Android Developer Fundamentals

Storing Data

SQLite Database

Contents

SQL Databases

- Store data in tables of rows and columns (spreadsheet…)
- Field = intersection of a row and column
- Fields contain data, references to other fields, or references to other tables
- Rows are identified by unique IDs
- Column names are unique per table

Tables

WORD_LIST_TABLE		
_id	word	definition
1	"alpha"	"first letter"
2	"beta"	"second letter"
3	"alpha"	"particle"

SQL basic operations

- Insert rows
- Delete rows
- Update values in rows
- Retrieve rows that meet given criteria

SQL Query

- SELECT word, definition
  FROM WORD_LIST_TABLE
  WHERE word="alpha"

Generic

- SELECT columns
  FROM table
  WHERE column="value"

SELECT columns FROM table

- **SELECT columns**
    - Select the columns to return
    - Use * to return all columns

- **FROM table**—specify the table from which to get results

WHERE column="value"

- **WHERE**—keyword for conditions that have to be met
- **column="value"**—the condition that has to be met
  - common operators: =, LIKE, <, >

AND, ORDER BY, LIMIT

SELECT _id FROM WORD_LIST_TABLE WHERE word="alpha" **AND** definition LIKE "%art%" **ORDER BY word DESC LIMIT 1**

- **AND, OR**—connect multiple conditions with logic operators
- **ORDER BY**—omit for default order, or ASC for ascending, DESC for descending
- **LIMIT**—get a limited number of results

Sample queries

WORD_LIST_TABLE		
_id	**word**	**definition**
1	"alpha"	"first letter"
2	"beta"	"second letter"
3	"alpha"	"particle"

1	SELECT * FROM WORD_LIST_TABLE	Get the whole table
2	SELECT word, definition FROM WORD_LIST_TABLE WHERE _id > 2	Returns [["alpha", "particle"]]

Sample queries

WORD_LIST_TABLE		
_id	**word**	**definition**
1	"alpha"	"first letter"
2	"beta"	"second letter"
3	"alpha"	"particle"

3	SELECT _id FROM WORD_LIST_TABLE WHERE word="alpha" AND definition LIKE "%art%"	Return id of word alpha with substring "art" in definition `[["3"]]`

Sample queries

WORD_LIST_TABLE		
_id	word	definition
1	"alpha"	"first letter"
2	"beta"	"second letter"
3	"alpha"	"particle"

4	SELECT * FROM WORD_LIST_TABLE ORDER BY word DESC LIMIT 1	Sort in reverse and get first item. Sorting is by the first column (_id) `[["3","alpha","particle"]]`

Last sample query

WORD_LIST_TABLE		
_id	**word**	**definition**
1	"alpha"	"first letter"
2	"beta"	"second letter"
3	"alpha"	"particle"

5	SELECT * FROM WORD_LIST_TABLE LIMIT 2,1	Returns 1 item starting at position 2. Position counting starts at 1 (not zero!). Returns `[["2","beta","second letter"]]`

SQLite Database

Using SQLite database

- Versatile and straightforward to implement
- Structured data that you need to store persistently
- Access, search, and change data frequently
- Primary storage for user or app data
- Cache and make available data fetched from the cloud
- Data can be represented as rows and columns

SQLite software library

Implements SQL database engine that is

- self-contained (requires no other components)
- serverless (requires no server backend)
- zero-configuration (does not need to be configured for your application)
- transactional (changes within a single transaction in SQLite either occur completely or not at all)

Cursors

Cursors

- Data type commonly used for results of queries
- Pointer into a row of structured data ...
- ... think of it as an array of rows
- Cursor class provides methods for moving cursor and getting data
- SQLiteDatabase always presents results as Cursor

