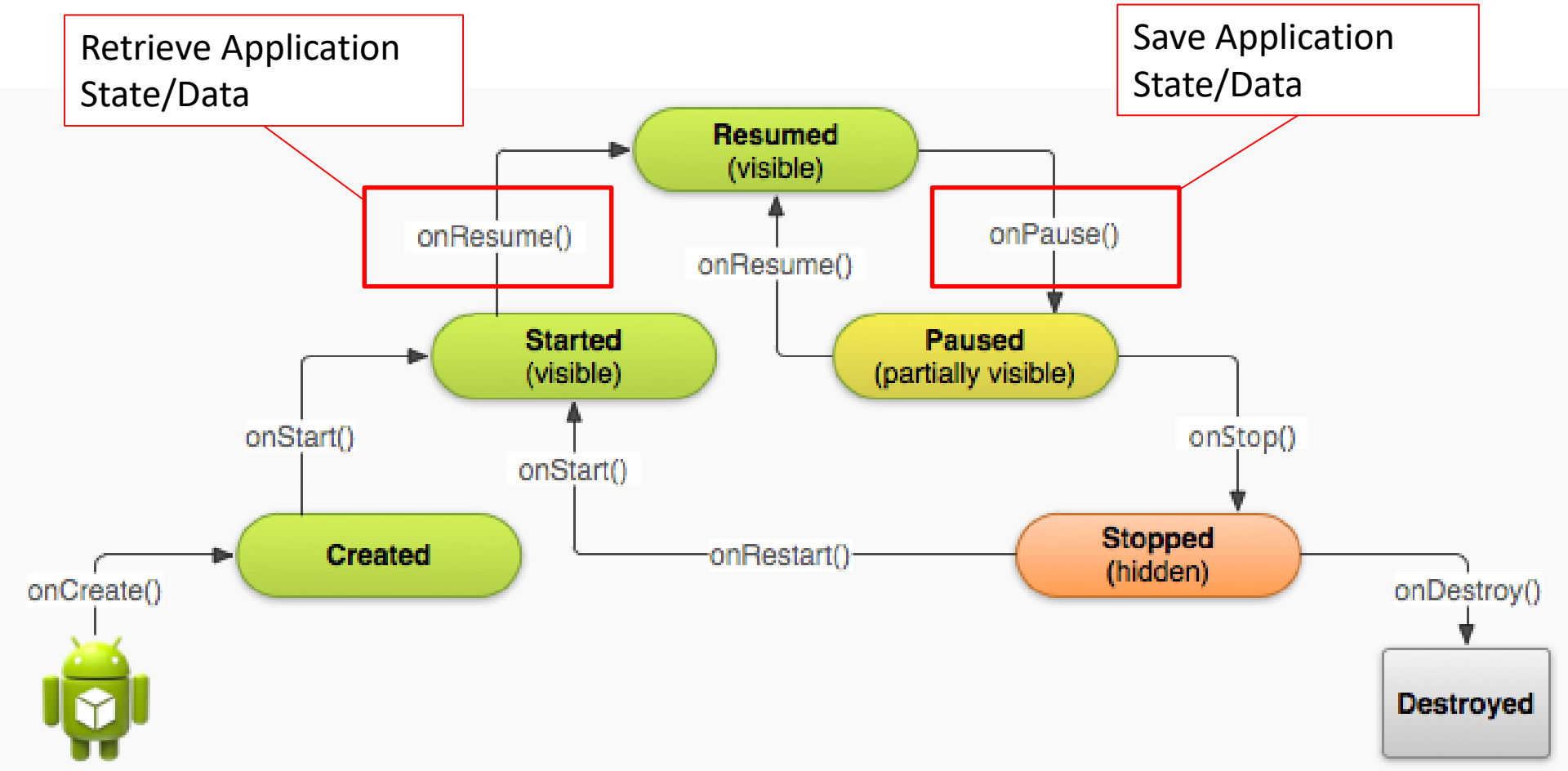
SharedPreferences and SQLite

Session 1

When to Save/Retrieve Data? Scenario 1



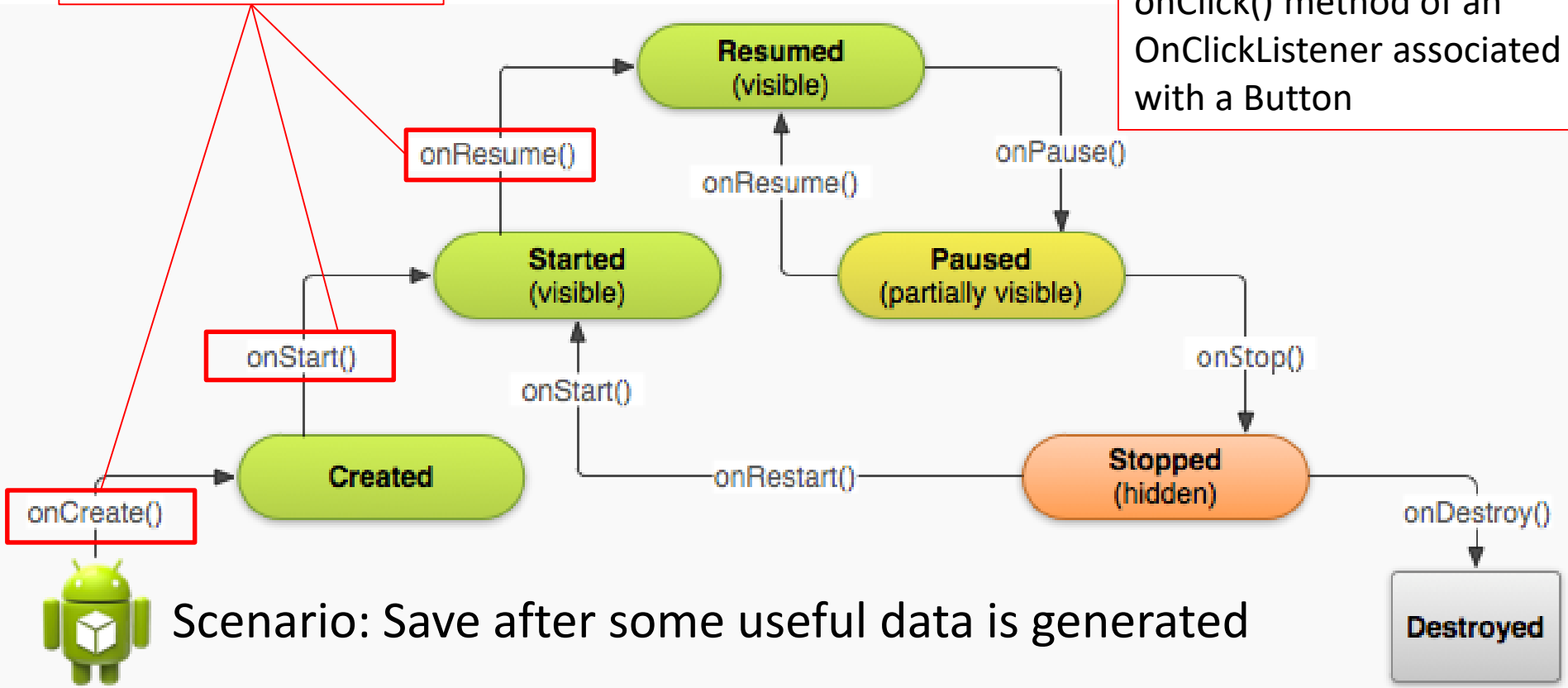
Scenario: Save when the Activity is about to go background, like there is an incoming call

When to Save/Retrieve Data? Scenario 2

Save Application State/Data

Retrieve Application State/Data

onClick() method of an OnClickListener associated with a Button



Scenario: Save after some useful data is generated

Exercise 1

- Refer to worksheet Section A: When to Save
- Follow the instruction to display a Toast every time onResume() is called

Exercise 1

- What happened when the app is terminated and started again?
  - Message “No greetings!” is displayed again.
  - In this process, the app went through states “Stopped” -> “Started” -> “Resumed”, and the “onResume()” method was called again.

