Report of CS 522 Assignment 4

Name: Yunfeng Wei
CWID: 10394963
E-mail: ywei14@stevens.edu
Video: In the ZIP Archive file
BookstoreWithContentProvider App: BookStoreWithContentProvider.MOV
BookstoreWIthEntityManager App: BookstoreWIthEntityManager.MOV
ChatServerWithContentProvider App: ChatServerWithContentProvider.MOV

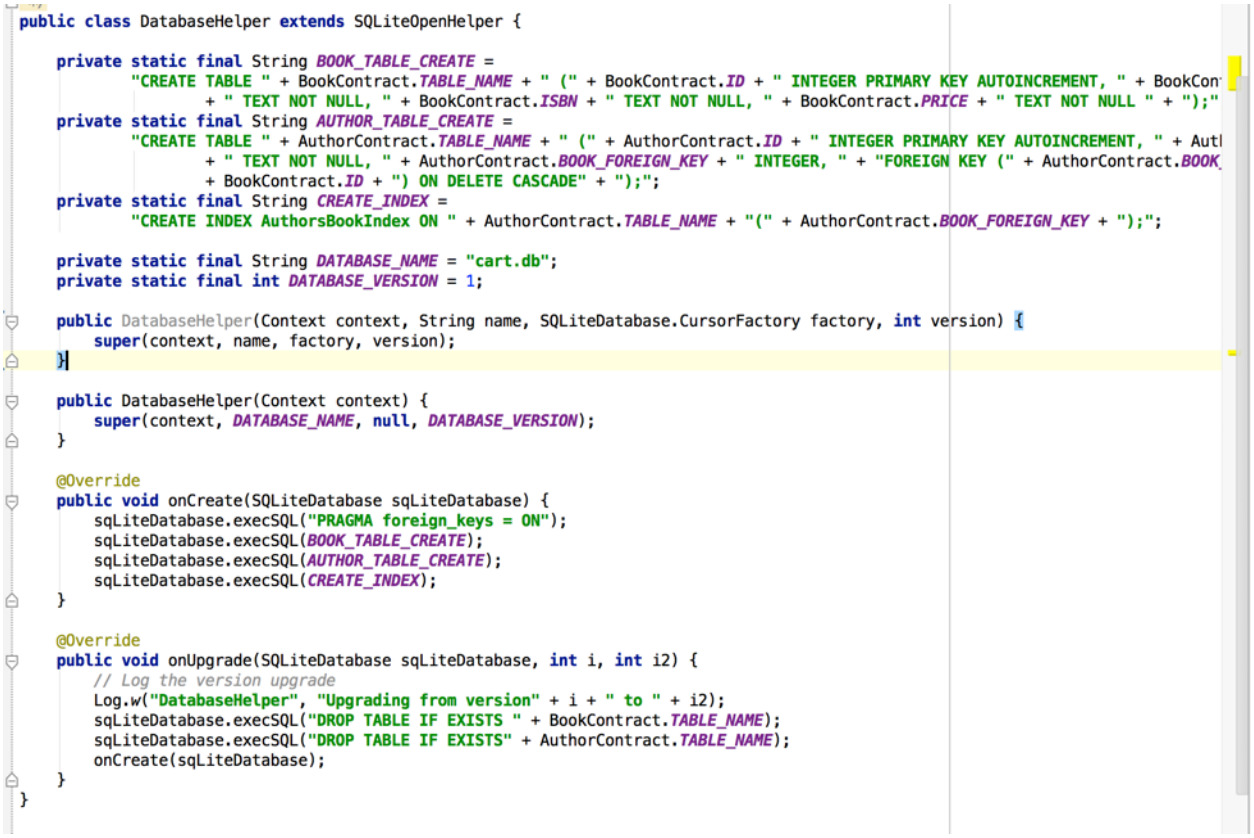
Part 1: BookStoreWithContentProvider

1. Define a BookAdapter class in the package, extending ResourceCursorAdapter and specializing the bindView method as Figure 1.1

```
/**
 * Created by wyf920621 on 2/27/15.
 */
public class BookAdapter extends ResourceCursorAdapter {
    protected final static int ROW_LAYOUT = android.R.layout.simple_list_item_2;

    public BookAdapter(Context context, Cursor cursor) { super(context, ROW_LAYOUT, cursor, 0); }

    @Override
    public View newView(Context context, Cursor cursor, ViewGroup parent) {
        LayoutInflater inflater = (LayoutInflater)context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        return inflater.inflate(ROW_LAYOUT, parent, false);
    }

    @Override
    public void bindView(View view, Context context, Cursor cursor) {
        TextView titleLine = (TextView)view.findViewById(android.R.id.text1);
        TextView authorLine = (TextView)view.findViewById(android.R.id.text2);
        String title = BookContract.getTitle(cursor);
        Author[] authors = BookContract.getAuthors(cursor);
        titleLine.setText(title);
        authorLine.setText(authors[0].toString());
    }
}
```

Figure 1.1

2. Move the DatabaseHelper out of the BookAdapter and remove the BookAdapter, as Figure 2.1



```
public class DatabaseHelper extends SQLiteOpenHelper {

    private static final String BOOK_TABLE_CREATE =
            "CREATE TABLE " + BookContract.TABLE_NAME + " (" + BookContract.ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " + BookCon
                    + " TEXT NOT NULL, " + BookContract.ISBN + " TEXT NOT NULL, " + BookContract.PRICE + " TEXT NOT NULL " + ");"
    private static final String AUTHOR_TABLE_CREATE =
            "CREATE TABLE " + AuthorContract.TABLE_NAME + " (" + AuthorContract.ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " + Aut
                    + " TEXT NOT NULL, " + AuthorContract.BOOK_FOREIGN_KEY + " INTEGER, " + "FOREIGN KEY (" + AuthorContract.BOOK
                    + BookContract.ID + ") ON DELETE CASCADE" + ");";
    private static final String CREATE_INDEX =
            "CREATE INDEX AuthorsBookIndex ON " + AuthorContract.TABLE_NAME + "(" + AuthorContract.BOOK_FOREIGN_KEY + ");";

    private static final String DATABASE_NAME = "cart.db";
    private static final int DATABASE_VERSION = 1;

    public DatabaseHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
    }

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

    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
        sqLiteDatabase.execSQL("PRAGMA foreign_keys = ON");
        sqLiteDatabase.execSQL(BOOK_TABLE_CREATE);
        sqLiteDatabase.execSQL(AUTHOR_TABLE_CREATE);
        sqLiteDatabase.execSQL(CREATE_INDEX);
    }

    @Override
    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i2) {
        // Log the version upgrade
        Log.w("DatabaseHelper", "Upgrading from version" + i + " to " + i2);
        sqLiteDatabase.execSQL("DROP TABLE IF EXISTS " + BookContract.TABLE_NAME);
        sqLiteDatabase.execSQL("DROP TABLE IF EXISTS" + AuthorContract.TABLE_NAME);
        onCreate(sqLiteDatabase);
    }
}
```

Figure 2.1

3. Define a content provider called BookProvider, place it in the sub package providers, define the string AUTHORITY, define the CONTENT_URI, CONTENT_PATH, contentType, contentItemType and withExtendedPath method as Figure 3.1. Define a uriMatcher and add separate URI into the matcher as Figure 3.2. Override the delete, insert, getType, query and update method as Figure 3.3, Figure 3.4, Figure 3.5 and Figure 3.6

```java
public class BookProvider extends ContentProvider {
    public static final String AUTHORITY = "edu.stevens.cs522.bookstore";
    // CONTENT_URI(authority, path)
    public static Uri CONTENT_URI(String authority, String path) {
        return new Uri.Builder().scheme("content").authority(authority).path(path).build();
    }

    // withExtendedPath(Uri uri, String... path)
    public static Uri withExtendedPath(Uri uri, String... path) {
        Uri.Builder builder = uri.buildUpon();
        for (String s : path)
            builder.appendPath(s);
        return builder.build();
    }

    // CONTENT_PATH(Uri uri)
    public static String CONTENT_PATH(Uri uri) { return uri.getPath().substring(1); }

    // contentType(content)
    public static String contentType(String content) {
        return "vnd.android.cursor/vnd." + AUTHORITY + "." + content + "s";
    }

    // contentItemType(content)
    public static String contentItemType(String content) {
        return "vnd.android.cursor/vnd." + AUTHORITY + "." + content;
    }
```

Figure 3.1

```java
    private DatabaseHelper databaseHelper;

    // To differentiate between the different URI requests
    private static final int BOOK_ALL_ROWS = 1;
    private static final int BOOK_SINGLE_ROW = 2;
    private static final int AUTHOR_ALL_ROWS = 3;
    private static final int AUTHOR_SINGLE_ROW = 4;

    // Used to dispatch operation based on URI
    private static final UriMatcher uriMatcher;
    static {
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI(AUTHORITY, BookContract.CONTENT_PATH, BOOK_ALL_ROWS); // Books
        uriMatcher.addURI(AUTHORITY, BookContract.CONTENT_PATH_ITEM, BOOK_SINGLE_ROW); // Books/#
        uriMatcher.addURI(AUTHORITY, AuthorContract.CONTENT_PATH, AUTHOR_ALL_ROWS); // Authors
        uriMatcher.addURI(AUTHORITY, AuthorContract.CONTENT_PATH_ITEM, AUTHOR_SINGLE_ROW); // Authors/#
    }
```

