13. БАЗА ДАННЫХ SQLITE

В данном разделе будет реализовано взаимодействие приложения BookDepository с базой данных SQLite. Будет рассмотрена работа основных классов поддержки SQLite в Android для выполнения операций с базами данных SQLite, а именно операций открытия, чтения и записи.

SQLite — реляционная база данных с открытым кодом, как и MySQL или Postgresql. В отличие от других баз данных, SQLite хранит свои данные в простых файлах. Более подробную информацию можно найти в полной документации SQLite по адресу http://www.sqlite.org.

Определение схемы

Прежде чем создавать базу данных, необходимо решить, какая информация будет в ней храниться. В BookDepository хранится только список книг, поэтому следует определить одну таблицу с именем books (рисунок 13.1).

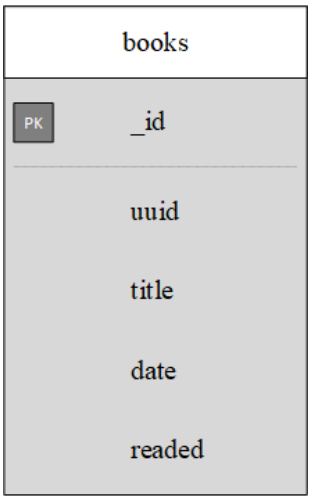

Рисунок 13.1 – Таблица books

Для определения в коде Java упрощенной схемы базы данных, в которой будет храниться имя таблицы с описанием ее столбцов, необходимо создать класс для хранения схемы. Этот класс будет называться **BookDbSchema**, но в диалоговом окне New Class следует ввести имя **database.BookDbSchema**. Файл BookDbSchema.java будет помещен в отдельный пакет database, который будет использоваться для организации всего кода, относящегося к базам данных.

В классе BookDbSchema определить внутренний класс BookTable для описания таблицы.

Листинг 13.1 – Определение BookTable (BookDbSchema.java)

```
public class BookDbSchema {
    public static final class BookTable {
        public static final String NAME = "books";
    }
}
```

Класс BookTable существует только для определения строковых констант, необходимых для описания основных частей определения таблицы. Определение начинается с имени таблицы в базе данных BookTable.NAME, за которым следуют описания столбцов.

Листинг 13.2 – Определение столбцов таблицы (BookDbSchema.java)

```
public class BookDbSchema {
    public static final class BookTable {
        public static final String NAME = "books";
        public static final class Cols {
            public static final String UUID = "uuid";
            public static final String TITLE = "title";
            public static final String DATE = "date";
            public static final String READED = "readed";
        }
    }
}
```

При наличии такого определения можно обращаться к столбцу с именем title в синтаксисе, безопасном для кода Java: BookTable.Cols.TITLE. Такой синтаксис существенно снижает риск изменения программы, если понадобится когда-нибудь изменить имя столбца или добавить новые данные в таблицу.

Построение исходной базы данных

После определения схемы можно переходить к созданию базы данных. Android предоставляет в классе Context низкоуровневые методы для открытия файла базы данных в экземпляр `SQLiteDatabase`: `openOrCreateDatabase(…)` и `databaseList()`.

Тем не менее на практике при открытии базы данных всегда следует выполнить ряд простых действий:

1. Проверить, существует ли база данных.

2. Если база данных не существует, создать ее, создать таблицы и заполнить их необходимыми исходными данными.

3. Если база данных существует, открыть ее и проверить версию BookDbSchema.

4. Если это старая версия, выполнить код преобразования ее в новую версию.

Android предоставляет класс `SQLiteOpenHelper`, который сделает все эти действия. Создать в пакете database класс с именем `BookBaseHelper`.

Листинг 13.3 – Создание BookBaseHelper (BookBaseHelper.java)

```
public class BookBaseHelper extends SQLiteOpenHelper {
    private static final int VERSION = 1;
    private static final String DATABASE_NAME = "bookBase.db";
    public BookBaseHelper(Context context) {
        super(context, DATABASE_NAME, null, VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
                newVersion) {
    }
}
```

Класс SQLiteOpenHelper избавляет разработчика от рутинной работы при открытии SQLiteDatabase. Его следует использовать в BookLab для создания базы данных.

Листинг 13.4 – Открытие SQLiteDatabase (BookLab.java)

```
public class BookLab {
    private static BookLab sBookLab;
    private List<Book> mBooks;
    private Context mContext;
    private SQLiteDatabase mDatabase;
    ...
    private BookLab(Context context) {
        mContext = context.getApplicationContext();
        mDatabase = new BookBaseHelper(mContext)
                .getWritableDatabase();
        mBooks = new ArrayList<>();
    }
...
```

При вызове `getWritableDatabase()` класс `BookBaseHelper`:

1) открывает /data/data/ru.rsue.android.bookdepository/databases/ bookBase.db. Если файл базы данных не существует, то он создается;

2) если база данных открывается впервые, вызывает метод `onCreate(SQLiteDatabase)` с последующим сохранением последнего номера версии;

3) если база данных открывается не впервые, проверяет номер ее версии. Если версия базы данных в `BookOpenHelper` выше, то вызывается метод `onUpgrade(SQLiteDatabase, int, int)`.

Код создания исходной базы данных размещается в `onCreate(SQLiteDatabase)`, код обновления – в `onUpgrade(SQLiteDatabase, int, int)`, а дальше все работает само собой.

Пока приложение BookDepository существует только в одной версии, так что на `onUpgrade(…)` можно не обращать внимания. Нужно только создать таблицы базы данных в `onCreate(…)`. Для этого будет использоваться класс `BookTable`, являющийся внутренним классом `BookDbSchema`.