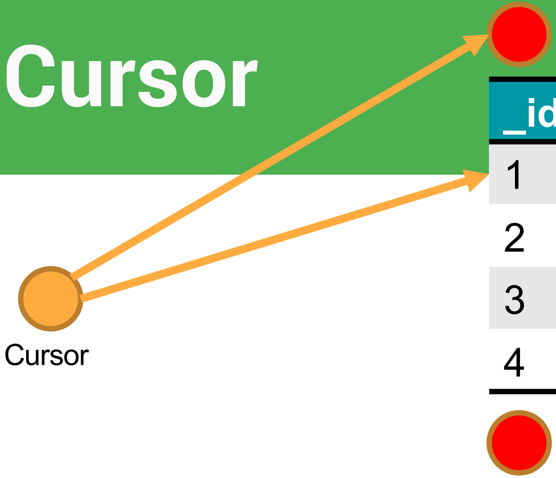
Cursor subclasses

- SQLiteCursor exposes results from a query on a SQLiteDatabase
- MatrixCursor is a mutable cursor implementation backed by an array of Objects that automatically expands internal capacity as needed

Cursor common operations

- getCount()—number of rows in cursor
- getColumnNames()—string array with column names
- getPosition()—current position of cursor
- getString(int column), getInt(int column), ...
- moveToFirst(), moveToNext(), ...
- close() releases all resources and invalidates cursor

Traversing a Cursor



_id	name	phone
1	Nguyen Van A	0987645677
2	Le Van B	0908763256
3	Tran Van C	0912345678
4	Tong Van D	0901234567

```
cursor.moveToPosition(-1)
cursor.isFirst()
cursor.isBeforeFirst()
cursor.moveToNext()
cursor.isFirst()
cursor.getInt(0)
cursor.getString(2)
Cursor.getInt(1)
```

```
cursor.moveToLast()
cursor.getInt(1)
cursor.getString(3)
cursor.moveToPrevious()
cursor.getString(1)
cursor.move(1)
cursor.getString(2)
```

Processing Cursors

```
// Store results of query in a cursor
Cursor cursor = db.rawQuery(...);
try {
    while (cursor.moveToNext()) {
        // Do something with data
     }
} finally {
    cursor.close();
}
```

Content Values

ContentValues

- An instance of ContentValues
  - Represents one table row
  - Stores data as key-value pairs
  - Key is the name of the column
  - Value is the value for the field
- Used to pass row data between methods

ContentValues

```
ContentValues values = new ContentValues();


// Inserts one row.
// Use a loop to insert multiple rows.
values.put(KEY_WORD, "Android");
values.put(KEY_DEFINITION, "Mobile operating system.");

db.insert(WORD_LIST_TABLE, null, values);
```

Implementing SQLite

You always need to ...

1. Create data model
2. Subclass SQLiteOpenHelper
    a. Create constants for tables
    b. onCreate()—create SQLiteDatabase with tables
    c. onUpgrade(), and optional methods
    d. Implement query(), insert(), delete(), update(), count()
3. In MainActivity, create instance of SQLiteOpenHelper
4. Call methods of SQLiteOpenHelper to work with database

Data model

- Class with getters and setters
- One "item" of data (for database, one record or one row)

```
public class WordItem {
    private int mId;
    private String mWord;
    private String mDefinition;
   ...
}
```

SQLiteOpenHelper

SQLite database represented as an SQLiteDatabase object
all interactions with database through SQLiteOpenHelper

- Executes your requests
- Manages your database
- Separates data and interaction from app
- Keeps complex apps manageable

Subclass SQLiteOpenHelper

```
public class WordListOpenHelper extends SQLiteOpenHelper {

    public WordListOpenHelper(Context context) {
       super(context, DATABASE_NAME, null, DATABASE_VERSION);
       Log.d(TAG, "Construct WordListOpenHelper");
    }
}
```

Declare constants for tables

```
private static final int DATABASE_VERSION = 1;
// Has to be 1 first time or app will crash.
private static final String DATABASE_NAME = "dictionary.db";
private static final String WORD_LIST_TABLE = "word_list";

// Column names...
public static final String KEY_ID = "_id";
public static final String KEY_WORD = "word";

// ... and a string array of columns.
private static final String[] COLUMNS = {KEY_ID, KEY_WORD};
```

Read data

```
helper = new WordListOpenHelper(this);
SQLiteDatabase db;
db = helper.getReadableDatabase();


// query or rawQuery using db


db.close();
```

Write data

```
helper = new WordListOpenHelper(this);
SQLiteDatabase db;
db = helper.getWritableDatabase();

// insert or update or delete

db.close();
```

Create an instance of your OpenHelper

```
SQLiteDatabase db;
helper = new WordListOpenHelper(this);
db = helper.getReadableDatabase();
db = helper.getWritableDatabase();
```

