Android SQL Implementation

Upon completion of this module, a student will be able to

- write a contract class to store SQL table structure and field constants
- write a database helper class to manage database creation and upgrades
- build a database access class to facilitate interactions with the database
- execute SQL statements to get data from the database
- work with Cursors to interpret data returned from the database
- use methods to interact with the database without SQL

Assignment

- Task
  - Add persistence to the xkcd app by tracking the user's history and allowing them to save favorites
- Repo
  - https://github.com/LambdaSchool/Android_Xkcd_Persistance
- Submission
  - Fork on github and submit pull request
- Notes
  - This is a two day assignment as tomorrow is a review day.

A Student Can

write a contract class to store SQL table structure and field constants

Database Contract

- Inner Class for each Table
- Data member for each column

```
public class SchoolDbContract {
    public static class GradesEntry implements BaseColumns {
        public static final String TABLE_NAME            = "grades";
        public static final String COLUMN_NAME_STUDENT_ID = "student_id";
        public static final String COLUMN_NAME_SUBJECT_ID = "subject_id";
        public static final String COLUMN_NAME_DATE       = "date";
        public static final String COLUMN_NAME_MARK       = "mark";

        ...
    }

    ...
}
```

CREATE/DELETE Queries

```
public class SchoolDbContract {
    public static class GradesEntry implements BaseColumns {
        ...

        public static final String SQL_CREATE_TABLE =
                "CREATE TABLE " + TABLE_NAME + " (" +
                _ID + " INTEGER PRIMARY KEY," +
                COLUMN_NAME_STUDENT_ID + " INTEGER," +
                COLUMN_NAME_SUBJECT_ID + " INTEGER," +
                COLUMN_NAME_DATE + " INTEGER," +
                COLUMN_NAME_MARK + " INTEGER);";

        public static final String SQL_DELETE_TABLE =
                "DROP TABLE IF EXISTS " + TABLE_NAME + ";";
    }

    ...
}
```

- Create
  - CREATE TABLE
  - Store Schema for Reference
- Delete
  - DROP TABLE

A Student Can

write a database helper class to manage database creation and upgrades

Database Helper

- Manages DB Access
- Creates/Deletes/Upgrades

```
public class SchoolDbHelper extends SQLiteOpenHelper {

    public static final int DATABASE_VERSION = 1;
    public static final String DATABASE_NAME = "SchoolDatabase.db";

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

    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
        sqLiteDatabase.execSQL(SchoolDbContract.GradesEntry.SQL_CREATE_TABLE);
        ...
    }

    @Override
    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int oldVersion, int newVersion) {
        sqLiteDatabase.execSQL(LightDbContract.GradesEntry.SQL_DELETE_TABLE);
        ...
    }

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

Executing SQL Statements

```
public class SchoolDbHelper extends SQLiteOpenHelper {
    ...
    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
        sqLiteDatabase.execSQL(SchoolDbContract.SubjectsEntry.SQL_CREATE_TABLE);
        sqLiteDatabase.execSQL(LightDbContract.GroupsEntry.SQL_CREATE_TABLE);
        sqLiteDatabase.execSQL(LightDbContract.TeachersEntry.SQL_CREATE_TABLE);
        sqLiteDatabase.execSQL(LightDbContract.StudentsEntry.SQL_CREATE_TABLE);
        ...
    }

    ...
}
```

- execSQL(String sqlStatement)

A Student Can

build a database access class to facilitate interactions with the database

DAO

- Completes tasks on database
- Tasks Independent of DB
  - Execution is DB specific

```
public class SchoolSqlDbDao {
    private SQLiteDatabase db;

    public SchoolSqlDbDao(Context context) {
        SchoolDbHelper dbHelper = new SchoolDbHelper(context);
        db = dbHelper.getWritableDatabase();
    }

    ...
}
```

A Student Can

execute SQL statements to get data from the database

Raw SQL

```
public class SchoolSqlDbDao {
    ...
    public ArrayList<StudentGrades> getStudent(int id) {
        Cursor cursor = db.rawQuery(
                String.format("SELECT %s, %s, %s, %s\n" +
                              "FROM %s\n" +
                              "WHERE %s = ?" +
                              "JOIN %s ON %s.%s = %s.%s\n" +
                              "JOIN %s ON %s.%s = %s.%s;",

                              SchoolDbContract.StudentsEntry.COLUMN_NAME_FIRST_NAME,
                              SchoolDbContract.StudentsEntry.COLUMN_NAME_LAST_NAME,
                              SchoolDbContract.GradesEntry.COLUMN_NAME_MARK,
                              SchoolDbContract.SubjectsEntry.COLUMN_NAME_TITLE,

