Data storage overview
SQLLite databases
http://www.android.com/
Data storage overview

- **Assets (assets) and Resources (res/raw)**
  - Private data installed with the application (**read-only**)
  - Use the `android.content.res.xxx` classes/methods (Resources and AssetManager) or the asset uri: "file:///android_asset/***" for access

- **Shared Preferences**
  - Private primitive application data in key-value pairs

- **Internal storage**
  - Private data on the device memory (files)

- **External storage**
  - Public data on the shared external storage (files). Apps can only write to files and folders that they have created or have taken ownership of

- **SQLite Databases**
  - Structured data in a private databases

- **Network Connection**
  - Data on the web; read write to server (e.g. a DB)

- When app is uninstalled or "resetted" all private data is removed
Databases

• Organize, store, and retrieve (large amounts of) structured data
• SQL (Structured Query Language)
• S( Search)CRUD
  – Create databases (including tables etc.)
  – Allow data creation and maintenance
  – Search for data and other access
• DBMS (DataBase Management Systems)
  – Atomicity - modifications must follow an "all or nothing" rule
  – Consistency - only valid data will be written to the database
  – Isolation - operations cannot access data that has been modified during a transaction that has not yet completed
  – Durability - once the user has been notified of a transaction's success the transaction will not be lost
SQLite

• SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. Some benefits are:
  – Lightweight, reliable, standards compliant, open-source, ...

• A SQLite database is an integrated part of the application that created it
  – Reducing external dependencies
  – Simplifies transaction locking and synchronization

• SQLite is the most widely deployed SQL database engine in the world
  – It is almost easier to mention products that NOT use SQLite than list products that use SQLite!
  – Chrome, Firefox, all mobile OSs except Windows phone, ...
  – Embedded systems in all kinds of industry, aeroplanes, …
For more complex data structures, a database provides a quicker and more flexible access method than flat files or shared preferences.

SQLite provides full relational database capability utilizing SQL commands.

Each application that uses SQLite has its own instance of the database, which by default is only accessible from the application itself.

- Apps signed with the same key may share the database.
- A **Content Provider** can be used to share the database information between other applications.

The database is stored in the `/data/data/<package_name>/databases` folder of an Android device.
Many native Android apps use SQLite DBs
- Messaging (SMS, MMS), People (contacts), MediaStore, etc.

Simple file based relation database
- Not recommended to store BLOBs (Binary Large Objects) as bitmaps, media, etc. Store only the Uri to BLOBs in DB!

adb push <local file> <phone file> or adb pull <phone file> <local file>
- Open on desktop computer with tools as Database 4 .NET, SQLiteSpy, SQLite Database Browser, etc.
SQLite Databases 3

• When developing it is convenient to log in via ADB and query the database
• To get a shell in the emulator or phone: `adb shell [-s (emu serial nr)]` To get the serial#: `adb devices`
  - Open DB with: `sqlite3 <path to DB>/db-name`
  - Issue commands as: `.databases`, `.tables`, `.help`, SQL, ...

![ADB Shell Example]

```plaintext
C:\android-sdk-windows\tools>adb shell
# sqlite3 /data/data/se.du.database/databases/books
sqlite3 /data/data/se.du.database/databases/books
SQLite version 3.6.22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .databases
.name file

0 main /data/data/se.du.database/databases/books

sqlite> .tables
.tables
android_metadata titles
sqlite> select * from titles;
select * from titles;
```
• Or use a browse plugin as com.questoid.sqlitebrowser_1.2.0.jar
• More advanced full access plugins are available with direct connection over USB or the network as DbAndroid
Open or create a database

• Executing some SQL statements (function-based)

```java
// Open a new private SQLiteDatabase associated with this Context's
// application package. Create the database file if it doesn't exist.
// SQLiteDatabase openOrCreateDatabase (String name, int mode,
//   SQLiteDatabase.CursorFactory factory)
SQLiteDatabase db = mContext.openOrCreateDatabase(
    "books.db",
    Context.MODE_PRIVATE,
    null);

// The language codes are two-letter lowercase ISO language codes
// (such as "en") as defined by ISO 639-1.
db.setLocale(Locale.getDefault());

// Sets the database version.
db.setVersion(1);

// Convenience methods for inserting, updating and deleteing
// rows in the database.
int num_rows_affected = db.insert(table, nullColumnHack, values);
num_rows_affected = db.update(table, values, whereClause, whereArgs);
num_rows_affected = db.delete(table, whereClause, whereArgs);

// Query the given table, returning a Cursor over the result set.
// The Cursor object is a reference to the data
Cursor cur = db.query(table, columns, selection, selectionArgs,
    groupBy, having, orderBy);
cur = db.query(distinct, table, columns, selection, selectionArgs,
    groupBy, having, orderBy, limit);
```
• query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)
• SELECT _id, word, definition FROM table_words
• Translates into

```java
db.query("table_word",
    new String[] {"_id",
    "word", "definition"},,
    null, null, null, null, null
);
```
Example DB, execSQL()