Data Persistence (Saving Data)

- Android provides a few methods of Data Persistence:
  - SharedPreferences
  - Internal Storage
  - External Storage
  - SQLite Database
  - Network Connection (E.g. Web Services)

What is SharedPreferences?

- **SharedPreferences** allows you to save and retrieve persistent *key-value pairs* of *primitive* data types like boolean, float, int and String.
- It is used to save a relatively *small* collection of key-value pairs, not limited to user preferences only.
- This data will persist across user sessions (*even if your app is killed*).

Saving to SharedPreferences

- **getPreferences(int mode)** returns an instance
- To save:
  1. Call **edit()** to get a **SharedPreferences.Editor**
  2. Add values using methods like **putBoolean(String key, boolean value)** or **putString(String key, String value)**
  3. Call **commit()** to save

Saving to SharedPreferences

```
@Override
protected void onPause() {
    super.onPause();

    // Retrieve the data from the UI elements
    String strName = etName.getText().toString();

    // Step 1: Obtain the SharedPreferences instance
    SharedPreferences prefs = getPreferences(MODE_PRIVATE);

    // Step 2: Create a SharedPreferences Editor by calling edit()
    SharedPreferences.Editor prefEdit = prefs.edit();

    // step 3: Set a key-value pair in the editor
    prefEdit.putString("name", strName);

    // Step 4: Call commit() to save the changes made to the SharedPreferences
    prefEdit.commit();
}
```

Retrieving from SharedPreferences

- **gePreferences(int mode)** returns an instance
- To retrieve:
  1. Use SharedPreferences instance methods like **getBoolean(String key, boolean defaultValue)** or **getString(String key, String defaultValue)**
  2. The methods return the preference value if the value exists or the default value if no such value exists.

Retrieving from SharedPreferences

```
protected void onResume() {
    super.onResume();

    // Step 1: Obtain the SharedPreferences instance
    SharedPreferences prefs = getPreferences(MODE_PRIVATE);

    // Step 2: Retrieve the saved data from the SharedPreferences
    //  with a default value if no matching data
    String strName = prefs.getString("name", "Default name");

    // Step 3: Update the UI element with the retrieved data
    etName.setText(strName);
}
```

Inside SharedPreferences File

- Key-Value pair
  - One Key is mapped to one Value
  - Different primitive data types: float, String, boolean, etc

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<map>
  <float name="gpa" value="3.5" />
  <string name="colour">red</string>
  <string name="gender">Male</string>
  <boolean name="like" value="true" />
  <string name="name">Joshua</string>
</map>
```

Exercise 2

- Refer to worksheet Section B: Saving and Retrieving Data in the App
- Follow the instruction to save to SharedPreferences in onPause()
- Retrieve from SharedPreferences in onResume()

Exercise 2

- Why do we save to SharedPreferences in onPause()?
  - We want to save the data when the app is no longer visible. From “visible” (Resumed) to “partially visible” (Paused) or to “hidden” (Stopped), the “onPause()” method is called.

Exercise 2

- What would the following code return?

```
prefs.getString("greetings", "No greetings")
```

  - The value of the key "greetings" would be returned but if there is no such key (like when the app is first run), the default value of "No greetings" would be return

Exercise 3

- Refer to worksheet Section C
- Create the EditText for name and GPA such that their last value would be shown when the app is open

Exercise 3

- Save in onPause()

```
// Get user input from EditText and store in a variable
String strName = etName.getText().toString();
float gpa= Float.parseFloat(etGPA.getText().toString());

// Obtain an instance of the SharedPreferences
SharedPreferences prefs = getPreferences(MODE_PRIVATE);

// Obtain an instance of SharedPreference Editor for update later
SharedPreferences.Editor prefEdit = prefs.edit();

// Add the key-value pair
prefEdit.putString("name", strName);
prefEdit.putFloat("gpa", gpa);

// Call commit() to save the changes into SharedPreferences
prefEdit.commit();
```

Exercise 3

- Populate EditText in onResume()

```
// Obtain an instance of the SharedPreferences
SharedPreferences prefs = getPreferences(MODE_PRIVATE);

// Retrieve saved data from SharedPreferences
String strName = prefs.getString("name", "John");
float gpa = prefs.getFloat("gpa", 0);

// Update the UI element with the value
etName.setText(strName);
etGPA.setText(gpa + "");
```

Learning Objectives

- Use SharedPreferences to store preferences
- Retrieve from SharedPreferences and populate UI elements with values
- Understand the Activity Lifecycle
  - Various places to save/retrieve data

Data Persistence

- Shared Preferences
  - Small amount of private primitive data in key-value pairs.
- Internal Storage
  - Private data on the device memory.
- External Storage
  - Public data on the shared external storage.
- **SQLite Databases**
  - Structured data in a private database.
- Network Connection
  - Store and retrieve data on the web with your own network server.

Data modeling

Java Data Structure

- In the Android App, this may be a data structure used to track tasks added and displayed

Task
-_id: int -description: String -date: String
+Task(_id: int, description: String, date: String) +getID(): int +getDescription(): String +getDate(): String

Database Structure

- To make the app more meaningful, the user created data needs to be persisted
- The data may be stored in the following manner

_id	description	date
1	Shopping	20 May 2016
2	Partying	31 May 2016

Note for this example:

Assume simple data, no complex relationship

Complex data structure may need multiple tables

SQLite Datatypes

- Supported datatypes
  - **`INTEGER`** positive & negative whole number
  - **`TEXT`** a string of anything, enclosed with single quote ' in SQL stmt
  - **`REAL`** positive & negative number with decimal points
  - **`BLOB`** binary large object (binary files, etc)

- Candidates for `INTEGER`
  - Number of siblings – 2, Number of seats – 28
- Candidates for `TEXT`
  - Address – 'Blk 100 Woodlands Ave 1 #10-20 Singapore 123100'
  - Content of book - 'Once upon a time, there was a beautiful princess … '
- Candidates for `REAL`
  - Selling price like 2.35 or height 180

SQL – Data Definition Language

- Table creation (usually done during the initialization of database)
- Syntax:

```
CREATE TABLE tablename (_id INTEGER primary key autoincrement,
        column2 datatype [not null],
        column3 datatype [not null])
```

- Example:

```
CREATE TABLE users (_id integer primary key autoincrement,
        name TEXT not null,
        gender TEXT not null,
        heightcm INTEGER)
```

Not null
- Indicates compulsory column

- For unique identity like employee id, a suggestion is to use

"_id INTEGER primary key autoincrement"

Column Name

Datatype

Unique

Auto fill column with running number

SQL – Data Definition Language

- Table deletion (not common, but may be done in database upgrade)
- Syntax:

```
DROP TABLE [IF EXISTS] tablename
```

- Example:

```
DROP TABLE IF EXISTS movies
```

If Exists,
- checks if table exists
- a good practice

- Proceed with caution, can't be undone once dropped

Exercise 4

- Refer WS, Section D

Task
-_id: int -description: String -date: String
+Task(_id: int, description: String, date: String) +getID(): int +getDescription(): String +getDate(): String

- Write the SQL statement to create a table to store the data structure shown above

Exercise 4 (ans)

- Refer WS, Section D

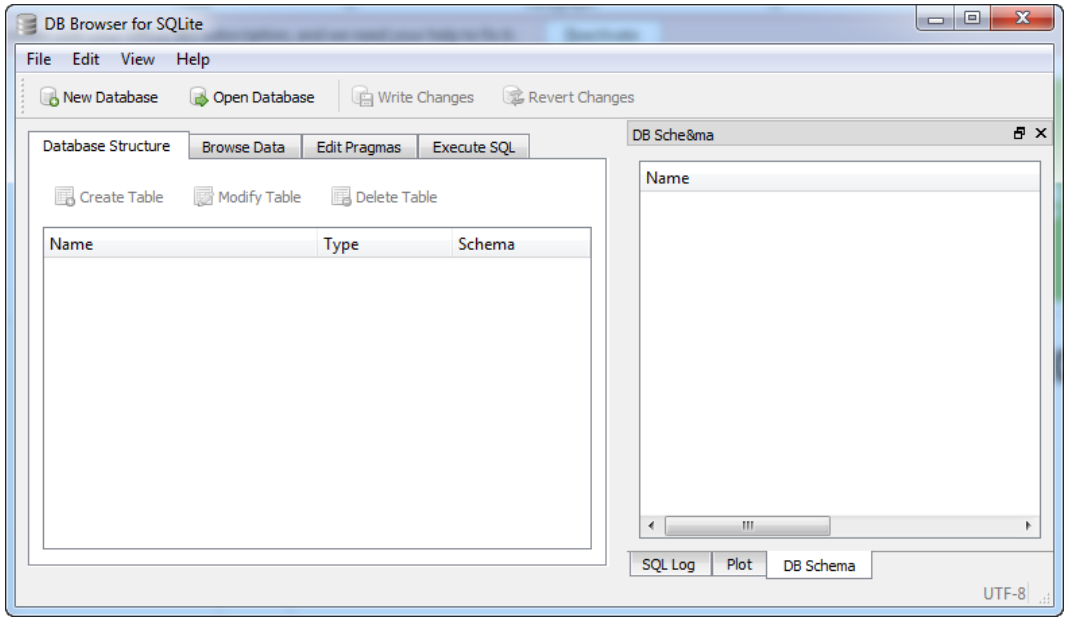
Task
-_id: int -description: String -date: String
+Task(_id: int, description: String, date: String) +getID(): int +getDescription(): String +getDate(): String

- Write the SQL statement to create a table to store the data structure shown above

