

4. ЗБЕРЕЖЕННЯ ТА ОБРОБКА ДАНИХ В МОБІЛЬНИХ ДОДАТКАХ

Постачальники контенту (Content providers) керують доступом до структурованого набору даних. Вони інкапсулюють дані і надають механізми забезпечення їх безпеки. Постачальники контенту – це стандартний інтерфейс для об'єднання даних в одному процесі з кодом, який виконується в іншому процесі [38–40].

Коли вам потрібен доступ до даних у постачальнику контенту, використовуйте об'єкт `ContentResolver` в інтерфейсі `Context` вашого програмного додатка, щоб підключитися до постачальника як клієнт. Об'єкт `ContentResolver` взаємодіє з об'єктом постачальника, який є екземпляром класу, який реалізує об'єкт `ContentProvider`. Об'єкт постачальника отримує від клієнтів запити даних, виконує запрошені дії і повертає результати.

Вам не потрібно розробляти власний постачальник, якщо ви не плануєте надавати доступ до своїх даних іншим програмам. Однак вам знадобиться власний постачальник для надання налаштованих пошукових підказок у вашому власному додатку. Вам також знадобиться власний постачальник, якщо необхідно копіювати і вставляти складні дані або файли зі свого програмного додатка в інші.

До складу системи Android входять постачальники контенту, які керують такими даними, як аудіо, відео, зображення і особиста контактна інформація. Деякі з постачальників вказані в довідковій документації для пакета `android.provider`. Працювати з цими постачальниками може будь-який додаток Android (проте з деякими обмеженнями) [38–40].

4.1. Основні відомості про постачальника контенту

Постачальник контенту управляє доступом до центрального сховища даних. Постачальник є компонентом програми Android, який часто має власний користувацький інтерфейс для роботи з даними. Однак постачальники контенту призначені в першу чергу для використання іншими програмними додатками, які отримують доступ до постачальника за допомогою клієнтського об'єкта постачальника. Разом постачальники і клієнти постачальників забезпечують узгоджений, стандартний інтерфейс доступу до даних, який також обробляє взаємодію між процесами і забезпечує захищений доступ до даних.

4.1.1. Огляд

Постачальник контенту надає дані зовнішнім програмним додаткам у вигляді однієї або декількох таблиць, аналогічних таблицям в реляційній базі да-

них. Рядок являє собою екземпляр деякого типу даних, що збираються постачальником, а кожен стовпець в цьому рядку – це окремий елемент даних, зібраних для екземпляра.

Прикладом вбудованого постачальника в платформі Android може служити користувацький словник, в якому зберігаються дані про написання нестандартних слів, доданих користувачем. У табл. 4.1 показано, як дані можуть виглядати в цій таблиці постачальника.

Таблиця 4.1 – Приклад таблиці словника

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

У кожному рядку цієї таблиці наведено екземпляр слова, яке відсутнє в стандартному словнику. У кожному її стовпці містяться деякі дані для слова, наприклад дані про мову, в якій це слово було вперше використано. Заголовки стовпців – це імена стовпців, які зберігаються в постачальнику. Щоб дізнатися мову рядка, необхідно звернутися до стовпця `locale`. У цьому постачальнику стовпець `_ID` виступає в ролі «основного ключа», який постачальник автоматично зберігає.

Примітка. У постачальника необов'язково повинен бути основний ключ, а також йому необов'язково використовувати `_ID` як ім'я стовпця основного ключа, якщо такий є. Однак, якщо необхідно прив'язати дані з постачальника до класу `ListView`, один із стовпців повинен іменуватися `_ID`.

4.1.2. Доступ до постачальника

Для доступу програми до даних з постачальника контенту використовується клієнтський об'єкт `ContentResolver`. В цьому об'єкті є методи, які викликають ідентичні методи в об'єкті постачальника, який є екземпляром одного з конкретних підкласів класу `ContentProvider`. У методах `ContentResolver` наведено основні функції CRUD (абревіатура `create`, `retrieve`, `update`, `delete` [створення, отримання, оновлення та видалення]) постійного сховища.

Об'єкт `ContentResolver` у процесі клієнтської програми і об'єкт `ContentProvider` у програмі, яка володіє постачальником, автоматично об-

роблюють взаємодію між процесами. Об'єкт `ContentProvider` також виступає в ролі рівня абстракції між репозиторієм даних і зовнішнім поданням даних у вигляді таблиць.

Примітка. Для доступу до постачальника ваша програма зазвичай має запросити певні дозволи в своєму файлі маніфесту.

Наприклад, щоб отримати з постачальника користувальницького словника список слів і мов, на яких вони наведені, викличте метод `ContentResolver.query()`. У свою чергу, метод `query()` викликає метод `ContentProvider.query()`, визначений постачальником словника. Далі у прикладі коду показаний виклик методу `ContentResolver.query()`:

```
// Запит до словника користувача і отримання результату
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI,    // URI слів таблиці
    mProjection,                        // Колонки для кожного
    рядка                               // Критерій вибірки
    mSelectionClause                    // Критерій вибірки
    mSelectionArgs,                    // Порядок для виводу
    mSortOrder);
рядків
```

У табл. 4.2 зазначено відповідність аргументів для методу запиту `query(Uri,projection,selection,selectionArgs,sortOrder)` SQL-команди `SELECT`.

Таблиця 4.2 – Порівняння методу `query()` і SQL-запиту

Аргумент методу <code>query()</code>	Параметр/ключове слово <code>SELECT</code>	Примітки
<code>Uri</code>	<code>FROM table_name</code>	<code>Uri</code> відповідає таблиці <code>table_name</code> у постачальнику
<code>projection</code>	<code>col,col,col,...</code>	<code>projection</code> – це масив стовпців, які необхідно включити у кожний отриманий рядок
<code>selection</code>	<code>WHERE col = value</code>	<code>selection</code> задає критерії для вибору рядків
<code>selectionArgs</code>	(Точний еквівалент відсутній. Заповнювачі ? замінюються аргументами вибору)	
<code>sortOrder</code>	<code>ORDER BY col,col,...</code>	<code>sortOrder</code> задає порядок відображення рядків в об'єкті <code>Cursor</code>

4.1.3. URI контенту

URI контенту – це URI, який визначає дані в постачальнику. URI контенту можуть включати символічне ім'я всього постачальника (його **центр**) і ім'я, яке вказує на таблицю (**шлях**). При виклику клієнтського методу для доступу до таблиці в постачальнику URI контенту цієї таблиці виступає в ролі одного з аргументів цього методу.

Константа `CONTENT_URI` в попередніх рядках коду містить URI контенту таблиці `words` в користувацькому словнику. Об'єкт `ContentResolver` аналізує центр URI і використовує його для «вирішення» постачальника шляхом порівняння центру з системною таблицею відомих постачальників. `ContentResolver` може відправити аргументи запиту до відповідного постачальника.

`ContentProvider` використовує частину URI контенту, в якій вказано шлях, для вибору таблиці для доступу. У постачальника зазвичай є шлях для кожної наданої їм таблиці.

У попередніх рядках коду повний URI для таблиці `words` виглядає таким чином:

```
content://user_dictionary/words
```

Рядок `user_dictionary` – це центр постачальника, а рядок `words` – це шлях до таблиці. Рядок `content://` (**схема**) присутній завжди; він визначає, що це URI контенту.

Багато постачальників надають доступ до одного рядка в таблиці шляхом додавання ідентифікатора в кінець URI. Наприклад, щоб витягти зі словника рядок, в стовпці `_ID` якого задано 4, можна скористатися таким URI контенту:

```
Uri singleUri =  
ContentUris.withAppendedId(UserDictionary.Words.CONTENT_URI,4);
```

Ідентифікатори часто використовуються у випадку, коли ви витягли набір рядків і оновлюєте або видаляєте один з них.

Примітка. У класах `Uri` і `Uri.Builder` є методи для зручного створення правильно оформлених об'єктів URI із рядків. `ContentUris` містить методи для зручного додавання ідентифікаторів до URI. У прикладі коду, наведеного вище, для додавання ідентифікатора до URI контенту `UserDictionary` використовується метод `withAppendedId()`.

4.1.4. Отримання даних від постачальника

У цьому підрозділі розглядається порядок отримання даних від постачальника на прикладі постачальника користувацького словника.

Для повної ясності в прикладах коду, наведених в цьому розділі, методи `ContentResolver.query()` викликаються в потоці призначеного для користувача інтерфейсу. У реальному коді запити слід виконувати асинхронно в окремому потоці. Одним із способів реалізувати це є використання класу `CursorLoader`.

Щоб отримати дані з постачальника, виконайте такі основні дії.

1. Запитайте у постачальника дозвіл на читання.
2. Визначте код, який відповідає за відправку запиту постачальнику.

Запит дозволу на читання. Щоб ваша програма могла отримувати дані від постачальника, програмі слід отримати від постачальника дозвіл на читання. Цей дозвіл неможливо отримати під час виконання; замість цього вам необхідно вказати в маніфесті програми, що вам потрібен такий дозвіл. Для цього скористайтесь елементом `<uses-permission>` і вкажіть точну назву дозволу, зазначену постачальником. Вказавши цей елемент в маніфесті, ви тим самим запитуєте необхідний дозвіл для вашої програми. Коли користувачі встановлюють вашу програму, вони отримують дозвіл на цей запит.

Щоб дізнатися точну назву дозволу на читання в використовуваному постачальнику, а також назви інших використовуваних в ньому дозволів на читання, зверніться до документації постачальника.

Постачальник користувацького словника задає дозвіл `android.permission.READ_USER_DICTIONARY` в своєму файлі маніфесту, тому програмі, якій потрібно виконати читання даних з постачальника, необхідно запросити саме цей дозвіл.

Створення запиту. Наступним етапом отримання даних від постачальника є створення запиту. У наступному фрагменті коду задаються деякі змінні для доступу до постачальника словника:

```
// projection – це масив стовпців, які необхідно включити у кож-
ний отриманий рядок
String[] mProjection =
{
    UserDictionary.Words._ID,      // Константа для стовпця _ID
    UserDictionary.Words.WORD,     // Константа для стовпця word
    UserDictionary.Words.LOCALE    // Константа для стовпця locale
};
```

```
// Визначає рядок, що містить умову вибору
String mSelectionClause = null;

// Визначає масив, що містить аргументи для вибірки
String[] mSelectionArgs = {""};
```

У наступному фрагменті коду демонструється порядок використання методу `ContentResolver.query()` (прикладом є постачальник словника): клієнтський запит постачальника аналогічний SQL-запиту. У ньому міститься набір стовпців, які повертаються, набір критеріїв вибірки і порядок сортування.

Набір стовпців, які повинен повернути запит, називається **проекцією** (змінна `mProjection`).

Вираз, який задає рядки для отримання, складається з пропозиції вибору і аргументів вибору. Пропозиція вибору являє собою поєднання логічних виразів, імен стовпців і значень (змінна `mSelectionClause`). Якщо замість значення вказати параметр, що підставляється, метод запиту витягує значення з масиву аргументів вибору (змінна `mSelectionArgs`).

У наступному фрагменті коду, якщо користувач не вказав слово, то для пропозиції вибору задається значення `null`, а запит повертає всі слова, наявні в постачальника. Якщо користувач вказав слово, то для пропозиції вибору задається значення `UserDictionary.Words.WORD + "=?"`, а для першого елемента в масиві аргументів вибору – введене користувачем слово.

```
* Завдається одноелементний масив типу String, що містить аргументи
вибірки.
*/
String[] mSelectionArgs = {""};

// Отримуємо слово з UI
mSearchString = mSearchWord.getText().toString();

// Не забудьте вставити тут код для перевірки даних на дійсність.

// Дії, якщо слово є порожнім рядком
if (TextUtils.isEmpty(mSearchString)) {
    // Встановлення значення null для змінної вибірки, таким чином ре-
    зультатом будуть усі можливі слова
    mSelectionClause = null;
    mSelectionArgs[0] = "";
} else {
    // Встановлює умови вибору, що відповідає слову, що ввів користу-
    вач.
    mSelectionClause = UserDictionary.Words.WORD + " = ?";
    // Переміщення рядка, що ввів користувач до аргументів вибірки.
    mSelectionArgs[0] = mSearchString;
}
```

```
// Запит до таблиці та повернення об'єкту типу Cursor
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI, // URI слів таблиці
    mProjection,                      // Колонки для кожного рядка
    mSelectionClause,                 // або null, або слово, що ввів
    користувач,                      // або пустий рядок, або рядок,
    mSelectionArgs,                   // або пустий рядок, або рядок,
    що ввів користувач,              // або пустий рядок, або рядок,
    mSortOrder);                     // Порядок для виводу рядків

// Деякі провайдери повертають нульове значення, якщо відбувається по-
милка, інші кидають виняткову ситуацію
if (null == mCursor) {
    /*
     * Вставте тут код для обробки виняткової ситуації. Не використо-
     вуйте курсор! Ви можете викликати android.util.Log.e, щоб записати ін-
     формацію про цю виняткову ситуацію.
     */
    // Якщо курсор пустий, то провайдер не знаходить співпадінь
} else if (mCursor.getCount() < 1) {

    /*
     * Вставте тут код, щоб повідомити користувача, що пошук не викона-
     вся успішно. Це не обов'язково.
     */

} else {
    // Вставте код тут, щоб обробити результати
}
}
```

Цей запит аналогічний SQL-інструкції:

```
SELECT _ID, word, locale FROM words WHERE word = <userinput>
ORDER BY word ASC;
```

У цій інструкції SQL замість констант класу-контракту використовуються фактичні імена стовпців.

Захист від введення шкідливого коду. Якщо дані, якими управляє поста-чальник контенту, знаходяться в базі даних SQL, то включення в необроблені інструкції SQL зовнішніх ненадійних даних може призвести до атаки шляхом вставлення коду SQL.

Розглянемо наступну умову на вибірку:

```
// Складаємо умову на вибірку шляхом складання змінної, що ввів
користувач та назви стовпця
String mSelectionClause = "var = " + mUserInput;
```

При використанні цієї умови, ви дозволяєте користувачеві зв'язати вашу інструкцію SQL з шкідливим кодом SQL. Наприклад, користувач може ввести

nothing; DROP TABLE *; для `mUserInput`, що призведе до створення наступної умови вибору: `var = nothing; DROP TABLE *;`. Оскільки умова вибору виконується в потоці як інструкція SQL, це може призвести до того, що постачальник видалить всі таблиці у відповідній базі даних SQLite (якщо тільки постачальник не налаштований на відстеження спроб вставити шкідливий код SQL).

Щоб уникнути цього, скористайтеся умовою вибору «?» виступає як параметр, що підставляється, або масивом аргументів вибору. Після цього введення даних користувачем буде пов'язане безпосередньо із запитом і не буде інтерпретуватися як частина інструкції SQL. Оскільки в цьому випадку запит, що ввів користувач, не розглядається як код SQL, то в нього не вдасться впровадити шкідливий код SQL. Замість об'єднання, яке варто включити в користувацький ввід даних, використовуйте наступну умову вибору:

```
// Створює пункт вибору зі змінним параметром
String mSelectionClause = "var = ?";
```

Налаштуйте масив аргументів вибору таким чином:

```
// Defines an array to contain the selection arguments
String[] selectionArgs = {""};
```

Вкажіть значення для масиву аргументів вибору:

```
// Sets the selection argument to the user's input
selectionArgs[0] = mUserInput;
```

Умова вибору, в якій «?» використовується як параметр, що підставляється, і масив аргументів вибору є кращим способом указання вибору, навіть якщо постачальник не використовує базу даних SQL.

Відображення результатів запиту. Клієнтський метод `ContentResolver.query()` завжди повертає об'єкт `Cursor`, що містить стовпці, зазначені в проекції запиту для рядків, які відповідають критеріям вибірки в запиті. Об'єкт `Cursor` надає прямий доступ на читання рядків і стовпців, що містяться в ньому. За допомогою методів `Cursor` можна виконати ітерацію по рядках в результатах, визначити тип даних для кожного стовпця, отримати дані з шпальти, а також перевірити інші властивості результатів. Деякі реалізації об'єкта `Cursor` автоматично оновлюють об'єкт при зміні даних в постачальнику або запускають виконання методів в об'єкті-спостерігачі при зміні об'єкта `Cursor`, або виконують і те, і інше.

Примітка. Постачальник може обмежити доступ до стовпців на основі характеру об'єкта, що виконує запит. Наприклад, постачальник контактів обмежує доступ адаптерів синхронізації до деяких стовпців, тому він не повертає їх у дію або службу.

Якщо рядки, які відповідають критеріям вибірки, відсутні, постачальник повертає об'єкт `Cursor`, в якому для методу `Cursor.getCount()` вказано значення «0» (порожній об'єкт `Cursor`).

При виникненні внутрішньої помилки результати запиту залежать від певного постачальника. Постачальник може повернути `null` або видати `Exception`.

Оскільки `Cursor` являє собою «список» рядків, то найкращим способом відобразити вміст об'єкта `Cursor` буде зв'язати його з `ListView` за допомогою `SimpleCursorAdapter`.

Наступний фрагмент коду є продовженням попереднього фрагмента. Він створює об'єкт `SimpleCursorAdapter`, що містить об'єкт `Cursor`, який був отриманий в запиті, а потім визначає цей об'єкт як адаптер для `ListView`:

```
// Визначає список стовпців для отримання з курсора і завантаження у рядок виводу
String[] mWordListColumns =
{
    UserDictionary.Words.WORD,    // містить назву стовпця word
    UserDictionary.Words.LOCALE  // містить назву стовпця locale
};

// Визначає список View IDs, які будуть отримувати стовпці курсора для кожного рядка
int[] mWordListItems = { R.id.dictWord, R.id.locale};

// Створює новий SimpleCursorAdapter
mCursorAdapter = new SimpleCursorAdapter(
    getApplicationContext(),           // Об'єкт Context програми
    R.layout.wordlistrow,               // Шаблон в XML для одного рядка в ListView
    mCursor,                           // Результат запиту
    mWordListColumns,                  // Масив імен колонок в курсорі
    mWordListItems,                    // Масив view IDs
    0);                                // Прапори (зазвичай нема в них необхідності)

// Встановлюється адаптер для the ListView
mWordList.setAdapter(mCursorAdapter);
```

Примітка. Щоб повернути `ListView` з об'єктом `Cursor`, об'єкт `cursor` повинен містити стовпець з ім'ям `_ID`. Тому показаний раніше запит отримує стовпець `_ID` для таблиці `words`, навіть якщо `ListView` не відображає її. Це обмеження також пояснює, чому в кожній таблиці постачальника є стовпець `_ID`.

Отримання даних з результатів запиту. Замість того, щоб просто переглянути результати запиту, можна використовувати їх для виконання інших завдань. Наприклад, можна отримати написання слів з словника користувача, а потім виконати їх пошук в інших постачальників. Для цього виконайте ітерацію по рядках в об'єкті `Cursor`:

```
// Визначаємо індекс стовпця, що має назву "word"
int index = mCursor.getColumnIndex(UserDictionary.Words.WORD);

/*
 * Виконується, якщо курсор є валідним. Провайдер словника кори-
 * стувача повертає нульове значення, якщо виникла внутрішня помил-
 * ка.
 * Інші провайдери можуть повернути помилку замість повернення ну-
 * ля.
 */

if (mCursor != null) {
    /*
     * Перехід до наступного рядка в курсорі. Перед першим
     * переміщенням у курсорі «Індекс рядка» -1, і якщо ви намагаєтеся
     * отримати дані в цій позиції, то ви отримуєте виняткову ситуацію
     * (помилку)
     */
    while (mCursor.moveToNext()) {

        // Отримуємо значення стовпця.
        newWord = mCursor.getString(index);

        // Вставте тут код для обробки отриманого слова
        ...

        // кінець циклу    }
    } else {

        // Вставте тут код для звіту про помилку, якщо курсор є null
        // або провайдер викинув виняткову ситуацію.
    }
}
```

Реалізації об'єкта `Cursor` містять кілька методів `get` для отримання об'єкта різних типів даних. Наприклад, в наступному фрагменті коду

використовується метод `getString()`. У них також є метод `getType()`, який повертає значення, яке вказує на тип даних стовпця.

4.1.5. Дозволи постачальника контенту

Додаток постачальника може задавати дозволи, які потрібні іншим додаткам для доступу до даних у постачальника. Такі дозволи гарантують, що користувач знає, до яких даних додаток буде намагатися отримати доступ. На основі вимог постачальника інші програми запитують дозволи, які потрібні їм для доступу до постачальника. Кінцеві користувачі бачать запитані дозволи при установленні програми.

Якщо додаток постачальника не задає ніяких дозволів, інші додатки не отримують доступу до даних постачальника. Однак компонентам додатка постачальника завжди надано повний доступ на читання і запис, незалежно від заданих дозволів.

Як вже було зазначено раніше, для отримання даних з постачальника користувацького словника потрібен дозвіл `android.permission.READ_USER_DICTIONARY`. У постачальника передбачений окремий дозвіл `android.permission.WRITE_USER_DICTIONARY` для вставки, оновлення або видалення даних. Щоб отримати необхідні дозволи для доступу до інтернету, програма запитує їх з допомогою елемента `<uses-permission>` у файлі маніфесту. При установленні менеджером пакетів Android програми користувачеві необхідно затвердити всі дозволи, що необхідні додатку. У разі затвердження всіх дозволів менеджер пакетів продовжує установлення; якщо ж користувач відхиляє їх, менеджер пакетів скасовує установлення.

Для запиту доступу на читання даних у постачальника користувацького словника використовується наступний елемент `<uses-permission>`:

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

4.1.6. Вставка, оновлення та видалення даних

Подібно до того, як отримуються дані від постачальника, також можна використовувати можливості взаємодії між клієнтом постачальника і об'єктом `ContentProvider` постачальника для зміни даних. Можна викликати метод об'єкта `ContentResolver`, вказавши аргументи, які були передані у відповідний метод об'єкта `ContentProvider`. Постачальник і клієнт постачальника автоматично обробляють взаємодію між процесами і забезпечують безпеку.

Вставка даних. Для вставки даних у постачальник викличте метод `ContentResolver.insert()`. Цей метод вставляє новий рядок у постачальник і повертає URI контенту для цього рядка. У наступному фрагменті коду демонструється порядок вставки нового слова у постачальник користувацького словника:

```
// Визначаємо новий об'єкт Uri, який отримує результат вставки
Uri mNewUri;
// Визначаємо об'єкт, який містить нові значення для вставки
ContentValues mNewValues = new ContentValues();
/*
 * Встановлюємо значення кожного стовпця і вставляємо слово.
 */
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,    // URI контенту словника
    користувача
    mNewValues                          // значення для вставки
);
```

Дані для нового рядка надходять в один об'єкт `ContentValues`, який аналогічний об'єкту `cursor` з одним рядком. Стовпці в цьому об'єкті не обов'язково повинні містити дані такого ж типу, і якщо ви не збираєтеся вказувати значення, то можна задати для стовпця значення `null` за допомогою методу `ContentValues.putNull()`.

Код в наведеному фрагменті не додає стовпець `_ID`, оскільки цей стовпець зберігається автоматично. Постачальник присвоює унікальне значення `_ID` кожному доданому рядку. Зазвичай постачальники використовують це значення як основний ключ таблиці.

URI контенту, повернений в елементі `newUri`, служить для ідентифікації нового доданого рядка в наступному форматі:

```
content://user_dictionary/words/<id_value>
```

`<id_value>` – це вміст стовпця `_ID` для нового рядка. Більшість постачальників автоматично визначають цю форму URI контенту, а потім виконують необхідну операцію з необхідним рядком.

Щоб отримати значення `_ID` з повернутого об'єкта `Uri`, викличте метод `ContentUris.parseId()`.

Оновлення даних. Щоб оновити рядок, використовуйте об'єкт `ContentValues` з оновленими значеннями (так само, як ви це робите при вставці) та критеріями вибірки (так само, як і з запитом). Використовуваний вами клієнтський метод називається `ContentResolver.update()`. Вам не потрібно додавати значення в об'єкт `ContentValues` для оновлюваних стовпців. Щоб очистити вміст стовпця, встановіть значення `null`.

Наступний фрагмент коду служить для зміни мови у всіх рядках, де у якості мови зазначено `en`, на значення `null`. Обчислене значення – це кількість рядків, які були оновлені.

Також слід перевірити введені користувачем дані при виклику методу `ContentResolver.update()`.

```
// Визначаємо об'єкт, який містить оновлені значення
ContentValues mUpdateValues = new ContentValues();

// Визначаємо критерії вибору для рядків, які ви хочете оновити
String mSelectionClause = UserDictionary.Words.LOCALE + "LIKE
?";
String[] mSelectionArgs = {"en_%"};

// Визначаємо змінну, щоб утримувати кількість оновлених рядків
int mRowsUpdated = 0;

/*
 * Встановлюємо оновлене значення і оновлюємо обрані слова.
 */
mUpdateValues.putNull(UserDictionary.Words.LOCALE);

mRowsUpdated = getContentResolver().update(
    UserDictionary.Words.CONTENT_URI,    // URI контенту словника
    користувача
    mUpdateValues                        // колонка для оновлення
    даних
    mSelectionClause                    // стовпець, по якому ви-
    бирають
    mSelectionArgs                      // значення для порівнян-
    ня
);
```

Видалення даних. Видалення даних аналогічно одержанню даних рядка: необхідно вказати критерії вибірки для рядків, які потрібно видалити, після чого клієнтський метод поверне кількість видалених рядків. Нижче наведено фрагмент коду для видалення рядків з ідентифікатором `appid user`. Метод повертає кількість видалених рядків.

Також слід перевірити дані, введені користувачем при виклику методу `ContentResolver.delete()`.

```
// Визначаємо критерії вибору для рядків, які ви хочете видалити
String mSelectionClause = UserDictionary.Words.APP_ID + "
LIKE ?";
String[] mSelectionArgs = {"user"};

// Визначаємо змінну, щоб отримувати кількість видалених рядків
int mRowsDeleted = 0;

...

// Видаляє слова, які відповідають критеріям вибору
mRowsDeleted = getContentResolver().delete(
    UserDictionary.Words.CONTENT_URI,    // URI контенту словника
    користувача
    mSelectionClause                      // стовпець, по якому ви-
    бирають
    mSelectionArgs                       // значення для
    порівняння
);
```

4.1.7. Типи даних постачальників

Постачальники контенту можуть надавати різні типи даних. Постачальник користувацького словника надає тільки текст, але він також може надавати такі формати:

- ціле число;
- довге ціле число (long);
- число з плаваючою комою;
- довге число з плаваючою комою (double).

Іншим типом даних, що пропонуються постачальником, є великий двійковий об'єкт (BLOB), реалізований як 64-розрядний масив. Щоб переглянути доступні типи даних, зверніться до методів `get` класу `Cursor`.

Тип даних для кожного стовпця в постачальнику зазвичай вказується у документації до постачальника. Типи даних для постачальника користувацького словника вказані в довідковій документації для класу-контракту `UserDictionary.Words`. Також визначити тип даних можна шляхом виклику методу `Cursor.getType()`.

Постачальники також зберігають інформацію про тип даних MIME для кожного обумовленого ними URI контенту. Цю інформацію можна використувати для визначення того, чи може ваш додаток обробляти пропоновані постачальником дані, а також для вибору типу обробки на основі типу MIME. Інформація про тип MIME зазвичай потрібна при роботі з постачальником, який

містить складні структури даних або файли. Наприклад, у таблиці `ContactsContract.Data` у постачальника контактів використовуються типи MIME для відзначення типу даних контакту, які зберігаються у кожному рядку. Щоб отримати тип MIME, відповідний до URI контенту, викличте метод `ContentResolver.getType()`.

Синтаксис стандартних і настроюваних типів MIME описаний у довідці за типами MIME.

4.1.8. Альтернативні форми доступу до постачальника

При розробці програми варто враховувати три альтернативні форми доступу до постачальника:

1. *Пакетний доступ*: можна створити пакет викликів доступу з використанням методів у класі `ContentProviderOperation`, а потім застосувати їх за допомогою методу `ContentResolver.applyBatch()`.

2. *Асинхронні запити*: запити слід виконувати в окремому потоці. Одним із способів реалізувати це є використання об'єкта `CursorLoader`.

3. *Доступ до даних за допомогою намірів*: незважаючи на те, що намір неможливо відправити безпосередньо в постачальник, можна відправити запит на додаток постачальника, в якому зазвичай є більше можливостей для зміни даних постачальника.

Пакетний доступ і зміна з допомогою намірів описані в наступних розділах.

Пакетний доступ. Пакетний доступ до постачальника корисно використовувати у випадках, коли необхідно вставити велику кількість рядків, або для вставки рядків в кілька таблиць в рамках одного виклику методу, а також у загальних випадках для виконання ряду операцій на кордонах процесів у вигляді транзакції (атомарної операції).

Для доступу до постачальника в «пакетному режимі» необхідно створити масив об'єктів `ContentProviderOperation`, а потім відправити їх в постачальник контенту за допомогою методу `ContentResolver.applyBatch()`. В цей метод необхідно передати *центр* постачальника контенту, а не певний URI контенту. Це дозволить кожному об'єкту `ContentProviderOperation` в масиві взаємодіяти з різними таблицями. Метод `ContentResolver.applyBatch()` повертає масив результатів.

В описі класу-контракту `ContactsContract.RawContacts` також наведено фрагмент коду, в якому демонструється вставка в пакетному режимі. У вихід-

ному файлі `ContactAdder.java` прикладу програми «Диспетчер контактів» є приклад пакетного доступу.

Доступ до даних за допомогою намірів. Наміри дозволяють в обхід отримувати доступ до свого контенту. Користувачам можна дозволити доступ до даних у постачальника, навіть у тому випадку, якщо у додатка відсутні дозволи на доступ, або шляхом отримання результуючого наміру від програми, у якої є необхідні дозволи, або шляхом активації програми, у якої є дозвіл і яка дозволяє користувачеві працювати з ним.

Отримання доступу з тимчасовими дозволами. Можна отримати доступ до даних у постачальника контенту, навіть тоді, коли у вас немає необхідних дозволів на доступ, шляхом надсилання наміру в додаток, у якого є такі дозволи, і отримання результуючого наміру, який містить дозволи URI. Ці дозволи для певного URI контенту діють доти, поки не буде завершена операція, яка отримала їх. Додаток, у якого є безстрокові дозволи, надає тимчасові дозволи шляхом встановлення відповідного прапорця в результуючому намірі:

- дозвіл на читання: `FLAG_GRANT_READ_URI_PERMISSION`;
- дозвіл на запис: `FLAG_GRANT_WRITE_URI_PERMISSION`.

Примітка. Ці прапорці не надають доступ на читання або запис постачальнику, центр якого вказаний в URI контенту. Доступ надається тільки самому URI.

Постачальник визначає дозволи URI для URI контенту у своєму маніфесті за допомогою атрибута `android:grantUriPermission` елемента `<provider>`, а також з допомогою дочірнього елемента `<grant-uri-permission>` елемента `<provider>`.

Наприклад, можна отримати дані про контакт з постачальника контактів, навіть якщо у вас немає дозволу `READ_CONTACTS`. Можливо, це потрібно реалізувати в додатку, який відправляє електронні привітання контакту в день його народження. Замість запиту `READ_CONTACTS`, коли ви отримуєте доступ до всіх контактів користувача та інформації про них, можна надати користувачеві можливість вказати, які контакти використовуються вашим додатком. Для цього скористайтеся наведеними нижче процесами.

1. Ваша програма відправляє намір, що містить дію `ACTION_PICK` і тип `MIME_CONTENT_ITEM_TYPE` контактів, використовуючи для цього метод `startActivityForResult()`.

