ТЕХНОТРЕК

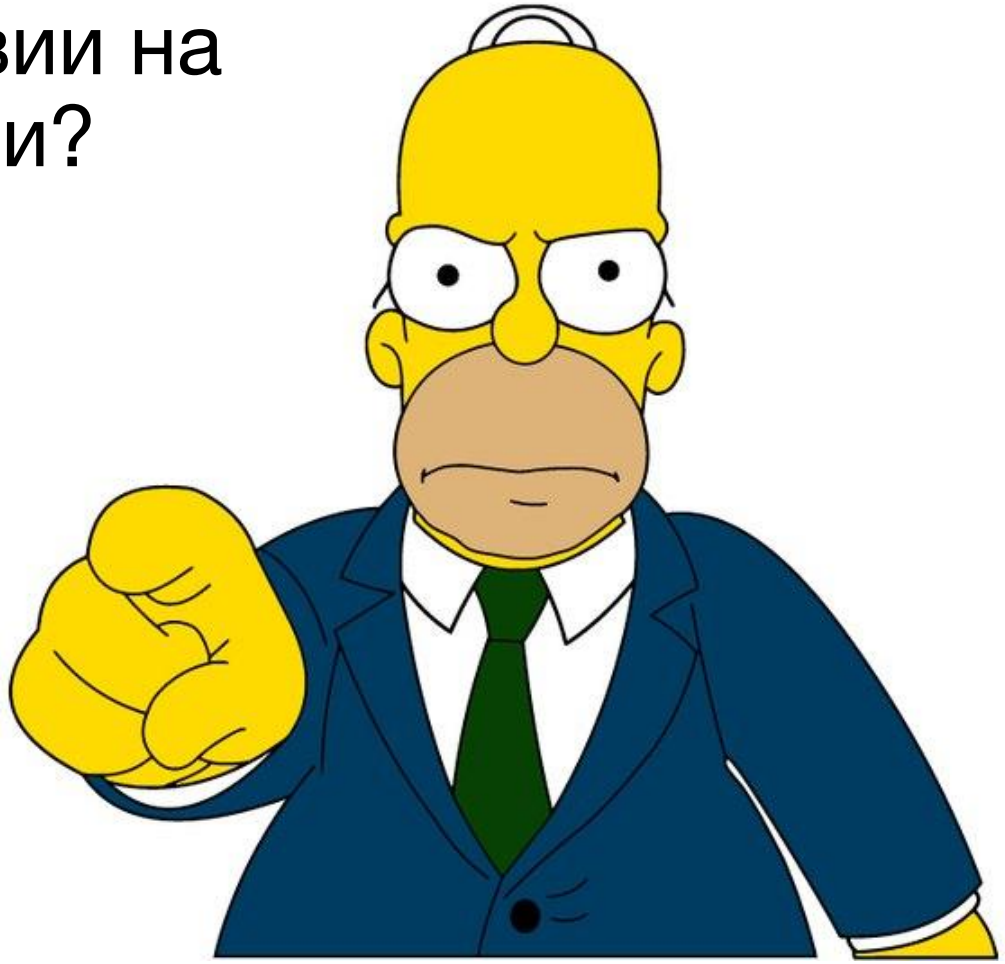
Занятие №6

Разработка приложений на Android

Кирилл Филимонов
Юрий Береза

Напоминание

А ты отметился о присутствии на занятии?



Agenda

1. Работа с файлами
2. LRU Cache
3. Shared Preferences
4. SQLite
5. Content Providers

Работаем с файлами

- Assets
- Internal Storage
- External Storage

Assets

Плюсы
- Сразу идут с приложением
- Всегда есть в наличии

Минусы
- Сразу идут с приложением
- Нельзя модифицировать

```
public Bitmap loadBitmapFromAssets(Context context, String filename) {
    AssetManager assetManager = context.getAssets();
    try {
        InputStream is = assetManager.open( "images/"+filename);
        BitmapFactory.Options opt =  new BitmapFactory.Options();
        return BitmapFactory.decodeStream(is, null, opt);
    }
    catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}
```

Internal Storage

Плюсы
- Всегда доступны (почти)