Процедура импортирования состоит из двух шагов. Сначала запишите начальную часть кода создания SQL:

Листинг 13.5 – Первая часть onCreate(…) (BookBaseHelper.java)

```
@Override
public void onCreate(SQLiteDatabase db) {
   db.execSQL("create table " + BookDbSchema.BookTable.NAME);
}
```

Навести курсор на слово BookTable и нажать Alt+Enter. Затем выбрать первый вариант: `Add import for 'ru.rsue.android.bookdepository.database.BookDbSchema.BookTable'`.

Android Studio генерирует директиву import, которая позволяет ссылаться на строковые константы из BookDbSchema.BookTable в форме BookTable.Cols.UUID (вместо того, чтобы вводить полное имя BookDbSchema.BookTable.Cols.UUID). Использовать это обстоятельство, чтобы завершить ввод кода определения таблицы.

Листинг 13.6 – Создание таблицы (BookBaseHelper.java)

```
@Override
public void onCreate(SQLiteDatabase db) {
   db.execSQL("create table " + BookTable.NAME + "(" +
      " _id integer primary key autoincrement, " +
      BookTable.Cols.UUID + ", " +
      BookTable.Cols.TITLE + ", " +
      BookTable.Cols.DATE + ", " +
      BookTable.Cols.READED + ")"
   );
}
```

Запустить BookDepository; приложение создаст базу данных. Если приложение выполняется в эмуляторе или на «рутованном» устройстве, то можно увидеть созданную базу данных. (На физическом устройстве это невозможно — база данных хранится в закрытой области.) Выполнить

команду Tools→Android→Android Device Monitor и заглянуть в каталог /data/data/ru.rsue.android.bookdepository/databases/ (рисунок 13.2).

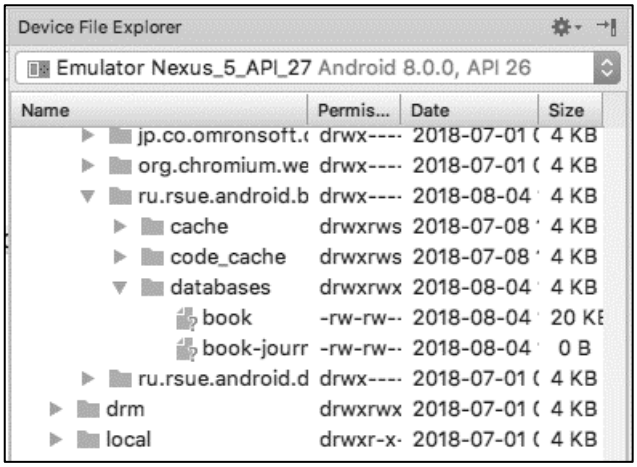

Рисунок 13.2 – Созданная база данных

Решение проблем при работе с базами данных

При написании кода для работы с базами данных SQLite иногда требуется слегка изменить структуру базы данных. Например, добавить новое поле. Для этого в таблицу придется добавить новый столбец. «Правильный» способ решения этой задачи заключается во включении в SQLiteOpenHelper кода повышения номера версии с последующим обновлением таблиц в `onUpgrade(…)`.

И этот «правильный» способ потребует изрядного объема кода — совершенно смехотворного, когда просто необходимо довести до ума первую или вторую версию базы данных. На практике лучше всего уничтожить базу данных и начать все заново, чтобы метод `SQLiteOpenHelper.onCreate(…)` был вызван снова.

Самый простой способ уничтожения базы — удаление приложения с устройства. А самый простой способ удаления приложений в стандартном варианте Android — открыть менеджер приложений и перетащить значок BookDepository к области Uninstall у верхнего края экрана. (Если на вашем устройстве используется нестандартная версия Android, процесс может выглядеть иначе.)

Следует помнить об этом приеме, если возникнут проблемы при работе с базами данных в этом разделе.

Изменение кода BookLab

Итак, теперь в приложении есть база данных, поэтому для дальнейшей работы следует изменить довольно большой объем кода в BookLab, чтобы хранение данных осуществлялось в базе данных mDatabase, а не в mBooks.

Для начала надо расчистить место для работы. Удалить из BookLab весь код, относящийся к mBooks.

Листинг 13.7 – Удаление лишнего (BookLab.java)

```
public class BookLab {
    private static BookLab sBookLab;
    private List<Book> mBooks;
    private Context mContext;
    private SQLiteDatabase mDatabase;
    public static BookLab get(Context context) {
        ...
    }
    private BookLab(Context context) {
        mContext = context.getApplicationContext();
        mDatabase = new BookBaseHelper(mContext)
        .getWritableDatabase();
        mBooks = new ArrayList<>();
    }
    public void addBook(Book b) {
        mBooks.add(b);
    }
    public List<Book> getBooks() {
        return mBooks;
        return new ArrayList<>();
    }
    public Book getBook(UUID id) {
        for (Book book : mBooks) {
            if (book.getId().equals(id)) {
                return book;
            }
        }
        return null;
    }
}
```

Приложение BookDepository остается в состоянии, которое вряд ли можно назвать работоспособным, но в дальнейшем это будет исправлено.

Запись в базу данных

Работа с SQLiteDatabase начинается с записи данных. Приложение должно вставлять новые записи в существующую таблицу, а также обновлять уже существующие данные при изменении информации.

Использование ContentValues

Запись и обновление баз данных осуществляются с помощью класса ContentValues. Класс ContentValues обеспечивает хранение пар «ключ-значение», как и контейнер Java HashMap или объекты Bundle. Однако в

отличие от HashMap или Bundle, он предназначен для хранения типов данных, которые могут содержаться в базах данных SQLite.

