PART-B ESSAY QUESTIONS WITH SOLUTIONS

5.1 INTRODUCTION TO SQLITE DATABASE

Q8. Give a brief introduction to SQLite database and SQLiteOpenHelper class.

Answer :

Model Paper-II, Q10(a)

SQLite Database

SQLite Database is a freely available database which is used to store data locally for offline applications and centrally for business logic based applications. It is a light weight, single tier and standards based relational database management system. It comes in the form of in-built library of Android operating system.

SQLiteOpenHelper Class

The SQLiteOpenHelper class is one of the abstract classes that allows a user to access a database in read mode or write mode. This class helps in performing the manipulation of data available in the database. The operations that can be performed on database includes creating, opening and upgrading. It user different methods such as getWritableDatabase( ) to write, getReadable-Database( ) to read and close( ) to close the database by returning an object of SQLiteDatabase. It also provides two other methods i.e., insert( ) and query( ). The insert( ) method is used for inserting the rows in the database table whereas query( ) method is used for executing various SQLite queries on the database table. The query( ) method can accept different parameters. In response, a cursor is returned that indicates row satisfying the given criteria. The cursor class allows different methods to perform traversing and also to change the positions of row and column as desired in the result set.

Q9. Describe the process of building SQLite project with an example.

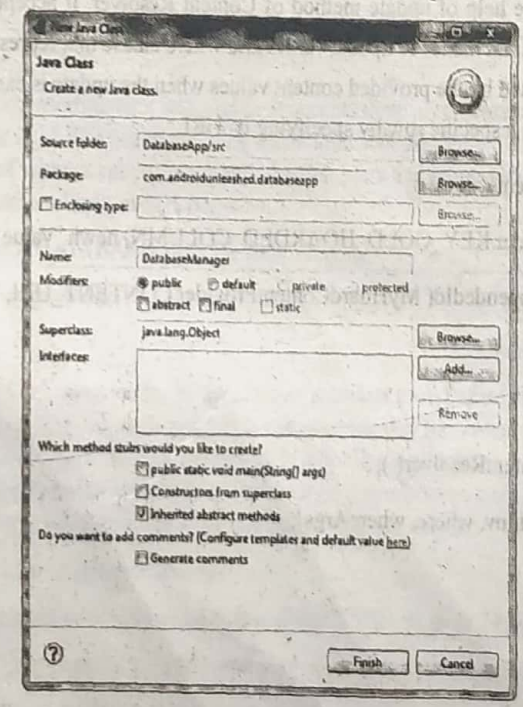
Answer :

Model Paper-I, Q11

Building SQLite Project

SQLite project can be built by following the steps given below,

1. Start Eclipse IDE application. Create a new android project with the name DatabaseApp1.
2. In the next step, create a database with the name student.
3. Create a table with the details that contain three columns (sid, sname, sdept) of the student.
4. Rows can be added to the table which can be displayed using textview control.
5. Right-click on the link com.androidunleashed.databaseapp package present in the window package explorer to add a java class. From the displayed options, select New and click on class option.
6. A dialog box appears showing the New Java Class in which values are required to be entered.



7. The value of source folder and package are entered as DatabaseApp/src and com.androidunleashed.databaseapp respectively, as shown in the figure.