2. Оскільки цей намір відповідає умовам відбору намірів для операції вибору програми «Контакти», ця операція переходить на передній план.

3. В операції вибору користувач вибирає контакт для оновлення. Коли це відбувається, операція вибору викликає метод `setResult(resultcode, intent)` для створення наміру, який буде передано назад у вашу програму. Намір містить URI вмісту вибраного користувачем контакту, а також прапори `FLAG_GRANT_READ_URI_PERMISSION` додаткових даних. Ці прапори надають вашому додатку дозвіл URI на читання даних контакту, на який вказує URI контенту. Потім операція вибору викликає метод `finish()`, щоб повернути управління вашим додатком.

4. Ваша операція повертається на передній план, а система викликає метод `onActivityResult()` операції. Цей метод отримує результуючий намір, створений операцією вибору у програмі «Контакти».

5. За допомогою URI контенту з результуючого наміру можна виконати читання даних контакту з постачальника контактів, навіть якщо ви не просили у постачальника постійний доступ на читання в своєму маніфесті. Можна отримати інформацію про день народження контакту або відомості про його адресу електронної пошти, а потім відправити контакту електронне привітання.

Використання іншого додатку. Простий спосіб дозволити користувачеві змінювати дані, на доступ до яких у вас немає доступу – це активувати додаток, у якого є такі дозволи, а потім надати користувачеві можливість виконувати необхідні дії в цьому додатку.

Наприклад, додаток «Календар» приймає намір `ACTION_INSERT`, за допомогою якого можна активувати призначений для користувача інтерфейс програми для вставки. Можна передати в цьому намірі додаткові дані, які додаток використовує для заповнення полів в інтерфейсі. Оскільки синтаксис повторюваних подій досить складний, то події слід вставляти в постачальник календаря шляхом активації програми «Календар» за допомогою дії `ACTION_INSERT` і подальшого надання користувачеві можливості самому вставити подію в цей додаток.

Відображення даних за допомогою допоміжного додатку. Якщо вашій програмі не надано дозволу, то, як і раніше, можна скористатися наміром для відображення даних в іншому додатку. Наприклад, додаток «Календар» приймає намір `ACTION_VIEW`, який дозволяє відобразити певну дату або подію. Завдяки цьому інформацію календаря можна відображати без необхідності створювати власний користувацький інтерфейс.

Додаток, в який надсилається намір, не обов'язково має бути пов'язаний з постачальником. Наприклад, у постачальника контактів можна створити форму

контакту, а потім відправити намір ACTION_VIEW, що містить URI контенту для зображення контакту в засобі перегляду зображень.

4.1.9. Класи-контракти

Клас-контракт визначає константи, які забезпечують для додатків можливість працювати з URI контенту, іменами стовпців, операціями наміру та іншими функціями постачальника вмісту. Класи-контракти не включені у постачальника; розробнику постачальника слід визначити їх і зробити їх доступними для інших розробників. Багато хто з постачальників, що включені в платформу Android, містять відповідні класи-контракти в пакеті android.provider.

Наприклад, у постачальника користувацького календаря є клас-контракт `UserDictionary`, що містить константи URI контенту та імен стовпців. URI вмісту таблиці `words` визначено у константі `UserDictionary.Words.CONTENT_URI`. У класі `UserDictionary.Words` також є константи імен стовпців, які використовуються у фрагментах коду прикладу програми. Наприклад, проекцію запиту можна визначити таким чином:

```
String[] mProjection =
{
    UserDictionary.Words._ID,
    UserDictionary.Words.WORD,
    UserDictionary.Words.LOCALE
};
```

Іншим класом-контрактом є клас `ContactsContract` для постачальника контактів. Один з його підкласів, `ContactsContract.Intents.Insert`, являє собою клас-контракт, який містить константи для намірів і їх даних.

4.1.10. Довідка за типами MIME

Постачальники контенту можуть повертати як стандартні типи мультимедіа MIME, так і рядки з налаштованим типом MIME, або обидва ці типи.

Типи MIME мають такий формат:

```
type/subtype
```

Наприклад, добре відомий тип MIME `text/html` має тип `text` і підтип `html`. Якщо постачальник повертає цей тип URI, це означає, що рядок запиту, в якому використовується цей URI, поверне текст з тегами HTML.

Рядки з налаштованим типом MIME, які також називаються типами MIME постачальника, мають більш складні значення типів і підтипів. Значення типу завжди таке:

```
vnd.android.cursor.dir
```

для декількох рядків, або

```
vnd.android.cursor.item
```

для одного рядка.

Підтип залежить від постачальника. Вбудовані постачальники Android зазвичай містять простий підтип. Наприклад, коли програма «Контакти» створює рядок для номера телефону, вона задає наступний тип MIME в цьому рядку:

```
vnd.android.cursor.item/phone_v2
```

Зверніть увагу, що значення підтипу просто `phone_v2`.

Розробники постачальників можуть створювати свої власні шаблони підтипів на основі центру і назв таблиць постачальника. Наприклад, розглянемо постачальник, який містить розклад руху поїздів. Центром постачальника є `com.example.trains`, в якому містяться таблиці `Line1`, `Line2` і `Line3`. У відповідь на таку URI вміст:

```
content://com.example.trains/Line1
```

для таблиці `Line1` постачальник повертає такий тип MIME:

```
vnd.android.cursor.dir/vnd.example.line1
```

У відповідь на такий URI вміст:

```
content://com.example.trains/Line2/5
```

для рядка 5 таблиці `Line2` постачальник повертає наступний тип MIME:

```
vnd.android.cursor.item/vnd.example.line2
```

У більшості постачальників контенту визначено константи класу-контракту для використовуваних у них типів MIME. Наприклад, клас-контракт `ContactsContract.RawContacts` постачальника контактів визначає константу `CONTENT_ITEM_TYPE` для типу MIME одного рядка необробленого контакту.

4.2. Створення постачальника контенту

Постачальник контенту керує доступом до центрального сховища даних. Реалізація постачальника включає один чи кілька класів у додатку Android, а

також елементи у файлі маніфесту. Один з класів реалізує підклас `ContentProvider`, який виступає в ролі інтерфейсу між постачальником і іншими додатками. Незважаючи на те, що постачальники контенту спочатку призначені для надання доступу до даних іншим програмам, у вашому додатку, безсумнівно, можуть міститися операції, які дозволяють запитувати і змінювати дані, керовані постачальником.

4.2.1. Підготовка до створення постачальника

Перш ніж приступити до створення постачальника, виконайте наведені нижче дії.

1. Вирішіть, чи потрібен взагалі вам постачальник контенту. Постачальник контенту потрібен у випадках, якщо ви хочете реалізувати в своєму додатку одну або кілька таких функцій:

- 1) надання складних даних або файлів інших програм;
- 2) надання користувачам можливості копіювати складні дані з вашого додатка в інші програми;
- 3) надання пошукових підказок, що налаштовуються, за допомогою платформи пошуку.

Вам *не потрібен* постачальник для роботи з базою даних `SQLite`, якщо ви плануєте використовувати її виключно у вашому додатку.

2. Якщо ви ще не прийняли остаточне рішення, ознайомтеся з розділом «Основні відомості про постачальника контенту», щоб дізнатися детальніше про постачальників контенту.

Після цього можна приступати до створення постачальника. Для цього виконайте наведені нижче дії.

1. Спроектуйте базове сховище для своїх даних. Постачальник контенту може надати дані таким чином:

– *Дані для файлів*. Дані, які зазвичай надходять у файли, такі, як фотографії, аудіо чи відео. Файли слід зберігати в закритому просторі вашого додатка. У відповідь на запит файлу з іншої програми ваш постачальник може запропонувати дескриптор файлу.

– *Структуровані дані*. Дані, які зазвичай надходять в базу даних, масив або аналогічну структуру, слід зберігати в тій формі, яка сумісна з таблицями з рядків і стовпців. Рядок являє собою об'єкт, наприклад, користувача, позицію або облікову одиницю, стовпець – деякі дані про об'єкт, наприклад, ім'я користувача або вартість одиниці. Зазвичай дані такого типу зберігаються в базі даних `SQLite`, однак можна використовувати постійне сховище будь-якого типу.

2. Визначте конкретну реалізацію класу `ContentProvider` і його необхідні методи. Цей клас виступає в ролі інтерфейсу між вашими даними та іншою частиною системи Android.

3. Визначте рядок центру постачальника, його URI контенту і стовпців. Якщо треба, щоб додаток постачальника обробляв наміри, то необхідно визначити дії намірів, додаткові дані і прапорці. Крім того, необхідно визначити дозволи, які будуть необхідні програмі для доступу до ваших даних. Всі ці значення слід визначити як константи в окремому класі-контракті; надалі цей клас можна надати іншим розробникам.

4. Додайте інші додаткові компоненти, наприклад, демонстраційні дані або реалізацію адаптера `AbstractThreadedSyncAdapter`, який служить для синхронізації даних між постачальником і хмарним сховищем даних.

4.2.2. Проектування сховища даних

Постачальник контенту являє собою інтерфейс для передачі даних, збережених у структурованому форматі. Перш ніж створювати інтерфейс, визначте спосіб зберігання даних. Дані можна зберігати в будь-якій формі, а потім спроектувати інтерфейс для читання та запису даних при необхідності.

В Android є деякі технології зберігання даних:

1. У системі Android є API бази даних `SQLite`, який використовується власними постачальниками Android для зберігання табличних даних. За допомогою класу `SQLiteOpenHelper` можна створювати бази даних, а клас `SQLiteDatabase` являє собою базовий клас для доступу до баз даних.

Зверніть увагу, що вам не обов'язково використовувати базу даних для реалізації свого репозиторія. Постачальник являє собою зовнішній набір таблиць, як у випадку з реляційною базою даних, однак це не є вимогою до внутрішньої реалізації постачальника.

2. Для зберігання файлів даних в Android передбачено різні API-інтерфейси для роботи з файлами. Якщо проектується постачальник, який пропонує мультимедійні дані, такі, як музика чи відео, можна створити постачальник, що об'єднує табличні дані і файли.

3. Для роботи з мережевими даними використовуйте класи `java.net` і `android.net`. Також ви можете синхронізувати мережеві дані з локальним сховищем даних (наприклад, з базою даних), а потім надати ці дані у вигляді таблиць або файлів. Такий тип синхронізації демонструється на прикладі програми адаптера синхронізації.

Рекомендації щодо проектування даних. Наведемо декілька порад і рекомендацій щодо проектування структури даних постачальника:

1. У табличних даних завжди повинен бути стовпець для «основного ключа», який постачальник зберігає у вигляді унікального числового значення для кожного рядка. Можете використовувати це значення для зв'язування рядка з рядками в інших таблицях (використовуючи його як «ключ»). Незважаючи на те, що можна використовувати будь-яке ім'я для цього стовпця, рекомендується вказати ім'я `BaseColumns._ID`, оскільки для зв'язування результатів запиту постачальника з `ListView` необхідно, щоб один з одержуваних стовпців називався `_ID`.

2. Якщо планується надавати растрові зображення або дуже великі фрагменти даних для файлів, то дані слід зберігати у файлах, а потім надавати їх окремо замість зберігання прямо в таблиці. У такому випадку вам необхідно повідомити користувачам вашого постачальника про те, що для доступу до даних їм потрібно скористатися методом `ContentResolver`.

3. Для зберігання даних різного розміру або з різною структурою використовуйте тип `BLOB`. Наприклад, стовпець `BLOB` можна використовувати для зберігання буфера протоколу або структури `JSON`.

`BLOB` також можна використовувати для реалізації таблиці, що *не залежить від схеми*. В таблиці такого типу визначаються стовпець основного ключа, стовпець типу `MIME` та один або кілька спільних стовпців `BLOB`. На зміст даних у стовпцях `BLOB` вказує значення у стовпці типу `MIME`. Завдяки цьому у тій самій таблиці можна зберігати рядки різних типів. Прикладом таблиці, що не залежить від схеми, може служити таблиця з даними постачальника контенту `ContactsContract.Data`.

4.2.3. Проектування URI контенту

URI контенту являє собою URI, який визначає дані у постачальника. URI контенту можуть включати символічне ім'я всього постачальника (його центр) і ім'я, яке вказує на таблицю або файл (шлях). Додаткова частина URI з ідентифікатором вказує на окремий рядок у таблиці. У кожного методу доступу до даних в класі `ContentProvider` є URI контенту (у вигляді аргументу); завдяки цьому можна визначити таблицю, рядок або файл для доступу.

Проектування центру постачальника. У постачальника зазвичай є тільки один центр, який виступає як внутрішнє ім'я у системі Android. Щоб уникнути конфліктів з іншими постачальниками як основа центру постачальника повинні виступати відомості про володіння доменом в Інтернеті (в зворотному поряд-

ку). Оскільки ця рекомендація також застосовується і до назв пакетів Android, можна визначити центр свого постачальника у вигляді розширення назви пакета, в якому міститься постачальник. Наприклад, якщо пакет Android називається `com.example.<appname>`, то центром вашого постачальника має бути `com.example.<appname>.provider`.

Проектування структури шляху. Зазвичай розробники створюють URI контенту на основі центру постачальника, додаючи до нього шлях, який вказує на окремі таблиці. Наприклад, якщо є дві таблиці (*table1* і *table2*), центр постачальника з попереднього прикладу слід об'єднати для формування наступних URI контенту:

`com.example.<appname>.provider/table1`

і

`com.example.<appname>.provider/table2.`

Шляхи не обмежені одним сегментом, і не на кожному рівні шляху є таблиця.

Обробка ідентифікаторів URI контенту. Зазвичай постачальники надають доступ до одного рядка в таблиці шляхом прийняття URI контенту, в кінці якого вказано значення ідентифікатора рядка. Також постачальники зазвичай перевіряють збіг значення ідентифікатора по стовпцю `_ID` в таблиці і надають запитуваний доступ до відповідного рядка.

Це спрощує створення загального методу проектування для програм, які отримують доступ до постачальника. Додаток відправляє запит постачальнику і відображає отриманий в результаті такого запиту об'єкт `Cursor` в об'єкті `ListView` з допомогою `CursorAdapter`. Для визначення `CursorAdapter` необхідно, щоб один із стовпців в об'єкті `Cursor` називався `_ID`.

Потім користувач вибирає в інтерфейсі один з рядків, щоб переглянути або змінити їх. Додаток отримує відповідний рядок із об'єкта `Cursor` в базовому об'єкті `ListView`, отримує значення `_ID` для цього рядка, додає його до URI контенту, а потім відправляє постачальнику запит на доступ. Після чого постачальник може запросити або змінити рядок, обраний користувачем.

Шаблони URI контенту. Для вибору дії з вхідним даними URI контенту, в API постачальника є клас `UriMatcher`, в якому є шаблони URI контенту з цілочисельними значеннями. Такі цілочисельні значення, які відповідають певним шаблонам, можна використовувати в операторі `switch`, який вибирає відповідну дію для URI контенту.

Для визначення збігу URI контенту з шаблоном використовуються такі символи:

* – відповідність рядку будь-якої довжини з будь-якими припустимими символами;

– відповідність рядку будь-якої довжини з цифрами.

Як приклад для проектування, написання коду для обробки URI контенту рекомендується використовувати центр поставщика `com.example.app.provider`, який розпізнає наступні URI контенту, що вказують на таблиці:

- `content://com.example.app.provider/table1`: таблиця `table1`;
- `content://com.example.app.provider/table2/dataset1`: таблиця `dataset1`;
- `content://com.example.app.provider/table2/dataset2`: таблиця `dataset2`;
- `content://com.example.app.provider/table3`: таблиця `table3`.

Постачальник також розпізнає URI контент, якщо до них додано ідентифікатор рядка (наприклад, `content://com.example.app.provider/table3/1` для рядка з ідентифікатором 1 в таблиці `table3`).

Можливе використання наступних шаблонів URI контенту:

```
content://com.example.app.provider/*
```

Збігається з будь-яким URI контенту у постачальника.

```
content://com.example.app.provider/table2/*:
```

Збігається з URI контенту в таблицях `dataset1` і `dataset2`, однак не збігається з URI контенту в таблиці `table1` або `table3`.

```
content://com.example.app.provider/table3/#:
```

Співпадає з URI контенту для окремих рядків у таблиці `table3`, такими, як `content://com.example.app.provider/table3/6` рядка з ідентифікатором 6.

У фрагменті коду, наведеного нижче, показано, як працюють методи в класі `UriMatcher`. Цей код обробляє URI для всієї таблиці іншим чином, ніж для URI для окремого рядка, використовуючи шаблон URI вмісту `content://<authority>/<path>` для таблиць і шаблон `content://<authority>/<path>/<id>` – для окремих рядків.

Метод `addURI()` зіставляє центр постачальника і його шлях з цілочисельним значенням. Метод `match()` повертає ціле значення для URI. Оператор

switch вибирає, що йому треба виконати: запит всієї таблиці або тільки окремого запису:

```
public class ExampleProvider extends ContentProvider {
    ...
    // Створюємо об'єкт UriMatcher.
    private static final UriMatcher sUriMatcher;
    ...
    /*
     * Виклики функції addURI() знаходяться тут, для всіх моделей URI контен-
     * ту, що слід розпізнати постачальнику. В цьому фрагменті коду показаний виклик
     * цієї функції лише до таблиці 3 */
    ...
    /*
     * Встановлюємо цілочисельне значення рівне 1 для кількох рядків у таблиці
     * 3 */
    sUriMatcher.addURI("com.example.app.provider", "table3", 1);

    /*
     * Встановлюємо код для одного рядку рівним 2. У цьому випадку
     * використовується знак "#" знак. content://com.example.app.provider/table3/3"
     * співпадає, але
     * "content://com.example.app.provider/table3 ні.
     */
    sUriMatcher.addURI("com.example.app.provider", "table3/#", 2);
    ...
    // Реалізуємо ContentProvider.query()
    public Cursor query(
        Uri uri,
        String[] projection,
        String selection,
        String[] selectionArgs,
        String sortOrder) {
    ...
        /*
         * Вибраємо таблицю для запиту і порядок сортування, ґрунтуючись на
         * коді для вхідного URI. Тут також тільки вирази для таблиці 3.
         */

        switch (sUriMatcher.match(uri)) {

            // Якщо вхідний URI був для всієї таблиці
            case 1:

                if (TextUtils.isEmpty(sortOrder)) sortOrder = "_ID ASC";
                break;

            // Якщо вхідний URI був для одного рядка
            case 2:

                selection = selection + "_ID = " + uri.getLastPathSegment();
                break;

            default:
                ...
                // Якщо URI не знайдений, ви повинні виконати обробку помилок
        }

        // місце, щоб здійснити запит
    }
```

```
}
```

Інший клас, `ContentUris`, надає зручні методи для роботи з частиною ід URI контенту. Класи `Uri` і `Uri.Builder` містять зручні методи для синтаксичного аналізу існуючих об'єктів `Uri` і для створення нових.

4.2.4. Реалізація класу `ContentProvider`

Екземпляр класу `ContentProvider` управляє доступом до структурованого набору даних шляхом опрацювання запитів від інших додатків. У кінцевому рахунку, при всіх формах доступу викликається метод `ContentResolver`, який потім викликає конкретний метод `ContentProvider` для отримання доступу.

Необхідні методи. В абстрактному класі `ContentProvider` визначено шість абстрактних методів, які необхідно реалізувати в рамках вашого власного конкретного підкласу. Всі наведені нижче методи, крім `onCreate()`, викликаються клієнтським додатком, який намагається отримати доступ до вашого постачальника контенту.

1. `query()`. Отримання даних від постачальника. Використовує аргументи для вибору таблиці для запиту, рядків і стовпців, які потрібно повернути, і зазначає порядок сортування результатів. Повертає дані у вигляді об'єкта `Cursor`.

2. `insert()`. Вставка рядка в ваш постачальник. Використовує аргументи для вибору кінцевої таблиці та отримання значень стовпця, які слід використовувати. Повертає URI контенту для нового вставленого рядка.

3. `update()`. Оновлення існуючих рядків у постачальника. Використовує аргументи для вибору таблиці або рядків для оновлення, а також для отримання оновлених значень стовпця. Повертає кількість оновлених рядків.

4. `delete()`. Видалення рядків з постачальника. Використовує аргументи для вибору таблиці або рядків для видалення. Повертає кількість видалених рядків.

5. `getType()`. Повернення типу MIME, що відповідає URI контенту.

6. `onCreate()`. Ініціалізація постачальника. Система Android викликає цей метод відразу після створення вашого постачальника. Зверніть увагу, що постачальник не буде створено до тих пір, поки об'єкт `ContentResolver` не припинить спроби отримати доступ до нього.

Підпис цих методів аналогічний до підпису для ідентичних методів в об'єкті `ContentResolver`.

При реалізації цих методів слід враховувати такі моменти.

1. Всі ці методи, крім `onCreate()`, можна викликати відразу з декількох потоків, тому вони повинні бути реалізовані із збереженням потокобезпеки.

2. Уникайте занадто довгих операцій в методі `onCreate()`. Відкладіть виконання завдань ініціалізації доти, поки вони не будуть потрібні.

3. Незважаючи на те, що повинні реалізувати ці методи, код необов'язково повинен виконувати будь-які інші дії, крім повернення очікуваного типу даних. Наприклад, може знадобитися, щоб інші програми не мали можливості вставляти дані в деякі таблиці. Для цього можна ігнорувати виклик методу `insert()` і повернути `0`.

Реалізація методом `query()`. Метод `ContentProvider.query()` повинен повертати об'єкт `Cursor`, а при збої видавати `Exception`. Якщо як сховище даних використовується база даних `SQLite`, можна просто повернути об'єкт `Cursor`, який був повернутий одним з методів `query()` класу `SQLiteDatabase`. Якщо запит не відповідає ні одному рядку, слід повернути екземпляр об'єкта `Cursor`; метод `getCount()` повертає `0`. `Null` слід повертати тільки в тому випадку, якщо під час обробки запиту сталася внутрішня помилка.

Якщо ви не використовуєте базу даних `SQLite` як сховище даних, зверніться до одного з конкретних підкласів об'єкта `Cursor`. Наприклад, клас `MatrixCursor` реалізує об'єкт `cursor`, в якому кожен рядок являє собою масив класу `Object`. За допомогою цього класу скористайтеся методом `addRow()`, щоб додати новий рядок.

Слід пам'ятати, що система `Android` повинна мати можливість взаємодіяти з `Exception` в межах процесу. Система `Android` дозволяє це робити для зазначених нижче винятків, які можуть бути корисні при обробці помилок запитів:

- `IllegalArgumentException` (цей виняток можна видати у разі, якщо постачальник одержує неправильний `URI` контенту);
- `NullPointerException`.

Реалізація методу `insert()`. Метод `insert()` додає новий рядок у відповідний рядок, використовуючи значення в аргументі `ContentValues`. Якщо в аргументі `ContentValues` відсутнє ім'я стовпця, можливо, знадобиться вказати для нього значення за замовчуванням (або в коді постачальника, або у схемі бази даних).

Цей метод повинен повертати `URI` контенту для нового рядка. Для цього додайте значення `_ID` нового рядка (або інший основний ключ) до `URI` контенту таблиці, використовуючи метод `withAppendedId()`.

Реалізація методу delete(). Методом delete() необов'язково фактично видаляти рядки з вашого сховища даних. Якщо для роботи з постачальником використовується адаптер синхронізації, розгляньте можливість позначки віддаленого рядка прапором delete замість остаточного видалення рядка. Адаптер синхронізації може перевірити наявність видалених рядків з прапором delete, щоб видалити їх з сервера перед видаленням їх з постачальника.

Реалізація методу update(). Метод update() приймає той же аргумент ContentValues, який застосовує метод insert(), і ті ж аргументи selection і selectionArgs, які використовуються методами delete() і ContentProvider.query(). Завдяки цьому код можна повторно використовувати між даними методами.

Реалізація методу onCreate(). Система Android викликає метод onCreate() при запуску постачальника. У цьому методі слід виконувати тільки завдання ініціалізації, що швидко виконуються, а створення бази даних і завантаження даних відкласти до моменту фактичного отримання постачальником запиту на інформацію. Занадто довгі операції в методі onCreate() призводять до збільшення часу запуску постачальника. У свою чергу, це збільшує час відгуку постачальника на запити від інших додатків.

Наприклад, якщо ви використовуєте базу даних SQLite, в методі ContentProvider.onCreate() можна створити новий об'єкт SQLiteOpenHelper, а потім створити таблиці SQL при першому відкритті бази даних. Щоб спростити це, при першому виклику методу getWritableDatabase() він автоматично викликає метод SQLiteOpenHelper.onCreate().

В наведених нижче фрагментах коду ілюструється взаємодія між методом ContentProvider.onCreate() і методом SQLiteOpenHelper.onCreate(). У першому фрагменті коду наведена реалізація методу ContentProvider.onCreate():

```
public class ExampleProvider extends ContentProvider

    /*
     * Визначаємо дескриптор для допоміжного об'єкта бази даних. Клас
     * MainDatabaseHelper визначається
     * у наступному фрагменті.
     */
    private MainDatabaseHelper mOpenHelper;
    // Визначаємо ім'я бази даних
    private static final String DBNAME = "mydb";

    // Зберігає об'єкт бази даних
    private SQLiteDatabase db;
```

```

public boolean onCreate() {
    /*
     * Створюємо новий допоміжний об'єкт. Цей спосіб завжди швидкий.
     */
    mOpenHelper = new MainDatabaseHelper(
        getContext(),          // контекст додатку
        DBNAME,                // ім'я бази даних)
        null,                  //використовуємо SQLite курсор за замовчуван-
ням
        1                      // номер версії
    );
    return true;
}

// Реалізуємо метод Insert постачальника
public Cursor insert(Uri uri, ContentValues values) {
    // Необхідно вставити тут код, щоб визначити, які таблиці відкриті,
    обробити помилки, і так далі

    ...

    /*
     * Отримуємо записи бази даних.
     */
    db = mOpenHelper.getWritableDatabase();
}
}

```

У наступному фрагменті коду наведена реалізація методу `SQLiteOpenHelper.onCreate()`, включаючи допоміжний клас:

```

...
// Рядок, яка визначає інструкцію SQL для створення таблиці
private static final String SQL_CREATE_MAIN = "CREATE TABLE " +
    "main " +                      // Назва таблиці
    "(" +                          // Столпці в таблиці
    " _ID INTEGER PRIMARY KEY, " +
    " WORD TEXT"
    " FREQUENCY INTEGER " +
    " LOCALE TEXT );";
...
/**
 * Клас Helper, який фактично створює і управляє основним сховищем даних про-
 * вайдера.
 */
protected static final class MainDatabaseHelper extends SQLiteOpenHelper {

    /*
     * Створює екземпляр відкритого помічника для SQLite сховища даних поста-
     * чальника
     * Не робити створення бази даних та оновлення тут.
     */
    MainDatabaseHelper(Context context) {
        super(context, DBNAME, null, 1);
    }

    /*
     * Створює сховище даних. Це викликається при спробі постачальника, щоб
     * відкрити

```

```

        * сховище і SQLite повідомляє, що він не існує.
        */
        public void onCreate(SQLiteDatabase db) {

            // Створює основну таблицю
            db.execSQL(SQL_CREATE_MAIN);

        }
    }
}

```

4.2.5. Реалізація типів MIME постачальника контенту

У класі `ContentProvider` передбачено два методи для повернення типів MIME:

1) `getType()` – один з необхідних методів, який потрібно реалізувати для кожного постачальника;

2) `getStreamTypes()` – метод, який потрібно реалізувати в разі, якщо постачальник надає файли.

Типи MIME для таблиць. Метод `getType()` повертає об'єкт `String` у форматі MIME, який описує тип даних, що повертаються аргументом URI контенту. Аргумент `Uri` може виступати як шаблон, а не як певний URI; в цьому випадку необхідно повернути тип даних, пов'язаний з URI контенту, який відповідає шаблонам.

Для загальних типів даних, таких, як текст, HTML або JPEG, метод `getType()` повинен повертати стандартний тип MIME. Повний список стандартних типів наведено на веб-сайті IANA MIME Media Types.

Для URI контенту, які вказують на один або кілька рядків табличних даних, метод `getType()` повинен повертати тип MIME у форматі MIME постачальника, який є в системі Android:

Частина типу: `vnd.`

Частина підтипу:

1) якщо шаблон URI призначений для одного рядка:

`android.cursor.item/;`

2) якщо шаблон URI призначений для декількох рядків:

`android.cursor.dir/.`

Частина постачальника: `vnd.<name>.<type>.`

Вказуєте `<name>` й `<type>`. Значення `<name>` має бути унікальним глобально, а значення `<type>` – унікальним для відповідного шаблону URI. Як `<name>` рекомендується використовувати назву вашої компанії або частину назви пакета Android вашого додатку. Як `<type>` рекомендується використовувати рядок, який визначає пов'язану з URI таблицю.

Наприклад, якщо центр постачальника є `com.example.app.provider`,

який надає таблицю **table1**, то тип MIME для кількох рядків у таблиці **table1** буде таким:

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

Для одного рядка в таблиці **table1** тип MIME буде таким:

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

Типи MIME для файлів. Якщо постачальник надає файли, необхідно реалізувати метод `getStreamTypes()`. Цей метод повертає масив `String` з типами MIME для файлів, що поведуться постачальником для заданого URI контенту. Запропоновані постачальником типи слід сортувати за допомогою аргументу фільтра типів MIME, щоб повертались тільки ті типи MIME, які необхідно обробити клієнту.

Наприклад, розглянемо постачальника, який надає фотографії у вигляді файлів у форматах `.jpg`, `.png` і `.gif`. Якщо додаток викликає метод `ContentResolver.getStreamTypes()` з рядком фільтра `image/*` (щось подібне до «зображення»), то метод `ContentProvider.getStreamTypes()` повинен повертати наступний масив:

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

Якщо ж додатку потрібні тільки файли `.jpg`, то він викликає метод `ContentResolver.getStreamTypes()` з рядком фільтра `*\jpeg`; метод `ContentProvider.getStreamTypes()` при цьому повинен повертати таке:

```
{ "image/jpeg" }
```

Якщо постачальник не надає жоден з типів MIME, запитаних в рядку фільтра, то метод `getStreamTypes()` повинен повертати `null`.

4.2.6. Реалізація класу-контракту

Клас-контракт являє собою клас `public final`, в якому містяться визначення констант для URI, імен стовпців, типів MIME та інших метаданих постачальника. Клас встановлює контрактні відносини між постачальником і іншими додатками шляхом забезпечення прямого доступу до постачальника навіть у разі зміни фактичних значень URI, імен стовпців і т. д.

Клас-контракт також корисний для розробників тим, що в ньому містяться мнемонічні імена для його констант, завдяки чому знижується ризик

того, що розробники скористаються неправильними значеннями для імен стовпців або URI. Оскільки це клас, він може містити документацію Javadoc. Інтегровані середовища розробки, такі, як Eclipse, можуть автоматично заповнювати імена констант з класу-контракту і відображати Javadoc для констант.

У розробників немає доступу до файлу класу-контракту з вашого додатка, проте вони можуть статично скомпілювати клас-контракт в свій додаток з наданого вами файлу `.jar`.

Прикладом класу-контракту може служити клас `ContactsContract` і його вкладені класи.

4.2.7. Реалізація дозволів постачальника контенту

Нижче наведено короткий огляд основних моментів.

1. За замовчуванням файли з даними зберігаються у внутрішньому сховищі пристрою і доступні тільки вашому додатку і постачальнику.

2. Створювані вами бази даних `SQLiteDatabase` також доступні тільки вашому додатку і постачальнику.