Figure 3.2

```java
@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
    SQLiteDatabase database = databaseHelper.getWritableDatabase();
    database.execSQL("PRAGMA foreign_keys = ON");
    // Implement this to handle requests to delete one or more rows.
    int res;
    switch (uriMatcher.match(uri)) {
        case BOOK_ALL_ROWS:
            res = database.delete(BookContract.TABLE_NAME, null, null);
            return res;
        case BOOK_SINGLE_ROW:
            selection = BookContract.ID + "=?";
            String[] args = {String.valueOf(BookContract.getId(uri))};
            res = database.delete(BookContract.TABLE_NAME, selection, args);
            return res;
        default:
            throw new IllegalArgumentException("Unsupported URI: " + uri);
    }
}

@Override
public String getType(Uri uri) {
    // TODO: Implement this to handle requests for the MIME type of the data
    // at the given URI.
    switch (uriMatcher.match(uri)) {
        case BOOK_ALL_ROWS:
            return BookContract.contentType;
        case BOOK_SINGLE_ROW:
            return BookContract.contentItemType;
        case AUTHOR_ALL_ROWS:
            return AuthorContract.contentType;
        case AUTHOR_SINGLE_ROW:
            return AuthorContract.contentItemType;
        default:
            throw new IllegalArgumentException("Unsupported URI: " + uri);
    }
}
```

Figure 3.3

```java
@Override
public Uri insert(Uri uri, ContentValues values) {
    SQLiteDatabase database = databaseHelper.getWritableDatabase();
    database.execSQL("PRAGMA foreign_keys = ON");
    long rowId;
    switch (uriMatcher.match(uri)) {
        case BOOK_ALL_ROWS: // INSERT A BOOK
            rowId = database.insert(BookContract.TABLE_NAME, null, values);
            if (rowId > 0) {
                Uri instanceUri = BookContract.CONTENT_URI(String.valueOf(rowId));
                ContentResolver cr = getContext().getContentResolver();
                cr.notifyChange(instanceUri, null);
                return instanceUri;
            }
            throw new SQLException("Insertion failed");
        case AUTHOR_ALL_ROWS: // INSERT AN AUTHOR
            rowId = database.insert(AuthorContract.TABLE_NAME, null, values);
            if (rowId > 0) {
                Uri instanceUri = AuthorContract.CONTENT_URI(String.valueOf(rowId));
                ContentResolver cr = getContext().getContentResolver();
                cr.notifyChange(instanceUri, null);
                return instanceUri;
            }
            throw new SQLException("Insertion failed");
        default:
            throw new SQLException("Insertion failed");
    }
}

@Override
public boolean onCreate() {
    // TODO: Implement this to initialize your content provider on startup.
    databaseHelper = new DatabaseHelper(getContext());
    databaseHelper.getWritableDatabase().execSQL("PRAGMA foreign_keys = ON");
    return true;
}
```

Figure 3.4

```
@Override
public Cursor query(Uri uri, String[] projection, String selection,
                    String[] selectionArgs, String sortOrder) {
    SQLiteDatabase database;
    try {
        database = databaseHelper.getWritableDatabase();
    } catch (SQLException e) {
        database = databaseHelper.getReadableDatabase();
    }
    database.execSQL("PRAGMA foreign_keys = ON");
    // TODO: Implement this to handle query requests from clients.
    String query;
    switch (uriMatcher.match(uri)) {
        case BOOK_ALL_ROWS:
            query = "SELECT " + BookContract.TABLE_NAME + "." + BookContract.ID + ", " + BookContract.TITLE + ", " + BookCont
                    + ", " + BookContract.ISBN + ", GROUP_CONCAT(" + AuthorContract.NAME + ",\'" + BookContract.SEPARATE_CHAR
                    BookContract.TABLE_NAME + " LEFT OUTER JOIN " + AuthorContract.TABLE_NAME + " ON " + BookContract.TABLE_N
                    AuthorContract.BOOK_FOREIGN_KEY + " GROUP BY " + BookContract.TABLE_NAME + "." + BookContract.ID + ", " +
                    + BookContract.PRICE + ", " + BookContract.ISBN + ";";
            return database.rawQuery(query, null);
        case BOOK_SINGLE_ROW:
            query = "SELECT " + BookContract.TABLE_NAME + "." + BookContract.ID + " AS " + BookContract.ID + ", " + BookContr
                    + ", " + BookContract.TABLE_NAME + "." + BookContract.ISBN + " AS " + BookContract.ISBN + ", GROUP_CONCAT
                    BookContract.TABLE_NAME + " LEFT OUTER JOIN " + AuthorContract.TABLE_NAME + " ON " + BookContract.TABLE_N
                    AuthorContract.BOOK_FOREIGN_KEY + " WHERE " + BookContract.TABLE_NAME + "." + BookContract.ID + " = ?" +
            return database.rawQuery(query, new String[] {String.valueOf(BookContract.getId(uri))});
        default:
            throw new UnsupportedOperationException("Unknown uri: " + uri);
    }

}
```

Figure 3.5

```
@Override
public int update(Uri uri, ContentValues values, String selection,
                  String[] selectionArgs) {
    // TODO: Implement this to handle requests to update one or more rows.
    SQLiteDatabase database = databaseHelper.getWritableDatabase();
    database.execSQL("PRAGMA foreign_keys = ON");
    int res;
    switch (uriMatcher.match(uri)) {
        case BOOK_ALL_ROWS:
            res = database.update(BookContract.TABLE_NAME, values, null, null);
            return res;
        case BOOK_SINGLE_ROW:
            selection = BookContract.ID + "=?";
            String[] args = {String.valueOf(BookContract.getId(uri))};
            res = database.update(BookContract.TABLE_NAME, values, selection, args);
            return res;
        default:
            throw new IllegalArgumentException("Unsupported URI: " + uri);
    }
}
```

Figure 3.6

4. Modify the AuthorContract and BookContract class, define TABLE_NAME, CONTENT, CONTENT_URI, CONTENT_URI(rowId), CONTENT_PATH, CONTENT_PATH_ITEM, contentType and contentItemType for each of them as Figure 4.1 and Figure 4.2

```java
 */
public class AuthorContract {
    public static final String TABLE_NAME = "Authors";
    public static final String ID = "_id";
    public static final String NAME = "name";
    public static final String BOOK_FOREIGN_KEY = "book_fk";
    public static final String CONTENT = "Author"; // content
    public static final Uri CONTENT_URI = BookProvider.CONTENT_URI(BookProvider.AUTHORITY, TABLE_NAME);
    public static String CONTENT_PATH = BookProvider.CONTENT_PATH(CONTENT_URI);
    public static String CONTENT_PATH_ITEM = BookProvider.CONTENT_PATH(CONTENT_URI("#"));
    public static final String contentType = BookProvider.contentType(CONTENT);
    public static final String contentItemType = BookProvider.contentItemType(CONTENT);

    // CONTENT_URI(id)
    public static Uri CONTENT_URI (String id) {
        return BookProvider.withExtendedPath(CONTENT_URI, id);
    }

    public static long getId(Cursor cursor) {
        return cursor.getLong(cursor.getColumnIndexOrThrow(ID));
    }

    public static long getId(Uri uri) { return Long.parseLong(uri.getLastPathSegment()); }


    public static void putId(ContentValues values, long id) { values.put(ID, id); }

    public static String getName(Cursor cursor) {
        return cursor.getString(cursor.getColumnIndexOrThrow(NAME));
    }

    public static void putName(ContentValues values, String name) { values.put(NAME, name); }

    public static long getBookForeignKey(Cursor cursor) {
        return cursor.getLong(cursor.getColumnIndexOrThrow(BOOK_FOREIGN_KEY));
    }

    public static void putBookForeignKey(ContentValues values, long book_fk) {
        values.put(BOOK_FOREIGN_KEY, book_fk);
    }
}
```

Figure 4.1

```java
public class DatabaseHelper extends SQLiteOpenHelper {

    private static final String BOOK_TABLE_CREATE =
            "CREATE TABLE " + BookContract.TABLE_NAME + " (" + BookContract.ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " + BookCon
                    + " TEXT NOT NULL, " + BookContract.ISBN + " TEXT NOT NULL, " + BookContract.PRICE + " TEXT NOT NULL " + ");"
    private static final String AUTHOR_TABLE_CREATE =
            "CREATE TABLE " + AuthorContract.TABLE_NAME + " (" + AuthorContract.ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " + Aut
                    + " TEXT NOT NULL, " + AuthorContract.BOOK_FOREIGN_KEY + " INTEGER, " + "FOREIGN KEY (" + AuthorContract.BOOK
                    + BookContract.ID + ") ON DELETE CASCADE" + ");";
    private static final String CREATE_INDEX =
            "CREATE INDEX AuthorsBookIndex ON " + AuthorContract.TABLE_NAME + "(" + AuthorContract.BOOK_FOREIGN_KEY + ");";

    private static final String DATABASE_NAME = "cart.db";
    private static final int DATABASE_VERSION = 1;

    public DatabaseHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
    }

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

    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
        sqLiteDatabase.execSQL("PRAGMA foreign_keys = ON");
        sqLiteDatabase.execSQL(BOOK_TABLE_CREATE);
        sqLiteDatabase.execSQL(AUTHOR_TABLE_CREATE);
        sqLiteDatabase.execSQL(CREATE_INDEX);
    }

    @Override
    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i2) {
        // Log the version upgrade
        Log.w("DatabaseHelper", "Upgrading from version" + i + " to " + i2);
        sqLiteDatabase.execSQL("DROP TABLE IF EXISTS " + BookContract.TABLE_NAME);
        sqLiteDatabase.execSQL("DROP TABLE IF EXISTS" + AuthorContract.TABLE_NAME);
        onCreate(sqLiteDatabase);
    }
}
```

Figure 4.2

5. Define an interface called IContinue and define a method called kontinue in it as Figure 5.1. Define a class called AsyncContentResolver which extends AsyncQueryHandler, define four methods to query, insert, delete and update the database asynchronously and override the onQueryComplete and onInsertComplete method to implement the kontinue in the main thread as Figure 5.2.

