

IAT359 Mobile Computing

Fall 2022

Instructor: Hanieh Shakeri

TA: Parnian Taghipour

lecture 6

- SQLite Databases for Android
 - Inserting data into SQLite Databases
 - Reading data from database
 - Selecting data from database
 - Linking database data to the UI

quiz 2

- Marks available on Canvas
- Quizzes can be viewed during Hanieh's office hours, or by appointment
- Quiz answers – we will review the answers today in class

week 6 check-in

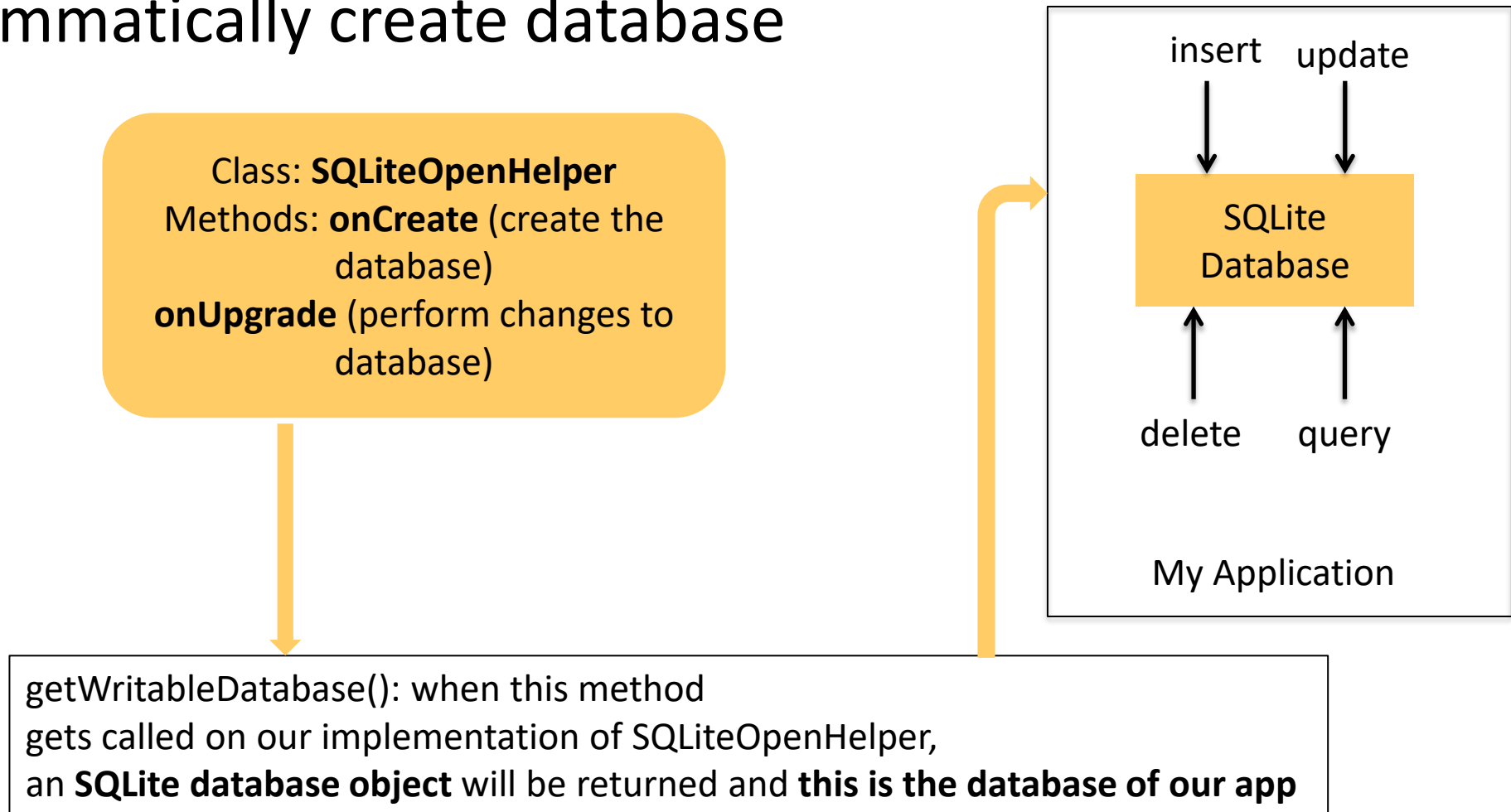
- Project Milestone 1 due TONIGHT
- Meeting with teams today during labs
- Assignment 2 due on October 25
- Start working on Milestone 2 early!
- Anonymous feedback quiz – thank you for your feedback!

SQLite databases for Android

- **SQLite**: language for managing data in relational databases
- Apps have their private SQLite databases
 - The database of an app is accessible by any class in the app, but not by other apps
- */data/data/<package-name>/database folder*
- App database
 - Self-contained
 - Transaction-based
 - No server

process

- Programmatically create database



creating an SQLite database schema

- Programmatically create database

1. Define schema – database name, version, table names, column names



2. Create the database – queries to create the database



3. Execute queries – insert, update, delete operations

step 1: define schema

Plant Database

PLANTSTABLE

_id	Name
1	Lavender
2	Rosemary
3	Hibiscus

Notes:

- **_id** is the primary key
- It identifies a row uniquely
- underscore: Android convention for primary keys

```
private static final String DATABASE_NAME = "plantdatabase";
```

```
private static final String TABLE_NAME = "PLANTSTABLE";
```

```
private static final String UID = "_id";
```

```
private static final String NAME = "Name";
```

```
private static final int DATABASE_VERSION = 1;
```

Constant
fields, do not
change values
within our app

Step 2: create the database – SQLiteOpenHelper

- Subclass of SQLiteOpenHelper
 - Implement onCreate() and onUpgrade()
- SQLiteOpenHelper class:
 - Opens the database if it exists
 - Creates the database if it does not exist
 - Updates the database as necessary

SQLiteOpenHelper - methods

- **onCreate()** – this method is called when the database is first created
 - Creation of tables
 - Initial data inside of tables
- **onUpgrade()** – this method is called whenever the database is updated
 - Drop, add tables
 - Anything that updates the database structure