3. Файли з даними, які зберігаються в зовнішньому сховищі, за замовчуванням є загальнодоступними, і їх може зчитати будь-який користувач. Вам не вдасться використовувати постачальника контенту для обмеження доступу до файлів, які зберігаються в зовнішньому сховищі, оскільки додатки можуть використовувати інші виклики API для їх читання або запису.

4. Виклики методу для відкриття або створення файлів або баз даних `SQLite`, що знаходяться у внутрішньому сховищі на вашому пристрої, потенційно можуть надати всім іншим додаткам доступ як на запис, так і на читання даних. Якщо використовується внутрішній файл або база даних як сховище постачальника, в якому виконати читання або запис даних може будь-який користувач, то дозволів, заданих в маніфесті постачальника, буде явно недостатньо для захисту ваших даних. За замовчуванням доступ до файлів і баз даних у внутрішньому сховищі є закритим, і вам не слід змінювати параметри доступу до сховища вашого постачальника.

Якщо необхідно використовувати дозволи постачальника контенту для управління доступом до даних, то зберігайте дані у внутрішніх файлах, в базах даних `SQLite` або в хмарі (наприклад, на віддаленому сервері), а доступ до файлів і баз даних має бути наданий тільки вашому додатку.

Реалізація дозволів. Будь-який додаток може виконувати читання даних в постачальнику або записувати їх, навіть якщо відповідні дані є закритими, оскільки за замовчуванням для постачальника не задані дозволи. Щоб змінити

ці настройки, задайте дозволи для постачальника у файлі маніфесту за допомогою атрибутів елемента `<provider>` або його дочірніх елементів. Можна задати дозволи, які застосовуються до всього постачальника або тільки до певних таблиць, або навіть тільки до певних записів або всього дерева.

Для завдання дозволів використовується один або кілька елементів `<permission>` у файлі маніфесту. Щоб дозволи були унікальними для постачальника, використовуйте області, аналогічні Java, для атрибута `android:name`. Наприклад, надайте дозволу на читання ім'я `com.example.app.provider.permission.READ_PROVIDER`.

Нижче перераховані області дозволів для постачальника, починаючи з дозволів, які застосовуються до всього постачальника, і закінчуючи більш детальними дозволами. Більш докладні дозволи мають перевагу над дозволами з більш широкими областями.

Одиничний дозвіл на читання / запис на рівні постачальника. Одиничний дозвіл, який управляє доступом як на читання, так і на запис для всього постачальника, задається за допомогою атрибута `android:permission` елемента `<provider>`.

Окремий дозвіл на читання / запис на рівні постачальника. Дозволи на читання і запис для всього постачальника задаються за допомогою атрибутів `android:readPermission` й `android:writePermission` елемента `<provider>`. Вони мають переважну силу над дозволом, заданим за допомогою атрибута `android:permission`.

Дозвіл на рівні шляху. Це дозвіл на читання, запис або читання / запис для URI контенту в постачальника. Кожен URI, що підлягає керуванню, задається за допомогою дочірнього елемента `<path-permission>` елемента `<provider>`. Для кожного зазначеного вище URI контенту можна задати дозвіл на читання / запис, тільки читання або тільки запис, або всі три дозволи. Дозволи на читання і запис мають переважну силу над дозволом на читання / запис. Крім того, дозволи на рівні шляху мають переважну силу над дозволами на рівні постачальника.

Тимчасовий дозвіл. Дозволи цього рівня надають додатку тимчасовий доступ, навіть якщо у додатка немає дозволів, які зазвичай потрібні. Функція тимчасового доступу обмежує набір дозволів, які з додатком необхідно запросити в своєму маніфесті. Якщо включені тимчасові дозволи, то єдиними додатками, яким потрібні «постійні» дозволи на роботу з постачальником, є ті, які безперервно отримують доступ до всіх ваших даних.

Розглянемо приклад з дозволами, які необхідно реалізувати для постачальника електронної пошти, та додатками, коли вам необхідно дозволити

зовнішньому додатку для перегляду зображень відображати вкладені в листи фотографії з постачальника. Щоб надати засобу перегляду зображень необхідний доступ без запиту дозволів, задайте тимчасові дозволи для URI контенту фотографій. Спроектуйте ваш додаток для роботи з електронною поштою таким чином, щоб у випадках, коли користувач бажає відобразити фотографію, додаток відправляв намір, в якому міститься URI контенту фотографії та прапорці дозволу для засобу перегляду зображень. Потім засіб перегляду зображень може надсилати операторові електронної пошти запит на отримання фотографії, навіть якщо у засоба перегляду відсутній звичайний дозвіл на читання даних з постачальника.

Щоб включити тимчасові дозволи, задайте атрибут `android:grantUriPermissions` для елемента `<provider>`, або додайте один або кілька дочірніх елементів `<grant-uri-permission>` в ваш елемент `<provider>`. Якщо використовуються тимчасові дозволи, вам необхідно викликати метод `Context.revokeUriPermission()` кожен раз, коли ви здійснюєте віддалену підтримку URI контенту з постачальника, а URI контенту пов'язаний з тимчасовим дозволом.

Значення атрибута визначає, яка частина постачальника доступна. Якщо для атрибута задано значення `true`, система надасть тимчасові дозволи для всього постачальника, скасовуючи тим самим будь-які інші дозволи, які потрібні на рівні постачальника або на рівні шляху.

Якщо для прапора задано значення `false`, вам необхідно додати дочірні елементи `<grant-uri-permission>` в свій елемент `<provider>`. Кожен дочірній елемент задає URI контенту, для яких надається тимчасовий дозвіл.

Щоб делегувати додатку тимчасовий доступ, в намірі повинні бути вказані прапорці `FLAG_GRANT_READ_URI_PERMISSION` або `FLAG_GRANT_WRITE_URI_PERMISSION`, або обидва прапорці. Ці прапорці задаються за допомогою методу `setFlags()`.

Якщо атрибут `android:grantUriPermissions` відсутній, передбачається, що його значення `false`.

4.2.8. Елемент `<provider>`

Як і компоненти `Activity` й `Service`, підклас класу `ContentProvider` повинен бути визначений у файлі маніфесту програми за допомогою елемента `<provider>`. Нижче вказана інформація, яку система Android отримує з цього елемента.

Центр постачальника (*android:authorities*). Символічні імена, які ідентифікують весь постачальник в системі.

Назва класу постачальника (android:name). Клас, який реалізує клас `ContentProvider`.

Дозволи. Атрибути, які визначають дозволи, необхідні іншим додаткам для доступу до даних в постачальника:

- `android:grantUriPermissions`: прапорець тимчасового дозволу;
- `android:permission`: одиничний дозвіл на читання / запис на рівні постачальника;
- `android:readPermission`: дозвіл на читання на рівні постачальника;
- `android:writePermission` дозвіл на запис на рівні постачальника.

Додаткові відомості про дозволи і відповідні атрибути подані в розділі «Реалізація дозволів постачальника контенту».

Атрибути запуску та управління. Наступні атрибути визначають порядок і час запуску постачальника системи Android, характеристики процесу постачальника, а також інші параметри виконання:

- `android:enabled`: прапорець, що дозволяє системі запускати постачальника;
- `android:exported`: прапорець, що дозволяє іншим програмам використовувати цей постачальник;
- `android:initOrder`: порядок запуску постачальника (щодо інших постачальників у тому ж самому процесі);
- `android:multiProcess`: прапорець, що дозволяє системі запускати постачальника в тому ж процесі, що і викликає клієнт;
- `android:process`: назва процесу, в якому запускається постачальник;
- `android:syncable`: прапорець, який вказує на те, що дані в постачальнику слід синхронізувати з даними на сервері.

Інформаційні атрибути. Додатковий значок і мітка для постачальника:

– `android:icon`: графічний ресурс, що містить значок для постачальника. Значок відображається поруч з міткою постачальника в списку додатків в розділі *Налаштування> Програми> Все*.

– `android:label`: інформаційна мітка з описом постачальника або його дані, або обидва описи. Мітка відображається в списку додатків в розділі *«Налаштування> Програми> Все»*.

4.2.9. Наміри і доступ до даних

Додатки можуть отримувати доступ до контенту за допомогою об'єктів `Intent`. Додаток при цьому не викликає будь-які методи класів `ContentResolver` чи `ContentProvider`. Замість цього він відправляє намір операції, що запускається, яка зазвичай є частиною власного додатка постачальника. Залежно від дії, зазначеної в намірі, кінцева операція також може запропонувати користувачеві внести зміни в дані постачальника. У намірі також можуть міститися додаткові дані, які кінцева операція відображає в інтерфейсі; потім користувачеві пропонується можливість змінити ці дані, перш ніж використовувати їх для зміни даних в постачальника.

Можливо, доступ за допомогою наміру потрібно використовувати для забезпечення цілісності даних. Для вставки, оновлення та видалення даних в постачальника може існувати суворо певний програмний код, який реалізує його функціональні можливості. У цьому випадку надання іншим додаткам прямого доступу для зміни даних може призвести до того, що інформація буде недійсною. Якщо необхідно надати розробникам можливість доступу за допомогою намірів, вам слід ретельно задокументувати таку функцію. Поясніть їм, чому доступ за допомогою намірів через призначений для користувача інтерфейс вашої програми набагато кращий від зміни даних за допомогою їх коду.

Обробка вхідного наміру для зміни даних постачальника нічим не відрізняється від обробки інших намірів.

4.3. Завантажувачі (Loaders)

Завантажувачі, які з'явилися в Android 3.0, спрощують асинхронне завантаження даних в операцію або фрагмент. Завантажувачі мають певні властивості, а саме, вони:

- є наявними для будь-яких операцій `Activity` і фрагментів `Fragment`;
- забезпечують асинхронне завантаження даних;
- відстежують джерело своїх даних і видають нові результати при зміні контенту;
- можуть автоматично перепідключатися до останнього курсора завантажувача при відтворенні після зміни конфігурації. Таким чином, їм не потрібно повторно запитувати свої дані.

Зведена інформація про API-інтерфейсі завантажувача. Є кілька класів і інтерфейсів, які можуть використовувати завантажувачі в додатку (табл. 4.3).

Таблиця 4.3 – Властивості завантажувачів

Клас / інтерфейс	Опис
LoaderManager	Абстрактний клас, що пов’язується з Activity чи Fragment для управління одним або декількома інтерфейсами Loader. Це дозволяє додатку керувати операціями, які виконуються досить довго разом з життєвим циклом Activity чи Fragment; найчастіше цей клас використовується з CursorLoader, однак для додатка можуть писати свої власні завантажувачі для роботи з іншими типами даних. Є тільки один клас LoaderManager на операцію або фрагмент. Однак у класу LoaderManager може бути кілька завантажувачів
LoaderManager.LoaderCallbacks	Інтерфейс зворотного виклику, що забезпечує взаємодію клієнта з LoaderManager. Наприклад, за допомогою методу зворотного виклику onCreateLoader() створюється новий завантажувач
Loader	Абстрактний клас, який виконує асинхронне завантаження даних. Це базовий клас для завантажувача. Зазвичай використовується CursorLoader, але можна реалізувати і власний підклас. Коли завантажувачі активні, вони повинні відслідковувати джерело своїх даних і видавати нові результати при зміні контенту
AsyncTaskLoader	Абстрактний завантажувач, який надає AsyncTask для виконання роботи.
CursorLoader	Підклас класу AsyncTaskLoader, який запитує ContentResolver і повертає Cursor. Цей клас реалізує протокол Loader стандартним способом для виконання запитів до курсора. Він будується на AsyncTaskLoader для виконання запиту до курсора у фоновому потоці, щоб не блокувати призначений для користувача інтерфейс програми. Використання цього завантажувача – це найкращий спосіб асинхронного завантаження даних з ContentProvider замість виконання керованого запиту через платформу або API-інтерфейси операції

Наведені в табл. 4.3 класи та інтерфейси є найбільш важливими компонентами, за допомогою яких в додатку реалізується завантажувач. При створенні кожного завантажувача не потрібно використовувати всі ці компоненти, проте завжди слід вказувати посилання на `LoaderManager` для ініціалізації завантажувача і використовувати реалізацію класу `Loader`, наприклад `CursorLoader`.

Використання завантажувачів в додатку. У цьому розділі описується використання завантажувачів в додатку для Android. У додатках, що використовують завантажувачі, зазвичай є такі елементи:

- `Activity` чи `Fragment`;
 - екземпляр `LoaderManager`;
 - `CursorLoader` для завантаження даних, які видаються `ContentProvider`.
- Також можна реалізувати власний підклас класу `Loader` чи `AsyncTaskLoader` для завантаження даних з іншого джерела;
- реалізація для `LoaderManager.LoaderCallbacks`. Саме тут створюються нові завантажувачі і ведеться управління посиланнями на існуючі завантажувачі;
 - спосіб відображення даних завантажувача, наприклад `SimpleCursorAdapter`;
 - джерело даних, наприклад `ContentProvider`, коли використовується `CursorLoader`.

Запуск завантажувача. `LoaderManager` управляє одним або декількома екземплярами `Loader` в `Activity` чи `Fragment`. Є тільки один `LoaderManager` на кожну операцію або кожен фрагмент.

`Loader` зазвичай ініціалізується в методі `onCreate()` операції або в методі фрагмента `onActivityCreated()`. Робиться це в такий спосіб:

```
// Prepare the loader. Either re-connect with an existing one,  
// or start a new one.  
getLoaderManager().initLoader(0, null, this);
```

Метод `initLoader()` приймає такі параметри:

- унікальний ідентифікатор, що позначає завантажувач. В даному прикладі ідентифікатором є 0;
- необов'язкові аргументи, які передаються завантажувачу при побудові (в даному прикладі це `null`);
- реалізація `LoaderManager.LoaderCallbacks`, яка викликає клас `LoaderManager` для видачі подій завантажувача. В даному прикладі локальний клас реалізує інтерфейс `LoaderManager.LoaderCallbacks`, тому він передає посилання самому собі: `this`.

Виклик `initLoader()` забезпечує ініціалізацію завантажувача. Можливий один з двох результатів:

1. Якщо завантажувач, вказаний за допомогою ідентифікатора, вже існує, то буде повторно використаний завантажувач, створений останнім.

2. Якщо завантажувач, вказаний за допомогою ідентифікатора, не існує, то `initLoader()` викликає метод `LoaderManager.LoaderCallbacks.onCreateLoader()`. Саме тут реалізується код для створення екземпляра і повернення нового завантажувача.

У будь-якому випадку реалізація `LoaderManager.LoaderCallbacks` зв'язується з завантажувачем і буде викликатися при зміні стану завантажувача. Якщо в момент цього виклику компонент знаходиться в запущеному стані, це означає, що запитаний завантажувач вже існує і сформував свої дані. В цьому випадку система відразу ж викличе `onLoadFinished()` (під час `initLoader()`), будьте готові до цього.

Зверніть увагу, що метод `initLoader()` повертає створюваний клас `Loader`, але записувати посилання на нього не потрібно; клас `LoaderManager` управляє життєвим циклом завантажувача автоматично. Клас `LoaderManager` почне завантажувати і завершується при необхідності, а також підтримує стан завантажувача і пов'язаного з ним контенту. А це означає, що ви будете рідко взаємодіяти з завантажувачами безпосередньо (проте приклад використання методів завантажувача для тонкої настройки його поведінки див. у зразку коду `LoaderThrottle`). Для втручання в процес завантаження при виникненні певних подій зазвичай використовуються методи `LoaderManager.LoaderCallbacks`.

Перезапуск завантажувача. При використанні методу `initLoader()`, як показано вище, задіюється існуючий завантажувач із зазначеним ідентифікатором – в разі його наявності. Якщо такого завантажувача немає, метод його створить. Однак іноді старі дані потрібно відкинути і почати все заново.

Для видалення старих даних використовується метод `restartLoader()`. Наприклад, ця реалізація методу `SearchView.OnQueryTextListener` перезавантажує завантажувач, коли змінюється запит користувача. Завантажувач необхідно перезавантажити, для того щоб він міг використовувати змінений фільтр пошуку для виконання нового запиту:

```
public boolean onQueryTextChanged(String newText) {
    // Called when the action bar search text has changed. Update
    // the search filter, and restart the loader to do a new query
    // with this filter.
    mCurFilter = !TextUtils.isEmpty(newText) ? newText : null;
    getLoaderManager().restartLoader(0, null, this);
}
```

```
    return true;
}
```

Використання зворотних викликів класу LoaderManager. **LoaderManager.LoaderCallbacks** являє собою інтерфейс зворотного виклику, який дозволяє клієнту взаємодіяти з класом **LoaderManager**.

Очікується, що завантажувачі, зокрема **CursorLoader**, зберігатимуть свої дані після їх зупинки. Це дозволяє додаткам зберігати свої дані в методах **onStop()** і **onStart()** операції або фрагмента, з тим щоб, коли користувач повернеться в додаток, йому не довелося чекати, поки дані завантажуться заново. Методи **LoaderManager.LoaderCallbacks** використовуються, щоб дізнатися, коли потрібно створити новий завантажувач, а також для того, щоб вказати додатку, коли прийшов час перестати використовувати дані завантажувача.

Інтерфейс **LoaderManager.LoaderCallbacks** використовує такі методи:

- **onCreateLoader()** – створює екземпляр і повертає новий клас **Loader** для даного ідентифікатора;
- **onLoadFinished()** – викликається, коли створений раніше завантажувач завершив завантаження;
- **onLoaderReset()** – викликається, коли стан створеного раніше завантажувача скидається, в результаті чого його дані губляться.

Метод onCreateLoader. При спробі доступу до завантажувача (наприклад, за допомогою методу **initLoader()**), можна перевірити, чи існує завантажувач, вказаний за допомогою ідентифікатора. Якщо завантажувач не існує, він викликає метод **LoaderManager.LoaderCallbacks.onCreateLoader()**. Саме тут і створюється новий завантажувач. Зазвичай це буде клас **CursorLoader**, однак можна реалізувати і власний підклас класу **Loader**.

У цьому прикладі метод зворотного виклику **onCreateLoader()** створює клас **CursorLoader**. Треба побудувати клас **CursorLoader** за допомогою його методу конструктора, для чого потрібен повний набір інформації, яка необхідна для виконання запиту до **ContentProvider**, зокрема:

- **uri** – URI контенту, який необхідно отримати;
- **projection** – список стовпців, які будуть повернуті; при передачі **null** будуть повернуті всі стовпці, а це неефективно;
- **selection** – фільтр, який оголошує, які рядки повертати, відформатований у вигляді пропозиції SQL WHERE (за винятком самого WHERE); при передачі **null** будуть повернуті всі рядки для даного URI;

– **selectionArgs** – у вибірку можна включити символи «?», які будуть замінені значеннями з **selectionArgs** в порядку проходження в вибірці. Значення будуть прив'язані як рядки;

– **sortOrder** – порядок розташування рядків, відформатований у вигляді пропозиції SQL ORDER BY (за винятком самого ORDER BY). При передачі **null** буде використовуватися стандартний порядок сортування, а тому, список, можливо, буде несортованим.

Наприклад:

```
// If non-null, this is the current filter the user has provided.
String mCurFilter;
...
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    // This is called when a new Loader needs to be created. This
    // sample only has one Loader, so we don't care about the ID.
    // First, pick the base URI to use depending on whether we are
    // currently filtering.
    Uri baseUri;
    if (mCurFilter != null) {
        baseUri = Uri.withAppendedPath(Contacts.CONTENT_FILTER_URI,
            Uri.encode(mCurFilter));
    } else {
        baseUri = Contacts.CONTENT_URI;
    }

    // Now create and return a CursorLoader that will take care of
    // creating a Cursor for the data being displayed.
    String select = "(" + Contacts.DISPLAY_NAME + " NOTNULL) AND ("
        + Contacts.HAS_PHONE_NUMBER + "=1) AND ("
        + Contacts.DISPLAY_NAME + " != '' )";
    return new CursorLoader(getActivity(), baseUri,
        CONTACTS_SUMMARY_PROJECTION, select, null,
        Contacts.DISPLAY_NAME + " COLLATE LOCALIZED ASC");
}
```

Метод onLoadFinished. Цей метод викликається, коли створений раніше завантажувач завершив завантаження. Цей метод гарантовано викликається до вивільнення останніх даних, які були надані цьому завантажувачу. До цього моменту необхідно повністю перестати використовувати старі дані (оскільки вони скоро будуть замінені). Однак цього не слід робити самостійно, оскільки даними володіє завантажувач і він подбає про це.

Завантажувач вивільнить дані, як тільки дізнається, що додаток їх більше не використовує. Наприклад, якщо даними є курсор з **CursorLoader**, не слід викликати **close()** самостійно. Якщо курсор розміщується в **CursorAdapter**, слід використовувати метод **swapCursor()** з тим, щоб старий **Cursor** не заклався. Наприклад:

```
// This is the Adapter being used to display the list's data.
SimpleCursorAdapter mAdapter;
...

public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
    // Swap the new cursor in. (The framework will take care of
    // closing the
    // old cursor once we return.)
    mAdapter.swapCursor(data);
}
```

Метод onLoaderReset. Цей метод викликається, коли стан створеного раніше завантажувача скидається, в результаті чого його дані губляться. Цей зворотний виклик дозволяє дізнатися, коли дані ось-ось будуть вивільнені, з тим щоб можна було видалити своє посилання на них.

Дана реалізація викликає `swapCursor()` зі значенням `null`:

```
// This is the Adapter being used to display the list's data.
SimpleCursorAdapter mAdapter;
...

public void onLoaderReset(Loader<Cursor> loader) {
    // This is called when the last Cursor provided to onLoadFinished()
    // above is about to be closed. We need to make sure we are no
    // longer using it.
    mAdapter.swapCursor(null);
}
```

Приклад. Як приклад далі наведена повна реалізація фрагмента `Fragment`, який відображає `ListView` з результатами запиту до постачальника такого контенту, як контакти. Для управління запитом до постачальника використовується клас `CursorLoader`.

Щоб додаток міг звертатися до контактів користувача, як показано в цьому прикладі, в його маніфесті повинен бути присутнім дозвіл `READ_CONTACTS`.

```
public static class CursorLoaderListFragment extends ListFragment
    implements OnQueryTextListener, LoaderManager.LoaderCallbacks<Cursor> {

    // This is the Adapter being used to display the list's data.
    SimpleCursorAdapter mAdapter;

    // If non-null, this is the current filter the user has provided.
    String mCurFilter;

    @Override public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);

        // Give some text to display if there is no data. In a real
        // application this would come from a resource.
        setEmptyText("No phone numbers");

        // We have a menu item to show in action bar.
        setHasOptionsMenu(true);
    }
}
```

```

// Create an empty adapter we will use to display the loaded data.
mAdapter = new SimpleCursorAdapter(getActivity(),
    android.R.layout.simple_list_item_2, null,
    new String[] { Contacts.DISPLAY_NAME, Contacts.CONTACT_STATUS},
    new int[] { android.R.id.text1, android.R.id.text2 }, 0);
setListAdapter(mAdapter);

// Prepare the loader. Either re-connect with an existing one,
// or start a new one.
getLoaderManager().initLoader(0, null, this);
}

@Override public void onCreateOptionsMenu(Menu menu, MenuInflater inflater){
    // Place an action bar item for searching.
    MenuItem item = menu.add("Search");
    item.setIcon(android.R.drawable.ic_menu_search);
    item.setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);
    SearchView sv = new SearchView(getActivity());
    sv.setOnQueryTextListener(this);
    item.setActionView(sv);
}

public boolean onQueryTextChange(String newText) {
    // Called when the action bar search text has changed. Update
    // the search filter, and restart the loader to do a new query
    // with this filter.
    mCurFilter = !TextUtils.isEmpty(newText) ? newText : null;
    getLoaderManager().restartLoader(0, null, this);
    return true;
}

@Override public boolean onQueryTextSubmit(String query) {
    // Don't care about this.
    return true;
}

@Override public void onListItemClick(ListView l, View v, int position, long id) {
    // Insert desired behavior here.
    Log.i("FragmentComplexList", "Item clicked: " + id);
}

// These are the Contacts rows that we will retrieve.
static final String[] CONTACTS_SUMMARY_PROJECTION = new String[] {
    Contacts._ID,
    Contacts.DISPLAY_NAME,
    Contacts.CONTACT_STATUS,
    Contacts.CONTACT_PRESENCE,
    Contacts.PHOTO_ID,
    Contacts.LOOKUP_KEY,
};

public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    // This is called when a new Loader needs to be created. This
    // sample only has one Loader, so we don't care about the ID.
    // First, pick the base URI to use depending on whether we are
    // currently filtering.
    Uri baseUri;
    if (mCurFilter != null) {
        baseUri = Uri.withAppendedPath(Contacts.CONTENT_FILTER_URI,
            Uri.encode(mCurFilter));
    } else {
        baseUri = Contacts.CONTENT_URI;
    }

    // Now create and return a CursorLoader that will take care of
    // creating a Cursor for the data being displayed.
    String select = "(" + Contacts.DISPLAY_NAME + " NOTNULL) AND ("
        + Contacts.HAS_PHONE_NUMBER + "=1) AND ("
        + Contacts.DISPLAY_NAME + " != '')";
    return new CursorLoader(getActivity(), baseUri,
        CONTACTS_SUMMARY_PROJECTION, select, null,
        Contacts.DISPLAY_NAME + " COLLATE LOCALIZED ASC");
}

```

```

    }

    public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
        // Swap the new cursor in. (The framework will take care of closing the
        // old cursor once we return.)
        mAdapter.swapCursor(data);
    }

    public void onLoaderReset(Loader<Cursor> loader) {
        // This is called when the last Cursor provided to onLoadFinished()
        // above is about to be closed. We need to make sure we are no
        // longer using it.
        mAdapter.swapCursor(null);
    }
}

```

Інші приклади. В *ApiDemos* є кілька різних прикладів, які ілюструють використання завантажувачів:

- **LoaderCursor** – повна версія показаного вище фрагмента;
- **LoaderThrottle** – приклад того, як використовувати регулювання для скорочення числа запитів, які виконуються постачальником контенту при зміні його даних.

4.4. Постачальник календаря

Постачальник календаря являє собою репозиторій для подій календаря користувача. API постачальника календаря дозволяє запитувати, вставляти, оновлювати і видаляти календарі, події, учасників, нагадування і т. д.

API постачальника календаря може використовуватися як додатки, так і адаптери синхронізації. Правила залежать від типу програми, яка виконує виклики. У цьому розділі головним чином розглядається використання API постачальника календаря як додатку.

Зазвичай, щоб переглянути або записати дані календаря, в маніфесті додатка повинні бути включені відповідні дозволи. Щоб спростити виконання часто використовуваних операцій, в постачальника календаря передбачений набір намірів. Ці наміри дозволяють користувачам переходити в додаток календаря для вставки, перегляду і редагування подій. Користувач після взаємодії з календарем повертається у вихідний додаток. Тому вашому додатку не треба запитувати дозволу, а також надавати призначений для користувача інтерфейс для перегляду або створення подій.

4.4.1. Основи

Постачальники контенту зберігають в собі дані і надають до них доступ для додатків. Постачальники контенту, запропоновані платформою Android (включаючи постачальник календаря), зазвичай подають дані у вигляді набору таблиць, в основі яких лежить модель реляційної бази даних. Кожен ря-

док в такій таблиці являє собою запис, а кожен стовпець – дані певного типу і значення. Завдяки API постачальника календаря програми та адаптери синхронізації отримують доступ на читання / запис до таблиць в базі даних, в яких наведені дані календаря користувача.

Кожен постачальник календаря надає загальнодоступний URI (упакований в об'єкті `Uri`), який служить унікальним ідентифікатором свого набору даних. Постачальник контенту, який керує кількома наборами даних (кілька таблиць), надає окремий URI для кожного набору. Всі URI постачальників починаються з рядка: `//`. Він визначає дані, які знаходяться під управлінням постачальника контенту. Постачальник календаря задає константи для URI кожного зі своїх класів (таблиць). Такі URI мають формат `<class>.CONTENT_URI`. Наприклад, `Events.CONTENT_URI`.

На рис. 4.1 зображено графічне подання моделі даних постачальника календаря. На ньому наведено основні таблиці і поля, які пов'язують їх один з одним.

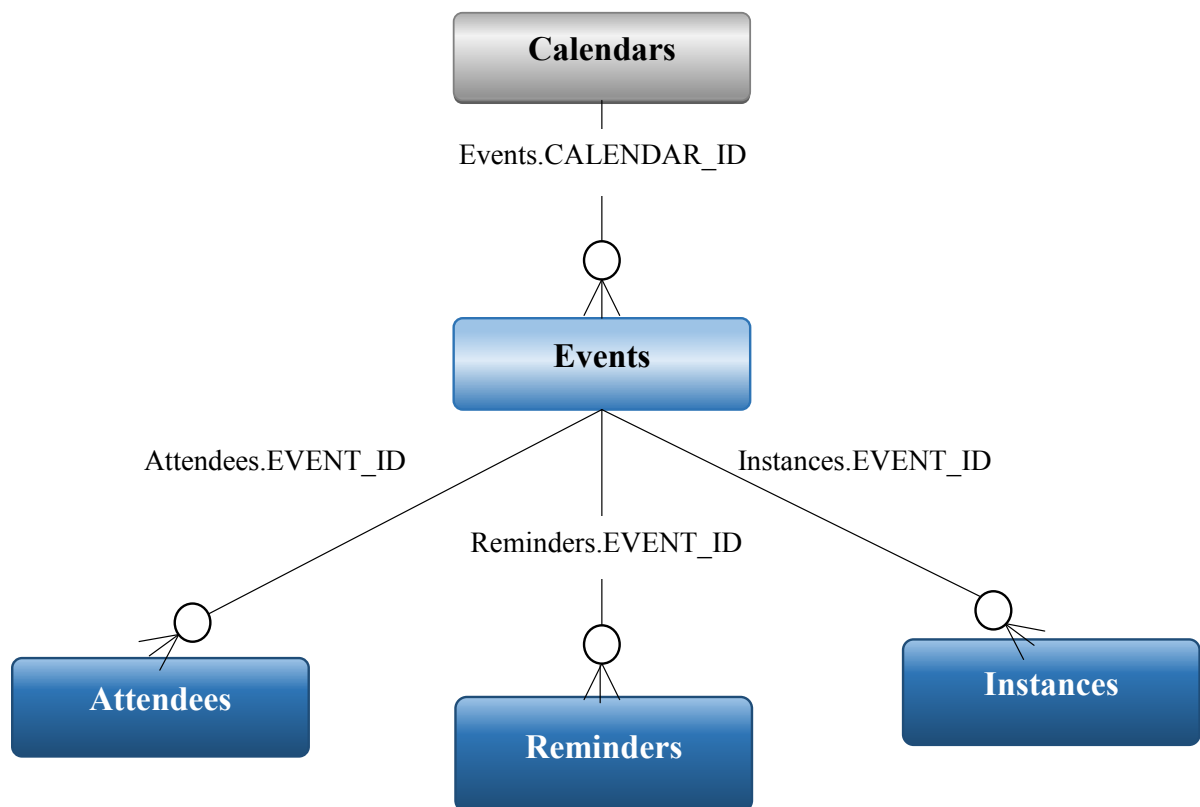


Рисунок 4.1 – Модель даних постачальника календаря

У користувача може бути кілька календарів, причому вони можуть бути пов'язані з акаунтами різних типів (Google Календар, Обмін і т. д.).

Клас **CalendarContract** визначає модель даних календаря і інформацію, що відноситься до подій. Ці дані зберігаються в різних таблицях, зазначених нижче.