- DatabaseTest example app - A database with book titles
  - A modified DatabaseActivity and DBAdapter class which in turn contains a static DatabaseHelper class which extends the SQLiteOpenHelper class

- execSQL() can execute a single SQL statement that is **NOT** a SELECT or any other SQL statement that does not return any data

- For complex queries returning a cursor - use SQLiteQueryBuilder.query()

```java
// Execute a single SQL statement that is NOT a SELECT or any other SQL statement that returns data.
db.execSQL("DROP TABLE IF EXISTS books");

String word = "alpha", definition = "aplpha is...";
String ins = "INSERT INTO table_word (word, definition) VALUES('" + word + "','" + definition + ");
db.execSQL(ins);

String upd = "UPDATE table_word set word = '" + word + ", definition = '" + definition + "' WHERE _id = " + row;
db.execSQL(upd);

// This is a convience class that helps build SQL queries to be sent to SQLiteDatabase objects.
SQLiteQueryBuilder qBuilder = new SQLiteQueryBuilder();
qBuilder.setTables("example et JOIN secondtable st ON et.id = st.example_id");
qBuilder.appendWhere(" et.someRow = ? ");
Cursor cursor = qBuilder.query(sqlitedatabase, projection, selection, selectionArgs, null, null, sortOrder);
startManagingCursor(cursor);
```
Create a table and delete a row

- The `Activity.startManagingCursor(cursor)` on previous slide allows the activity to take care of managing the given Cursor's lifecycle based on the activity's lifecycle - it requires that the row key id is named "_id".
- `startManagingCursor(cursor)` is deprecated since API 11. Use the new CursorLoader class with LoaderManager instead! It will manage the cursor in a similar way.

```java
private static final String DATABASE_CREATE =
    "create table titles (_id integer primary key autoincrement,"
    + "isbn text not null, title text not null, "
    + "publisher text not null);

public void CreateDB(SQLiteDatabase db) {
    try {
        db.execSQL(DATABASE_CREATE);
    } catch (SQLiteException ex) {
        Log.e("Create table exception", ex.getMessage());
    }
}

// deletes a particular row in a db - delete(table, whereClause, whereArgs);
public int deleteRow(SQLiteDatabase db, long rowId) {
    int ret = -1;
    try {
        ret = db.delete("titles", "_id" + "=" + rowId, null);
    } catch (SQLiteException ex) {
        Log.e("deleteRow exception caught", ex.getMessage());
    }
    // delete() returns the number of rows affected if a whereClause is passed in, 0 otherwise.
    return ret;
}
Insert and update

- Use **ContentValues** to provide column names and column values

```java
public long insertTitle(SQLiteDatabase db, String isbn, String title, String publisher) {
    long ret = -1;
    ContentValues initialValues = new ContentValues();
    initialValues.put("isbn", isbn);
    initialValues.put("title", title);
    initialValues.put("publisher", publisher);
    try{
        ret = db.insert("titles", null, initialValues);
    } catch(SQLiteException ex) {
        Log.e("insertTitle exception caught", ex.getMessage());
    }
    return ret; // insert() returns the row id or -1
}