SQLiteDatabase class

- An object representing the database that we have just created
- Method: **public void execSQL(String sql)**
 - Executes a single SQL statement
 - Notice that the method returns void – this is an indication to what type of SQL statements this method can execute

example application

- 1. Create the SQLite Database schema

```
public class HelperClass extends SQLiteOpenHelper{
    private static final String DATABASE_NAME = "plantdatabase";
    private static final String TABLE_NAME = "PLANTSTABLE";
    private static final String UID = "_id";
    private static final String NAME = "Name";
    private static final int DATABASE_VERSION = 1;

    public HelperClass (Context context){
        super (context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
}
```

onCreate() method

- Statement to create database:

_id	Name
1	Lavender
2	Rosemary
3	Hibiscus

```
CREATE TABLE PLANTSTABLE(_id INTEGER PRIMARY KEY  
AUTOINCREMENT, Name VARCHAR(255));
```

This can also be written as:

```
private static final String CREATE_TABLE =  
" CREATE TABLE " +TABLE_NAME+ " (" +UID+ " INTEGER PRIMARY KEY AUTOINCREMENT, " +NAME+ " VARCHAR(255));";
```

onCreate() method

```
public class HelperClass extends SQLiteOpenHelper{
    private Context context;
    private static final String DATABASE_NAME = "plantdatabase";
    private static final String TABLE_NAME = "PLANTSTABLE";
    private static final String UID = "_id";
    private static final String NAME = "Name";
    private static final int DATABASE_VERSION = 1;
    private static final String CREATE_TABLE =
        " CREATE TABLE " +TABLE_NAME+ " (" +UID+ " INTEGER PRIMARY KEY AUTOINCREMENT, " +NAME+ " VARCHAR(255));";

    @Override
    public void onCreate(SQLiteDatabase db) {
        try {
            db.execSQL(CREATE_TABLE);
        } catch (SQLException e) {
            Toast.makeText(context, "exception onCreate() db", Toast.LENGTH_LONG).show();
        }
    }
}
```

onUpgrade() method

- In this method we can do anything that will change the database schema
 - Modify the table
 - Delete the table
 - Add, delete columns
 - Drop table

onUpgrade() method

- Example: drop table query

```
private static final String DROP_TABLE = "DROP TABLE IF EXISTS " +TABLE_NAME;
```

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    try {
        db.execSQL(DROP_TABLE);
        onCreate(db);
    } catch (SQLException e) {
        Toast.makeText(context, "exception onUpgrade() db", Toast.LENGTH_LONG).show();
    }
}
```


sample code: main activity

- Create an object of the HelperClass
- Then, run the app to see what happens

```
public class MainActivity extends Activity {
```

```
    HelperClass helper;
```

```
@Override
```

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    helper = new HelperClass(this);  
}
```

nothing happens!!!

- Why??
- **IMPORTANT:** the database gets created only when there is an attempt to access it for the first time
 - Since there was no access to our database, the database was not created
 - This means that onCreate() and onUpgrade() methods have not been called
 - We have to call the method **getWritableDatabase()** – returns a **database object**

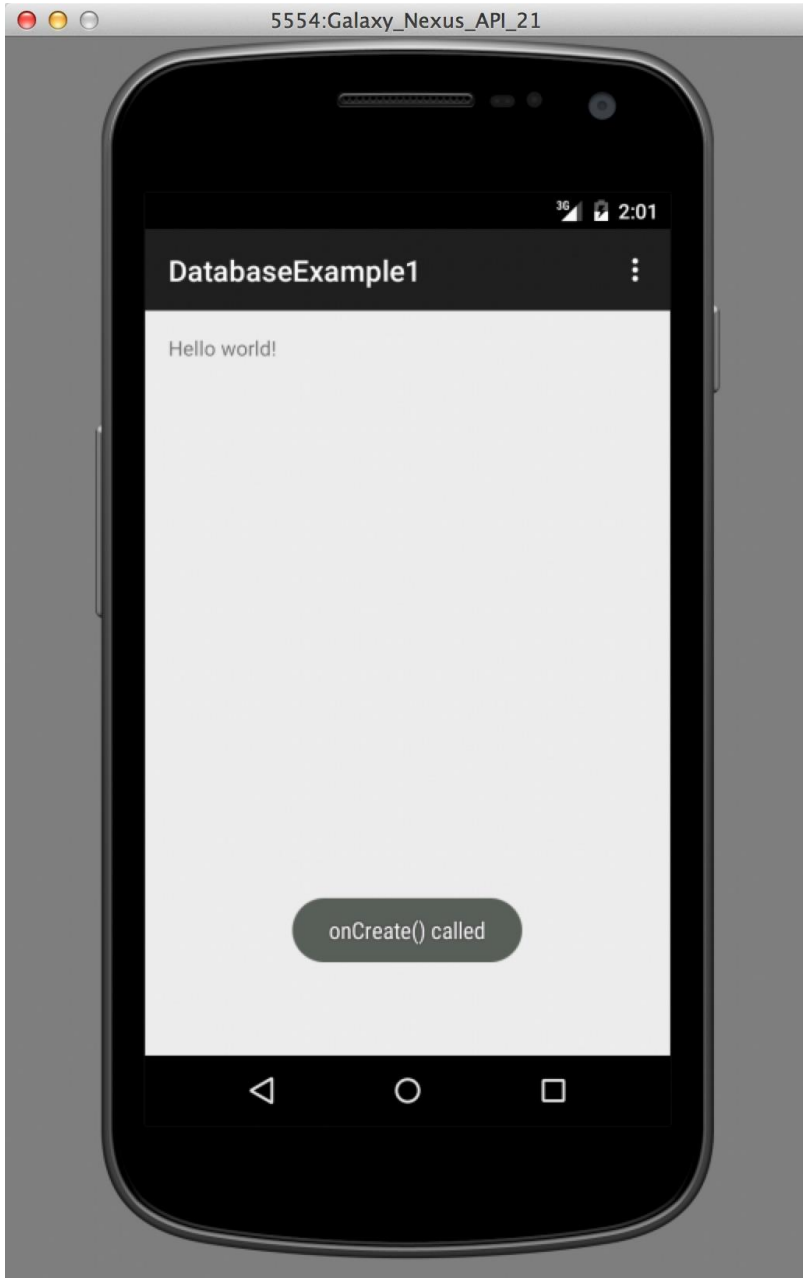
Accessing the database

```
SQLiteDatabase myDatabase = helper.getWritableDatabase();
```

getWritableDatabase() will return an SQLite database object.

The returned object is a reference to the database that we just created.

