№7 Хранение информации, SQLite, AsyncTask, адаптеры

Задание

1. Клонируйте проект №6_5 , измените:
    1) Способ работы с данными. Создайте базу данных для хранения объетов приложения. Создайте свой класс помощника (…DatabaseHelper) для управления базой и обеспечения доступа к ней из Activity.
    2) Обеспечить сохранение, вставку и обновление данных о товарах, выборку (функции поиска и сортировки) и удаление. Используйте курсоры. Напишите дополнительную функцию – выбора избранных товаров (checkbox). Для этого добавьте дополнительный столбец в таблицу (выбан/не выбран) и запрос вывода избранных товаров.
    3) Для вывода списка товаров используйте SimpleCursorAdapter или CursorAdapter связанный с ListView.
    4) Клонируйте проект (для сравнения оставьте старый код).Вынести код работы с базой данных из основного потока событий и выполните его в отдельном потоке в фоновом режиме потоке с использованием AsyncTask (асинхронное выполнение операций). Оцените скорость работы приложения.

Вопросы при защите:

1. Где хранится база данных вашего приложения? Как создать базу данных? Что будет если не задать ей имя?
2. Объясните назначение объектов SQLiteOpenHelper, SQLiteDatabase, Cursor.
3. Как создать таблицу базы данных?
4. Что такое класс – контракт и для чего он создается?
5. Какие методы нужно определить в классе-помощнике SQL Helper?
6. Зачем указывается номер версии базы данных? На что он влияет?
7. Приведите пример выполнение команды SQL (например обновления таблицы).
8. Как вставить данные в таблицу? Что такое ContentValues?
9. Какой формат написания query? Какие дополнительные операторы можно использовать в запросах?
10. Как удалить данные?
11. В чем разница между методами getReadableDatabase() и getWritableDatabase()?
12. Как прочитать данные из курсора?
13. Что такоеCursorAdapter?
14. Что должно выполняться в основном потоке, а что в фоновом?
15. Для чего и как используют класс AsyncTask? Охарактеризуйте его методы: назначение, выполнение, доступ к UI, тип параметра и порядок вызова.

16. Что означают типы в generic классе AsyncTask?
17. Как запустить асинхронную задачу?
18. Как отменить выполнение асинхронной задачи?
19. Что такое ContentProvider?
20. Что такое Firebase?

Теория

Механизм работы с базами данных в Android позволяет хранить и обрабатывать структурированную информацию. Любое приложение может создавать свои собственные базы данных, над которыми оно будет иметь полный контроль.

В Android используется библиотека SQLite, представляющую из себя реляционную СУБД, обладающую следующими характерными особенностями: свободно распространяемая (open source), поддерживающая стандартный язык запросов и транзакции, легковесная, одноуровневая (встраиваемая), отказоустойчивая.

Курсоры (Cursor) и ContentValues

Запросы к СУБД возвращают объекты типа Cursor. Для экономии ресурсов используется подход, когда при извлечении данных не возвращаются копии их значений из СУБД, а создается Cursor, обеспечивающий навигацию и доступ к запрошенному набору исходных данных. Методы объекта Cursor предоставляют различные возможности навигации, назначение которых, как правило, понятно из названия:

- *moveToFirst*
- *moveToNext*
- *moveToPrevious*
- *getCount*
- *getColumnIndexOrThrow*
- *getColumnName*
- *getColumnNames*
- *moveToPosition*
- *getPosition*

При добавлении данных в таблицы СУБД применяются объекты класса ContentValues. Каждый такой объект содержит данные одной строки в таблице и, по сути, является ассоциативным массивом с именами столбцов и соответствующими значениями.

Работа с СУБД SQLite

При создании приложений, использующих СУБД, во многих случаях удобно применять инструменты, называющиеся ORM (Object-Relationship

Mapping), отображающие данные из одной или нескольких таблиц на объекты используемого языка программирования. Кроме того, ORM позволяют абстрагироваться от конкретной реализации и структуры таблиц и берут на себя обязанности по взаимодействию с СУБД. К сожалению, в силу ограниченности ресурсов мобильной платформы ORM в настоящий момент в Android практически не применяется. Тем не менее, разумным подходом при разработке приложения будет инкапсуляция всех взаимодействий с СУБД в одном классе, методы которого будут предоставлять необходимые услуги остальным компонентам приложения.

