Android Developer

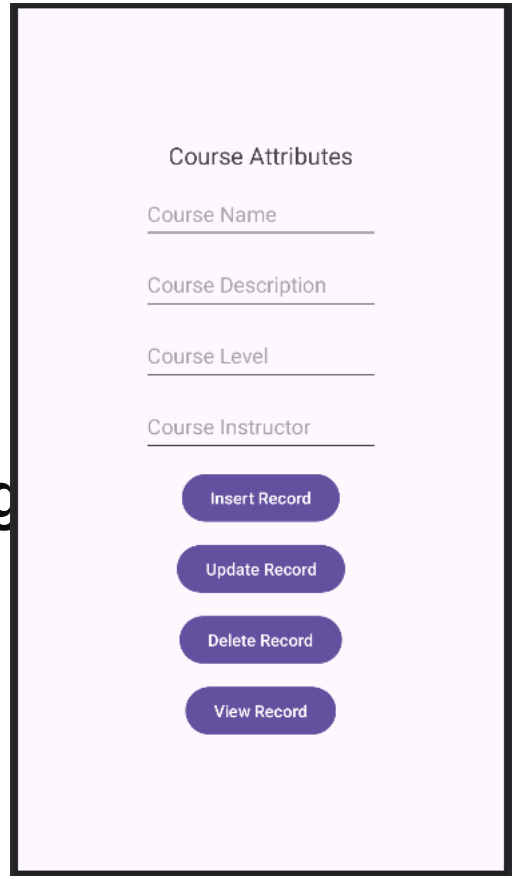
WORKING WITH DATABASE

CRUD (Create, Read, Update, Delete)

- **Steps:**
- **1. Project Setup**
  - **Create a New Project**: Start by creating a new Android Studio project with an empty activity.
  - **Project Structure**: Organize your project structure with separate packages for activities, layouts, and database handling.

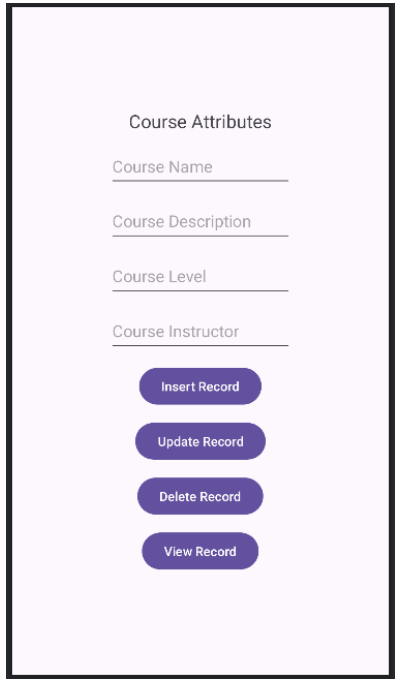
Layout File

- **2. Layout Design**
  - **Design the Layout**: Create a layout XML file to design the UI for your CRUD app.
  - **Course Attributes**: Define fields for course attributes like **name**, **description**, **level**, and **instructor**.
  - **Buttons**: Add buttons for **inserting**, **updating**, **deleting** and **viewing** course records.



Layout Design

- **Implement LinearLayout**
- Add this attribute
- android:gravity="center"
  android:orientation="vertical"




```
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Course Attributes"
  android:textSize="20sp"
  />
<EditText
  android:id="@+id/edtCourseName"
  android:layout_width="200dp"
  android:layout_height="48dp"
  android:layout_marginTop="12dp"
  android:hint="Course Name"
  android:inputType="text"/>
<EditText
  android:id="@+id/edtCourseDescription"
  android:layout_width="200dp"
  android:layout_height="48dp"
  android:layout_marginTop="12dp"
  android:hint="Course Description"
  android:inputType="text"/>

<EditText
  android:id="@+id/edtCourseLevel"
  android:layout_width="200dp"
  android:layout_height="48dp"
  android:layout_marginTop="12dp"
  android:hint="Course Level"
  android:inputType="text"/>
```

```
<EditText
  android:id="@+id/edtCourseInstructor"
  android:layout_width="200dp"
  android:layout_height="48dp"
  android:layout_marginTop="12dp"
  android:hint="Course Instructor"
  android:inputType="text"/>

<Button
  android:id="@+id/btnInsert"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_marginTop="12dp"
  android:text="Insert Record"
  />

<Button
  android:id="@+id/btnUpdate"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_marginTop="12dp"
  android:text="Update Record"
  />

<Button
android:id="@+id/btnDelete"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginTop="12dp"
android:text="Delete Record"
/>

<Button
  android:id="@+id/btnView"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_marginTop="12dp"
  android:text="View Record"
  />
```