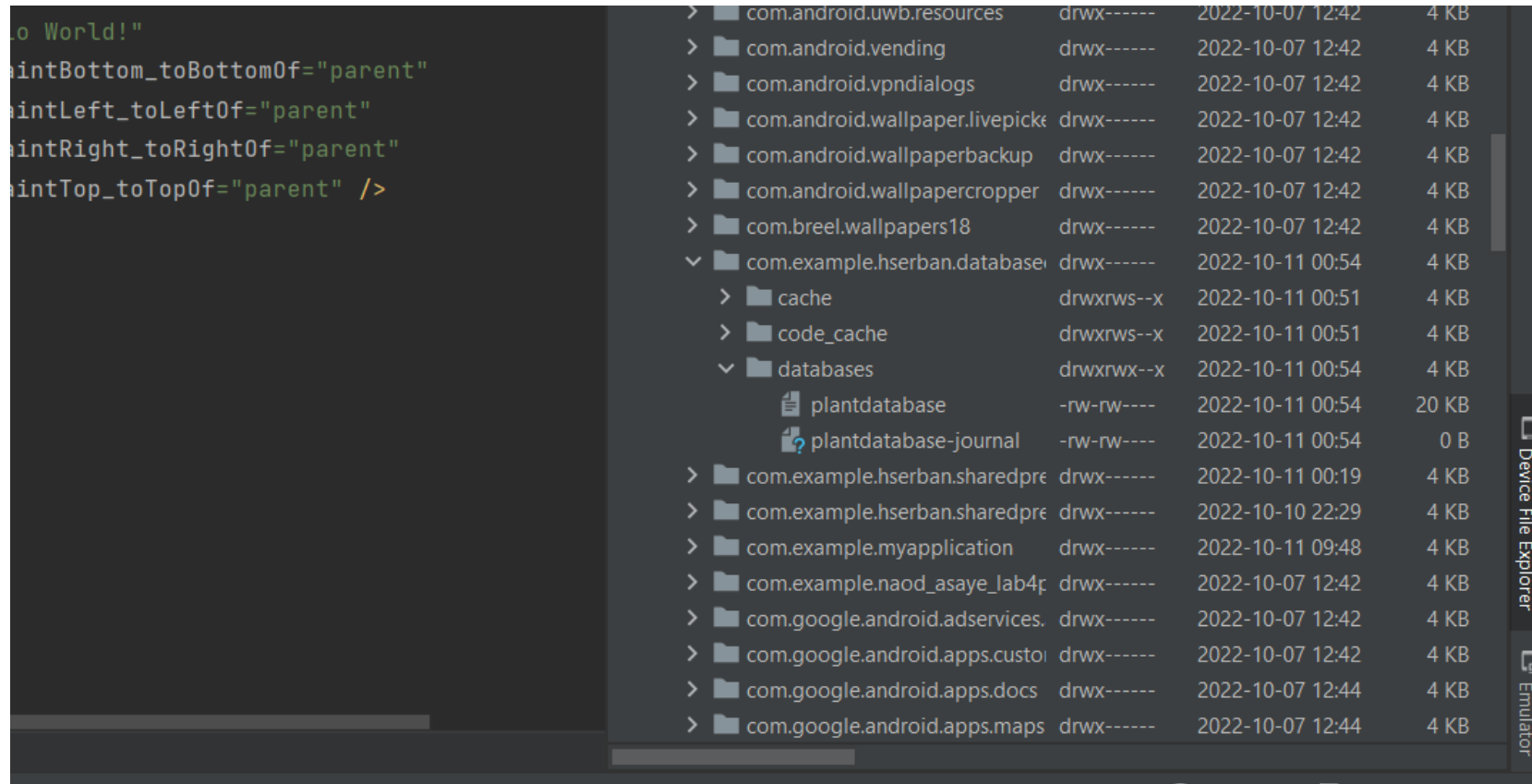
Next: run the code again



`onCreate()` called because the database was accessed for the first time and this triggered the database creation

viewing the database file

- Device File Explorer



Insert query

- Schema of our database

PLANTSTABLE

_id	Name
1	Lavender
2	Rosemary
3	Hibiscus

Notes:

- **_id** is the primary key
- It identifies a row uniquely
- underscore: Android convention for primary keys

```
private static final String DATABASE_NAME = "plantdatabase";
```

```
private static final String TABLE_NAME = "PLANTSTABLE";
```

```
private static final String UID = "_id";
```

```
private static final String NAME = "Name";
```

```
private static final int DATABASE_VERSION = 1;
```

Constant
fields, do not
change values
within our app


our database

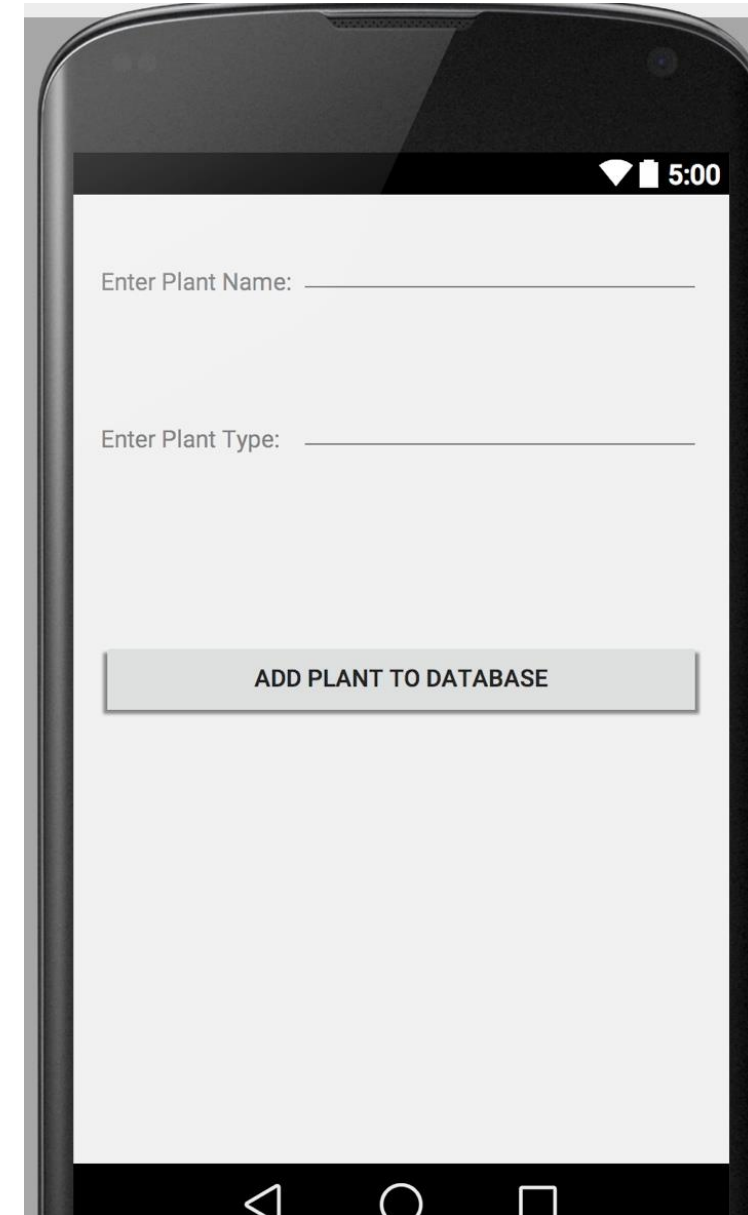
- Let's see how we can insert these records in our plant database