Хорошей практикой является создание вспомогательного класса, берущего на себя работу с СУБД. Данный класс обычно инкапсулирует взаимодействия с базой данных, предоставляя интуитивно понятный строго типизированный способ удаления, добавления и изменения объектов. Такой Адаптер базы данных также должен обрабатывать запросы к БД и переопределять методы для открытия, закрытия и создания базы данных. Его также обычно используют как удобное место для хранения статических констант, относящихся к базе данных, таких, например, как имена таблиц и полей. Ниже показан пример каркаса для реализации подобного Адаптера:

```
public class SampleDBAdapter { private static final String
      DATABASE_NAME = "SampleDatabase.db"; private static final
      String DATABASE_TABLE = "SampleTable"; private static final
      int DATABASE_VERSION = 1;

      // Имя поля индекса для
      public static final String KEY_ID = "_id";

      // Название и номер п/п (индекс) каждого поля
      public static final String KEY_NAME = "name";
      public static final int NAME_COLUMN = 1;
      // Для каждого поля опишите константы аналогичным образом...

      // SQL-запрос для создания БД
      private static final String DATABASE_CREATE = "create table "
                  + DATABASE_TABLE + " (" + KEY_ID
                  + " integer primary key autoincrement, " + KEY_NAME
                  + " text not null);";

      // Переменная для хранения объекта БД
      private SQLiteDatabase db;
      // Контекст приложения для
      private final Context context;

      // Экземпляр вспомогательного класса для открытия и обновления БД
      private myDbHelper dbHelper;

      // Конструктор
      public SampleDBAdapter(Context _context) {
            context = _context;
            dbHelper = new myDbHelper(context, DATABASE_NAME, null,
                        DATABASE_VERSION);
      }

      // «Открывашка» БД
      public SampleDBAdapter open() throws
            SQLException { try { db =
            dbHelper.getWritableDatabase();
```

```
        } catch (SQLiteException e) { db =
              dbHelper.getReadableDatabase();
        }
        return this;
}

// Метод для закрытия БД
public void close() {
db.close();
}

// Метод для добавления данных, возвращает индекс
// свежевставленного объекта
public long insertEntry(SampleObject _SampleObject) {
        // Здесь создается объект ContentValues, содержащий
        // нужные поля и  производится вставка
        return index;
}

// Метод для удаления строки таблицы по индексу
public boolean removeEntry(long _rowIndex) {
return db.delete(DATABASE_TABLE, KEY_ID + "=" +
              _rowIndex, null) > 0;


}

// Метод для получения всех данных.
// Возвращает курсор, который можно использовать для // привязки к
адаптерам типа SimpleCursorAdapter public Cursor getAllEntries() {
return db.query(DATABASE_TABLE, new String[] { KEY_ID, KEY_NAME },
null, null, null, null, null);
}

// Возвращает экземпляр объекта по индексу
public SampleObject getEntry(long _rowIndex) {
        // Получите курсор, указывающий на нужные данные из БД
        // и создайте новый объект, используя этими данными
        // Если ничего не найдено, верните null
        return objectInstance;
}

// Изменяет объект по индексу
// Увы, это не ORM :(
public boolean updateEntry(long _rowIndex, SampleObject _SampleObject) {
        // Создайте объект ContentValues на основе свойств SampleObject
        // и используйте его для обновления строки в таблице
        return true; // Если удалось обновить, иначе false :)
}

// Вспомогательный класс для открытия и обновления БД
private static class myDbHelper extends SQLiteOpenHelper
{ public myDbHelper(Context context, String name,
CursorFactory factory, int version) { super(context,
name, factory, version);
        }

        // Вызывается при необходимости создания БД
        @Override
        public void onCreate(SQLiteDatabase _db) {
              _db.execSQL(DATABASE_CREATE);
        }
```

```
            // Вызывается для обновления БД, когда текущая версия БД
            // в приложении новее, чем у БД на диске
            @Override
            public void onUpgrade(SQLiteDatabase _db, int _oldVersion,
                        int _newVersion) {
                  // Выдача сообщения в журнал, полезно при отладке
                  Log.w("TaskDBAdapter", "Upgrading from version " +
                  _oldVersion
                              + " to " + _newVersion
                               + ", which will destroy all old data");

                  // Обновляем БД до новой версии.
                  // В простейшем случае убиваем старую БД
                  // и заново создаем новую.
                  // В реальной жизни стоит подумать о пользователях
                  // вашего приложения и их реакцию на потерю старых
                  данных. _db.execSQL("DROP TABLE IF EXISTS " +
                  DATABASE_TABLE); onCreate(_db);
            }
      }
}
```

Особенности работы с БД в Android

При работе с базами данных в Android следует избегать хранения BLOB'ов с таблицах изза резкого падения эффективности работы.

Как показано в примере адаптера БД, для каждой таблицы рекомендуется создавать автоикрементное поле _id, которое будет уникальным индексом для строк. Если же планируется делегировать доступ к БД с помощью контент-провайдеров, такое поле является обязательным.

