CMPS 121
Databases

Luca de Alfaro

Portions of this presentations are based on content taken from the Android Developers Site. The work was created and shared by the Android Open Source Project and used according to terms described in the Creative Commons 2.5 Attribution License.
What do Databases do?

Two main things:

● Achieve persistency of structured data, with easy retrieval.
● Handle concurrent access.
Opening a database with a helper

```java
public class DictionaryOpenHelper extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION = 2;
    private static final String DICTIONARY_TABLE_NAME = "dictionary";
    private static final String DICTIONARY_TABLE_CREATE =
        "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (
            KEY_WORD + " TEXT, " +
            KEY_DEFINITION + " TEXT);";

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

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(DICTIONARY_TABLE_CREATE);
    }
}
```
Opening a database directly

Not recommended, but:

```java
private static final String DATABASE_NAME = "myDatabase.db";
private static final String DATABASE_TABLE = "mainTable";

private static final String DATABASE_CREATE =
    "create table " + DATABASE_TABLE + " ( _id integer primary key autoincrement," +
    "column_one text not null);";

SQLiteDatabase myDatabase;

private void createDatabase() {
    myDatabase = openOrCreateDatabase(DATABASE_NAME, Context.MODE_PRIVATE, null);
    myDatabase.execSQL(DATABASE_CREATE);
}
```
public Cursor query (boolean distinct, String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)

- **distinct**  true if you want each row to be unique, false otherwise.
- **table**  The table name to compile the query against.
- **columns**  A list of which columns to return. Passing null will return all columns, which is discouraged to prevent reading data from storage that isn't going to be used.
- **selection**  A filter declaring which rows to return, formatted as an SQL WHERE clause (excluding the WHERE itself). Passing null will return all rows for the given table.
- **selectionArgs**  You may include ?s in selection, which will be replaced by the values from selectionArgs, in order that they appear in the selection. The values will be bound as Strings.
- **groupBy**  A filter declaring how to group rows, formatted as an SQL GROUP BY clause (excluding the GROUP BY itself). Passing null will cause the rows to not be grouped.
- **having**  A filter declare which row groups to include in the cursor, if row grouping is being used, formatted as an SQL HAVING clause (excluding the HAVING itself). Passing null will cause all row groups to be included, and is required when row grouping is not being used.
- **orderBy**  How to order the rows, formatted as an SQL ORDER BY clause (excluding the ORDER BY itself). Passing null will use the default sort order, which may be unordered.
- **limit**  Limits the number of rows returned by the query, formatted as LIMIT clause. Passing null denotes no LIMIT clause.
A BAD example

// Return all rows for columns one and three, no duplicates
String[] result_columns = new String[] {KEY_ID, KEY_COL1, KEY_COL3};

Cursor allRows = myDatabase.query(true, DATABASE_TABLE, result_columns,
   null, null, null, null, null, null);

// Return all columns for rows where column 3 equals a set value
// and the rows are ordered by column 5.
String where = KEY_COL3 + "=" + requiredValue;
String order = KEY_COL5;
Cursor myResult = myDatabase.query(DATABASE_TABLE, null, where,
   null, null, null, order);

Why is this bad?
Using a cursor

Methods:

- `close()` - closes it
- `deactivate(); requery()` - temporary release
- `getColumnCount(), getColumnNames(), getColumnColumnName(int idx)`
- `getPosition()` - idx of current row in row set
- `getInt(int col_idx), getLong(int col_idx), getString(int col_idx)`

To move around:

- `isFirst(), isLast()`
- `moveToFirst(), moveToLast(), moveToNext()` -- return a boolean if the move succeeds
int GOLD_HOARDED_COLUMN = 2;
float totalHoard = 0f;

Cursor myGold = myDatabase.query("GoldHoards", null, null, null, null, null, null, null);

// Make sure there is at least one row.
if (myGold.moveToFirst()) {
    // Iterate over each cursor.
    do {
        float hoard = myGold.getFloat(GOLD_HOARDED_COLUMN);
        totalHoard += hoard;
    } while(myGold.moveToNext());
}

Float averageHoard = totalHoard / myGold.getCount();
Inserting values in a database

// Create a new row of values to insert.
ContentValues newValues = new ContentValues();

// Assign values for each row.
newValues.put(COLUMN_NAME, newValue);
// [ ... Repeat for each column ... ]

// Insert the row into your table
myDatabase.insert(DATABASE_TABLE, null, newValues);
Updating, deleting, etc.

