Competency Based Learning Materials (CBLM)

Android Mobile Application Development

Level-4

Module: Managing Database

Code: CBLM-OU-ICT-AMAD-04-L4-V1

National Skills Development Authority
Prime Minister's Office
Government of the People's Republic of Bangladesh

Copyright

National Skills Development Authority
Prime Minister's Office
Level: 10-11, Biniyog Bhaban,
E-6 / B, Agargaon, Sher-E-Bangla Nagar Dhaka-1207, Bangladesh.
Email: ec@nsda.gov.bd
Website: www.nsda.gov.bd.
National Skills Portal: http:\\skillsportal.gov.bd

This Competency Based Learning Materials (CBLM) on "Managing Database" under the Android Mobile Application Development, Level-4 qualification is developed based on the national competency standard approved by National Skills Development Authority (NSDA)

This document is to be used as a key reference point by the competency-based learning materials developers, teachers/trainers/assessors as a base on which to build instructional activities.

This Competency Based Learning Materials is a document for the development of curricula, teaching and learning materials, and assessment tools. It also serves as the document for providing training consistent with the requirements of industry in order to meet the qualification of individuals who graduated through the established standard via competency-based assessment for a relevant job.

This document has been developed by NSDA in association with industry representatives, academia, related specialist, trainer, and related employee.

Public and private institutions may use the information contained in this CBLM for activities benefitting Bangladesh.

Approved by __ th Authority Meeting of NSDA Held on --------------

How To Use This Competency Based Learning Materials (CBLMS)

The module, Managing Database contains training materials and activities for you to complete. These activities may be completed as part of structured classroom activities or you may be required you to work at your own pace. These activities will ask you to complete associated learning and practice activities in order to gain knowledge and skills you need to achieve the learning outcomes.

1. Review the **Learning Activity** page to understand the sequence of learning activities you will undergo. This page will serve as your road map towards the achievement of competence.

2. Read the **Information Sheets.** This will give you an understanding of the jobs or tasks you are going to learn how to do. Once you have finished reading the **Information Sheets** complete the questions in the **Self-Check.**

3. **Self-**Checks are found after each **Information Sheet**. **Self-Checks** are designed to help you know how you are progressing. If you are unable to answer the questions in the **Self-Check** you will need to re-read the relevant **Information Sheet**. Once you have completed all the questions check your answers by reading the relevant **Answer Keys** found at the end of this module.

4. Next move on to the **Job Sheets. Job Sheets** provide detailed information about *how to do the job* you are being trained in. Some **Job Sheets** will also have a series of **Activity Sheets**. These sheets have been designed to introduce you to the job step by step. This is where you will apply the new knowledge you gained by reading the Information Sheets. This is your opportunity to practice the job. You may need to practice the job or activity several times before you become competent.

5. Specification **sheets**, specifying the details of the job to be performed will be provided where appropriate.

6. A review of competency is provided on the last page to help remind if all the required assessment criteria have been met. This record is for your own information and guidance and is not an official record of competency

When working though this Module always be aware of your safety and the safety of others in the training room. Should you require assistance or clarification please consult your trainer or facilitator.

When you have satisfactorily completed all the Jobs and/or Activities outlined in this module, an assessment event will be scheduled to assess if you have achieved competency in the specified learning outcomes. You will then be ready to move onto the next Unit of Competency or Modul

Table of Content

Module Content

Unit of Competency	**Manage Database**
Unit Code	**OU-ICT-AMAD-04-L4-V1**
Module Title	**Managing Database**
Module Descriptor	This module discusses the aspects that must be given attention when Managing Database. It shows the knowledge and skills requirements for using of preferences & file system, working with basic database, operating SQLite Database, using lists and adapters and using content providers.
Nominal Hours	**40** Hours
Lerning Outcome	After completing the practice of the module, the trainees will be able to perform the following jobs: 1. Use of preferences & file system 2. Work with basic database 3. Operate SQLite Database 4. Use lists and adapters 5. Use content providers

Assessment Criteria:

1. Basic concept of shared preferences and file system in android is explained.
2. Saving and getting data in shared preference is performed.
3. Database is interpreted
4. Data type is defined
5. Table is created
6. Data manipulation is performed
7. SQLite and Database design is described.
8. Managing SQLite database is explained.
9. Table in SQLite is created.
10. CRUD operation in database is made.
11. Data operation from database using Android Applications made.
12. List is created.
13. List is used.
14. Collection of items using list is used.
15. Working with list in Android is performed.
16. Custom list is created using adapter.
17. Method to create content provider is explained.
18. Content provider is used.
19. Data from one process to another is passed.
20. Database operation is simplified.

Learning Outcome 1: Use Preferences & File System

Assessment Criteria

1. Basic concept of shared preferences and file system in android is explained.
2. Saving and getting data in shared preference is performed

Content:

1. Basic concept of shared preferences and file system in android
2. Saving and getting data in shared preference

Resources Required/ Conditions:

The trainees must be provided with the following:

- Handouts or reference materials/books/ CBLMs on the above stated contents
- PCs/printers or laptop/printer with internet access
- Digital projector and Screen
- Bond paper
- Ball pens/pencils and other office supplies and materials
- Relevant learning materials
- Workplace or simulated environment

Methodologies

- Lecture/discussion
- Demonstration/application
- Presentation
- Blended delivery methods

Assessment Methods

- Written test
- Demonstration
- Observation with checklist
- Oral questioning
- Portfolio

Learning Experience 1: Prepare For Troubleshoot

In order to achieve the objectives stated in this learning guide, you must perform the learning steps below. Beside each step are the resources or special instructions you will use to accomplish the corresponding activity.

Learning Steps	Resources Specific Instructions
1. Trainee will ask the instructor about Use of preferences & file system	1. Instructor will provide the learning materials "**Managing Database**"
2. Read the Information sheet/s	2. Information Sheet No: 1 Use of preferences & file system
3. Complete the Self-Checks & Check answer sheets.	3. Self-Check/s Self-Check No: 1 Use of preferences & file system Answer key No. 1 Use of preferences & file system
4. Read the Job Sheet and Specification Sheet and perform job	4. Job- Sheet No: 1- Use of preferences & file system Specification Sheet 1 – Use of preferences & file system

Information Sheet 1: Use Of Preferences & File System

Learning Objectives:

After completion of this information sheet, the learners will be able to:

1.1 Explain Basic concept of shared preferences and file system in android.
1.2 Perform Saving and getting data in shared preference

1.1 Basic concept of shared preferences and file system in android

One of the most Interesting Data Storage options Android provides its users is Shared Preferences. Shared Preferences is the way in which one can store and retrieve small amounts of primitive data as key/value pairs to a file on the device storage such as String, int, float, Boolean that make up your preferences in an XML file inside the app on the device storage. Shared Preferences can be thought of as a dictionary or a key/value pair. For example, you might have a key being "username" and for the value, you might store the user's username. And then you could retrieve that by its key (here username). You can have a simple shared preference API that you can use to store preferences and pull them back as and when needed. The shared Preferences class provides APIs for reading, writing, and managing this data. A sample GIF is given below to get an idea about what we are going to do in this article. The code for that has been given in both Java and Kotlin Programming Language for Android.

Shared Preferences are suitable for different situations. For example, when the user's settings need to be saved or to store data that can be used in different activities within the app. As you know, onPause() will always be called before your activity is placed in the background or destroyed, so for the data to be saved persistently, it's preferred to save it in onPause(), which could be restored in on Create () of the activity. The data stored using shared preferences are kept private within the scope of the application. However, shared preferences are different from that activity's instance state.

Shared Preferences different from Saved Instance State

Shared Preferences	Saved Instance State
Persist Data across user sessions, even if the app is killed and restarted, or the device is rebooted	Preserves state data across activity instances in the same user session.
Data that should be remembered across sessions, such as the user's preferred settings or game score.	Data that should not be remembered across sessions, such as the currently selected tab or current state of activity.
A common use is to store user preferences	A common use is to recreate the state after the device has been rotated

a. To Create Shared Preferences

The first thing we need to do is to create one shared preferences file per app. So name it with the package name of your app- unique and easy to associate with the app. When you want to get the values, call the getSharedPreferences() method. Shared Preferences provide modes of storing the data (private mode and public mode). It is for backward compatibility- use only MODE_PRIVATE to be secure.

public abstract SharedPreferences getSharedPreferences (String name, int mode)

This method takes two arguments, the first being the name of the SharedPreference(SP) file and the other is the context mode that we want to store our file in.

MODE_PUBLIC will make the file public which could be accessible by other applications on the device

MODE_PRIVATE keeps the files private and secures the user's data.

MODE_APPEND is used while reading the data from the SP file.

b. Nested classes of Shared Preferences

SharedPreferences.Editor: Interface used to write(edit) data in the SP file. Once editing has been done, one must commit() or apply() the changes made to the file.

SharedPreferences.OnSharedPreferenceChangeListener(): Called when a shared preference is changed, added, or removed. This may be called even if a preference is set to its existing value. This callback will be run on your main thread.

c. Following are the methods of Shared Preferences

contains(String key): This method is used to check whether the preferences contain a preference.

edit(): This method is used to create a new Editor for these preferences, through which you can make modifications to the data in the preferences and atomically commit those changes back to the SharedPreferences object.

getAll(): This method is used to retrieve all values from the preferences.
getBoolean(String key, boolean defValue): This method is used to retrieve a boolean value from the preferences.
getFloat(String key, float defValue): This method is used to retrieve a float value from the preferences.
getInt(String key, int defValue): This method is used to retrieve an int value from the preferences.
getLong(String key, long defValue): This method is used to retrieve a long value from the preferences.
getString(String key, String defValue): This method is used to retrieve a String value from the preferences.
getStringSet(String key, Set defValues): This method is used to retrieve a set of String values from the preferences.

registerOnSharedPreferencechangeListener(SharedPreferences.OnSharedPreferencechangeListener listener): This method is used to register a callback to be invoked when a change happens to a preference.
unregisterOnSharedPreferencechangeListener(SharedPreferences.OnSharedPreferencechangeListener listener): This method is used to unregister a previous callback.

Following is a sample byte code on how to write Data in Shared Preferences:

```
// Storing data into SharedPreferences
SharedPreferences sharedPreferences =
getSharedPreferences("MySharedPref",MODE_PRIVATE);

// Creating an Editor object to edit(write to the
file)
SharedPreferences.Editor myEdit =
sharedPreferences.edit();

// Storing the key and its value as the data fetched
from edittext
myEdit.putString("name", name.getText().toString());
myEdit.putInt("age",
Integer.parseInt(age.getText().toString()));

// Once the changes have been made, we need to
commit to apply those changes made,
// otherwise, it will throw an error
myEdit.commit();
```

Following is the sample byte code on how to read Data in Shared Preferences:

```
// Retrieving the value using its keys the file name
 must be same in both saving and retrieving the data
SharedPreferences sh =
getSharedPreferences("MySharedPref", MODE_APPEND);

// The value will be default as empty string because
for
 the very
// first time when the app is opened, there is
nothing to show
String s1 = sh.getString("name", "");
int a = sh.getInt("age", 0);

// We can then use the data
name.setText(s1);
age.setText(String.valueOf(a));
```

Saving and getting data in shared preference

Example to Demonstrate the use of Shared Preferences in Android
Below is the small demo for Shared Preferences. In this particular demo, there are two EditTexts, which save and retain the data entered earlier in them. This type of feature can be seen in applications with forms. Using Shared Preferences, the user will not have to fill in details again and again. Invoke the following code inside the activity_main.xml file to implement the UI:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity"
  tools:ignore="HardcodedText">

  <TextView
          android:id="@+id/textview"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_centerHorizontal="true"
          android:layout_marginTop="32dp"
          android:text="Shared Preferences Demo"
          android:textColor="@android:color/black"
          android:textSize="24sp" />

  <!--EditText to take the data from the user and save the data in SharedPreferences-->
  <EditText
          android:id="@+id/edit1"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:layout_below="@+id/textview"
          android:layout_marginStart="16dp"
          android:layout_marginTop="8dp"
          android:layout_marginEnd="16dp"
          android:hint="Enter your Name"
          android:padding="10dp" />

  <!--EditText to take the data from the user and save the data in SharedPreferences-->
  <EditText
          android:id="@+id/edit2"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:layout_below="@+id/edit1"
```

```
        android:layout_marginStart="16dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="16dp"
        android:hint="Enter your Age"
        android:inputType="number"
        android:padding="10dp" />
```

</RelativeLayout>

Working with the MainActivity file to handle the two of the EditText to save the data entered by the user inside the SharedPreferences.

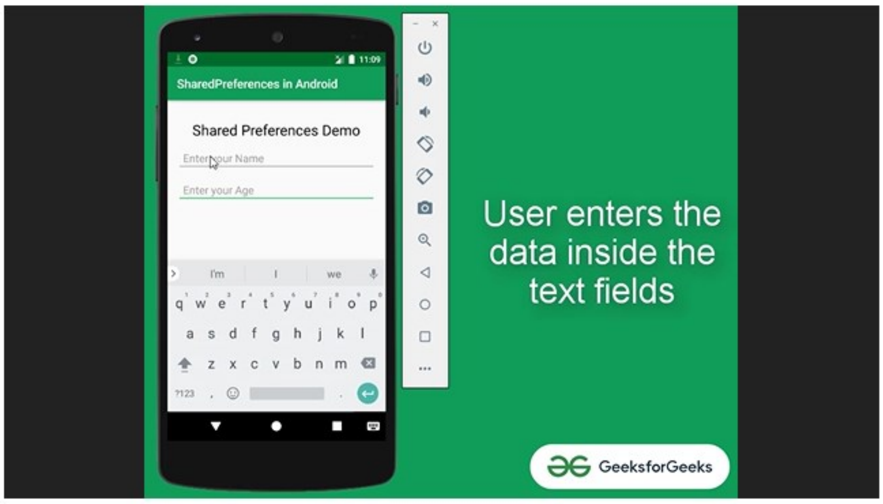
Below is the code for the MainActivity file. Comments are added inside the code to understand the code in more detail.