Минусы
- Необходимо туда загрузить
- Занимают место

```
context.getCacheDir(); context.getFilesDir(); contex.getFileStreamPath()
```

```
public Bitmap loadBitmapFromCacheDir(Context context, String filename) {
    File cacheDir = context.getCacheDir();
    File file = new File(cacheDir, filename);
    if (file.exists()) {
        try {
            InputStream is = new FileInputStream(file);
            BitmapFactory.Options opt = new BitmapFactory.Options();
            return BitmapFactory.decodeStream(is, null, opt);
        }
        catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
    return null;
}
```

External Storage

Плюсы
- Доступно больше места

Минусы
- Необходимо туда загрузить
- Не всегда доступны

`context.getExternalCacheDir(); context.getExternalFilesDir();`
`Environment.getExternalStorageDirectory();`

```
public Bitmap loadBitmapFromCacheDir(Context context, String filename) {
    String state = Environment.getExternalStorageState();
    if (state.equals(Environment.MEDIA MOUNTED)) {
        File cacheDir = Environment.getExternalStorageDirectory();
        File file = new File(cacheDir, filename); //ROOT DIR!!!
        if (file.exists()) {
            // DO WHAT YOU NEED
        }
    }
    return null;
}
```

LruCache

- **LRU** (least recently used) — это алгоритм, при котором вытесняются значения, которые дольше всего не запрашивались.
- Мы резервируем какой-то объем памяти для хранения картинок и отдает на откуп этому классу управление кешем.
- Прежде чем загружать всегда проверяем кеш.

```
private LruCache<String, Bitmap> mMemoryCache;

@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    // Get max available VM memory, exceeding this amount will throw an
    // OutOfMemory exception. Stored in kilobytes as LruCache takes an
    // int in its constructor.
    final int maxMemory = (int) (Runtime.getRuntime().maxMemory() / 1024);

    // Use 1/8th of the available memory for this memory cache.
    final int cacheSize = maxMemory / 8;

    mMemoryCache = new LruCache<String, Bitmap>(cacheSize) {
        @Override
        protected int sizeOf(String key, Bitmap bitmap) {
            // The cache size will be measured in kilobytes rather than
            // number of items.
            return bitmap.getByteCount() / 1024;
        }
    };
    ...
}

public void addBitmapToMemoryCache(String key, Bitmap bitmap) {
    if (getBitmapFromMemCache(key) == null) {
        mMemoryCache.put(key, bitmap);
    }
}

public Bitmap getBitmapFromMemCache(String key) {
    return mMemoryCache.get(key);
}
```

Shared Preferences

- Хранение данных внутри приложения
- Что хранить в Shared Preferences
- А что не хранить
- В чем удобство использования Shared Preferences

Shared Preferences

Context.getSharedPreferences и Activity.getPreferences

```
SharedPreferences sh = getSharedPreferences("settings", Context.MODE_PRIVATE);
SharedPreferences.Editor e = sh.edit();
e.putInt("value", 1);
e.apply();

SharedPreferences sh = getSharedPreferences("settings", Context.MODE_PRIVATE);
int value = sh.getInt("value");
```

SQLite

- SQLite – встраиваемая реляционная база данных
- Все хранит в одном файле
- Этот файл можно править различными инструментами
- Позволяет читать из множества потоков, но лочится на запись
- Есть некоторые проблемы с локализацией
- Надо помнить об эффективности запросов

Работа с базой в Android

- ContentValues – для вставки и обновления данных
- Cursor (SQLiteCursor) – для обработки выборки
- SQLiteOpenHelper – для создания, открытия и обновления версий базы
  - `abstract void onCreate(SQLiteDatabase db)` – вызывается при создании базы когда ее еще нету
  - `abstract void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)` – вызывается если в конструкторе была передана версия больше чем текущая
- SQLiteDatabase – класс работы с базой
  - query– выбор из базы SELECT
  - delete – удаление DELETE
  - update – обновление UPDATE
  - insert – вставка нового значения INSERT
  - execSql – выполнить произвольный запрос
  - replace – аналог запроса INSERT OR REPLACE

SQL Injection

- Внедрение кода в ваш запрос
- Не доверяйте пользовательским данным
- Не вводите их напрямую в запросы
- Активно используйте значения аргументов фильтра в функциях query, delete, update, execSQL, rawQuery

Неправильно:

```
Db.execSQL("DELETE FROM some_table WHERE name = '" +
userInputString + "' ");
```

Правильно:

```
Db.execSQL("DELETE FROM some_table WHERE name = ?",
new String[]{ userInputString });
```

query

```
Cursor query(
      String table,
      String[] columns,
      String selection,
      String[] selectionArgs,
      String groupBy,
      String having,
      String orderBy,
      String limit
)
```