Database Operations

Database operations

- query()
- insert()
- update()
- delete()

Database methods for executing queries

- SQLiteDatabase.rawQuery()
  Use when data is under your control and supplied only by your app

- SQLiteDatabase.query()
  Use for all other queries

SQLiteDatabase.rawQuery() format

```
rawQuery(String sql, String[] selectionArgs)
```

- First parameter is SQLite query string
- Second parameter contains the arguments
- Only use if your data is supplied by app and under your full control

rawQuery()

```
String query = "SELECT * FROM WORD_LIST_TABLE";
rawQuery(query, null);

query = "SELECT word, definition FROM
WORD_LIST_TABLE WHERE _id> ? ";

String[] selectionArgs = new String[]{"2"}
rawQuery(query, selectionArgs);
```

SQLiteDatabase.query() format

```
Cursor query (boolean distinct,  String table,
              String[] columns, String selection,
              String[] selectionArgs, String groupBy,
              String having, String orderBy,String limit);
```

query()

```
SELECT * FROM
WORD_LIST_TABLE
WHERE word="alpha"
ORDER BY word ASC
LIMIT 2,1;
```

Returns:

```
[["alpha",
"particle"]]
```

```
String table = "WORD_LIST_TABLE"
String[] columns = new String[]{"*"};
String selection = "word = ?"
String[] selectionArgs = new String[]{"alpha"};
String groupBy = null;
String having = null;
String orderBy = "word ASC"
String limit = "2,1"

query(table, columns, selection, selectionArgs,
groupBy, having, orderBy, limit);
```

insert() format

```
long insert(String table, String nullColumnHack,
                  ContentValues values)
```

- First argument is the table name.
- Second argument is a `String nullColumnHack`.
    - Workaround that allows you to insert empty rows
    - Use `null`
- Third argument must be a ContentValues with values for the row
- Returns the id of the newly inserted item

insert() example

```
newId = db.insert(
    WORD_LIST_TABLE,
    null,
    values);
```

delete() format

```
int delete (String table,
            String whereClause, String[] whereArgs)
```

- First argument is table name
- Second argument is WHERE clause
- Third argument are arguments to WHERE clause

delete() example

```
deleted = db.delete(
    WORD_LIST_TABLE,
    KEY_ID + " =? ",
    new String[]{String.valueOf(id)});
```

update() format

```
int update(String table, ContentValues values,
           String whereClause, String[] whereArgs)
```

- First argument is table name
- Second argument must be ContentValues with new values for the row
- Third argument is WHERE clause
- Fourth argument are the arguments to the WHERE clause

update() example

```
ContentValues values = new ContentValues();
values.put(KEY_WORD, word);

mNumberOfRowsUpdated = db.update(
                WORD_LIST_TABLE,
                values, // new values to insert
                KEY_ID + " = ?",
                new String[]{String.valueOf(id)});
```

Always!

- Always put database operations in try-catch blocks
- Always validate user input and SQL queries

SimpleCursorAdapter

```
public SimpleCursorAdapter (
        Context context,
        int layout, Cursor c,
        String[] from, int[] to,
        int flags)
```

context	Context: The context where the ListView associated with this SimpleListItemFactory is running
layout	int: resource identifier of a layout file that defines the views for this list item. The layout file should include at least those named views defined in "to"
c	Cursor: The database cursor. Can be null if the cursor is not available yet.
from	String: A list of column names representing the data to bind to the UI. Can be null if the cursor is not available yet.
to	int: The views that should display column in the "from" parameter. These should all be TextViews. The first N views in this list are given the values of the first N columns in the from parameter. Can be null if the cursor is not available yet.
flags	int: Flags used to determine the behavior of the adapter, as per CursorAdapter.CursorAdapter(Context, Cursor, int).

Learn more

- Storage Options
- Saving Data in SQL Databases
- SQLiteDatabase class
- ContentValues class
- SQLiteOpenHelper class
- Cursor class
- SQLiteAssetHelper class from Github

What's Next?

- Concept Chapter: 10.2 C SQLite Database
- Practical:
    - 10.2A P SQLite Data Storage
    - 10.2B P Searching an SQLite Database

END