8. Click on Finish button.

A new Java file is created with the following code,

Example

DatabaseManager.java

```
package com.androidunleashed.databaseapp;
import android.database.sqlite.SQLiteDatabase;
import android.content.Context;
import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;
public class DatabaseManager
{
    public static final String NAME = "Student";
    public static final String TABLE = "details";
    public static final int VERSION = 1;
    private.static final String CREATE_TABLE = "CREATE TABLE" + TABLE +
    "(sid INTEGER PRIMARY KEY, sname TEXT, dept TEXT);";
    private SQLHelper h;
    private SQLiteDatabase d;
    private Context con;
    public DatabaseManager(Context ct)
    {
        this.con = ct;
        h = new SQLHelper (ct);
        this.d = h.getWritableDatabase( );
    }
    public DatabaseManager OpenReadable( ) throws android.database.SQLException
    {
        h = new SQLHelper(con);
        d = h.getReadableDatabase( );
        return this;
    }
    public void close( )
    {
        h.close ( );
    }
```

```
public void addRow(Integer i, String n, string dp)
{
    ContentValue s = new ContentValues( );
    s.put("sid"; i);
    s.put("sname", n);
    s.put("dept", dp);
    try{d.insertOrThrow(Table, null, s);
}
    catch(Exception ex)
    {
        Log.ex("Error found while inserting rows", e.toString( ));
        exprintStackTrace( );
    }
}
public String retrieveRows( )
{
    String[ ] col = new String[ ] {"sid", "sname", "dept"};
    Cursor cur = d.query(TABLE, col, null, null, null, null, null);
    String rows = " ";
    cur.moveToFirst( );
    while(cur.isAfterLast( ) = = false)
    {
        rows = rows + cur.getInt(0) + "," +cur.getString(1)+ "," + cur.getString(2) + "\n";
    cur.moveToNext( );
    }
    if(cur ! = null && !cur.isClose( ))
    {
        cur.close( );
    }
    return rows;
}
public class SQLHelper extends SQLiteOpenHelper
{
    public SQLHelper(Context (ct)
    {
        super(ct, NAME, null, VERSION);
    }
}
```

```
@Override
public void onCreate(SQLiteDatabase d)
{
    d.execSQL(CREATE_TABLE);
}
@Override
public void onUpgrade(SQLiteDatabase d, int oldV, int newV)
{
    Log.w("details table", "Upgrading database i.e., dropping table and recreating it");
    d.execSQL("DROP TABLE IF EXISTS" + TABLE);
    onCreate(d);
}
}
```

5.2 CREATING AND OPENING A DATABASE, CREATING TABLES, INSERTING, RETRIEVING AND DELETING DATA

Q10. Write the process of creating and opening a database.

Answer :

Creating a Database

Model Paper-III, Q11

SQLiteOpenHelper is an abstract class that implements a best practice pattern to create, open and upgrade the databases. The SQLiteOpenHelper is implemented to encapsulate and hide the logic that decides whether the data base must be created or upgraded before it is opened. It is a good thing to hold up the process of creating and opening databases until they are actually required. This is supported by SQLiteOpen Helper by caching database instances once they are opened. Then users can request to open before performing any query. The below code depicts the creation of new database. For this it makes use of onCreate( ) and onUpgrade( ) methods to handle creation.

```
public static class HoardDBOpenHelper extends SQLiteOpenHelper
{
    public static final String DB = "DBI.db";
    public static final String TB = "GH";
    public static final int DV = 1;
    private static final String DC = "create table" + TB + "("HoardContract. KEY_ID + "integer primary autoincrement,"
    + HoardContract_KEY_Col + "text not null," + HoardContract.K_CN + "float," + HoardContract.K_AC + "integer);";
    public HoardDBOpenHelper(Context c, String names SQLifeDatabase.CursorFactory f, int v)
    {
        super (c, name, f, v)
    }
    public void onCreate(SQLiteDatabase db)
    {
        db.execSQL(DC);
    }
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
    {
        Log.w("TaskDBAdapter", "Upgrading from version" + oldversion + "to" + newversion +" which destroys the data");
        db.execSQL("DROP TABLE IF EXISTS" + TB);
        OnCreate (db);
    }
}
```

Opening a Database

Database can be accessed using getWritableDatabase or getReadableDatabase of SQLiteOpenHelper class. However, onCreate handler is triggered if the searched database does not exists. Sometimes, a getWritableDatabase might not work for various reasons like permission, disk space or other reasons. In such cases, getReadableDatabase method is used.

```
HoardDBOpenHelper hb = new HoardDBOpenHelper(c, HoardDBOpenHelper.name, null, HoardDBOpenHelper.v);
SQLiteDatabase db;
try
{
    db = hoardDBOpenHelper.getWritabaleDatabase( );
}
catch (SQLiteException e)
{
    db = hoardDBOpenHelper.getReadableDatabase( );
}
```

Q11. Discuss in brief about the following,

(i) Creating tables

(ii) Inserting data

(iii) Deleting data.