- **table** — имя таблицы, к которой передается запрос;
- **columns** — список имен возвращаемых полей. При передаче null возвращаются все столбцы;
- **selection** — параметр, формирующий выражение WHERE
- **selectionArgs** — значения аргументов фильтра;
- **groupBy** - параметр, формирующий выражение GROUP BY
- **having** — параметр, формирующий выражение HAVING
- **sortOrder** — параметр, форматирующий выражение ORDER BY
- **limit** - параметр ограничивающий количесво строк в выдаче

insert

```
long insert(String table, String nullColumnHack, ContentValues
values)
long insertOrThrow(String table, String nullColumnHack,
ContentValues values)
long insertWithOnConflict(String table, String nullColumnHack,
ContentValues initialValues, int conflictAlgorithm)
```

- **table** — имя таблицы, в которую будет вставлена запись;
- **nullColumnHack** — в базе данных SQLite не разрешается вставлять полностью пустую строку, и если строка, полученная от клиента контент-провайдера, будет пустой, то только этому столбцу явно будет назначено значение null;
- **values** — карта отображений (класс Map и его наследники), которая содержит пары ключ-значение. Ключи в карте должны быть названиями столбцов таблицы, значения — вставляемыми данными.
- **conflictAlgorithm** – как обрабатывать конфликты (replace, rollback, ignore none)
- возвращает идентификатор _iD вставленной строки или -1в случае ошибки.

delete и update

```
int update (
     String table,
     ContentValues values,
     String whereClause,
     String[] whereArgs
)
int delete (
     String table,
     String whereClause,
     String[] whereArgs
)
```

- **table** — имя таблицы, к которой передается запрос;
- **whereClause** — параметр, формирующий выражение WHERE
- **whereArgs**— значения аргументов фильтра;
- **values** — значения
- возвращают количество измененных или удаленных строк

execSQL и rawQuery

```
void execSQL(String sql)
void execSQL(String sql, Object[] bindArgs)
Cursor rawQuery(String sql, String[] selectionArgs)
```

- **sql** — запрос
- **bindArgs** — значения аргументов фильтра;
- execSQL ничего не возвращает
- execSQL не рекомендуется для SELECT, UPDATE, DELETE

Как заполнять базу

Из других данных

1. Создать xml или json с данными, положить в assets
2. Заполнять ее инсертами в onCreate

Из файла базы

3. Создать базу вне приложения заполнить ее
4. Положить файл базы в assets и затем скопировать ее по значению
5. Заархивировать и положить в raw, затем скопировать как файл на sdcard

18

Особенности работы

- **database is locked** – возникает при многопоточной записи в базу.
- **database is closed** – может возникнуть при работе с базой из разных частей программы, например, Activity и Service.
- **corrupted database** – возникает, если файл базы данных был испорчен либо пользователем, либо при неожиданном прерывании записи в базу
- **низкая производительность** при работе с базой данных

Database is locked

- блокировки в SQLite выполнены на уровне файла.
- читать базу может много потоков, а писать только один
- если вы пишете из двух потоков одного соединения, то один поток будет ждать, пока закончит писать другой.
- если вы пишете из двух потоков разных соединений, то произойдет ошибка – приложение вылетит с *SQLiteDatabaseLockedException*.
- приложение всегда должно иметь только один экземпляр *SQLiteOpenHelper*(именно открытого соединения), иначе в любой момент может возникнуть *SQLiteDatabaseLockedException*.
- *SQLiteOpenHelper* имеет 2 метода предоставляющих доступ *SQLiteOpenHelper.getReadableDatabase()==SQLiteOpenHelper.getWritableDatabase()*
- внутри класса *SQLiteDatabase* есть собственные блокировки – переменная mLock.
- поскольку на чтение и запись экземпляр *SQLiteDatabase* один, то чтение данных тоже блокируется.