```
import androidx.appcompat.app.AppCompatActivity;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {
        private EditText name, age;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);
                name = findViewById(R.id.edit1);
                age = findViewById(R.id.edit2);
        }

        // Fetch the stored data in onResume() Because this is what will be called when
the app opens again
        @Override
        protected void onResume() {
                super.onResume();
                // Fetching the stored data from the SharedPreference
                SharedPreferences sh = getSharedPreferences("MySharedPref",
MODE_PRIVATE);
                String s1 = sh.getString("name", "");
                int a = sh.getInt("age", 0);

                // Setting the fetched data in the EditTexts
                name.setText(s1);
                age.setText(String.valueOf(a));
        }

        // Store the data in the SharedPreference in the onPause() method
```

```
	// When the user closes the application onPause() will be called and data will be
stored
	@Override
	protected void onPause() {
		super.onPause();
		// Creating a shared pref object with a file name "MySharedPref" in
private mode
		SharedPreferences		sharedPreferences		=
getSharedPreferences("MySharedPref", MODE_PRIVATE);
		SharedPreferences.Editor myEdit = sharedPreferences.edit();

		// write all the data entered by the user in SharedPreference and apply
		myEdit.putString("name", name.getText().toString());
		myEdit.putInt("age", Integer.parseInt(age.getText().toString()));
		myEdit.apply();
	}
}

import android.os.Bundle
import android.widget.EditText
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {
	private lateinit var name: EditText
	private lateinit var age: EditText

	override fun onCreate(savedInstanceState: Bundle?) {
		super.onCreate(savedInstanceState)
		setContentView(R.layout.activity_main)
		name = findViewById(R.id.edit1)
		age = findViewById(R.id.edit2)
	}

	// Fetch the stored data in onResume() Because this is what will be called when
the app opens again
	override fun onResume() {
		super.onResume()
		// Fetching the stored data from the SharedPreference
		val sh = getSharedPreferences("MySharedPref", MODE_PRIVATE)
		val s1 = sh.getString("name", "")
		val a = sh.getInt("age", 0)

		// Setting the fetched data in the EditTexts
		name.setText(s1)
		age.setText(a.toString())
```

```
        }

        // Store the data in the SharedPreference in the onPause() method
        // When the user closes the application onPause() will be called and data will be
stored
        override fun onPause() {
                super.onPause()
                // Creating a shared pref object with a file name "MySharedPref" in
private mode
                val sharedPreferences = getSharedPreferences("MySharedPref",
MODE_PRIVATE)
                val myEdit = sharedPreferences.edit()

                // write all the data entered by the user in SharedPreference and apply
                myEdit.putString("name", name.text.toString())
                myEdit.putInt("age", age.text.toString().toInt())
                myEdit.apply()
        }
}
```

Output:


SharedPreferences in Android
Shared Preferences Demo
Enter your Name
Enter your Age
User enters the data inside the text fields
GeeksforGeeks

Save simple data with SharedPreferences

If you have a relatively small collection of key-values that you'd like to save, you can use the SharedPreferences APIs. A SharedPreferences object points to a file containing key-value pairs and provides simple methods to read and write them. Each SharedPreferences file is managed by the framework and can be private or shared.

The SharedPreferences APIs are for reading and writing key-value pairs, and you shouldn't confuse them with the Preference APIs, which help you build a user interface for your app settings (although they also use SharedPreferences to save the user's settings).

Get a handle to shared preferences

You can create a new shared preference file or access an existing one by calling one of these methods:

getSharedPreferences(): Use this if you need multiple shared preference files identified by name, which you specify with the first parameter. You can call this from any Context in your app.

getPreferences(): Use this from an Activity if you need to use only one shared preference file for the activity. Because this retrieves a default shared preference file that belongs to the activity, you don't need to supply a name.

For example, the following code accesses the shared preferences file that's identified by the resource string R.string.preference_file_key and opens it using the private mode so the file is accessible by only your app:

Java

```
Context context = getActivity();

SharedPreferences sharedPref = context.getSharedPreferences(

getString(R.string.preference_file_key), Context.MODE_PRIVATE);
```

Kotlin

```
val sharedPref = activity?.getSharedPreferences(
    getString(R.string.preference_file_key), Context.MODE_PRIVATE)
```

When naming your shared preference files, you should use a name that's uniquely identifiable to your app. A good way to do this is prefix the file name with your application ID. For example: "com.example.myapp.PREFERENCE_FILE_KEY"

Alternatively, if you need just one shared preference file for your activity, you can use the getPreferences() method:

Kitlin

```
val sharedPref = activity?.getPreferences(Context.MODE_PRIVATE)
```

Java

```
SharedPreferences sharedPref =
getActivity().getPreferences(Context.MODE_PRIVATE);
```

If you're using the SharedPreferences API to save app settings, you should instead use getDefaultSharedPreferences() to get the default shared preference file for your entire app. For more information, see the Settings developer guide.

To write to a shared preferences file, create a SharedPreferences.Editor by calling edit() on your SharedPreferences.

Pass the keys and values you want to write with methods such as: putInt() andputString(). Then call apply() or commit() to save the changes. For example:

Java:

```
SharedPreferences sharedPref =
getActivity().getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putInt(getString(R.string.saved_high_score_key), newHighScore);
editor.apply();
```

Kotlin

```
val sharedPref = activity?.getPreferences(Context.MODE_PRIVATE) ?: return
with (sharedPref.edit()) {
   putInt(getString(R.string.saved_high_score_key), newHighScore)
   apply()
}
```

apply() changes the in-memory SharedPreferences object immediately but writes the updates to disk asynchronously. Alternatively, you can use commit() to write the data to disk synchronously. But because commit() is synchronous, you should avoid calling it from your main thread because it could pause your UI rendering.

Read from shared preferences

To retrieve values from a shared preferences file, call methods such as getInt() and getString(), providing the key for the value you want, and optionally a default value to return if the key isn't present. For example:

Kotlin

```
val sharedPref = activity?.getPreferences(Context.MODE_PRIVATE) ?: return
val defaultValue = resources.getInteger(R.integer.saved_high_score_default_key)
val highScore = sharedPref.getInt(getString(R.string.saved_high_score_key),
defaultValue)
```

Java

```
SharedPreferences sharedPref =
getActivity().getPreferences(Context.MODE_PRIVATE);
int defaultValue =
getResources().getInteger(R.integer.saved_high_score_default_key);
int highScore = sharedPref.getInt(getString(R.string.saved_high_score_key),
defaultValue);
```

Self-Check1: Use Of Preferences & File System

1. What are Shared Preferences in Android?
   Answer:

2. How do you save data in Shared Preferences?
   Answer:

3. How do you retrieve data from Shared Preferences?
   Answer:

4. What is the Android file system used for?
   Answer:

5. Can other apps access your Shared Preferences?
   Answer:

Answer Key 1: Use Of Preferences & File System

1. **What are Shared Preferences in Android?**
   Answer: Shared Preferences are a way to store small collections of key-value pairs in Android, which allows you to save simple application data such as configurations and settings.

2. **How do you save data in Shared Preferences?**
   Answer: To save data in Shared Preferences, you use a SharedPreferences.Editor to put key-value pairs and then call apply() or commit() to save the changes persistently.

3. **How do you retrieve data from Shared Preferences?**
   Answer: Data is retrieved from Shared Preferences using methods like getString() or getInt(), providing the key and a default value if the key doesn't exist.

4. **What is the Android file system used for?**
   Answer: The Android file system is used to store and manage data files. It provides a structured way to store persistent data that is larger and more complex than what is suitable for Shared Preferences.

5. **Can other apps access your Shared Preferences?**
   Answer: No, Shared Preferences are private to your application by default. Other apps cannot access your Shared Preferences unless you explicitly share them.

Activity Sheet 1.1: Android Shared Preferences and File System

Task 1: Understanding Shared Preferences

Task 1.1: Write a brief explanation of what Shared Preferences are and their common use cases.

Task 1.2: List the data types that can be stored in Shared Preferences.

Task 2: Implementing Shared Preferences

Task 2.1: Create a new Android project and implement Shared Preferences to save a user's username and login status.

Task 2.2: Retrieve and display the saved username and login status when the app restarts.

Task 3: Data Management

Task 3.1: Explain the difference between commit() and apply() when saving data in Shared Preferences.

Task 3.2: Write a method to clear all data stored in Shared Preferences.

Task 4: Exploring the File System

Task 4.1: Describe the Android file system structure and how it differs from Shared Preferences.

Task 4.2: Write a function to create a text file in the app's internal storage and write some data to it.

Task 5: Saving and Retrieving Data

Task 5.1: Implement a feature to save a high score in a game using Shared Preferences.

Task 5.2: Retrieve the high score when the app launches and display it on the screen.

Task 6: Security Considerations

Task 6.1: Discuss the security aspects of Shared Preferences and how to make them more secure.

Task 6.2: Research and summarize alternative methods for storing sensitive data in Android.

Task 7: Practical Application

Task 7.1: Create a settings screen for an app that uses Shared Preferences to save user preferences.

Task 7.2: Ensure the settings are applied throughout the app and persist even after the app is closed.

Task 8: Review and Reflect

Task 8.1: Review the code you've written for tasks 2 to 7 and refactor for efficiency and readability.

Task 8.2: Write a reflection on what you learned about Shared Preferences and the file system in Android.

Learning Outcome 2: Work With Basic Database

Assessment Criteria:

2.1 Database is interpreted
2.2 Data type is defined
2.3 Table is created
2.4 Data manipulation is performed

Content:

1. Database
2. Data type
3. Table
4. Data manipulation

    a. CRUD
        1. Create
        2. Retrieve/read
        3. Update
        4. Delete

    b. Join

Resources Required/ Conditions:
The trainees must be provided with the following:

- Handouts or reference materials/books/ CBLMs on the above stated contents
- PCs/printers or laptop/printer with internet access
- Digital projector and Screen
- Bond paper
- Ball pens/pencils and other office supplies and materials
- Relevant learning materials
- Workplace or simulated environment

Methodologies

- Lecture/discussion
- Demonstration/application
- Presentation
- Blended delivery methods

Assessment Methods

- Written test
- Demonstration
- Observation with checklist
- Oral questioning
- Portfolio

Learning Experience 2: Work With Basic Database

In order to achieve the objectives stated in this learning guide, you must perform the learning steps below. Beside each step are the resources or special instructions you will use to accomplish the corresponding activity.

Learning Steps	Resources specific instructions
1. Student will ask the instructor about Work with basic database	1. Instructor will provide the learning materials “**Managing Database**”
2. Read the Information sheet/s	2. Information Sheet No: 2 Work with basic database
3. Complete the Self Checks & Check answer sheets.	3. Self-Check/s Self-Check No: 2 - Work with basic database Answer key No. 2 - Work with basic database
4. Read the Job Sheet and Specification Sheet and perform job	4. Job- Sheet No: 2 - Work with basic database Specification Sheet: 2- Work with basic database

Information Sheet 2: Work With Basic Database

Learning Objectives:

After completion of this information sheet, the learners will be able to:

2.1 Interpret Database
2.2 Define Data type
2.3 Create Table
2.4 Perform Data manipulation

2.1 Database

A database is an organized collection of structured information, or data, typically stored electronically in a computer system. It is designed to efficiently store, retrieve, manage, and update data.

Database languages

Database languages, also known as query languages or data query languages, are specialized programming languages used to create, maintain, and query databases. They enable users to perform tasks such as defining data structures, manipulating data, and querying data within a database management system (DBMS).

2.2 Data Types

Data types are classifications that specify the different kinds of data that a variable can hold in programming languages. They are important because they help to ensure that data is handled in ways that are consistent with its nature.

A brief overview of common data types:

a. **Primitive Data Types:** These are the basic data types that are built into the language and represent single values.
b. **Integer:** Represents whole numbers without fractions, like 42 or -7.
c. **Float (Floating Point): Represents** numbers with fractional parts, like 3.14 or -0.001.
d. **Char (Character):** Represents single characters, like 'a', '4', or '@'.
e. **Boolean:** Represents truth values, typically true or false.
f. **Composite Data Types:** These are made up of primitive data types and represent a collection of values.

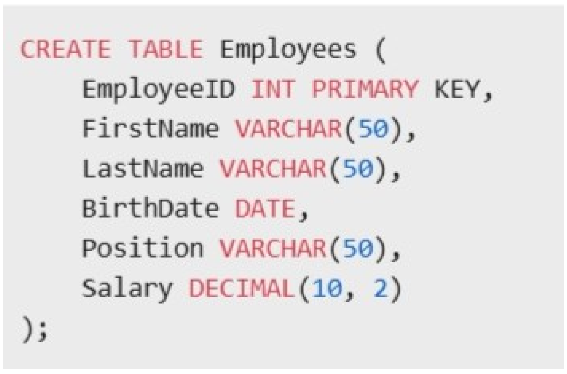
g. **Array:** A collection of elements of the same type, like an array of integers.
h. **Struct (Structure):** A collection of elements of potentially different types, grouped together.
i. **Reference Data Types**: These refer to memory locations where data is stored.
j. **String:** Represents a sequence of characters, like "Hello, World!".
k. **Pointer:** Holds the memory address of another variable.
l. **User-Defined Data Types:** These are defined by the programmer and can be structured as classes or objects in object-oriented languages.
Each data type has its own set of operations that can be performed on it, and choosing the correct data type is crucial for the efficiency and correctness of a program.

2.3 Table

Creating a table in a database is a fundamental task that involves defining a table’s structure, including its column names, data types, and any other constraints. Here’s a general explanation of how to create a table using SQL, which is the standard language for interacting with relational databases:

a. **Define the Table Name:** Choose a name for your table that reflects the data it will hold. The name should be unique within the database.
b. **Specify Columns and Data Types:** For each column in the table, you need to specify a name and a data type. The data type determines the kind of data the column can hold (e.g., VARCHAR for strings, INT for integers, DATE for dates).
c. **Set Constraints (Optional):** You can define rules for the data in each column, such as NOT NULL (the column must always have a value) or PRIMARY KEY (the column’s values uniquely identify each row in the table).

An example of a SQL statement that creates a table:



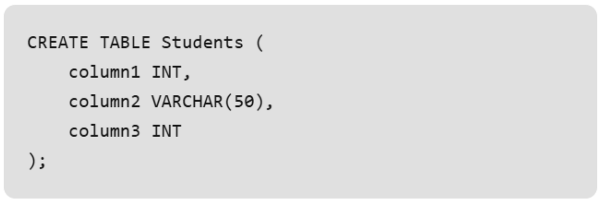
```sql
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    BirthDate DATE,
    Position VARCHAR(50),
    Salary DECIMAL(10, 2)
);
```

In this example, the Employees table has five columns: EmployeeID, FirstName, LastName, BirthDate, Position, and Salary. The EmployeeID column is an integer and is the primary key, meaning each employee will have a unique ID.

Four main categories of database languages:

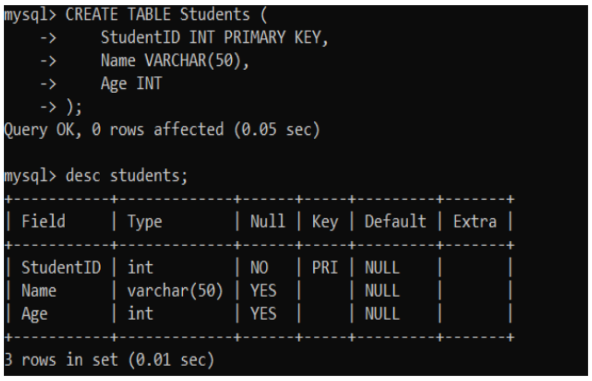
Data Definition Language (DDL): This type of language is used to define and modify the structure of database objects like tables, indexes, and schemas. Common DDL commands include CREATE, ALTER, DROP, RENAME, and TRUNCATE.

Syntax:



```
CREATE TABLE Students (
    column1 INT,
    column2 VARCHAR(50),
    column3 INT
);
```

Example:



```
mysql> CREATE TABLE Students (
    ->     StudentID INT PRIMARY KEY,
    ->     Name VARCHAR(50),
    ->     Age INT
    -> );
Query OK, 0 rows affected (0.05 sec)

mysql> desc students;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| StudentID | int         | NO   | PRI | NULL    |       |
| Name      | varchar(50) | YES  |     | NULL    |       |
| Age       | int         | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)
```

- Definition Language, Which is used for define the database's internal structure and Pattern of the Database.
- Basically, The DDL is used for creating tables, indexes, constraints, and schema in the Database.
- By using DDL statements we can able to create the architecture of the required database.

Below I list out types of database commands that are used in DDL. By using those queries we can able to perform the DDL on the Database.

- Create It is used to create objects in the database
- Alter It is used for change or alter the structure of the database objects
- Drop It is used for delete objects from the database
- Truncate It is used for remove all records from a table
- Rename It is used to rename the object in database
- Comment It is used for comment on the data dictionary.

Now I will explain each command with example for better understanding the concepts.

2.4 Create

It is one the command in DDL which is used for creating objects in database means creating Tables, Users, Triggers, functions and other objects. Here I will show how to create a table by using create command from DDL.

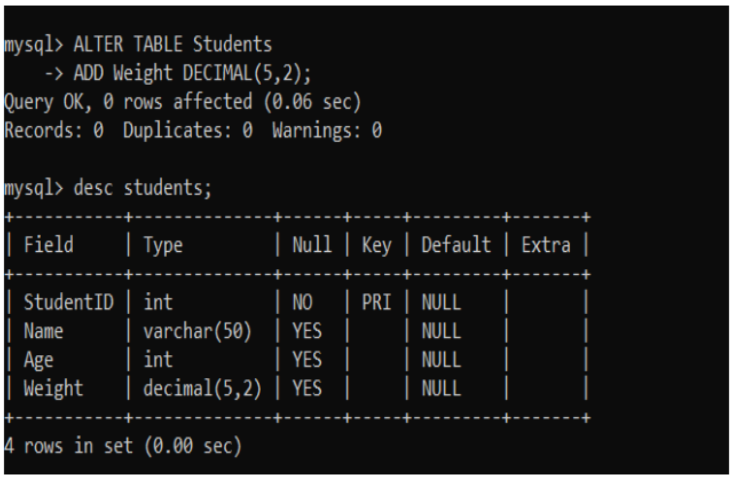
a. Alter

It is one the command in DDL which is used for change or alter the structure of the database or table. I already created students table with some columns you can see in the above image. Now by using Alter command I add new column that weight to the students table.

Syntax:

```
ALTER TABLE Students ADD column_name;
```

Example:



```
mysql> ALTER TABLE Students
    -> ADD Weight DECIMAL(5,2);
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc students;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| StudentID | int          | NO   | PRI | NULL    |       |
| Name      | varchar(50)  | YES  |     | NULL    |       |
| Age       | int          | YES  |     | NULL    |       |
| Weight    | decimal(5,2) | YES  |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

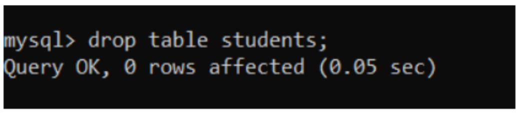
b. Drop

This Drop command is used for delete objects from the database. In this example I drop the students table

Syntax:

```
DROP Table Table_name;
```

Example:



```
mysql> drop table students;
Query OK, 0 rows affected (0.05 sec)
```

drop

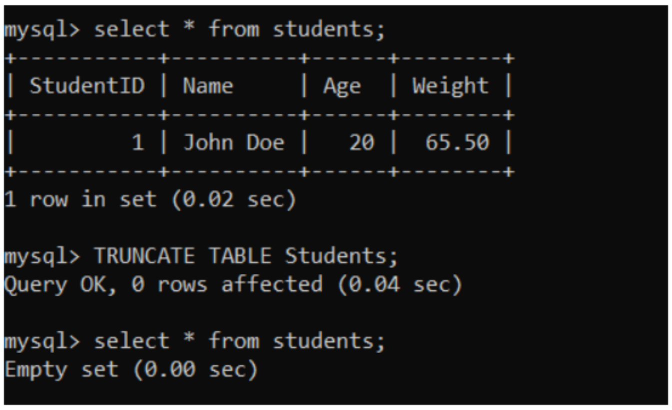
c. Truncate

Truncate command is used for remove all records from a table. Now I remove rows from the Student table.

Syntax:

```
TRUNCATE TABLE table_name;
```

Example:

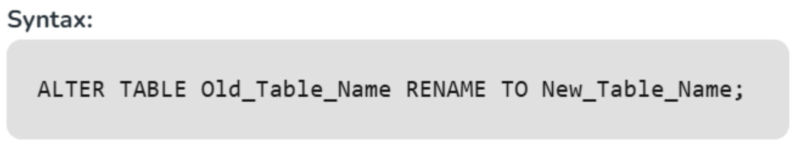


```
mysql> select * from students;
+-----------+----------+------+--------+
| StudentID | Name     | Age  | Weight |
+-----------+----------+------+--------+
|         1 | John Doe |   20 |  65.50 |
+-----------+----------+------+--------+
1 row in set (0.02 sec)

mysql> TRUNCATE TABLE Students;
Query OK, 0 rows affected (0.04 sec)

mysql> select * from students;
Empty set (0.00 sec)
```

truncate

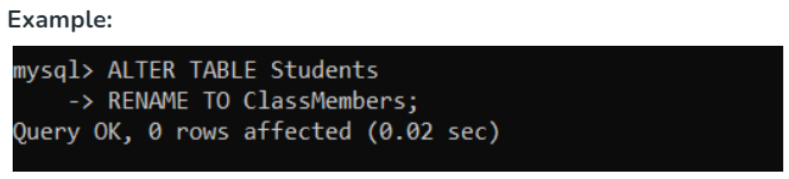
d. Rename

It is used to rename the object in database. Now I rename the table from students to ClassMembers. Below I provide the example for better understanding.



Syntax:

```
ALTER TABLE Old_Table_Name RENAME TO New_Table_Name;
```



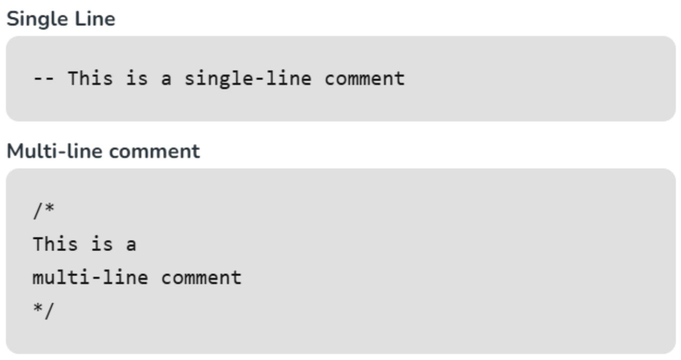
Example:

```
mysql> ALTER TABLE Students
    -> RENAME TO ClassMembers;
Query OK, 0 rows affected (0.02 sec)
```

Rename

e. Comment

It is used for comment on the data dictionary. We have two different comments namely Single line comment and Multi-line comments.



Single Line

```
-- This is a single-line comment
```

Multi-line comment

```
/*
This is a
multi-line comment
*/
```

2.5 Data Manipulation Language (DML)

DML deals with the manipulation of data itself. It includes commands like INSERT to add new data, UPDATE to modify existing data, DELETE to remove data, and SELECT to retrieve data from the database.

The Data Manipulation Language is used to Manipulate the data in the database by using different commands. In this category we can able to perform Insert new data into Table, Update existing data in the Table, Delete Data from the Table and other functions we can perform on data by using these DML commands. Below I listed those commands for your reference

- Select It is used for select data from the Table based on the requirements
- Insert It is sued for Inserting data into existing table
- Update It is used for update data in the Table based on the requirement
- Delete It is used for delete data from the Table

a. CRUD: Crud is an acronym for Create, Read, Update, and Delete, which are the four basic operations of persistent storage in database management systems. Here's a brief explanation of each operation:

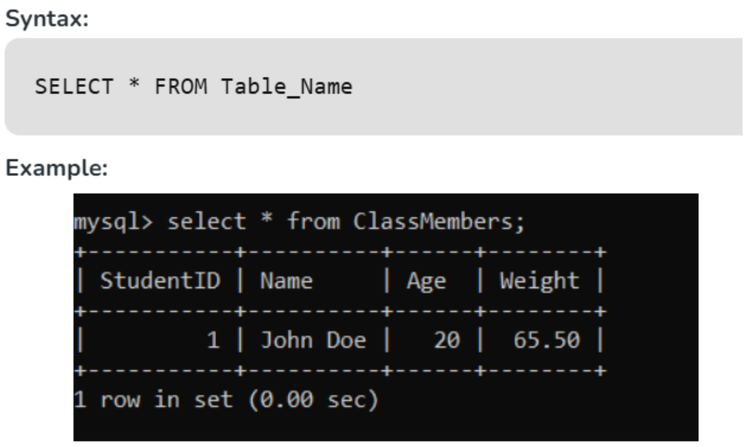
b. Create: This operation involves creating new records in a database. In SQL, the INSERT statement is used to add new rows to a table.

c. Read: This operation is used to retrieve data from a database. The SELECT statement in SQL allows you to query the database and retrieve rows that match specified criteria.

d. Update: This operation allows you to modify existing data in the database. The UPDATE statement in SQL is used to change the values of one or more columns in existing rows.

e. Delete: This operation is used to remove data from a database. The DELETE statement in SQL enables you to remove rows from a table based on certain conditions.

f. Select: The Select command is used for select required data based on conditions from a existing Table. Here I select all data from the ClassMembers Table.


Syntax:

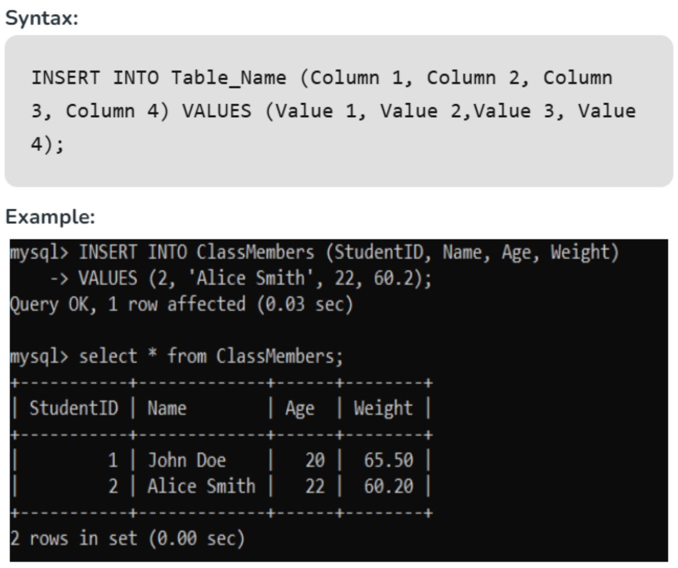
```
SELECT * FROM Table_Name
```

Example:

```
mysql> select * from ClassMembers;
+-----------+----------+------+--------+
| StudentID | Name     | Age  | Weight |
+-----------+----------+------+--------+
|         1 | John Doe |   20 |  65.50 |
+-----------+----------+------+--------+
1 row in set (0.00 sec)
```

Select Command

g. Insert: The Insert command is used for inserting new data into Table. Now I insert a new data into ClassMembers Table. Below I provide the example.


Syntax:

```
INSERT INTO Table_Name (Column 1, Column 2, Column
3, Column 4) VALUES (Value 1, Value 2,Value 3, Value
4);
```

Example:

```
mysql> INSERT INTO ClassMembers (StudentID, Name, Age, Weight)
    -> VALUES (2, 'Alice Smith', 22, 60.2);
Query OK, 1 row affected (0.03 sec)

mysql> select * from ClassMembers;
+-----------+-------------+------+--------+
| StudentID | Name        | Age  | Weight |
+-----------+-------------+------+--------+
|         1 | John Doe    |   20 |  65.50 |
|         2 | Alice Smith |   22 |  60.20 |
+-----------+-------------+------+--------+
2 rows in set (0.00 sec)
```

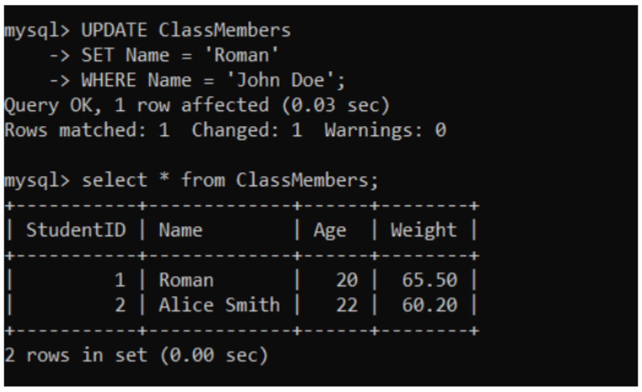
Insert command

h. Update: The Update command is used for update information In the Table. Now I will update name John Doe to Roman in the ClassMemebers Table. Below I provide that Example you can update any row or columns data.

Syntax:

```
UPDATE Table_Name SET Name = 'New_Value' WHERE Name
= 'Ola_Value';
```

Example:



```
mysql> UPDATE ClassMembers
    -> SET Name = 'Roman'
    -> WHERE Name = 'John Doe';
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from ClassMembers;
+-----------+-------------+------+--------+
| StudentID | Name        | Age  | Weight |
+-----------+-------------+------+--------+
|         1 | Roman       |   20 |  65.50 |
|         2 | Alice Smith |   22 |  60.20 |
+-----------+-------------+------+--------+
2 rows in set (0.00 sec)
```

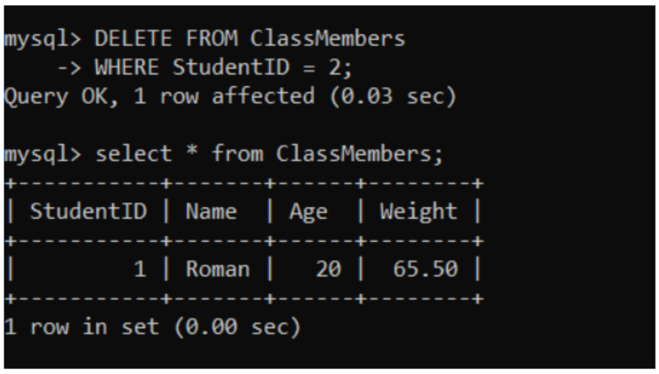
Update

i. **Delete:** The Delete command is used for delete data from the Table. Here I delete Student Id with 2 from the ClassMembers. Below I provide the Example for your reference.

