Assignment FOUR — Content providers

Xuefan Wu

Part 1: Book Store

.contracts.BookContract class

Defines AUTHORITY, CONTENT_URI, CONTENT_PATH and column names

```
public static final String TITLE = "title";
public static final String AUTHORS = "authors";
public static final String ISBN = "isbn";
public static final String PRICE = "price";
public static final String COLUMN_ID = "_id";
public static final String AUTHORITY = "edu.stevens.cs522.bookstore";
public static final String PATH ="BookTable";
public static final Uri CONTENT_URI = CONTENT_URI(AUTHORITY,PATH);
public static final String CONTENT_PATH = CONTENT_PATH(CONTENT_URI);
public static final String CONTENT_PATH_ITEM = CONTENT_PATH(CONTENT_URI("#"));
```

Defines getter (Cursor) and putter (ContentValues) operations

```
public static String getPrice(Cursor cursor) {
    String s = null;
    if(cursor != null && cursor.moveToFirst()){
        s= cursor.getString(cursor.getColumnIndexOrThrow(PRICE));
    }
    return s;
}

public static void putTitle(ContentValues values, String title) {

    values.put(TITLE, title);
}
```

.entities.Book entity class

Book(Cursor) constructor

```
public Book(Cursor cursor) {
    this.id = BookContract.getColumnId(cursor);
    this.title = BookContract.getTitle(cursor);
    this.authors = BookContract.getAuthors(cursor);
    this.isbn = BookContract.getIsbn(cursor);
    this.price = BookContract.getPrice(cursor);
}
```

writeToProvider(ContentValues) method

```
public void writeToProvider(ContentValues values,Book book) {
    //BookContract.putColumnID(values,book.id);
    BookContract.putTitle(values,book.title);
    BookContract.putAuthors(values, book.authors[0].toString());
    BookContract.putIsbn(values, book.isbn);
    BookContract.putPrice(values,book.price);
}
```

.providers.BookProvider class

Literals as before, CREATE_DATABASE follows spec, and is constructed using column names from contract

```
rivate static final String DATABASE_CREATE = "create table "+
        BOOK_TABLE + " (" + BookContract.COLUMN_ID + " integer primary key autoincrement, "
        +BookContract.TITLE +", "
        + BookContract.ISBN+ ", "
        +BookContract.PRICE+", "
        +BookContract.AUTHORS+")";
```

Use of SQLiteOpenHelper (subclass) to create/open database

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("PRAGMA foreign_keys=ON;");
    db.execSQL(DATABASE_CREATE);
    db.execSQL(DATABASE_AUTHOR_CREATE);
    db.execSQL("create index "+KEY_AUTHORSBOOK_INDEX+" on AuthorTable("+KEY_BOOK_FK_COLUMN+")")

}
```

query, insert, delete implemented, insert and delete implement notifyChange(), query sets notify listener in cursor

```
@Override
public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, St
    SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
    //qb.setTables(BOOK_TABLE);
    qb.setTables(BOOK_TABLE+" LEFT OUTER JOIN AuthorTable ON (BookTable._id = AuthorTable.book
    switch (uriMatcher.match(uri)){
        case BOOK:
            //qb.setProjectionMap(BOOK_PROJECTION_MAP);
            break;
        case BOOK_ID:
            qb.appendWhere(BookContract.COLUMN_ID+ "=" + uri.getPathSegments().get(1));
            break;
        case AUTHOR:

        default:
            throw new IllegalArgumentException("Unknown URI " + uri);
    }
    Cursor c = qb.query(db, projection,  selection, selectionArgs, " BookTable._id, BookTable.
    c.setNotificationUri(getContext().getContentResolver(), uri);
    return c;
}
```

```
public Uri insert(Uri uri, ContentValues values) {
    Uri _uri = null;
    switch (uriMatcher.match(uri)){
        case BOOK:
            long rowId = db.insert(BOOK_TABLE,null,values);
            if(rowId>0){
                _uri = ContentUris.withAppendedId(CONTENT_URI,rowId);
                ContentResolver cr = getContext().getContentResolver();
                cr.notifyChange(_uri, null);
            }
            break;
        case AUTHOR:
            long rowId1 = db.insert(AUTHOR_TABLE,null,values);
            if(rowId1>0){
                _uri = ContentUris.withAppendedId(Author_CONTENT_URI,rowId1);
                ContentResolver cr = getContext().getContentResolver();
                cr.notifyChange(_uri, null);
            }
            break;
        default:
            try {
                throw new SQLException("Failed to insert row into " + uri);
            } catch (SQLException e) {
                e.printStackTrace();
            }
    }
    return _uri;
    }
```