Database is closed

- Возникает когда с базой работает и активити и сервисы
- При каждом обращении к базе проверять,- закрыта база или нет, и если закрыта, то переоткрывать её заново.
- Принудительно добавить фиктивную ссылку на базу и держать её пока база используется
- Использовать ContentProvider для доступа к базе. Причем желательно использовать именно один провайдер – это легко реализовать, поскольку ему можно добавить неограниченное количество Uri

Corrupted database

- Причина – испортился файл базы
  - Проблемы с питанием
  - Падение приложения
  - Глюки устройства
- Новую базу устройство создаст само на onCreate
- Можно вызвать VACUUM – база будет пересоздана

Оптимизация работы с базой

- Не пишите запросы которые возвращают больше 1000 строк или мегабайта данных
- Не используйте смещение в LIMIT
- Если нужно создать таблицу из существующей используйте INSERT AS SELECT
- Очищайте память после больших запросов вызовом *SQLiteDatabase.releaseMemory()*
- Используйте индексы
- Не используйте like
- В условиях ставьте легкие или индексированные запросы сначала
- Для интернациональных раскладок некорректно обрабатываются заглавные буквы в запросе.
- Правильно использовать JOIN
- Избегать фрагментации

Контент провайдер

- Что такое и для чего
- Какие есть в системе
- Как использовать
- Как создавать самому
- Какие есть подводные камни
- Как избежать кражи данных

Content Providers

- Механизм обмена данными между процессами
- Позволяет использовать данные системы
- Процесс может пользоваться контентом другого процесса
- Процесс может предоставлять свои данные другим процессам
- Позволяет гибко настраивать доступ

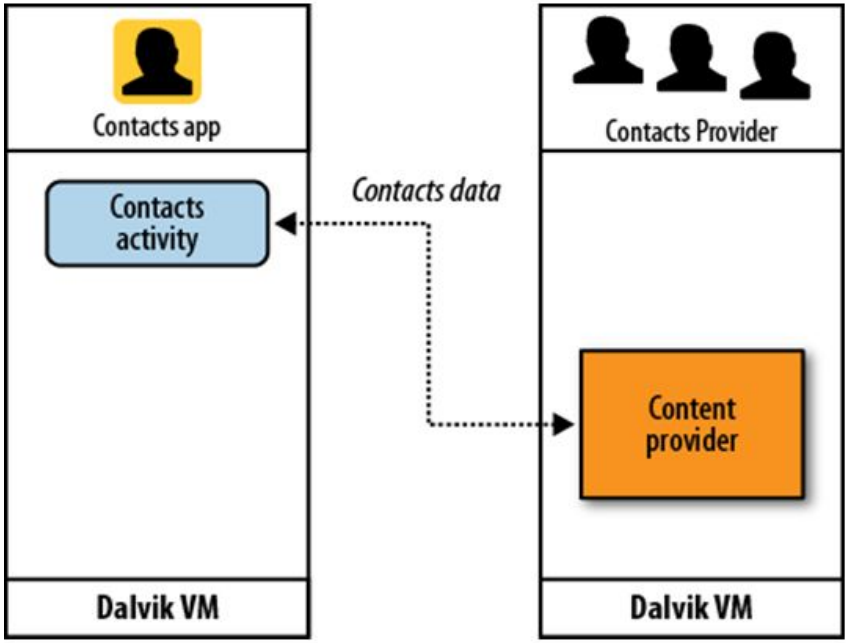
Как он работает

- Обращение происходит по специальному REST-подобному URI вида content://providerName/table
- Все данные организованны как таблицы
- Запросы похожи на SQL запросы
- Вы делаете запрос, получаете данные и работаете с ним

word	app id	frequency	locale	_ID
mapreduce	user1	100	en_US	1
precompiler	user14	200	fr_FR	2
applet	user2	225	fr_CA	3
const	user1	255	pt_BR	4
int	user5	100	en_UK	5

Какие есть в системе провайдеры

- Browser
- CallLog
- Contacts (People, Phones, Photos, Groups)
- MediaStore (Audio,Images, Video)
- Settings



Как использовать

- Запрашиваем разрешение в манифесте
- Формируем нужный нам URI может быть пути или конкретных данных
- На чтение
  - Делаем запрос в ответ получаем курсор
  - Читаем последовательно
- Вставка данных
  - Делаем insert
- Обновление данных
  - Делаем update
- Удаление
  - Делаем delete

Разрешения

Разрешения нужны для системных провайдеров

Пишутся в манифесте в разделе <user-permission…>

```
<uses-permission android:name="android.permission.READ_USER_DICTIONARY">
<uses-permission android:name="android.permission.READ_CONTACTS" />
```

Нужно для того чтобы информировать пользователя , устанавливающего
приложение, о том что его данные могут быть прочитаны или изменены