Таблиця 4.4 – Модель даних календаря і події

Таблиця (клас) CalendarContract	Опис
1	2
.Calendars	У цій таблиці знаходиться інформація про календарі. У кожному рядку цієї таблиці наведені відомості про окремий календар, наприклад, його назва, колір, інформація про синхронізацію і т. д.
.Events	У цій таблиці знаходиться інформація про події. У кожному рядку цієї таблиці міститься інформація про окрему подію, наприклад, заголовок події, місце проведення, час початку, час завершення і т. д. Подія може бути одноразовою або повторюваною. Відомості про учасників, нагадування і розширені властивості зберігаються в окремих таблицях. У кожній з них є цілочисельна змінна EVENT_ID , яка посилається на об'єкт ID в таблиці подій
.Instances	У цій таблиці містяться дані про час початку і закінчення кожної повторюваної події. У кожному рядку цієї таблиці наведено одне повторення події. Одноразові події зіставляються з повтореннями один до одного. Для повторюваних подій автоматично створюються кілька рядків, що відповідають декільком повторенням події.
.Attendees	У цій таблиці знаходиться інформація про учасників (гостей). У кожному рядку цієї таблиці вказаний один гість. У ній вказується тип гостя і інформація про те, чи відвідає він подію
.Reminders	У цій таблиці знаходиться дані повідомлень або оповіщень. У кожному рядку цієї таблиці вказано одне повідомлення або оповіщення. Для однієї події можна створити кілька нагадувань. Максимальна кількість таких нагадувань для події задається за допомогою цілочисельної змінної MAX_REMINDERS , значення якої задає адаптер синхронізації, що володіє вказаним календарем. Нагадування задається в хвилинах до початку події і має метод, який визначає порядок повідомлення користувача

API постачальника календаря забезпечує достатню гнучкість і ефективність. У той же час важливо надати інтерфейс, який буде зручним для користувача, і забезпечити захист цілісності календаря і його даних. Тому існує ряд моментів, які слід враховувати при використанні цього API.

1. Вставка, оновлення і перегляд подій календаря. Щоб вставити, змінити і переглянути події безпосередньо з постачальника календаря, потрібні відповідні дозволи. Однак якщо не планується створювати повнофункціональний додаток календаря або адаптер синхронізації, запитувати такі дозволи не обов'язково. Замість цього можна використовувати намір, підтримуваний додатком «Календар» Android, для обробки операцій читання і запису в цьому додатку. При використанні намірів ваш додаток відправляє користувачам додаток «Календар» для виконання необхідної операції в попередньо заповненій формі. По її завершенні вони повертаються до додатку. Завдяки можливості виконання часто використовуваних операцій через додаток «Календар», для користувачів забезпечується однаковий і функціональний користувацький інтерфейс. Рекомендуємо використовувати саме такий підхід.

2. Адаптери синхронізації. Адаптер синхронізації синхронізує дані календаря на пристрої користувача з даними на сервері або в іншому джерелі даних. У таблицях `CalendarContract.Calendars` і `CalendarContract.Events` є стовпці, зарезервовані для адаптерів синхронізації. Ні постачальник, ні програми не повинні змінювати їх. Фактично вони приховані доти, поки адаптер синхронізації не почне використовувати їх.

4.4.2. Дозволи користувачів

Щоб прочитати дані календаря, у файл маніфесту додатка необхідно включати дозвіл `READ_CALENDAR`. Також в нього слід включити дозвіл `WRITE_CALENDAR` для видалення, вставки або поновлення даних календаря:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"...>
  <uses-sdk android:minSdkVersion="14" />
  <uses-permission an-
droid:name="android.permission.READ_CALENDAR" />
  <uses-permission an-
droid:name="android.permission.WRITE_CALENDAR" />
  ...
</manifest>
```

4.4.3. Таблиця календарів

У таблиці `CalendarContract.Calendars` містяться докладні відомості про кожний окремий календар (табл. 4.5). Виконувати запис у зазначених нижче стовпцях цієї таблиці може і додаток, і адаптер синхронізації. Повний список підтримуваних полів наведено в довідці по класу `CalendarContract.Calendars`.

Таблиця 4.5 – Відомості про календарі

Константа	Опис
NAME	Назва календаря
CALENDAR_DISPLAY_NAME	Назва цього календаря, яка відображається для користувача
VISIBLE	Логічне значення, що показує, чи вибраний календар для відображення. Значення «0» вказує на те, що події, пов'язані з цим календарем, не відображаються. Значення «1» вказує на те, що події, пов'язані з цим календарем, відображаються. Це значення впливає на створення рядків в таблиці <code>CalendarContract.Instances</code>
SYNC_EVENTS	Логічне значення, що позначає, чи слід синхронізувати календар і зберігати наявні в ньому події на пристрої. Значення «0» вказує, що не слід синхронізувати цей календар або зберігати наявні в ньому події на пристрої. Значення «1» вказує, що цей календар слід синхронізувати і зберігати наявні в ньому події на пристрої

Запит календаря. Нижче наведено приклад того, як отримати календарі, якими володіє певний користувач. Для простоти демонстрації операція запиту в цьому прикладі знаходиться в потоці призначеного для користувача інтерфейсу («основний потік»). На практиці це слід робити в асинхронному потоці, а не в основному. Якщо ж ви не тільки зчитуєте дані, але і вносите в них зміни, зверніться до довідки по класу `AsyncQueryHandler`.

```
// Масив проєкції. Створення індексів для цього масиву, замість
// динамічних пошуків підвищує продуктивність.
public static final String[] EVENT_PROJECTION = new String[] {
    Calendars._ID,                      // 0
    Calendars.ACCOUNT_NAME,             // 1
    Calendars.CALENDAR_DISPLAY_NAME,    // 2
    Calendars.OWNER_ACCOUNT             // 3
};

// Індeksi масиву проєкції вище.
private static final int PROJECTION_ID_INDEX = 0;
private static final int PROJECTION_ACCOUNT_NAME_INDEX = 1;
```



```
private static final int PROJECTION_DISPLAY_NAME_INDEX = 2;
private static final int PROJECTION_OWNER_ACCOUNT_INDEX = 3;
```

В наступній частині прикладу створюється запит. За допомогою вибору визначаються критерії для запиту. У цьому прикладі виконується пошук календарів з такими значеннями параметрів: ACCOUNT_NAME – sampleuser@google.com, ACCOUNT_TYPE – com.google та OWNER_ACCOUNT – sampleuser@google.com. Для перегляду всіх переглянутих користувачем календарів, а не тільки наявних у нього, не вказуйте параметр OWNER_ACCOUNT. Запит повертає об'єкт Cursor, який можна використовувати для перебору результатів.

Повернутий запит до бази даних.

```
// Виконати запит
Cursor cur = null;
ContentResolver cr = getContentResolver();
Uri uri = Calendars.CONTENT_URI;
String selection = "(" + Calendars.ACCOUNT_NAME + " = ?) AND ("
                    + Calendars.ACCOUNT_TYPE + " = ?) AND ("
                    + Calendars.OWNER_ACCOUNT + " = ?))";
String[] selectionArgs = new String[] {"sampleuser@gmail.com",
    "com.google",
    "sampleuser@gmail.com"};
// Відправити запит і отримати об'єкт курсору назад.
cur = cr.query(uri, EVENT_PROJECTION, selection, selectionArgs,
    null);
```

У наступному розділі коду виконується покроковий огляд набору результатів за допомогою курсора. У ньому використовуються константи, які були задані на початку прикладу, для отримання значень для кожного з полів.

```
// Використовуйте курсор для покрокового повернутих записів
while (cur.moveToNext()) {
    long calID = 0;
    String displayName = null;
    String accountName = null;
    String ownerName = null;

    // Отримати значення полів
    calID = cur.getLong(PROJECTION_ID_INDEX);
    displayName = cur.getString(PROJECTION_DISPLAY_NAME_INDEX);
    accountName = cur.getString(PROJECTION_ACCOUNT_NAME_INDEX);
    ownerName = cur.getString(PROJECTION_OWNER_ACCOUNT_INDEX);

    // Зробити що-небудь зі значеннями...

    ...
}
```

4.4.4. Навіщо необхідно вказувати параметр `ACCOUNT_TYPE`?

При створенні запиту `Calendars.ACCOUNT_NAME` необхідно також вказати `Calendars.ACCOUNT_TYPE`. Це необхідно зробити з огляду на те, що вказаний акаунт вважається унікальним тільки тоді, коли для нього вказані і параметр `ACCOUNT_NAME`, й параметр `ACCOUNT_TYPE`. Параметр `ACCOUNT_TYPE` в рядку позначає структуру перевірки існування акаунта, яка використовувалася при реєстрації облікового запису за допомогою `AccountManager`. Існує також особливий тип акаунтів, званий `ACCOUNT_TYPE_LOCAL`. Він використовується для календарів, які не пов'язані з членством пристрою. Акаунти `ACCOUNT_TYPE_LOCAL` не синхронізуються.

Зміна календаря. Щоб оновити календар, можна вказати `_ID` календаря: або у вигляді ідентифікатора, доданого до URI (`withAppendedId()`), або як перший елемент виділення. Виділення повинне починатися з «`_id=?`», а перший аргумент `selectionArg` повинен бути `_ID` календаря. Також для виконання оновлень можна закодувати ідентифікатор в URI. У цьому прикладі для редагування Вашого календаря використовується підхід (`withAppendedId()`):

```
private static final String DEBUG_TAG = "MyActivity";
...
long calID = 2;
ContentValues values = new ContentValues();
// Нова дисплей назва для календаря
values.put(Calendars.CALENDAR_DISPLAY_NAME, "Trevor's Calendar");
Uri updateUri = ContentUris.withAppendedId(Calendars.CONTENT_URI,
calID);
int rows = getContentResolver().update(updateUri, values, null,
null);
Log.i(DEBUG_TAG, "Rows updated: " + rows);
```

Вставка календаря. Для управління календарями в основному використовуються адаптери синхронізації, тому нові календарі слід вставляти виключно як адаптер синхронізації. Здебільшого додатки можуть вносити в календарі тільки поверхневі зміни, такі, як зміна імен. Якщо додатку потрібно створити локальний календар, це можна зробити шляхом вставки календаря у вигляді адаптера синхронізації за допомогою параметра `ACCOUNT_TYPE` типу `ACCOUNT_TYPE_LOCAL.ACCOUNT_TYPE_LOCAL` являє собою особливий тип акаунтів для календарів. Календарі цього типу не синхронізуються з сервером.

4.4.5. Таблиця подій

У таблиці `CalendarContract.Events` містяться докладні відомості про кожну окремому подію. Щоб отримати можливість додавати, оновлювати або видаляти події, у файл маніфесту додатка необхідно включати дозвіл `WRITE_CALENDAR`.

Виконувати запис в зазначені нижче стовпці цієї таблиці можуть і додаток, і адаптер синхронізації. Повний список підтримуваних полів наведено в довідці по класу `CalendarContract.Events`.

Таблиця 4.6 – Відомості про календар

Константа	Опис
<code>CALENDAR_ID</code>	<code>_ID</code> календаря, до якого належить подія
<code>ORGANIZER</code>	Адреса ел. пошти організатора (власника) події
<code>TITLE</code>	Назва події
<code>EVENT_LOCATION</code>	Місце проведення
<code>DESCRIPTION</code>	Опис події
<code>DTSTART</code>	Час початку події за UTC (в мілісекундах) від точки відліку
<code>DTEND</code>	Час закінчення події по UTC (в мілісекундах) від точки відліку
<code>EVENT_TIMEZONE</code>	Часовий пояс події
<code>EVENT_END_TIMEZONE</code>	Часовий пояс для часу закінчення події
<code>DURATION</code>	Тривалість події в форматі RFC5545. Наприклад, значення "PT1H" означає, що подія має тривати одну годину, а значення "P2W" вказує на тривалість в 2 тижні
<code>ALL_DAY</code>	Значення «1» означає, що ця подія триває весь день за місцевим часовим поясом. Значення «0» вказує на те, що це регулярна подія, яке може початися і завершитися в будь-який час протягом дня
<code>RRULE</code>	Правило повторення для формату події. Наприклад, "FREQ=WEEKLY;COUNT=10;WKST=SU". З іншими прикладами можна ознайомитися тут
<code>RDATE</code>	Дати повторення події. Зазвичай <code>RDATE</code> використовується разом з <code>RRULE</code> для визначення агрегованого набору повторюваних подій. Додаткові відомості наведено в специфікації RFC5545
<code>AVAILABILITY</code>	Подія вважається як зайнята або як вільний час, доступний для планування
<code>GUESTS_CAN_MODIFY</code>	Вказує, чи можуть гості вносити зміни в подію
<code>GUESTS_CAN_INVITE_OTHERS</code>	Вказує, чи можуть гості запрошувати інших гостей
<code>GUESTS_CAN_SEE_GUESTS</code>	Вказує, чи можуть гості переглядати список учасників

Додавання подій. Коли ваш додаток вставляє нову подію, ми рекомендуємо використовувати намір `INSERT`. Однак за необхідності ви можете вставляти події безпосередньо.

Правила, якими слід керуватися для вставки нової події, такі:

1. Необхідно вказати `CALENDAR_ID` й `DTSTART`.
2. Необхідно вказати `EVENT_TIMEZONE`. Щоб отримати список встановлених в системі ідентифікаторів часових поясів, скористайтеся методом `getAvailableIDs()`. Зверніть увагу, що це правило не застосовується при вставці події за допомогою наміру `INSERT`, – в цьому випадку використовується часовий пояс за замовчуванням.
3. Для одноразових подій необхідно вказати `DTEND`.
4. Для повторюваних подій необхідно вказати `DURATION` на додаток до `RRULE` чи `RDATE`. Зверніть увагу, що це правило не застосовується при вставці події за допомогою наміру `INSERT`, – в цьому випадку можна використовувати `RRULE` в поєднанні з `DTSTART` й `DTEND`; крім того, додаток «Календар» автоматично перетворює вказаний період тривалості.

Нижче наведено приклад вставки події. Для простоти все це виконується в потоці призначеного для користувача інтерфейсу. На практиці ж всі вставки і оновлення слід виконувати в асинхронному потоці, щоб перемістити операцію в фоновому потоці. Додаткові відомості наведено в довідці по `AsyncQueryHandler`.

```
long calID = 3;
long startMillis = 0;
long endMillis = 0;
Calendar beginTime = Calendar.getInstance();
beginTime.set(2012, 9, 14, 7, 30);
startMillis = beginTime.getTimeInMillis();
Calendar endTime = Calendar.getInstance();
endTime.set(2012, 9, 14, 8, 45);
endMillis = endTime.getTimeInMillis();
...

ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
values.put(Events.DTSTART, startMillis);
values.put(Events.DTEND, endMillis);
values.put(Events.TITLE, "Jazzercise");
values.put(Events.DESCRPTION, "Group workout");
values.put(Events.CALENDAR_ID, calID);
values.put(Events.EVENT_TIMEZONE, "America/Los_Angeles");
Uri uri = cr.insert(Events.CONTENT_URI, values);
```

```
// get the event ID that is the last element in the Uri
long eventID = Long.parseLong(uri.getLastPathSegment());
//
// ... do something with event ID
//
//
```

Примітка. Нижче демонструється, як в прикладі коду виконується захоплення ідентифікатора події після створення цієї події. Це найпростіший спосіб отримати ідентифікатор події. Найчастіше ідентифікатор події необхідний для виконання інших дій з календарем – наприклад, для додавання учасників або нагадувань про подію.

Оновлення подій. Коли ваш додаток хоче надати користувачеві можливість змінити подію, ми рекомендуємо використовувати намір EDIT. Однак за необхідності можна редагувати події безпосередньо. Щоб оновити подію, можна вказати `_ID` події: або у вигляді ідентифікатора, доданого до URI (`withAppendedId()`), або як перший елемент виділення. Виділення повинне починатися з `"_id=?"`, а першим аргументом `selectionArg` повинен бути `_ID` події. Також можна оновлювати виділення без ідентифікаторів. Нижче наведений приклад поновлення події. Це приклад зміни назви події за допомогою методу `withAppendedId()`:

```
private static final String DEBUG_TAG = "MyActivity";
...
long eventID = 188;
...
ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
Uri updateUri = null;
// Нова назва для події
values.put(Events.TITLE, "Kickboxing");
updateUri = ContentUris.withAppendedId(Events.CONTENT_URI,
eventID);
int rows = getContentResolver().update(updateUri, values, null,
null);
Log.i(DEBUG_TAG, "Rows updated: " + rows);
```

Видалення подій. Видалити запис можна по його `_ID`, який доданий як ідентифікатор до URI, або за допомогою стандартного виділення. У разі використання доданого ідентифікатора неможливо виконати і виділення. Існує дві версії операції видалення: для додатка і для адаптера синхронізації. При видаленні для додатка в стовпці *deleted* встановлюється значення «1». Цей пра-

пор повідомляє адаптера синхронізації про те, що рядок був видалений і інформацію про видалення слід передати серверу. При видаленні для адаптера синхронізації подія видаляється з бази даних разом з усіма пов'язаними з нею даними. Нижче наведено приклад видалення події для додатка по його `_ID`:

```
private static final String DEBUG_TAG = "MyActivity";
...
long eventID = 201;
...
ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
Uri deleteUri = null;
deleteUri = ContentUris.withAppendedId(Events.CONTENT_URI, eventID);
int rows = getContentResolver().delete(deleteUri, null, null);
Log.i(DEBUG_TAG, "Rows deleted: " + rows);
```

4.4.6. Таблиця учасників

У кожному рядку таблиці `CalendarContract.Attendees` вказано один учасник або гість події. При виклику методу `query()` повертається список учасників для події з заданим `EVENT_ID`. Цей `EVENT_ID` повинен відповідати `_ID` певної події.

У табл. 4.7 вказані поля, доступні для запису. При вставці нового учасника необхідно вказати всі ці поля, крім `ATTENDEE_NAME`.

Таблиця 4.7 – Таблиця учасників

Константа	Опис
<code>EVENT_ID</code>	Ідентифікатор події
<code>ATTENDEE_NAME</code>	Ім'я учасника
<code>ATTENDEE_EMAIL</code>	Адреса ел. пошти учасника
<code>ATTENDEE_RELATIONSHIP</code>	Зв'язок учасника з подією. Один з наступних: <ul style="list-style-type: none"> • <code>RELATIONSHIP_ATTENDEE</code> • <code>RELATIONSHIP_NONE</code> • <code>RELATIONSHIP_ORGANIZER</code> • <code>RELATIONSHIP_PERFORMER</code> • <code>RELATIONSHIP_SPEAKER</code>
<code>ATTENDEE_TYPE</code>	Тип учасника. Один з наступних: <ul style="list-style-type: none"> • <code>TYPE_REQUIRED</code> • <code>TYPE_OPTIONAL</code>
<code>ATTENDEE_STATUS</code>	Статус відвідування події учасником. Один з наступних: <ul style="list-style-type: none"> <code>ATTENDEE_STATUS_ACCEPTED</code> <code>ATTENDEE_STATUS_DECLINED</code> <code>ATTENDEE_STATUS_INVITED</code> <code>ATTENDEE_STATUS_NONE</code> <code>ATTENDEE_STATUS_TENTATIVE</code>

Додавання учасників. Нижче наведено приклад додавання одного учасника події. Зверніть увагу, що потрібно в обов'язковому порядку вказати EVENT_ID.

```
long eventID = 202;
...
ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
values.put(Attendees.ATTENDEE_NAME, "Trevor");
values.put(Attendees.ATTENDEE_EMAIL, "trevor@example.com");
values.put(Attendees.ATTENDEE_RELATIONSHIP, Attendees.RELATIONSHIP_ATTENDEE);
values.put(Attendees.ATTENDEE_TYPE, Attendees.TYPE_OPTIONAL);
values.put(Attendees.ATTENDEE_STATUS, Attendees.ATTENDEE_STATUS_INVITED);
values.put(Attendees.EVENT_ID, eventID);
Uri uri = cr.insert(Attendees.CONTENT_URI, values);
```

Додавання подій. У кожному рядку таблиці CalendarContract.Reminders вказано одне нагадування про подію. При виклику методу query() повертається список нагадування для події з заданим EVENT_ID.

У табл. 4.8 вказані поля, доступні для запису. При вставці нового нагадування необхідно вказати всі ці поля. Зверніть увагу, що адаптери синхронізації задають типи нагадувань, які вони підтримують, в таблиці CalendarContract.Calendars. Докладну інформацію див. в довідці по ALLOWED_REMINDERS.

Таблиця 4.8 – Таблиця нагадувань

Константа	Опис
EVENT_ID	Ідентифікатор події
MINUTES	Час спрацювання повідомлення (в хвилинах) до початку події
METHOD	Метод повідомлення, заданий на сервері. Один з наступного: <ul style="list-style-type: none"> • METHOD_ALERT • METHOD_DEFAULT • METHOD_EMAIL • METHOD_SMS

Додавання нагадувань. Нижче наведено приклад додавання нагадування в подію. Нагадування спрацює за 15 хвилин до початку події:

```
long eventID = 221;
...
```

```

ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
values.put(Reminders.MINUTES, 15);
values.put(Reminders.EVENT_ID, eventID);
values.put(Reminders.METHOD, Reminders.METHOD_ALERT);
Uri uri = cr.insert(Reminders.CONTENT_URI, values);

```

4.4.7. Таблиця екземплярів

У таблиці `CalendarContract.Instances` (табл. 4.9) містяться дані про час початку і закінчення повторень події. У кожному рядку цієї таблиці наведено одне повторення події. Таблиця екземплярів недоступна для запису; вона надає тільки можливість запитувати повторення подій. У ній перераховані деякі з полів, які можна запросити для екземпляра. Зверніть увагу, що часовий пояс задається параметрами `KEY_TIMEZONE_TYPE` та `KEY_TIMEZONE_INSTANCES`.

Таблиця 4.9 – Таблиця екземплярів

Константа	Опис
BEGIN	Час початку екземпляра в форматі UTC (в мілісекундах)
END	Час закінчення екземпляра в форматі UTC (в мілісекундах)
END_DAY	День закінчення екземпляра за юліанським календарем щодо часового поясу додатка «Календар»
END_MINUTE	Хвилина закінчення екземпляра, обчислена від півночі за часовим поясом додатка «Календар». <code>_ID</code> події для цього екземпляра
EVENT_ID	День початку екземпляра за юліанським календарем щодо часового поясу додатка «Календар»
START_DAY	Хвилина початку екземпляра, обчислена від півночі за часовим поясом додатка «Календар»
START_MINUTE	День закінчення екземпляра за юліанським календарем щодо часового поясу додатка «Календар»

Запит таблиці екземплярів. Щоб запросити таблицю екземплярів, необхідно вказати проміжок часу для запиту в URI. У цьому прикладі `CalendarContract.Instances` отримує доступ до поля `TITLE` за допомогою своєї реалізації інтерфейсу `CalendarContract.EventsColumns`. Іншими словами, `TITLE` повертається за допомогою звернення до бази даних, а не шляхом запиту таблиці `CalendarContract.Instances` з необробленими даними:

```

private static final String DEBUG_TAG = "MyActivity";
public static final String[] INSTANCE_PROJECTION = new String[] {

```



```

        Instances.EVENT_ID,        // 0
        Instances.BEGIN,          // 1
        Instances.TITLE           // 2
    };

    // Індeksi масиву проєкцій вище.
    private static final int PROJECTION_ID_INDEX = 0;
    private static final int PROJECTION_BEGIN_INDEX = 1;
    private static final int PROJECTION_TITLE_INDEX = 2;
    ...

    // Вкажіть діапазон дат, який ви хочете шукати для повторювання
    // екземплярів подій
    Calendar beginTime = Calendar.getInstance();
    beginTime.set(2011, 9, 23, 8, 0);
    long startMillis = beginTime.getTimeInMillis();
    Calendar endTime = Calendar.getInstance();
    endTime.set(2011, 10, 24, 8, 0);
    long endMillis = endTime.getTimeInMillis();

    Cursor cur = null;
    ContentResolver cr = getContentResolver();

    // Ідентифікатор повторюваної події, екземпляри якої ви шукаєте
    // в таблиці Примірники
    String selection = Instances.EVENT_ID + " = ?";
    String[] selectionArgs = new String[] {"207"};

    // Побудувати запит з бажаним діапазоном дат.
    Uri.Builder builder = Instances.CONTENT_URI.buildUpon();
    ContentUris.appendId(builder, startMillis);
    ContentUris.appendId(builder, endMillis);

    // Відправити запит
    cur = cr.query(builder.build(),
        INSTANCE_PROJECTION,
        selection,
        selectionArgs,
        null);

    while (cur.moveToNext()) {
        String title = null;
        long eventID = 0;
        long beginVal = 0;

        // Отримати значення полів
        eventID = cur.getLong(PROJECTION_ID_INDEX);
        beginVal = cur.getLong(PROJECTION_BEGIN_INDEX);
        title = cur.getString(PROJECTION_TITLE_INDEX);

        // Зробити що-небудь зі значеннями.
    }

```

```

    Log.i(DEBUG_TAG, "Event: " + title);
    Calendar calendar = Calendar.getInstance();
    calendar.setTimeInMillis(beginVal);
    DateFormat formatter = new SimpleDateFormat("MM/dd/yyyy");
    Log.i(DEBUG_TAG, "Date: " + formatter.
    format(calendar.getTime()));
}
}

```

4.4.8. Наміри календаря

Вашому додатку не потрібно запитувати дозволи на читання і запис даних календаря. Замість цього можна використовувати наміри, підтримувані додатком «Календар» Android, для обробки операцій читання і запису в цьому додатку. У табл. 4.10 вказані наміри, підтримувані постачальником календаря, у табл. 4.11 вказані додаткові дані намірів, які підтримуються постачальником календаря.

Таблиця 4.11 – Додаткові дані наміру

Додаткові дані наміру	Опис
Events.TITLE	Назва події
CalendarContract.EXTRA_EVENT_BEGIN_TIME	Час початку події (в мілісекундах) від епохи
CalendarContract.EXTRA_EVENT_END_TIME	Час закінчення події (в мілісекундах) від епохи
CalendarContract.EXTRA_EVENT_ALLDAY	Логічне значення, яке позначає, що ця подія на весь день. Значення може бути true чи false
Events.EVENT_LOCATION	Місце проведення події
Events.DESCRPTION	Опис події
Intent.EXTRA_EMAIL	Адреси ел. пошти запрошених (через кому)
Events.RRULE	Правило повторення для події
Events.ACCESS_LEVEL	Вказує на те, чи є подія загальнодоступною або закритою
Events.AVAILABILITY	Подія вважається як зайнята або як вільний час, доступний для планування

У наступних розділах зазначено порядок використання даних намірів.

Таблиця 4.10 – Наміри календаря

Дія	URI	Опис	Додаткові дані
VIEW	content://com.android.calendar/time/<ms_since_epoch> Зіслатися на URI також можна за допомогою CalendarContract.CONTENT_URI	Відкриття календаря під час, заданий па- раметром <ms_since_epoch>.	Відсутні
VIEW	content://com.android.calendar/events/<event_id> Зіслатися на URI також можна за допомогою Events.CONTENT_URI	Перегляд події, вказаної за допо- могою <event_id>	CalendarContract.EXTRA_EVENT _BEGIN_TIME
			CalendarContract.EXTRA_EVENT _END_TIME
EDIT	content://com.android.calendar/events/<event_id> Зіслатися на URI також можна за допомогою Events.CONTENT_URI	Редагування події, вказаної за допо- могою <event_id>	CalendarContract.EXTRA_EVENT _BEGIN_TIME
			CalendarContract.EXTRA_EVENT _END_TIME
EDIT	content://com.android.calendar/events	Створення події	Будь-які з додаткових даних, вказа- них в табл. 4.11
INSERT	Events.CONTENT_URI		

Використання наміру для вставки події. За допомогою наміру INSERT ваш додаток може відправляти завдання вставки події прямо в додаток «Календар». Завдяки цьому у файл маніфесту вашого додатка не потрібно включати дозвіл WRITE_CALENDAR.

Коли користувачі працюють з додатком, в якому використовується такий підхід, додаток відправляє їх в «Календар» для завершення додавання події. Намір INSERT використовує додаткові поля для попередньої вказівки у формі відомостей про подію в додатку «Календар». Після цього користувачі можуть скасувати подію, відредагувати форму або зберегти подію в своєму календарі.

Нижче подано фрагмент коду, в якому на 19 січня 2012 р. планується подія, яка буде проходити з 7:30 до 8:30:

```
Calendar beginTime = Calendar.getInstance();
beginTime.set(2012, 0, 19, 7, 30);
Calendar endTime = Calendar.getInstance();
endTime.set(2012, 0, 19, 8, 30);
Intent intent = new Intent(Intent.ACTION_INSERT)
    .setData(Events.CONTENT_URI)
    .putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME,
beginTime.getTimeInMillis())
    .putExtra(CalendarContract.EXTRA_EVENT_END_TIME,
endTime.getTimeInMillis())
    .putExtra(Events.TITLE, "Yoga")
    .putExtra(Events.DESRIPTION, "Group class")
    .putExtra(Events.EVENT_LOCATION, "The gym")
    .putExtra(Events.AVAILABILITY, Events.AVAILABILITY_BUSY)
    .putExtra(Intent.EXTRA_EMAIL, "ro-
wan@example.com,trevor@example.com");
startActivity(intent);
```

Однак слід врахувати деякі моменти, що стосуються цього прикладу коду:

- `Events.CONTENT_URI` задається в ньому як URI;
- для попередньої вказівки у формі відомостей про час події у ній використовуються додаткові поля `CalendarContract.EXTRA_EVENT_BEGIN_TIME` і `CalendarContract.EXTRA_EVENT_END_TIME`. Значення часу повинні бути вказані в форматі UTC та в мілісекундах від епохи;
- для надання списку учасників, розділених комами (їх адреси ел. пошти) ,у ньому використовується додаткове поле `Intent.EXTRA_EMAIL`.

Використання наміру для редагування події. Подію можна відредагувати безпосередньо. Завдяки наміру EDIT додаток, у якого немає дозволу, може делегувати редагування події додатку «Календар». По завершенні редагування події в додатку «Календар» користувачі повертаються у вихідний додаток.

Нижче подано приклад наміру, що задає новий заголовок для вказаної події і що дозволяє користувачам редагувати подію в додатку «Календар».