Syntax:

```
DELETE FROM Table_Name WHERE Column  = Value;
```

Example:



```
mysql> DELETE FROM ClassMembers
    -> WHERE StudentID = 2;
Query OK, 1 row affected (0.03 sec)

mysql> select * from ClassMembers;
+-----------+-------+------+--------+
| StudentID | Name  | Age  | Weight |
+-----------+-------+------+--------+
|         1 | Roman |   20 |  65.50 |
+-----------+-------+------+--------+
1 row in set (0.00 sec)
```

Delete Command

j. **Merge/Join:** The Merge command is used for perform upsert operation means It inserts rows that doesn’t exist and updates rows that do.

Example:

```
MERGE INTO target_table AS target
USING source_table AS source
ON (target.id = source.id)
WHEN MATCHED THEN
    UPDATE SET target.name = source.name
WHEN NOT MATCHED THEN
    INSERT (id, name) VALUES (source.id,
source.name);
```

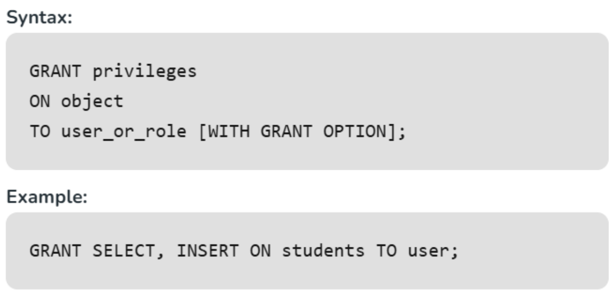
Data Control Language (DCL): DCL is used to control access to data in the database. It includes commands like GRANT, which gives users permissions, and REVOKE, which removes permissions.

The DCL stands for Data Control Language means these commands are used to retrieve the saved data from database. And one more thing is the DCL execution is Transactional that means It have roll back parameters. we have two tasks under the Data Control Language below I listed them

- Grant It is used for give user access to the database
- Revoke It is used for to take back the access or permissions from the user
- Now I will explain these commands with proper examples for better understanding

Grant

Basically, the grant command is used for provide database access to the new user. Here I create one user then I give the access to the database.



Syntax:

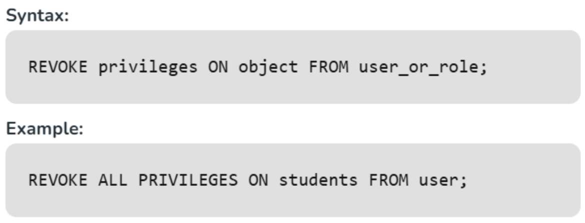
```
GRANT privileges
ON object
TO user_or_role [WITH GRANT OPTION];
```

Example:

```
GRANT SELECT, INSERT ON students TO user;
```

Revoke

The Revoke command is used to to take back database access from the user. Now I provide the example in the below



Syntax:

```
REVOKE privileges ON object FROM user_or_role;
```

Example:

```
REVOKE ALL PRIVILEGES ON students FROM user;
```

Transaction Control Language (TCL): TCL is used to manage transactions within a database. It includes commands like COMMIT to save a transaction, ROLLBACK to undo transactions, and SAVEPOINT to create points within groups of transactions where you can roll back.

Databases are used to store information. The Database is related to both software and hardware here The Software is used for accessing the data in the form of Software applications, and The Hardware is used for storing the data in the memory or hard disk.

Any Database provides an interface between the end user and the database by using this interface, the end user can access the database data. It is a very secure means before accessing database data we need to authenticate first, and then we get access to the database. We have different types of Database Languages which are represented in the below image.

Self-Check 2: Work With Basic Database

1. What is a database?

   Answer

2. Can you name a common database management system (DBMS)?

   Answer

3. What is a data type in the context of databases?

   Answer

4. Why is it important to define data types in a database table?

   Answer

5. How do you create a table in a SQL database?

   Answer

6. What does CRUD stand for in the context of data manipulation?

   Answer

7. How do you insert data into a table?

   Answer

8. What SQL statement would you use to change existing data in a table?

   Answer

9. How can you retrieve data from a database table?

   Answer

10. What command is used to remove data from a table?

    Answer

Answer Key 2: Work With Basic Database

1. **What is a database?**
   Answer: A database is an organized collection of data that is stored and accessed electronically from a computer system.

2. **Can you name a common database management system (DBMS)?**
   Answer: Yes, a common DBMS is MySQL. Other examples include PostgreSQL, Oracle Database, and Microsoft SQL Server.

3. **What is a data type in the context of databases?**
   Answer: A data type in databases is an attribute that specifies the type of data that a column can hold, such as integer, float, varchar, or date.

4. **Why is it important to define data types in a database table?**
   Answer: Defining data types is important because it ensures that the data is stored in a consistent format, which is crucial for data integrity and efficient querying.

5. **How do you create a table in a SQL database?**
   Answer: You create a table using the CREATE TABLE statement, followed by the table name and a list of columns with their respective data types and constraints.

6. **What does CRUD stand for in the context of data manipulation?**
   Answer: CRUD stands for Create, Read, Update, and Delete, which are the four basic operations you can perform on data in a database.

7. **How do you insert data into a table?**
   Answer: You insert data into a table using the INSERT INTO statement, followed by the table name and a list of values to add.

8. **What SQL statement would you use to change existing data in a table?**
   Answer: To change existing data, you would use the UPDATE statement, specifying the table, the columns to update, and the new values, along with a WHERE clause to select the rows to update.

9. **How can you retrieve data from a database table?**
   Answer: You can retrieve data using the SELECT statement, specifying the columns to retrieve and the table to query, with optional conditions to filter the results.

10. **What command is used to remove data from a table?**
    Answer: The DELETE FROM statement is used to remove data from a table, often accompanied by a WHERE clause to specify which rows should be deleted.

Task Sheet 2.1: Work With Basic Database

Task 1: Create Table

- Task 1.1: Write an SQL statement to create a table named Students with columns for StudentID, Name, Age, Major, and EnrollmentDate.
- Task 1.2: Add constraints to the Students table to ensure StudentID is a unique identifier and Age cannot be null.

Task 2: Perform Data Manipulation

- Task 2.1: Insert sample data into the Students table for five students.
- Task 2.2: Write an SQL query to update the Major of a student whose StudentID is 3.
- Task 2.3: Construct an SQL query to delete a student record where the Name is 'John Doe'.
- Task 2.4: Formulate an SQL query to select all students who are majoring in 'Computer Science'.

Task 3: Advanced Data Manipulation

- Task 3.1: Write an SQL statement to add a new column GPA to the Students table.
- Task 3.2: Create an SQL query to calculate the average age of all students in the Students table.
- Task 3.3: Develop an SQL script to create a view named CS_Majors that lists all students majoring in 'Computer Science'.

Task 4: Reflection

- Task 4.1: After completing the above tasks, reflect on the challenges you faced and how you overcame them.
- Task 4.2: Discuss how understanding these database concepts can be applied to a software development project you're interested in.

Learning Outcome 3: Operate Sqlite Database

Assessment Criteria:

3.1 SQLite and Database design is described.
3.2 Managing SQLite database is explained.
3.3 Table in SQLite is created.
3.4 CRUD operation in database is made.
3.5 Data operation from database using Android Applications made.

Content:

3.1 SQLite and Database design.
3.2 Managing SQLite database.
3.3 Table in SQLite.
3.4 CRUD operation in database.
3.5 Data operation

  5.1 Select
  5.2 Save
  5.3 Modify
  5.4 Edit
  5.5 Add
  5.6 Insert()
  5.7 Update()
  5.8 Delete()
  5.9 query()

Resources Required/ Conditions:

The trainees must be provided with the following:

- Handouts or reference materials/books/ CBLMs on the above stated contents
- PCs/printers or laptop/printer with internet access
- Digital projector and Screen
- Bond paper
- Ball pens/pencils and other office supplies and materials
- Relevant learning materials
- Workplace or simulated environment

Methodologies

- Lecture/discussion
- Demonstration/application
- Presentation
- Blended delivery methods

Assessment Methods

- Written test
- Demonstration
- Observation with checklist
- Oral questioning
- Portfolio

Learning Experience 3: Operate Sqlite Database

In order to achieve the objectives stated in this learning guide, you must perform the learning steps below. Beside each step are the resources or special instructions you will use to accomplish the corresponding activity.

Learning Steps	Resources specific instructions
1. Student will ask the instructor about Operate SQLite Database	1. Instructor will provide the learning materials "**Managing Database"**
2. Read the Information sheet/s	2. Information Sheet No 3: Operate SQLite Database
3. Complete the Self Checks & Check answer sheets.	3. Self-Check/s Self-Check No 3: Operate SQLite Database Answer key No. 3: Operate SQLite Database
4. Read the Job Sheet and Specification Sheet and perform job	Job- Sheet No 3-1: Operate SQLite Database Specification Sheet 3-1: Operate SQLite Database

Information Sheet 3: Operate Sqlite Database

Learning Objectives:

After completion of this information sheet, the learners will be able to:

3.1 Describe SQLite and Database design.
3.2 Explain Managing SQLite database.
3.3 Create Table in SQLite.
3.4 Make CRUD operation in database.
3.5 Make Data operation from database using Android Applications.

3.1 SQLite

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. It is a popular choice as an embedded database for local/client storage in application software such as web browsers. It is also used in many other applications that need a lightweight, embedded database.

SQLite is ACID-compliant and implements most of the SQL standards, using a dynamically and weakly typed SQL syntax that does not guarantee domain integrity.

To use SQLite in a C/C++ program, you can use the SQLite API, which provides a lightweight, simple, self-contained, high-reliability, full-featured, and SQL database engine. The API is implemented as a library of C functions that can be called from your program. One of the main benefits of using SQLite is that it is very easy to get started with. To create a new database in SQLite, you simply need to create a new file on your filesystem and connect to it using the SQLite API. For example, in C:

```
#include <sqlite3.h>

int main(int argc, char **argv) {
sqlite3 *db;
int rc;

rc = sqlite3_open("test.db", &db);
if (rc != SQLITE_OK) {
   // error handling
}

// do something with the database

sqlite3_close(db);
return 0;
}
```

A. **Some characteristics of SQLite**

a. **Ease of use:** SQLite is very easy to get started with, as it requires no setup or configuration. You can simply include the library in your project and start using it.

b. **Embeddability:** SQLite is designed to be embedded into other applications. It is a self-contained, serverless database engine, which means you can include it in your application without the need for a separate database server.

c. **Lightweight:** SQLite is a very lightweight database engine, with a small library size (typically less than 1MB). This makes it well-suited for use in applications where the database is embedded directly into the application binary, such as mobile apps.

d. **Serverless:** As mentioned earlier, SQLite is a serverless database engine, which means there is no need to set up and maintain a separate database server process. This makes it easy to deploy and manage, as there are no additional dependencies to worry about.

e. **Cross-platform:** SQLite is available on many platforms, including Linux, macOS, and Windows, making it a good choice for cross-platform development.

f. **Standalone:** SQLite stores all of the data in a single file on the filesystem, which makes it easy to copy or backup the database.

g. **High reliability:** SQLite has been widely tested and used in production systems for many years, and has a reputation for being a reliable and robust database engine.

B. **Database design principles**

database design principles, they are guidelines that help create efficient, reliable, and scalable databases.

a. **Avoid Redundancy**: Ensure that each piece of data is stored only once to prevent inconsistencies and save storage space6.
Use Primary Keys: Every table should have a primary key column that uniquely identifies each row.

b. **Handle Null Values:** Decide how to represent missing or unknown data and ensure the database can handle such cases appropriately.

c. **Ensure Referential Integrity:** Use foreign keys to enforce relationships between tables, ensuring that references are valid and data is consistent.

d. **Normalize Data:** Organize data to minimize duplication and dependency, which enhances data integrity and optimizes performance.

e. **Indexing:** Use indexes to speed up queries, especially for large datasets.

f. **Scalability:** Design the database to accommodate growth without compromising performance or integrity.

g. **Security:** Implement measures to control access to data and protect sensitive information.

h. **Maintainability**: Use clear naming conventions and keep documentation up-to-date to make the database easy to manage and modify

3.2 Managing an SQLite database

Managing an SQLite database involves several key tasks that ensure the database is efficient, reliable, and meets the needs of the application it supports. Here's an overview of the main aspects of managing an SQLite database:

Installation and Setup: Download and install SQLite tools on your computer. For many environments, SQLite comes pre-installed12.

a. **Database Creation:** Create a new SQLite database file using SQLite commands or through programming languages like Python. SQLite databases are simple files on your system2.
b. **Schema Design:** Define the structure of the database by creating tables with appropriate columns, data types, and constraints. SQLite uses dynamic typing and supports common data types like INTEGER, TEXT, BLOB, REAL, etc2.
   Data Manipulation: Use SQL commands to insert, update, delete, and query data. Familiarize yourself with key SQL operations like CREATE TABLE, INSERT INTO, SELECT, UPDATE, and DELETE2.
c. **Indexing:** Create indexes on columns to speed up queries. SQLite's optimizer automatically decides whether to use an index or not2.
d. **Transaction Management:** Ensure that all transactions are atomic, consistent, isolated, and durable (ACID-compliant) to maintain data integrity2.
e. **Backup and Recovery:** Regularly back up the database file and be prepared with a recovery plan in case of data loss.
f. **Performance Tuning:** Monitor the database's performance and optimize queries, indexing, and schema design as necessary.
g. **Security:** Implement security measures to protect the database file, especially if the application is distributed.
h. **Maintenance:** Regularly check the integrity of the database, vacuum to reclaim space and reorganize the database file structure for efficiency.
   SQLite is designed to be easy to manage, with a focus on simplicity and reliability. It's well-suited for applications where a full-scale RDBMS would be overkill, such as mobile apps, small to medium web projects, and desktop applications

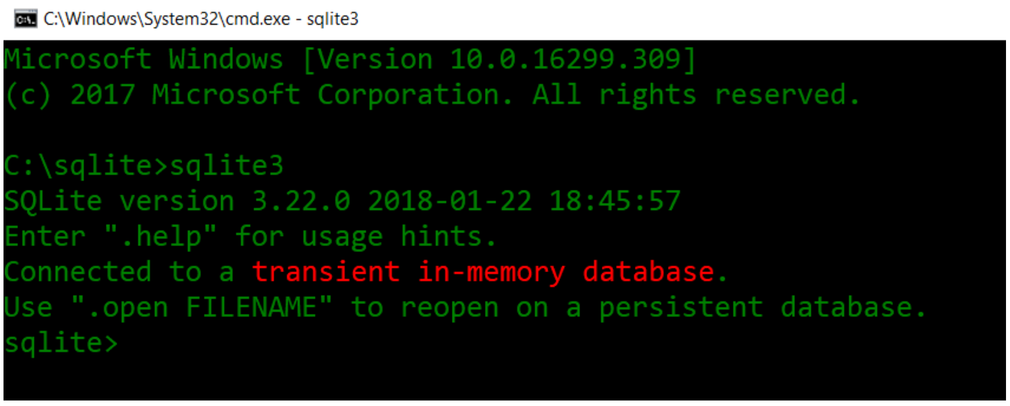
A. **Install SQLite:**
   SQLite is an embedded database, you actually don't need to 'download' it in the same way that you would download MySQL or PostgreSQL for example. You can create and interact with SQLite databases using a range of tools.