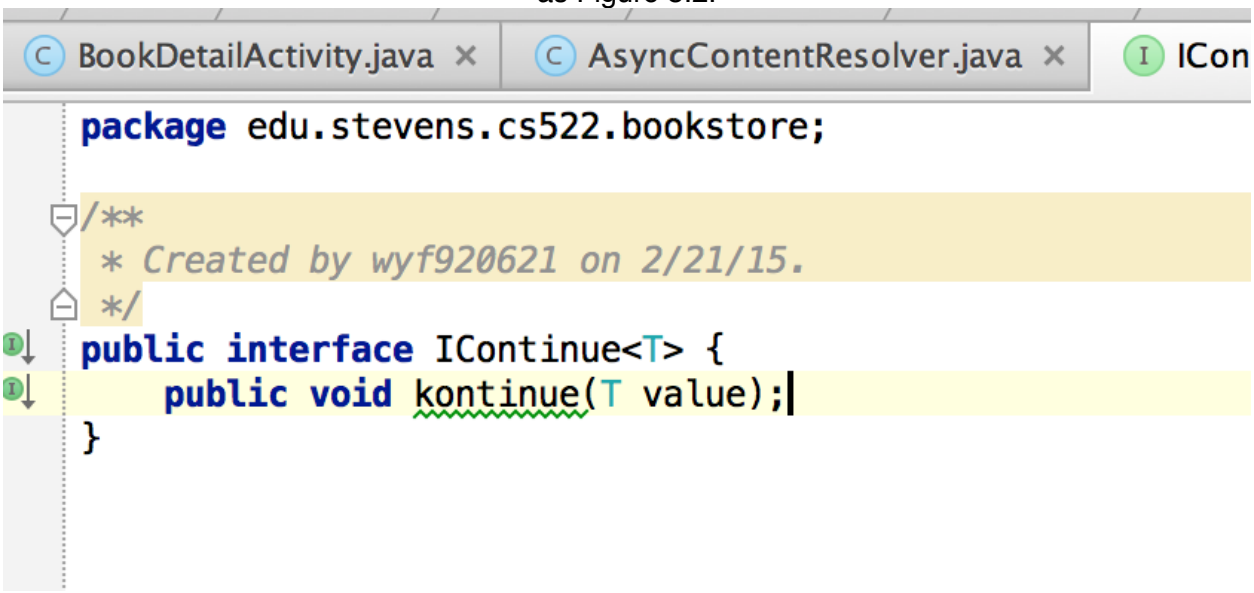


Figure 5.1

```
package edu.stevens.cs522.bookstore.providers;

import ...

/**
 * Created by wyf920621 on 2/21/15.
 */
// AsyncContentResolver
public class AsyncContentResolver extends AsyncQueryHandler {
    public AsyncContentResolver(ContentResolver cr) { super(cr); }
    public void insertAsync(Uri uri, ContentValues values, IContinue<Uri> callback) {
        this.startInsert(0, callback, uri, values);
    }

    public void deleteAsync(Uri uri, String selection, String[] selectionArgs) {
        this.startDelete(0, null, uri, selection, selectionArgs);
    }

    public void updateAsync(Uri uri, ContentValues values, String selection, String[] selectionArgs) {
        startUpdate(0, null, uri, values, selection, selectionArgs);
    }

    public void queryAsync(Uri uri, String[] projection, String selection, String[] selectionArgs, String orderBy, IContinue<Curs
        this.startQuery(0, callback, uri, projection, selection, selectionArgs, orderBy);
    }

    @Override
    protected void onQueryComplete(int token, Object cookie, Cursor cursor) {
        if (cookie != null) {
            /unchecked/
            IContinue<Cursor> callback = (IContinue<Cursor>)cookie;
            callback.kontinue(cursor);
        }
    }

    @Override
    protected void onInsertComplete(int token, Object cookie, Uri uri) {
        if (cookie != null) {
            /unchecked/
            IContinue<Uri> callback = (IContinue<Uri>)cookie;
            callback.kontinue(uri);
        }
    }
}
```

Figure 5.2

6. Modify the BookStoreActivity which implements the interface LoaderManager.LoaderCallbacks<Cursor>. In the activity, define a flag called MY_LOADER_ID for the loader manager. Replace the SimpleCursorAdapter with BookAdapter and define a fillData method to set the adapter onto the list view as Figure 6.1. Override the onCreateLoader, onLoadFinished and onLoadReset method to get the CursorLoader and use it to refresh the BookAdapter as Figure 6.2. Define an asyncContentProvider to implement query and delete operation in after delete button is clicked or checkout activity is finished as Figure 6.3 and Figure 6.4.

```
public class BookStoreActivity extends ListActivity implements LoaderManager.LoaderCallbacks<Cursor>  {
    // LOADER_ID
    private static final int MY_LOADER_ID = 1;

    // Use this when logging errors and warnings.
    /unused/
    private static final String TAG = BookStoreActivity.class.getCanonicalName();

    // These are request codes for subactivity request calls
    static final private int ADD_REQUEST = 1;

    /unused/
    static final private int CHECKOUT_REQUEST = ADD_REQUEST + 1;

    // There is a reason this must be an ArrayList instead of a List.
    /unused/

    private ActionMode mActionMode = null;
    private BookAdapter adapter = null;
    private AsyncContentResolver asyncContentResolver;
    ListView listView;

    public static final String BOOK_STORE_KEY = "BOOK_STORE_KEY";

    private void fillData(Cursor c) {
        String[] to = new String[] { BookContract.TITLE, BookContract.AUTHORS};
        int[] from = new int[] {R.id.cart_row_title, R.id.cart_row_author};
        adapter = new BookAdapter(this, null);
        listView.setAdapter(this.adapter);
    }
```

Figure 6.1

```
    @Override
    public Loader<Cursor> onCreateLoader(int id, Bundle bundle) {
        switch (id) {
            case MY_LOADER_ID:
                String[] projection = {BookContract.ID, BookContract.TITLE, BookContract.AUTHORS};
                return new CursorLoader(this, BookContract.CONTENT_URI, projection, null, null, null);
            default:
                return null;
        }
    }

    @Override
    public void onLoadFinished(Loader<Cursor> cursorLoader, Cursor cursor) {
        this.adapter.swapCursor(cursor);
        cursor.setNotificationUri(getContentResolver(), BookContract.CONTENT_URI);
    }

    @Override
    public void onLoaderReset(Loader<Cursor> cursorLoader) { this.adapter.swapCursor(null); }
```

Figure 6.2

```
    // CHECKOUT: empty the shopping cart.
    if (requestCode == ADD_REQUEST) {
        if (resultCode == RESULT_OK) {
            Log.v("ADD_REQUEST: ", "RESULT_OK");
            getContentResolver().notifyChange(BookContract.CONTENT_URI, null);
        }
    }
    else if (requestCode == CHECKOUT_REQUEST) {
        if (resultCode == RESULT_OK) {
            // delete all
            Log.v("CHECKOUT_REQUEST", "RESULT_OK");
            asyncContentResolver.deleteAsync(BookContract.CONTENT_URI, null, null);
            getContentResolver().notifyChange(BookContract.CONTENT_URI, null);
        }
    }
    //getLoaderManager().restartLoader(MY_LOADER_ID, null, BookStoreActivity.this);
}
```

Figure 6.3

```
@Override
public boolean onActionItemClicked(ActionMode actionMode, final MenuItem menuItem) {
    final long rowId = (Long)mActionMode.getTag();
    String[] projection = new String[] {BookContract.ID, BookContract.TITLE, BookContract.PRICE, BookContract.ISBN, BookC
    String selection = BookContract.ID + "=?";
    String[] selectionArgs = new String[] {String.valueOf(rowId)};
    asyncContentResolver.queryAsync(BookContract.CONTENT_URI(String.valueOf(rowId)), projection, selection, selectionArgs
            if(value.moveToFirst()) {
                Book book = new Book(value);
                switch (menuItem.getItemId()) {
                    case R.id.detail:
                        Intent intent = new Intent(BookStoreActivity.this, BookDetailActivity.class);
                        intent.putExtra(BOOK_STORE_KEY, book);
                        startActivity(intent);
                        break;
                    case R.id.delete:
                        String selection = BookContract.ID + "=?";
                        String[] selectionArgs = new String[]{String.valueOf(rowId)};
                        asyncContentResolver.deleteAsync(BookContract.CONTENT_URI(String.valueOf(rowId)), selection, sele
                        getContentResolver().notifyChange(BookContract.CONTENT_URI, null);
                        break;
                }
            }
            value.close();
    });
    return true;
}
```

Figure 6.4

7. In the addBookActivity, use asyncContentResolver to insert the Book and Author information into the database as Figure 7.1

```java
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    super.onOptionsItemSelected(item);
    // TODO

    // SEARCH: return the book details to the BookStore activity

    // CANCEL: cancel the search request
    switch (item.getItemId()) {
        case R.id.search:
            final Book newBook = searchBook();
            if (newBook == null) {
                searchTitle.setText("");
                searchAuthor.setText("");
                searchIsbn.setText("");
                searchPrice.setText("");
                return false;
            }
            final ContentValues values = new ContentValues();
            newBook.writeToProvider(values);
            asyncContentResolver.insertAsync(BookContract.CONTENT_URI, values, (value) -> {
                    newBook.id = BookContract.getId(value);
                    for (final Author author : newBook.authors) {
                        values.clear();
                        author.writeToProvider(values, newBook.id);
                        asyncContentResolver.insertAsync(AuthorContract.CONTENT_URI, values, (value) -> {
                                author.id = AuthorContract.getId(value);
                        });
                    }
            });
            setResult(RESULT_OK, intent);
            finish();
            break;
        case R.id.search_cancel:
            setResult(RESULT_CANCELED);
            finish();
            break;
    }
    return true;
}
```