```
CREATE TABLE `Task` (
                `_id`  INTEGER PRIMARY KEY AUTOINCREMENT,
                `description`  TEXT,
                `date`  TEXT
        );
```

Creating the database

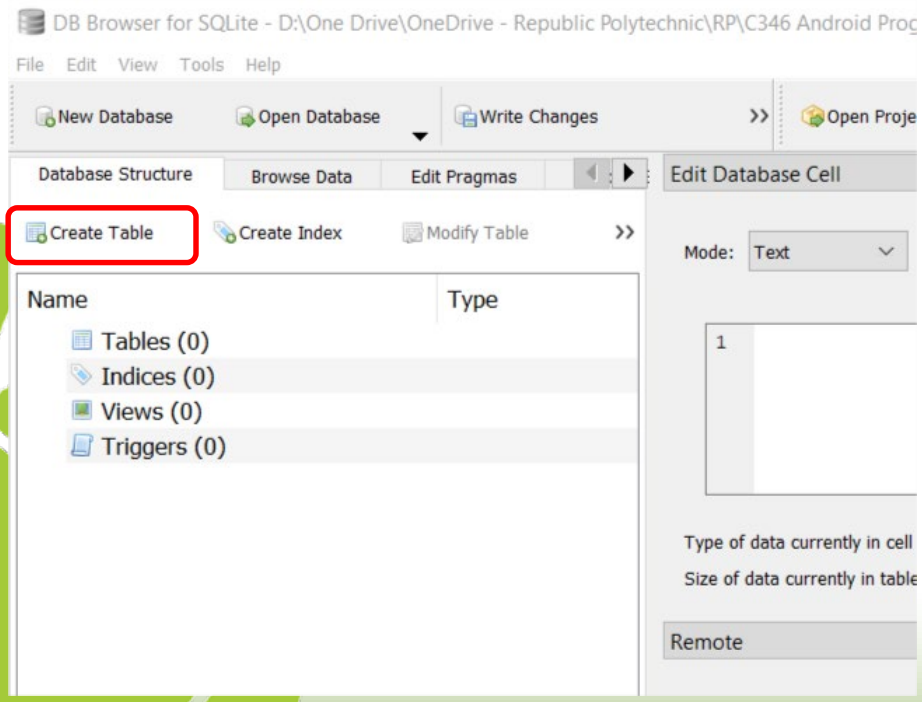
- We will use a SQLite tool to do this
- One example is “DB Browser for SQLite”
- It provides a GUI to manage the SQLite database

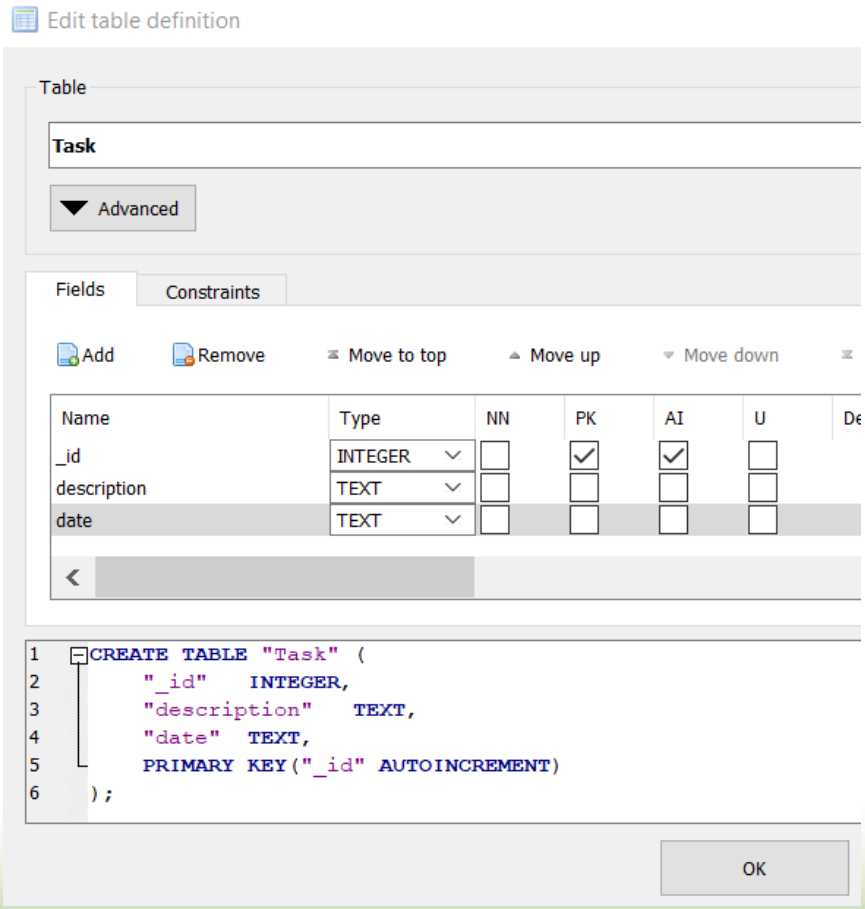


- Get PortableApp for Windows at https://sqlitebrowser.org/blog/version-3-12-2-released/

Creating the database (con't)

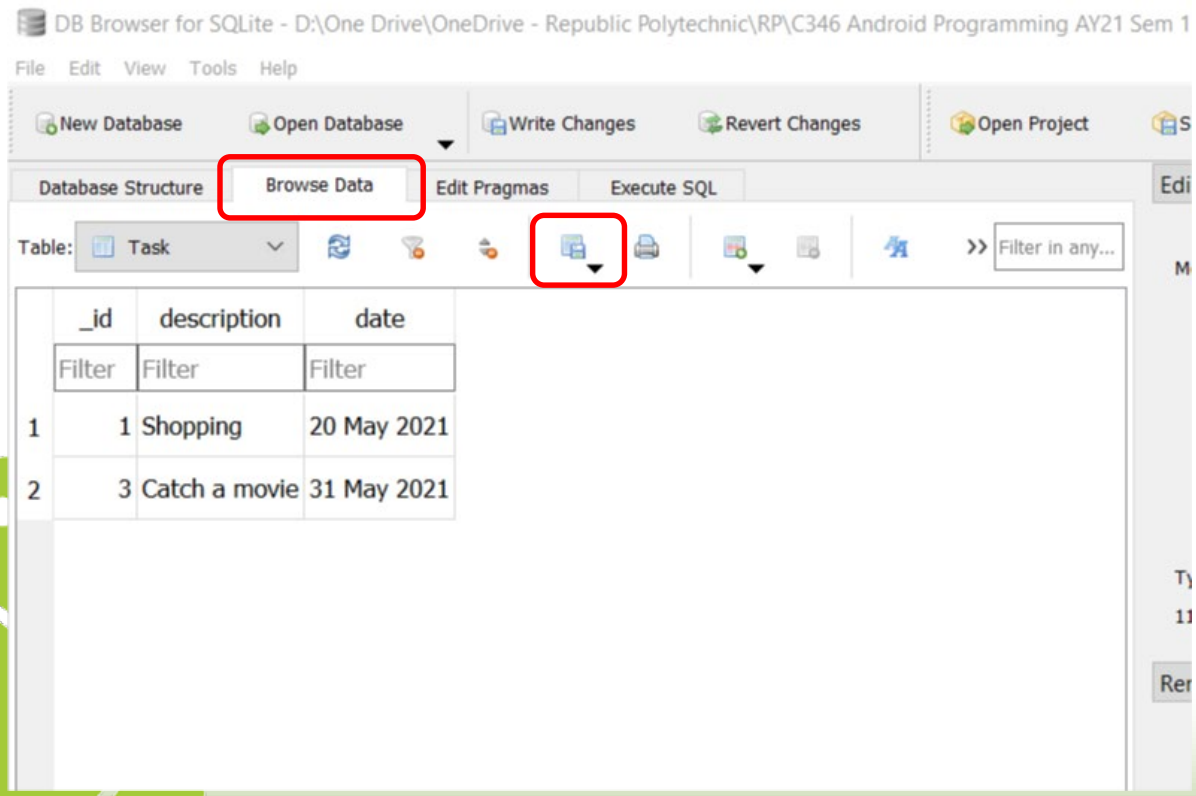
- Create a new database
  - Store the database file in P08 workspace
- Create a new table as follows
  - Use the table structure in Exercise 4





Inserting sample data

- Proceed to input some sample data
  - Browse Data
  - New Record



SQL – Data Manipulation Language

Create, **Retrieve**, **Update**, **Delete** (CRUD)

- **INSERT** INTO tablename (column1, column2) VALUES(values1, values2)
  - INSERT INTO customers (name, gender) VALUES('Adrian', 'm')

- **SELECT** column1, column2 FROM tablename WHERE condition
  - SELECT name, tel FROM users WHERE heightcm > 160

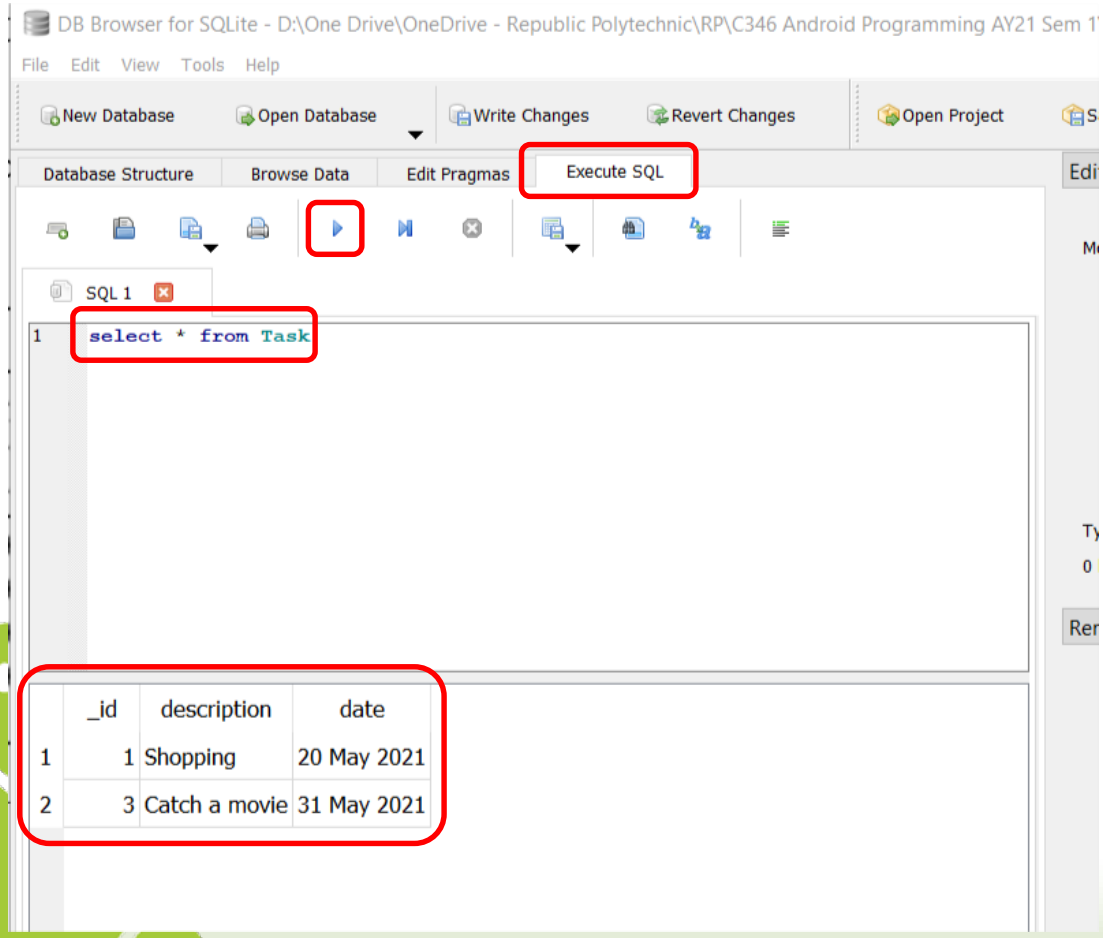