   An easy way to get started would be to:
   Download an example SQLite dataset, like the Sakila dataset, available here on GitHub
   Download a GUI program to access the database, like Beekeeper Studio, or DBeaver
   Double-click the '.db' file to open it.
   Using these tools, you can navigate your SQLite file in a spreadsheet-like way.

B. Installation on Windows:

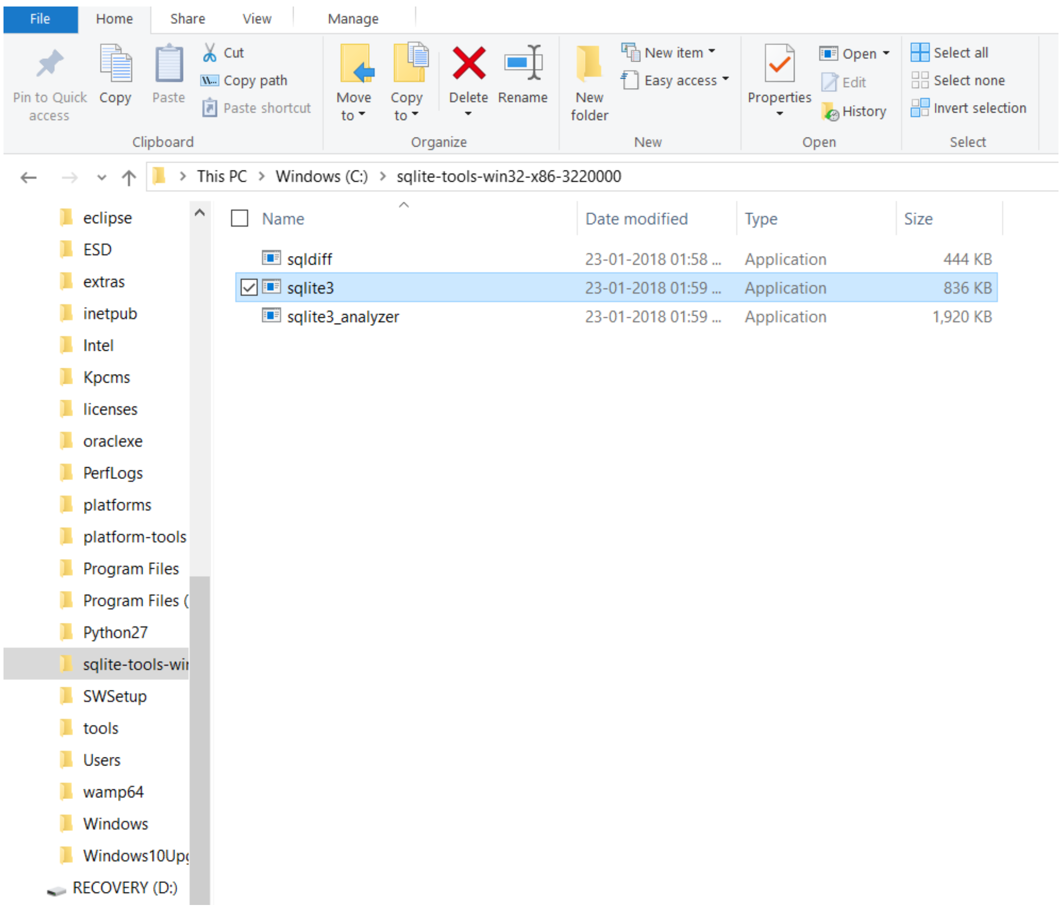
If you want to install the official SQLite binary and interact with the database using the terminal, you can follow these directions:

1. Visit the official website of SQLite to download the zip file.
2. Download that zip file.
3. Create a folder in C or D (wherever you want) for storing SQLite by expanding the zip file.
4. Open the command prompt and set the path for the location of the SQLite folder given in the previous step. After that write "sqlite3" and press enter.



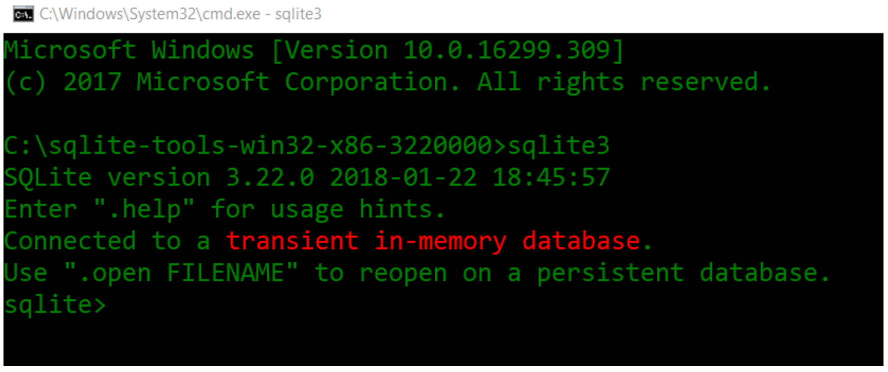
```
C:\Windows\System32\cmd.exe - sqlite3
Microsoft Windows [Version 10.0.16299.309]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\sqlite>sqlite3
SQLite version 3.22.0 2018-01-22 18:45:57
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

You can also directly open the .exe file from the folder where you have stored the SQLite whole thing.



Name	Date modified	Type	Size
sqldiff	23-01-2018 01:58 ...	Application	444 KB
sqlite3	23-01-2018 01:59 ...	Application	836 KB
sqlite3_analyzer	23-01-2018 01:59 ...	Application	1,920 KB

After clicking on the selected .exe file it will open SQLite application



```
Microsoft Windows [Version 10.0.16299.309]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\sqlite-tools-win32-x86-3220000>sqlite3
SQLite version 3.22.0 2018-01-22 18:45:57
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

C. Features of SQLite

1. The transactions follow ACID properties i.e. atomicity, consistency, isolation, and durability even after system crashes and power failures.
2. The configuration process is very easy, no setup or administration is needed.
3. All the features of SQL are implemented in it with some additional features like partial indexes, indexes on expressions, JSON, and common table expressions.
4. Sometimes it is faster than the direct file system I/O.
5. It supports terabyte-sized databases and gigabyte-sized strings and blobs.
6. Almost all OS supports SQLite like Android, BSD, iOS, Linux, Mac, Solaris, VxWorks, and Windows (Win32, WinCE, etc. It is very much easy to port to other systems.
7. A complete database can be stored in a single cross-platform disk file.

D. Applications of SQLite

1. Due to its small code print and efficient usage of memory, it is the popular choice for the database engine in cell phones, PDAs, MP3 players, set-top boxes, and other electronic gadgets.
2. It is used as an alternative for open to writing XML, JSON, CSV, or some proprietary format into disk files used by the application.
3. As it has no complication for configuration and easily stores file in an ordinary disk file, so it can be used as a database for small to medium sized websites.
4. It is faster and accessible through a wide variety of third-party tools, so it has great applications in different software platforms.

E. SQLite Commands

In SQLite, DDL (Data Definition Language) is used to create and modify database objects such as tables, indices, and views. Some examples of DDL statements in SQLite are:
CREATE TABLE: creates a new table in the database
ALTER TABLE: modifies an existing table in the database
DROP TABLE: deletes a table from the database
CREATE INDEX: creates a new index on a table
DROP INDEX: deletes an index from a table

DML (Data Modification Language) is used to modify the data stored in the database. Some examples of DML statements in SQLite are:

INSERT INTO: inserts a new row into a table
UPDATE: updates the data in one or more rows of a table
DELETE FROM: deletes one or more rows from a table
DQL (Data Query Language) is used to retrieve data from the database. Some examples of DQL statements in SQLite are:
SELECT: retrieves data from one or more tables in the database
JOIN: retrieves data from multiple tables based on a common field
GROUP BY: groups the results of a query by one or more fields
HAVING: filters the results of a query based on a condition

3.3 Creating a table in an SQLite database

Creating a table in an SQLite database involves using the CREATE TABLE SQL statement. Here's a step-by-step guide on how to create a basic table:

a. **Open SQLite Database:** Use an SQLite client or a programming language with SQLite support to open your database file.
b. **Write CREATE TABLE Statement:** Define the structure of your table including the table name, columns, data types, and any constraints.
c. **Execute the Statement:** Run the CREATE TABLE statement to create the table in your database.

Here's an example of a SQL statement that creates a simple table named Users:

SQL

```
CREATE TABLE IF NOT EXISTS Users (
   UserID INTEGER PRIMARY KEY AUTOINCREMENT,
   Username TEXT NOT NULL UNIQUE,
   Email TEXT NOT NULL,
   CreatedAt DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

In this example:

IF NOT EXISTS is used to create the table only if it doesn't already exist.
UserID is an integer column that automatically increments for each new row and serves as the primary key.
Username is a text column that must be unique for each user.
Email is a text column that cannot be null.
CreatedAt is a datetime column that defaults to the current timestamp when a new row is inserted.

3.4 CRUD operation in database.

The abbreviation CRUD expands to Create, Read, Update and Delete. These four are fundamental operations in a database. In the sample database, we will create it, and do some operations. Let's discuss these operations one by one with the help of examples.

A. **Create (Insert Data)**
   The create command is used to create the table in database. First we will go through its syntax then understand with an example.
   Syntax: CREATE TABLE table_name (Attr1 Type1, Attr2 Type2, … , Attrn Typen) ;

   To insert data into a table, use the INSERT INTO statement:
   INSERT INTO TableName (Column1, Column2) VALUES (Value1, Value2);
B. **Read (Query Data)**
   To read or select data from a table, use the SELECT statement:
   SELECT * FROM TableName WHERE condition;

C. **Update (Modify Data)**
   To update existing data in a table, use the UPDATE statement:

   UPDATE TableName SET Column1 = Value1 WHERE condition;

D. **Delete (Remove Data)**
   To delete data from a table, use the DELETE FROM statement:
   DELETE FROM TableName WHERE condition;

An example of how these operations can be performed on a table named Users:
Create: Insert a new user into the Users table.
SQL
INSERT INTO Users (Username, Email) VALUES ('JohnDoe', 'john@example.com');

Read: Select all users from the Users table.
SQL
SELECT * FROM Users;

Update: Change the email of a user with the username ‘JohnDoe’.
SQL
UPDATE Users SET Email = 'newjohn@example.com' WHERE Username = 'JohnDoe';

Delete: Remove the user with the username ‘JohnDoe’ from the Users table.
SQL
DELETE FROM Users WHERE Username = 'JohnDoe';
Data operation
Select (Query Data) To select or query data from a table:
SELECT column1, column2 FROM TableName WHERE condition;
Example:
SELECT Username, Email FROM Users WHERE UserID = 1;

Save (Insert Data) To save or insert data into a table:

SQL

INSERT INTO TableName (column1, column2) VALUES (value1, value2);

Example:

INSERT INTO Users (Username, Email) VALUES ('newuser', 'newuser@example.com');

Modify (Update Data) To modify or update existing data in a table:

UPDATE TableName SET column1 = value1 WHERE condition;

Example:

UPDATE Users SET Email = 'updateduser@example.com' WHERE Username = 'newuser';

Edit (Update Data) The command to edit data is the same as the modify command, which is the UPDATE statement.

Add (Insert Data) The command to add new data is the same as the save command, which is the INSERT INTO statement.

Insert: The INSERT statement is used to populate a table with rows:

INSERT INTO products VALUES (1, 'Keyboard', 15, 19.99);
INSERT INTO products (name, quantity, price) VALUES ('Mouse', 25, 9.99);
INSERT INTO users (name, email, country) VALUES ('John', 'john@example.com', 'UK');

The first INSERT statement specifies values for all columns in order.
Second statement lists specific columns to insert data into.
Use multiple INSERT statements one by one to populate the table with data.
Fetch the rows using SELECT to verify the inserted data.

UPDATE

Use the UPDATE statement to modify existing data in a table:

UPDATE products SET price = price * 1.1;
UPDATE users SET country = 'CA' WHERE name LIKE 'A%';

- SET clause specifies the columns to update and their new values.
- The optional WHERE clause picks which rows to update based on a condition.
- UPDATE statements can modify data in single or multiple rows at a time.
- Fetch and verify the updated rows using SELECT statement.

Make sure to add a WHERE clause to UPDATE/DELETE unless you wish to modify all rows in the table.

To delete data from SQLite tables, use the DELETE statement:

DELETE FROM products WHERE quantity = 0;
DELETE FROM users WHERE id IN (5, 18, 22);

- DELETE removes entire rows (all columns) from the table.
- Add a WHERE clause to pick which rows to delete.
- DELETE without WHERE will empty the entire table!

Use SELECT to verify that the rows have been deleted as expected.

Self-Check Sheet 3: Operate Sqlite Database

1. What is SQLite?

   Answer:

2. What are the core principles of database design?

   Answer:

3. How do you manage an SQLite database?
4. **Answer**:

5. What is the basic command to create a table in SQLite?

   Answer:

6. What does CRUD stand for, and how is it implemented in SQLite?
7. **Answer**:

8. How do you add a new record in an SQLite database?

   Answer:

9. How can you retrieve data from an SQLite database?
10. **Answer**:

11. What command would you use to modify data in an SQLite table?
12. **Answer**:

13. How do you delete a record from an SQLite database?
14. **Answer**:

15. How do Android applications perform data operations with an SQLite database?
16. **Answer**:

Answer Key 3: Operate SQLite Database

1. **What is SQLite?**
   Answer: SQLite is a lightweight, file-based database management system that's widely used for local/client storage in application development.

2. **What are the core principles of database design?**
   Answer: Core principles include defining clear objectives, ensuring data integrity and consistency, optimizing for performance, and planning for security and scalability.

3. **How do you manage an SQLite database?**
   Answer: Managing an SQLite database involves creating and structuring tables, performing CRUD operations, ensuring data integrity, and optimizing database performance.

4. **What is the basic command to create a table in SQLite?**
   Answer: The basic command is CREATE TABLE TableName (Column1 DataType, Column2 DataType, ...);.

5. **What does CRUD stand for, and how is it implemented in SQLite?**
   Answer: CRUD stands for Create, Read, Update, and Delete. In SQLite, these operations are implemented using INSERT, SELECT, UPDATE, and DELETE SQL statements, respectively.

6. **How do you add a new record in an SQLite database?**
   Answer: You can add a new record using the INSERT INTO TableName (Column1, Column2) VALUES (Value1, Value2); command.

7. **How can you retrieve data from an SQLite database?**
   Answer: Data can be retrieved using the SELECT * FROM TableName; command, which selects all columns from the specified table.

8. **What command would you use to modify data in an SQLite table?**
   Answer: To modify data, use the UPDATE TableName SET Column1 = Value1 WHERE condition; command.

9. **How do you delete a record from an SQLite database?**
   Answer: A record can be deleted using the DELETE FROM TableName WHERE condition; command.

10. **How do Android applications perform data operations with an SQLite database?**
    Answer: Android applications use the SQLiteOpenHelper class and SQLiteDatabase class to manage data operations like creating databases, executing SQL commands, and handling transactions.

Task Sheet 3.1: Operate Sqlite Database

Task 1: Create Table in SQLite

a. Write an SQL statement to create a table named Products with columns for ProductID, ProductName, Price, and Quantity.
b. Add constraints to the Products table to ensure ProductID is a unique identifier and ProductName cannot be null.

Task 2: Perform CRUD Operations in SQLite

a. Insert sample data into the Products table for three different products.
b. Write an SQL query to update the Price of a product where ProductID is 2.
c. Construct an SQL query to delete a product record where the Quantity is 0.
d. Formulate an SQL query to select all products with a Price greater than 50.

Task 3: Data Operations from Database Using Android Applications

a. Create an Android app with a simple UI to display the list of products from the Products table.
b. Implement functionality to add a new product to the database from the app interface.
c. Add buttons to each product entry to allow updating and deleting products from the database.
d. Ensure the app reflects the changes in the database in real-time.

Task 4: Advanced SQLite Features

a. Write an SQL statement to add a new column Category to the Products table.
b. Create an SQL query to calculate the total value of all products in stock (Price * Quantity).

Learning Outcome 4: Use Lists and Adapters

Assessment Criteria:
1. List is created.
2. List is used.
3. Collection of items using list is used.
4. Working with list in Android is performed.
5. Custom list is created using adapter.

Content:

1. List
2. Collection of items using list
3. Working with list in Android.
4. Custom list.

Resources Required/ Conditions:
The trainees must be provided with the following:

- Handouts or reference materials/books/ CBLMs on the above stated contents
- PCs/printers or laptop/printer with internet access
- Digital projector and Screen
- Bond paper
- Ball pens/pencils and other office supplies and materials
- Relevant learning materials
- Workplace or simulated environment

Methodologies

- Lecture/discussion
- Demonstration/application
- Presentation
- Blended delivery methods

Assessment Methods

- Written test
- Demonstration
- Observation with checklist
- Oral questioning
- Portfolio

Learning Experience 4: Use Lists and Adapters

In order to achieve the objectives stated in this learning guide, you must perform the learning steps below. Beside each step are the resources or special instructions you will use to accomplish the corresponding activity.

Learning Steps	Resources specific instructions
1. Student will ask the instructor about Use lists and adapters	1. Instructor will provide the learning materials "**Managing Database**"
2. Read the Information sheet/s	2. Information Sheet No: 4 Use lists and adapters
3. Complete the Self Checks & Check answer sheets.	3. Self-Check/s Self-Check No: 4 Use lists and adapters Answer key No. 4 Use lists and adapters
4. Read the Job Sheet and Specification Sheet and perform job	4. Job- Sheet No: 4 Use lists and adapters Specification Sheet: 4 Use lists and adapters

Information Sheet 4: Use Lists and Adapters

Learning Objectives:

After completion of this information sheet, the learners will be able to:

4.1 Create List
4.2 Use List
4.3 Use Collection of items using list
4.4 Perform Working with list in Android
4.5 Create Custom list using adapter.

4.1 List

In the context of an SQLite database, the term "list" can refer to a few different concepts.

A. **List of Databases:** When using the SQLite command-line interface, you can list all databases attached to the current connection with the .databases command. This command shows the names and file paths of each attached database. Additionally, the PRAGMA database_list; statement can be used to return a list of databases attached to the current database connection1.

B. **List of Tables:** Within an SQLite database, you can list all the tables using the .tables command in the SQLite command-line interface. This will display a list of all tables present in the database.

C. **List of Columns:** To list all columns within a specific table, you can use the PRAGMA table_info(table_name); command, where table_name is the name of the table whose columns you want to list. This command returns details about each column, such as the column name, data type, whether the column can be null, and other constraints.

D. **List of Rows:** In the context of querying data, a "list" often refers to the result set returned by a SELECT statement. For example, SELECT * FROM table_name; will list all rows from the specified table.

E. **List of Indexes:** You can list all indexes on a table with the PRAGMA index_list(table_name); command. This is useful for understanding the indexing structure of a table to optimize queries.
List of SQLite Commands: The .help command in the SQLite command-line interface provides a list of available SQLite dot commands that can be used to interact with the database.

F. **List in SQL Queries:** In SQL, the term "list" can also refer to a set of values provided in an IN clause, for example, SELECT * FROM table_name WHERE column_name IN (value1, value2, value3);. This query will return a list of rows where column_name matches any value in the list provided.