Figure 7.1

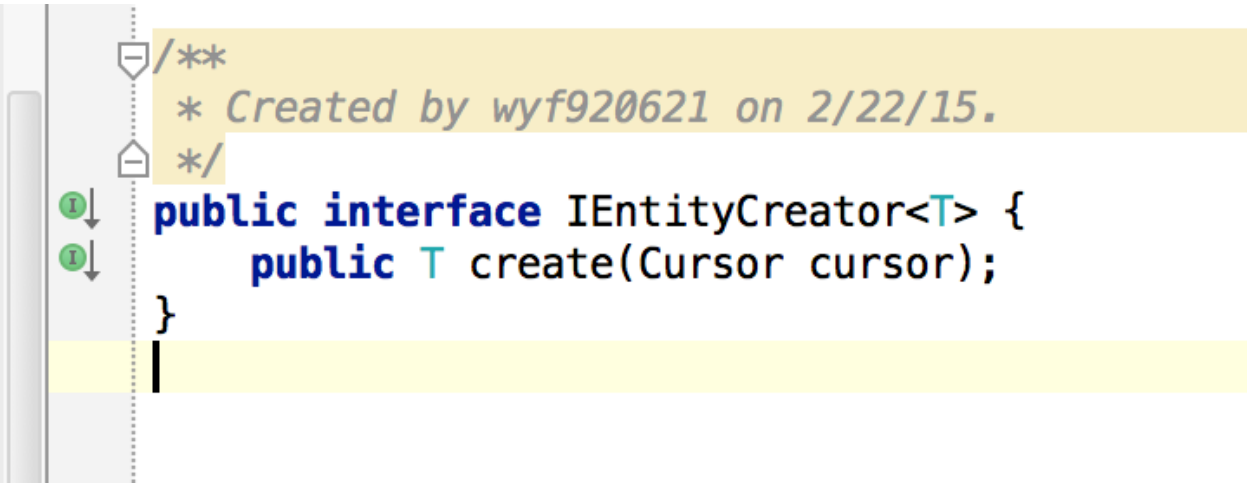
8. Run the BookStoreWithContentProvider as the video shows.

Part 2. BookStoreWithEntityManager

1. Follow the instruction from Part 1 until step 5. Next, define several interfaces including IContinue, IEntityCreator, IQueryListener, ISimpleQueryListener as Figure 1.1, Figure 1.2, Figure 1.3, Figure 1.4

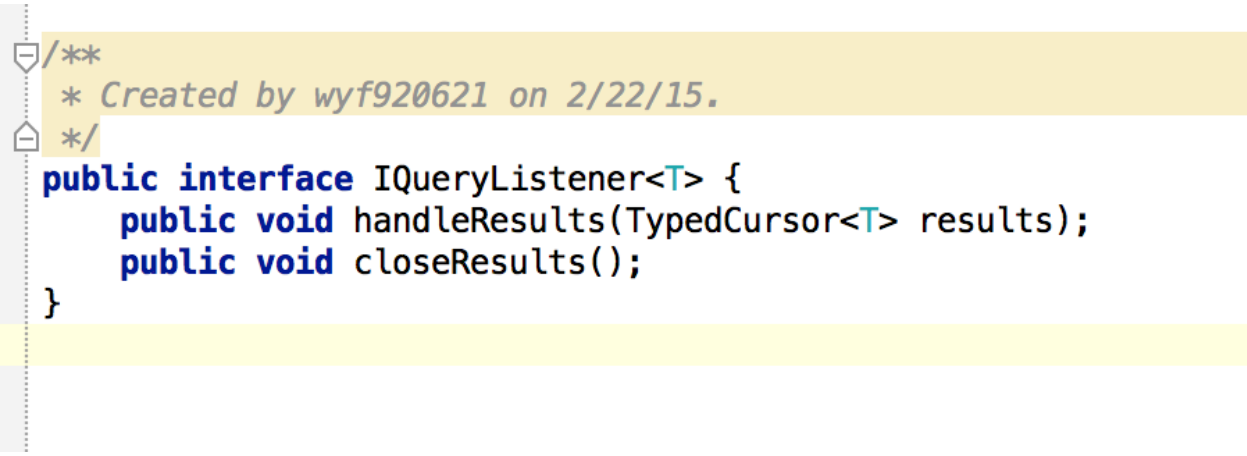
```java
 */
public interface IContinue<T> {
    public void kontinue(T value);
}
```

Figure 1.1



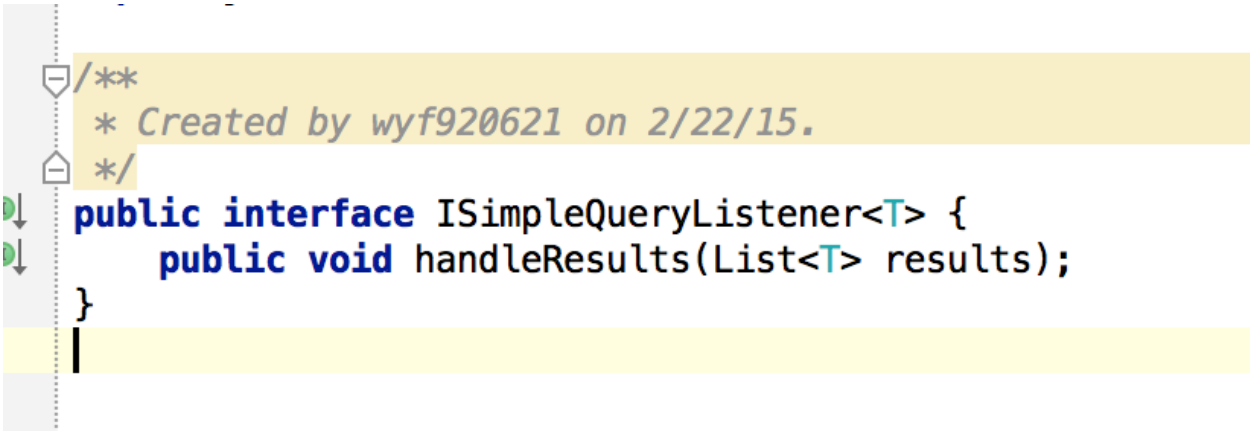
```
/**
 * Created by wyf920621 on 2/22/15.
 */
public interface IEntityCreator<T> {
    public T create(Cursor cursor);
}
```

Figure 1.2



```
/**
 * Created by wyf920621 on 2/22/15.
 */
public interface IQueryListener<T> {
    public void handleResults(TypedCursor<T> results);
    public void closeResults();
}
```

Figure 1.3



```
/**
 * Created by wyf920621 on 2/22/15.
 */
public interface ISimpleQueryListener<T> {
    public void handleResults(List<T> results);
}
```

Figure 1.4

2. Create a TypedCursor class as the assignment gives.

```
*/
public class TypedCursor<T> {

    private Cursor cursor;

    private IEntityCreator<T> creator;

    public int getCount() { return cursor.getCount(); }

    public boolean moveToFirst() { return cursor.moveToFirst(); }

    public boolean moveToNext() { return cursor.moveToNext(); }

    public T getEntity() { return creator.create(cursor); }

    public Cursor getCursor() { return cursor; }

    public T create(Cursor cursor) { return creator.create(cursor); }

    public void close() { cursor.close(); }

    public void registerContentObserver(ContentObserver observer) {
        cursor.registerContentObserver(observer);
    }

    public void unregisterContentObserver(ContentObserver observer) {
        cursor.unregisterContentObserver(observer);
    }

    public TypedCursor(Cursor cursor, IEntityCreator<T> creator) {
        this.cursor = cursor;
        this.creator = creator;
    }

}
```

Figure 2.1

3. Define a SimpleQueryBuilder class which implements IContinue interface as Figure 3.1. The SimpleQueryBuilder is used to query a book's details.

```
*/
public class SimpleQueryBuilder<T> implements IContinue<Cursor> {
    private IEntityCreator<T> helper;
    private ISimpleQueryListener<T> listener;

    private SimpleQueryBuilder(
            IEntityCreator<T> helper,
            ISimpleQueryListener<T> listener) {
        this.helper = helper;
        this.listener = listener;
    }

    public static <T> void executeQuery(Activity context, Uri uri, IEntityCreator<T> helper, ISimpleQueryListener<T> listener) {
        SimpleQueryBuilder<T> builder = new SimpleQueryBuilder<T>(helper, listener);
        AsyncContentResolver resolver = new AsyncContentResolver(context.getContentResolver());
        resolver.queryAsync(uri, null, null, null, null, builder);
    }

    public static <T> void executeQuery(Activity context, Uri uri, String[] projection, String selection, String[] selectionArgs, I
        SimpleQueryBuilder<T> builder = new SimpleQueryBuilder<T>(helper, listener);
        AsyncContentResolver resolver = new AsyncContentResolver(context.getContentResolver());
        resolver.queryAsync(uri, projection, selection, selectionArgs, null, builder);
    }

    @Override
    public void kontinue(Cursor value) {
        List<T> instances = new ArrayList<T>();
        if (value.moveToFirst()) {
            do {
                T instance = helper.create(value);
                instances.add(instance);
            } while (value.moveToNext());
        }
        value.close();
        listener.handleResults(instances);
    }
}
```

Figure 3.1

4. Define a QueryBuilder class which implements LoaderManager.LoaderCallbacks<Cursor> as Figure 4.1, Figure 4.2, Figure 4.3. The QueryBuilder class is used to query all books and manage the list view by the TypedCursor loader.

```
public class QueryBuilder<T> implements LoaderManager.LoaderCallbacks<Cursor> {
    public static final String PROJECTION = "bookstore.books.projection";
    public static final String SELECTION = "bookstore.books.selection";
    public static final String SELECTIONARGS = "bookstore.books.selectionArgs";
    private String tag;
    private Context context;
    private Uri uri;
    private int loaderID;
    private IEntityCreator<T> creator;
    private IQueryListener<T> listener;
    private QueryBuilder(String tag,
                         Context context,
                         Uri uri,
                         int loaderID,
                         IEntityCreator<T> creator,
                         IQueryListener<T> listener) {
        this.tag = tag;
        this.context = context;
        this.uri = uri;
        this.loaderID = loaderID;
        this.creator = creator;
        this.listener = listener;
    }

    @Override
    public Loader<Cursor> onCreateLoader(int id, Bundle bundle) {
        if (bundle == null) {
            if (id == loaderID) {
                return new CursorLoader(context, uri, null, null, null, null);
            }
        }
        else {
            String[] projection = bundle.getStringArray(PROJECTION);
            String selection = bundle.getString(SELECTION);
            String[] selectionArgs = bundle.getStringArray(SELECTIONARGS);
            if (id == loaderID) {
                return new CursorLoader(context, uri, projection, selection, selectionArgs, null);
            }
        }
        return null;
    }
```

