Hi, I'm Sunny Nguyen, and this is my code review for CS 499.

Today I'll be reviewing my Event Tracker application from CS 360, focusing on DatabaseHelper.java as the central artifact.

I'll walk through three enhancement categories: Software Engineering & Design, Algorithms & Data Structures, and Databases.

Show Android Studio with project open

This is an Android event tracking app with user authentication, SQLite storage, and notification reminders.

Show File structure in project explorer

The app has these key components:

- MainActivity.java - login/registration
- EventsGridActivity.java - main UI for adding/viewing events
- **DatabaseHelper.java** - our focus today - the data persistence layer
- EventReminderReceiver.java - handles notifications

Show Quick app demo (10 seconds) - login, add event, see list

DatabaseHelper is the foundation—it manages both user accounts and event data.

Now I'd like to talk about **category 1, which is software design & engineering**

Let's look at Section 1 of category 1: Existing Functionality

Show: Open DatabaseHelper.java in Android Studio

- DatabaseHelper extends SQLiteOpenHelper and manages two tables: events and users.
- **Let's look at onCreate method on line 37**: we can see the database tables being created, including all the columns for events.
- **Let's look at insertEvent method on line 61**: This method adds a new event into the database.
- **Let's look at getAllEvents method on line 72**: This method loads all events and turns each row into an Event object, sorted by date and time.
- **Let's look at validateUser method on line 135**: This method checks if the username and password match, and is used for login.

Now let's look at the architecture.

- The DatabaseHelper class uses the normal Android SQLiteOpenHelper pattern, with onCreate to build the tables and onUpgrade to handle future changes.
- The EventsGridActivity uses this helper to save and load events, and MainActivity uses it to check the user's login

Now let's look at Section 2 of category 1: Code Analysis

Let's open the Code Review Checklist:

For the Structure Issues:

- First, the question is "is storage use efficient?" Currently, no. because there are no indexes on common fields like date
- Second, Are there any modules excessively complex? Not yet, but if I keep adding more features later, I will need to break this class into smaller pieces to keep it clean and organized.

For the Defensive Programming:

- The question is: Are files checked for existence? The answer is no — the code does not check if the database is ready before it runs the query, as we can see in the getAllEvents method on line 72.
- The question is Are imported data tested for validity? The answer is there is no input validation, so the app could insert empty or invalid data into the database, as we can see in the insertEvent method on line 61.

For the Documentation:

- The question is **Is the code clearly documented?** Not really. There are only a few comments. The emoji check marks are fun, but real professional code needs clearer comments that explain what each part does.

For the Variables:

- The question is **Are all variables properly defined?** Yes, but I am repeating magic strings like 'events' and 'users'. Those should be stored as constants instead.

For the Security:

- As we can see in the the validateUser method on line 135: There is a big security issue. The passwords are stored as plain text. This is not safe. Passwords should be hashed before storing them.

In short, the code works, but it has some problems. It is not very efficient, it does not check input carefully, the comments are weak, and there is a major security issue with how passwords are stored.

Let's look at Section 3 of category 1: Enhancements

For Software Engineering, I have once enhacement: recurring event support, like daily, weekly, and monthly repeated events.

The architectural changes needed are

- Let's look at the TABLE_EVENTS on line 19: First, I will add a new column called recurrence_type, which can be NONE, DAILY, WEEKLY, or MONTHLY.
- Then I will add a new method called generateNextRecurringEvent that will copy an event and schedule the next one automatically based on the type of recurrence.

Show screen code: Category 1 – Section 3: Enhancements

Here is the pseudocode I made in Module One. These two functions show how the recurring event feature will work. The scheduleEvent function will save events and set the alarms. Then onNotificationTriggered will run when the alarm goes off, and if the event is set to repeat, it will automatically create the next event.

So for Engineering improvements.

- First, this improvement shows modular design, because the repeating event logic is in its own method.
- Second, it shows that the app can grow in the future, because more repeat types like yearly or custom can be added later.
- Last, it matches Course Outcome 3, because it uses good computer science practices and handles design trade-offs properly.

Now I'd like to talk about **category 2, which is algorithms & data structures**

Let's look at Section 1 of Category 2, which is Existing Functionality

Show DatabaseHelper.java

Now let's look at this file from a data structure and algorithm perspective.

- Let's look at getAllEvents method on line 72: the code uses a Cursor to loop through every row and turns each row into an Event object. The problem is that the SQL query has no index, so it has to scan the whole table every time.
- Let's look at getEventsForDate method on line 94: The query also scans the whole table just to find events for one date.

Let's look at the Algorithm analysis. Right now, getting events is O(n) because the database has to check every row. As more events get added, the app will get slower. Sorting with ORDER BY also becomes slow without indexes, because the database has to sort all the results every time.

Show screen code Category 2 - Section 1: Existing Functionality

There is no WHERE optimization and no indexes, so the database is just doing a plain scan through every row in order.

Let's look at Section 2 of Category 2, which is Code Analysis

Let's look at the Code Review Checklist:

For Loops and Branches:

- **First, the question is Are loop termination conditions obvious?** Yes, in the cursor loop, the loop ending is easy to understand because moveToNext method is clear.
- **Second, the question is Can any statements be placed outside loops?** The answer is Some things inside the loop could be moved out, like the getColumnIndexOrThrow method. Those should be done once before the loop, not every time inside the loop.

For Arithmetic Operations:

- This part does not apply here, because there are no floating-point math operations in this code.