```
CREATE TABLE IF NOT EXISTS ItemList (
   ItemID INTEGER PRIMARY KEY AUTOINCREMENT,
   ItemName TEXT NOT NULL
);
```

In this example, ItemList is the name of the table, which is akin to a list. ItemID is a unique identifier for each item in the list, and ItemName is the name of the item. The AUTOINCREMENT keyword ensures that ItemID automatically increases with each new item added, similar to the index in a list.

To add items to this "list," you would use the INSERT statement:

```
INSERT INTO ItemList (ItemName) VALUES ('Item1');
INSERT INTO ItemList (ItemName) VALUES ('Item2');
INSERT INTO ItemList (ItemName) VALUES ('Item3');
```

Each INSERT statement adds a new item to the ItemList table. The ItemID is not specified because it auto-increments. This table now functions similarly to a list, with ItemID serving as the index and ItemName as the value

4.2 Using List

To use lists in SQLite for an Android application, you typically store the list items in a format that can be easily converted back and forth between a list and a format that SQLite can store, such as a string. Here are the general steps to achieve this:

1. **Convert List to String for Storage**: Before storing the list in SQLite, convert it into a single string. Common formats for this conversion include JSON, comma-separated values (CSV), or any other delimiter-based format.
2. **Store the String in SQLite**: Insert the string representation of the list into a column in your SQLite table.
3. **Retrieve the String from SQLite**: When you need the list, retrieve the string from the database.
4. **Convert String back to List**: Convert the retrieved string back to a list format in your application.

Example using JSON

an example demonstrating these steps in an Android application using JSON for the list conversion:

1. Adding dependencies

First, add the dependency for JSON processing. You can use org.json which is included in the Android SDK, or a library like Gson.

gradle

Copy code

```
// If you choose to use Gson, add this to your build.gradle
implementation 'com.google.code.gson:gson:2.8.6'
```

2. Convert List to JSON String

java

Copy code

```
import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;
import java.lang.reflect.Type;
import java.util.ArrayList;
import java.util.List;
```

```
// Method to convert a list to a JSON string
public String listToJson(List<String> list) {
   Gson gson = new Gson();
   return gson.toJson(list);
}
```

3. Store JSON String in SQLite

Assume you have a SQLite table with a column named list_column.

java

Copy code

```
SQLiteDatabase db = this.getWritableDatabase();
ContentValues values = new ContentValues();
values.put("list_column", listToJson(yourList)); // yourList is the list you want to store
db.insert("your_table", null, values);
```

4. Retrieve JSON String from SQLite

java

Copy code

```
SQLiteDatabase db = this.getReadableDatabase();
Cursor cursor = db.rawQuery("SELECT list_column FROM your_table WHERE id = ?", new
String[]{yourId});

if (cursor.moveToFirst()) {
   String listJson = cursor.getString(cursor.getColumnIndex("list_column"));
   List<String> yourList = jsonToList(listJson);
}
cursor.close();
```

5. Convert JSON String back to List

java

Copy code

```
// Method to convert a JSON string back to a list
public List<String> jsonToList(String jsonString) {
   Gson gson = new Gson();
   Type type = new TypeToken<ArrayList<String>>(){}.getType();
   return gson.fromJson(jsonString, type);
}
```

Example using CSV

Alternatively, you can use CSV format:

1. Convert List to CSV String

java

Copy code

```
public String listToCsv(List<String> list) {
   StringBuilder csvBuilder = new StringBuilder();
   for (String item : list) {
      csvBuilder.append(item).append(",");
   }
```

```
    return csvBuilder.toString();
}
```
2. Store CSV String in SQLite

java

Copy code

```
SQLiteDatabase db = this.getWritableDatabase();
ContentValues values = new ContentValues();
values.put("list_column", listToCsv(yourList)); // yourList is the list you want to store
db.insert("your_table", null, values);
```
3. Retrieve CSV String from SQLite

java

Copy code

```
SQLiteDatabase db = this.getReadableDatabase();
Cursor cursor = db.rawQuery("SELECT list_column FROM your_table WHERE id = ?", new
String[]{yourId});

if (cursor.moveToFirst()) {
    String listCsv = cursor.getString(cursor.getColumnIndex("list_column"));
    List<String> yourList = csvToList(listCsv);
}
cursor.close();
```
4. Convert CSV String back to List

java

Copy code

```
public List<String> csvToList(String csvString) {
    String[] items = csvString.split(",");
    return new ArrayList<>(Arrays.asList(items));
}
```
Considerations

- **Data Size**: If the lists are very large, consider breaking them into smaller chunks or using a more efficient storage method.
- **Database Schema**: Ensure your database schema is designed to handle these string columns efficiently.
- **Error Handling**: Always include error handling for database operations and string conversions to handle potential issues like null values or malformed strings.

4.3 Collection of items using list

In the context of an SQLite database, a "list" can refer to a collection of items stored in a table. an example of how you might use a list in SQLite:

Let's say you have a table named ShoppingList that stores items you need to buy at the grocery store. The table might look like this:

SQL

```
CREATE TABLE ShoppingList (
   ItemID INTEGER PRIMARY KEY AUTOINCREMENT,
   ItemName TEXT NOT NULL,
   Quantity INTEGER
);
```
AI-generated code. Review and use carefully. More info on FAQ.
To add items to your shopping list, you would use the INSERT command:

SQL

```
INSERT INTO ShoppingList (ItemName, Quantity) VALUES ('Apples', 4);
INSERT INTO ShoppingList (ItemName, Quantity) VALUES ('Bread', 1);
INSERT INTO ShoppingList (ItemName, Quantity) VALUES ('Milk', 2);
```
AI-generated code. Review and use carefully. More info on FAQ.
To view your shopping list, you would use the SELECT command:

SQL

```
SELECT * FROM ShoppingList;
```
AI-generated code. Review and use carefully. More info on FAQ.
This command will return a list of all items in your ShoppingList table, which you can use as your grocery shopping guide. The ItemID serves as a unique identifier for each item, ItemName is the name of the grocery item, and Quantity is the number of units you need to buy.

If you want to update the quantity of an item, say Apples, you would use the UPDATE command:

SQL

```
UPDATE ShoppingList SET Quantity = 6 WHERE ItemName = 'Apples';
```
AI-generated code. Review and use carefully. More info on FAQ.
And if you want to remove an item from the list after purchasing it, you would use the DELETE command:

SQL

```
DELETE FROM ShoppingList WHERE ItemName = 'Apples';
```
AI-generated code. Review and use carefully. More info on FAQ.
This is a simple example of how a list can be created and managed in an SQLite database. It demonstrates the CRUD operations—Create (INSERT), Read (SELECT), Update (UPDATE), and Delete (DELETE)—that are fundamental to interacting with databases.

Creating a custom list in an SQLite database typically involves designing a table schema that suits the specific needs of your application. Here’s an example of how you might create a custom list to manage a personal library:

SQL

```
CREATE TABLE LibraryList (
   BookID INTEGER PRIMARY KEY AUTOINCREMENT,
   Title TEXT NOT NULL,
   Author TEXT NOT NULL,
   Genre TEXT,
   Read BOOLEAN DEFAULT 0 -- 0 for not read, 1 for read
);
```

AI-generated code. Review and use carefully. More info on FAQ.
In this LibraryList table:

BookID is a unique identifier for each book.
Title is the name of the book.
Author is the name of the author.
Genre is the category of the book.
Read is a boolean flag indicating whether the book has been read.
To add books to your library list, you would use the INSERT command:

SQL

```
INSERT INTO LibraryList (Title, Author, Genre, Read) VALUES ('1984', 'George Orwell', 'Dystopian', 1);
INSERT INTO LibraryList (Title, Author, Genre) VALUES ('To Kill a Mockingbird', 'Harper Lee', 'Fiction');
```

AI-generated code. Review and use carefully. More info on FAQ.
To retrieve all books that you haven’t read yet, you could use the SELECT command with a WHERE clause:

SQL

```
SELECT * FROM LibraryList WHERE Read = 0;
```

AI-generated code. Review and use carefully. More info on FAQ.
This custom list can be extended with additional columns and features to suit more complex requirements, such as tracking loaned books, ratings, or reading progress. The key is to design the table structure to capture all the necessary details for your custom list.

4.4 Working with list in Android

Working with lists in Android involves using various components and techniques to display, manage, and interact with lists of data. The most common approach is using RecyclerView, which is a flexible and efficient way to display a large set of data.
Here’s an overview of working with lists in Android:
1. RecyclerView
RecyclerView is the recommended way to display lists in Android. It’s more efficient and flexible than the older ListView.
a. Add RecyclerView to Layout
First, add a RecyclerView to your layout XML file.
xml
Copy code

```
<androidx.recyclerview.widget.RecyclerView
   android:id="@+id/recyclerView"
   android:layout_width="match_parent"
   android:layout_height="match_parent"/>
```

b. Create a Data Model
Define a data class that represents the items in your list.
java
Copy code

```
public class Item {
   private String name;

   public Item(String name) {
      this.name = name;
   }

   public String getName() {
      return name;
   }
}
```

c. Create a RecyclerView Adapter
Create an adapter to bind your data to the RecyclerView.
java
Copy code

```
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;
import java.util.List;

public class ItemAdapter extends RecyclerView.Adapter<ItemAdapter.ViewHolder> {
    private List<Item> itemList;

    public ItemAdapter(List<Item> itemList) {
        this.itemList = itemList;
    }

    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType)
{
        View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.item_layout,
parent, false);
        return new ViewHolder(view);
    }

    @Override
    public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
        Item item = itemList.get(position);
        holder.textView.setText(item.getName());
    }

    @Override
    public int getItemCount() {
        return itemList.size();
    }

    public static class ViewHolder extends RecyclerView.ViewHolder {
        public TextView textView;

        public ViewHolder(View view) {
            super(view);
            textView = view.findViewById(R.id.textView);
        }
    }
}
```
d. Create a Layout for Each Item
Create an XML layout file for each item in the list (e.g., item_layout.xml).
xml
Copy code
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:padding="16dp">

    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="18sp" />
</LinearLayout>
```
e. Set Up RecyclerView in Activity or Fragment