Figure 4.1

```
    @Override
    public void onLoadFinished(Loader<Cursor> cursorLoader, Cursor cursor) {
        if (cursorLoader.getId() == loaderID) {
            listener.handleResults(new TypedCursor<T>(cursor, creator));
        } else {
            throw new IllegalStateException("Unexpected loader callback");
        }
    }

    @Override
    public void onLoaderReset(Loader<Cursor> cursorLoader) {
        if (cursorLoader.getId() == loaderID) {
            listener.closeResults();
        } else {
            throw new IllegalStateException("Unexpected loader callback");
        }
    }

    public static <T> void executeQuery(String tag, Activity context, Uri uri, int loaderID, IEntityCreator<T> creator, IQueryListe
        QueryBuilder<T> builder = new QueryBuilder<T>(tag, context, uri, loaderID, creator, listener);
        LoaderManager lm = context.getLoaderManager();
        if(lm.getLoader(loaderID) == null) {
            lm.initLoader(loaderID, null, builder);
        }

    }

    public static <T> void executeQuery(String tag, Activity context, Uri uri, int loaderID, String[] projection, String selection,
        QueryBuilder<T> builder = new QueryBuilder<T>(tag, context, uri, loaderID, creator, listener);
        LoaderManager lm = context.getLoaderManager();
        Bundle bundle = new Bundle();
        bundle.putStringArray(PROJECTION, projection);
        bundle.putString(SELECTION, selection);
        bundle.putStringArray(SELECTIONARGS, selectionArgs);
        if(lm.getLoader(loaderID) == null) {
            lm.initLoader(loaderID, null, builder);
        }
    }
```

Figure 4.2

```
    public static <T> void reexecuteQuery(String tag, Activity context, Uri uri, int loaderID, IEntityCreator<T> creator, IQueryLis
        QueryBuilder<T> builder = new QueryBuilder<T>(tag, context, uri, loaderID, creator, listener);
        LoaderManager lm = context.getLoaderManager();
        if(lm.getLoader(loaderID) != null) {
            lm.restartLoader(loaderID, null, builder);
        }
    }

    public static <T> void reexecuteQuery(String tag, Activity context, Uri uri, int loaderID, String[] projection, String selectio
        QueryBuilder<T> builder = new QueryBuilder<T>(tag, context, uri, loaderID, creator, listener);
        LoaderManager lm = context.getLoaderManager();
        Bundle bundle = new Bundle();
        bundle.putStringArray(PROJECTION, projection);
        bundle.putString(SELECTION, selection);
        bundle.putStringArray(SELECTIONARGS, selectionArgs);
        if(lm.getLoader(loaderID) != null) {
            lm.restartLoader(loaderID, bundle, builder);
        }
    }
}
```

Figure 4.3

5. Define a Manager<T> generic class which has methods to execute SimpleQuery or normal Query asynchronously as Figure 5.1, Figure 5.2.

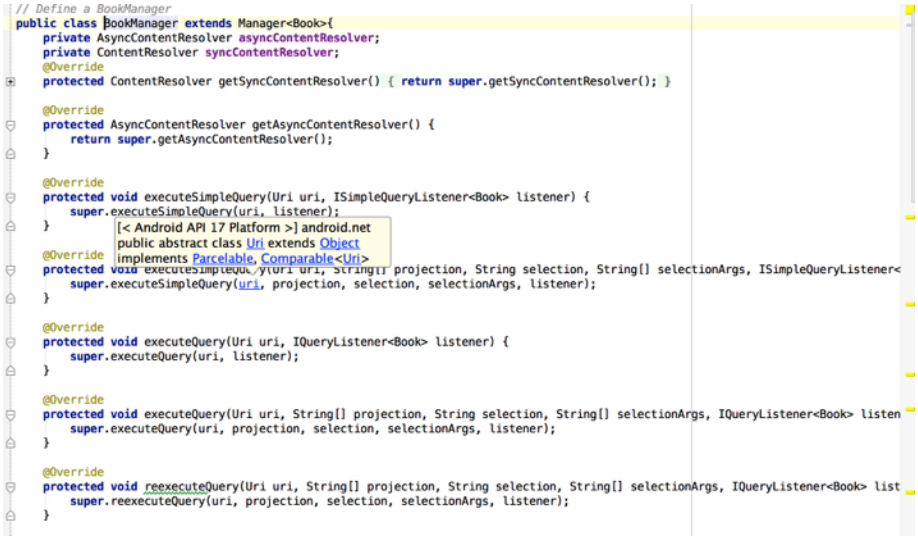
```java
/**
 * Created by wyf920621 on 2/22/15.
 */
public abstract class Manager<T> {
    private final Context context;
    private final IEntityCreator<T> creator;
    private final int loaderID;
    private final String tag;
    private ContentResolver syncContentResolver;
    private AsyncContentResolver asyncContentResolver;

    protected Manager(Context context, IEntityCreator<T> creator, int loaderID) {
        this.context = context;
        this.creator = creator;
        this.loaderID = loaderID;
        this.tag = this.getClass().getCanonicalName();
        Activity activity = (Activity)context;
    }
    protected ContentResolver getSyncContentResolver() {
        if (syncContentResolver == null)
            syncContentResolver = context.getContentResolver();
        return syncContentResolver;
    }

    protected AsyncContentResolver getAsyncContentResolver() {
        if (asyncContentResolver == null)
            asyncContentResolver = new AsyncContentResolver(context.getContentResolver());
        return asyncContentResolver;
    }
    protected void executeSimpleQuery(Uri uri, ISimpleQueryListener<T> listener) {
        SimpleQueryBuilder.executeQuery((Activity)context, uri, creator, listener);
    }
```

Figure 5.1

```java
    protected void executeSimpleQuery(Uri uri, String[] projection, String selection, String[] selectionArgs, ISimpleQueryListener<
        SimpleQueryBuilder.executeQuery((Activity)context, uri, projection, selection, selectionArgs, creator, listener);
    }

    protected void executeQuery(Uri uri, IQueryListener<T> listener) {
        QueryBuilder.executeQuery(tag, (Activity)context, uri, loaderID, creator, listener);
    }

    protected void executeQuery(Uri uri, String[] projection, String selection, String[] selectionArgs, IQueryListener<T> listener)
        QueryBuilder.executeQuery(tag, (Activity)context, uri, loaderID, projection, selection, selectionArgs, creator, listener);
    }

    protected void reexcuteQuery(Uri uri, IQueryListener<T> listener) {
        QueryBuilder.reexecuteQuery(tag, (Activity)context, uri, loaderID, creator, listener);
    }

    protected void reexecuteQuery(Uri uri, String[] projection, String selection, String[] selectionArgs, IQueryListener<T> listene
        QueryBuilder.reexecuteQuery(tag, (Activity)context, uri, loaderID, projection, selection, selectionArgs, creator, listener)
    }
}
```

Figure 5.2

6. Define a BookManager class which extends Manger<T> class. The class implements each query or simple query as Figure 6.1. Furthermore, the class also implements synchronous and asynchronous query, insert, delete and update operations as Figure 6.2, Figure 6.3.



```
// Define a BookManager
public class BookManager extends Manager<Book>{
    private AsyncContentResolver asyncContentResolver;
    private ContentResolver syncContentResolver;
    @Override
    protected ContentResolver getSyncContentResolver() { return super.getSyncContentResolver(); }

    @Override
    protected AsyncContentResolver getAsyncContentResolver() {
        return super.getAsyncContentResolver();
    }

    @Override
    protected void executeSimpleQuery(Uri uri, ISimpleQueryListener<Book> listener) {
        super.executeSimpleQuery(uri, listener);
    }
                [< Android API 17 Platform >] android.net
                public abstract class Uri extends Object
                implements Parcelable, Comparable<Uri>
    @Override
    protected void executeSimpleQuery(Uri uri, String[] projection, String selection, String[] selectionArgs, ISimpleQueryListener<
        super.executeSimpleQuery(uri, projection, selection, selectionArgs, listener);
    }

    @Override
    protected void executeQuery(Uri uri, IQueryListener<Book> listener) {
        super.executeQuery(uri, listener);
    }

    @Override
    protected void executeQuery(Uri uri, String[] projection, String selection, String[] selectionArgs, IQueryListener<Book> listen
        super.executeQuery(uri, projection, selection, selectionArgs, listener);
    }

    @Override
    protected void reexecuteQuery(Uri uri, String[] projection, String selection, String[] selectionArgs, IQueryListener<Book> list
        super.reexecuteQuery(uri, projection, selection, selectionArgs, listener);
    }
```

Figure 6.1

```
    private final Uri CONTENT_URI = BookContract.CONTENT_URI;

    public BookManager(Context context, IEntityCreator<Book> creator, int loaderID) {
        super(context, creator, loaderID);
        this.asyncContentResolver = getAsyncContentResolver();
        this.syncContentResolver = getSyncContentResolver();
    }
    // Sync insert
    public Uri insert(Book book) {
        ContentValues values = new ContentValues();
        book.writeToProvider(values);
        Uri uri = syncContentResolver.insert(BookContract.CONTENT_URI, values);
        book.id = BookContract.getId(uri);
        for (Author author : book.authors) {
            values.clear();
            author.writeToProvider(values, book.id);
            Uri authorUri = syncContentResolver.insert(AuthorContract.CONTENT_URI, values);
            author.id = AuthorContract.getId(authorUri);
        }
        return uri;
    }
    // Async insert
    public void persistAsync(final Book book) {
        final ContentValues values = new ContentValues();
        book.writeToProvider(values);
        asyncContentResolver.insertAsync(CONTENT_URI, values, (value) -> {
                book.id = BookContract.getId(value);
                ArrayList<ContentValues> valueses = new ArrayList<~>(book.authors.length);
                for (int i = 0; i < book.authors.length; i++) {
                    valueses.add(new ContentValues());
                }
                for (int i = 0; i < book.authors.length; i++) {
                    Author author = book.authors[i];
                    Log.v("Author to be inserted: ", author.toString());
                    author.writeToProvider(valueses.get(i), book.id);
                    asyncContentResolver.insertAsync(AuthorContract.CONTENT_URI, valueses.get(i), (value) -> {
                            Log.v("Author id: ", String.valueOf(AuthorContract.getId(value)));
                    });
                }

        });
```