public int updateTitle(SQLiteDatabase db, long rowId, String isbn, String title, String publisher) {
    int ret = -1;
    ContentValues args = new ContentValues();
    args.put("isbn", isbn);
    args.put("title", title);
    args.put("publisher", publisher);
    try{
        ret = db.update("titles", args, "_id" + "=" + rowId, null);
    } catch(SQLiteException ex) {
        Log.e("updateTitle exception caught", ex.getMessage());
    }
    return ret; // update() returns the number of rows updated
}
```
Query and Cursor 1

• Query results are accessed using a Cursor, allowing random access to the query result.
• Common used Cursor methods
  - `Move`ToFirst() /...ToNext() /...ToPrevious() /...ToPosition(), ...
  - `getCount()` /...`ColumnName()` /...`ColumnNames()` /...`Position()`, ...
• For longer cursor tasks – manage the cursor as part of the application lifecycle
  - `onPause()` - deactivate cursor (deprecated API 16)
  - `onResume()` - requery cursor (deprecated API 11)
  - `OnDestroy()` - close cursor
• CursorLoader class and LoaderManager
  - An asynchronous framework which offloads the UI thread and simplifies your cursor management.
  - Manage the Loader in the Activity/Fragment lifecycle methods and special loader callback methods.
public void iterateAllTitles()
{
    mDBA.open();
    Cursor c = mDBA.getAllTitles();
    Toast.makeText(this, "iterateAllTitles()", Toast.LENGTH_SHORT).show();

    if(c.moveToFirst()){
        do{
            DisplayTitle(c);
        }while (c.moveToNext());
    }
    mTV.setText(mstTextView);
    mDBA.close();
}

public Cursor getAllTitles()
{
    Cursor c = null;
    try{
        // public Cursor query (String table, String[] columns, String selection, 
        // String[] selectionArgs, String groupBy, String having, String orderBy)
        c = mDB.query("titles", new String[] {
            ",isbn", "title", "publisher"},
            null, null, null, null, null);
    }
    catch(SQLiteException ex) {
        Log.d("getAllTitles exception caught", ex.getMessage());
    }
    return c;
}

// display the columns (0, 1, 2, 3)
// from where the row cursor is at
public void DisplayTitle(Cursor c)
{
    String row = "id: " + c.getString(0) + "\n" +
                "ISBN: " + c.getString(1) + "\n" +
                "TITLE: " + c.getString(2) + "\n" +
                "PUBLISHER: " + c.getString(3) + "\n";

    mstTextView += row + "\n";
SimpleCursorAdapter

- An easy adapter to map columns from a cursor to TextViews or ImageViews defined in an XML file – Note: Runs on the UI thread!
- You can specify which columns you want, which views you want to display the columns, and the XML file that defines the appearance of these views
- You should use a Loader instead (Loader examples in next presentation)

```java
public class MyListActivity extends ListActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Cursor mCursor = getContacts(); // returns a cursor to the queried contacts DB in the phone
        startManagingCursor(mCursor); // deprecated since API 11
        int flags = 0;
        // Now create a new list adapter bound to the cursor.
        // ListAdapter is designed for binding to a Cursor.
        ListAdapter adapter = new SimpleCursorAdapter(this, // Context.
            android.R.layout.two_line_list_item, // Specify the row template to use (here, two columns bound to the two retrieved cursor rows).
            mCursor, // Pass in the cursor to bind to.
            new int[] { android.R.id.text1, android.R.id.text2 }, flags);
        // Bind to our new adapter.
        setListAdapter(adapter);
    }
}
```
Transactions

- Handle multiple operations that should happen all together, or not at all
- `setTransactionSuccessful()` plus `endTransaction()` commits the changes
- `endTransaction()` without `setTransactionSuccessful()` causes a roll back on all changes

```java
// Begins a transaction in EXCLUSIVE mode.
db.beginTransaction();

try {
    // insert/delete/update records
    // Marks the current transaction as successful.
    db.setTransactionSuccessful();
} catch (SQLiteException ex) {
    Log.d("Transaction exception", ex.getMessage());
}
finally {
    // End a transaction.
    db.endTransaction();
}
```
SQLiteOpenHelper

• Create a subclass implementing `onCreate(SQLiteDatabase), onUpgrade(SQLiteDatabase, int, int)` and optionally `onOpen(SQLiteDatabase), and this class takes care of opening the database if it exists, creating it if it does not, and upgrading it as necessary.

• Transactions are used to make sure the database is always in a sensible state.

• This class makes it easy for ContentProvider implementations to delay opening and upgrading the database until first use, to avoid blocking application startup with long-running database upgrades.