Database Schema

- **Database Name**: CourseInfo.db
- **Database Version**: 1
- **Table Name: my_course**

- **Columns**:
  - **c_id**: INTEGER (Primary Key, Autoincrement)
  - **c_name**: TEXT
  - **c_description**: TEXT
  - **c_level**: TEXT
  - **c_instructor**: TEXT

my_course

c_id	c_name	c_description	c_level	c_instructor

SQL queries

- SQL queries for insert, update, delete, and view operations

Insert Query

INSERT INTO my_course (c_name, c_description, c_level, c_instructor) VALUES (?, ?, ?, ?)

Update Query

UPDATE my_course SET c_description = ?, c_level = ?, c_instructor = ? WHERE c_name = ?

Delete Query

DELETE FROM my_course WHERE c_name = ?

View Query

SELECT * FROM my_course

Implement MainActivity.java

- **Initialize Views**: Initialize **EditText** fields and **buttons** in the MainActivity.
- **onClickListener**
  - **Handle Insertion**: Implement **onClickListener** for the insert button to add a new course record to the database.
  - **Handle Updating**: Implement **onClickListener** for the update button to modify existing course records.
  - **Handle Deletion**: Implement **onClickListener** for the delete button to remove course records from the database.
  - **Handle Viewing**: Implement **onClickListener** for the view button to retrieve and display course records.

Implement this in the **MainActivity.java**

```java
// declare these variables
EditText courseName, courseDes, courseLevel, courseInstructor;
Button btnInsert, btnUpdate, btnDelete, btnView;
DatabaseHelper myDB;
```

```java
// intialize the views
courseDes = findViewById(R.id.edtCourseDescription);
courseLevel = findViewById(R.id.edtCourseLevel);
courseInstructor = findViewById(R.id.edtCourseInstructor);
courseName = findViewById(R.id.edtCourseName);

btnInsert = findViewById(R.id.btnInsert);
btnUpdate = findViewById(R.id.btnUpdate);
btnDelete = findViewById(R.id.btnDelete);
btnView = findViewById(R.id.btnView);

// intialize the DBAgent
myDB = new DatabaseHelper(this);
```

Insertion

```java
String course_Name = courseName.getText().toString();
String course_Des = courseDes.getText().toString();
String course_Level = courseLevel.getText().toString();
String course_Instructor = courseInstructor.getText().toString();

myDB.insertCourse(course_Name, course_Des, course_Level, course_Instructor);
```

Update

```java
String course_Name = courseName.getText().toString();
String course_Des = courseDes.getText().toString();
String course_Level = courseLevel.getText().toString();
String course_Instructor = courseInstructor.getText().toString();

myDB.updateCourse(course_Name, course_Des, course_Level, course_Instructor);
```

Delete

```java
String course_Name = courseName.getText().toString();

myDB.deleteCourse(course_Name);
```

Design the Database and its Classes

- **Design your Database:** Know your data structure such as its column and data types
- **Create Database Helper Class**: Implement a **SQLiteOpenHelper** subclass to manage database creation and version management.
- **Define Database Schema**: Define the **table name** and **column names** for storing course information.
- **Implement CRUD Methods**: Implement methods to **insert**, **update**, **delete**, and **view** course records in the database.