```
long eventID = 208;
Uri uri = ContentUris.withAppendedId(Events.CONTENT_URI,
eventID);
Intent intent = new Intent(Intent.ACTION_EDIT)
    .setData(uri)
    .putExtra(Events.TITLE, "My New Title");
startActivity(intent);
```

Використання наміру для перегляду даних календаря. Постачальник календаря дозволяє використовувати намір VIEW двома різними способами:

- через відкриття додатка «Календар» на певній даті;
- перегляд події.

Нижче подано приклад відкриття додатку «Календар» на певній даті:

```
// Дата-час, вказані в мілісекундах з початку епохи.
long startMillis;
...
Uri.Builder builder = CalendarContract.CONTENT_URI.buildUpon();
builder.appendPath("time");
ContentUris.appendId(builder, startMillis);
Intent intent = new Intent(Intent.ACTION_VIEW)
    .setData(builder.build());
startActivity(intent);
```

Приклад відкриття події для його перегляду:

```
long eventID = 208;
...
Uri uri = ContentUris.withAppendedId(Events.CONTENT_URI,
eventID);
Intent intent = new Intent(Intent.ACTION_VIEW)
    .setData(uri);
startActivity(intent);
```

4.4.10. Адаптери синхронізації

Існують лише незначні відмінності в тому, як додаток і адаптер синхронізації отримують доступ до постачальника календаря:

- адаптеру синхронізації необхідно вказати, що він є таким, задавши для параметра CALLER_IS_SYNCADAPTER значення **true**;
- адаптеру синхронізації необхідно надати ACCOUNT_NAME і ACCOUNT_TYPE як параметри запиту в URI;

- адаптер синхронізації має доступ на запис до більшої кількості стовпців, ніж додаток або віджет. Наприклад, додаток може змінювати тільки деякі характеристики календаря, такі, як назва, ім'я, настройка видимості і синхронізація. Тоді як адаптер синхронізації має доступ не тільки до цих стовпців, а й до багатьох інших його характеристик, таких, як колір календаря, часовий пояс, рівень доступу, місце розташування і т. д. Однак адаптер синхронізації обмежений заданими їм параметрами ACCOUNT_NAME і ACCOUNT_TYPE.

Нижче подано метод, який можна використовувати, щоб отримати URI для використання разом з адаптером синхронізації:

```
static Uri asSyncAdapter(Uri uri, String account, String
accountType) {
    return uri.buildUpon()

        .appendQueryParameter(android.provider.CalendarContract.CALLER_IS
        _SYNCADAPTER, "true")
        .appendQueryParameter(CalendarContract.ACCOUNT_NAME, account)
        .appendQueryParameter(CalendarContract.ACCOUNT_TYPE,
        accountType).build();
}
```

4.5. Постачальник контактів

Постачальник контактів являє собою ефективний і гнучкий компонент Android, який керує центральним репозиторієм пристрою, в якому зберігаються призначені для користувача дані. Постачальник контактів – це джерело даних, які відображаються в меню «Контакти» на вашому пристрої. Також можна отримати доступ до цих даних в своєму власному додатку і організувати обмін такими даними між пристроєм і службами в Інтернеті. Постачальник взаємодіє з широким набором джерел даних і намагається організувати управління якомога більшим набіром даних про кожну людину, тому організація постачальника досить складна. З цієї причини API постачальника містить широкий набір класів-контрактів та інтерфейси, що відповідають як за отримання даних, так і за їх зміну.

У цьому керівництві розглядаються такі питання:

- основна структура постачальника;
- порядок отримання даних від постачальника;
- порядок зміни даних в постачальнику;
- порядок запису адаптера синхронізації для синхронізації даних, отриманих з вашого сервера, з даними в постачальнику контактів.

При вивченні даного матеріалу маєтеся на увазі, що ви вже знайомі з основами постачальників контенту Android. Приклад адаптера синхронізації є прикладом використання такого додатка для обміну даними між постачальником контактів і додатком, розміщеним в веб-службах Google.

4.5.1. Структура постачальника контактів

Постачальник контактів являє собою постачальник контенту Android. Він містить в собі три типи даних про користувача, кожен з яких вказаний в окремій таблиці, що надається постачальником, як показано на рис. 4.2.

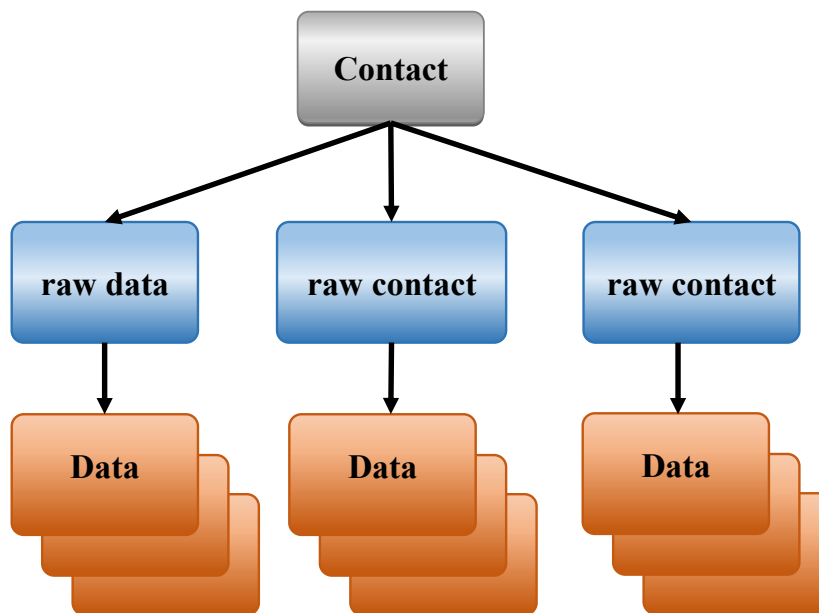


Рисунок 4.2 – Структура таблиці постачальника контактів

Як імена цих трьох таблиць зазвичай використовуються назви відповідних класів-контрактів. Ці класи визначають константи для URI контенту, назв стовпців і значень в стовпцях цих таблиць.

Таблиця **ContactsContract.Contacts**

Рядки в цій таблиці містять дані про різних користувачів, отримані шляхом агрегації рядків необроблених контактів.

Таблиця **ContactsContract.RawContacts**

Рядки в цій таблиці містять зведені дані про користувача, що відноситься до призначеного для користувача акаунта і його типу.

Таблиця **ContactsContract.Data**

Рядки в цій таблиці містять відомості про необроблені контакти, такі, як адреса ел. пошти або номери телефонів.

Інші таблиці, представлені класами-контрактами в **ContactsContract**, являють собою допоміжні таблиці, які постачальник контактів використовує

для управління своїми операціями або для підтримки певних функцій, наявних в додатку пристрою «Контакти» або в додатках для телефонного зв'язку.

4.5.2. Необроблені контакти

Необроблений контакт являє собою дані про користувача, що надходить з одного акаунта певного типу. Оскільки як джерело даних про користувача в постачальнику контактів може виступати відразу кілька онлайн-служб, то для того ж самого користувача в постачальнику контактів може існувати кілька необроблених контактів. Це також дозволяє користувачеві об'єднувати призначені для користувача дані з декількох акаунтів одного того самого типу.

Більша частина даних необробленого контакту не зберігається в таблиці `ContactsContract.RawContacts`. Замість цього вони зберігаються в одній або декількох рядках в таблиці `ContactsContract.Data`. У кожному рядку даних є стовпець `Data.RAW_CONTACT_ID`, в якому міститься значення `android.provider.BaseColumns#_ID` `RawContacts._ID` його батьківського рядка `ContactsContract.RawContacts`.

Важливі стовпці необроблених контактів. У табл. 4.12 вказані стовпці таблиці `ContactsContract.RawContacts`, які мають велике значення. Обов'язково ознайомтеся з примітками, наведеними після цієї таблиці.

Таблиця 4.12 – Важливі стовпці необроблених контактів

Назва стовпця	Використання	Примітки
ACCOUNT_NAME	Ім'я облікового запису для типу акаунта, який виступає в ролі джерела даних для необробленого контакту. Наприклад, ім'я облікового запису Google є одним з ел. адрес пошти Gmail власника пристрою. Додаткові відомості див. у наступному записі ACCOUNT_TYPE	Формат цього імені залежить від типу облікового запису. Це не обов'язково адреса ел. пошти
ACCOUNT_TYPE	Тип акаунта, який виступає в ролі джерела даних для необробленого контакту. Наприклад, тип акаунта Google – <code>com.google</code> . Завжди вказуйте тип акаунта і ідентифікатор домену, яким ви володієте або керуєте. Це дозволить гарантувати унікальність типу вашого облікового запису	Тип акаунта, який виступає у ролі джерела даних контактів, зазвичай має пов'язаний з ним адаптер синхронізації, який синхронізує дані з постачальником контактів

Продовження табл. 4.12

1	2	3
DELETED	Прапорець deleted для необробленого контакту	Цей прапорець дозволяє постачальнику контактів зберігати рядок всередині себе доти, поки адаптери синхронізації не зможуть видалити цей рядок з серверів, а потім видалити його зі сховищ

Примітки

Наведемо важливі примітки до табл. 4.12 `ContactsContract.RawContacts`:

- Ім'я необробленого контакту не зберігається в його рядку в таблиці `ContactsContract.RawContacts`. Замість цього воно зберігається в таблиці `ContactsContract.Data` у рядку `ContactsContract.CommonDataKinds.StructuredName`. У необробленого контакту в таблиці `ContactsContract.Data` є тільки один рядок такого типу.

- **Увага!** Щоб використовувати в рядку необробленого контакту дані власного облікового запису, рядок спочатку необхідно зареєструвати в класі `AccountManager`. Для цього запропонуйте користувачам додати тип акаунта і його ім'я в список акаунтів. Якщо не зробити цього, постачальник контактів автоматично видалить ваш рядок необробленого контакту.

Наприклад, якщо необхідно, щоб у вашому додатку зберігалися дані контактів для веб-служби в домені `com.example.dataservice`, і акаунт користувача служби виглядав таким чином, як `becky.sharp@dataservice.example.com`, то користувачеві спочатку необхідно додати «тип» акаунта (`com.example.dataservice`) і його «ім'я» (`becky.smart@dataservice.example.com`) перш ніж ваш додаток зможе додавати рядки необроблених контактів. Цю вимогу можна вказати в документації для користувачів; також можна запропонувати користувачеві додати тип і ім'я свого облікового запису, або реалізувати обидва ці варіанти.

Джерела даних необроблених контактів. Щоб зрозуміти, що таке необроблений контакт, розглянемо приклад з користувачем Emily Dickinson, на пристрої якої є три наступних акаунти:

- `emily.dickinson@gmail.com`;
- `emilyd@gmail.com`;
- `belle_of_amherst` у Twitter.

Вона включила функцію *Синхронізувати контакти* для всіх трьох цих акаунтів в налаштуваннях *Акаунти*.

Припустимо, що Emily Dickinson відкриває браузер, входить в Gmail під ім'ям `emily.dickinson@gmail.com`, потім відкриває Контакти і додає новий контакт Thomas Higginson. Пізніше вона знову входить в Gmail під ім'ям `emilyd@gmail.com` і відправляє листа користувачеві Thomas Higginson, який автоматично додається в її контакти. Також вона підписана на новини від `colonel_tom` (акаунт користувача Thomas Higginson в Twitter) в Twitter. В результаті цих дій постачальник контактів створює три необроблених контакти:

1. Необроблений контакт Thomas Higginson, пов'язаний з акаунтом `emily.dickinson@gmail.com`. Тип цього акаунта – Google.

2. Другий необроблений контакт Thomas Higginson, пов'язаний з акаунтом `emilyd@gmail.com`. Тип цього акаунта – також Google. Другий необроблений контакт створюється навіть в тому випадку, якщо його ім'я повністю збігається з ім'ям попереднього контакту, оскільки перший був доданий для іншого облікового запису.

3. Третій необроблений контакт Thomas Higginson, пов'язаний з акаунтом `belle_of_amherst`. Тип цього акаунта – Twitter.

4.5.3. Дані

Як вже зазначалося раніше, дані необробленого контакту зберігаються в рядку `ContactsContract.Data`, який пов'язаний зі значенням `_ID` необробленого контакту. Завдяки цьому для одного необробленого контакту може існувати декілька екземплярів того самого типу даних (наприклад, адрес ел. пошти або номерів телефонів). Наприклад, якщо у контакту Thomas Higginson для акаунта `emilyd@gmail.com` (рядок необробленого контакту Thomas Higginson, пов'язаний з обліковим записом Google `emilyd@gmail.com`) є адреса ел. пошти `thigg@gmail.com` і службова адреса ел. пошти `thomas.higginson@gmail.com`, то постачальник контактів зберігає два рядки адреси ел. пошти і пов'язує їх з цим необробленим контактом.

Зверніть увагу, що в цій таблиці зберігаються дані різних типів. Рядки з відомостями про ім'я, номер телефону, адреса ел. пошти, поштова адреса, фото

і веб-сайт, що відображаються, зберігаються в таблиці `ContactsContract.Data`. Для спрощення управління цими даними в таблиці `ContactsContract.Data` передбачені стовпці з описовими іменами, а також інші стовпці з універсальними іменами. Вміст у стовпці з описовим ім'ям має те ж значення, незалежно від типу даних в рядку, тоді як вміст стовпчика з універсальним ім'ям може мати різне значення залежно від типу даних.

Описові імена стовпців. Ось деякі приклади стовпців з описовими іменами:

- `RAW_CONTACT_ID`. Значення в стовпці `_ID` необробленого контакту для цих даних.

- `MIMETYPE`. Тип даних, що зберігаються в цьому рядку, у вигляді типу MIME, що настраюється. Постачальник контактів використовує типи MIME, задані в підкласах класу `ContactsContract.CommonDataKinds`. Ці типи MIME є відкритими, і їх може використовувати будь-який додаток або адаптер синхронізації, що підтримує роботу з постачальником контактів.

- `IS_PRIMARY`. Якщо цей тип даних для необробленого контакту зустрічається кілька разів, то стовпець `IS_PRIMARY` позначає прапорцем рядки даних, в яких містяться основні дані для цього типу. Наприклад, якщо користувач натиснув і утримує номер телефону контакту і вибрав параметр **Використовувати за замовчуванням**, то в стовпці `ContactsContract.Data` з цим номером телефону у відповідному стовпці `IS_PRIMARY` задається значення, відмінне від нуля.

Універсальні імена стовпців. Існує 15 загальнодоступних стовпців з універсальними іменами (`DATA1–DATA15`) і чотири додаткових стовпчики (`SYNC1–SYNC4`), які використовуються тільки адаптерами синхронізації. Константи стовпців з універсальними іменами застосовуються завжди, незалежно від типу даних, що містяться в стовпці.

Стовпець `DATA1` є індексованим. Постачальник контактів завжди використовує цей стовпець для даних, які, як він очікує, будуть цільовими в запитах. Наприклад, в рядку з адресами ел. пошти в цьому стовпці вказується фактична адреса ел. пошти.

Зазвичай стовпець `DATA15` зарезервований для даних великих двійкових об'єктів (BLOB), таких, як мініатюри фотографій.

Імена стовпців за типами строк. Для спрощення роботи за допомогою стовпців певного типу рядків у постачальнику контактів також передбачені константи для імен стовпців за типами рядків, які визначені в підкласах класу `ContactsContract.CommonDataKinds`. Ці константи просто привласнюють тому самому імені стовпця різні константи, що дозволяє вам отримувати доступ до даних в рядку певного типу.

Наприклад, клас `ContactsContract.CommonDataKinds.Email` визначає константи імені стовпця для рядка `ContactsContract.Data`, у якому є тип MIME `Email.CONTENT_ITEM_TYPE`. У цьому класі міститься константа `ADDRESS` для стовпця адреси ел. пошти. Фактичне значення `ADDRESS` – `data1`, яке збігається з універсальним ім'ям стовпця.

Увага! Не додавайте свої дані, що настроюються, в таблицю `ContactsContract.Data` за допомогою рядка, в якому є один із попередньо визначених постачальником типів MIME. В іншому випадку можна втратити дані або викликати несправності в роботі постачальника. Наприклад, не слід додавати рядок з типом MIME `Email.CONTENT_ITEM_TYPE`, у якому в стовпці **DATA1** замість адреси електронної пошти міститься ім'я користувача. Якщо вкажете в рядку власний тип MIME, що настроюється, можна вільно вказувати власні імена стовпців за типами рядків і використовувати їх так, як забажаєте.

На рис. 4.3 показано, як стовпці з описовими іменами і стовпці даних відображаються у рядку `ContactsContract.Data`, а також як імена стовпців за типом рядків «накладаються» на універсальні імена стовпців.

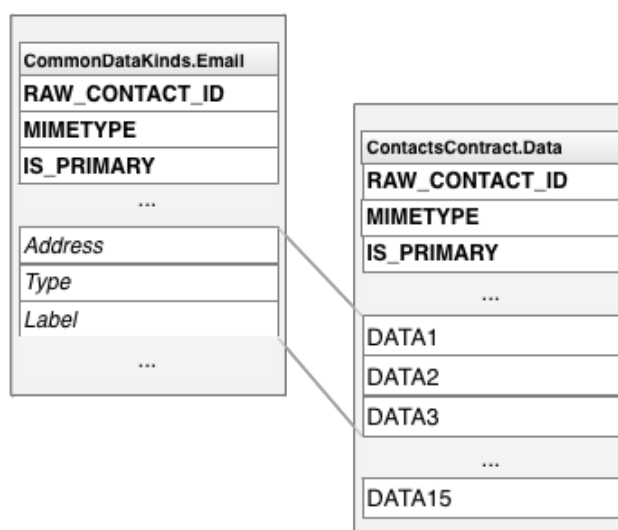


Рисунок 4.3 – Імена стовпців за типами рядків та універсальні імена стовпців

Класи імен стовпців за типами строк. У табл. 4.13 перераховані класи імен стовпців за типами рядків, які частіше за все використовуються.

Таблиця 4.13 – Класи імен стовпців за типами рядків

Клас зіставлення	Тип даних	Примітки
ContactsContract.CommonDataKinds.StructuredName	Дані про ім'я необробленого контакту, пов'язаного з цим рядком даних	У необробленого контакту є тільки один рядок такого типу
ContactsContract.CommonDataKinds.Photo	Основна фотографія необробленого контакту, пов'язаного з цим рядком даних	У необробленого контакту є тільки один рядок такого типу
ContactsContract.CommonDataKinds.Email	Адреса ел. пошти необробленого контакту, пов'язаного з цим рядком даних	У необробленого контакту може бути декілька адрес ел. пошти
ContactsContract.CommonDataKinds.StructuredPostal	Поштова адреса необробленого контакту, пов'язаного з цим рядком даних	У необробленого контакту може бути декілька поштових адрес
ContactsContract.CommonDataKinds.GroupMembership	Ідентифікатор, за допомогою якого необроблений контакт пов'язаний із однією з груп у постачальнику контактів	Групи є обов'язковим компонентом для типу акаунта і імені акаунта

Контакти. Постачальник контактів об'єднує рядки з необробленими контактами для всіх типів акаунтів і імен акаунтів для створення **контакту**. Це дозволяє спростити відображення і зміну всіх даних, зібраних щодо користувача. Постачальник контактів управляє процесом створення рядків контактів і агрегуванням необроблених контактів, у яких є рядок контакту. Ні додаткам, ні адаптерам синхронізації не дозволяється додавати контакти, а деякі стовпці в рядку контакту доступні тільки для читання.

Примітка. Якщо спробувати додати контакт в постачальник контактів за допомогою методу `insert()`, буде видано виняток

`UnsupportedOperationException`. Якщо ви спробуєте оновити стовпець, доступний тільки для читання, цю дію буде проігноровано.

При додаванні нового необробленого контакту, який не відповідає жодному з існуючих контактів, постачальник контактів створює новий контакт. Постачальник чинить аналогічно у разі, якщо дані в рядку існуючого необробленого контакту змінюються таким чином, що вони більше не відповідають контакту, з яким вони раніше були пов'язані. При створенні додатком або адаптером синхронізації нового контакту, який відповідає існуючому контакту, новий контакт об'єднується з існуючим контактом.

Постачальник контактів пов'язує рядок контакту з його рядками необробленого контакту за допомогою стовпчика `_ID` рядка контакту у таблиці `Contacts`. Стовпець `CONTACT_ID` в таблиці необроблених контактів `ContactsContract.RawContacts` містить значення `_ID` для рядка контактів, пов'язаного з кожним рядком необроблених контактів.

У таблиці `ContactsContract.Contacts` також є стовпець `android.provider.ContactsContract.ContactsColumns#LOOKUP_KEY`, який виступає у ролі «постійного посилання» на рядок контакту. Оскільки постачальник контактів автоматично зберігає контакти, у відповідь на агрегування або синхронізацію він може змінити значення `android.provider.BaseColumns#_ID` рядка контакту. Навіть якщо це станеться, `URI` контенту `CONTENT_LOOKUP_URI`, об'єднаний з `android.provider.ContactsContract.ContactsColumns#LOOKUP_KEY` контакту, буде, як і раніше, вказувати на рядок контакту, тому можна сміливо використовувати `android.provider.ContactsContract.ContactsColumns#LOOKUP_KEY` для збереження посилань на «обрані» контакти та ін. Стовпець має власний формат, який не пов'язаний з форматом стовпчика `android.provider.BaseColumns#_ID`.

На рис. 4.4 показано взаємозв'язок цих трьох основних таблиць.

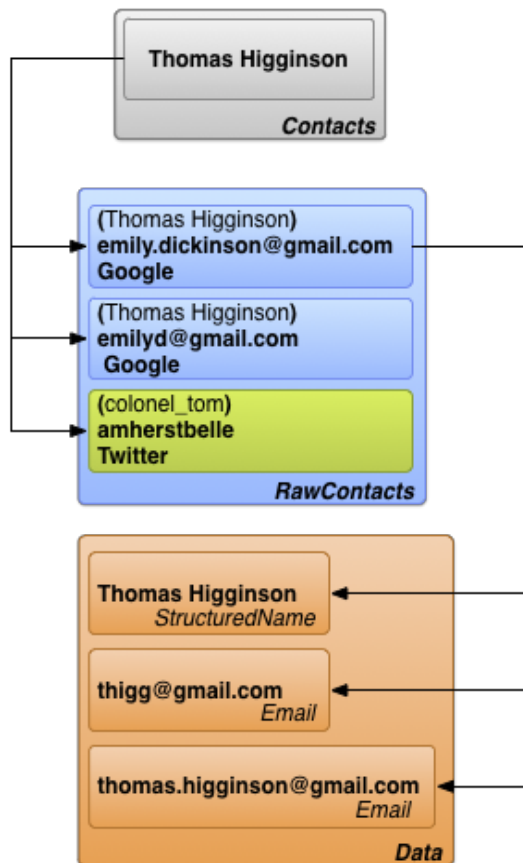


Рисунок 4.4 – Взаємозв’язок між таблицями контактів, необроблених контактів і відомостей

4.5.4. Дані, отримані від адаптера синхронізації

Користувач вводить дані контактів прямо на пристрої, проте дані також надходять в постачальник контактів з веб-служб за допомогою **адаптерів синхронізації**, що дозволяє автоматизувати обмін даними між пристроєм і службами в Інтернеті. Адаптери синхронізації виконуються у фоновому режимі під управлінням системи, і для управління даними вони викликають методи `ContentResolver`.

В Android веб-служба, з якою працює адаптер синхронізації, визначається за типом акаунта. Кожен адаптер синхронізації працює з одним типом акаунта, проте він може підтримувати кілька імен акаунтів такого типу. Зазначені нижче визначення дозволяють точніше охарактеризувати зв’язок між типами і іменами акаунтів і адаптерами синхронізації і службами.

Тип акаунта. Визначає службу, в якій користувач зберігає дані. У більшості випадків для роботи зі службою користувачеві необхідно пройти перевірку справжності. Наприклад, Google Контакти являє собою тип акаунтів, що

позначається кодом google.com. Це значення відповідає типу акаунта, використовуваного AccountManager.

Ім'я акаунта. Визначає конкретний акаунт або ім'я для входу для типу акаунта. Акаунти «Контакти Google» – це те саме, що й акаунти Google, в якості імені яких використовується адреса ел. пошти. В інших службах може використовуватися ім'я користувача, що складається з одного слова, або числовий ідентифікатор.

Типи акаунтів не обов'язково повинні бути унікальними. Користувач може створити кілька облікових записів Google Контакти та завантажити дані з них у постачальник контактів; це може статися у разі, якщо у користувача є один набір персональних контактів для особистого облікового запису, а інший набір – для службового акаунта. Імена акаунтів зазвичай унікальні. Разом вони формують певний потік даних між постачальником контактів і зовнішніми службами.

Якщо необхідно передати дані зі служби в постачальник контактів, то треба створити власний адаптер синхронізації.

На рис. 4.5 показано, яку роль виконує постачальник контактів в потоці передачі даних про користувачів. В області, зазначеній на цьому рисунку як *sync adapters*, кожен адаптер промаркований відповідно до підтримуваних ним типом акаунтів.

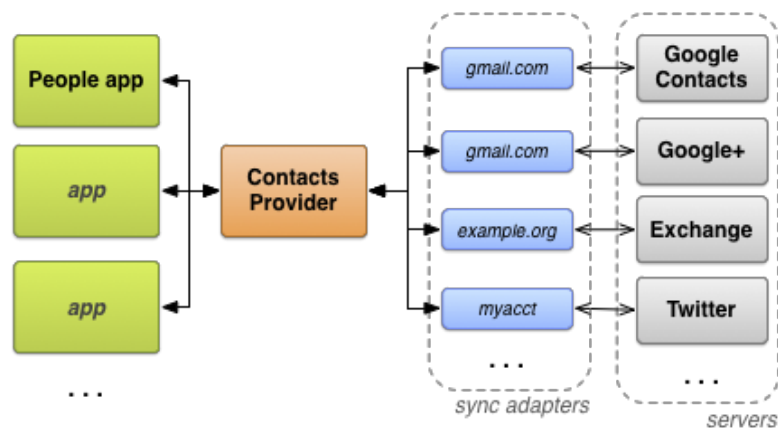


Рисунок 4.5 – Потік передачі даних через постачальника контактів

4.5.5. Необхідні дозволи

Додатки, яким потрібен доступ до постачальника контактів, повинні запросити такі дозволи:

– *Доступ на читання однієї або декількох таблиць.* READ_CONTACTS, вказаний у файлі AndroidManifest.xml з елементом <uses-permission> у

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

– *Доступ на запис в одну або кілька таблиць.* `WRITE_CONTACTS`, вказаний у файлі `AndroidManifest.xml` з елементом `<uses-permission>` у вигляді `<uses-permission android:name="android.permission.WRITE_CONTACTS">`.

Ці дозволи не поширюються на роботу з даними профілю користувача.

Пам'ятайте, що дані про контакти користувача є особистими і конфіденційними. Користувачі піклуються про свою конфіденційність, а тому не хочуть, щоб програми збирали дані про них самих або їхні контакти. Якщо користувачеві не до кінця ясно, для чого потрібен дозвіл на доступ до даних контактів, вони можуть привласнити вашому додатку низький рейтинг або взагалі відмовляться його встановлювати.

4.5.6. Профіль користувача

У таблиці `ContactsContract.Contacts` є всього один рядок, що містить дані профілю для пристрою користувача. Ці дані описують `user` пристрою, а не один з контактів користувача. Рядок контактів профілю пов'язаний з рядком необроблених контактів в кожній із систем, в якій використовується профіль. У кожному рядку необробленого контакту профілю може міститися декілька рядків даних. Константи для доступу до профілю користувача подані в класі `ContactsContract.Profile`.

Для доступу до профілю користувача потрібні особливі дозволи. Крім дозволів `READ_CONTACTS` та `WRITE_CONTACTS`, які потрібні для читання і запису, щоб отримати доступ до профілю користувача, необхідні дозволи `android.Manifest.permission#READ_PROFILE` і `android.Manifest.permission#WRITE_PROFILE` на читання і запис відповідно.

Завжди слід пам'ятати, що профіль користувача являє собою конфіденційну інформацію. Дозвіл `android.Manifest.permission #READ_PROFILE` надає вам доступ до особистої інформації на пристрої користувача. В описі свого додатку обов'язково вкажіть, для чого вам потрібен доступ до профілю користувача.

Щоб отримати рядок контакту з профілем користувача, викличте метод `ContentResolver.query()`. Задайте для `URI` контенту значення `CONTENT_URI` і не вказуйте ніяких критеріїв вибірки. Цей `URI` контенту також можна використовувати як основний `URI` для отримання необроблених контактів або деталей свого профілю. Нижче подано фрагмент коду для отримання даних профілю:

```
// Встановить стовпці для вилучення для профілю користувача
mProjection = new String[]
{
    Profile._ID,
    Profile.DISPLAY_NAME_PRIMARY,
    Profile.LOOKUP_KEY,
    Profile.PHOTO_THUMBNAIL_URI
};

// Витягує профіль з контактів постачальника
mProfileCursor =
    getContentResolver().query(
        Profile.CONTENT_URI,
        mProjection ,
        null,
        null,
        null);
```

Примітка. Якщо при витяганні кількох рядків контактів необхідно визначити, який з них є профілем користувача, зверніться до колонки `IS_USER_PROFILE` цього рядка. Якщо в цьому стовпці вказано значення «1», то в цьому рядку знаходиться профіль користувача.

4.5.7. Метадані постачальника контактів

Постачальник контактів управляє даними, які використовуються для відстеження стану контактної інформації в репозиторії. Ці метадані про репозиторії зберігаються в різних місцях, включаючи рядки необроблених контактів, дані і рядки таблиці контактів, таблицю `ContactsContract.Settings` і таблицю `ContactsContract.SyncState`. У табл. 4.14 вказано значення кожного з цих фрагментів метаданих.

4.5.8. Доступ до постачальника контактів

У цьому розділі розглядаються інструкції по отриманню доступу до даних з постачальника контактів, зокрема такі:

- запити об'єктів;
- пакетна зміна;
- отримання і зміна даних за допомогою намірів;
- цілісність даних.

Запит об'єктів. Таблиці постачальника контактів мають ієрархічну структуру, тому часто корисно витягати рядок і всі пов'язані з ним його дочірні рядки. Наприклад, для відображення всієї інформації про користувача ви, можливо, захочете отримати всі рядки `ContactsContract.RawContacts` для одного рядка

`ContactsContract.Contacts` або всі рядки `ContactsContract.CommonDataKinds.Email` для одного рядка `ContactsContract.RawContacts`. Щоб спростити цей процес, в постачальнику контактів є **об’єкти**, які виступають в ролі з’єднувачів бази даних між таблицями.

Таблиця 4.14 – Метадані в постачальнику контактів

Таблиця	Стовпець	Значення	Опис
1	2	3	4
<code>ContactsContract.RawContacts</code>	DIRTY	«0» – з моменту останньої синхронізації зміни не вносилися	Служить для позначки необроблених контактів, які були змінені на пристрої і які необхідно знову синхронізувати з сервером. Значення встановлюється автоматично постачальником контактів при оновленні рядків додатками Android. Адаптери синхронізації, які вносять зміни в необроблені контакти або таблиці даних, повинні завжди додавати до використовуваного ними URI контенту рядок <code>CALLER_IS_SYNCADAPTER</code> . Це дозволяє запобігти позначенню таких рядків постачальником як «брудних». В іншому випадку зміни, внесені адаптером синхронізації, будуть розглядатися як локальні зміни, які підлягають відправленню на сервер, навіть якщо джерелом таких змін був сам сервер
		«1» – з моменту останньої синхронізації були внесені зміни, які необхідно відобразити і на сервері	
<code>ContactsContract.RawContacts</code>	VERSION	Номер версії цього рядка	Постачальник контактів автоматично збільшує це значення при кожній зміні в рядку або в пов’язаних з ним даних
<code>ContactsContract.Data</code>	DATA_VERSION	Номер версії цього рядка	Постачальник контактів автоматично збільшує це значення при кожній зміні в рядку даних
<code>ContactsContract.Groups</code>	GROUP_VISIBLE	«0» – контакти, подані в цій групі, не повинні відображатися в інтерфейсах користувача додатків Android	Цей стовпець призначений для відомостей про сумісність з серверами, що дозволяють користувачам приховувати контакти в певних групах.
		«1» – контакти, подані в цій групі, можуть відображатися в інтерфейсах користувача додатків	

Продовження табл. 4.14

1	2	3	4
ContactsContract.RawContacts	SOURCE_ID	Рядкове значення, яке є унікальним ідентифікатором даного необробленого контакту в акаунті, в якому він був створений	<p>Коли адаптер синхронізації створює новий необроблений контакт, в цьому стовпці слід вказати унікальний ідентифікатор цього необробленого контакту на сервері. Коли ж додаток Android створює новий необроблений контакт, то у додатку слід залишити цей стовпець порожнім. Це служить сигналом для адаптера синхронізації для створення нового необробленого контакту на сервері і отримання значення SOURCE_ID.</p> <p>Зокрема, ідентифікатор джерела повинен бути унікальним для кожного типу акаунта і незмінним при синхронізації.</p> <p>Унікальний: у кожного необробленого контакту для акаунта повинен бути свій власний ідентифікатор джерела. Якщо не застосувати цю вимогу, у вас обов'язково виникнуть проблеми з додатком «Контакти». Зверніть увагу, що два необроблених контакти того самого <i>типу</i> акаунта можуть мати однаковий ідентифікатор джерела. Наприклад, необроблений контакт Thomas Higginson для акаунта emily.dickinson@gmail.com може мати такий же ідентифікатор джерела, що і контакт Thomas Higginson для акаунта emilyd@gmail.com.</p> <p>Стабільний: ідентифікатори джерел являють собою незмінну частину даних онлайн-служби для необробленого контакту. Наприклад, якщо користувач виконує повторну синхронізацію після очищення сховища контактів в налаштуваннях програми, ідентифікатори джерел відновлених необроблених контактів повинні бути такими ж, як і раніше. Якщо не застосувати цю вимогу, перестануть працювати ярлики</p>

Закінчення табл. 4.14

1	2	3	4
<code>ContactsContract.SyncState</code>	(yci)	Ця таблиця використовується для зберігання метаданих для вашого адаптера синхронізації	За допомогою цієї таблиці можна на постійній основі зберігати на пристрої відомості про стан синхронізації та інші пов'язані з синхронізацією дані
<code>ContactsContract.Settings</code>	<code>UNGROUPE_D_VISIBLE</code>	<p>«0» – для цього облікового запису і акаунтів цього типу контакти, які не належать до групи, не відображаються в інтерфейсах користувача додатків Android</p> <p>«1» – для цього облікового запису і акаунтів цього типу контакти, які не належать до групи, відображаються в інтерфейсах користувача додатків Android</p>	За замовчуванням контакти приховані, якщо жоден з їх необроблених контактів не належить групі (належність необробленого контакту до групи вказується в одному або декількох рядках <code>ContactsContract.CommonDataKinds.GroupMembership</code> у таблиці <code>ContactsContract.Data</code>). Встановивши цей прапорець в рядку таблиці <code>ContactsContract.Settings</code> для типу акаунта і імені акаунта, можна примусово зробити видимими контакти, які не належать будь-якій групі. Один з варіантів використання цього прапорця – відображення контактів з серверів, на яких не використовуються групи

Об'єкт являє собою подобу таблиці, що складається з обраних стовпців батьківської таблиці і її дочірньої таблиці. При запиті об'єкта надається проекція і критерії пошуку на основі доступних в об'єкті стовпців. В результаті ви отримуєте об'єкт `Cursor`, в якому міститься один рядок для кожного витягнутого рядка дочірньої таблиці. Наприклад, якщо запросити об'єкт `ContactsContract.Contacts.Entity` для імені контакту і всі рядки `ContactsContract.CommonDataKinds.Email` для всіх необроблених контактів, що відповідають цьому імені, ви отримаєте об'єкт `Cursor`, що містить по одному рядку для кожного рядка `ContactsContract.CommonDataKinds.Email`.

Використання об'єктів спрощує запити. За допомогою об'єкта можна витягти відразу всі дані для контакту або необробленого контакту, замість того, щоб спочатку робити запит до батьківської таблиці для отримання ідентифікатора і подальшого запиту до дочірньої таблиці з використанням отриманого

ідентифікатора. Крім того, постачальник контактів обробляє запит об'єкта за одну транзакцію, що гарантує внутрішню узгодженість отриманих даних.

Примітка. Об'єкт зазвичай містить не всі стовпці батьківської і дочірньої таблиць. Якщо ви спробуєте змінити ім'я стовпця, який відсутній в списку констант імені стовпця для об'єкта, то отримаєте `Exception`.

Нижче наведено приклад коду для отримання всіх рядків необробленого контакту для контакту. Це фрагмент більшого додатка, в якому є дві операції: основна і для отримання відомостей. Основна операція служить для отримання списку рядків контактів; при виборі користувачем контакту операція відправляє його ідентифікатор в операцію для отримання відомостей. Операція для отримання відомостей використовує об'єкт `ContactsContract.Contacts.Entity` для відображення всіх рядків даних, отриманих від усіх необроблених контактів, які пов'язані з обраним контактом.

Фрагмент коду операції «для отримання відомостей»:

```
...
/*
 * Додає шлях суті до URI. У разі постачальника контактів,
 * очікується URI змісту://com.google.contacts/#/entity (# це
 значення ідентифікатора).
 */
mContactUri = Uri.withAppendedPath(
    mContactUri,
    ContactsContract.Contacts.Entity.CONTENT_DIRECTORY);

// Ініціалізовує завантажувач ідентифікованого за LOADER_ID.
getLoaderManager().initLoader(
    LOADER_ID, // Ідентифікатор завантажувача для
ініціалізації
    null,      // Аргументи для завантажувача (в даному ви-
падку, немає)
    this);    // контекст діяльності

// Створює новий адаптер курсора, щоб прикріпити до перегляду
списку
mCursorAdapter = new SimpleCursorAdapter(
    this,          // контекст діяльності
    R.layout.detail_list_item, // вид елемента, що містить
деталі віджетів
    mCursor,       // курсор заднього плану
    mFromColumns,  // стовпці в курсорі, які
надають дані
    mToViews,      // види в пункті подання, що
відображають дані
    0);            // прапори

// Установлює адаптер ListView.
```

```

        mRawContactList.setAdapter(mCursorAdapter);
    ...
    @Override
    public Loader<Cursor> onCreateLoader(int id, Bundle args) {

        /*
         * Встановлює стовпці для вилучення.
         * RAW_CONTACT_ID включено для ідентифікації необробленого кон-
         такту, пов'язаного з рядком даних.
         * DATA1 містить перший стовпець в рядку даних (як правило,
         найважливіший з них).
         * MIMETYPE вказує тип даних в рядку даних.
         */
        String[] projection =
            {
                ContactsContract.Contacts.Entity.RAW_CONTACT_ID,
                ContactsContract.Contacts.Entity.DATA1,
                ContactsContract.Contacts.Entity.MIMETYPE
            };

        /*
         * Проводить сортування витягнутого курсора необробленого ID кон-
         такта, щоб зберегти всі рядки даних для одного
         * контакту, що зіставляються один з одним.
         */
        String sortOrder =
            ContactsContract.Contacts.Entity.RAW_CONTACT_ID +
            " ASC";

        /*
         * Повертає новий CursorLoader. Аргументи аналогічні
         * ContentResolver.query(), крім контексту аргументу, який
         постачає місце проведення
         * ContentResolver
         */
        return new CursorLoader(
            getApplicationContext(), // контекст діяльності
            mContactUri,             // Зміст об'єкта URI для одного
контакту
            projection,               // Колонки для вилучення
            null,                     // Отримати всі вихідні
контакти і їхні ряди даних
            null,                     //
            sortOrder);              // Сортування по ID необробле-
ного контакту.
    }

```

По завершенні завантаження **LoaderManager** виконує зворотний виклик методу **onLoadFinished()**. Одним із вхідних аргументів для цього методу є **Cursor** з результатом запиту. У своєму додатку можна отримати дані з цього об'єкта **Cursor** для його відображення або подальшої роботи з ним.

Пакетне перетворення. При кожній нагоді дані в постачальнику контактів слід вставляти, оновлювати і видаляти в «пакетному режимі» шляхом створення `ArrayList` з об'єктів `ContentProviderOperation` і виклику методу `applyBatch()`. Оскільки постачальник контактів виконує всі операції в методі `applyBatch()` за одну транзакцію ще раз, ваші зміни завжди знаходитимуться в рамках сховища контактів і завжди будуть узгодженими. Пакетне перетворення також спрощує вставку необробленого контакту одночасно з його детальними даними.

Примітка. Щоб змінити *окремий* необроблений контакт, рекомендується відправити намір в додаток для роботи з контактами на пристрої замість обробки зміни в вашому додатку.

Межі. Пакетне перетворення великої кількості операцій може заблокувати виконання інших процесів, що може негативно позначитися на роботі користувача з додатком. Щоб упорядкувати всі необхідні зміни в рамках якомога меншої кількості окремих списків і при цьому запобігти блокуванню роботи системи, слід задати **межі** для однієї або декількох операцій. Межа є об'єктом `ContentProviderOperation`, в якості значення параметра `isYieldAllowed()` якого задано `true`. При досягненні постачальником контактів межі він призупиняє свою роботу, щоб могли виконуватися інші процеси, і закриває поточну транзакцію. Коли постачальник запускається знову, він виконує наступну операцію в `ArrayList` і запускає нову транзакцію.

Використання меж дозволяє за один виклик методу `applyBatch()` виконувати більше однієї транзакції. З цієї причини вам слід задати межі для останньої операції з набором пов'язаних рядків. Наприклад, необхідно задати межі для останньої операції в наборі, яка додає рядки необробленого контакту і пов'язані з ним рядки даних, або для останньої операції з набором рядків, пов'язаних з одним контактом.

Межі також є одиницями атомарних операцій. Всі операції доступу в проміжку між двома межами завершуються або успіхом, або збоєм в рамках однієї одиниці. Якщо будь-яка з меж не задана, найменшою атомарною операцією вважається весь пакет операцій. Використання меж дозволяє запобігти зниженню продуктивності системи і одночасно забезпечити виконання набору операцій на атомарному рівні.

Зміна зворотніх посилань. При вставлянні нового рядка необробленого контакту і пов'язаних з ним рядків даних у вигляді набору об'єктів `ContentProviderOperation` вам доводиться пов'язувати рядки даних з ряд-

ком необробленого контакту шляхом вставляння значення `android.provider.BaseColumns#_ID` необробленого контакту у вигляді значення `RAW_CONTACT_ID`. Однак це значення недоступно, поки створюєте `ContentProviderOperation` для рядка даних, оскільки ще не застосували `ContentProviderOperation` для рядка необробленого контакту. Щоб обійти це обмеження, в класі `ContentProviderOperation.Builder` передбачений метод `withValueBackReference()`. За допомогою цього методу можна встановлювати або змінювати стовпець з результатом попередньої операції.

У методі `withValueBackReference()` передбачено два аргументи:

1) `key` – ключ для пари «ключ-значення». Значенням цього аргументу має бути ім'я стовпця в таблиці, яку змінюєте.

2) `previousResult` – індекс значення, що починається з «0», в масиві об'єктів `ContentProviderResult` з методу `applyBatch()`. По мірі виконання пакетних операцій результат кожної операції зберігається в проміжному масиві результатів. Значенням `previousResult` є індекс одного з цих результатів, який витягується і зберігається зі значенням `key`. Завдяки цьому можна вставити новий запис необробленого контакту і отримати назад його значення `android.provider.BaseColumns#_ID`, а потім створити «зворотнє посилання» на значення при додаванні рядка `ContactsContract.Data`.

Цілком весь результат створюється при першому виклику методу `applyBatch()`. Розмір результату дорівнює розміру `ArrayList` наданих вами об'єктів `ContentProviderOperation`. Однак усім елементам в масиві результатів присвоюється значення `null`, і при спробі скористатися зворотнім посиланням на результат операції, що ще не була виконана, метод `withValueBackReference()` видає `Exception`.

Нижче наведено фрагменти коду для вставляння нового необробленого контакту і його даних в пакетному режимі. Вони включають код, який задає межа і використовує зворотнє посилання. Ці фрагменти яляють собою розширену версію методу `createContacEntry()`, який входить в клас `ContactAdder` у прикладі додатка `Contact Manager`.

Перший фрагмент коду служить для отримання даних контакту з призначеного для користувача інтерфейсу. На цьому етапі користувач уже вибрав акаунт, для якого необхідно додати новий необроблений контакт:

```
// Створює контакт з поточних значень призначеного для користувача інтерфейсу, з використанням поточного обраного рахунку.  
protected void createContactEntry() {  
    /*  
    * Отримує значення з призначеного для користувача інтерфейсу
```

```

    */
    String name = mContactNameEditText.getText().toString();
    String phone = mContactPhoneEditText.getText().toString();
    String email = mContactEmailEditText.getText().toString();

    int phoneType = mContactPhoneTypes.get(
        mContactPhoneTypeSpinner.getSelectedItemPosition());

    int emailType = mContactEmailTypes.get(
        mContactEmailTypeSpinner.getSelectedItemPosition());

```

У наступному фрагменті коду створюється операція для вставляння рядка необробленого контакту в таблицю `ContactsContract.RawContacts`:

```

    /*
     * Підготовлює пакетну обробку для вставки нового необробленого
     * контакту і його дані. Навіть якщо
     * Постачальник Контактів не має ніяких даних для цієї людини, ви
     * не можете додати контакт,
     * тільки необроблений контакт. Постачальник Контактів потім до-
     * дасть контакт автоматично.
     */
    // Створює новий масив об'єктів ContentProviderOperation.
    ArrayList<ContentProviderOperation> ops =
        new ArrayList<ContentProviderOperation>();

    /*
     * Створює новий необроблений контакт з його типом облікового за-
     * пису (тип сервера) і ім'я облікового запису
     * (рахунок користувача). Пам'ятайте, що псевдонім не зберігається
     * в цьому рядку, а в
     * рядку даних StructuredName. Інші дані не потрібні.
     */
    ContentProviderOperation.Builder op =

    ContentProviderOperation.newInsert(ContactsContract.RawContacts.CONTENT_
    T_URI)
        .withValue(ContactsContract.RawContacts.ACCOUNT_TYPE,
        mSelectedAccount.getType())
        .withValue(ContactsContract.RawContacts.ACCOUNT_NAME,
        mSelectedAccount.getName());
    // Будує операцію і додає її до масиву операцій
    ops.add(op.build());

```

Потім код створює рядки даних для рядків імені, телефону і адреси ел. пошти, що відображуються.

Кожен об'єкт використовує метод `withValueBackReference()` для отримання `RAW_CONTACT_ID`. Посилання повертається назад до об'єкту `ContentProviderResult` з першої операції, в результаті чого додається рядок необробленого контакту і повертається його нове значення

`android.provider.BaseColumns#_ID`. Після цього кожен рядок даних автоматично пов'язується за своїм `RAW_CONTACT_ID` з новим рядком `ContactsContract.RawContacts`, якому він належить.

Об'єкт `ContentProviderOperation.Builder`, який додає рядок адреси ел. пошти, позначається прапором за допомогою методу `withYieldAllowed()`, який задає межі:

```
// Створює ім'я для нового необробленого контакту, як ряд даних
StructuredName.
op =

ContentProviderOperation.newInsert(ContactsContract.Data.CONTENT_URI)
/*
 * з ValueBackReference встановлює значення першого аргу-
менту до значення
 * ContentProviderResult, що індексується другим аргумен-
том. В даному конкретному
 * виклику, ID колонки необробленого контакту
StructuredName рядок даних встановлюється
 * на значення результату, що повертається першою
операцією, саме яка
 * фактично додає необроблений рядок контакту.
 */

.withValueBackReference(ContactsContract.Data.RAW_CONTACT_ID, 0)

// Установлює тип даних MIME до StructuredName
.withValue(ContactsContract.Data.MIMETYPE,

ContactsContract.CommonDataKinds.StructuredName.CONTENT_ITEM_TYPE)

// Задає коротке ім'я рядка даних для імені в інтерфейсі.
.withValue(ContactsContract.CommonDataKinds.StructuredName.DISPLAY_NAME, name);

// Будує операцію і додає її в масив операцій
ops.add(op.build());

// Вставляє вказаний телефонний номер і тип як рядок даних теле-
фону
op =

ContentProviderOperation.newInsert(ContactsContract.Data.CONTENT_URI)
/*
 * Встановлює значення вихідного стовпця ідентифікатора
контакту до нового необробленого ID контакту повернутого
 * першою операцією в пакеті.
 */
```

```

.withValueBackReference(ContactsContract.Data.RAW_CONTACT_ID, 0)

        // Задає рядок даних MIME-типу для телефону
        .withValue(ContactsContract.Data.MIMETYPE,
ContactsContract.CommonDataKinds.Phone.CONTENT_ITEM_TYPE)

        // Встановлює номер і тип телефону
        .withValue(ContactsContract.CommonDataKinds.Phone.NUMBER,
phone)
        .withValue(ContactsContract.CommonDataKinds.Phone.TYPE,
phoneType);

    // Будує операцію і додає її в масив операцій
    ops.add(op.build());

    // Вставляє вказану адресу електронної пошти та тип як рядок да-
них телефону
    op =
ContentProviderOperation.newInsert(ContactsContract.Data.CONTENT_URI)
    /*
        * Встановлює значення вихідного стовпця ідентифікатора
контакту до нового необробленого ID контакту, повернутого
        * першою операцією в пакеті.
        */

.withValueBackReference(ContactsContract.Data.RAW_CONTACT_ID, 0)

        // Задає рядок даних MIME-типу для ел. адреси
        .withValue(ContactsContract.Data.MIMETYPE,
ContactsContract.CommonDataKinds.Email.CONTENT_ITEM_TYPE)

        // Встановлює ел. адресу та тип

.withValue(ContactsContract.CommonDataKinds.Email.ADDRESS, email)
        .withValue(ContactsContract.CommonDataKinds.Email.TYPE,
emailType);

    /*
        * Демонструється вихідна точка. По закінченню цієї вставки,
партія пакетних операцій
        * дасть пріоритет для інших потоків. Використовуйте після кожно-
го набору операцій, які впливають на
        * один контакт, щоб уникнути зниження продуктивності.
        */
    op.withYieldAllowed(true);

    // Будує операцію і додає її в масив операцій
    ops.add(op.build());

```

В останньому фрагменті коду наведено виклик методу `applyBatch()` для вставляння нового необробленого контакту і його рядків даних:

```
// Попросить постачальника контактів створити новий контакт
Log.d(TAG, "Selected account: " + mSelectedAccount.getName() + " ("
+
    mSelectedAccount.getType() + ")");
Log.d(TAG, "Creating contact: " + name);

/*
 * Застосовує масив об'єктів ContentProviderOperation в пакетному
режимі. Результати
 * відкидаються.
 */
try {
    getContentResolver().applyBatch(ContactsContract.AUTHORITY, ops);
} catch (Exception e) {
    // Показує застереження
    Context ctx = getApplicationContext();
    CharSequence txt =
    getString(R.string.contactCreationFailure);
    int duration = Toast.LENGTH_SHORT;
    Toast toast = Toast.makeText(ctx, txt, duration);
    toast.show();
    // Ввод застереження
    Log.e(TAG, "Exception encountered while inserting contact:
" + e);
}
}
```

За допомогою пакетних операцій можна також реалізувати **оптимістичне управління паралелізмом**. Це метод застосування транзакцій зміни без необхідності блокувати базовий репозиторій. Щоб скористатися цим методом, необхідно застосувати транзакцію і перевірити наявність інших змін, які могли бути внесені в цей же час. Якщо виявиться, що є неузгоджена зміна, транзакцію можна відкотити і виконати повторно.

Оптимістичне управління паралелізмом підходить для мобільних пристроїв, де з системою одночасно взаємодіє тільки один користувач, а одночасний доступ до сховища даних декількох користувачів – досить рідкісне явище. Оскільки не застосовується блокування, економиться дорогоцінний час, який потрібен на установлення блокувань або очікування того, коли інші транзакції скасують свої блокування.

Порядок використання оптимістичного управління паралелізмом при оновленні одного рядка `ContactsContract.RawContacts` такий:

1. Вийміть стовпець `VERSION` необробленого контакту, а також інші дані, які необхідно витягти.

2. Створіть об'єкт `ContentProviderOperation.Builder`, відповідний для застосування обмеження, за допомогою методу `newAssertQuery(Uri)`. Для URI контенту використовуйте `RawContacts.CONTENT_URI`, додавши до нього `android.provider.BaseColumns#_ID` необробленого контакту.

3. Для об'єкта `ContentProviderOperation.Builder` викличте метод `withValue()`, щоб порівняти стовпець `VERSION` з номером версії, яку ви тільки що отримали.

4. Для того ж об'єкта `ContentProviderOperation.Builder` викличте метод `withExpectedCount()`, щоб переконатися в тому, що перевірюче твердження перевіряє тільки один рядок.

5. Викличте метод `build()`, щоб створити об'єкт `ContentProviderOperation`, потім додайте цей об'єкт як перший об'єкт в `ArrayList`, що передається в метод `applyBatch()`.

6. Застосуйте пакетну транзакцію.

Якщо в проміжку між зчитуванням рядка і спробою його зміни рядок необробленого контакту був оновлений іншою операцією, `assert ContentProviderOperation` завершиться збоєм, і у виконанні всього пакета операцій буде відмовлено. Можна вибрати повторне виконання пакета або виконати іншу дію.

У прикладі коду демонструється, як створити `assert ContentProviderOperation` після запиту одного необробленого контакту з допомогою `CursorLoader`:

```
/*
 * Додаток використовує CursorLoader для того, щоб запросити таблицю
 * необроблених контактів. Система викликає цей метод
 * коли закінчиться завантаження.
 */
public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {

    // Отримує _ID і VERSION значення необробленого контакту
    mRawContactID =
        cursor.getLong(cursor.getColumnIndex(BaseColumns._ID));
    mVersion = cur-
        sor.getInt(cursor.getColumnIndex(SyncColumns.VERSION));
}

...

// Встановлює Uri для операції Assert
Uri rawContactUri =
    ContentUris.withAppendedId(RawContacts.CONTENT_URI, mRawContactID);
```

```

// Створює конструктор для операції Assert
ContentProviderOperation.Builder assertOp =
ContentProviderOperation.netAssertQuery(rawContactUri);

// Додає затвердження до операції Assert: перевіряє версію і
кількість рядків, що тестуються
assertOp.withValue(SyncColumns.VERSION, mVersion);
assertOp.withExpectedCount(1);

// Створює ArrayList для зберігання об'єктів ContentProviderOperation
ArrayList ops = new ArrayList<ContentProviderOperation>;

ops.add(assertOp.build());

// Ви можете додати решту ваших пакетних операцій до «OPS» тут
...

// Використовує групу. Якщо ствердження провалюється, виникає
виключення
try
{
    ContentProviderResult[] results =
        getContentResolver().applyBatch(AUTHORITY, ops);
} catch (OperationApplicationException e) {

    // Дії, які ви хочете прийняти, якщо операція провалюється
}

```

Отримання і зміна даних за допомогою намірів. За допомогою відправлення намірів в додаток для роботи з контактами, які записано на пристрій, можна в обхід отримати доступ до постачальника контактів. Намір запускає інтерфейс користувача програми на пристрої, за допомогою якого користувач може працювати з контактами. Такий тип доступу дозволяє користувачеві виконувати наступні дії:

- вибір контактів зі списку та їх повернення в додаток для подальшої роботи;
- зміна даних існуючого контакту;
- вставляння нового необробленого контакту для будь-яких інших акаунтів;
- видалення контакту або його даних.

Якщо користувач вставляє або оновлює дані, можна спочатку зібрати ці дані, а потім відправити їх разом із наміром.

При використанні намірів для доступу до постачальника контактів за допомогою додатка для роботи з контактами, наявного на пристрої, вам не потрібно створювати власний інтерфейс користувача або код для доступу до постачальника. Також вам не потрібно запитувати дозвіл на читання або запис в постачальнику. Додаток для роботи з контактами, наявний на пристрої, може делегувати вам дозвіл на читання контакту, і, оскільки ви вносите зміни в постачальник через інший додаток, вам не потрібен дозвіл на запис.

У табл. 4.15 наведено зведені відомості про операції, тип MIME і значення даних, які використовуються для доступних задач. Значення додаткових даних, які можна використовувати для `putExtra()`, вказані в довідковій документації по `ContactsContract.Intents.Insert`.

Додаток для роботи з контактами, наявний на пристрої, не дозволяє вам видаляти необроблені контакти або будь-які його дані за допомогою намірів. Замість цього для обробки необробленого контакту використовуйте метод `ContentResolver.delete()` або `ContentProviderOperation.newDelete()`.

Нижче наведено фрагмент коду для створення і відправлення намірів, який вставляє новий необроблений контакт і його дані:

```
// Отримує значення з призначеного для користувача інтерфейсу
String name = mContactNameEditText.getText().toString();
String phone = mContactPhoneEditText.getText().toString();
String email = mContactEmailEditText.getText().toString();

String company = mCompanyName.getText().toString();
String jobtitle = mJobTitle.getText().toString();

// Створює новий намір для відправки в додаток контактів пристрою
Intent insertIntent = new Intent(ContactsContract.Intents.Insert.ACTION);

// Встановлює тип MIME
insertIntent.setType(ContactsContract.RawContacts.CONTENT_TYPE);

// Встановлює нове ім'я контакту
insertIntent.putExtra(ContactsContract.Intents.Insert.NAME, name);

// Встановлює нову компанію і посаду
insertIntent.putExtra(ContactsContract.Intents.Insert.COMPANY, company);
insertIntent.putExtra(ContactsContract.Intents.Insert.JOB_TITLE,
jobtitle);

/*
 * Демонструє додавання рядків даних в якості списку масиву, пов'язаного
 з ключем DATA
 */
```



```

// Визначає список масиву, щоб утримувати об'єкти ContentValues для кож-
ного рядка
ArrayList<ContentValues> contactData = new ArrayList<ContentValues>();

/*
 * Визначає необроблений рядок контакту
 */

// Встановлює рядок в якості об'єкта ContentValues
ContentValues rawContactRow = new ContentValues();

// Додає тип облікового запису та ім'я в рядок
rawContactRow.put(ContactsContract.RawContacts.ACCOUNT_TYPE,
mSelectedAccount.getType());
rawContactRow.put(ContactsContract.RawContacts.ACCOUNT_NAME,
mSelectedAccount.getName());

// Додає рядок до масиву
contactData.add(rawContactRow);

/*
 * Встановлює рядок даних телефонного номера
 */

// Встановлює рядок в якості об'єкта ContentValues
ContentValues phoneRow = new ContentValues();

// Вказує тип MIME для цього рядка даних (всі рядки даних повинні бути
відзначені за їх типом)
phoneRow.put(
    ContactsContract.Data.MIMETYPE,
    ContactsContract.CommonDataKinds.Phone.CONTENT_ITEM_TYPE
);

// Додає номер телефону і його тип до ряду
phoneRow.put(ContactsContract.CommonDataKinds.Phone.NUMBER, phone);

// Додає рядок в масив
contactData.add(phoneRow);

/*
 * Встановлює рядок даних електронної пошти
 */

// Встановлює рядок в якості об'єкта ContentValues
ContentValues emailRow = new ContentValues();

// Вказує тип MIME для цього рядка даних (всі рядки даних повинні бути
відзначені за їх типом)
emailRow.put(
    ContactsContract.Data.MIMETYPE,
    ContactsContract.CommonDataKinds.Email.CONTENT_ITEM_TYPE
);

```

```

);
// Додає адресу електронної пошти та тип рядка
emailRow.put(ContactsContract.CommonDataKinds.Email.ADDRESS, email);

// Додає рядок в масив
contactData.add(emailRow);

insertIntent.putParcelableArrayListExtra(ContactsContract.Intents.Insert.
DATA, contactData);