Выполнение запросов для доступа к данным

Для повышения эффективности использования ресурсов мобильной платформы запросы к БД возвращают объект типа Cursor, используемый в дальнейшем для навигации и получения значений полей. Выполнение запросов осуществляется с помощью метода query экземпляра БД, параметры которого позволяют гибко управлять критериями выборки:

```
// Получить поля индекса, а также первое и третье из таблицы,
// без дубликатов
String[] result_columns = new String[] {KEY_ID, KEY_COL1, KEY_COL3};
Cursor allRows = myDatabase.query(true, DATABASE_TABLE, result_columns,
null, null, null, null, null, null);

// Получить все поля для строк, где третье поле равно требуемому
// значению,
// результат отсортировать по пятому полю
String where = KEY_COL3 + "=" + requiredValue;
String order = KEY_COL5;
Cursor myResult = myDatabase.query(DATABASE_TABLE, null, where,
null, null, null, order);
```

Доступ к результатам с помощью курсора

Для получения результатов запроса необходимо установить курсор на нужную строку с помощью методов вида moveToМестоположение,

перечисленных выше. После этого используются типизированные методы getТип, получающие в качестве параметра индекс (номер) поля в строке. Как правило, эти значения являются статическими константами адаптера БД:

```
String columnValue = myResult.getString(columnIndex);
```

В примере ниже показано, как можно просуммировать все поля (типа float) из результатов выполнения запроса (и получить среднюю сумму счета):

```
int KEY_AMOUNT = 4;
Cursor myAccounts = myDatabase.query("my_bank_accounts", null,
null, null, null, null, null); float totalAmount = 0f;
// Убеждаемся, что курсор
непустой if
(myAccounts.moveToFirst()) {
      // Проходимся по каждой строке do { float amount =
      myAccounts.getFloat(KEY_AMOUNT); totalAmount +=
      amount;
      } while(myAccounts.moveToNext());
} float averageAmount = totalAmount /

myAccounts.getCount();
```

Изменение данных в БД

В классе SQLiteDatabase, содержащем методы для работы с БД, имеются методы insert, update и delete, которые инкапсулируют операторы SQL, необходимые для выполнения соответствующих действий. Кроме того, метод execSQL позволяет выполнить любой допустимый код SQL(если вы, например, захотите увеличить долю ручного труда при создании приложения). Следует помнить, что при наличии активных курсоров после любого изменения данных следует вызывать метод refreshQuery для всех курсоров, которые имеют отношение к изменяемым данным (таблицам).

Вставка строк

Метод insert хочет получать (кроме других параметров) объект ContentValues, содержащий значения полей вставляемой строки и возвращает значение индекса:

```
ContentValues newRow = new ContentValues();

// Выполним для каждого нужного поля в строке
newRow.put(COLUMN_NAME, columnValue);
db.insert(DATABASE_TABLE, null, newRow);
```

Обновление строк

Также используется ContentValues, содержащий подлежащие изменению поля, а для указания, какие именно строки нужно изменить, используется параметр where, имеющий стандартный для SQL вид:

```
ContentValues updatedValues = new ContentValues();

// Повторяем для всех нужных полей
updatedValues.put(COLUMN_NAME, newValue);

// Указываем условие
```

```
String where = COLUMN_NAME + "=" + "'Бармаглот'";

// Обновляем db.update(DATABASE_TABLE,
updatedValues, where, null);
```

Метод update возвращает количество измененных строк. Если в качестве параметра where передать null, будут изменены все строки таблицы.

Удаление строк

Выполняет похожим на update образом:

```
db.delete(DATABASE_TABLE, KEY_ID + "=" + rowId, null);
```

Использование SimpleCursorAdapter

SimpleCursorAdapter позволяет привязать курсор к ListView. используя описание разметки для отображения строк и полей. Для однозначного определения того, в каких элементах разметки какие поля, получаемые через курсор, следует отображать, используются два массива: строковый с именами полей строк, и целочисленный с идентификаторами элементов разметки:

```
Cursor cursor = [ . . . запрос к БД . . . ];

String[] fromColumns = new String[] {KEY_NAME, KEY_NUMBER}; int[]
toLayoutIDs = new int[] { R.id.nameTextView, R.id.numberTextView};

SimpleCursorAdapter myAdapter;
myAdapter = new SimpleCursorAdapter(this, R.layout.item_layout, cursor,
            fromColumns, toLayoutIDs);

myListView.setAdapter(myAdapter);
```

Контент-провайдеры

Контент-провайдеры предоставляют интерфейс для публикации и потребления структурированных наборов данных, основанный на URI с использованием простой схемы content://. Их использование позволяет отделить код приложения от данных, делая программу менее чувствительной к изменениям в источниках данных.

Для взаимодействия с контент-провайдером используется уникальный URI, который обычно формируется следующим образом:

content://<домен-разработчика-наоборот>.provider.<имя-приложения>/<путь-к-даным>

Классы, реализующие контент-провайдеры, чаще всего имеют статическую строковую константу CONTENT_URI, которая используется для обращения к данному контентпровайдеру.

Контент-провайдеры являются единственным способом доступа к данным других приложений и используются для получения результатов запросов, обновления, добавления и удаления данных. Если у приложения есть нужные полномочия, оно может запрашивать и модифицировать соответствующие данные, принадлежащие другому приложению, в том числе данные стандартных БД Android. В общем случае, контент-провайдеры следует создавать только тогда, когда требуется предоставить другим

приложениям доступ к данным вашего приложения. В остальных случаях рекомендуется использовать СУБД (SQLite). Тем не менее, иногда контент-провайдеры используются внутри одного приложения для поиска и обработки специфических запросов к данным.