Methods and Constructors involved in the SQLiteOpenHelper

- **DatabaseHelper**(@Nullable Context context):
  - Is a **constructor method** for the DatabaseHelper class.
  - We use it to **initializes the DatabaseHelper object with the provided Context**.
  - The **Context** is **necessary for interacting with the Android framework and accessing resources or system services**.

```
public DatabaseHelper(@Nullable Context context) {
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
    this.context = context;
}
```

Conti…

- **onCreate**(SQLiteDatabase db):
  - We call this method when the database is created for the first time.
  - Inside this method, we **define the structure of your database tables by executing SQL commands**.
  - It creates a table and its columns.

```
@Override
public void onCreate(SQLiteDatabase db) {
    String query =
            "CREATE TABLE " + TABLE_NAME +
                    " (" + COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
                    COLUMN_NAME + " TEXT, " +
                    COLUMN_DESCRIPTION + " TEXT, " +
                    COLUMN_LEVEL + " TEXT, " +
                    COLUMN_INSTRUCTOR + " TEXT);";
    db.execSQL(query);
}
```

- **DATABASE_NAME**: The name of the database file.
- **DATABASE_VERSION**: The version of the database. Increment this when making schema changes to trigger the onUpgrade() method.
- **TABLE_NAME**: The name of the database table.
- **Column variables** (COLUMN_ID, COLUMN_NAME, etc.): Define the names of columns in the table.

Conti…

- **onUpgrade**(SQLiteDatabase db, int **oldVersion**, int **newVersion**):
  - We use it to handle database schema changes when the application is updated.
  - This method is called **when the database needs to be upgraded**, usually
  - when the version number of the database is incremented.
  - drops the existing table if it exists and then calls the **onCreate**() **ema**.

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
    onCreate(db);
}
```

When Data is Lost

- Data loss occurs if the **onUpgrade** method is triggered because it drops the existing table. This can happen in the following scenarios:
  - **Database Version Increment**: The version of the database is changed by incrementing the DATABASE_VERSION variable.
  - **Schema Change**: You change the schema of the database (e.g., adding or removing columns) and increment the database version to apply these changes.
- **Why You Haven't Lost Data Yet**
- You haven't lost any data so far because:
  - **No Version Change**: The database version (DATABASE_VERSION) has not been incremented. It is still set to 1.
  - **No Schema Change**: The schema of the database hasn't been altered.
- As a result, the **onUpgrade** method hasn't been called, so the table hasn't been dropped and recreated.

Database Agent Class

Step One

```
public class DatabaseHelper extends SQLiteOpenHelper
```

Extend your class to **SQLiteOpenHelper**

Step Two

```
// Optional to create variables
private static final String DATABASE_NAME = "CourseInfo.db";
private static final int DATABASE_VERSION =1;
private static final String TABLE_NAME = "my_course";
private static final String COLUMN_ID = "c_id";
private static final String COLUMN_NAME = "c_name";
private static final String COLUMN_DESCRIPTION= "c_description";
private static final String COLUMN_LEVEL = "c_level";
private static final String COLUMN_INSTRUCTOR = "c_instructor";

private Context context;
```

context To provide access to the application's environment (e.g., for displaying Toast messages).

Step Three

```
public DatabaseHelper(@Nullable Context context) {
   super(context, DATABASE_NAME, null, DATABASE_VERSION);
   this.context = context; // Initialize the context field
}
```

Step Four

```
public void onCreate(SQLiteDatabase db) {
   String query =
         "CREATE TABLE " + TABLE_NAME +
               " (" + COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
               COLUMN_NAME + " TEXT, " +
               COLUMN_DESCRIPTION + " TEXT, " +
               COLUMN_LEVEL + " TEXT, " +
               COLUMN_INSTRUCTOR + " TEXT);";
   db.execSQL(query);
}
```

Step three: explained

- **super(...)**: Calls the constructor of the parent class (SQLiteOpenHelper) with:
- **context**: Application context for accessing resources.
- **DATABASE_NAME**: Database file name.
- **null**: Indicates no custom cursor factory is provided.
- **DATABASE_VERSION**: Version for database upgrades.

- **this.context = context**: Assigns the passed context to the context field for later use (e.g., in Toast messages).

Step Five