Initialize the RecyclerView in your activity or fragment.

java

Copy code
```java
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import java.util.ArrayList;
import java.util.List;

public class MainActivity extends AppCompatActivity {

    private RecyclerView recyclerView;
    private ItemAdapter itemAdapter;
    private List<Item> itemList;

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

        recyclerView = findViewById(R.id.recyclerView);
        recyclerView.setLayoutManager(new LinearLayoutManager(this));

        itemList = new ArrayList<>();
        itemList.add(new Item("Item 1"));
        itemList.add(new Item("Item 2"));
        itemList.add(new Item("Item 3"));

        itemAdapter = new ItemAdapter(itemList);
        recyclerView.setAdapter(itemAdapter);
    }
}
```
2. Handling Item Clicks

To handle item clicks in a RecyclerView, you can modify the ViewHolder and Adapter to include a click listener.

a. Modify ViewHolder

java

Copy code
```java
public static class ViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
    public TextView textView;
    private OnItemClickListener onItemClickListener;
    public ViewHolder(View view, OnItemClickListener onItemClickListener) {
        super(view);
        textView = view.findViewById(R.id.textView);
        this.onItemClickListener = onItemClickListener;
        view.setOnClickListener(this);
    }

    @Override
    public void onClick(View view) {
        onItemClickListener.onItemClick(getAdapterPosition());
    }
}
```

b. Define an Interface for Click Listener

java

Copy code

```
public interface OnItemClickListener {
    void onItemClick(int position);
}
```

c. Modify Adapter to Accept Listener

java

Copy code

```
private OnItemClickListener onItemClickListener;

public ItemAdapter(List<Item> itemList, OnItemClickListener onItemClickListener) {
    this.itemList = itemList;
    this.onItemClickListener = onItemClickListener;
}

@Override
public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
    Item item = itemList.get(position);
    holder.textView.setText(item.getName());
    holder.onItemClickListener = onItemClickListener;
}
```

d. Implement Listener in Activity

java

Copy code

```
public class MainActivity extends AppCompatActivity implements OnItemClickListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // ... (same as before)

        itemAdapter = new ItemAdapter(itemList, this);
        recyclerView.setAdapter(itemAdapter);
    }

    @Override
    public void onItemClick(int position) {
        // Handle item click
        Item clickedItem = itemList.get(position);
        Toast.makeText(this, "Clicked: " + clickedItem.getName(), Toast.LENGTH_SHORT).show();
    }
}
```

3. Updating the List

To update the list dynamically, modify the data set and notify the adapter.

java

Copy code

```
public void addItem(Item newItem) {
    itemList.add(newItem);
    itemAdapter.notifyItemInserted(itemList.size() - 1);
}

public void removeItem(int position) {
    itemList.remove(position);
    itemAdapter.notifyItemRemoved(position);
}
```

4. Using ListView (Deprecated)

Though RecyclerView is recommended, here's a brief look at ListView for completeness.

a. Add ListView to Layout

xml

Copy code

```
<ListView
    android:id="@+id/listView"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

b. Create an ArrayAdapter

java

Copy code

```
ArrayAdapter<String> adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1, yourStringList);
ListView listView = findViewById(R.id.listView);
listView.setAdapter(adapter);
```

c. Handle Item Clicks

java

Copy code

```
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        String item = yourStringList.get(position);
        Toast.makeText(MainActivity.this, "Clicked: " + item, Toast.LENGTH_SHORT).show();
    }
});
```

4.5 Create Custom list using adapter.

Creating a custom list using an adapter in Android involves defining a custom layout for your list items and creating a custom adapter to bind your data to the list. This allows you to display complex data structures in a RecyclerView or ListView.

Steps to Create a Custom List Using an Adapter

1. **Define the Data Model**
2. **Create a Custom Layout for List Items**
3. **Create a Custom Adapter**
4. **Set Up the RecyclerView or ListView in Your Activity or Fragment**
5. **Handle Item Clicks (Optional)**

1. Define the Data Model

First, create a class to represent the data you want to display in the list.

java

Copy code

```
public class CustomItem {
    private String title;
    private String description;
    private int imageResourceId;

    public CustomItem(String title, String description, int imageResourceId) {
        this.title = title;
        this.description = description;
        this.imageResourceId = imageResourceId;
    }

    public String getTitle() {
        return title;
    }
    public String getDescription() {
        return description;
    }

    public int getImageResourceId() {
        return imageResourceId;
    }
}
```

2. Create a Custom Layout for List Items

Create an XML layout file for the list items (e.g., item_custom.xml).

xml

Copy code

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:padding="16dp">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:src="@drawable/ic_launcher_foreground" />

    <TextView
        android:id="@+id/titleTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:textStyle="bold" />

    <TextView
        android:id="@+id/descriptionTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="14sp" />
</LinearLayout>
```

3. Create a Custom Adapter

Create a custom adapter to bind your data to the RecyclerView.

java

Copy code

```
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;
import java.util.List;

public class CustomAdapter extends RecyclerView.Adapter<CustomAdapter.ViewHolder> {

    private List<CustomItem> itemList;
    public CustomAdapter(List<CustomItem> itemList) {
        this.itemList = itemList;
    }

    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.item_custom,
parent, false);
        return new ViewHolder(view);
    }

    @Override
    public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
        CustomItem item = itemList.get(position);
        holder.titleTextView.setText(item.getTitle());
        holder.descriptionTextView.setText(item.getDescription());
        holder.imageView.setImageResource(item.getImageResourceId());
    }

    @Override
    public int getItemCount() {
        return itemList.size();
    }

    public static class ViewHolder extends RecyclerView.ViewHolder {
        public ImageView imageView;
        public TextView titleTextView;
        public TextView descriptionTextView;

        public ViewHolder(View view) {
            super(view);
            imageView = view.findViewById(R.id.imageView);
            titleTextView = view.findViewById(R.id.titleTextView);
            descriptionTextView = view.findViewById(R.id.descriptionTextView);
        }
    }
}
```

4. Set Up the RecyclerView in Your Activity or Fragment

Initialize the RecyclerView and set the custom adapter.

java

Copy code

```
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import java.util.ArrayList;
import java.util.List;

public class MainActivity extends AppCompatActivity {

    private RecyclerView recyclerView;
    private CustomAdapter customAdapter;
    private List<CustomItem> itemList;

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

        recyclerView = findViewById(R.id.recyclerView);
        recyclerView.setLayoutManager(new LinearLayoutManager(this));

        itemList = new ArrayList<>();
        itemList.add(new CustomItem("Title 1", "Description 1", R.drawable.image1));
        itemList.add(new CustomItem("Title 2", "Description 2", R.drawable.image2));
        itemList.add(new CustomItem("Title 3", "Description 3", R.drawable.image3));

        customAdapter = new CustomAdapter(itemList);
        recyclerView.setAdapter(customAdapter);
    }
}
```

5. Handle Item Clicks (Optional)

To handle item clicks in a RecyclerView, you can modify the ViewHolder and Adapter to include a click listener.

java

Copy code

```
public class CustomAdapter extends RecyclerView.Adapter<CustomAdapter.ViewHolder> {

    private List<CustomItem> itemList;
    private OnItemClickListener onItemClickListener;

    public interface OnItemClickListener {
        void onItemClick(int position);
    }

    public void setOnItemClickListener(OnItemClickListener onItemClickListener) {
        this.onItemClickListener = onItemClickListener;
    }

    public CustomAdapter(List<CustomItem> itemList) {
        this.itemList = itemList;
    }

    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.item_custom,
parent, false);
        return new ViewHolder(view, onItemClickListener);
    }

    @Override
    public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
        CustomItem item = itemList.get(position);
        holder.titleTextView.setText(item.getTitle());
        holder.descriptionTextView.setText(item.getDescription());
        holder.imageView.setImageResource(item.getImageResourceId());
    }

    @Override
    public int getItemCount() {
        return itemList.size();
    }


    public static class ViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
        public ImageView imageView;
        public TextView titleTextView;
        public TextView descriptionTextView;
        OnItemClickListener onItemClickListener;

        public ViewHolder(View view, OnItemClickListener onItemClickListener) {
            super(view);
            imageView = view.findViewById(R.id.imageView);
            titleTextView = view.findViewById(R.id.titleTextView);
            descriptionTextView = view.findViewById(R.id.descriptionTextView);
            this.onItemClickListener = onItemClickListener;
            view.setOnClickListener(this);
        }

        @Override
        public void onClick(View view) {
            onItemClickListener.onItemClick(getAdapterPosition());
        }
    }
}
```

Then, implement the OnItemClickListener in your activity or fragment.

java

Copy code

```
public class MainActivity extends AppCompatActivity implements CustomAdapter.OnItemClickListener {


   @Override
   protected void onCreate(Bundle savedInstanceState) {
      // ... (same as before)


      customAdapter = new CustomAdapter(itemList);
      customAdapter.setOnItemClickListener(this);
      recyclerView.setAdapter(customAdapter);
   }


   @Override
   public void onItemClick(int position) {
      CustomItem clickedItem = itemList.get(position);
      Toast.makeText(this, "Clicked: " + clickedItem.getTitle(), Toast.LENGTH_SHORT).show();
   }
}
```

Self-Check Sheet 4: Use Lists And Adapters

1. How do you create a simple list in Android?
   Answer:

2. What is an Array Adapter in Android?
   Answer:

3. How can you use a custom object in a list?
   Answer:

4. What is the role of the View Holder pattern in lists?
   Answer:

5. How do you handle item clicks in a List View?
   Answer:

6. What is a Recycler View and how is it different from List View?
   Answer:

7. How do you add or remove items from a list in Android?
   Answer:

8. What is a Collection in Android?
   Answer:

9. How do you sort a list of items in Android?
10. **Answer**:

11. What is the purpose of a custom adapter in Android?

Answer:

`Answer Key 4: Use Lists and Adapters

1. **How do you create a simple list in Android?**
   Answer: You create a simple list in Android using the List View widget and an Array Adapter to bind array data to the List View.

2. **What is an ArrayAdapter in Android?**
   Answer: An ArrayAdapter connects an array of data to a ListView or Spinner, converting each array item into a view by calling toString() on the item and placing the result in a TextView.

3. **How can you use a custom object in a list?**
   Answer: To use a custom object in a list, you need to create a custom adapter that extends BaseAdapter or ArrayAdapter and override the getView() method to define how the data is displayed1.

4. **What is the role of the ViewHolder pattern in lists?**
   Answer: The ViewHolder pattern increases the performance of lists by avoiding unnecessary findViewById() calls when recycling views in a ListView.

5. **How do you handle item clicks in a ListView?**
   Answer: Item clicks in a ListView are handled by setting an onItemClickListener to the ListView and overriding the onItemClick() method.

6. **What is a RecyclerView and how is it different from ListView?**
   Answer: A RecyclerView is a more advanced and flexible version of ListView that supports better item animations, layout transitions, and allows for more complex item layouts.

7. **How do you add or remove items from a list in Android?**
   Answer: Items can be added or removed from a list by modifying the underlying data set and notifying the adapter with notifyDataSetChanged().

8. **What is a Collection in Android?**
   Answer: A Collection in Android is a group of single or multiple types of objects represented as a single unit. You can use Java collections like ArrayList or HashSet in Android.

9. **How do you sort a list of items in Android?**
   Answer: You can sort a list of items by using Collections.sort() with a custom Comparator to define the sorting logic2.
10. **What is the purpose of a custom adapter in Android?**
   Answer: A custom adapter allows you to control the data model and the way it's presented in a ListView or RecyclerView, enabling custom row layouts and data binding

Task Sheet 4.1: Use Lists And Adapters

Task 01: Create a Basic List
Task 01: Implement a ListView using ArrayAdapter.
Goal: Display a simple list of strings.
Expected Outcome: A ListView that shows an array of predefined strings.

Use a List with Custom Objects
Task 02: Define a custom class with at least two attributes and display objects of this class in a ListView.
Goal: Learn to display complex data in lists.
Expected Outcome: A ListView that shows custom objects with more than one detail per item.

Implement a Collection of Items Using a List
Task 03: Use a Java ArrayList to manage a dynamic collection of items.
Goal: Understand how to add and remove items from a list at runtime.
Expected Outcome: Ability to modify the list contents during app execution.

Perform Operations with a List
Task 04: Implement sorting and searching within a list.
Goal: Learn to manipulate list data programmatically.
Expected Outcome: A ListView where items can be sorted and searched.
Create a Custom List Using an Adapter
Task 05: Create a custom ArrayAdapter to change the appearance of list items.
Goal: Customize how list items are displayed.
Expected Outcome: A ListView with a custom layout for items.

Task 06: Extend BaseAdapter to create a more complex adapter for your list.
Goal: Gain deeper control over list data and presentation.
Expected Outcome: A ListView or RecyclerView with complex data binding and custom row types.

Learning Outcome 5: Use Content Providers

Assessment Criteria:

1. Method to create content provider is explained.
2. Content provider is used.
3. Data from one process to another is passed.
4. Database operation is simplified

Content:

1. Method to create content provider.
2. Data from one process to another.
3. Simplifying Database operation

Resources Required/ Conditions:

The trainees must be provided with the following:

- Handouts or reference materials/books/ CBLMs on the above stated contents
- PCs/printers or laptop/printer with internet access
- Digital projector and Screen
- Bond paper
- Ball pens/pencils and other office supplies and materials
- Relevant learning materials
- Workplace or simulated environment

Methodologies

- Lecture/discussion
- Demonstration/application
- Presentation
- Blended delivery methods

Assessment Methods

- Written test
- Demonstration
- Observation with checklist
- Oral questioning
- Portfolio

Learning Experience 5: Use Content Providers

In order to achieve the objectives stated in this learning guide, you must perform the learning steps below. Beside each step are the resources or special instructions you will use to accomplish the corresponding activity.

Learning Steps	Resources specific instructions
1. Student will ask the instructor about Use content providers	1. Instructor will provide the learning materials "**Managing Database**"
2. Read the Information sheet/s	2. Information Sheet No: 5 Use content providers
3. Complete the Self Checks & Check answer sheets.	3. Self-Check/s Self-Check No: 5 Use content providers Answer key No. 5 Use content providers
4. Read the Job Sheet and Specification Sheet and perform job	4. Job- Sheet No: 5 Use content providers Specification Sheet: 5 Use content providers

Information Sheet 5: Use content providers

Learning Objectives:

After completion of this information sheet, the learners will be able to:

5.1 Explain Method to create content provider.
5.2 Use Content provider.
5.3 Pass Data from one process to another.
5.4 Simplify Database operation.

5.1 Method to create content provider

Creating a content provider in Android using an SQLite database involves several steps. Here's a high-level overview of the method:

A. Define a Contract Class:

Create a contract class that defines the structure of your data schema, including URIs for the content provider and column names for the database tables.