Figure 6.2

```
// Sync Query
public Book search(long rowId) {
    Cursor cursor = syncContentResolver.query(BookContract.CONTENT_URI(String.valueOf(rowId)),
            new String[] {BookContract.ID, BookContract.TITLE, BookContract.PRICE, BookContract.ISBN, BookContract.AUTHORS},
            BookContract.ID + "=?",
            new String[] {String.valueOf(rowId)},
            null);
    Book book = new Book(BookContract.getTitle(cursor), BookContract.getAuthors(cursor), BookContract.getIsbn(cursor), BookCont
    book.id = rowId;
    return book;
}

// Async Query
public void queryAsync(Uri uri, IQueryListener<Book> listener) { executeQuery(uri, listener); }

public void queryAsync(long rowId, IContinue<Cursor> iContinue) { // rowId: book rowID
    asyncContentResolver.queryAsync(BookContract.CONTENT_URI(String.valueOf(rowId)), new String[] {BookContract.ID, BookContrac
}

// Async Requery
public void requeryAsync(Uri uri, IQueryListener<Book> listener) {
    reexcuteQuery(uri, listener);
}

// Sync delete
public int delete(Book book) {
    long id = book.id;
    return syncContentResolver.delete(BookContract.CONTENT_URI(String.valueOf(id)), BookContract.ID + "=?", new String[] {Strin
}

// Async delete
public void deleteAsync(final Book book) {
    asyncContentResolver.deleteAsync(BookContract.CONTENT_URI(String.valueOf(book.id)), BookContract.ID + "=?", new String[] {S
}
// Sync deleteAll
public int deleteAll() {
    return syncContentResolver.delete(BookContract.CONTENT_URI, null, null);
}

// Async deleteAll
public void deleteAllAsync() { asyncContentResolver.deleteAsync(CONTENT_URI, null, null); }
```

Figure 6.3

7. In the BookStoreActivity, define a BookManager and use the bookManager to handle the list view and the BookAdapter which is also defined in the activity as Figure 7.1. When a book is added or deleted, or a checkout is finished, use the bookManager to requery the database and refresh the list view as Figure 7.2 and Figure 7.3.

```
private void fillData(Cursor c) {
    String[] to = new String[] { BookContract.TITLE, BookContract.AUTHORS};
    int[] from = new int[] {R.id.cart_row_title, R.id.cart_row_author};
    adapter = new BookAdapter(this, null);
    listView.setAdapter(this.adapter);
}

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.cart);

    listView = (ListView)findViewById(android.R.id.list);
    fillData(null); // take no cursor
    listView.setChoiceMode(ListView.CHOICE_MODE_SINGLE);
    listView.setOnItemLongClickListener((adapterView, view, position, id) -> {
            view.setSelected(true);
            Long rowId = adapterView.getAdapter().getItemId(position);
            if (mActionMode != null) {
                return false;
            }
            mActionMode = BookStoreActivity.this.startActionMode(mActionModeCallback);
            mActionMode.setTag(rowId);
            return true;
    });
    bookManager = new BookManager(this,
            (cursor) -> {
                    Book book = new Book(cursor);
                    book.id = BookContract.getId(cursor);
                    return book;
            },
            MY_LOADER_ID
    );
    bookManager.queryAsync(BookContract.CONTENT_URI, new IQueryListener<Book>() {
        @Override
        public void handleResults(TypedCursor<Book> results) {
            adapter.swapCursor(results.getCursor());
            results.getCursor().setNotificationUri(getContentResolver(), BookContract.CONTENT_URI);
        }

        @Override
        public void closeResults() { adapter.swapCursor(null); }
    });
```

Figure 7.1

```
@Override
protected void onActivityResult(int requestCode, final int resultCode,
        Intent intent) {
    super.onActivityResult(requestCode, resultCode, intent);
    // Use SEARCH_REQUEST and CHECKOUT_REQUEST codes to distinguish the cases.

    // SEARCH: add the book that is returned to the shopping cart.

    // CHECKOUT: empty the shopping cart.
    if (requestCode == ADD_REQUEST) {
        if (resultCode == RESULT_OK) {
            Log.v("ADD_REQUEST", "RESULT OK");
            getContentResolver().notifyChange(BookContract.CONTENT_URI, null);
        }
    }
    else if (requestCode == CHECKOUT_REQUEST) {
        if (resultCode == RESULT_OK) {
            bookManager.deleteAllAsync();
            getContentResolver().notifyChange(BookContract.CONTENT_URI, null);
        }
    }
}
```

Figure 7.2

```
@Override
protected void onActivityResult(int requestCode, final int resultCode,
        Intent intent) {
    super.onActivityResult(requestCode, resultCode, intent);
    // Use SEARCH_REQUEST and CHECKOUT_REQUEST codes to distinguish the cases.

    // SEARCH: add the book that is returned to the shopping cart.

    // CHECKOUT: empty the shopping cart.
    if (requestCode == ADD_REQUEST) {
        if (resultCode == RESULT_OK) {
            Log.v("ADD_REQUEST", "RESULT OK");
            getContentResolver().notifyChange(BookContract.CONTENT_URI, null);
        }
    }
    else if (requestCode == CHECKOUT_REQUEST) {
        if (resultCode == RESULT_OK) {
            bookManager.deleteAllAsync();
            getContentResolver().notifyChange(BookContract.CONTENT_URI, null);
        }
    }
}
```

Figure 7.3

8. In the BookDetailActivity, use bookManager to query the detail information of the book as Figure 8.1.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_book_detail);
    bookManager = new BookManager(this, (cursor) -> {
            Book book = new Book(cursor);
            book.id = BookContract.getId(cursor);
            return book;
    }, Book_Detail_ID);

    id = (TextView)findViewById(R.id.detail_id);
    title = (TextView)findViewById(R.id.detail_title);
    author = (TextView)findViewById(R.id.detail_author);
    isbn = (TextView)findViewById(R.id.detail_isbn);
    price = (TextView)findViewById(R.id.detail_price);
    Intent intent = getIntent();
    long book_id = intent.getLongExtra(BookStoreActivity.BOOK_STORE_KEY, 0);
    String[] projection = new String[] {BookContract.ID, BookContract.TITLE, BookContract.AUTHORS, BookContract.ISBN, BookContr
    String selection = BookContract.ID + "=?";
    String[] selectionArgs = new String[] {String.valueOf(book_id)};
    bookManager.simpleQueryAsync(BookContract.CONTENT_URI(String.valueOf(book_id)), projection, selection, selectionArgs, (resu
            for (Book book : results) {
                id.setText(String.valueOf(book.id));
                title.setText(book.title);
                isbn.setText(book.isbn);
                price.setText(book.price);
                Author[] authors = book.authors;
                String author_name = "";
                for (int i = 0; i < authors.length; i++) {
                    Log.v("Author Readed: ", authors[i].toString());
                    author_name = author_name + authors[i].toString();
                    if (i != authors.length - 1){
                        author_name += "; ";
                    }
                }
                author.setText(author_name);
            }
    });
}
```

Figure 8.1

9. In the AddBookActivity, use the bookManager to persist the book asynchronously as Figure 9.1.

```java
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    super.onOptionsItemSelected(item);

    // SEARCH: return the book details to the BookStore activity

    // CANCEL: cancel the search request
    switch (item.getItemId()) {
        case R.id.search:
            Book newBook = searchBook();
            if (newBook == null) {
                searchTitle.setText("");
                searchAuthor.setText("");
                searchIsbn.setText("");
                searchPrice.setText("");
                return false;
            }
            bookManager.persistAsync(newBook);
            setResult(RESULT_OK, intent);
            finish();
            break;
        case R.id.search_cancel:
            setResult(RESULT_CANCELED);
            finish();
            break;
    }
    return true;
}
```

Figure 9.1

10. Run the App as the video shows.

Part 3. Persistent Chat App

1. In the Chat Client App, create a PreferenceActivity, accept a name from user and set the name as clientName and store it into the SharedPreference as Figure 1.1 and Figure 1.2.

```
public class PreferenceActivity extends Activity {
    public static final String USER_KEY = "myPrefUsername";

    public static final String MY_PREFS = "myPreferences";
    public static final String USERNAME = "username";

    EditText userText;
    Button buttonOk;
    Button buttonCancel;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_preference);
        userText = (EditText)findViewById(R.id.text_username);
        buttonOk = (Button)findViewById(R.id.button_ok);
        buttonCancel = (Button)findViewById(R.id.button_cancel);
        userText.setText(loadPreferences());
        buttonOk.setOnClickListener((view) -> {
                String username = userText.getText().toString();
                savePreferences(username);
                Intent intent = new Intent(PreferenceActivity.this, ChatClient.class);
                intent.putExtra(USER_KEY, username);
                setResult(RESULT_OK, intent);
                finish();
        });
        buttonCancel.setOnClickListener((view) -> {
                setResult(RESULT_CANCELED);
                finish();
        });
    }
```

Figure 1.1

```
    protected void savePreferences(String chatUsername) {
        SharedPreferences sharedPreferences = getSharedPreferences(MY_PREFS, Activity.MODE_PRIVATE);
        SharedPreferences.Editor editor = sharedPreferences.edit();

        // store new object
        editor.putString(USERNAME, chatUsername);

        // commit
        editor.apply();
    }

    public String loadPreferences() {
        SharedPreferences sharedPreferences = getSharedPreferences(MY_PREFS, Activity.MODE_PRIVATE);
        return sharedPreferences.getString(USERNAME, "client");
    }
}
```

Figure 1.2

2. In the Chat Server App, in order to use the entity manager, define the IContinue, IEntityCreator, IQueryListener, ISimpleQueryListener interfaces. Also define the Manager<T>, AsyncContentResolver, ChatDbProvider, QueryBuilder and SimpleQueryBuilder as Part 1 and Part 2 says.

3. Figure 3.1, Figure 3.2, Figure 3.3, Figure 3.4, Figure 3.5 show the structure of the ChatDbProvider.