```java
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
    onCreate(db);
}
```

Step Six

```java
// create CRUD METHODS
void insertCourse(String c_name, String c_description, String c_level, String c_instructor){
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues cv = new ContentValues();

    cv.put(COLUMN_NAME, c_name);
    cv.put(COLUMN_DESCRIPTION, c_description);
    cv.put(COLUMN_LEVEL, c_level);
    cv.put(COLUMN_INSTRUCTOR, c_instructor);

    long result = db.insert(TABLE_NAME, null, cv);

    if(result == -1){
        Toast.makeText(context, "Failed to Insert", Toast.LENGTH_SHORT).show();
    }else {
        Toast.makeText(context, "Successfully Inserted new row", Toast.LENGTH_SHORT).show();

    }
}
```

Step Seven

```java
void updateCourse(String c_name, String c_description, String c_level, String c_instructor) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues cv = new ContentValues();

    cv.put(COLUMN_DESCRIPTION, c_description);
    cv.put(COLUMN_LEVEL, c_level);
    cv.put(COLUMN_INSTRUCTOR, c_instructor);

    int result = db.update(TABLE_NAME, cv, COLUMN_NAME + "=?", new String[]{c_name});

    if (result == -1) {
        Toast.makeText(context, "Failed to Update", Toast.LENGTH_SHORT).show();
    } else {
        Toast.makeText(context, "Successfully Updated the row", Toast.LENGTH_SHORT).show();
    }
}
```

Step Eight

```java
void deleteCourse(String c_name) {
    SQLiteDatabase db = this.getWritableDatabase();
    int result = db.delete(TABLE_NAME, COLUMN_NAME + "=?", new String[]{c_name});

    if (result == -1) {
        Toast.makeText(context, "Failed to Delete", Toast.LENGTH_SHORT).show();
    } else {
        Toast.makeText(context, "Successfully Deleted the row", Toast.LENGTH_SHORT).show();
    }
}
```

Step five: explained

- **Drop Existing Table**: Deletes the current table (my_course) if it exists. This is useful during schema changes.
- **Recreate Table**: Calls onCreate(db) to create the updated table structure.

Step six: Explained

Get

Get Writeable Database:

- SQLiteDatabase db = this.getWritableDatabase(); opens the database for writing operations.

Prepare

Prepare Values:

- ContentValues: A key-value store used to map column names to values for insertion.
- Adds values for COLUMN_NAME, COLUMN_DESCRIPTION, COLUMN_LEVEL, and COLUMN_INSTRUCTOR.

Insert

Insert into Table:

- db.insert(TABLE_NAME, null, cv): Inserts the values into the my_course table. Returns:

Show

Show Results Using Toast:

- Displays a Toast message indicating success or failure.

Some of the instance you need

- **SQLiteDatabase db = this.getWritableDatabase();**
  - **Why**: We need to perform write operations (inserting data) on the database.
  - **What It Provides**: Access to a writable instance of the database.

gets a writable instance of the database. It allows us to perform read and write operations on the database. We need this to insert new data into the database.

- **ContentValues cv = new ContentValues();**
  - **Why**: We need a way to store and organize the data we want to insert into the database.
  - **What It Provides**: An object that holds the data as key-value pairs, making it easy to insert into the database.

creates an instance of **ContentValues**, which is used to store a set of key-value pairs. In this case, it will store the course details (name, description, level, and instructor).

The view

Step Nine

Put this method in your DatabaseAgent class

```
Cursor viewCourse() {
    SQLiteDatabase db = this.getWritableDatabase();
    return db.rawQuery("SELECT * FROM " + TABLE_NAME, null);
}
```

Step Ten

Put this method in your MainActivity.java class