// Розсилає намір для старту додатка на пристрої та активності контакту.
startActivity(insertIntent);

```

Цілісність даних. Оскільки в репозиторії контактів міститься важлива і конфіденційна інформація, яка, як очікує користувач, вірна і актуальна, в постачальнику контактів передбачені чітко визначені правила для забезпечення цілісності даних. При зміні даних контактів ви зобов'язані дотримуватися цих правил. Ось найважливіші з цих правил:

1. Завжди додавайте рядок `ContactsContract.CommonDataKinds.StructuredName` до кожного рядка, що додаєте в `ContactsContract.RawContacts`.

2. Якщо в таблиці `ContactsContract.Data` є рядок `ContactsContract.RawContacts` без рядка `ContactsContract.CommonDataKinds.StructuredName`, то можуть виникнути проблеми під час агрегування.

3. Завжди зв'язуйте нові рядки `ContactsContract.Data` з їх батьківськими рядками `ContactsContract.RawContacts`.

4. Рядок `ContactsContract.Data`, який не пов'язаний з `ContactsContract.RawContacts`, не буде показаний у додатку для роботи з контактами, що є на пристрої, а також може призвести до проблем з адаптерами синхронізації.

Слід змінювати дані тільки тих необроблених контактів, якими ви володієте.

Пам'ятайте про те, що постачальник контактів зазвичай управляє даними з декількох акаунтів різних типів або послуг через Інтернет. Необхідно перекоонатися в тому, що ваш додаток змінює або видаляє дані тільки для тих рядків, які належать вам, а також в тому, що він вставляє дані з використанням тільки тих типів і імені акаунта, якими ви керуєте.

Завжди використовуйте для центрів, URI контенту, шляхів URI, імен стовпців, типів MIME та значень TYPE тільки ті константи, які визначені в класі `ContactsContract` і його підкласах.

Це дозволить уникнути помилок. Якщо яка-небудь константа застаріла, компілятор повідомить про це за допомогою відповідного попередження.

Таблиця 4.15 – Наміри в постачальнику контактів

Задача	Дія	Дані	Тип MIME	Примітки
1	2	3	4	5
Вибір контакту зі списку	ACTION_PICK	Одне з нижченаведеного: Contacts.CONTENT_URI (відображення списку контактів); Phone.CONTENT_URI (відображення номерів телефонів для необробленого контакту); StructuredPostal.CONTENT_URI (відображення списку поштових адрес для необробленого контакту); Email.CONTENT_URI (відображення адрес ел. пошти для необробленого контакту)	Не використовується	Відображення списку необроблених контактів або списку даних необробленого контакту (залежно від наданого URI контенту). Викличте метод <code>startActivityForResult()</code> , який повертає URI контенту для вибраного рядка. URI поданий у формі URI контенту таблиці, до якого доданий LOOKUP_ID рядка. Додаток для роботи з контактами, встановлений на пристрої, делегує цьому URI контенту дозвіл на читання і запис, які діють протягом усього життєвого циклу вашої операції
Вставка нового необробленого контакту	Insert.ACTION	Н/Д	RawContacts.CONTENT_TYPE , тип MIME для набору необроблених контактів	Відображення екрану Додати контакт в додатку для роботи з контактами, який встановлено на пристрої. Відображаються значення додаткових даних, доданих вами в намір. При відправленні за допомогою методу <code>startActivityForResult()</code> URI контенту нового доданого необробленого контакту передається назад в метод зворотного виклику <code>onActivityResult()</code> вашої операції в аргументі <code>Intent</code> у полі <code>data</code> . Щоб отримати значення, викличте метод <code>getData()</code>

Продовження табл. 4.15

1	2	3	4	5
Зміна контакту	ACTION_EDIT	CONTENT_LOOKUP_URI контакту. Операція редактора дозволить користувачеві змінити будь-які дані, пов'язані з цим контактом	Contacts.CONTENT_ITEM_TYPE Один контакт.	Відображення екрана редагування контакту в додатку для роботи з контактами. Відображаються значення додаткових даних, поданих вами в намір. Коли користувач натискає на кнопку Готово для збереження внесених змін, ваша операція повертається на попередній план
Відображення засобу вибору, в якому також можна додавати дані	ACTION_INSERT_OR_EDIT	Н/Д	CONTENT_ITEM_TYPE	Цей намір також відображає екран засобу вибору програми для роботи з контактами. Користувач може обрати контакт для зміни або додати новий контакт. Залежно від вибору відображається екран редагування або додавання контакту, і відображаються додаткові дані, передані вами в намір. Якщо ваш додаток відображає такі дані контакту, як адреса ел. пошти або номер телефону, скористайтесь цим наміром, щоб дозволити користувачеві додавати дані до існуючого контакту. Примітка. Вам не потрібно відправляти значення імені в додаткові дані свого наміру, оскільки користувач завжди обирає існуюче ім'я або додає нове. Крім того, якщо ви вносите зміни, додаток для роботи з контактами відобразить відправлене вами ім'я, перезапитуючи попереднє значення. Якщо користувач не помітить цього і збереже зміни, старе значення буде втрачено

Рядки даних, які можна налаштувати. Створивши і використовуючи власні типи MIME, можна вставляти, змінювати, видаляти і витягувати в таблиці `ContactsContract.Data` власні рядки даних. Ваші рядки обмежені використанням стовпчика, який визначений в `ContactsContract.DataColumns`, однак можна порівняти ваші власні імена стовпців за типами рядків з іменами стовпців за замовчуванням. Дані для ваших рядків відображаються в додатку для роботи з контактами, який записано на пристрій, проте їх не вдасться змінити або видалити, а користувачі не зможуть додати додаткові дані. Щоб дозволити користувачам змінювати ваші рядки даних, які можна налаштувати, необхідно реалізувати у вашому додатку операцію редактора.

Для відображення даних, що налаштовуються, вкажіть файл `contacts.xml`, що містить елемент `<ContactsAccountType>` і один або декілька його `<ContactsDataKind>` дочірніх елементів.

4.5.9. Адаптери синхронізації постачальника контактів

Постачальник контактів розроблений спеціально для обробки операцій синхронізації даних контактів між пристроєм і службою в Інтернеті. Завдяки цьому користувачі можуть завантажувати існуючі дані на новий пристрій і відправляти їх в новий обліковий запис. Синхронізація також забезпечує надання користувачеві завжди актуальних відомостей, незалежно від джерела їх додавання і внесених до них змін. Ще одна перевага синхронізації полягає в тому, що дані контактів доступні навіть в тому випадку, якщо пристрій не підключений до мережі.

Незважаючи на безліч різних способів реалізації синхронізації даних, в системі Android підключається платформа синхронізації, яка дозволяє автоматизувати виконання таких завдань:

- перевірка доступності мережі;
- планування і виконання синхронізації на основі уподобань користувача;
- перезапуск зупинених процесів синхронізації.

Для використання платформи надається модуль адаптера синхронізації. Кожен адаптер синхронізації є унікальним для служби і постачальника контенту, проте він здатний працювати з декількома обліковими записами в одній службі. У платформі також передбачена можливість використовувати кілька адаптерів синхронізації для тієї ж самої служби або постачальника.

Класи і файли адаптера синхронізації. Адаптер синхронізації реалізується як підклас класу `AbstractThreadedSyncAdapter` і встановлюється в складі

додатка Android. Система дізнається про наявність адаптера синхронізації з елементів в маніфесті вашого додатка, а також з особливого файлу XML, на який є вказівка в маніфесті. У файлі XML визначаються тип акаунта в онлайн-службі і центр постачальника контенту, які разом служать унікальними ідентифікаторами адаптера. Адаптер синхронізації знаходиться в неактивному стані доти, поки користувач не додасть тип акаунта і не включить синхронізацію з постачальником контенту. На цьому етапі система вступає в управління адаптером, за необхідності викликаючи його для синхронізації даних між постачальником контенту і сервером.

Примітка. Використання типу акаунта для ідентифікації адаптера синхронізації дозволяє системі виявляти і групувати адаптери синхронізації, які звертаються до різних служб з тієї ж самої організації. Наприклад, у всіх адаптерів синхронізації для онлайн-служб Google той самий тип акаунта – `com.google`. Коли користувачі додають на свої пристрої облікового запису Google, всі встановлені адаптери синхронізації групуються разом; кожен з адаптерів синхронізується тільки з окремим постачальником контенту на пристрої.

Оскільки в більшості служб користувачам спочатку необхідно підтвердити свою автентичність, перш ніж вони зможуть отримати доступ до даних, система Android пропонує платформу аутентифікації, яка аналогічна платформі адаптера синхронізації і часто використовується разом з нею. Платформа аутентифікації використовує спільні структури перевірки автентичності, які являють собою підкласи класу `AbstractAccountAuthenticator`. Така структура перевіряє справжність користувача таким чином:

1. Спочатку збираються відомості про ім'я користувача і його пароль або аналогічна інформація (облікові дані користувача).
2. Потім облікові дані оговтуються в службу.
3. Нарешті, вивчається відповідь, отримана від служби.

Якщо служба прийняла облікові дані, структура перевірки автентичності може зберегти їх для використання в подальшому. Завдяки використанню структур перевірки автентичності `AccountManager` може надати доступ до будь-яких маркерів аутентифікації, які підтримує структура перевірки автентичності і які вона вирішує надати, наприклад, до маркерів аутентифікації `OAuth2`.

Незважаючи на те що аутентифікація не вимагається, вона використовується більшістю служб для роботи з контактами. Проте вам не обов'язково використовувати платформу аутентифікації Android для перевірки автентичності.

Реалізація адаптера синхронізації. Щоб реалізувати адаптер синхронізації для постачальника контактів, спочатку треба створити додаток Android, який містить такі компоненти:

- Компонент **Service**, Який відповідає на запити системи для прив'язки до адаптера синхронізації. Коли системі потрібно запустити синхронізацію, вона викликає метод `onBind()` служби, щоб отримати об'єкт `IBinder` для адаптера синхронізації. Завдяки цьому система може викликати методи адаптера між процесами.

У прикладі адаптера синхронізації ім'ям класу цієї служби є `com.example.android.samplesync.syncadapter.SyncService`.

- Безпосередньо адаптер синхронізації, реалізований як конкретний підклас класу `AbstractThreadedSyncAdapter`. Цей клас не підходить для завантаження даних з сервера, відправки даних з пристрою і вирішення конфліктів. Основну свою роботу адаптер виконує в методі `onPerformSync()`. Цей клас допускає створення тільки одного примірника.

У прикладі адаптера синхронізації адаптер визначається в класі `com.example.android.samplesync.syncadapter.SyncAdapter`.

Підклас класу `Application` виступає в ролі фабрики для єдиного екземпляра адаптера синхронізації. Скористайтесь методом `onCreate()`, щоб створити екземпляр адаптера синхронізації, а потім надайте статичний метод `get`, щоб повернути єдиний екземпляр в метод `onBind()` служби адаптера синхронізації.

- **Не обов'язково:** компонент **Service**, який відповідає на запити системи для аутентифікації користувачів. `AccountManager` запускає службу, щоб почати процес аутентифікації. Метод `onCreate()` служби створює екземпляр об'єкта структури перевірки автентичності. Коли системі потрібно запустити аутентифікацію акаунта користувача для адаптера синхронізації додатка, вона викликає метод `onBind()` служби, щоб отримати об'єкт `IBinder` для структури перевірки автентичності. Завдяки цьому система може викликати методи структури перевірки автентичності між процесами.

У прикладі адаптера синхронізації ім'ям класу цієї служби є `com.example.android.samplesync.authenticator.AuthenticationService`.

- **Не обов'язково:** конкретний підклас класу `AbstractAccountAuthenticator`, який обробляє запити на аутентифікацію. У цьому класі є методи, які `AccountManager` викликає для перевірки автентичності облікових даних користувача на сервері. Подробиці процесу аутентифікації значно різняться залежно від технології, що використовується на сервері. Щоб

дізнатися докладніше про аутентифікацію, зверніться до відповідної документації програмного забезпечення використовуваного сервера.

У прикладі адаптера синхронізації структура перевірки автентичності визначається в класі – `com.example.android.samplesync.authenticator.Authenticator`.

– Файли XML, в яких визначаються адаптер синхронізації і структура перевірки автентичності для системи.

Описані раніше компоненти служби адаптера синхронізації і структури перевірки автентичності визначаються в елементах `<service>` в маніфесті додатка. Ці елементи включають дочірні елементи `<meta-data>`, в яких є певні дані для системи.

Елемент `<meta-data>` для служби адаптера синхронізації вказує на файл XML `res/xml/syncadapter.xml`. У свою чергу, в цьому файлі задається URI веб-служби для синхронізації з постачальником контактів, а також тип акаунта для цієї веб-служби.

Не обов'язково: елемент `<meta-data>` для структури перевірки автентичності вказує на файл XML `res/xml/authenticator.xml`. У свою чергу, в цьому файлі задається тип акаунта, який підтримує структура перевірки автентичності, а також ресурси для користувача інтерфейсу, які відображаються в процесі аутентифікації. Тип акаунта, зазначений в цьому елементі, повинен збігатися з типом акаунта, який заданий для адаптера синхронізації.

4.5.10. Потoki даних із соціальних мереж

Для управління вхідними даними з соціальних мереж використовуються таблиці `android.provider.ContactsContract.StreamItems` і `android.provider.ContactsContract.StreamItemPhotos`. Можна створити адаптер синхронізації, який додає потік даних з вашої власної мережі в ці таблиці, або можна зчитувати потік даних з цих таблиць і відображати їх у власному додатку. Можна також реалізувати обидва ці способи. За допомогою цих функцій можна інтегрувати служби соціальних мереж в компоненти Android для роботи з соціальними мережами.

Текст з потоку даних із соціальних мереж. Елементи потоку завжди асоціюються з необробленим контактом. Ідентифікатор `android.provider.ContactsContract.StreamItems.Columns#RAW_CONTACT_ID` зв'язується зі значенням `_ID` необробленого контакту. Тип акаунта і його ім'я для необробленого контакту також зберігаються в рядку елемента потоку.

Дані з потоку слід зберігати в таких стовпцях:

– **android.provider.ContactsContract.StreamItemsColumns#ACCOUNT_TYPE**

Обов'язковий. Тип аккаунта користувача для необробленого контакту, пов'язаного з цим елементом потоку. Не забудьте поставити це значення при вставлянні елемента потоку.

– **android.provider.ContactsContract.StreamItemsColumns#ACCOUNT_NAME**

Обов'язковий. Ім'я облікового запису користувача для необробленого контакту, пов'язаного з цим елементом потоку. Не забудьте поставити це значення при вставлянні елемента потоку.

Стовпці ідентифікатора.

Обов'язковий. При вставлянні елемента потоку необхідно вставити такі стовпці ідентифікатора:

• **android.provider.ContactsContract.StreamItemsColumns#CONTACT_ID:**

значення `android.provider.BaseColumns#_ID` для контакту, з яким асоційований цей елемент потоку;

• **android.provider.ContactsContract.StreamItemsColumns#CONTACT_LOOKUP_KEY:**

значення `android.provider.ContactsContract.ContactsColumns # LOOKUP_KEY` для контакту, з яким асоційований цей елемент потоку;

• **android.provider.ContactsContract.StreamItemsColumns#RAW_CONTACT_ID:**

значення `android.provider.BaseColumns # _ID` для необробленого контакту, з яким асоційований цей елемент потоку.

– **android.provider.ContactsContract.StreamItemsColumns#COMMENTS**

Необов'язковий. У ньому зберігається зведена інформація, яку можна відобразити на початку елемента потоку.

– **android.provider.ContactsContract.StreamItemsColumns # TEXT**

Текст елемента потоку: або контент, опублікований джерелом елемента, або опис деякої дії, згенерованої елементом потоку. У цьому стовпці може міститися будь-яке форматування і вбудовані зображення ресурсів, рендеринг яких можна виконати за допомогою методу `fromHtml()`. Постачальник може обрізати занадто довгий контент або замінити його частину трьома крапками, однак він спробує уникнути порушення тегів.

– **android.provider.ContactsContract.StreamItemsColumns#TIMESTAMP**

Текстовий рядок з інформацією про час вставки або поновлення елемента в мілісекундах від початку відліку часу. Обслуговуванням цієї шпальти займаються додатки, які вставляють або оновлюють елементи потоку; постачальник контактів не виконує це автоматично.

Для відображення ідентифікаційної інформації для елементів потоку скористайтеся

```
android.provider.ContactsContract.StreamItemsColumns#RES_ICON,  
android.provider.ContactsContract.StreamItemsColumns#RES_LABEL,  
android.provider.ContactsContract.StreamItemsColumns#RES_PACKAGE
```

для зв'язування з ресурсами в вашому додатку.

У таблиці `android.provider.ContactsContract.StreamItems` також є стовпці `android.provider.ContactsContract.StreamItemsColumns#SYNC1`, `android.provider.ContactsContract.StreamItemsColumns#SYNC4`, які призначені виключно для адаптерів синхронізації.

Фотографії з потоку даних із соціальних мереж. Фотографії, пов'язані з елементом потоку, зберігаються в таблиці `android.provider.ContactsContract.StreamItemPhotos`. Стовпець `android.provider.ContactsContract.StreamItemPhotosColumns#STREAM_ITEM_ID` в цій таблиці пов'язаний зі стовпцем `android.provider.BaseColumns #_ID` в таблиці `android.provider.ContactsContract.StreamItems`. Посилання на фотографії зберігаються в таких стовпчиках таблиці:

– `android.provider.ContactsContract.StreamItemPhotos # PHOTO` (об'єкт BLOB). Подання фотографії в довічному форматі і зі зміненням постачальником розміром для її зберігання і відображення. Цей стовпець доступний для забезпечення зворотної сумісності з попередніми версіями постачальника контактів, які використовувалися для зберігання фотографій. Однак в поточній версії постачальника ми не рекомендуємо використовувати цей стовпець для зберігання фотографій. Замість цього скористайтеся стовпцем `android.provider.ContactsContract.StreamItemPhotosColumns#PHOTO_FILE_ID` або `android.provider.ContactsContract.StreamItemPhotosColumns#PHOTO_URI` (або обома стовпцями, як описано далі) для зберігання фотографій в файлі. У цьому стовпці тепер зберігаються мініатюри фотографій, доступних для читання:

– `android.provider.ContactsContract.StreamItemPhotosColumns # PHOTO_FILE_ID`. Числовий ідентифікатор фотографії для необробленого контакту. Додайте це значення до константи `DisplayPhoto.CONTENT_URI`, щоб отримати URI контенту для одного файлу фотографії, а потім викличте метод `openAssetFileDescriptor()`, щоб отримати засіб обробки файлу фотографії. `android.provider.ContactsContract.StreamItemPhotosColumns # PHOTO_URI`

URI контент, який вказує на файл фотографії, викличе метод `openAssetFileDescriptor()`, передавши в нього цей URI, щоб отримати оброблений файл фотографії.

– *Використання таблиць з потоку даних із соціальних мереж.* Ці таблиці працюють аналогічно іншим основним таблицями в постачальнику контактів, за винятком зазначених нижче моментів:

– робота з цими таблицями потребує більше дозволів на доступ. Для читання даних з них вашому додатку має бути наданий дозвіл `android.Manifest.permission # READ_SOCIAL_STREAM`. Для зміни таблиць ваш додаток повинен мати дозвіл `android.Manifest.permission#WRITE_SOCIAL_STREAM`;

– для `android.provider.ContactsContract.StreamItems` таблиці існує обмеження на кількість рядків, яке можна зберігати для кожного необробленого контакту. Після досягнення цього обмеження постачальник контактів звільняє місце для нових рядків елементів потоку шляхом автоматичного видалення рядків з найстарішою міткою `android.provider.ContactsContract.StreamItemsColumns#TIMESTAMP`. Щоб отримати це обмеження, запросіть URI контенту `android.provider.ContactsContract.StreamItems#CONTENT_LIMIT_URI`. Для всіх аргументів, відмінних від URI контенту, можна залишити значення `null`. Запит повертає об'єкт `Cursor`, в якому міститься один рядок з одним стовпцем `android.provider.ContactsContract.StreamItems #MAX_ITEMS`.

Клас `android.provider.ContactsContract.StreamItems.StreamItemPhotos` визначає дочірню таблицю об'єктів `android.provider.ContactsContract.StreamItemPhotos`, в якій містяться рядки для одного елемента потоку.

Взаємодія з потоками даних із соціальних мереж. Управління потоком даних із соціальних мереж здійснюється постачальником контактів спільно з додатком для управління контактами, наявними на пристрої. Такий підхід дозволяє організувати ефективне використання даних із соціальних мереж з даними про існуючі контакти і має такі функції:

1. Організувавши синхронізацію даних з соціальної служби з постачальником контактів за допомогою адаптера синхронізації, можна отримувати дані про недавню активність контактів користувача і зберігати такі дані в таблицях `android.provider.ContactsContract.StreamItems` і `android.provider.ContactsContract.StreamItemPhotos` для використання в подальшому.

2. Крім регулярної синхронізації, адаптер синхронізації можна налаштувати на отримання додаткових даних при виборі користувачем контакту для перегляду. Завдяки цьому ваш адаптер синхронізації може отримувати фотографії високої роздільної здатності та найактуальніші елементи потоку для контакту.

3. Реєструючи повідомлення в додатку для роботи з контактами і в постачальнику контактів, можна отримувати наміри при перегляді контакту і оновлювати на цьому етапі дані про стан контакту з вашої служби. Такий підхід може забезпечити більшу швидкість і менший обсяг використання смуги пропускання, ніж виконання повної синхронізації за допомогою адаптера синхронізації.

4. Користувачі можуть додати контакт у вашу службу соціальної мережі, звернувшись до контакту в додатку, який записано на пристрій. Це реалізується за допомогою функції «запросити контакт», для включення якої використовується поєднання операції, яка додає існуючий контакт в вашу мережу, і файлу XML, в якому наведено відомості про ваш додаток для постачальника контактів і додатки для роботи з контактами.

Регулярна синхронізація елементів потоку за допомогою постачальника контактів виконується так само, як і будь-яка інша синхронізація. Реєстрація повідомлень та запрошення контактів розглядаються в наступних двох розділах.

Реєстрація для обробки переглядів контактів в соціальних мережах. Щоб зареєструвати адаптер синхронізації для отримання повідомлень про перегляди користувачами контакту, управління яким здійснюється вашим адаптером синхронізації, виконайте такі дії:

1. У каталозі `res/xml/` свого проекту створіть файл `contacts.xml`. Якщо у вас вже є цей файл, переходите до наступного кроку.

2. У цьому файлі додайте об'єкт `<ContactsAccountType xmlns: android = "http://schemas.android.com/apk/res/android">`. Якщо цей елемент уже існує, можете переходити до наступного кроку.

3. Щоб зареєструвати службу, якій надсилається повідомлення при відкритті користувачем сторінки з відомостями про контакт в додатку для роботи з контактами, який записано на пристрій, додайте атрибут `viewContactNotifyService = "serviceclass"` до елемента, де `serviceclass` – це повне ім'я класу служби, яка повинна отримати намір з програми для роботи з контактами. Для служби-повідомника використовуйте клас, який є розширенням класу `IntentService`, щоб дозволити службі отримувати наміри. Дані у вхідному намірі містять URI контенту необробленого контакту, обраного користувачем. У службі-повідомнику можна прив'язати адаптер синхронізації, а потім викликати його для поновлення даних для необробленого контакту.

Щоб зареєструвати операцію, яку слід викликати при виборі користувачем елемента потоку або фотографії (або обох елементів), виконайте такі дії:

1. У каталозі `res/xml/` свого проекту створіть файл `contacts.xml`. Якщо у вас вже є цей файл, перейдіть до наступного кроку.

2. У цьому файлі додайте об'єкт `<ContactsAccountType xmlns: android = "http://schemas.android.com/apk/res/android">`. Якщо цей елемент уже існує, можна переходити до наступного кроку.

3. Щоб зареєструвати одну з ваших операцій для обробки вибору користувачем елемента потоку в додатку, який записано на пристрій, додайте атрибут `viewStreamItemActivity = "activityclass"` до елемента, де `activityclass` – це повне ім'я класу операції, яка повинна отримати намір з програми для роботи з контактами.

4. Щоб зареєструвати одну з ваших операцій для обробки вибору користувачем фотографії в потоці в додатку, який записано на пристрій, додайте атрибут `viewStreamItemPhotoActivity = "activityclass"` до елемента, де `activityclass` – це повне ім'я класу операції, яка повинна отримати намір з програми для роботи з контактами.

Дані у вхідному намірі містять URI контенту елемента або фотографії, обраних користувачем. Щоб використовувати різні операції для текстових елементів і фотографій, використовуйте обидва атрибути в одному файлі.

Взаємодія зі службою соціальної мережі. Користувачам не обов'язково виходити з додатку, який записаний на пристрій, щоб запросити контакт на сайт соціальної мережі. Замість цього додаток для роботи з контактами може відправити намір для запрошення контакту в одну з ваших операцій. Для цього виконайте такі дії:

1. У каталозі `res/xml/` свого проекту створіть файл `contacts.xml`. Якщо у вас вже є цей файл, перейдіть до наступного кроку.

2. У цьому файлі додайте об'єкт `<ContactsAccountType xmlns: android = "http://schemas.android.com/apk/res/android">`. Якщо цей елемент уже існує, можна переходити до наступного кроку.

3. Додайте такі атрибути:

–`inviteContactActivity = "activityclass";`

–`inviteContactActionLabel = "@ string/invite_action_label".`

Значення `activityclass` являє собою повне ім'я класу операції, яка повинна отримати намір. Значення `invite_action_label` – це текстовий рядок, який відображається в меню «Додати підключення» в додатку для роботи з контактами.

Примітка. `ContactsSource` – це застаріле ім'я тега для `ContactsAccountType`, яке більше не використовується.

Посилання contacts.xml. У файлі `contacts.xml` містяться елементи XML, які керують взаємодією вашого адаптера синхронізації і вашої програми з постачальником контактів і додатком для роботи з контактами. Ці елементи описані в наступних розділах.

Елемент `<ContactsAccountType>` управляє взаємодією вашої програми з додатком для роботи з контактами. Нижче наведено його синтаксис.

```
<ContactsAccountType
    xmlns:android = "http://schemas.android.com/apk/res/android"
    inviteContactActivity = "activity_name"
    inviteContactActionLabel = "invite_command_text"
    viewContactNotifyService = "view_notify_service"
    viewGroupActivity = "group_view_activity"
    viewGroupActionLabel = "group_action_text"
    viewStreamItemActivity = "viewstream_activity_name"
    viewStreamItemPhotoActivity = "viewphotostream_activity_name">
```

Елемент знаходиться в `res/xml/contacts.xml`, і може містити `<ContactsDataKind>`

Опис: цей елемент оголошує компоненти і елементи призначеного для користувача інтерфейсу, за допомогою яких користувачі можуть запрошувати свої контакти в соціальну мережу, повідомляти користувачів при оновленні одного з їх потоків даних з соціальних мереж та ін.

Зверніть увагу, що префікс атрибута `android:` необов'язково використовувати для атрибутів `<ContactsAccountType>`.

Атрибути:

`inviteContactActivity`

Повне ім'я класу операції в вашому додатку, який необхідно активувати при виборі користувачем елемента «Додати підключення» в додатку для роботи з контактами, який записано на пристрій.

`inviteContactActionLabel`

Текстовий рядок, який відображається для операції, заданої в `inviteContactActivity`, в меню «Додати підключення», наприклад, можна вказати фразу «Слідкуйте за новинами в моїй мережі». Для цього елемента можна використовувати ідентифікатор рядкового ресурсу.

`viewContactNotifyService`

Повне ім'я класу служби у вашому додатку, яка повинна отримувати повідомлення при перегляді контакту користувачем. Таке повідомлення надсилається додатком, який записано на пристрій; завдяки цьому ваш додаток може відкласти виконання операцій, що вимагають обробки великого обсягу да-

них, доти, поки це не буде потрібно. Наприклад, ваш додаток може реагувати на таке повідомлення шляхом зчитування і відображення фотографії контакту з високою роздільною здатністю і найактуальніших елементів потоку даних з соціальної мережі. Приклад служби повідомлень наведено у файлі `NotifierService.java`.

`viewGroupActivity`

Повне ім'я класу операцій, яка може відобразити інформацію про групу, при натисканні користувачем на мітку групи.

`viewGroupActionLabel`

Мітка, яка відображається додатком для елемента користувацького інтерфейсу, за допомогою якої користувач може переглянути групи в вашому додатку.

Наприклад, якщо встановити додаток Google+ на ваш пристрій і виконати синхронізацію даних в Google+ з додатком для роботи з контактами, то кола Google+ будуть позначені в додатку для роботи з контактами як групи на вкладці «Групи». При натисканні на коло Google+ учасники кола відобразяться як група контактів. У верхній частині екрану знаходиться значок Google+; якщо натиснути на нього, управління перейде в додаток Google+. У додатку для управління контактами це реалізовано за допомогою `viewGroupActivity`, в якій значок Google+ використовується як значення `viewGroupActionLabel`.

Для цього атрибута можна використовувати ідентифікатор рядкового ресурсу:

`viewStreamItemActivity`

Повне ім'я класу операції в вашому додатку, який запускає додаток для роботи з контактами, коли користувач вибирає елемент потоку для необробленого контакту.

`viewStreamItemPhotoActivity`

Повне ім'я класу операції в вашому додатку, яку запускає додаток для роботи з контактами, коли користувач вибирає фотографію в елементі потоку для необробленого контакту.

Елемент <ContactsDataKind>. Елемент `<ContactsDataKind>` управляє відображенням рядків даних вашого додатка, які налаштовуються, в інтерфейсі додатка для роботи з контактами, який записано на пристрій. Нижче наведено його синтаксис.