_id	Name	Type
1	Lavender	Perennial Herb
2	Rosemary	Evergreen Herb
3	Hibiscus	Evergreen Broadleaf
4	Purple Smoke Tree	Deciduous Broadleaf

XML layout

- User can enter the plant name and the plant type.
- When the 'Add Plant to Database' button is clicked, the method `addPlant()` is triggered:

Properties	
maxHeight	
maxLength	
maxLines	
maxWidth	
minHeight	
minLines	
minWidth	
nestedScrollingEnabled	<input type="checkbox"/>
onClick	 <code>addPlant</code>
outlineProvider	
padding	<code>[]</code>
paddingEnd	
paddingStart	
shadowColor	
singleLine	<input type="checkbox"/>



MainActivity Java code

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    plantName = (EditText)findViewById(R.id.plantNameEditText);
    plantType = (EditText)findViewById(R.id.plantTypeEditText);

    helper = new MyHelperClass(this);
}
```

```
public void addPlant (View view)
{
    String name = plantName.getText().toString();
    String type = plantType.getText().toString();
}
```

Insert query

- Create an object of class ContentValues

```
ContentValues contentValues = new ContentValues();
```

- It takes a key and a value
 - Key = column
 - Value = data that we are going to insert
- To insert an new row (plant name, plant type):

```
ContentValues contentValues = new ContentValues();  
contentValues.put(Constants.NAME, name);  
contentValues.put(Constants.TYPE, type);  
long id = db.insert(Constants.TABLE_NAME, null, contentValues);
```

Plant name

Plant type

Insert query

```
public long insert (String table, String nullColumnHack, ContentValues values)
```

Added in [API level 15](#)

Convenience method for inserting a row into the database.

Parameters

<i>table</i>	the table to insert the row into
<i>nullColumnHack</i>	optional; may be <code>null</code> . SQL doesn't allow inserting a completely empty row without naming at least one column name. If your provided <code>values</code> is empty, no column names are known and an empty row can't be inserted. If not set to null, the <code>nullColumnHack</code> parameter provides the name of nullable column name to explicitly insert a NULL into in the case where your <code>values</code> is empty.
<i>values</i>	this map contains the initial column values for the row. The keys should be the column names and the values the column values

Returns

the row ID of the newly inserted row, or -1 if an error occurred

the database code

- We want to have code that is re-usable
- Keep separate UI code and database code
- Three classes:

MyDatabase class

Contains SQLite
Database instance and
helper.

MyHelper class

Extends the
SQLiteOpenHelper
class.
Provides methods for
creating and updating
the database
(onCreate()) and
onUpgrade()

Constants class

Hold all the String
constants (table names,
column names,
etc). These constants
are used in both
MyDatabase and
MyHelper classes

Constants class

```
public class Constants {  
    public static final String DATABASE_NAME = "plantdatabase";  
    public static final String TABLE_NAME = "PLANTSTABLE";  
    public static final String UID = "_id";  
    public static final String NAME = "Name";  
    public static final String TYPE = "Type";  
    public static final int DATABASE_VERSION = 9;  
}
```

MyHelper class

```
public class MyHelper extends SQLiteOpenHelper {  
  
    private Context context;  
  
    private static final String CREATE_TABLE =  
        "CREATE TABLE "+  
        Constants.TABLE_NAME + " (" +  
        Constants.UID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +  
        Constants.NAME + " TEXT, " +  
        Constants.TYPE + " TEXT);" ;  
  
    private static final String DROP_TABLE = "DROP TABLE IF EXISTS " + Constants.TABLE_NAME;  
  
    public MyHelper(Context context){  
        super (context, Constants.DATABASE_NAME, null, Constants.DATABASE_VERSION);  
        this.context = context;  
    }  
}
```


MyHelper class - methods

```
@Override
public void onCreate(SQLiteDatabase db) {
    try {
        db.execSQL(CREATE_TABLE);
        Toast.makeText(context, "onCreate() called", Toast.LENGTH_LONG).show();
    } catch (SQLException e) {
        Toast.makeText(context, "exception onCreate() db", Toast.LENGTH_LONG).show();
    }
}
```

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    try {
        db.execSQL(DROP_TABLE);
        onCreate(db);
        Toast.makeText(context, "onUpgrade called", Toast.LENGTH_LONG).show();
    } catch (SQLException e) {
        Toast.makeText(context, "exception onUpgrade() db", Toast.LENGTH_LONG).show();
    }
}
```

MyDatabase class

- Notice the insertData() method

```
public class MyDatabase {  
    private SQLiteDatabase db;  
    private Context context;  
    private final MyHelper helper;  
  
    public MyDatabase (Context c){  
        context = c;  
        helper = new MyHelper(context);  
    }  
  
    public long insertData (String name, String type)  
    {  
        SQLiteDatabase db = helper.getWritableDatabase();  
        ContentValues contentValues = new ContentValues();  
        contentValues.put(Constants.NAME, name);  
        contentValues.put(Constants.TYPE, type);  
        long id = db.insert(Constants.TABLE_NAME, null, contentValues);  
        return id;  
    }  
}
```


MainActivity Java code

```
public class MainActivity extends ActionBarActivity {
```

```
    EditText plantName, plantType;  
    MyDatabase db;
```

```
@Override
```

```
protected void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);
```

```
    plantName = (EditText)findViewById(R.id.plantNameEditText);  
    plantType = (EditText)findViewById(R.id.plantTypeEditText);
```