Answer : Model Paper-II, Q10(b)

(i) Creating Tables

For answer refer Unit-V, Q10, Topic: Creating a Database.

(ii) Inserting Data

A new row in database can be created using ContentValues object where its associated puts method can be used to add column name and its value.

The method insert( ) is used for inserting data into a table. It accepts three arguments namely table name, null column hack and name-value pairs of the specific field. The argument table name indicates the name of the table, null column hack prevents an error when inserting a new record without specifying the field-value pairs and name-value pairs are the actual data that need to be inserted in table. This is shown as follows in SQLite,

```
db.insert(HoardDBOpenHelper.DATABASE_NAME, null, values);
```

(iii) Deleting Data

Data can be deleted from the table by using delete( ) method. This query must include table name and where clause depicting the rows to be deleted.

```
String where = HoardContract.KEY_ID + "=?";
String whereArgs [ ] = {hoardId};
SQLiteDatabase db = hoardDBOpenHelper.getWritableDatabase( );
db.delete(HoardDBOpenHelper.DATABASE_TABLE, where, whereArgs);
```

5.3 REGISTERING CONTENT PROVIDERS

Q12. Define content provider. Discuss about registering content providers

Answer :

Content Provider

A content provider can be defined as an interface which is used to publish and utilize the data over URI addressing model.

Registering Content Providers

The content providers need to be registered in the application manifest ever fore being discovered by content resolver. A provider tag is used for this purpos This tag accompanies name attribute that represents providers classname and authorities tag. The latter tag is used to define the base URI of providers authority. The authority of a content provider is used by content resolver in the form of address to find the required data for interaction. It needs to be unique so, it would be better to use base URIPath on package name. The content providers authority can be defined as follows,

com.<CompanyName>.provider.<ApplicationName>

The provider tag that is completed must follow the below defined XML snippet format,

<provider android : name = ".MyContentProvider" android : authorities = "com.paad.skeletondbprovider"/>

Publishing the URI Address of Content Provider

The content provider exposes the authority by using public static CONTENT_URI property so that it becomes discoverable easily. With this, a data path gets added to primary content as shown below,

public static final Uri CONTENT_URI = Uri.parse("content://com.paad.skeletondbprovider/elements");

These URIs are used to access the content provider by using content resolver. This type of query depicts request for rows and the appended trailing /<rownumber> depicts request for single record,

content : //com.paad.skeletondbprovider/elements/5

The class UriMatcher parses the URIs and determines their forms. It even supports access to provider in the above defined forms.

```
private static final int ALLROWS = 1;
private static final int SINGLE_ROW = 2;
private static final UriMatcher urim;
static
{
    urim = new UriMatcher (UriMatcher.NO_MATCH);
    urim addURI ("com.paad.skeletondbprovider", "elements", ALLROWS);
    urim.addURI("com.paad.skeletondbprovider", "elements/#", SINGLE_ROW);
}
```

The content providers database can be created as shown below,

```
private MySQLiteOpenHelper mop;
public boolean onCreate( )
{
    mop = new MySQLiteOpenHelper(getContext( ), MySQLiteOpenHelper.DB_NAME, null,
                                                    MySQLiteOpenHelper.DB_VERSION);
    return true;
}
```

When the content provider is created then the data source planned through it can be initialized.

For remaining answer refer Unit-V, Q13.

5.4 USING CONTENT PROVIDERS (INSERT, DELETE, RETRIEVE AND UPDATE)

Q13. Explain how content is inserted, deleted and updated. Model Paper-III, Q10

Answer :

Various operations to be performed on content providers make use of methods provided by ContentResolver. It acts as a class for retrieving and performing transactions on content providers. It provides abstract representation of content providers.