Экземпляры ContentValues будут несколько раз создаваться из Book в коде BookLab. Добавить закрытый метод, который будет преобразовывать объект Book в ContentValues.

Листинг 13.8 – Создание ContentValues (BookLab.java)

```
...
    public getBook(UUID id) {
        return null;
    }
    private static ContentValues getContentValues(Book book) {
        ContentValues values = new ContentValues();
        values.put(BookTable.Cols.UUID, book.getId().toString());
        values.put(BookTable.Cols.TITLE, book.getTitle());
        values.put(BookTable.Cols.DATE, book.getDate().getTime());
        values.put(BookTable.Cols.READED, book.isReaded() ? 1 : 0);
        return values;
    }
}
```

В качестве ключей использовать имена столбцов. Эти имена не выбираются произвольно; они определяют столбцы, которые должны вставляться или обновляться в базе данных. Если имя отличается от содержащегося в базе данных (например, из-за опечатки), операция вставки или обновления завершится неудачей. В приведенном фрагменте указаны все столбцы, кроме столбца _id, который генерируется автоматически для однозначной идентификации записи.

Вставка и обновление записей

Объект ContentValues создан, можно переходить к добавлению записи в базу данных. Заполнить метод addBook(Book) новой реализацией.

Листинг 13.9 – Вставка записи (BookLab.java)

```
public void addBook(Book b) {
    ContentValues values = getContentValues(b);
    mDatabase.insert(BookTable.NAME, null, values);
}
```

Метод `insert(String, String, ContentValues)` получает два важных аргумента; еще один аргумент используется относительно редко. Первый аргумент определяет таблицу, в которую выполняется вставка, — в данном случае BookTable.NAME. Последний аргумент содержит вставляемые данные.

Далее следует добавить метод обновления строк в базе данных.

Листинг 13.10 – Обновление записи (BookLab.java)

```
public Book getBook(UUID id) {
    return null;
}
public void updateBook(Book book) {
    String uuidString = book.getId().toString();
    ContentValues values = getContentValues(book);
    mDatabase.update(BookTable.NAME, values,
    BookTable.Cols.UUID + " = ?",
    new String[] {
        uuidString
    });
}
private static ContentValues getContentValues(Book book) {
    ContentValues values = new ContentValues();
    values.put(BookTable.Cols.UUID, book.getId().toString());
...
```

Метод `update(String, ContentValues, String, String[])` начинается так же, как `insert(…)`, при вызове передается имя таблицы и объект ContentValues, который должен быть присвоен каждой обновляемой записи. Однако последняя часть отличается, потому что в этом случае необходимо указать, какие именно записи должны обновляться. Для этого строится условие WHERE (третий аргумент), за которым следуют значения аргументов в условии WHERE (завершающий массив String[]).

Экземпляры Book, изменяемые в BookFragment, должны быть записаны в базу данных при завершении BookFragment. Добавить переопределение `BookFragment.onPause()`, которое обновляет копию Book из BookLab.

Листинг 13.11 – Запись обновлений (BookFragment.java)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    UUID bookId = (UUID) getArguments().getSerializable(ARG_BOOK_ID);
    mBook = BookLab.get(getActivity()).getBook(bookId);
}
@Override
public void onPause() {
    super.onPause();
    BookLab.get(getActivity()).updateBook(mBook);
}
```

К сожалению, проверить работоспособность этого кода пока не удастся. С проверкой придется подождать, пока приложение не научится читать обновленные данные. Чтобы убедиться в том, что программа нормально компилируется, необходимо запустить BookDepository еще один

раз, прежде чем переходить к следующему разделу. В приложении должен отображаться пустой список.

Чтение из базы данных

Чтение данных из SQLite осуществляется методом query(…). При вызове `SQLiteDatabase.query(…)` происходит много различных операций; метод существует в нескольких перегруженных версиях. Версия, которая будет использована, выглядит так:

```
public Cursor query(
      String table,
      String[] columns,
      String where,
      String[] whereArgs,
      String groupBy,
      String having,
      String orderBy,
      String limit)
```

Аргумент table содержит таблицу, к которой обращен запрос. Аргумент columns определяет столбцы, значения которых нужны, и порядок их извлечения. Наконец, where и whereArgs работают так же, как и в команде `update(…)`.

Создать вспомогательный метод, в котором будет вызываться метод `query(…)` для таблицы BookTable.

Листинг 13.12 – Запрос для получения данных Book (BookLab.java)

```
...
   values.put(BookTable.Cols.DATE, book.getDate().getTime());
   values.put(BookTable.Cols.READED, book.isReaded() ? 1 : 0);
   return values;
}
private Cursor queryBooks(String whereClause, String[] whereArgs) {
   Cursor cursor = mDatabase.query(
      BookTable.NAME,
      null, // Columns - null выбирает все столбцы
      whereClause,
      whereArgs,
      null, // groupBy
      null, // having
      null // orderBy
   );
   return cursor;
}
```

Использование CursorWrapper

Класс Cursor как средство работы с данными таблиц оставляет желать лучшего. По сути, он просто возвращает низкоуровневые значения столбцов. Процедура получения данных из Cursor выглядит так:

```
String uuidString = cursor.getString(
           cursor.getColumnIndex(BookTable.Cols.UUID));
String title = cursor.getString(
           cursor.getColumnIndex(BookTable.Cols.TITLE));
long date = cursor.getLong(
           cursor.getColumnIndex(BookTable.Cols.DATE));
int isReaded = cursor.getInt(
           cursor.getColumnIndex(BookTable.Cols.READED));
```

Каждый раз, когда будет происходить извлечение Book из курсора, этот код придется записывать снова. (Не говоря уже о коде создания экземпляра Book с этими значениями!)

Поэтому следует создать собственный субкласс Cursor, который выполняет эту операцию в одном месте. Для написания субкласса курсора проще всего воспользоваться CursorWrapper — этот класс позволяет дополнить класс Cursor, полученный извне, новыми методами.

Создать новый класс в пакете базы данных с именем BookCursorWrapper.

Листинг 13.13 – Создание BookCursorWrapper (BookCursorWrapper.java)

```
public class BookCursorWrapper extends CursorWrapper {
   public BookCursorWrapper(Cursor cursor) {
      super(cursor);
   }
}
```

Класс создает тонкую «обертку» для Cursor. Он содержит те же методы, что и инкапсулированный класс Cursor, и вызов этих методов приводит ровно к тем же последствиям. Все это не имело бы смысла, если бы не возможность добавления новых методов для работы с инкапсулированным классом Cursor.

Добавить метод `getBook()` для извлечения данных столбцов. (Использовать для BookTable прием импортирования, состоящий из двух шагов, как это было сделано ранее.)

Листинг 13.14 – Добавление метода getBook() (BookCursorWrapper.java)

```
public class BookCursorWrapper extends CursorWrapper {
   public BookCursorWrapper(Cursor cursor) {
      super(cursor);
   }
```

```
    public Book getBook() {
        String uuidString =
                  getString(getColumnIndex(BookTable.Cols.UUID));
        String title = getString(getColumnIndex(BookTable.Cols.TITLE));
        long date = getLong(getColumnIndex(BookTable.Cols.DATE));
        int isReaded = getInt(getColumnIndex(BookTable.Cols.READED));
        return null;
    }
}
```

Метод должен возвращать объект Book с соответствующим значением UUID. Добавить в Book конструктор для выполнения этой операции.

Листинг 13.15 – Добавление конструктора Book (Book.java)

```
public Book() {
    this(UUID.randomUUID());
    mId = UUID.randomUUID();
    mDate = new Date();
}
public Book(UUID id) {
    mId = id;
    mDate = new Date();
}
```

А затем доработать метод `getBook()`.

Листинг 13.16 – Окончательная версия getBook() (BookCursorWrapper.java)

```
public Book getBook() {
    ...
    int isReaded = getInt(getColumnIndex(BookTable.Cols.READED));
    Book book = new Book(UUID.fromString(uuidString));
    book.setTitle(title);
    book.setDate(new Date(date));
    book.setReaded(isReaded != 0);
    return book;
    return null;
}
```

(Android Studio предлагает выбрать между java.util.Date и java.sql.Date. И хотя работа осуществляется с базой данных, но здесь следует выбрать java.util.Date.)

Преобразование в объекты модели

С BookCursorWrapper процесс получения List<Book> от BookLab достаточно прямолинеен. Курсор, полученный при запросе, упаковывается в BookCursorWrapper, после чего его содержимое перебирается методом `getBook()` для получения объектов Book.

Для первой части перевести queryBooks(…) на использование BookCursorWrapper.

Листинг 13.17 – Использование CursorWrapper (BookLab.java)

```
~~private Cursor queryBooks(String whereClause, String[] whereArgs) {~~
private BookCursorWrapper queryBooks(String whereClause, String[]
          whereArgs) {
   Cursor cursor = mDatabase.query(
      ...
   );
   ~~return cursor;~~
   return new BookCursorWrapper(cursor);
}
```

Метод getBooks() принял нужную форму. Добавить код запроса всех книг, организовать перебор курсора и заполнения списка Book.

Листинг 13.18 – Возвращение списка книг (BookLab.java)

```
public List<Book> getBooks() {
   ~~return new ArrayList<>();~~
   List<Book> books = new ArrayList<>();
   BookCursorWrapper cursor = queryBooks(null, null);
   try {
      cursor.moveToFirst();
      while (!cursor.isAfterLast()) {
         books.add(cursor.getBook());
         cursor.moveToNext();
      }
   } finally {
      cursor.close();
   }
   return books;
}
```

Курсоры базы данных всегда устанавливаются в определенную позицию в результатах запроса. Таким образом, чтобы извлечь данные из курсора, его следует перевести к первому элементу вызовом moveToFirst(), а затем прочитать данные строки. Каждый раз, когда потребуется перейти к следующей записи, вызывается moveToNext(), пока isAfterLast(), наконец, не сообщит, что указатель вышел за пределы набора данных.

Последнее, что осталось сделать, — вызвать close() для объекта Cursor. Не следует забывать об этой служебной операции, это важно. Если забыть закрыть курсор, устройство Android начнет выдавать в журнал сообщения об ошибках. Со временем это приведет к исчерпанию файловых дескрипторов и сбою приложения. Итак, курсоры нужно закрывать.

Метод `BookLab.getBook(UUID)` похож на `getBooks()`, не считая того, что он должен извлечь только первый элемент данных (если тот присутствует).

Листинг 13.19 – Переработка getBook(UUID) (BookLab.java)