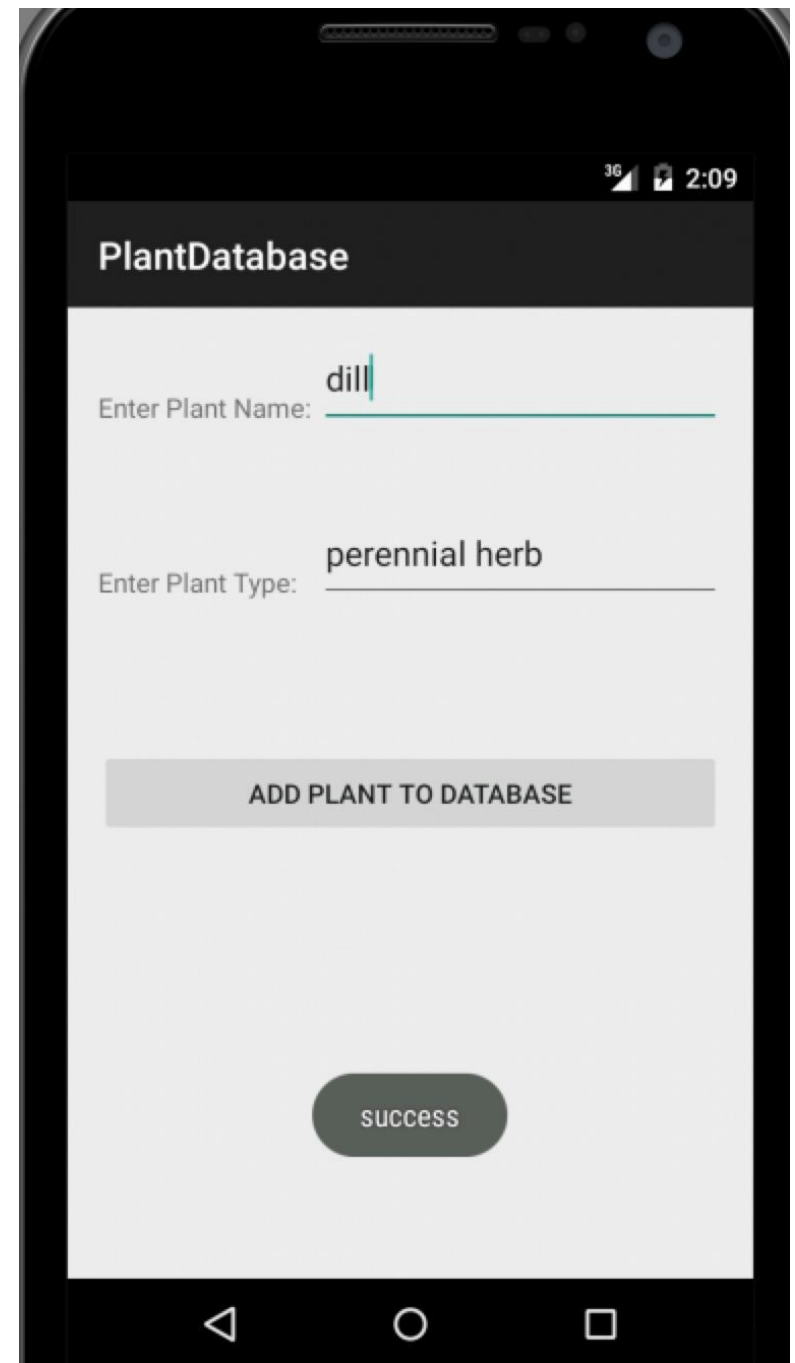
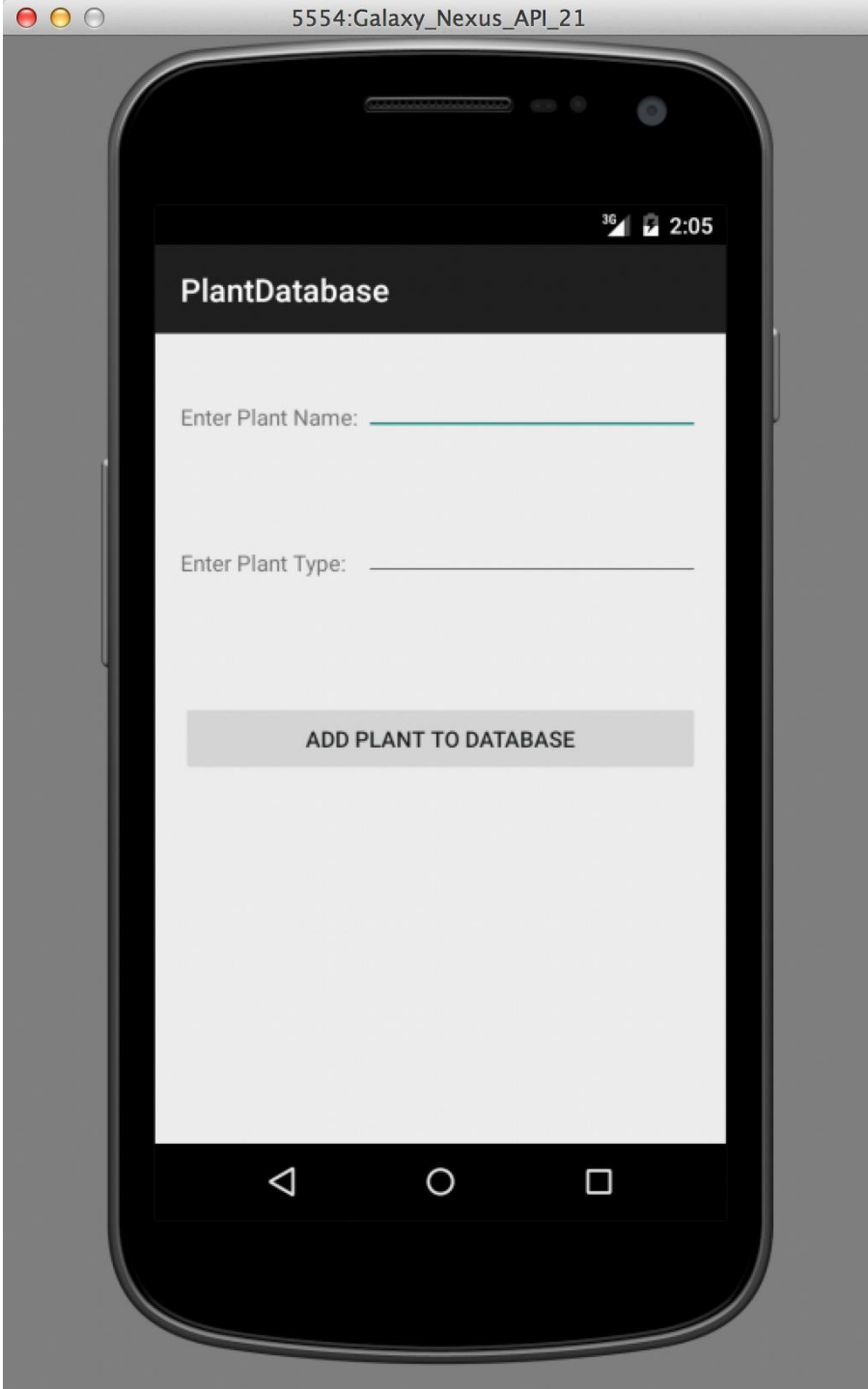
```
    db = new MyDatabase(this);
```

```
}
```