```java
public class ChatDbProvider extends ContentProvider {
    public static final String AUTHORITY = "edu.stevens.cs522.chat.oneway.server";

    public static String CONTENT_PATH(Uri uri) {
        return uri.getPath().substring(1); // Trim leading "/"
    }


    public static Uri CONTENT_URI(String authority, String path) {
        return new Uri.Builder().scheme("content").authority(authority).path(path).build();
    }

    public static Uri withExtendedPath(Uri uri, String... path){
        Uri.Builder builder = uri.buildUpon();
        for (String p : path)
            builder.appendPath(p);
        return builder.build();
    }

    public static String contentType(String content) {
        return "vnd.android.cursor/vnd." + AUTHORITY + "." + content + "s";
    }

    public static String contentItemType(String content) {
        return "vnd.android.cursor.item/vnd." + AUTHORITY + "." + content;
    }

    private DatabaseHelper databaseHelper;
    private static final String DATABASE_NAME = "chat.db";
    private static final int DATABASE_VERSION = 1;
```

Figure 3.1

```java
    private static final int MESSAGE_ALL_ROWS = 1;
    private static final int MESSAGE_SINGLE_ROW = 2;
    private static final int PEER_ALL_ROWS = 3;
    private static final int PEER_SINGLE_ROW = 4;
    private static final UriMatcher uriMatcher;
    static {
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI(AUTHORITY, MessageContract.CONTENT_PATH, MESSAGE_ALL_ROWS);
        uriMatcher.addURI(AUTHORITY, MessageContract.CONTENT_PATH_ITEM, MESSAGE_SINGLE_ROW);
        uriMatcher.addURI(AUTHORITY, PeerContract.CONTENT_PATH, PEER_ALL_ROWS);
        uriMatcher.addURI(AUTHORITY, PeerContract.CONTENT_PATH_ITEM, PEER_SINGLE_ROW);
    }


    public ChatDbProvider() {
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        SQLiteDatabase database = databaseHelper.getWritableDatabase();
        databaseHelper.getWritableDatabase().execSQL("PRAGMA foreign_keys = ON");
        switch (uriMatcher.match(uri)) {
            case MESSAGE_ALL_ROWS:
                return database.delete(MessageContract.TABLE_NAME, null, null);
            case MESSAGE_SINGLE_ROW:
                selection = MessageContract.ID + "=?";
                selectionArgs = new String[] {String.valueOf(MessageContract.getId(uri))};
                return database.delete(MessageContract.TABLE_NAME, selection, selectionArgs);
            default:
                throw new IllegalArgumentException("Unsupported URI: " + uri);
        }
    }
```

Figure 3.2

```
    @Override
    public String getType(Uri uri) {
        switch (uriMatcher.match(uri)) {
            case MESSAGE_ALL_ROWS:
                return contentType(MessageContract.CONTENT);
            case MESSAGE_SINGLE_ROW:
                return contentItemType(MessageContract.CONTENT);
            case PEER_ALL_ROWS:
                return contentType(PeerContract.CONTENT);
            case PEER_SINGLE_ROW:
                return contentItemType(PeerContract.CONTENT);
            default:
                throw new IllegalArgumentException("Unsupported URI: " + uri);
        }
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        SQLiteDatabase database = databaseHelper.getWritableDatabase();
        databaseHelper.getWritableDatabase().execSQL("PRAGMA foreign_keys = ON");
        long rowId;
        switch (uriMatcher.match(uri)) {
            case MESSAGE_ALL_ROWS:
                rowId = database.insert(MessageContract.TABLE_NAME, null, values);
                if (rowId > 0) {
                    Uri instanceUri = MessageContract.CONTENT_URI(String.valueOf(rowId));
                    ContentResolver contentResolver = getContext().getContentResolver();
                    contentResolver.notifyChange(uri, null);
                    return instanceUri;
                }
                throw new SQLException("Insertion Failed");
            case PEER_ALL_ROWS:
                rowId = database.insert(PeerContract.TABLE_NAME, null, values);
                if (rowId > 0) {
                    Uri instanceUri = PeerContract.CONTENT_URI(String.valueOf(rowId));
                    ContentResolver contentResolver = getContext().getContentResolver();
                    contentResolver.notifyChange(MessageContract.CONTENT_URI, null);
                    return instanceUri;
                }
                throw new SQLException("Insertion Failed");
            default:
                throw new SQLException("Insertion Failed");
```

Figure 3.3

```
@Override
public Cursor query(Uri uri, String[] projection, String selection,
                    String[] selectionArgs, String sortOrder) {
    SQLiteDatabase database;
    try {
        database = databaseHelper.getWritableDatabase();
    } catch (SQLiteException e) {
        database = databaseHelper.getReadableDatabase();
    }
    database.execSQL("PRAGMA foreign_keys = ON");
    switch (uriMatcher.match(uri)) {
        case MESSAGE_ALL_ROWS:
            String query = "SELECT " + MessageContract.TABLE_NAME + "." + MessageContract.ID + " AS " + MessageContract.
                    MessageContract.TABLE_NAME + "." + MessageContract.MESSAGE_TEXT + " AS " + MessageContract.MESSAGE_T
                    "." + MessageContract.PEER_FK + ";";
            return database.rawQuery(query, null);
        case MESSAGE_SINGLE_ROW:
            return database.query(MessageContract.TABLE_NAME, projection, selection, selectionArgs, null, null, sortOrde
        case PEE[< Android API 17 Platform >] android.net
            proj public abstract class Uri extends Object
            retu implements Parcelable, Comparable<Uri>
        case PEER_SINGLE_ROW:
            if (PeerContract.getId(uri) == 0) {
                return database.query(PeerContract.TABLE_NAME, projection, selection, selectionArgs, null, null, null);
            }
            else {
                long rowId = PeerContract.getId(uri);
                selection = MessageContract.PEER_FK + "=?";
                selectionArgs = new String[] {String.valueOf(rowId)};
                projection = new String[] {MessageContract.ID, MessageContract.MESSAGE_TEXT, MessageContract.SENDER};
                return database.query(MessageContract.TABLE_NAME, projection, selection, selectionArgs, null, null, null
            }
        default:
            throw new IllegalArgumentException("Unknown Uri: " + uri);
    }
}
```

Figure 3.4

```
@Override
public int update(Uri uri, ContentValues values, String selection,
                  String[] selectionArgs) {
    SQLiteDatabase database = databaseHelper.getWritableDatabase();
    database.execSQL("PRAGMA foreign_keys = ON");
    switch (uriMatcher.match(uri)) {
        case MESSAGE_ALL_ROWS:
            return database.update(MessageContract.TABLE_NAME, values, null, null);
        case MESSAGE_SINGLE_ROW:
            selection = MessageContract.ID + "=?";
            selectionArgs = new String[] {String.valueOf(MessageContract.getId(uri))};
            return database.update(MessageContract.TABLE_NAME, values, selection, selectionArgs);
        default:
            throw new IllegalArgumentException("Unsupported URI: " + uri);
    }
}
```

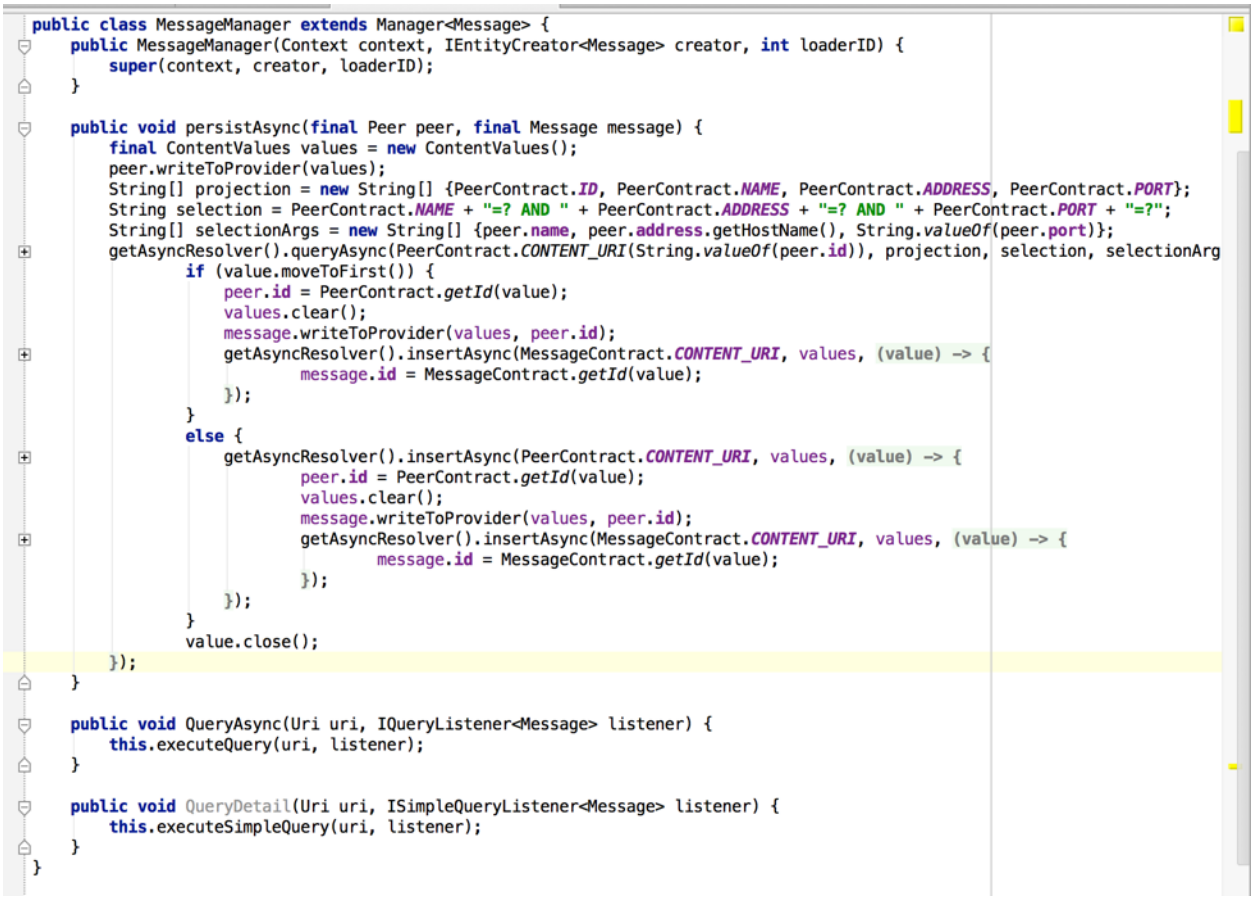
Figure 3.5

4. Define a PeerManager class which extends Manager class. The PeerManager is used to query the peer's details. Figure 4.1 shows the structure of the PeerManager.