```java
private static class DatabaseHelper extends SQLiteOpenHelper {
    DatabaseHelper(Context context, String name, CursorFactory factory, int version) {
        super(context, name, factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        try {
            db.execSQL(DATABASE_CREATE);
        } catch (SQLiteException ex) {
            Log.d("Create table exception", ex.getMessage());
        }
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.w(Consts.TAG, "Upgrading database from version " + oldVersion + " to " + newVersion + ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS " + Consts.DATABASE_TABLE);
        onCreate(db);
    }
}
```
(Android) Databases, design considerations

- Use (extend from) SQLiteOpenHelper to manage database creation and version management
- Write an “adapter class”, with (strongly typed) methods, hiding the database manipulation, and constants representing keys
  - Example DbAdapter.java in the books Database test example app
- Model rows as class instances
- SQLite does not enforce foreign key constraints – use triggers instead (via execSQL), trigger == attached stored procedure
- Don’t store large files (media etc.) in the database
- Data type integrity and referential integrity is not maintained in SQLite
- Full Unicode support (UTF-16) is optional, UTF-8 is used by default
Example database 1

- It's good practice to create a DB adapter class to encapsulate all the complexities of accessing the database so it's transparent to the calling code

```java
// From: http://www.devx.com/wireless/Article/40842
private static final String DATABASE_CREATE = "create table titles (_id integer primary key autoincrement, " + "isbn text not null, title text not null, publisher text not null);";
private final Context mContext;
private DatabaseHelper mDBHelper;
private SQLiteDatabase mDB;
// SQLiteDatabase has methods to create, delete, execute SQL commands and perform other common database management tasks
public DBAdapter(Context ctx) {
    this.mContext = ctx;
    mDBHelper = new DatabaseHelper(mContext, DATABASE_NAME, null, DATABASE_VERSION);
}
// Within the DBAdapter class, we extend the DataBaseHelper with the SQLiteOpenHelper class –
// an Android helper class for database creation and versioning management.
// In particular, we override the onCreate() and onUpgrade() methods.
private static class DatabaseHelper extends SQLiteOpenHelper {
    DatabaseHelper(Context context, String name, CursorFactory factory, int version) {
        super(context, name, factory, version);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(DATABASE_CREATE); //in this string we have our SQL create table statement
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.w(TAG, "Upgrading database from version " + oldVersion + " to " + newVersion + ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS " + DATABASE_TABLE);
        onCreate(db);
    }
```
Example database 2

• Some usage examples from DatabaseActivity and DBAdapter

```java
public class DatabaseActivity extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mDBA = new DBAdapter(this);
    }
    ...
    mDBA.open();
    mDBA.insertTitle("0470285818", "C# 2008 Programmer's Reference", "Wrox");
}
```

```java
public class DBAdapter {
    private DatabaseHelper mDBHelper;
    // SQLiteDatabase has methods to create, delete, execute SQL commands and perform other common database management tasks
    private SQLiteDatabase mDB;

    //---opens the database---
    public DBAdapter open() throws SQLiteException {
        return mDB = mDBHelper.getWritableDatabase();
    }
    //---insert a title into the database, returns row id, -1 if error---
    public long insertTitle(String isbn, String title, String publisher) {
        ContentValues initialValues = new ContentValues();
        initialValues.put(KEY_ISBN, isbn);
        initialValues.put(KEY_TITLE, title);
        initialValues.put(KEY_PUBLISHER, publisher);
        return mDB.insert(DATABASE_TABLE, null, initialValues);
    }
}
```
Example database 3

• Some more usage examples from the DatabaseActivity using the DBAdepter class

```java
public class DatabaseActivity extends Activity {

    //---get a cursor for a title and put it in a string---
    Cursor c = mDBA.getTitle(id);
    // display the columns (0, 1, 2, 3) from where the row cursor is at
    String row = "id: " + c.getString(0) + "\n" + "ISBN: " + c.getString(1) + "\n" + "TITLE: " + c.getString(2) + "\n" + "PUBLISHER: " + c.getString(3) + "\n";
    mDBA.deleteTitle(id);
}
```

```
public class DBAdepter {

    //---retrieves a cursor for a particular title---
    //public Cursor query (boolean distinct, String table, String[] columns, String selection,
    //String[] selectionArgs, String groupBy, String having, String orderBy, String limit)
    public Cursor getTitle(long rowId) {
        return mDB.query(true, DATABASE_TABLE, new String[] {
            KEY_ROWID, KEY_ISBN, KEY_TITLE, KEY_PUBLISHER
        },
        KEY_ROWID + "=" + rowId, null, null, null, null, null);
    }

    //---deletes a particular title---
    public boolean deleteTitle(long rowId) {
        return mDB.delete(DATABASE_TABLE, KEY_ROWID + "=" + rowId, null) > 0;
    }
}
```