MainActivity – button click

```
public void addPlant (View view)
{
    String name = plantName.getText().toString();
    String type = plantType.getText().toString();
    Toast.makeText(this, name + type, Toast.LENGTH_SHORT).show();
    long id = db.insertData(name, type);
    if (id < 0)
    {
        Toast.makeText(this, "fail", Toast.LENGTH_SHORT).show();
    }
    else
    {
        Toast.makeText(this, "success", Toast.LENGTH_SHORT).show();
    }
}
```

result



read data from database

- Use the **query()** method
- To this method we pass the **selection criteria** and **the desired columns**
- Results from the query are returned in a **Cursor** object

query() method

```
public Cursor query (boolean distinct, String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)
```

Added in [API level](#)

Query the given URL, returning a [Cursor](#) over the result set.

Parameters

<i>distinct</i>	true if you want each row to be unique, false otherwise.
<i>table</i>	The table name to compile the query against.
<i>columns</i>	A list of which columns to return. Passing null will return all columns, which is discouraged to prevent reading data from storage that isn't going to be used.
<i>selection</i>	A filter declaring which rows to return, formatted as an SQL WHERE clause (excluding the WHERE itself). Passing null will return all rows for the given table.
<i>selectionArgs</i>	You may include ?s in selection, which will be replaced by the values from selectionArgs, in order that they appear in the selection. The values will be bound as Strings.
<i>groupBy</i>	A filter declaring how to group rows, formatted as an SQL GROUP BY clause (excluding the GROUP BY itself). Passing null will cause the rows to not be grouped.
<i>having</i>	A filter declare which row groups to include in the cursor, if row grouping is being used, formatted as an SQL HAVING clause (excluding the HAVING itself). Passing null will cause all row groups to be included, and is required when row grouping is not being used.
<i>orderBy</i>	How to order the rows, formatted as an SQL ORDER BY clause (excluding the ORDER BY itself). Passing null will use the default sort order, which may be unordered.
<i>limit</i>	Limits the number of rows returned by the query, formatted as LIMIT clause. Passing null denotes no LIMIT clause.

Returns

A [Cursor](#) object, which is positioned before the first entry. Note that [Cursors](#) are not synchronized, see the documentation for more details.

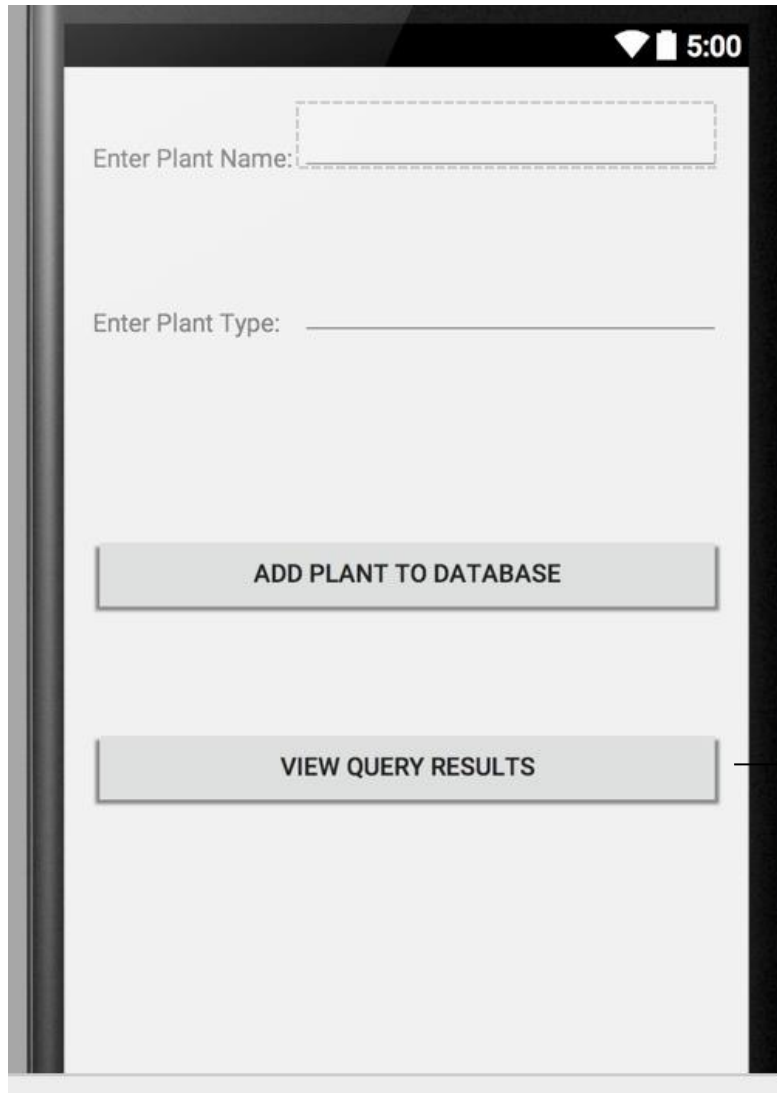
Cursor object

- android.database.Cursor
- Gives us access to the results returned from a database query
 - E.g., results: entire column, subset of table, etc
- Cursor allows navigation through the result set

relevant methods for Cursor

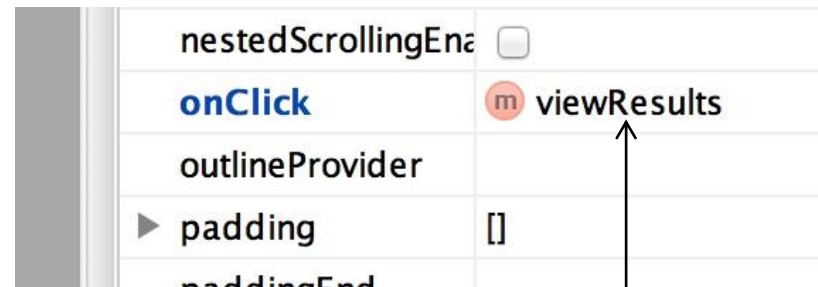
- `String getColumnName (int columnIndex)`
- `int getCount()` – how many rows were returned in the result
 - We can use this method to test for empty result
- `boolean moveToNext()` – navigation through rows of result set
- ... and others. see online android documentation