Update:
update(String table, ContentValues values, String whereClause, String[] whereArgs)

Delete:
delete(String table, String whereClause, String[] whereArgs)

Execute non-query statements (table creations, etc):
execSQL(String sql)
Content providers

An abstraction for a database, that can be shared among applications. Fundamental way of sharing data in Android (e.g., contacts)
Using a content provider

// Queries the user dictionary and returns results
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI, // The content URI of the words table
    mProjection, // The columns to return for each row
    mSelectionClause, // Selection criteria
    mSelectionArgs, // Selection criteria
    mSortOrder); // The sort order for the returned rows

Example of URI: content://user_dictionary/words

Or, build it like this:

Uri singleUri =
    ContentUri.withAppendedId(UserDictionary.Words.CONTENT_URI, 4);
Constructing the query

// A "projection" defines the columns that will be returned for each row
String[] mProjection =
{
    UserDictionary.Words._ID, // Contract class constant for the _ID column name
    UserDictionary.Words.WORD, // Contract class constant for the word column name
    UserDictionary.Words.LOCALE // Contract class constant for the locale column name
};

// Defines a string to contain the selection clause
String mSelectionClause = null;

// Initializes an array to contain selection arguments
String[] mSelectionArgs = {""};
Executing the query

// Does a query against the table and returns a Cursor object
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI, // The content URI of the words table
    mProjection, // The columns to return for each row
    mSelectionClause, // Either null, or the word the user entered
    mSelectionArgs, // Either empty, or the string the user entered
    mSortOrder); // The sort order for the returned rows

// Some providers return null if an error occurs, others throw an exception
if (null == mCursor) {
    // ...
}

// If the Cursor is empty, the provider found no matches
else if (mCursor.getCount() < 1) {
    // ...
}

} else {
    // Insert code here to do something with the results
}
Getting data

// Determine the column index of the column named "word"
int index = mCursor.getColumnIndex(UserDictionary.Words.WORD);

if (mCursor != null) {
    /* Moves to the next row in the cursor. Before the first movement in the cursor, the
    * "row pointer" is -1, and if you try to retrieve data at that position you will get an
    * exception.
    */
    while (mCursor.moveToNext()) {

        // Gets the value from the column.
        newWord = mCursor.getString(index);
        // ...
    }
} else {
    // Insert code here to report an error if the cursor is null or the provider threw an exception.
}
Inserting data

// Defines a new Uri object that receives the result of the insertion
Uri mNewUri;

// Defines an object to contain the new values to insert
ContentValues mNewValues = new ContentValues();

/*
 * Sets the values of each column and inserts the word. The arguments to the "put"
 * method are "column name" and "value"
 */
mNewValues.put(UserDictionary.Words.APP_ID, "example.user");
mNewValues.put(UserDictionary.Words.LOCALE, "en_US");
mNewValues.put(UserDictionary.Words.WORD, "insert");
mNewValues.put(UserDictionary.Words.FREQUENCY, "100");

mNewUri = getContentResolver().insert(
    UserDictionary.Word.CONTENT_URI, // the user dictionary content URI
    mNewValues // the values to insert
);
Updating data

// Defines an object to contain the updated values
ContentValues mUpdateValues = new ContentValues();

// Defines selection criteria for the rows you want to update
String mSelectionClause = UserDictionary.Words.LOCALE + "LIKE ?";
String[] mSelectionArgs = {"en_%"};

// Defines a variable to contain the number of updated rows
int mRowsUpdated = 0;

// Sets the updated value and updates the selected words.
mUpdateValues.putNull(UserDictionary.Words.LOCALE);

mRowsUpdated = getContentResolver().update(
    UserDictionary.Words.CONTENT_URI, // the user dictionary content URI
    mUpdateValues // the columns to update
    mSelectionClause // the column to select on
    mSelectionArgs // the value to compare to
);
Deleting data

// Defines selection criteria for the rows you want to delete
String mSelectionClause = UserDictionary.Words.APP_ID + " LIKE ?";
String[] mSelectionArgs = {"user"};

// Defines a variable to contain the number of rows deleted
int mRowsDeleted = 0;

...  

// Deletes the words that match the selection criteria
mRowsDeleted = getContentResolver().delete(
    UserDictionary.Words.CONTENT_URI, // the user dictionary content URI
    mSelectionClause, // the column to select on
    mSelectionArgs // the value to compare to
);