```
/**
 * Created by wyf920621 on 2/23/15.
 */
public class PeerManager extends Manager<Peer> {
    public PeerManager(Context context, IEntityCreator<Peer> creator, int loaderID) {
        super(context, creator, loaderID);
    }

    public void QueryAsync(Uri uri, IQueryListener<Peer> listener) {
        this.executeQuery(uri, listener);
    }
}
```

Figure 4.1

5. Define a MessageManager class which is used to query the messages and manage the list view by QueryBuilder which implements LoaderManager.LoaderCallbacks<Cursor> interfaces. Figure 5.1 shows the structure of the MessageManager.



```
public class MessageManager extends Manager<Message> {
    public MessageManager(Context context, IEntityCreator<Message> creator, int loaderID) {
        super(context, creator, loaderID);
    }

    public void persistAsync(final Peer peer, final Message message) {
        final ContentValues values = new ContentValues();
        peer.writeToProvider(values);
        String[] projection = new String[] {PeerContract.ID, PeerContract.NAME, PeerContract.ADDRESS, PeerContract.PORT};
        String selection = PeerContract.NAME + "=? AND " + PeerContract.ADDRESS + "=? AND " + PeerContract.PORT + "=?";
        String[] selectionArgs = new String[] {peer.name, peer.address.getHostName(), String.valueOf(peer.port)};
        getAsyncResolver().queryAsync(PeerContract.CONTENT_URI(String.valueOf(peer.id)), projection, selection, selectionArg
                if (value.moveToFirst()) {
                    peer.id = PeerContract.getId(value);
                    values.clear();
                    message.writeToProvider(values, peer.id);
                    getAsyncResolver().insertAsync(MessageContract.CONTENT_URI, values, (value) -> {
                            message.id = MessageContract.getId(value);
                    });
                }
                else {
                    getAsyncResolver().insertAsync(PeerContract.CONTENT_URI, values, (value) -> {
                            peer.id = PeerContract.getId(value);
                            values.clear();
                            message.writeToProvider(values, peer.id);
                            getAsyncResolver().insertAsync(MessageContract.CONTENT_URI, values, (value) -> {
                                    message.id = MessageContract.getId(value);
                            });
                    });
                }
                value.close();
        });
    }

    public void QueryAsync(Uri uri, IQueryListener<Message> listener) {
        this.executeQuery(uri, listener);
    }

    public void QueryDetail(Uri uri, ISimpleQueryListener<Message> listener) {
        this.executeSimpleQuery(uri, listener);
    }
}
```

Figure 5.1

6. In the ChatServer Activity, use the MessageManager to manage the list view as Figure 6.1. It is also used to insert new data into database asynchronously as Figure 6.2.

```
manager = new MessageManager(this, (cursor) -> {
        return new Message(cursor);
}, CHAT_SERVER_LOADER_ID);

String[] from = new String[] {PeerContract.NAME, MessageContract.MESSAGE_TEXT};
int[] to = new int[] {R.id.peer_row, R.id.message_row};
cursorAdapter = new SimpleCursorAdapter(this, R.layout.peer_row, null, from, to, 0);
messageList.setAdapter(cursorAdapter);

manager.QueryAsync(MessageContract.CONTENT_URI, new IQueryListener<Message>() {
    public void handleResults(TypedCursor<Message> cursor) {
        cursorAdapter.swapCursor(cursor.getCursor());
        cursor.getCursor().setNotificationUri(getContentResolver(), MessageContract.CONTENT_URI);
    }

    public void closeResults() { cursorAdapter.changeCursor(null); }
});
/*
 * End Todo
 */
```

Figure 6.1

```
public void onClick(View v) {

    byte[] receiveData = new byte[1024];

    DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

    try {

        serverSocket.receive(receivePacket);
        Log.i(TAG, "Received a packet");

        InetAddress sourceIPAddress = receivePacket.getAddress();
        Log.i(TAG, "Source IP Address: " + sourceIPAddress);

        /*
         * TODO: Extract sender and receiver from message and display.
         */
        receiveData = receivePacket.getData();
        String temp = new String(receiveData, 0, receivePacket.getLength());
        if (!temp.isEmpty()) {
            String[] nameAndContent = SEPARATOR.split(temp);
            Peer peer = new Peer(nameAndContent[0], sourceIPAddress, serverSocket.getLocalPort());
            Message message = new Message(nameAndContent[1], nameAndContent[0]);
            manager.persistAsync(peer, message);
            getContentResolver().notifyChange(MessageContract.CONTENT_URI, null);
        }
        /*
         * End Todo
         */

    } catch (Exception e) {
        Log.e(TAG, e.getMessage());
        socketOK = false;
    }
}
```

Figure 6.2

7. In the PeerActivity, use the PeerManager to manage the list view to show all the peers' names as Figure 7.1.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Intent intent = getIntent();
    setContentView(R.layout.activity_peer);
    manager = new PeerManager(this,
            (cursor) -> {
                    return new Peer(cursor);
            }, PEER_ACTIVITY_LOADER_ID);

    listView = (ListView)findViewById(android.R.id.list);
    String[] from = new String[] {PeerContract.NAME};
    int[] to = new int[] {R.id.peer_list};
    cursorAdapter = new SimpleCursorAdapter(this, R.layout.peer_list, null, from, to, 0);
    setListAdapter(cursorAdapter);
    manager.QueryAsync(PeerContract.CONTENT_URI,
            new IQueryListener<Peer>() {
                public void handleResults(TypedCursor<Peer> cursor) {
                    cursorAdapter.swapCursor(cursor.getCursor());
                    cursor.getCursor().setNotificationUri(getContentResolver(), PeerContract.CONTENT_URI);
                }

                public void closeResults() { cursorAdapter.swapCursor(null); }
            });

    getListView().setChoiceMode(ListView.CHOICE_MODE_SINGLE);
    getListView().setOnItemClickListener((adapterView, view, position, rowId) -> {
            Cursor cursor = cursorAdapter.getCursor();
            cursor.moveToPosition(position);
            Peer peer = new Peer(cursor);
            peer.id = PeerContract.getId(cursor);
            Intent mIntent = new Intent(PeerActivity.this, PeerDetailActivity.class);
            mIntent.putExtra(PEER_ACTIVITY_KEY, peer);
            startActivity(mIntent);
    });
}
```

Figure 7.1

8. In the PeerDetailActivity, use the MessageManager to query the messages sent by a specific client, and show the details on the view. Figure 8.1 shows the structure of the PeerDetailActivity.

```
public class PeerDetailActivity extends Activity {
    public static final int PEER_DETIAL_ACTIVITY_LOADER_ID = 3;
    MessageManager messageManager;
    SimpleCursorAdapter cursorAdapter;
    ListView listView;
    TextView addressView;
    TextView portView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_peer_detail);
        Intent intent = getIntent();
        final Peer peer = intent.getParcelableExtra(PeerActivity.PEER_ACTIVITY_KEY);
        addressView = (TextView) findViewById(R.id.peer_ip);
        portView = (TextView) findViewById(R.id.peer_port);
        addressView.setText("IP Address: " + peer.address.getHostAddress());
        portView.setText("Port: " + peer.port);
        messageManager = new MessageManager(this,
                (cursor) -> {
                        return new Message(cursor);
                }, PEER_DETIAL_ACTIVITY_LOADER_ID);

        String[] from = new String[]{MessageContract.MESSAGE_TEXT};
        int[] to = new int[]{R.id.peer_message};

        cursorAdapter = new SimpleCursorAdapter(this, R.layout.peer_message, null, from, to, CursorAdapter.FLAG_REGISTER_CON
        listView = (ListView) findViewById(R.id.peer_messages);
        listView.setAdapter(cursorAdapter);
        Log.v("Peer detail id: ", String.valueOf(peer.id));
        messageManager.QueryAsync(PeerContract.CONTENT_URI(String.valueOf(peer.id)), new IQueryListener<Message>() {
            public void handleResults(TypedCursor<Message> cursor) {
                cursorAdapter.swapCursor(cursor.getCursor());
                cursor.getCursor().setNotificationUri(getContentResolver(), PeerContract.CONTENT_URI(String.valueOf(peer.id)
            }

            public void closeResults() { cursorAdapter.swapCursor(null); }
        });
    }
}
```

Figure 8.1

9. Finally, run the App as the video shows.