UI – another button



In MainActivity java code:

```
public void viewResults(View view)
{
}
}
```



MyDatabase class

Add a new method to retrieve data from the database:

```
public String getData()
{
    SQLiteDatabase db = helper.getWritableDatabase();
    String[] columns = {Constants.UID, Constants.NAME, Constants.TYPE};
    Cursor cursor = db.query(Constants.TABLE_NAME, columns, null, null, null, null, null);

    StringBuffer buffer = new StringBuffer();

    while (cursor.moveToNext()) {
        int index = cursor.getInt(0);
        String name = cursor.getString(1);
        String type = cursor.getString(2);
        buffer.append(index + " " + name + " " + type + "\n");
    }
    return buffer.toString();
}
```

In MainActivity Java

- In the ViewResults() method:

```
public void viewResults(View view)
{
    String data = db.getData();
    Toast.makeText(this, data, Toast.LENGTH_LONG).show();
}
```

final outcome

PlantDatabase

Enter Plant Name:

Enter Plant Type:

ADD PLANT TO DATABASE

- 1 parsley herb
- 2 lavender perennial
- 3 cherry tree fruit tree
- 4 tulip flower
- 5 potato plant vegetable

1 parsley herb
2 lavender perennial
3 cherry tree fruit tree
4 tulip flower
5 potato plant vegetable

select query with condition

- For example, select and display all plants from the database that have the type 'herb'
- Format:
- **Constants.TYPE = 'herb'**
- In code:

```
String selection = Constants.TYPE + "='" + type + "'"; //Constants.TYPE = 'type'  
Cursor cursor = db.query(Constants.TABLE_NAME, columns, selection, null, null, null, null);
```

MyDatabase class – add a new method for query with condition

```
public String getSelectedData(String type)
{
    //select plants from database of type 'herb'
    SQLiteDatabase db = helper.getWritableDatabase();
    String[] columns = {Constants.NAME, Constants.TYPE};

    String selection = Constants.TYPE + "='" + type + "'"; //Constants.TYPE = 'type'
    Cursor cursor = db.query(Constants.TABLE_NAME, columns, selection, null, null, null, null);

    StringBuffer buffer = new StringBuffer();
    while (cursor.moveToNext()) {

        int index1 = cursor.getColumnIndex(Constants.NAME);
        int index2 = cursor.getColumnIndex(Constants.TYPE);
        String plantName = cursor.getString(index1);
        String plantType = cursor.getString(index2);
        buffer.append(plantName + " " + plantType + "\n");
    }
    return buffer.toString();
}
```

UI: new button for condition query

- Added a new TextView and EditText, to give the user the ability to input the 'type' for which they want to perform a query



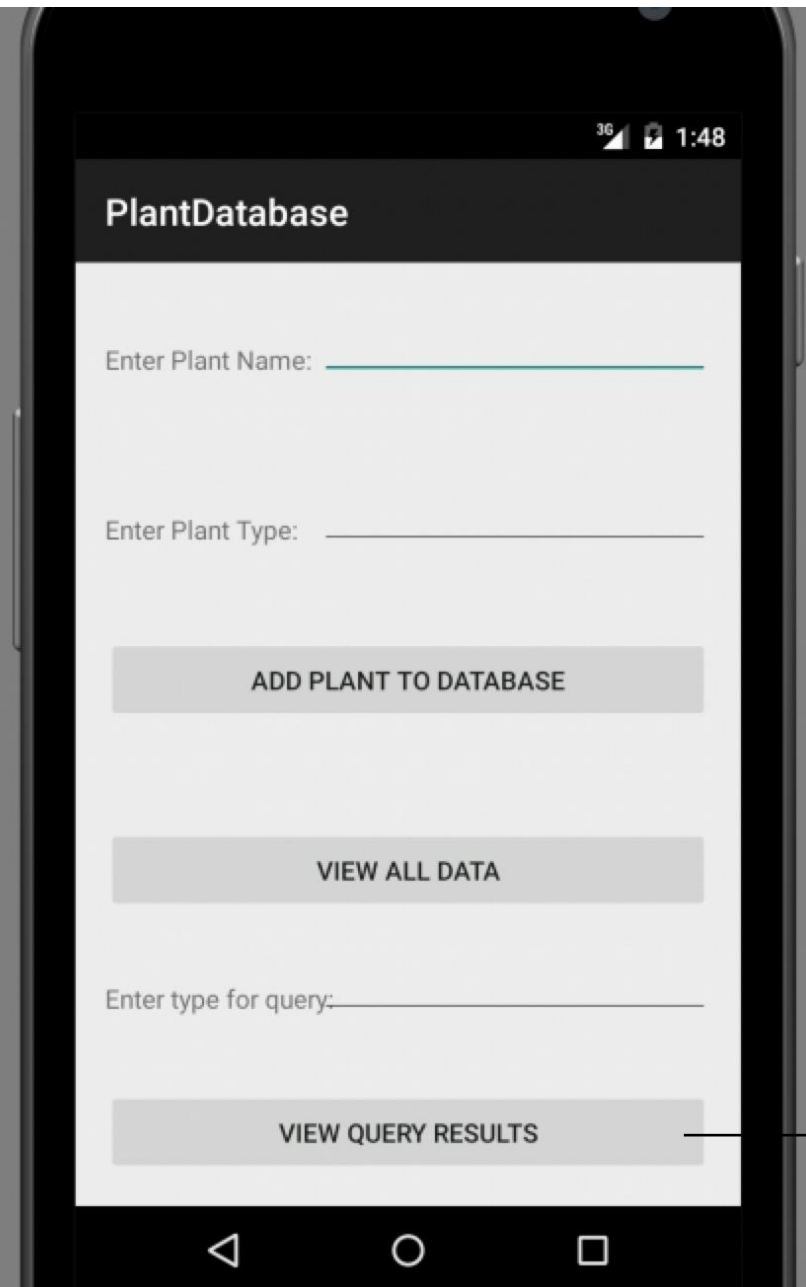
Enter type for query:_____

- New Button – when clicked, the query results will be displayed in a Toast message