For Storage Efficiency: Storage is not very efficient right now because there are no indexes, so every query has to scan the whole table. The Cursor is good for memory because it loads rows one at a time, but without indexes the search speed is still slow.

For Defensive Programming:

- In the getAllEvents method the getColumnIndexOrThrow method will crash if the column doesn't exist, but we don't catch that error or handle it, so the app could crash instead of recovering safely**.**

For Performance Issues:

- The first problem is that there are no database indexes, so every search is O(n). The second problem is that the code calls getColumnIndexOrThrow multiple times inside the loop, which is wasted work and slows things down.

In short, the current code works, but it is not very efficient. As more data is added, the app will get slower. This is a perfect example of where algorithm improvements can make a big difference.

Let's look at Section 3 of category 2: Enhancements

For Algorithms & Data Structures, I have two enhancements: add database indexes and optimize query logic.

Now Let's look at the Enhancements 1 code: Add Indexes

Show screen code Category 2 - Section 3: Enhancement 1

- This turns searches from O(n) to O(log n) because the database can use B-tree indexing. For example, if there are 1,000 events, instead of checking about 1,000 rows, it would only take around 10 steps to find the matching rows.

Let's look at the Enhancements 2 code: Optimize Query

Show screen code Category 2 - Section 3: Enhancement 2

- I will add a new method called getUpcomingEvents(int limit). This method uses indexes to filter quickly and also limits how many results we return, so it runs much faster.

Let's look at the Enhancements 3 code: Cache Column Indices

Show screen code Category 2 - Section 3: Enhancement 3

- I cache the column indexes one time before the loop, so the loop runs faster and doesn't do extra work.

So, the improvements show that I understand time complexity, I can find slow parts in the code and fix them with better data structures, and it matches Course Outcome 3 by using smart algorithm choices to make the program faster.

Now I'd like to talk about **category 3, which is Databases**

Let's look at Section 1 of Category 3, which is Existing Functionality

Let's look at onCreate method on line 37

For the current database design:

- This app uses a single-tier, local-only SQLite database.
- there are two tables: events with 5 columns, and users with 3 columns. The design is simple. There are no foreign keys or relationships between the tables, and it uses standard CRUD actions like Create, Read, Update, and Delete.

Show Screen Code: Current Database architecture

For the Data flow, data only moves one way: from the app into the SQLite file and back to the app. There is no backup, no syncing, and no support for using the app on more than one device. All changes only stay on the device. So if the app is deleted or the phone is lost, all the data is lost too.

Let's look at Section 2 of Category 3, which is Code Analysis

Let's look at the Code Review Checklist:

For Structure Issues:

- **First, the question is Does the code conform to database standards?** Well, it follows some Android SQLite best practices, but it does not really check if the tables are designed in the best way.
- **Second, can any code be replaced by external components?** The answer is yes — because instead of only using the phone storage, this could use a backend server to handle data better.

For Defensive Programming:

- The first Question is **Are files checked for existence?** SQLiteOpenHelper does that part for us. But we do not check if the database is still okay after the app crashes.
- Second, **Are all files left in correct state upon termination?** Yes, database closes properly.

For Data Integrity Issues:

- First, the only real rule is that the username must be unique. This means we could still end up with duplicate events or leftover data
- Second, there is no link between events and users, so any user could see all events, which is not good for privacy or proper data control.

For Scalability Issues:

- First, the app only stores data on the device, which is a single point of failure.
- Second, there is no backup plan, so data could be lost easily.
- Last, there is no control if more than one process tries to use the database at the same time, which could mess up data.

For Security Issues:

- First, there is no encryption, so the SQLite file is stored as plain text on the device.
- Second, there are no security tokens, and passwords are checked only on the device, which is not very secure.

In short, the database is okay for a small prototype on one device, but it does not have the features needed for real apps, like backups, syncing, strong security, and support for multiple users.

Let's look at Section 3 of category 3: Enhancements

For the Database part, I will add cloud sync so the app uses both local storage and online storage together.

Show Screen Code: New Database Architecture.

I have 2 enhancements: Add Remote API Endpoints and Implement Sync-On-Launch

Let's look at Add Remote API Endpoints

- Show Screen Code: Pseudocode for Remote API Endpoints.
- I'll create a simple REST API with POST/GET endpoints for event sync.

Let's look at Implement Sync-On-Launch

- Show Screen Code: Pseudocode for Implement Sync-On-Launch.
- This GET endpoint downloads all remote events on app launch and inserts any missing ones into the local SQLite cache.

The Database improvements are.

- The upgrades change the app from only saving data on the phone to also saving data in the cloud. Now, the app can copy and sync events between the device and the server. This means the data is safer, backed up, and can be used on more than one device. It also shows that the app uses modern web APIs, just like real apps do today.
- This aligns to Course Outcome 4 because cloud sync is a modern technique used in real apps to add value. It also connects to Course Outcome 3 because it shows I can make smart choices between local speed and cloud reliability, and balance both to make the app better.

Show Screen Code: Thank You.

To summarize, I reviewed DatabaseHelper.java across three computer science areas. In Software Engineering, I showed how adding recurring events makes the app easier to expand. In Algorithms, I improved speed by using indexes, which makes searching faster. In Databases, I upgraded the app by adding cloud sync so it works like a modern app, not just a local one

Overall, all three improvements come from one file, DatabaseHelper.java, so the whole plan fits together nicely. These changes will help move the app from a school project level to something closer to a real professional app.

Thank you for watching. I'm happy to answer any questions about my code review or my improvement plans.