```java
@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {

    int count = 0;
    switch (uriMatcher.match(uri)){
        case BOOK:
            count = db.delete(BOOK_TABLE,selection,selectionArgs);
            break;
        case BOOK_ID:
            String id = uri.getPathSegments().get(1);
            count = db.delete( BOOK_TABLE, BookContract.COLUMN_ID +  " = " + id +
                    (!TextUtils.isEmpty(selection) ? " AND ("
                            + selection + ')' : ""), selectionArgs);
            break;
        default:
            throw new IllegalArgumentException("Unknown URI"+uri);
    }
    getContext().getContentResolver().notifyChange(uri, null);
    return count;
}
```

Authors in separate table in database, use of join and aggregation (GROUP-BY with GROUP-CONCAT) to return authors in query

```java
qb.setTables(BOOK_TABLE+" LEFT OUTER JOIN AuthorTable ON " +
        "(BookTable._id = AuthorTable.book_fk)");
```

```java
Cursor c = qb.query(db, projection,  selection, selectionArgs,
        " BookTable._id, BookTable.title, BookTable.price, BookTable.isbn",
        null, sortOrder);
```

BookstoreWithContentProvider:

Main activity uses loader manager for queries, callbacks are defined as methods in the main activity

```
LoaderManager lm = getLoaderManager();
lm.initLoader(BOOK_LOADER_ID, null, this);
```

```
public Loader<Cursor> onCreateLoader(int id, Bundle args) {

    switch (id){
        case BOOK_LOADER_ID:
            String[] projections = {BookContract.COLUMN_ID, BookContract.TITLE,
                    BookContract.AUTHORS };
            return new CursorLoader(this,
                    BookProvider.CONTENT_URI,
                    projections,null,null,null);
        case AUTHORS_LOADER_ID:
            //return

        default:
            return null;
    }
}

@Override
public void onLoadFinished(Loader<Cursor> loader, Cursor data) {

    this.mAdapter.swapCursor(data);
}

@Override
public void onLoaderReset(Loader<Cursor> loader) {

    this.mAdapter.swapCursor(null);
}
```

Main activity supports deletion of multiple books using multi-selection CAB

```
listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE_MODAL);
listView.setMultiChoiceModeListener(new AbsListView.MultiChoiceModeListener() {
```

Use of custom cursor adapter to display first author only in main activity

```
private class BookAdapter extends ResourceCursorAdapter {

    protected final static int ROW_LAYOUT = android.R.layout.simple_list_item_2;
    private HashMap<Integer, Boolean> mSelection = new HashMap<~>();

    public BookAdapter(Context context, int layout, Cursor c) { super(context, layout, c,0); }
```

BookstoreWithEntityManager:

AsyncContentResolver class derives from AsyncQueryHandler

```
public class AsyncContentResolver extends AsyncQueryHandler {
    public AsyncContentResolver(ContentResolver cr) { super(cr); }

    //INSERT
    public void insertAsync(Uri uri,
                            ContentValues values,
                            IContinue<Uri> callback){
        this.startInsert(0, callback, uri, values);
    }
    @Override
    protected void onInsertComplete(int token, Object cookie, Uri uri) {
        if(cookie!=null){
            IContinue<Uri> callback = (IContinue<Uri>) cookie;
            callback.kontinue(uri);
        }
        //super.onInsertComplete(token, cookie, uri);
    }
```

SimpleQueryBuilder uses AsyncContentResolver

```
public class SimpleQueryBuilder<T> implements IContinue<Cursor> {

    private IEntityCreator<T> helper;
    private ISimpleQueryListener 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> qb =
                new SimpleQueryBuilder<T>(helper,listener);

        AsyncContentResolver resolver =
                new AsyncContentResolver(context.getContentResolver());

        resolver.queryAsync(uri,null,null,null,null,qb);
    }
```

QueryBuilder uses LoaderManager

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

BookstoreManager used to access data, uses all three of the classes defined above, AsyncContentResolver used in BookstoreManager for asynchronous insert/update

```
public BookManager(Context context, IEntityCreator creator, int loadID) {
    super(context, creator, loadID);
}

public void persistAsync(final Book book){
    ContentValues contentValues = new ContentValues();
    book.writeToProvider(contentValues,book);
    getAsyncContentResolver().insertAsync(BookContract.CONTENT_URI, contentValues,
            (value) -> {
                    for(int i = 0; i<book.authors.length;i++) {
                        ContentValues values = new ContentValues();
                        values.put(AuthorContract.AUTHOR_NAME, book.authors[i].toStr
                        values.put(AuthorContract.BOOK_FK, BookProvider.getBook_fk(b
                        getAsyncContentResolver().insertAsync(AuthorContract.CONTENT
                    }


            }
    );
```