B. Create a Database Helper:

Implement a subclass of SQLiteOpenHelper that creates and manages the database itself.

5.2 Implement the Content Provider:

Extend the ContentProvider class and implement the abstract methods: onCreate(), query(), insert(), update(), delete(), and getType().

In the onCreate() method, initialize your database helper to access the SQLite database.

The query(), insert(), update(), and delete() methods will interact with the SQLite database using the SQLiteDatabase object obtained from your database helper.

A. Register the Content Provider:

In your application's manifest file (AndroidManifest.xml), declare the <provider> element and specify the authority, which is a unique string that identifies the content provider.

B. Use URIs to Perform Operations:

Define URIs that the content provider can handle and use a UriMatcher to match the URIs to operations in the content provider methods.

For example, to insert data, you would use the getContentResolver().insert() method with the appropriate URI and ContentValues.

C. Notify Changes:

When you perform insert, update, or delete operations, call getContext().getContentResolver().notifyChange() to notify any observers that the data has changed.

D. Example of a content provider implementation:

Java

```
public class MyContentProvider extends ContentProvider {
    MyDatabaseHelper dbHelper;

// Initialize your provider and the database helper object
```

```
    @Override
    public boolean onCreate() {
        dbHelper = new MyDatabaseHelper(getContext());
        return true;
    }

    // Implement query() method
    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
                        String[] selectionArgs, String sortOrder) {
        // Code to query the database
    }

    // Implement insert() method
    @Override
    public Uri insert(Uri uri, ContentValues values) {
        // Code to insert data into the database
    }

    // Implement update() method
    @Override
    public int update(Uri uri, ContentValues values, String selection,
                      String[] selectionArgs) {
        // Code to update the database
    }

    // Implement delete() method
    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        // Code to delete data from the database
    }

    // Implement getType() method
    @Override
    public String getType(Uri uri) {
        // Code to return the MIME type of data
    }
}
```

In the context of Android development, content providers are a key component used to pass data from one process to another. They encapsulate the data and provide a standardized interface for other apps to perform data operations securely, even across different applications running in separate processes.

Here's how data is passed between processes using content providers:

Define a Content Provider: A content provider is created by extending the ContentProvider class and implementing its core methods. This provider will manage access to the underlying data storage, whether it's an SQLite database, files, or other forms.
Declare the Content Provider in the Manifest: The content provider must be declared in the AndroidManifest.xml file with an <provider> tag, specifying an authority which is a unique name that identifies the content provider to the system.
Use URIs to Access Data: Content providers are accessed via URIs (Uniform Resource Identifiers). Each URI corresponds to a table or a specific record in the content provider. The structure of a content URI typically follows the format: content://authority/path/id, where:
content:// is the scheme,
authority is the name of the content provider,
path specifies the type of data, and
id is used to access a particular record.
Perform CRUD Operations: Other applications use an instance of ContentResolver to perform create, read, update, and delete (CRUD) operations on the content provider's data. The ContentResolver communicates with the content provider to carry out these operations.
Data Security: Content providers offer granular control over data permissions. You can configure different permissions for reading and writing data, ensuring that only authorized components can access or modify the data.
Data Sharing: When a request is made to a content provider from another app, the content provider processes the request and returns the result. This allows for secure data sharing between apps without exposing the underlying data storage mechanism.
Notifications on Data Changes: Content providers can notify observers of changes to the data, allowing UI components like CursorLoader to update automatically when the underlying data changes.
Content providers are particularly useful when you need to share data between multiple applications or when you're dealing with complex data that requires a structured approach to access and modify.

This mechanism ensures that data can be passed back and forth between processes in a secure, modular, and scalable way, adhering to Android's security best practices.

5.3 Pass Data from one process to another

Passing data between processes in Android is often referred to as Inter-Process Communication (IPC). There are several methods available, each suitable for different scenarios. Here's an overview of the main methods:
1. Using Intents and BroadcastReceivers
Intents can be used to pass data between different components (activities, services, broadcast receivers) of the same or different applications, but they are limited to lightweight data and are generally used for simple communications.
a. Passing Data with Intents
You can pass data between activities using Intent.
java

Copy code
```
// In the sending activity
Intent intent = new Intent(this, SecondActivity.class);
intent.putExtra("key", "value");
startActivity(intent);

// In the receiving activity
String value = getIntent().getStringExtra("key");
```
2. Using Binder
Binder is the primary IPC (Inter-Process Communication) mechanism in Android. It's used when you need to pass complex data or perform complex operations between processes.
a. Creating a Bound Service
Define a bound service in your app:
java
Copy code
```
public class MyService extends Service {

    private final IBinder binder = new LocalBinder();

    public class LocalBinder extends Binder {
        MyService getService() {
            return MyService.this;
        }
    }

    @Override
    public IBinder onBind(Intent intent) {
        return binder;
    }

    public String getData() {
        return "some data";
    }
}
```
Bind to the service from an activity:
java
Copy code
```
public class MainActivity extends AppCompatActivity {
    MyService myService;
    boolean isBound = false;

    private ServiceConnection connection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            MyService.LocalBinder binder = (MyService.LocalBinder) service;
            myService = binder.getService();
            isBound = true;
        }

        @Override
        public void onServiceDisconnected(ComponentName name) {
            isBound = false;
        }
    };

    @Override
    protected void onStart() {
        super.onStart();
        Intent intent = new Intent(this, MyService.class);
        bindService(intent, connection, Context.BIND_AUTO_CREATE);
    }

    @Override
    protected void onStop() {
        super.onStop();
        if (isBound) {
            unbindService(connection);
            isBound = false;
        }
    }
}
```
3. Using Messenger
Messenger is used for simple IPC where you need to pass messages between processes.
a. Service Side
java
Copy code
```
public class MyService extends Service {
    private final Messenger messenger = new Messenger(new IncomingHandler());

    private static class IncomingHandler extends Handler {
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
                case 1:
                    // Handle message
                    break;
                default:
                    super.handleMessage(msg);
            }
        }
    }

    @Override
    public IBinder onBind(Intent intent) {
        return messenger.getBinder();
    }
}
```
b. Client Side
java
Copy code
```
public class MainActivity extends AppCompatActivity {
    Messenger serviceMessenger;
    boolean isBound = false;

    private ServiceConnection connection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            serviceMessenger = new Messenger(service);
            isBound = true;
        }

        @Override
        public void onServiceDisconnected(ComponentName name) {
            serviceMessenger = null;
            isBound = false;
        }
    };

    @Override
    protected void onStart() {
        super.onStart();
        Intent intent = new Intent(this, MyService.class);
        bindService(intent, connection, Context.BIND_AUTO_CREATE);
    }

    @Override
    protected void onStop() {
        super.onStop();
        if (isBound) {
            unbindService(connection);
            isBound = false;
        }
    }

    private void sendMessage() {
        if (isBound) {
            Message msg = Message.obtain(null, 1, 0, 0);
            try {
                serviceMessenger.send(msg);
            } catch (RemoteException e) {
                e.printStackTrace();
            }
        }
    }
}
```
4. Using ContentProvider

ContentProvider is used to manage access to a structured set of data. It can be used to share data between different applications.

a. Define ContentProvider in Manifest

xml

Copy code
```
<provider
    android:name=".MyContentProvider"
    android:authorities="com.example.myapp.provider"
    android:exported="true" />
```
b. Implement ContentProvider

java

Copy code
```
public class MyContentProvider extends ContentProvider {
    private static final String AUTHORITY = "com.example.myapp.provider";
    private static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY +
"/items");

    @Override
    public boolean onCreate() {
        return true;
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection, String[]
selectionArgs, String sortOrder) {
        // Implement query logic
        return null;
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        // Implement insert logic
        return null;
    }

    // Other required methods (delete, update, getType)
}
```
c. Access ContentProvider

java

Copy code
```
Uri uri = Uri.parse("content://com.example.myapp.provider/items");
Cursor cursor = getContentResolver().query(uri, null, null, null, null);

if (cursor != null) {
    while (cursor.moveToNext()) {
        // Process data
    }
    cursor.close();
}
```
5. Using AIDL (Android Interface Definition Language)
AIDL is used for more complex IPC where you need to pass objects between processes.
a. Create AIDL File
Create an AIDL file in src/main/aidl.
aidl
Copy code
```
// IMyAidlInterface.aidl
package com.example.myapp;

interface IMyAidlInterface {
    String getData();
}
```
b. Implement AIDL Interface in Service
java
Copy code
```
public class MyService extends Service {

    private final IMyAidlInterface.Stub binder = new IMyAidlInterface.Stub() {
        @Override
        public String getData() {
            return "some data";
        }
    };

    @Override
    public IBinder onBind(Intent intent) {
        return binder;
    }
}
```
c. Bind to AIDL Service
java
Copy code
```
public class MainActivity extends AppCompatActivity {
    IMyAidlInterface myAidlService;

    private ServiceConnection connection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            myAidlService = IMyAidlInterface.Stub.asInterface(service);
        }

        @Override
        public void onServiceDisconnected(ComponentName name) {
            myAidlService = null;
        }
    };

    @Override
    protected void onStart() {
        super.onStart();
        Intent intent = new Intent(this, MyService.class);
        bindService(intent, connection, Context.BIND_AUTO_CREATE);
    }

    @Override
    protected void onStop() {
        super.onStop();
        unbindService(connection);
    }

    private void fetchData() {
        try {
            String data = myAidlService.getData();
            // Use data
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
}
```

5.4 Simplifying database operations

Simplifying database operations in the context of an SQLite database can be achieved through various techniques and practices that streamline the process of interacting with the database. Here are some strategies to simplify SQLite database operations:

Use SQLite's Built-in Functions: SQLite comes with a range of built-in functions that can simplify common database tasks, such as string manipulation, date and time calculations, and mathematical operations.
Leverage Parameterized Queries: Instead of constructing SQL queries with string concatenation, use parameterized queries to insert values into your SQL statements. This approach not only simplifies the code but also enhances security by preventing SQL injection attacks.
Utilize SQLite's Command-Line Shell: The SQLite command-line interface provides dot commands and other utilities that can simplify database management tasks1.
Employ Python's sqlite3 Module: For developers using Python, the sqlite3 module provides a convenient and straightforward API for working with SQLite databases. It simplifies database operations by abstracting the complexities of direct SQL interaction2.
Implement ORM Libraries: Object-Relational Mapping (ORM) libraries can abstract the database layer, allowing you to work with database records as if they were objects in your programming language. This can greatly simplify CRUD operations.

Optimize Your Database Schema: A well-designed database schema can simplify operations by reducing the complexity of queries and improving performance.
Use Transactions: Group multiple related operations into a single transaction to ensure atomicity and consistency. This can simplify error handling and rollback procedures.
Regularly Refactor and Optimize Queries: Over time, review and refactor your SQL queries to ensure they remain efficient and easy to understand.
Create Views: Views can encapsulate complex queries, presenting a simplified interface for data retrieval.
Automate Routine Tasks: Use scripts or tools to automate routine database tasks like backups, data migration, and schema changes to reduce manual effort.

Self-Check Sheet 5: Use Content Providers

1. How do you create a content provider in Android?
2. What is the use of a content provider?
3. How can data be passed from one process to another in Android?
4. What are some methods to simplify database operations?
5. Can content providers be used to pass data between processes?

Answer Key 5: Use Content Providers

1. **How do you create a content provider in Android?**
   Answer: To create a content provider, you extend the ContentProvider class, implement its abstract methods, define a URI scheme in a contract class, and register it in the AndroidManifest.xml with the <provider> tag.

2. **What is the use of a content provider?**
   Answer: A content provider is used to manage access to a structured set of data, providing a consistent interface for other applications to perform data operations like query, insert, update, and delete, regardless of the data storage mechanism.

3. **How can data be passed from one process to another in Android?**
   Answer: Data can be passed between processes using content providers, intents, binders, or IPC mechanisms like AIDL. Content providers are a common method for sharing structured data between different applications.

4. **What are some methods to simplify database operations?**
   Answer: Database operations can be simplified by using stored procedures, employing ORMs, optimizing queries, normalizing the database schema, and automating routine tasks to improve efficiency and maintainability.

5. **Can content providers be used to pass data between processes?**
   Answer: Yes, content providers can be used to pass data between processes. They serve as an interface that connects data in one process with code running in another, allowing secure and standardized data access

Task Sheet 5.1: Use Content Providers

Create a Content Provider

1. Task: Implement a custom content provider.
2. Goal: Understand the structure and creation of a content provider.
3. Expected Outcome: A functioning content provider that can handle CRUD operations.

Use the Content Provider from Another Application

1. Task: Access the content provider from a separate app.
2. Goal: Practice interprocess communication using content providers.
3. Expected Outcome: Successful data retrieval and manipulation from another process.

Pass Data Between Processes

1. Task: Implement a mechanism to pass data between processes.
2. Goal: Explore different IPC mechanisms in Android.
3. Expected Outcome: Data passed securely and efficiently between two processes.

Simplify Database Operations

1. Task: Refactor an existing database interaction to use simpler operations.
2. Goal: Apply best practices to simplify complex database operations.
3. Expected Outcome: More efficient and readable database code.

Review Of Competency

Below is yourself assessment rating for module **"Managing Database"**

SL no	Assessment of performance Criteria	Yes	No
1.	Basic concept of shared preferences and file system in android is explained.		
2.	Saving and getting data in shared preference is performed.		
3.	Database is interpreted		
4.	Data type is defined		
5.	SQLite and Database design is described.		
6.	Managing SQLite database is explained.		
7.	Table in SQLite is created.		
8.	CRUD operation in database is made.		
9.	Data operation from database using Android Applications made.		
10.	List is created.		
11.	List is used.		
12.	Collection of items using list is used.		
13.	Working with list in Android is performed.		
14.	Custom list is created using adapter.		
15.	Method to create content provider is explained.		
16.	Content provider is used.		
17.	Data from one process to another is passed.		
18.	Database operation is simplified.		

I now feel ready to undertake my formal competency assessment.

Signed:

Date:

Development of CBLM

The Competency based Learning Material (CBLM) of "Managing Database" (Occupation: Android Mobile Application Development, Level-4) for National Skills Certificate is developed by NSDA with the assistance of SIMEC System Ltd., ECF Consultancy & SIMEC Institute of Technology JV (Joint Venture Firm) in the month of July, 2024 under the contract number of package SD-9B dated 15th January 2024.

SL No.	Name & Address	Designation	Contact Number
1	Md. Abdul Al Hossain	Writer	01778-926438
2	Md Belayet Hossain	Editor	01714-117032
3	Engr Md. Zuwel Parves	Co-Ordinator	01737-278906
4	Md. Abdur Razzaque	Reviewer	01713-304824

Reference

1. Shared Preferences in Android with Example - GeeksforGeeks
2. SQLite Create Table with Examples (sqlitetutorial.net)
3. python - Add list to sqlite database - Stack Overflow
4. https://chatgpt.com/
5. https://gemini.google.com/