VIEW QUERY RESULTS

UI



UI - MainActivity

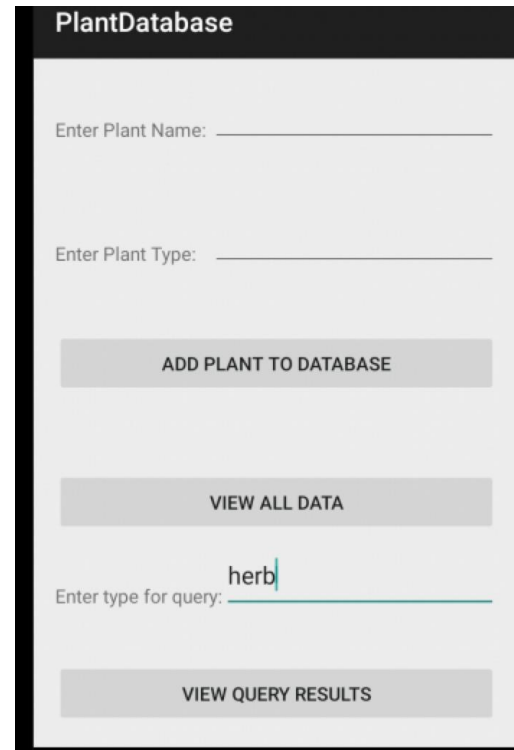
- Method for button click:

```
public void viewQueryResults (View view)
{
    String userInputType = selectType.getText().toString();
    String queryResults = db.getSelectedData(userInputType);
    Toast.makeText(this, queryResults, Toast.LENGTH_LONG).show();
}
```


final outcome



All data



Selection condition



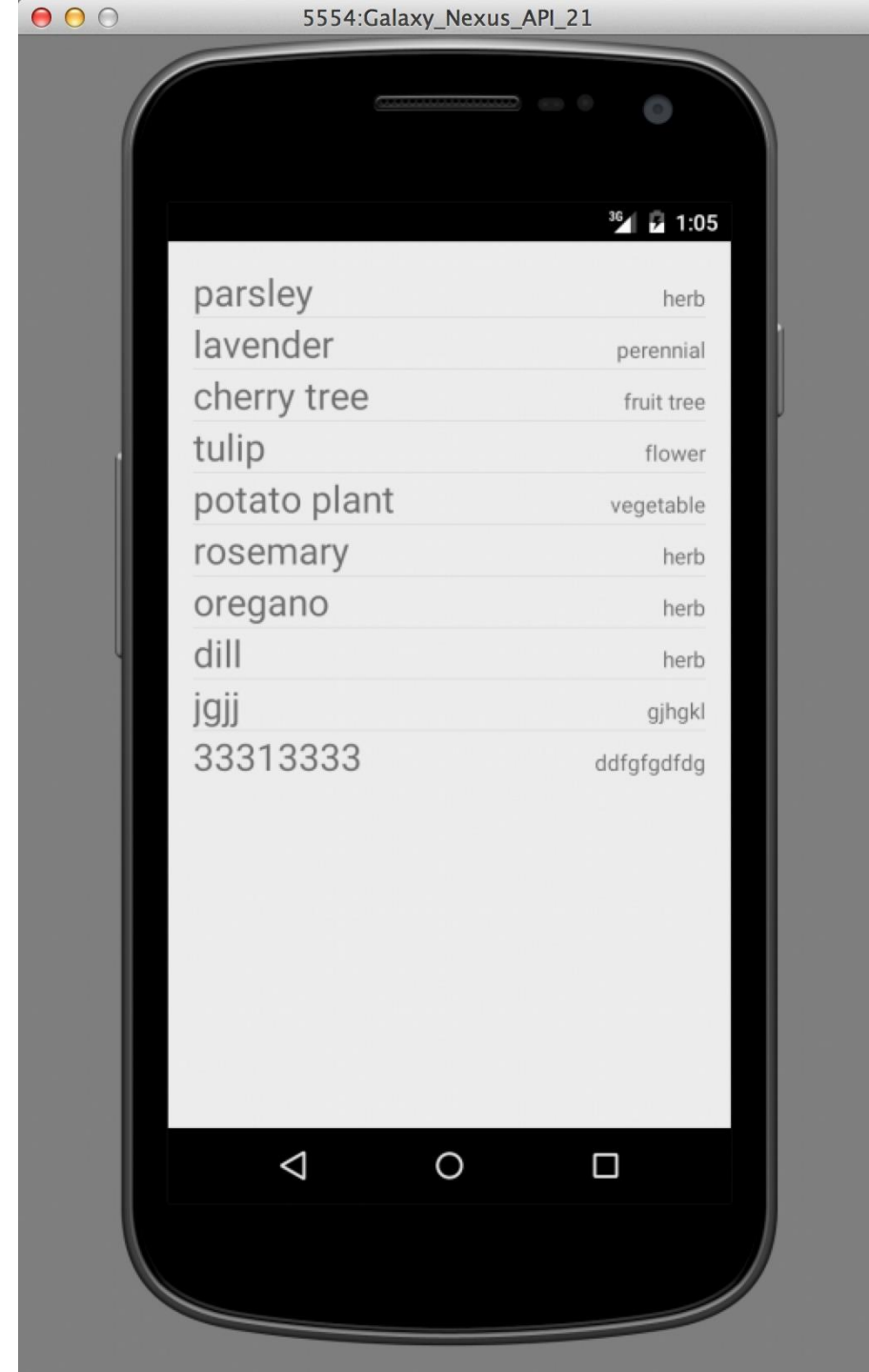
Query result

linking database data to the UI

- Suppose we want to display the results from our database query in a RecyclerView
- To do this, we will create a new activity:
 - **RecyclerViewActivity**

outcome

- View 'All Data' query



summary of today's class

- Inserting data into database: INSERT
- Reading data from database:
 - Query method
 - Cursor Object
- Selecting data from database:
 - SELECT query with condition
- Linking database data to UI
 - Displaying into RecyclerView

resources

- Put Information into Database:
<https://developer.android.com/training/data-storage/sqlite#WriteDbRow>
- Read Information from Database:
<https://developer.android.com/training/data-storage/sqlite#ReadDbRow>
- Cursor:
<http://developer.android.com/reference/android/database/Cursor.html>