- **UPDATE** tablename SET column1=value1, column2=value2 WHERE condition
  - UPDATE movies SET stars=5, status='akan datang' WHERE title='Harry Potter 6‘

- **DELETE** FROM tablename WHERE condition
  - DELETE FROM tasks WHERE status <> 'active'

We’ll focus on Insert and Select in P08

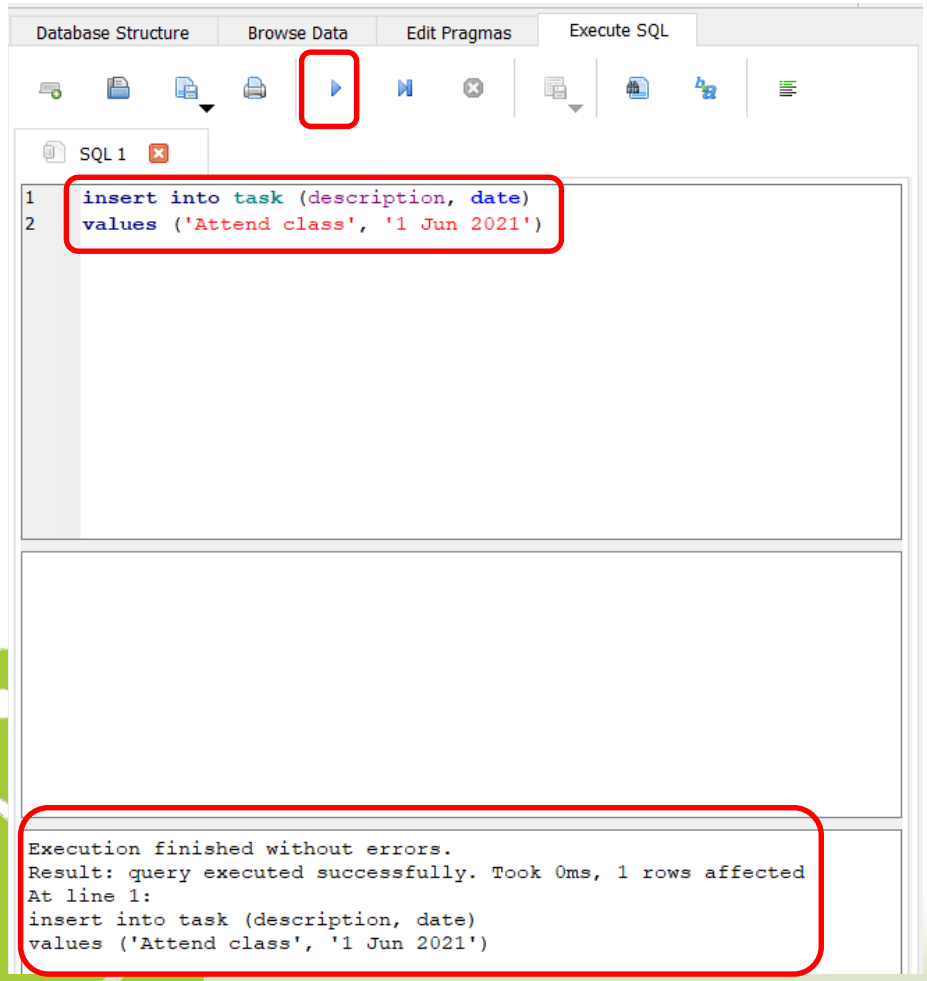
DML - Select

- Perform a Select SQL statement in DB Browser



DML - Insert

- Perform a Insert SQL statement in DB Browser



SQLite database in Android App

- Android supports SQLite
- Android provides a helper class as a framework to manage the database in apps
  - SQLiteOpenHelper
- The helper class will manage the database in the under the following scenario
  - Database creation for App installed for the first time
  - Database upgrade (changes) to an existing installed App
    - This is tricky as you don't want to lose the data in an existing App!
- The database is private, accessible to the App only

SQLite database in Android App

- Requires a Java class extending from SQLiteOpenHelper
- The helper class will require the following:
  - A constructor **DBHelper()**
  - A method named **onCreate()**
  - A method named **onUpgrade()**