Формируем правильный URI

- URI контент провайдера представляет собой уникальные в системе идентификатор
- Включает имя провайдера
- Имя таблицы в провайдере

Пример content://user_dictionary/words

content:// - всегда присутствует, говорит о том что это URI контента

user_dictionary - идентификатор провайдера

words - имя таблицы

- Может быть ссылкой на данные

Пример content://user_dictionary/words/23

23 - элемент с _ID = 23

Запрашиваем данные

Используем функцию

public final Cursor query (Uri uri, String[] projection, String selection,
String[] selectionArgs,
String sortOrder)

```
// Queries the user dictionary and returns results
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI,   // The content URI of the words table
    mProjection,                        // The columns to return for each row
    mSelectionClause                    // Selection criteria
    mSelectionArgs,                     // Selection criteria
    mSortOrder);                        // The sort order for the returned rows
```

Query()	SELECT
Uri	FROM table_name
projection	Col, col, col, ….
selection	WHERE col= value
selectionArgs	Альтернатива ? при использовании placeholder
sortOrder	ORDER BY col, col, …

Код запроса к провайдеру

```
String[] mSelectionArgs = {""};
mSearchString = mSearchWord.getText().toString();
// Надо проверять ввод
if (TextUtils.isEmpty(mSearchString)) {
    mSelectionClause = null;
    mSelectionArgs[0] = "";
} else {
    mSelectionClause = UserDictionary.Words.WORD + " = ?";
    mSelectionArgs[0] = mSearchString;
}
mCursor = getContentResolver().query(…);
if (null == mCursor) {
    // Ошибка! Ее надо обработать
} else if (mCursor.getCount() < 1) {
    // Пустой курсор, ничего не нашли по запросу
} else {
    // Обрабатываем полученный данные
}
```

Безопасность запросов

Никогда не используйте напрямую пользовательские данные в запросе

```
String mSelectionClause =  "var = " + mUserInput;
```

Вместо этого используйте поле для аументов

```
String mSelectionClause =  "var = ?”;
String[] selectionArgs = {""};
selectionArgs[0] = mUserInput;
```

Отображение данных

1. Через курсор адаптер с помощью контейнера (ListView например)

```
String[] mWordListColumns ={
   UserDictionary.Words.WORD, e
   UserDictionary.Words.LOCALE
};
int[] mWordListItems = { R.id.dictWord, R.id.locale};
mCursorAdapter = new SimpleCursorAdapter(getApplicationContext(),
   R.layout.wordlistrow, mCursor, mWordListColumns, mWordListItems, 0);
mWordList.setAdapter(mCursorAdapter);
```

2. Самостоятельно как пожелаете

```
if (mCursor != null) {
      while (mCursor.moveToNext()) {
      newWord = mCursor.getString(index);
   }
}
```

Вставка данных

Используем ContentResolver.insert()

```
Uri mNewUri;

ContentValues mNewValues = new ContentValues();

mNewValues.put(UserDictionary.Words.APP_ID, "example.user");
mNewValues.put(UserDictionary.Words.LOCALE, "en_US");
mNewValues.put(UserDictionary.Words.WORD, "insert");
mNewValues.put(UserDictionary.Words.FREQUENCY, "100");

mNewUri = getContentResolver().insert(
   UserDictionary.Word.CONTENT_URI,
   mNewValues
);
```

Обновление данных

Используем ContentResolver.update()

```
ContentValues mUpdateValues = new ContentValues();

String mSelectionClause = UserDictionary.Words.LOCALE +  "LIKE ?";
String[] mSelectionArgs = {"en_%"};
int mRowsUpdated = 0;
mUpdateValues.putNull(UserDictionary.Words.LOCALE);

mRowsUpdated = getContentResolver().update(
   UserDictionary.Words.CONTENT_URI,
   mUpdateValues
   mSelectionClause
   mSelectionArgs
);
```

Удаление данных

Используем ContentResolver.delete()

```
String mSelectionClause = UserDictionary.Words.APP_ID + " LIKE ?";
String[] mSelectionArgs = {"user"};

int mRowsDeleted = 0;

mRowsDeleted = getContentResolver().delete(
    UserDictionary.Words.CONTENT_URI,
    mSelectionClause
    mSelectionArgs
);
```

Создание собственного провайдера

- Как понять, что нужен именно он
- Понять как данные будут хранится
- Выбрать имя для провайдера и таблиц
- Реализовать собственно провайдер
- Реализовать дополнительные классы