```
public Book getBook(UUID id) {
    return null;
    BookCursorWrapper cursor = queryBooks(
        BookTable.Cols.UUID + " = ?",
        new String[] { id.toString() }
    );
    try {
        if (cursor.getCount() == 0) {
            return null;
        }
        cursor.moveToFirst();
        return cursor.getBook();
    } finally {
        cursor.close();
    }
}
```

На данный момент удалось сделать следующее:

– реализована возможность вставлять новые книги, так что код, добавляющий Book в BookLab при нажатии элемента действия New Book, теперь работает;

– приложение обращается с запросами к базе данных, так что BookPagerActivity видит все объекты Book в BookLab;

– метод BookLab.getBook(UUID) тоже работает, так что каждый экземпляр BookFragment, отображаемый в BookPagerActivity, отображает существующий объект Book.

Теперь при нажатии *Навая книга* в BookPagerActivity появляется новый объект Book.

Запустить BookDepository и убедиться в том, что эта функциональность работает. Если что-то пошло не так, перепроверить свои реализации из этого раздела.

Обновление данных модели

Однако работа еще не закончена. Книги сохраняются в базе данных, но данные не читаются из нее. Таким образом, если нажать кнопку Back в процессе редактирования новой книги, оно не появится в BookListActivity.

Дело в том, что BookLab теперь работает немного иначе. Прежде существовал только один список List<Book> и один объект для каждой книги: тот, что содержится в List<Book>. Это объяснялось тем, что

переменная mBooks была единственным авторитетным источником данных о книгах, известный приложению.

Сейчас ситуация изменилась. Переменная mBooks исчезла, так что список List<Book>, возвращаемый getBooks(), представляет собой «моментальный снимок» данных Book на некоторый момент времени. Чтобы обновить BookListActivity, необходимо обновить этот снимок.

Большинство необходимых составляющих уже готово. BookListActivity уже вызывает `updateUI()` для обновления других частей интерфейса. Остается лишь заставить этот метод обновить его представление BookLab.

Сначала добавить в BookAdapter метод `setBooks(List<Book>)`, чтобы закрепить отображаемые в нем данные.

Листинг 13.20 – Добавление setBooks(List<Book>) (BookListFragment.java)

```
private class BookAdapter extends RecyclerView.Adapter<BookHolder> {
    ...
    @Override
    public int getItemCount() {
        return mBooks.size();
    }
    public void setBooks(List<Book> books) {
        mBooks = books;
    }
}
```

Вызвать setBooks(List<Book>) в updateUI().

Листинг 13.21 – Вызов setBook(List<>) (BookListFragment.java)

```
private void updateUI() {
    ...
    if (mAdapter == null) {
        ...
    } else {
        mAdapter.setBooks(books);
        mAdapter.notifyItemChanged(position);
    }
    updateSubtitle();
}
```

Теперь все должно работать правильно. Запустить BookDepository и убедиться в том, что есть возможность добавить книгу, нажать кнопку Back, и эта книга появится в BookListActivity.

Заодно можно проверить, что вызовы `updateBook(Book)` в BookFragment работают. Нажать на книгу и отредактировать её краткое описание в BookPagerActivity. Нажать кнопку Back и убедиться в том, что новый текст появился в списке.

Самостоятельные задания.

Задание 1. Удаление книги

В ранее добавленный элемент действия *Удалить книгу*, необходимо дополнить возможностью удаления информации из базы данных. Для этого следует вызывать метод `deleteBook(Book)` для BookLab, который вызывает метод `mDatabase.delete(…)` для завершения работы.

14. НЕЯВНЫЕ ИНТЕНТЫ

В Android можно запустить активность из другого приложения на устройстве при помощи *неявного интента* (implicit intent). В явном интенте задается класс запускаемой активности, а ОС запускает его. В неявном интенте описывается операция, которая должна быть выполнена, а ОС запускает активность соответствующего приложения.

В приложении BookDepository будет использован неявный интент для отправки текстовых отчетов о книге. Пользователь получит позможность выбрать приложение их список для отправки отчета (рисунок 14.1).

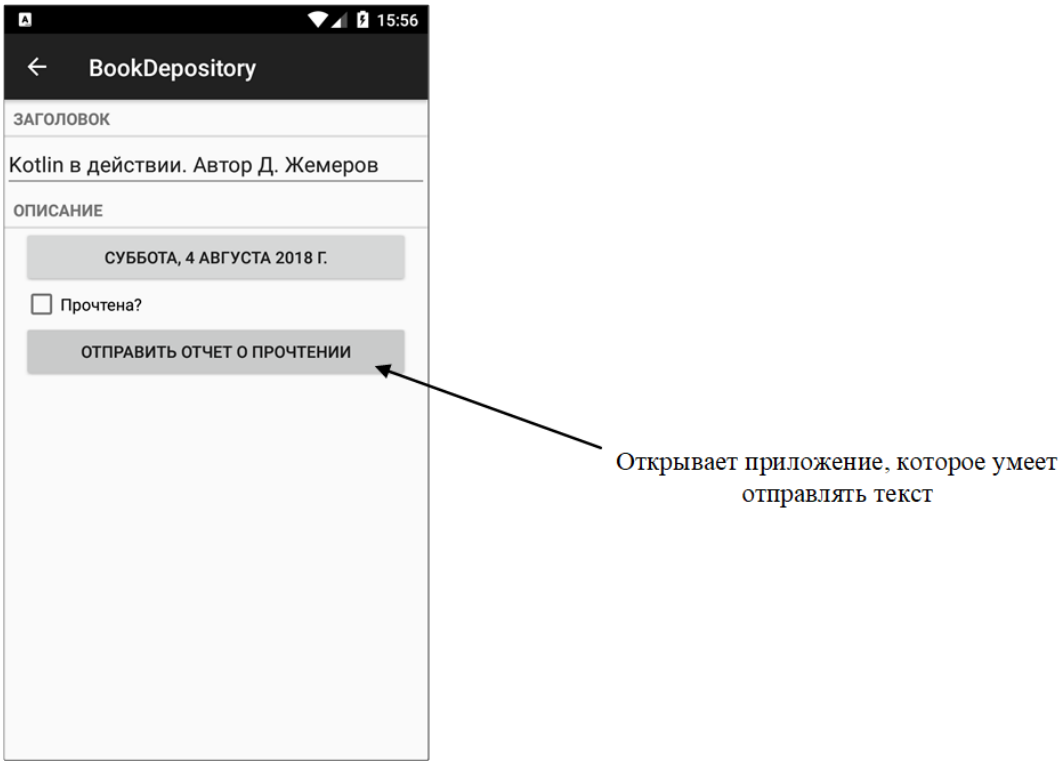

Рисунок 14.1 – Открытие приложений для отправки отчетов

Использовать функциональность других приложений при помощи неявных интентов намного проще, чем писать собственные реализации стандартных задач. Пользователям также нравится работать с приложениями, которые им уже хорошо знакомы, в сочетании с используемым приложением.

Прежде чем создавать неявные интенты, необходимо выполнить с BookDepository ряд подготовительных действий:

- добавить в макеты BookFragment кнопу отправки отчета;
- создать отчет о прочтении книги с использованием форматных строк ресурсов.

Добавление кнопок

На первом этапе необходимо включить в макеты BookFragment новую кнопку и добавить строку в ресурсы, которая будет отображаться на кнопке.

Листинг 14.1 – Добавление строки для надписи на кнопке (strings.xml)