```java
public class DBHelper extends SQLiteOpenHelper{

    public DBHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
    }

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

Exercise 5

- Referring to WS Section E, create a project as below

Project Name	Demo Database
Package	com.myapplicationdev.android.demodatabase
Activity Name	MainActivity
Layout Name	activity_main.xml
Min SDK	API 16

- Create a new Java Class named DBHelper.java
- Extend the class with SQLiteOpenHelper

```
package com.myapplicationdev.android.demodatabase;

import android.database.sqlite.SQLiteOpenHelper;

public class DBHelper extends SQLiteOpenHelper {

}
```

Exercise 5 (con't)

- Implement the following skeleton code:
  - A constructor **DBHelper()**
  - A method named **onCreate()**
  - A method named **onUpgrade()**

```java
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class DBHelper extends SQLiteOpenHelper{

    public DBHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
    }

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

Exercise 5 (con't)

- We'll proceed to verify the database creation after this

Examine database file from Android

- Be default, the SQLite database file is private to the app
- There's no way to access the database of an App in a non-rooted Android phone

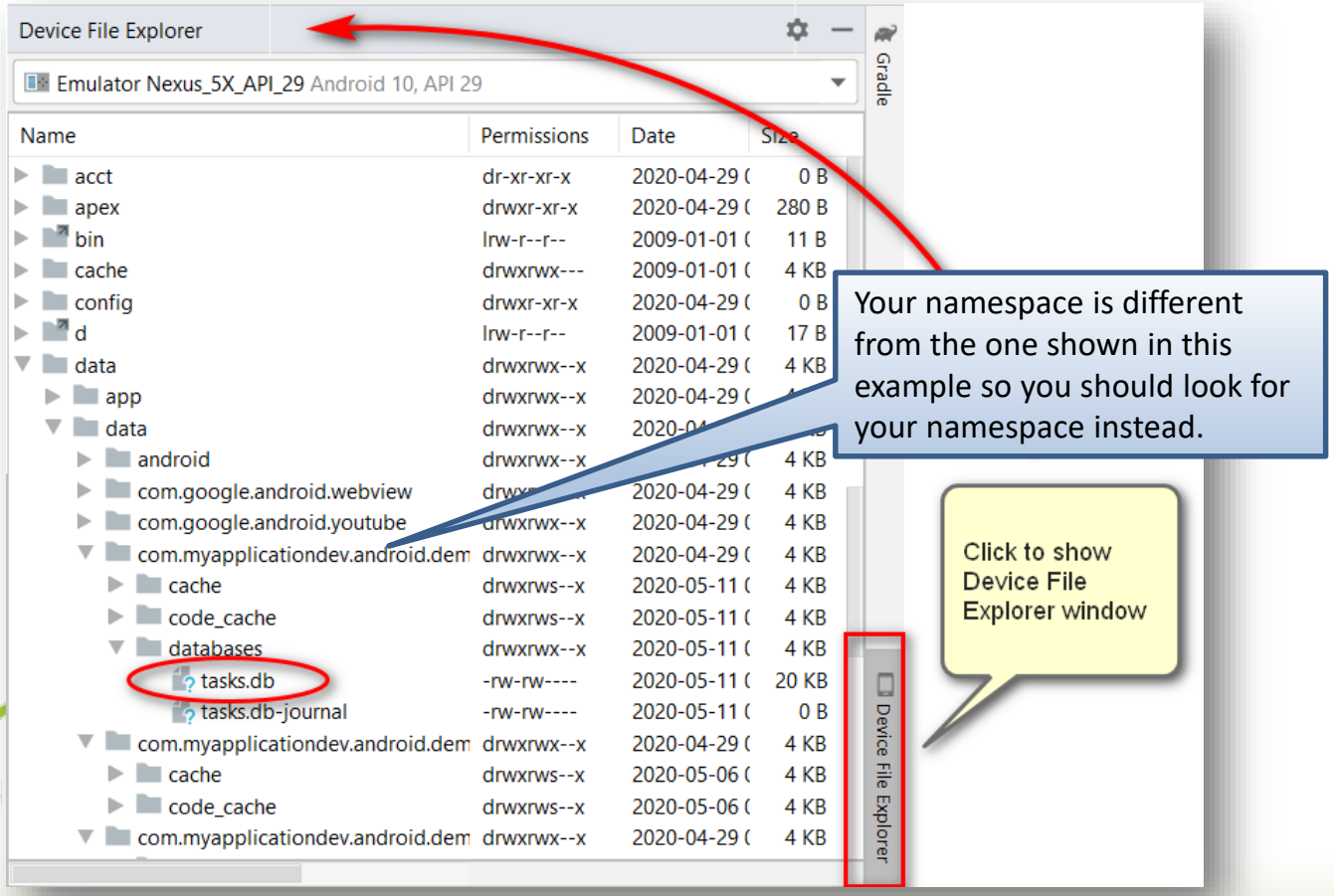
- For development purposes, we could access the database in the **Android emulator**

- The SQLite database is a file located under
  - /data/data/<package_name>/databases

- We will use the Device File Explorer in Android Studio, made available since Android 3.0 to examine the files.

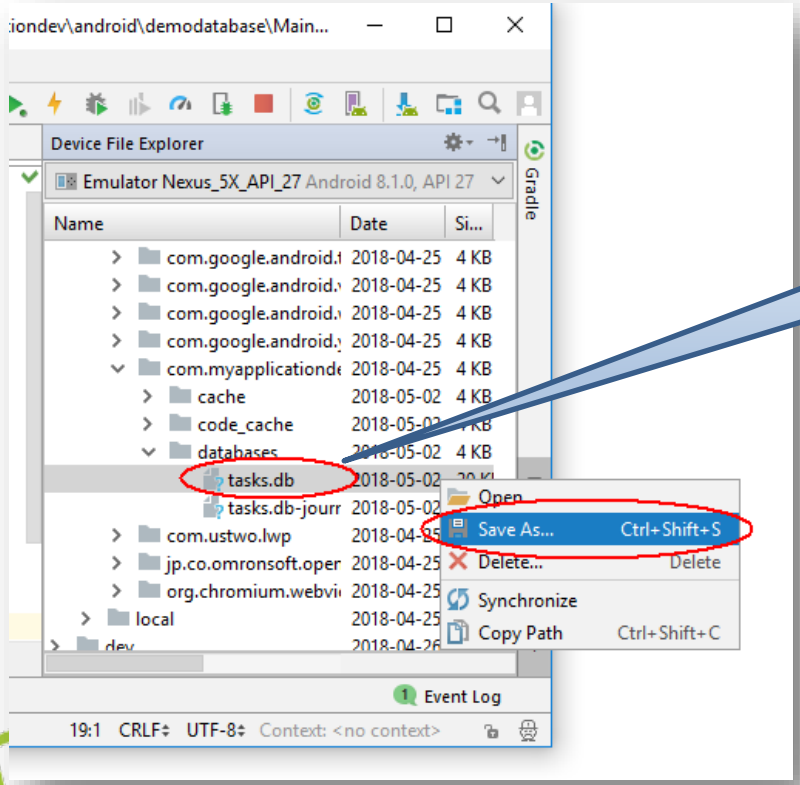
Examine database file from Android

- We will use the Device File Explorer to locate the database file



Examine database file from Android

- Location of database file -> /data/data/package_name/databases



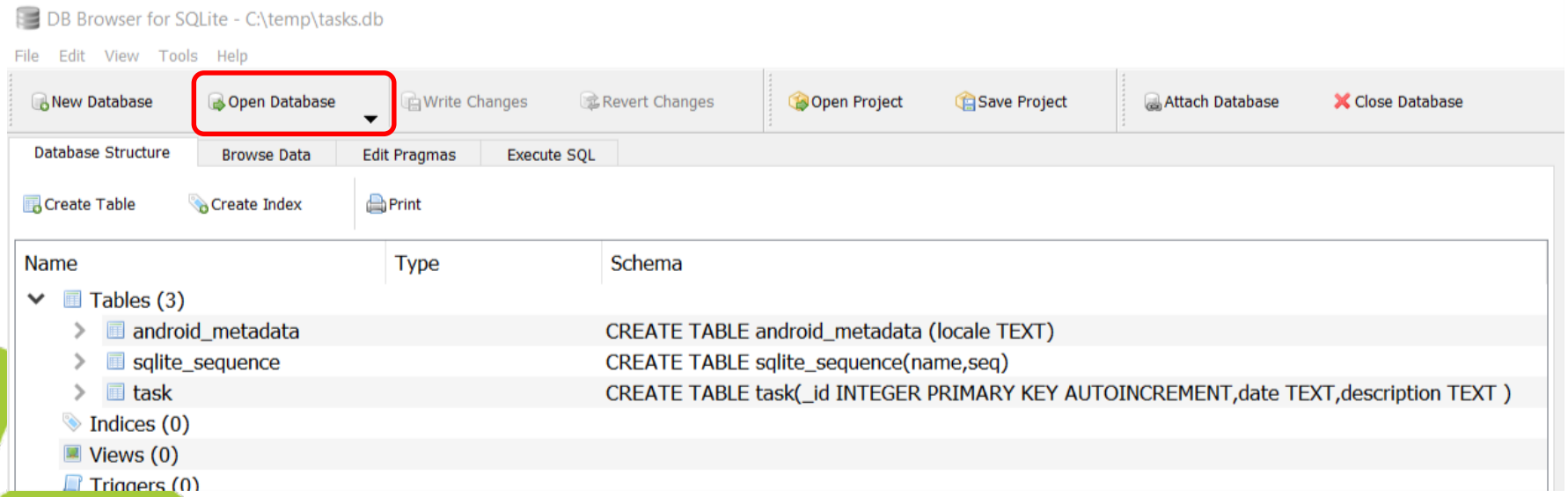
Highlight the file – “tasks.db”

Right click to save the file in the same workspace of the project

- With Device File Explorer, you can extract a file from the device
- Similarly, you could also upload a file into the folder

Examine database file from Android

- Open the database file for examination with the DB Browser for SQLite



Inserting a record

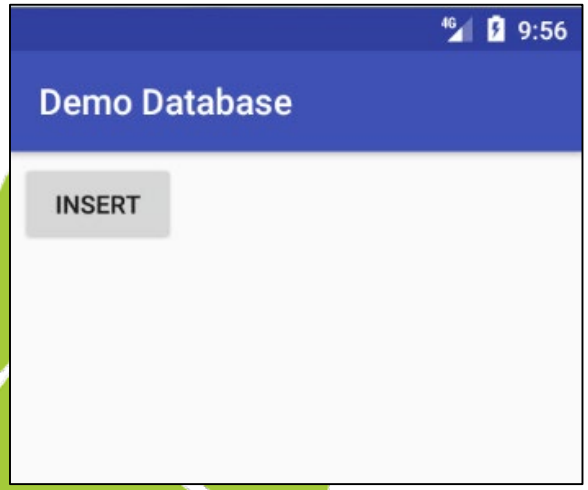
- Usually it is a method in the Helper class
- Argument list is customizable
- Typically, ordered as below:
  - Retrieve the database object
  - Create a ContentValues object
  - Put values into the ContentValues object
  - Insert the data
  - Close the database object

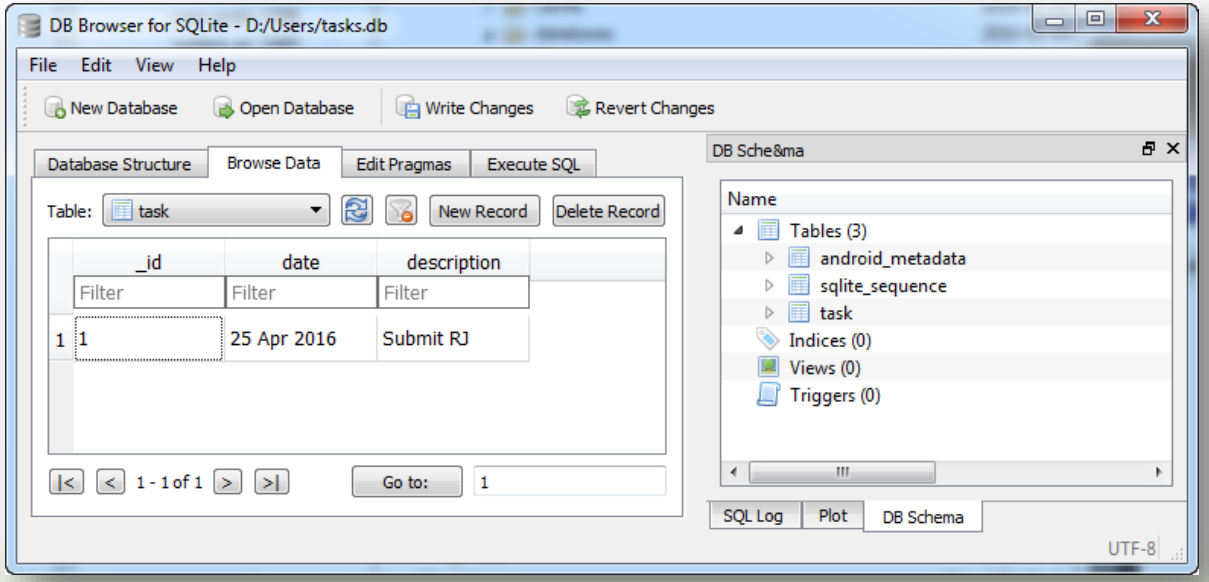
```
1   public class DBHelper extends SQLiteOpenHelper{
      …
      …
37    public void insertTask(String description, String date){
38            // Get an instance of the database for writing
39            SQLiteDatabase db = this.getWritableDatabase();
40
41            // ContentValues object to store the values for the db operation
42            ContentValues values = new ContentValues();
43
44            // Store the column name as key and the date as value
45            values.put(COLUMN_DESCRIPTION, description);
46            values.put(COLUMN_DATE, date);
47
48            // Insert the row into the TABLE_NOTE
49            db.insert(TABLE_TASK, null, values);
50
51            // Close the database connection
52            db.close();
53    }
54  }
```

Exercise 6

- Refer to Section F of the Worksheet

- Implement the method for record insertion in the DBHelper
- Create a button on the MainActivity, upon clicking of the button, it will insert a record into the database.
- Examine the content of the database file using DB Browser





Exercise 6 (ans)

