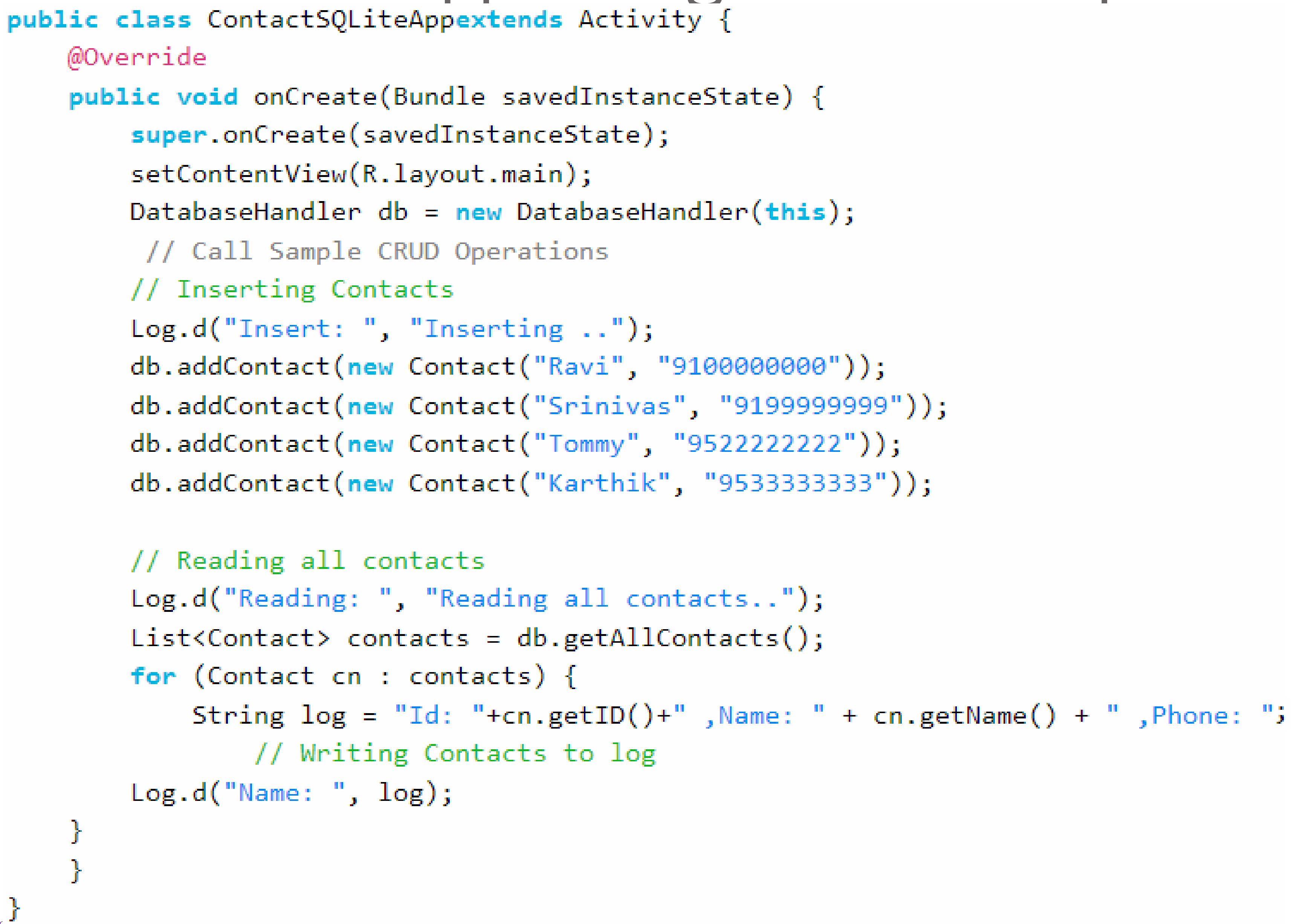
ContactsDBApp: The DatbaseHelper



deleteContact()

```
    // Deleting single contact
public void deleteContact(Contact contact) {
    SQLiteDatabase db = this.getWritableDatabase();
    db.delete(TABLE_CONTACTS, KEY_ID + " = ?",
            new String[] { String.valueOf(contact.getID()) });
    db.close();
}
```

ContactsDBApp: Using DatbaseHelper



```
public class ContactSQLiteAppextends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        DatabaseHandler db = new DatabaseHandler(this);
         // Call Sample CRUD Operations
        // Inserting Contacts
        Log.d("Insert: ", "Inserting ..");
        db.addContact(new Contact("Ravi", "9100000000"));
        db.addContact(new Contact("Srinivas", "9199999999"));
        db.addContact(new Contact("Tommy", "9522222222"));
        db.addContact(new Contact("Karthik", "9533333333"));

        // Reading all contacts
        Log.d("Reading: ", "Reading all contacts..");
        List<Contact> contacts = db.getAllContacts();
        for (Contact cn : contacts) {
            String log = "Id: "+cn.getID()+" ,Name: " + cn.getName() + " ,Phone: ";
                // Writing Contacts to log
        Log.d("Name: ", log);
    }
    }
}
```