```
...
    <string name="subtitle_format">%1$s books</string>
    <string name="book_report_text">Отправить отчет о прочтении</string>
</resources>
```

Добавить в файл layout/fragment_book.xml виджет Button, представленный на рисунке 14.2.

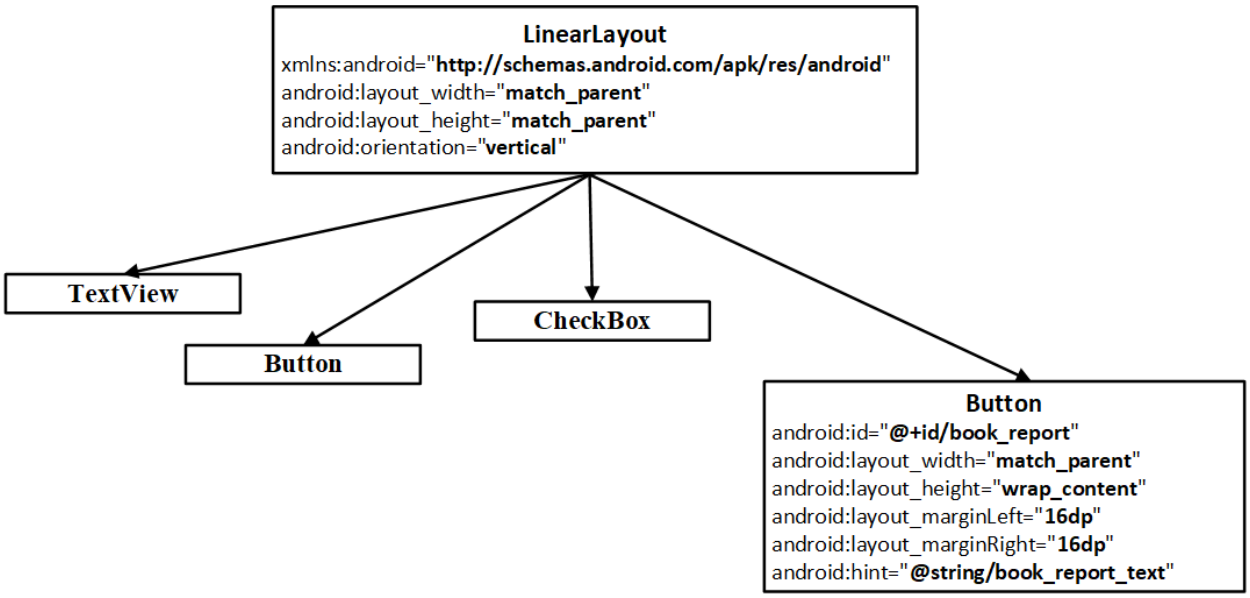

Рисунок 14.2 – Добавление кнопки для отправки отчетов (layout/fragment_book.xml)

В альбомном макете изменения будут аналогичными. Измененный макет изображен на рисунке 14.3.

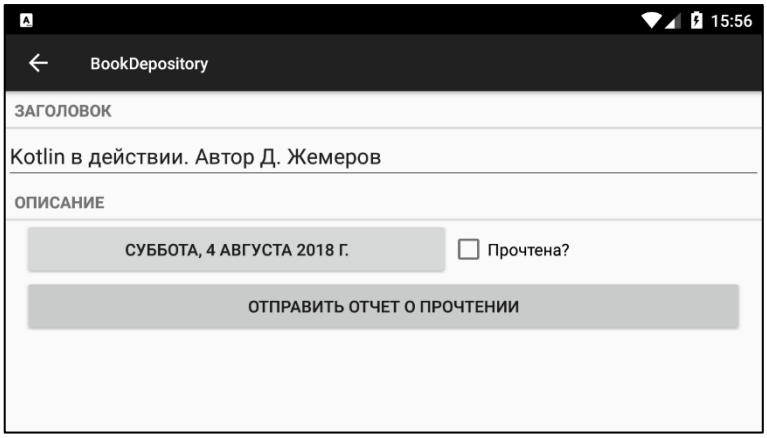

Рисунок 14.3 – Измененный альбомный макет

На этой стадии можно проверить макеты в области предварительного просмотра или запустить приложение BookDepository, чтобы убедиться в правильности расположения новой кнопки.

Форматные строки

Вторым подготовительным шагом станет создание шаблона отчета о прочтении книги, который заполняется информацией о конкретной книге, т.к. подробная информация недоступна до стадии выполнения, необходимо использовать форматную строку с заполнителями, которые будут заменяться во время выполнения. Форматная строка будет выглядеть так:

```
<string name="book_report">%1$s Книга была запланирована к прочтению %2$s и %3$s </string>
```

Поля %1$s, %2$s и т. д. — заполнители для строковых аргументов. В коде вызывается `getString(…)`, куда передается форматная строка и необходимое количество строк в том порядке, в каком они должны заменять заполнители.

Сначала следует добавить в strings.xml строки из листинга 14.2.

Листинг 14.2 – Добавление строковых ресурсов (strings.xml)

```
...
   <string name="book_report_text">Отправить отчет о прочтении</string>
   <string name="book_report">%1$s
       Книга была запланирована к прочтению %2$s и %3$s
   </string>
   <string name="book_report_readed">Книга прочитана</string>
   <string name="book_report_unreaded">Книга не прочитана</string>
   <string name="book_report_subject">BookDepository отчет о прочтении
             </string>
   <string name="send_report">Отправить отчет о прочтении через</string>
 </resources>
```

В файле BookFragment.java необходимо добавить метод, который создает три строки, соединяет их и возвращает полный отчет.

Листинг 14.3 – Добавление метода getBookReport() (BookFragment.java)

```
...
private void updateDate() {
    mDateButton.setText(...);
}
private String getBookReport() {
    String readedString = null;
    if (mBook.isReaded()){
        readedString = getString(R.string.book_report_readed);
    }else {
        readedString = getString(R.string.book_report_unreaded);
    }
```

```
    String dateFormat = "EEE, MMM dd";
    String dateString = DateFormat
              .getDateInstance(DateFormat.MEDIUM).format(mBook.getDate());
    String report = getString(R.string.book_report,
        mBook.getTitle(), dateString, readedString);
    return report;
}
```

Обратить внимание: класс `DateFormat` существует в двух версиях, `android.text.format.DateFormat` и `java.text.DateFormat`. Используйте `android.text.format.DateFormat`.

Использование неявных интентов

Объект Intent описывает для ОС некую операцию, которую необходимо выполнить. Для *явных* интентов, использовавшихся до настоящего момента, разработчик явно указывает активность, которую должна запустить ОС.

```
Intent intent = new Intent(getActivity(), BookPagerActivity.class);
intent.putExtra(EXTRA_BOOK_ID, bookId);
startActivity(intent);
```

Для *неявных* интентов разработчик описывает выполняемую операцию, а ОС запускает активность, которая ранее сообщила о том, что она способна выполнять эту операцию. Если ОС находит несколько таких активностей, пользователю предлагается выбрать нужную.

Строение неявного интента

Ниже перечислены важнейшие составляющие интента, используемые для определения выполняемой операции:

- выполняемое *действие* (action) — обычно определяется константами из класса Intent. Так, для просмотра URL-адреса используется константа Intent.ACTION_VIEW, а для отправки данных — константа Intent.ACTION_SEND;
- местонахождение *данных* — это может быть как ссылка на данные, которые находятся за пределами устройства (например, URL веб-страницы), так и URI файла или URI контента, который ссылается на запись ContentProvider;
- *тип* данных, с которыми работает действие, — тип MIME (например, text/html или audio/mpeg3). Если в интент включено местонахождение данных, то тип обычно удается определить по этим данным;
- необязательные *категории* — если действие указывает, *что* нужно сделать, категория обычно описывает, где, когда или как следует использовать операцию. Android использует категорию

android.intent.category.LAUNCHER для обозначения активностей, которые должны отображаться в лаунчере приложений верхнего уровня. С другой стороны, категория android.intent.category.INFO обозначает активность, которая выдает пользователю информацию о пакете, но не отображается в лаунчере.

Простой неявный интент для просмотра веб-сайта включает действие Intent.ACTION_VIEW и объект данных Uri с URL-адресом сайта.

На основании этой информации ОС запускает соответствующую активность соответствующего приложения. (Если ОС обнаруживает более одного кандидата, пользователю предлагается принять решение.)

Активность сообщает о себе как об исполнителе для ACTION_VIEW при помощи фильтра интентов в манифесте. Например, если создается приложение-браузер, то включается следующий фильтр интентов в объявление активности, реагирующей на ACTION_VIEW.

```
<activity
  android:name=".BrowserActivity"
  android:label="@string/app_name" >
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:scheme="http" android:host="developer.android.com"/>
  </intent-filter>
</activity>
```

Категория DEFAULT должна явно задаваться в фильтрах интентов. Элемент action в фильтре интентов сообщает ОС, что активность способна выполнять операцию, а категория DEFAULT — что она желает рассматриваться среди кандидатов на выполнение операции. Категория DEFAULT неявно добавляется к почти любому неявному интенту.

Неявные интенты, как и явные, также могут включать дополнения. Однако дополнения неявного интента не используются ОС для поиска соответствующей активности. Также следует отметить, что компоненты действия и данных интента могут использоваться в сочетании с явными интентами.

Отправка отчета

Чтобы закрепить на практике описанную схему, будет создан неявный интент для отправки отчета о прочтении книг в приложении BookDepository. Операция, которую нужно выполнить, — отправка простого текста; отчет представляет собой строку. Таким образом, действие неявного интента будет представлено константой ACTION_SEND. Интент не содержит ссылок на данные и не имеет категорий, но определяет тип text/plain.

В методе `BookFragment.onCreateView(…)` получить ссылку на кнопку Send Book Report и назначить для нее слушателя. В реализации слушателя создать неявный интент и передать его `startActivity(Intent)`.

Листинг 14.4 – Отправка отчета о книге (BookFragment.java)