```
Cursor cursor = myDB.viewCourse();

if (cursor.getCount() == 0) {
    // If no data is available, display a message
    Toast.makeText(MainActivity.this, "No data available", Toast.LENGTH_SHORT).show();
    return;
} else {
    // If data is available, build the string to display
    StringBuffer buffer = new StringBuffer();
    while (cursor.moveToNext()) {
        buffer.append("ID: ").append(cursor.getString(0)).append("\n");
        buffer.append("Name: ").append(cursor.getString(1)).append("\n");
        buffer.append("Description: ").append(cursor.getString(2)).append("\n");
        buffer.append("Level: ").append(cursor.getString(3)).append("\n");
        buffer.append("Instructor: ").append(cursor.getString(4)).append("\n\n");
    }
    // Create and show the AlertDialog with the data
    AlertDialog.Builder builder;
    builder = new AlertDialog.Builder(MainActivity.this);
    builder.setCancelable(true);
    builder.setTitle("Course Information");
    builder.setMessage(buffer.toString());
    builder.show();

}
```

Cursor

```
Cursor viewCourse() {
    SQLiteDatabase db = this.getWritableDatabase();
    return db.rawQuery("SELECT * FROM " + TABLE_NAME, null);
}
```

1. **Cursor**
    1. **What it is**: A Cursor in Android is an interface that provides read-write access to the result set returned by a database query.
    2. **Usage**:
        1. Here, the viewCourse() method uses Cursor to fetch all rows from the database table (TABLE_NAME) using a raw SQL query (SELECT * FROM ...).
        2. Cursor is used because it allows efficient navigation through the rows in the result set.
    3. **Why we use it**: It provides a way to read individual rows and columns returned from a database query.

Cursor

- **cursor.getCount()**
- **What it is**: This method returns the number of rows in the result set.
- **Usage**:
  - Used to check if the query returned any data.
  - If getCount() is 0, it means the database table is empty or the query found no matches.
- **Why we use it**: It helps handle scenarios where there is no data and avoids proceeding with empty results.

```
if (cursor.getCount() == 0) {
    Toast.makeText(MainActivity.this, "No data available", Toast.LENGTH_SHORT).
    return;
}
```

StringBuffer

- **StringBuffer**
- **What it is**: A StringBuffer is a thread-safe, mutable sequence of characters used to build strings.
- **Usage**:
  - Here, StringBuffer collects and concatenates the data from the Cursor.
  - It appends each piece of data (ID, Name, etc.) in a formatted way.
- **Why we use it**: It is more memory-efficient and faster for string manipulation compared to using concatenation (+) repeatedly, especially when working with large data.

```java
StringBuffer buffer = new StringBuffer();
while (cursor.moveToNext()) {
    buffer.append("ID: ").append(cursor.getString(0)).append("\n");
    buffer.append("Name: ").append(cursor.getString(1)).append("\n");
    buffer.append("Description: ").append(cursor.getString(2)).append("\n");
    buffer.append("Level: ").append(cursor.getString(3)).append("\n");
    buffer.append("Instructor: ").append(cursor.getString(4)).append("\n\n");
}
```

AlertDialog

- **AlertDialog.Builder**
- **What it is**: A class used to create and customize a dialog box that can display messages or interact with the user.
- **Usage**:
  - Here, it is used to create a dialog that shows the concatenated string (buffer.toString()) containing course details fetched from the database.
- **Why we use it**: It provides a clean and user-friendly way to display data in the form of a popup dialog.

```
AlertDialog.Builder builder;
builder = new AlertDialog.Builder(MainActivity.this);
builder.setCancelable(true);
builder.setTitle("Course Information");
builder.setMessage(buffer.toString());
builder.show();
```

Cursor Interface

- read-write access to the **result set returned by a database query**.
- Is used to **iterate over the rows** in the result set and **retrieve the values of the columns for each row**
- close a `Cursor` when you are done with it to release the resources it holds.
- This is usually done with `cursor.close()`.

Summary: Workflow of the Code

1. **Fetch data**: The viewCourse() method retrieves data from the database using a Cursor.
2. **Check data availability**: cursor.getCount() verifies if there are any results.
    1. If no results, show a Toast message: "No data available."
    2. If results exist, proceed to build the output.
3. **Build data string**: A StringBuffer concatenates course details row by row from the Cursor.
4. **Show the data**: An AlertDialog.Builder is used to display the collected course data to the user.