```java
public class MainActivity extends AppCompatActivity {

    Button btnInsert;

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

        btnInsert = (Button) findViewById(R.id.btnInsert);
        btnInsert.setOnClickListener(new View.OnClickListener(){

            @Override
            public void onClick(View v) {
                // Create the DBHelper object, passing in the activity's Context
                DBHelper db = new DBHelper(MainActivity.this);

                // Insert a task
                db.insertTask("Submit RJ", "25 Apr 2016");
                db.close();
            }});

    }
}
```

If the database is not created yet, onCreate will be called after the constructor

```java
public DBHelper(Context context) { super(context, DATABASE_NAME, null, DATABASE_VER); }

@Override
public void onCreate(SQLiteDatabase db) {...}

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

public void insertTask(String description, String date){
    // Get an instance of the database for writing
    SQLiteDatabase db = this.getWritableDatabase();
    // We use ContentValues object to store the values for
    //  the db operation
    ContentValues values = new ContentValues();
    // Store the column name as key and the description as value
    values.put(COLUMN_DESCRIPTION, description);
    // Store the column name as key and the date as value
    values.put(COLUMN_DATE, date);
    // Insert the row into the TABLE_NOTE
    db.insert(TABLE_TASK, null, values);
    // Close the database connection
    db.close();
}
```

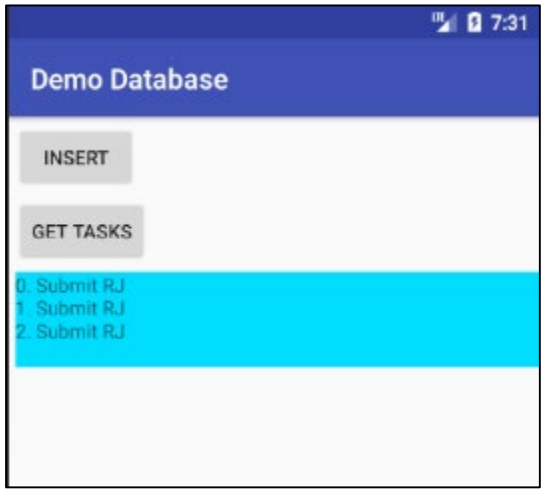
Retrieving record(s)

- Usually it is a method in the Helper class
- Typically, ordered as below:
  - Retrieve the database object
  - Execute a Select SQL statement, a Cursor object will be returned
  - Retrieve the data using the Cursor object
  - Construct the data
  - Close the database object
  - Return the data

```
1   public class DBHelper extends SQLiteOpenHelper{
       …
       …
53     public ArrayList<String> getTaskContent() {
54        ArrayList<String> tasks = new ArrayList<String>();
55
56        String selectQuery = "SELECT " + COLUMN_DESCRIPTION
             + " FROM " + TABLE_TASK;
57
58        SQLiteDatabase db = this.getReadableDatabase();
59        Cursor cursor = db.rawQuery(selectQuery, null);
60
61        // moveToFirst() moves to first row
62        // will passed if statement if empty results
63        if (cursor.moveToFirst()) {
64           do {
65              //retrieve all the records, one row at a time
66              tasks.add(cursor.getString(0));
67           } while (cursor.moveToNext());
68        }
69
70        cursor.close();
71        db.close();
72        return tasks;
73     }
74
75  }
76
```

Exercise 7

- Refer to Section G of the Worksheet
- Retrieve the data from the database and display it

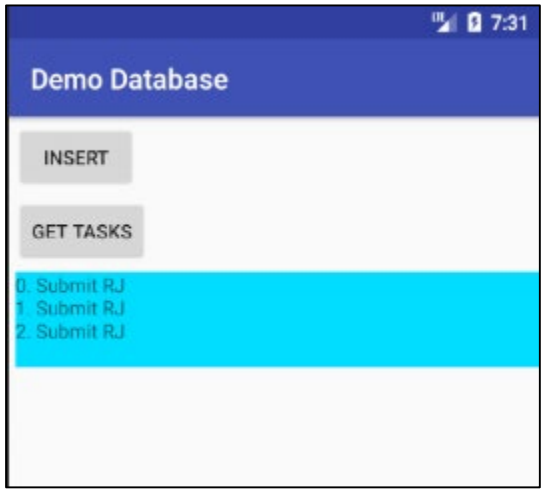