                              SchoolDbContract.StudentsEntry.TABLE_NAME,

                              SchoolDbContract.StudentsEntry._ID,

                              SchoolDbContract.GradesEntry.TABLE_NAME,
                              SchoolDbContract.StudentsEntry.TABLE_NAME,
                              SchoolDbContract.StudentsEntry._ID,
                              SchoolDbContract.GradesEntry.TABLE_NAME,
                              SchoolDbContract.GradesEntry.COLUMN_NAME_STUDENT_ID,

                              SchoolDbContract.SubjectsEntry.TABLE_NAME,
                              SchoolDbContract.GradesEntry.TABLE_NAME,
                              SchoolDbContract.GradesEntry.COLUMN_NAME_SUBJECT_ID,
                              SchoolDbContract.SubjectsEntry.TABLE_NAME,
                              SchoolDbContract.SubjectsEntry._ID
                             ),
                new String[]{Integer.toString(id)});
        ...
    }
}
```

- Executes SQL Statement
- Returns Cursor Object

A Student Can

work with Cursors to interpret data returned from the database

Cursor

- Points to results in database
- Step through results
- Store values

```
public class SchoolSqlDbDao {
    ...
    public ArrayList<StudentGrades> getStudent(int id) {
        Cursor cursor = db.rawQuery(...);

        ArrayList<StudentGrades> rows = new ArrayList<>();
        int index;
        while(cursor.moveToNext()){
            index = cursor.getColumnIndexOrThrow(SchoolDbContract.StudentsEntry.COLUMN_NAME_FIRST_NAME;
            String firstName = cursor.getString(index));

            index = cursor.getColumnIndexOrThrow(SchoolDbContract.StudentsEntry.COLUMN_NAME_LAST_NAME;
            String lastName = cursor.getString(index));

            index = cursor.getColumnIndexOrThrow(SchoolDbContract.GradesEntry.COLUMN_NAME_MARK;
            int mark = cursor.getInt(index));

            index = cursor.getColumnIndexOrThrow(SchoolDbContract.SubjectsEntry.COLUMN_NAME_TITLE;
            int title = cursor.getInt(index));

            rows.add(new StudentGrades(firstName, lastName, mark, title));
        }
        cursor.close();
        ...
    }
}
```

Store Query Results

```
public class Grade {
    int    id, mark;
    String subject;

    public Grade(int id, int mark, String subject) {
        this.id      = id;
        this.mark    = mark;
        this.subject = subject;
    }

    public int getId() {
        return id;
    }
    ...
}

public class Student {
    int              id, groupId;
    String           firstName, lastName;
    ArrayList<Grade> grades;

    public Student(int id, String firstName, String lastName, int groupId) {
        this.id        = id;
        this.firstName = firstName;
        this.lastName  = lastName;
        this.groupId   = groupId;
        this.grades    = new ArrayList<>();
    }

    public void addGrade(Grade grade) {
        grades.add(grade);
    }
    ...
}
```

- Model Object
- Store data in a format that your app can use efficiently

A Student Can

use methods to interact with the database without SQL

Content Values

- Key-Value Pairs
- Translated into SQL parameters

Create

```
ContentValues values = new ContentValues();
values.put(SchoolDbContract.StudentEntry.COLUMN_NAME_FIRST_NAME, studentGrade.getFirstName());
values.put(SchoolDbContract.StudentEntry.COLUMN_NAME_LAST_NAME, studentGrade.getLastName());
values.put(SchoolDbContract.StudentEntry.COLUMN_NAME_GROUP_ID, studentGrade.getGroupId());
int newId = db.insert(SchoolDbContract.StudentEntry.TABLE_NAME, null, values);
```

- Parameters
  - Table Name
  - null
  - ContentValues
    - Columns and their desired values

Update

- Parameters
  - Table Name
  - ContentValues
    - Columns and their desired values
  - WHERE Clause
  - WHERE arguments array

```
ContentValues values = new ContentValues();
values.put(SchoolDbContract.StudentEntry.COLUMN_NAME_FIRST_NAME, studentGrade.getFirstName());
values.put(SchoolDbContract.StudentEntry.COLUMN_NAME_LAST_NAME, studentGrade.getLastName());
values.put(SchoolDbContract.StudentEntry.COLUMN_NAME_GROUP_ID, studentGrade.getGroupId());
int affectedRows = db.update(SchoolDbContract.StudentEntry.TABLE_NAME,
                             values,
                             SchoolDbContract.StudentEntry._ID + "=?",
                             new String[] { Integer.toString(studentGrade.getId()) });
```

Delete

```
int affectedRows = db.delete(SchoolDbContract.StudentEntry.TABLE_NAME,
                             String.format("%s = ?", SchoolDbContract.StudentEntry._ID),
                             new String[] { Integer.toString(studentGrade.getId()) });
```

- Parameters
  - Table Name
  - WHERE Clause
  - WHERE arguments array