А нужен ли контент провайдер

- Вы собираетесь отдавать данные в другое приложение
- Вы собираетесь отдавать сложные данные
- Вы собираетесь отдавать потенциально много данных
- Вы собираетесь отдавать данные или файлы по запросу от другого приложения

Вам не нужен контент провайдер если вы собираетесь работать с SQLite только в вашем приложении в достаточно простой конфигурации

Данные

Файлы:

Таблицы:

CategoryID	ParentID	Title	SortOrder
1	0	Electronics	9835
2	1	Mobile Phones	10000
3	1	DVD Systems	10100
4	2	Sony Ericsson	10000
5	2	Nokia	10100
6	2	Motorola	10200
7	2	Samsung	10300
8	0	Apparel	100
9	8	John Players	10000
10	8	Women Sarees	10100
11	9	Shirts	10000
12	9	Pants	10100
13	10	Banarasi Sarees	10000
14	10	Kurta Salwar	10100
16	0	Beverages	9975
17	0	Computer and Accessories	55555
18	0	Baby Items	10400.2
19	0	Health and Beauty	10400.4
20	0	Jewelry	10193.85

URI

- Должно быть уникальным в системе
- Рекомендуется использовать applicationID, например com.example.<appname>.provider
- Имена таблиц лучше выбирать исходя из данных за который они отвечают
- Путь URI может содержать много сегментов
- Каждый уровень URI не означает таблицу

content://com.example.app.provider/table1
content://com.example.app.provider/table2/dataset1
content://com.example.app.provider/table2/dataset2

UriMatcher

Нужен для поиска Uri по шаблону

Поддерживает следующие шаблоны

***:** Совпадение строки с любыми валидными символами любой длинны

#: Совпадает со строкой цифр любой длинны.

```
private static final UriMatcher sUriMatcher;
   sUriMatcher.addURI("com.example.app.provider", "table3", 1);
   sUriMatcher.addURI("com.example.app.provider", "table3/#", 2);
public Cursor query(
   switch (sUriMatcher.match(uri)) {
            case 1:
               break;
            case 2:
```

Манифест

android:authorities - идентификатор провайдера

android:name - класс обработчик

android:process – принцип работы похож на service, начало “:” заставит создать новый именованный процесс

android:readPermission

android:writePermission – строка , разрешения нужные для процесса который будет читать или писать в этот провайдер

```
<provider android:authorities="list"
        android:enabled=["true" | "false"]
        android:exported=["true" | "false"]
        android:grantUriPermissions=["true" | "false"]
        android:icon="drawable resource"
        android:initOrder="integer"
        android:label="string resource"
        android:multiprocess=["true" | "false"]
        android:name="string"
        android:permission="string"
        android:process="string"
        android:readPermission="string"
        android:syncable=["true" | "false"]
        android:writePermission="string" >
    . . .
</provider>
```

Класс реализующий функционал

Наследуемся от ContentProvider

```
public class MyContactsProvider extends ContentProvider {
    public boolean onCreate() {}
    public Cursor query(
            Uri uri,
            String[] projection,
            String selection,
            String[] selectionArgs,
            String sortOrder) {}
    public Uri insert(Uri uri, ContentValues values) {}
    public int delete(Uri uri, String selection, String[] selectionArgs)
    {}
    public String getType(Uri uri) {}
    public int update(Uri uri, ContentValues values, String selection,
          String[] selectionArgs) {}

}
```

MIME type

Для данных таблиц и прочего - getType(URI)

- Начинается с vnd
- Для одной строки android.cursor.item/
- Для более чем одной строки android.cursor.dir/
- Путь специфичный для вашего приложения

vnd.android.cursor.**dir**/vnd.com.example.provider.table1

vnd.android.cursor.**item**/vnd.com.example.provider.table1

Для файлов - используется getStreamType(URI)

- Обычные MIME типы файлов

{ "image/jpeg", "image/png", "image/gif"}

Проблемы большинства реализаций

- Несанкционированный доступ к персональным данным пользователя и чувствительной информации.
  - Пермишены
- Уязвимости типа SQL injection в провайдерах, работающих с базами данных.
  - Конструкции типа “val = “ + userData

ТЕХНОТРЕК

Спасибо за внимание!

Кирилл Филимонов Kirill.Filimonov@gmail.com

Юрий Береза – ybereza@gmail.com