```
logcat   Monitors                                                    Verbose
05-03 10:06:35.655 4380-4380/com.myapplicationdev.android.demodatabase D/Database Content: 0. Submit RJ
05-03 10:06:35.656 4380-4380/com.myapplicationdev.android.demodatabase D/Database Content: 1. Submit RJ
05-03 10:06:35.656 4380-4380/com.myapplicationdev.android.demodatabase D/Database Content: 2. Submit RJ
```

Exercise 7

- Sample output

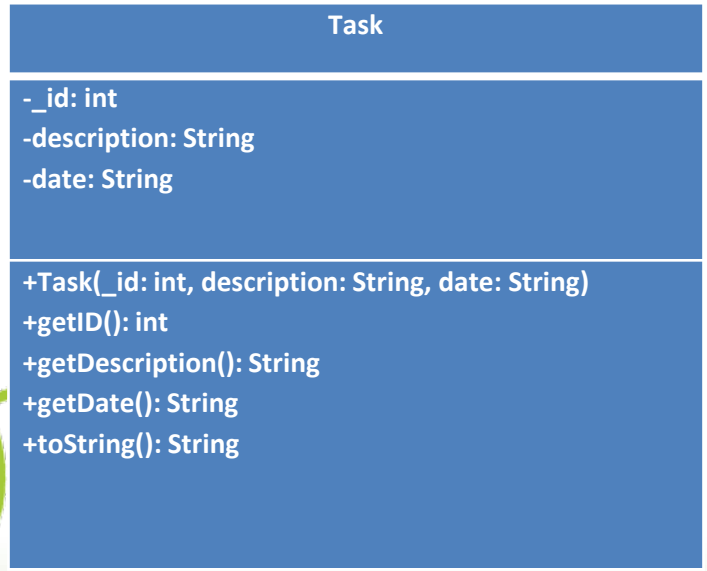
```java
btnGetTasks.setOnClickListener(new View.OnClickListener(){

    @Override
    public void onClick(View v) {
        // Create the DBHelper object, passing in the activity's Context
        DBHelper db = new DBHelper(MainActivity.this);

        ArrayList<String> data = db.getTaskContent();
        db.close();

        String txt = "";
        for (int i = 0; i < data.size(); i++) {
            Log.d("Database Content",  i + ". " + data.get(i) );
            txt += i + ". " + data.get(i) + "\n";
        }
        tvResults.setText(txt);
    }

});
```




```
logcat   Monitors                                                    Verbose
05-03 10:06:35.655 4380-4380/com.myapplicationdev.android.demodatabase D/Database Content: 0. Submit RJ
05-03 10:06:35.656 4380-4380/com.myapplicationdev.android.demodatabase D/Database Content: 1. Submit RJ
05-03 10:06:35.656 4380-4380/com.myapplicationdev.android.demodatabase D/Database Content: 2. Submit RJ
```

Data modelling (from database to Java objects)

- Instead of retrieving as String content, we could model it into a Java object
- Create a corresponding method to retrieve data in Java objects form




```java
public class Task {
    private int id;
    private String description;
    private String date;

    public Task(int id, String description, String date) {
        this.id = id;
        this.description = description;
        this.date = date;
    }

    public int getId() {
        return id;
    }

    public String getDescription() {
        return description;
    }

    public String getDate() {
        return date;
    }

    public String toString() {
        return id + "\n" + description + "\n" + date;
    }
}
```

Data modelling (from database to Java objects)

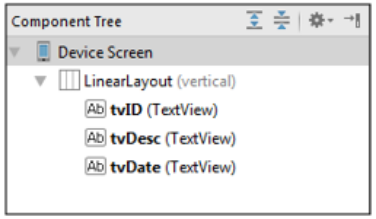
- Usually it is a method in the Helper class
- Similar to earlier method, but:
  - The SQL Statement needs to get as many data as the Java class requires
  - Need to retrieve the individual data according to selected column
  - Construct the object
  - Close the database object
  - Return the data in ArrayList

```
public class DBHelper extends SQLiteOpenHelper{
  ...
  ...
  public ArrayList<Task> getTasks() {
    ArrayList<Task> tasks = new ArrayList<Task>();
    String selectQuery = "SELECT " + COLUMN_ID + ", "
        + COLUMN_DESCRIPTION + ", "
        + COLUMN_DATE
        + " FROM " + TABLE_TASK;

    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery(selectQuery, null);

    if (cursor.moveToFirst()) {
        do {
                int id = cursor.getInt(0);
                String description = cursor.getString(1);
                String date = cursor.getString(2);
                Task obj = new Task(id, description, date);
                tasks.add(obj);
        } while (cursor.moveToNext());
    }
    cursor.close();
    db.close();
    return tasks;
  }
}
```

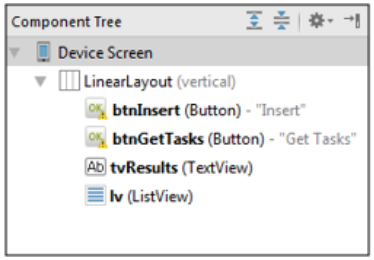
Task
-_id: int -description: String -date: String
+Task (int, String, String)

Exercise 8

- Refer to Section H of the Worksheet
  - Retrieval of database records in Java objects
- Refer to Section I of the Worksheet
  - Presenting the data in Custom ListView



row.xml



activity_main.xml

Exercise 8

- With the Custom ArrayAdapter along with the relevant Layout XML file, just bind the components together to form the solution

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


    …

    lv = (ListView)  findViewById(R.id.lv);

    btnGetTasks.setOnClickListener(new View.OnClickListener(){
        @Override
        public void onClick(View v) {
            …

            DBHelper db2 = new DBHelper(MainActivity.this);
            al = db2.getTasks();
            db2.close();

            aa = new TaskArrayAdapter(MainActivity.this, R.layout.row, al);
            lv.setAdapter(aa);
        }
    });
}
```

```java
public class DBHelper extends SQLiteOpenHelper{
  …
  …
  public ArrayList<Task> getTasks() {
    ArrayList<Task> tasks = new ArrayList<Task>();
    String selectQuery = "SELECT " + COLUMN_ID + ", "
        + COLUMN_DESCRIPTION + ", "
        + COLUMN_DATE
        + " FROM " + TABLE_TASK;

    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery(selectQuery, null);

    if (cursor.moveToFirst()) {
        do {
                int id = cursor.getInt(0);
                String description = cursor.getString(1);
                String date = cursor.getString(2);
                Task obj = new Task(id, description, date);
                tasks.add(obj);
        } while (cursor.moveToNext());
    }
    cursor.close();
    db.close();
    return tasks;
  }
}
```

SQLite Comparison

Operator	Function	Example
<	Less than	height < 160
<=	Less than or equal to	price <= 3.5
>	Greater than	age > 60
>=	Greater than or equal to	networth >= 1000000
=	Equals to	name = 'C.S. Lewis'
!= or <>	Not equals to	color <> 'black'
LIKE	Similar to	name LIKE '%son' (matches 'Jackson', 'Nealson', 'Son' etc) name LIKE '%a%' (matches names with 'a' inside)

SQLite Logical Ops

Operator	Statement	Result
AND	SELECT price > 2 AND litre > 3	true AND false → false
OR	SELECT price > 4 OR litre < 3	false OR true → true
NOT	SELECT NOT (litre = 2)	NOT true → false
AND	SELECT price > 3 AND litre >= 2	true AND true → true
NOT	SELECT NOT((price / litre) > 2)	NOT false → true

Break

- Continue with Session 2