```
private Book mBook;
private EditText mTitleField;
private Button mDateButton;
private Button mReportButton;
...
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    ...
    mReportButton = (Button)v.findViewById(R.id.book_report);
    mReportButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            Intent i = new Intent(Intent.ACTION_SEND);
            i.setType("text/plain");
            i.putExtra(Intent.EXTRA_TEXT, getBookReport());
            i.putExtra(Intent.EXTRA_SUBJECT,
                    getString(R.string.book_report_subject));
            startActivity(i);
        }
    });
    return v;
}
```

Здесь используется конструктор Intent, который получает строку с константой, описывающей действие. Также существуют другие конструкторы, которые могут использоваться в зависимости от вида создаваемого неявного интента. Информацию о них можно найти в справочной документации Intent.

Текст отчета и строка темы включаются в дополнения. Обратите внимание на использование в них констант, определенных в классе Intent. Любая активность, реагирующая на интент, знает эти константы и то, что следует делать с ассоциированными значениями.

Запустить приложение BookDepository и нажать кнопку Send Book Report. Так как этот интент с большой вероятностью совпадет со многими активностями на устройстве, скорее всего, на экране появится список активностей (рисунок 14.4).

Если список не появился, это может означать одно из двух: либо приложение по умолчанию уже было назначено для идентичного неявного интента, либо на устройстве имеется всего одна активность, способная реагировать на этот интент.

Также имеется возможность создать список, который будет отображаться каждый раз при использовании неявного интента для запуска активности. После создания неявного интента способом, показанным ранее, вызывается следующий метод Intent и передается ему неявный интент и строку с заголовком:

```
public static Intent createChooser(Intent target, String title).
```

Затем интент, возвращенный `createChooser(…)`, передается `startActivity(…)`.

В файле BookFragment.java создать список выбора для отображения активностей, реагирующих на неявный интент.

Листинг 14.5 – Использование списка выбора (BookFragment.java)

```
public void onClick(View v) {
    ...
    i.putExtra(Intent.EXTRA_SUBJECT,
            getString(R.string.book_report_subject));
    i = Intent.createChooser(i, getString(R.string.send_report));
    startActivity(i);
}
```

Запустить приложение BookDepository и нажать кнопку Send Book Report. Если в системе имеется несколько активностей, способных обработать интент, на экране появляется список для выбора (рисунок 14.5).

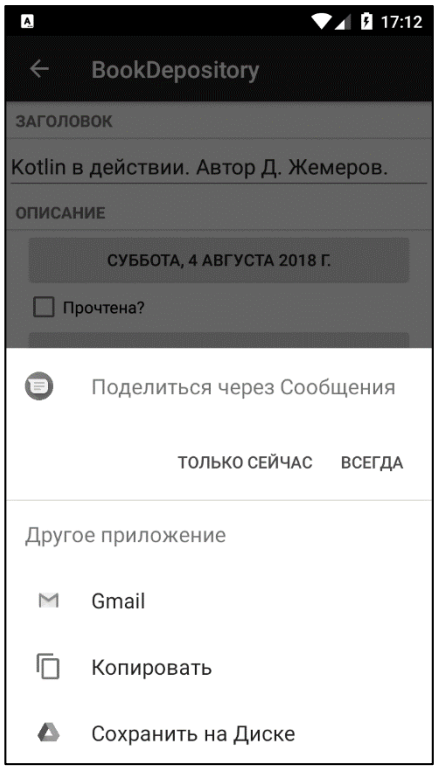

Рисунок 14.4 – Приложения, готовые отправить отчет

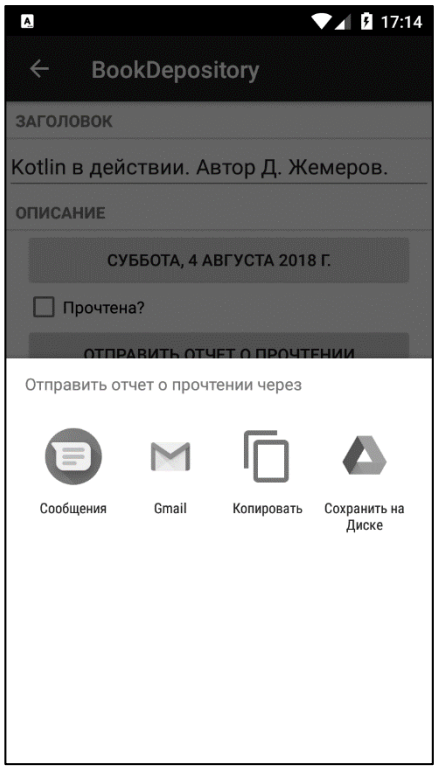

Рисунок 14.5 – Отправка текста с выбором активности

Самостоятельные задания.

Задание. ShareCompat

В библиотеку поддержки Android входит класс ShareCompat с внутренним классом IntentBuilder. ShareCompat.IntentBuilder упрощает построение интентов точно такого вида, какой был использован для кнопки отчета. Необходимо в слушателе OnClickListener кнопки mReportButton использовать для построения интента класс ShareCompat.IntentBuilder (вместо того, чтобы строить его вручную).