Inserting Content

The records can be inserted into content provider by using two methods of content resolver. They are insert and bulkInsert methods. The insert method accepts contentValues object and URI of content provider as parameters. It returns URI of the added record. The bulkInsert method accepts array and URI of content provider as parameters. It returns the number of added rows.

```
ContentValues cv = new ContentValues( );
cv.put(MyHoardContentProvider.KEY_GOLD_HOARD_NAME_COLUMN, h_Name);
cv.put(MyHoardContentProvider.KEY_GOLD_HOARDEDCOLUMN, h_Value);
cv.put(MyHoardContentProvider.KEY_GOLD_ACCESIBLE_COLUMN, h_Accessible);
ContentResolver cr1 = getContentResolver( );
Uri uri = cr1.insert(MyHoardContentProvider.CONTENT_URI, cv);
```

Deleting Content

A single record can be deleted by using a delete( ) on content resolver. This method accepts the URI of the row to be deleted. A where clause can also be specified to remove multiple rows. The code to delete number of rows matching the specified condition is as follows,

```
String where = MyHoardContentProvider.KEY_GOLD_HOARDED_COLUMN + "=" + 0;
String whereArgs[ ] = null;
ContentResolver cr1 = getContentResolver( );
int del_RC = cr1.delete (MyHoardContentProvider.CONTENT_URI, where, whereArgs);
```

Updating Content

Rows can be updated with the help of update method of Content Resolver. It accepts URI of target content provider, contentValues object which maps column names to update values and where clause that represents rows to be updated. The rows matching with where clause are updated by the provided content values when the update is executed. As a result, the updates are returned. It is even possible to update a specific row by specifying its URI.

```
ContentValues uv = new ContentValues( );
uv.put(MyHoardContentProvider.KEY_GOLD_HOARDED_COLUMN, newh_Value);
Uri ruri = ContentUris.withAppendedId( MyHoardContentProvider.CONTENT_URI, h_Id);
String where = null;
String whereArgs[ ] = null;
ContentResolver cr1 = getContentResolver( );
int updatedrc = cr1.update(ruri, uv, where, whereArgs);
```

Q14. Explain about content retrieval.

Model Paper-II, Q11

Answer :

Retrieving Content

The files stored in a content provider can be retrieved or accessed by using openOutputStream or openInputStream methods of content resolver. Pass the URI to content provider row containing the required file. The content provider will then interpret and returns input or output stream to the requested file,

```
public void addnewHoardWithImage(String h_Name, float h_Value, boolean h_Accessible, Bitmap bm)
{
ContentValues cv = new ContentValues( );
cv.put(MyHoardContentProvider.KEY_GOLD_HOARD_NAME_COLUMN, h_Name);
cv.put(MyHoardContentProvider.KEY_GOLD_HOARDED_COLUMN, h_Value);
cv.put(MyHoardContentProvider.KEY_GOLD_HOARD_ACCESSIBLE_COLUMN, h_Accessible);
ContentResolver cr = getContentResolver( );
Uri u = cr.insert (MyHoardContentProvider.CONTENT_URI, cv);
try
{
    OutputStream os = cr openOutputStream(u);
    bitmap.compress(Bitmap.CompressFormat.JPEG, 80, os);
}
catch(FileNotFoundException ex)
{
    Log.d(TAG,"There is no file for the record.");
}
}
public Bitmap getHoardImage(long r_Id)
{
    Uri u = ContentUris.withAppendId(MyHoardContentProvider.CONTENT_URI, r_Id);
    try
    {
        InputStream is = getContentResolver( ).openInputStream(u);
        Bitmap bm = BitmapFactory.decodeStream (is);
        return bm;
    }
    catch (FileNotFound Exception ex)
    {
        Log.d(TAG, "There no file for the record");
        return null;
    }
}
```