```
<ContactsDataKind
    android: mimeType = "MIMETYPE"
    android: icon = "icon_resources"
    android: summaryColumn="Column_name"
    android: detailColumn = "column_name">
```

Елемент знаходиться в `<ContactsAccountType>`.

Опис: використовуйте цей елемент для відображення вмісту рядків даних, що налаштовуються, в додатку для роботи з контактами як частини відомостей про необроблений контакт. Кожен дочірній елемент `<ContactsDataKind>` елемента `<ContactsAccountType>` являє собою тип рядка даних, що налаштовується, який адаптер синхронізації додає в таблицю `ContactsContract.Data`. Для кожного використовуваного вами типу MIME, що налаштовується, необхідно додати один елемент `<ContactsDataKind>`. Вам не потрібно додавати елемент, якщо у вас є рядок даних, що налаштовується, для якого не потрібно відображати дані.

Атрибути:

`android: mimeType`

Певні типи MIME, які ви налаштовуєте, для одного з ваших типів рядків даних в таблиці `ContactsContract.Data`. Наприклад, значення `vnd.android.cursor.item/vnd.example.locationstatus` може бути налаштованим типом MIME для рядка даних, в якій знаходяться записи про останнє відоме місцезнаходження контакту.

`android: icon`

Графічний ресурс Android, який додаток для роботи з контактами відображає поруч з вашими даними. Використовуйте його для позначення того, що ці дані отримані з вашої служби.

`android: summaryColumn`

Ім'я стовпця для першого з двох значень, отриманих з рядка даних. Значення відображається у вигляді першого рядка запису в цьому рядку даних. Перший рядок призначений для використання як зведені дані, однак він не обов'язковий. Див. також `android:detailColumn`.

`android: detailColumn`

Ім'я стовпця для другого з двох значень, отриманих з рядка даних. Значення відображається у вигляді другого рядка запису в цьому рядку даних. Див. також `android: summaryColumn`.

4.5.11. Додаткові можливості постачальника контактів

Крім основних функцій, описаних вище, в постачальника контактів передбачені такі корисні функції для роботи з даними контактів:

- групи контактів;
- функції роботи з фотографіями.

Групи контактів. Постачальник контактів може додатково зазначити колекції пов'язаних контактів з даними про групу. Якщо серверу, який пов'язаний з обліковим записом користувача, потрібно зберегти групи, адаптеру синхронізації для типу цього акаунта слід передати дані про групи з постачальника контактів на сервер. При додаванні користувачем нового контакту на сервер і подальшому приміщенні цього контакту в нову групу адаптер синхронізації повинен додати цю нову групу в таблицю `ContactsContract.Groups`. Група або групи, в які входить необроблений контакт, зберігаються в таблиці `ContactsContract.Data` з використанням типу `MIME ContactsContract.CommonDataKinds.GroupMembership`.

Якщо необхідно створити адаптер синхронізації, який буде додавати дані необробленого контакту з сервера в постачальник контактів, а не використовувати групи, то вам необхідно вказати для постачальника, щоб він зробив ваші дані видимими. У коді, який виконується при додаванні користувачем акаунта на пристрій, поновіть рядок `ContactsContract.Settings`, яку постачальник контактів додає для цього облікового запису. У цьому рядку вкажіть в стовпці `Settings.UNGROUPED_VISIBLE` значення «1». Після цього постачальник контактів завжди буде робити ваші дані видимими, навіть якщо ви не використовуєте групи.

Фотографії контактів. У таблиці `ContactsContract.Data` зберігаються фотографії у вигляді рядків `Photo.CONTENT_ITEM_TYPE` типу `MIME`. Стовпець `CONTACT_ID` в рядку пов'язаний зі стовпцем `android.provider.BaseColumns#_ID` необробленого контакту, якому він належить. Клас `ContactsContract.Contacts.Photo` визначає вкладену таблицю `ContactsContract.Contacts`, у якій міститься інформація про основну фотографію контакту (яка є основною фотографією основного необробленого контакту цього контакту). Аналогічним чином клас `ContactsContract.RawContacts.DisplayPhoto` визначає вкладену таблицю `ContactsContract.RawContacts`, у якій міститься інформація про основну фотографію необробленого контакту.

У довідковій документації по `ContactsContract.Contacts.Photo` і `ContactsContract.RawContacts.DisplayPhoto` містяться приклади отримання інформації про фотографії. На жаль, відсутній клас для зручного витягання мініатюри основної фотографії необробленого контакту, проте можна відправити запит в таблицю `ContactsContract.Data`, вибрати `android.provider.BaseColumns#_ID` необробленого контакту,

Photo.CONTENT_ITEM_TYPE і стовпець IS_PRIMARY, щоб знайти рядок основної фотографії необробленого контакту.

Потоки даних із соціальних мереж також можуть включати фотографії. Вони знаходяться в таблиці `android.provider.ContactsContract.StreamItemPhotos`.

4.6. Платформа доступу до сховища (Storage Access Framework)

Платформа доступу до сховища (Storage Access Framework, SAF) вперше з'явилася в Android версії 4.4 (API рівня 19). Платформа SAF полегшує користувачам пошук і відкриття документів, зображень та інших файлів у сховищах всіх постачальників, з якими вони працюють. Стандартний зручний інтерфейс дозволяє користувачам застосовувати єдиний для всіх додатків і постачальників спосіб пошуку файлів і доступу до останніх доданих файлів.

Хмарні або локальні служби зберігання можуть приєднатися до цієї екосистеми, реалізувавши клас `DocumentsProvider`, що інкапсулює їхні послуги. Клієнтські програми, яким потрібен доступ до документів постачальника, можуть інтегруватися з SAF за допомогою всього декількох рядків коду.

Платформа SAF включає в себе такі компоненти:

1. Постачальник документів – постачальник контенту, що дозволяє службі зберігання (наприклад, Диск Google) показувати файли, якими він управляє. Постачальник документів реалізується як підклас класу `DocumentsProvider`. Його схема заснована на традиційній файлової ієрархії, однак фізичний спосіб зберігання даних в постачальнику документів залишається на розсуд розробника. Платформа Android включає в себе кілька вбудованих постачальників документів, таких, як «Завантаження», «Зображення» і «Відео».

2. Клієнтська програма – користувацький додаток, що викликає намір `ACTION_OPEN_DOCUMENT` і/або `ACTION_CREATE_DOCUMENT` і приймає файли, які повертаються постачальниками документів.

3. Елемент вибору – системний елемент, що забезпечує користувачам доступ до документів у всіх постачальників документів, які задовольняють критерії пошуку, заданим в клієнтському додатку.

Платформа SAF серед інших надає такі функції:

- дозволяє користувачам шукати контент у всіх постачальників документів, а не тільки у однієї програми;
- забезпечує додатку можливість довготривалого, постійного доступу до документів, що належить постачальнику документів. Завдяки такому доступу

користувачі можуть додавати, редагувати, зберігати і видаляти файли, що зберігаються у постачальника;

– підтримує декілька облікових записів і тимчасові кореневі каталоги, наприклад постачальники на USB-накопичувачах, які з'являються, тільки коли накопичувач вставлений в порт.

4.6.1. Огляд

У центрі платформи SAF знаходиться постачальник контенту, який є підкласом класу `DocumentsProvider`. Усередині постачальника документів дані мають структуру традиційної файлової ієрархії (рис. 4.6):

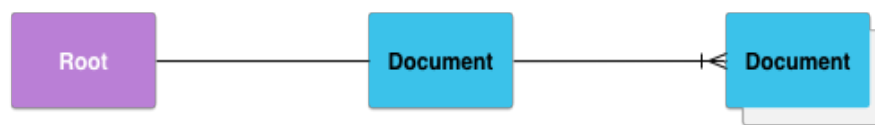


Рисунок 4.6 – Модель даних постачальника документів

На цьому рисунку **Root** (кореневий каталог) вказує на один об'єкт **Document** (документ), який потім розгалужується в ціле дерево.

Зверніть увагу на таке:

- Кожен постачальник документів надає один або кілька «корневих каталогів», що є відправними точками при обході дерева документів. Кожен кореневий каталог має унікальний ідентифікатор `COLUMN_ROOT_ID` і вказує на документ (каталог), який подає вміст на рівні, що нижчий від кореневого. Кореневі каталоги динамічні за своєю конструкцією, щоб забезпечувати підтримку таких варіантів використання, як кілька облікових записів, тимчасові сховища на USB-накопичувачах і можливість для користувача увійти в систему і вийти з неї.

- У кожному кореновому каталозі знаходиться один документ. Цей документ вказує на кількість документів N , кожен з яких, в свою чергу, може вказувати на один або N документів.

- Кожен сервер сховища показує окремі файли і каталоги, посилаючись на них за допомогою унікального ідентифікатора `COLUMN_DOCUMENT_ID`. Ідентифікатори документів повинні бути унікальними і не змінюватися після присвоєння, оскільки вони використовуються для видачі постійних URI, що не залежать від перезавантаження пристрою.

- Документ – це або файл, що відкривається (має конкретний MIME-тип), або каталог, що містить інші документи (з MIME-типом `MIME_TYPE_DIR`).

- Кожен документ може мати різні властивості, описувані прапорами COLUMN_FLAGS, такими, як FLAG_SUPPORTS_WRITE, FLAG_SUPPORTS_DELETE і FLAG_SUPPORTS_THUMBNAIL. Документ з тим самим ідентифікатором COLUMN_DOCUMENT_ID може знаходитися в декількох каталогах.

4.6.2. Потік управління

Як було сказано вище, модель даних постачальника документів заснована на традиційній файлової ієрархії. Однак фізичний спосіб зберігання даних залишається на розсуд розробника, за умови, що до них можна звертатися через API-інтерфейс DocumentsProvider. Наприклад, можна використовувати для даних хмарне сховище на основі тегів.

На рис. 4.7 наведено приклад того, як додаток обробки фотографій може використовувати SAF для доступу до збережених даних.

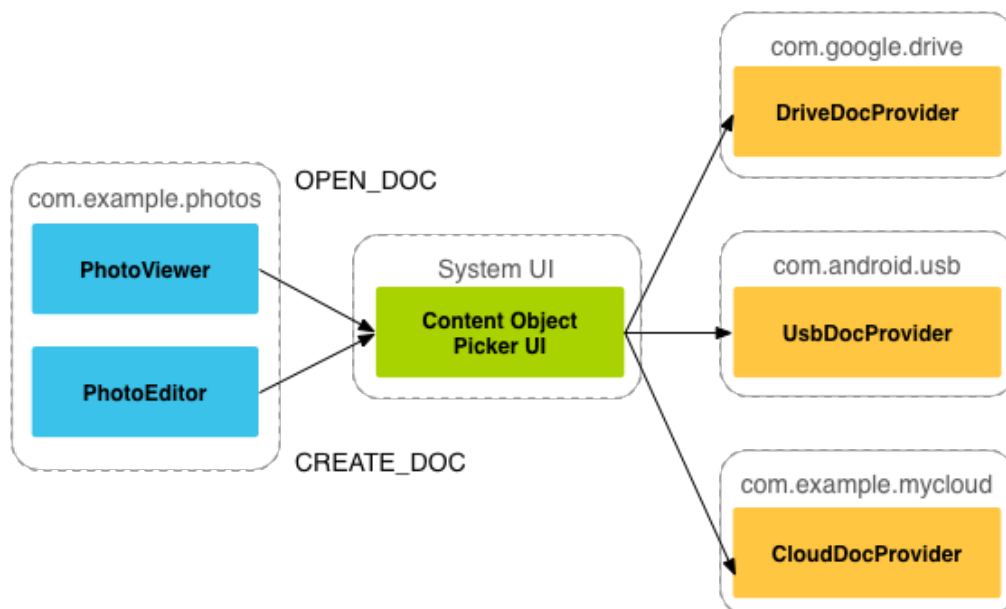


Рисунок 4.7 – Потік управління Storage Access Framework

Зверніть увагу на таке:

- На платформі SAF постачальники і клієнти не взаємодіють безпосередньо. Клієнт запитує дозвіл на взаємодію з файлами (тобто на читання, редагування, створення або видалення файлів).

- Взаємодія починається, коли додаток (в нашому прикладі обробляє фотографії) активізує намір ACTION_OPEN_DOCUMENT або ACTION_CREATE_DOCUMENT. Намір може включати в себе фільтри для уточнення критеріїв, наприклад, «надати відкриватися файлам з MIME-типом image».

- Коли намір спрацьовує, системний елемент вибору переходить до кожного зареєстрованого постачальника і показує користувачеві кореневі каталоги з контентом, відповідним до запиту.

- Елемент вибору надає користувачеві стандартний інтерфейс, навіть якщо постачальники документів значно різняться. Як приклад на рис. 4.7 зображені Диск Google, постачальник на USB-накопичувачі і хмарний постачальник.

На рис. 4.8 наведено елемент вибору, в якому користувач для пошуку зображень вибрав обліковий запис Диск Google.

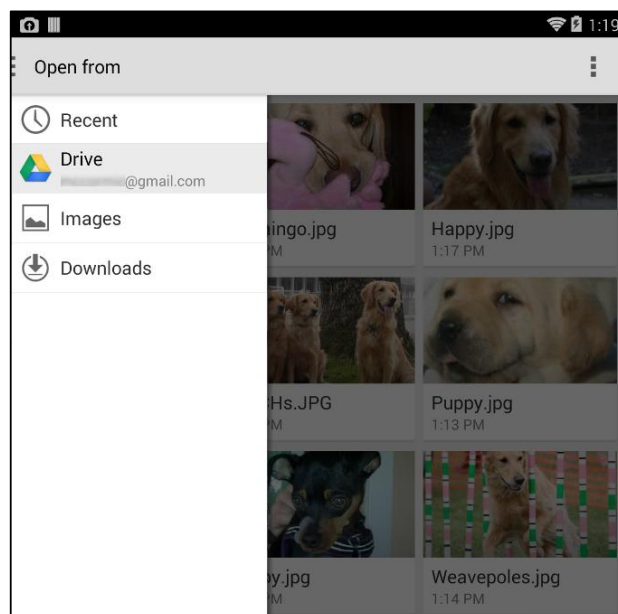


Рисунок 4.8 – Елемент вибору

Коли користувач вибирає Диск Google, зображення відображаються так, як показано на рис. 4.9. З цього моменту користувач може взаємодіяти з ними будь-якими способами, які підтримуються постачальником і клієнтським додатком.

4.6.3 Створення клієнтської програми

В Android версії 4.3 і нижче для того, щоб додаток міг отримувати файл від іншого додатка, він повинен активізувати намір, наприклад, ACTION_PICK або ACTION_GET_CONTENT. Після цього користувач повинен має вибрати якийсь один додаток, щоб отримати файл, а той, в свою чергу, повинен надати користувачеві інтерфейс, за допомогою якого він зможе вибирати і отримувати файли.

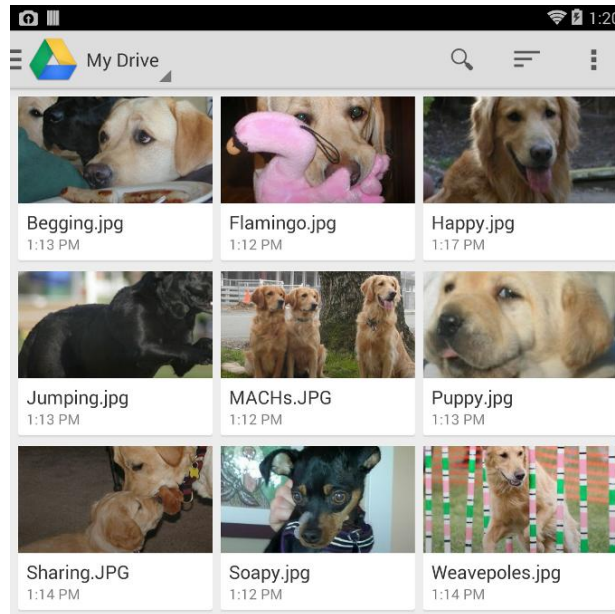


Рисунок 4.9 – Зображення

Починаючи з Android 4.4 і вище, у розробника є додаткова можливість – намір `ACTION_OPEN_DOCUMENT`, який відображає користувацький інтерфейс елемента вибору, керованого системою. Цей елемент надає користувачеві можливість огляду всіх файлів, доступних в інших додатках. Завдяки цьому єдиному інтерфейсу, користувач може вибрати файл в будь-якому з підтримуваних додатків.

Намір `ACTION_OPEN_DOCUMENT` не є заміною для наміру `ACTION_GET_CONTENT`. Розробнику слід використовувати те, що краще відповідає потребам програми:

1. Використовувати `ACTION_GET_CONTENT`, коли програму потрібно просто прочитати або імпортувати дані. При такому підході додаток імпортує копію даних, наприклад файл із зображенням.

2. Використовувати `ACTION_OPEN_DOCUMENT`, коли програмі потрібна можливість довготривалого, постійного доступу до документів, що належить постачальнику документів. Як приклад можна назвати редактор фотографій, що дозволяє користувачам обробляти зображення, які зберігаються в постачальнику документів.

У цьому розділі показано, як написати клієнтську програму, що використовує наміри `ACTION_OPEN_DOCUMENT` і `ACTION_CREATE_DOCUMENT`.

Пошук документів. У наступному фрагменті коду намір `ACTION_OPEN_DOCUMENT` використовується для пошуку постачальників документів, що містять файли зображень:

```

private static final int READ_REQUEST_CODE = 42; ...
/ ** * Запускає намір розкрити «вибір файлу» UI і вибрати зобра-
ження. * /
public void performFileSearch () {

    // ACTION_OPEN_DOCUMENT намір вибрати файл через файл системи //
браузер
    Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);

    // Фільтр, що показує тільки результати, що можуть бути «відкриті»,
    такі як // файл (на відміну від списку контактів або часових поясів)
    intent.addCategory (Intent.CATEGORY_OPENABLE);

    // Фільтр для відображення тільки зображень, використовуючи тип
    MIME даних зображення. // Якщо хтось хоче знайти Ogg Vorbis файлів,
    тип буде «аудіо / OGG». // Для пошуку всіх документів, доступних че-
    рез встановлений провайдер зберігання,
    // це було б "*" / "*".
    intent.setType ("image/*");

    startActivityForResult (intent, READ_REQUEST_CODE);}

```

Зверніть увагу на таке.

- Коли додаток активізує намір ACTION_OPEN_DOCUMENT, він запускає елемент вибору, який відображає всіх постачальників документів, що відповідають заданим критеріям.
- Додавання категорії CATEGORY_OPENABLE у фільтри намірів призводить до відображення тільки тих документів, які можна відкрити (наприклад, файлів із зображеннями).
- Оператор `intent.setType ("image / *")` виконує подальшу фільтрацію, щоб відображалися тільки документи з MIME-типом `image`.

Обробка результатів. Коли користувач вибирає документ в елементі вибору, викликається метод `onActivityResult()`. Ідентифікатор URI, який вказує на обраний документ, міститься в параметрі `resultData`. Щоб витягти URI, слід викликати `getData()`. Цей URI можна використовувати для отримання документа, потрібного користувачеві. Наприклад:

```

@Override
public void onActivityResult(int requestCode, int resultCode,
    Intent resultData) {

    // The ACTION_OPEN_DOCUMENT intent was sent with the request
    code

```

```

        // READ_REQUEST_CODE. If the request code seen here does not
        match, it's the
        // response to some other intent, and the code below should not
        run at all.

        if (requestCode == READ_REQUEST_CODE && resultCode == Activi-
        ty.RESULT_OK) {
            // The document selected by the user will not be returned in
            the intent.
            // Instead, a URI to that document will be contained in the
            return intent
            // provided to this method as a parameter.
            // Pull that URI using resultData.getData ().
            Uri uri = null;
            if (resultData != null) {
                uri = resultData.getData();
                Log.i(TAG, "Uri:" + uri());
            }
            showImage(uri);
        }
    }
}

```

Вивчення метаданих документа. Маючи в своєму розпорядженні URI документа, розробник отримує доступ до його метаданих. У наступному фрагменті коду метадані документа, що визначається ідентифікатором URI, зчитуються і записуються в журнал:

```

public void dumpImageMetaData(Uri uri) {

    // Запит, так як це відноситься тільки до одного документу, буде
    тільки повернення
    // один рядок. Там немає необхідності фільтрувати, сортувати,
    або вибрати поля, так як ми хочемо
    // всі поля для одного документа.
    Cursor cursor = getActivity().getContentResolver()
        .query(uri, null, null, null, null, null);

    try {
        // moveToFirst () повертає брехня, якщо курсор має 0 рядків. Ду-
        же зручно для
        // «якщо є щось дивитися, дивитися на нього» умовні.
        if (cursor != null && cursor.moveToFirst()) {

            // Зверніть увагу, що називається «Display Name». Це
            // постачальник-специфічним і може бути не обов'язково ім'я фай-
            лу.

            String displayName = cursor.getString(
            cursor.getColumnIndex(OpenableColumns.DISPLAY_NAME));
            Log.i(TAG, "Display Name:" + displayName);
        }
    }
}

```



```

        int sizeIndex = cursor.getColumnIndex(OpenableColumns.SIZE);
        // Якщо розмір невідомий, значення, що зберігається
        // дорівнює нулю. Але так як
        // INT не може бути порожнім в Java, поведінка залежить
        // від конкретної реалізації,
        // який просто фантазія термін «непередбачуваний». з
        // тим
        // правило, перевірити, якщо це нуль перед призначенням
        // на міжнр. Це буде
        // часто трапляється: API-інтерфейс для зберігання
        // дозволяє видалені файли, чиї
        // розмір не може бути локально відомо.
        String size = null;
        if (!cursor.isNull(sizeIndex)) {
            // З технічної точки зору, стовпець зберігає Int, але
            cursor.getString ()
            // буде автоматично виконувати перетворення.
            size = cursor.getString(sizeIndex);
        } else {
            size = "Unknown";
        }
        Log.i(TAG, "Size:" + size);
    }
} finally {
    cursor.close();
}
}

```

Відкриття документа. Отримавши URI документа, розробник може відкривати його і в цілому робити з ним все, що завгодно.

Об'єкт растрових зображень. Наведемо приклад коду для відкриття об'єкта Bitmap:

```

private Bitmap getBitmapFromUri(Uri uri) throws IOException {
    ParcelFileDescriptor parcelFileDescriptor =
    getContentResolver().openFileDescriptor(uri, "r");
    FileDescriptor fileDescriptor =
    parcelFileDescriptor.getFileDescriptor();
    Bitmap image =
    BitmapFactory.decodeFileDescriptor(fileDescriptor);
    parcelFileDescriptor.close();
    return image;
}

```

Зверніть увагу, що не слід проводити цю операцію в потоці призначеного для користувача інтерфейсу. Її потрібно виконувати у фоновому режимі, за допомогою AsyncTask. Коли файл з растровим зображенням відкриється, його можна відобразити в віджеті ImageView.

Отримання об'єкта InputStream. Нижче наведено приклад того, як можна отримати об'єкт `InputStream` за ідентифікатором URI. У цьому фрагменті коду зчитуються в об'єкт рядкового типу:

```
private String readTextFromUri(Uri uri) throws IOException {
    InputStream inputStream = getContentResolver().openInputStream(uri);
    BufferedReader reader = new BufferedReader(new InputStreamReader(
inputStream));
    StringBuilder stringBuilder = new StringBuilder();
    String line;
    while ((line = reader.readLine()) != null) {
stringBuilder.append(line);
    }
    fileInputStream.close();
    parcelFileDescriptor.close();
    return stringBuilder();
}
```

Створення нового документа. Додаток може створити новий документ в постачальнику документів, використовуючи намір `ACTION_CREATE_DOCUMENT`. Щоб створити файл, потрібно вказати в намірі MIME-тип і ім'я файлу, а потім запустити його з унікальним кодом запиту. Про інше подбає платформа:

```
// Ось деякі приклади того, як ви могли б назвати цей метод.
// Перший параметр є тип MIME, а другий параметр є ім'ям
// з створюваного файлу:
//
// CreateFile ("текст / звичайний", "foobar.txt")
// CreateFile ("зображення / PNG", "mypicture.png");
// Унікальний код запиту.
private static final int WRITE_REQUEST_CODE = 43;
...
private void createFile(String mimeType, String fileName) {
    Intent intent = new Intent(Intent.ACTION_CREATE_DOCUMENT);

    // Фільтр тільки результати показують, що можуть бути
    «відкритими», наприклад,
    // файл (на відміну від списку контактів або часових поясів).
    intent.addCategory(Intent.CATEGORY_OPENABLE);

    //Створення файлу з запитуваною типом MIME.
    intent.setType(mimeType);
    intent.putExtra(Intent.EXTRA_TITLE, fileName);
    startActivityResult(intent, WRITE_REQUEST_CODE);
}
```

Після створення нового документа можна отримати його URI за допомогою методу `onActivityResult ()`, щоб мати можливість записувати в нього дані.

Видалення документа. Якщо у розробника є URI документа, а об'єкт `Document.COLUMN_FLAGS` основних напрямів містить прапор `SUPPORTS_DELETE`, то документ можна видалити, наприклад:

```
DocumentsContract.deleteDocument (getContentResolver (), uri);
```

Редагування документа. Платформа SAF дозволяє редагувати текстові документи на місці. У наступному фрагменті коду активізується намір `ACTION_OPEN_DOCUMENT`, а категорія `CATEGORY_OPENABLE` використовується, щоб відображалися тільки документи, які можна відкрити. Потім проводиться подальша фільтрація, щоб відображалися тільки текстові файли:

```
private static final int EDIT_REQUEST_CODE = 44;
/** * Відкрити файл для запису і додати текст до нього. * /
private void editDocument() {
    // ACTION_OPEN_DOCUMENT - намір вибрати файл за допомогою системи
    // файловий браузер.
    Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);

    // Фільтр тільки результати показують, що можуть бути «відкриті»,
    такі як
    // файл (на відміну від списку контактів або часових поясів).
    intent.addCategory(Intent.CATEGORY_OPENABLE);

    // Фільтр для відображення тільки текстові файли.
    intent.setType("Text / plain");

    startActivityForResult(intent, EDIT_REQUEST_CODE);
}
```

Далі, з методу `onActivityResult()` (див. «Обробка результатів») можна викликати код для виконання редагування. У наступному фрагменті коду об'єкт `FileOutputStream` отриманий за допомогою об'єкта класу `ContentResolver`. За замовчуванням використовується режим запису. Рекомендується запитувати мінімально необхідні права доступу, тому не треба запитувати читання/запис, коли програмі необхідно тільки записати у файл:

```
private void alterDocument(Uri uri) {
    try {
        ParcelFileDescriptor pfd =
        getActivity().getContentResolver().
```

```

openFileDescriptor(uri, "W");
    FileOutputStream fileOutputStream =
        new FileOutputStream(pfd.getFileDescriptor());
fileOutputStream.write(("Overwritten by MyCloud at" +
    System.currentTimeMillis() + "\ N").getBytes());
    // Нехай провайдер документа знаю, що ви зробили, закривши
потік.
fileOutputStream.close();
pfd.close();
    } catch (FileNotFoundException e) {
e.printStackTrace();
    } catch (IOException e) {
e.printStackTrace();
    }
}

```

Утримання прав доступу. Коли додаток відкриває файл для читання або запису, система надає йому URI-дозвіл на цей файл. Дозвіл діє аж до перезавантаження пристрою. Припустимо, що в графічному редакторі потрібно, щоб у користувача була можливість відкрити безпосередньо в цьому додатку останні п'ять зображень, які він редагував. Якщо він перезапустив пристрій, виникає необхідність знову відсилати його до системного елемента вибору для пошуку файлів. Очевидно, це далеко не ідеальний варіант.

Щоб уникнути такої ситуації, розробник може утримати права доступу, надані системою його додатком. Додаток фактично бере постійний URI-дозвіл, пропонуваний системою. В результаті користувач отримує безперервний доступ до файлів з програми, незалежно від перезавантаження пристрою:

```

final int takeFlags = intent.getFlags()
    & (Intent.FLAG_GRANT_READ_URI_PERMISSION
    | Intent.FLAG_GRANT_WRITE_URI_PERMISSION);
// Перевіряє найсвіжіші дані.
getContentResolver().takePersistableUriPermission(uri, takeFlags);

```

Залишається один заключний крок. Можна зберегти останні URI-ідентифікатори, з якими працював додаток. Однак не виключено, що вони втраять актуальність, оскільки інший додаток може видалити або модифікувати документ. Тому слід завжди викликати `getContentResolver().takePersistableUriPermission()`, щоб отримувати актуальні дані.

4.6.4. Створення власного постачальника документів

При розробці програми, що надає послуги зі зберігання файлів (наприклад, служби зберігання в хмарі), можна надати доступ до файлів за допомогою

SAF, написавши власний постачальник документів. У цьому розділі показано, як це зробити.

Маніфест. Щоб реалізувати власний постачальник документів, необхідно додати в маніфест додатка таку інформацію:

1. Цільовий API-інтерфейс рівня 19 або вище.
2. Елемент `<Provider>`, у якому оголошується нестандартний постачальник сховища.

3. Ім'я постачальника, тобто, ім'я його класу з ім'ям пакета, наприклад: `com.example.android.storageprovider.MyCloudProvider`.

4. Ім'я центру постачальника, тобто ім'я пакета (в цьому прикладі – `com.example.android.storageprovider`) з типом постачальника контенту (documents).
наприклад, `com.example.android.storageprovider.documents`.

5. Атрибут `android:exported`, встановлений в значення "True". Необхідно експортувати постачальник, щоб його було видно іншим додаткам.

6. Атрибут `android:grantUriPermissions`, встановлений в значення "True". Цей параметр дозволяє системі надавати іншим програмам доступ до контенту постачальника.

7. Дозвіл `MANAGE_DOCUMENTS`. За замовчуванням постачальник доступний всім. Додавання цього дозволу в маніфест робить постачальник доступним тільки системі. Це важливо для забезпечення безпеки.

8. Атрибут `android:enabled`, що має логічне значення, визначене в файлі ресурсів. Цей атрибут призначений для відключення постачальника на пристроях під управлінням Android версії 4.3 і нижче. Наприклад: `android:enabled="@bool/atLeastKitKat"`. Крім включення цього атрибута в маніфест, необхідно зробити таке:

- 1) у файл ресурсів `bool.xml`, що розташований в каталозі `res/values/`, додати рядок

```
<bool name="AtLeastKitKat">false</ Bool>
```

- 2) у файл ресурсів `bool.xml`, що розташований в каталозі `res/values-v19/`, додати рядок

```
<bool name="AtLeastKitKat">true</ Bool>
```

9. Фільтр наміру `android.content.action.DOCUMENTS_PROVIDER`, дозволяє відобразити постачальників в елементі вибору, коли система буде шукати постачальників.

Нижче наведено уривки зі зразка маніфесту, що включає в себе постачальник:

```
<manifest... >
...<Uses-sdk
    android:minSdkVersion="19"
    android:targetSdkVersion="19" />
....<provider
    an-
droid:name="com.example.android.storageprovider.MyCloudProvider"
    an-
droid:authorities="com.example.android.storageprovider.documents"
    android:grantUriPermissions="True"
    android:exported="True"
    android:permission="Android.permission.MANAGE_DOCUMENTS"
    android:enabled="@ Bool / atLeastKitKat">
    <Intent-filter>
        <action android:
name="Android.content.action.DOCUMENTS_PROVIDER" />
    </Intent-filter>
    </provider>
</ Application>

</ Manifest>
```

Підтримка пристроїв під управлінням Android версії 4.3 і нижче. Намір ACTION_OPEN_DOCUMENT доступний тільки на пристроях з Android версії 4.4 і вище. Якщо додаток має підтримувати ACTION_GET_CONTENT, щоб обслуговувати пристрої, які працюють під управлінням Android 4.3 і нижче, необхідно відключити фільтр наміру ACTION_GET_CONTENT у маніфесті для пристроїв з Android версії 4.4 і вище. Постачальник документів і намір ACTION_GET_CONTENT слід вважати взаємовиключними. Якщо додаток підтримує їх одночасно, він буде з'являтися в інтерфейсі системного елемента вибору двічі, пропонуючи два різні способи доступу до збережених даних. Це заплутає користувачів.

Відключати фільтр наміру ACTION_GET_CONTENT на пристроях з Android версії 4.4 і вище рекомендується таким чином:

1. У файл ресурсів bool.xml, розташований в каталозі res/values/, додати наступний рядок:

```
<bool name="AtMostJellyBeanMR2">true</ Bool>
```

2. У файл ресурсів bool.xml, розташований в каталозі res/values-v19/, додати наступний рядок:

```
<bool name="AtMostJellyBeanMR2">false</ Bool>
```

3. Додати псевдонім операції, щоб відключити фільтр наміри ACTION_GET_CONTENT для версій 4.4 (API рівня 19) і вище. Наприклад:

```
<!-- Цей псевдонім активності додається таким чином, щоб GET_CONTENT наміри  
фільтр може бути відключений для збірки на рівні API 19 і вище. -->  
<Activity-alias android: name="Com.android.example.app.MyPicker"  
    android: targetActivity="Com.android.example.app.MyActivity"  
    ...android: enabled="@ Bool / atMostJellyBeanMR2">  
    <Intent-filter>  
        <action android: name="Android.intent.action.GET_CONTENT" />  
        <category android: name="Android.intent.category.OPENABLE" />  
        <category android: name="Android.intent.category.DEFAULT" />  
        <data android: mimeType="Image / *" />  
        <data android: mimeType="Video / *" />  
    </ Intent-filter>  
</ Activity-alias>
```

Контракти. Як правило, при створенні нестандартного постачальника контенту одним із завдань є реалізація класів-контрактів. Клас-контракт являє собою клас `public final`, в якому містяться визначення констант для URI, імен стовпців, типів MIME та інших метаданих постачальника. Платформа SAF надає розробнику наступні класи-контракти, а тому йому не потрібно писати власні:

- DocumentsContract.Document;
- DocumentsContract.Root.

Наприклад, коли до постачальника документів приходить запит на документи або кореневий каталог, можна повертати в курсорі такі стовпчики:

```
private static final String[] DEFAULT_ROOT_PROJECTION =  
    new String[] {Root.COLUMN_ROOT_ID, Root.COLUMN_MIME_TYPES,  
        Root.COLUMN_FLAGS, Root.COLUMN_ICON, Root.COLUMN_TITLE,  
        Root.COLUMN_SUMMARY, Root.COLUMN_DOCUMENT_ID,  
        Root.COLUMN_AVAILABLE_BYTES,};  
private static final String[] DEFAULT_DOCUMENT_PROJECTION = new  
    String[] {Document.COLUMN_DOCUMENT_ID, Document.  
COLUMN_MIME_TYPE,  
        Document.COLUMN_DISPLAY_NAME, Document.COLUMN_LAST_MODIFIED,  
        Document.COLUMN_FLAGS, Document.COLUMN_SIZE,};
```

Створення підкласу класу DocumentsProvider. Наступним кроком в розробці власного постачальника документів є створення підкласу абстрактного класу DocumentsProvider. Як мінімум, необхідно реалізувати такі методи:

- queryRoots();

- `queryChildDocuments()`;
- `queryDocument()`;
- `openDocument()`.

Це єдині методи, реалізація яких суворо обов'язкова, проте існує набагато більше методів, які, можливо, теж доведеться реалізувати. Подробиці наводяться в описі класу `DocumentsProvider`.

Реалізація методу `queryRoots`. Реалізація методу `queryRoots()` повинна повертати об'єкт `Cursor`, який вказує на всі кореневі каталоги постачальників документів, використовуючи стовпці, визначені в `DocumentsContract.Root`.

У наступному фрагменті коду параметр `projection` представляє конкретні поля, потрібні об'єкту, що викликається. У цьому коді створюється курсор, і до нього додається один рядок, що відповідає одному кореневому каталогу (каталогу верхнього рівня), наприклад, «Завантаження» або «Зображення». Більшість постачальників має тільки один кореневий каталог. Однак ніщо не заважає мати кілька кореневих каталогів, наприклад, за наявності декількох облікових записів. У цьому випадку досить додати в курсор ще один рядок.

```
@Override
public Cursor queryRoots(String[] projection) throws
FileNotFoundException {

    // Створення курсора або з запитуваних полів або за замовчуван-
    ням
    // проекція, якщо «проекція» дорівнює нулю.
    final MatrixCursor result =
        new MatrixCursor(resolveRootProjection(projection));

    // Якщо користувач не увійшов в систему, повертає порожній коре-
    невої курсор. Це усуває OUR
    // постачальник зі списку цілком.
    if (!isUserLoggedIn()) {
        return result;
    }

    // Можна мати кілька коренів (наприклад, для декількох облікових
    записів в
    // те ж саме додаток) - просто додайте кілька рядків курсора.
    // Побудувати один рядок для кореня під назвою «MyCloud».
    final MatrixCursor.RowBuilder row = result.newRow();
    row.add(Root.COLUMN_ROOT_ID, ROOT);
    row.add(Root.COLUMN_SUMMARY,
        getContext().getString(R.string.root_summary));

    // FLAG_SUPPORTS_CREATE означає, щонайменше, один каталог під ко-
    ренем опор
    // створення документів. FLAG_SUPPORTS_RECENTS означає, щонай-
```



```

менш, один каталог під коренем підтримує
// Створення документа.
// FLAG_SUPPORTS_SEARCH дозволяє користувачам шукати всі докумен-
ти додатки
// розділяють
row.add(Root.COLUMN_FLAGS, Root.FLAG_SUPPORTS_CREATE |
        Root.FLAG_SUPPORTS_RECENTS |
        Root.FLAG_SUPPORTS_SEARCH);

// COLUMN_TITLE це корнева назва (наприклад, галерея, Drive).
row.add(Root.COLUMN_TITLE, getContext().getString(R.string.title));

// Цей ідентифікатор документа не може змінитися, як тільки він
розподілений.
row.add(Root.COLUMN_DOCUMENT_ID, getDocIdForFile(mBaseDir));

// Дочірні типи MIME використовуються для фільтрації коріння і
присутній тільки в
// Корені користувачів, які містять потрібний тип десь в їх іє-
рархії файлів.
row.add(Root.COLUMN_MIME_TYPES, getChildMimeTypes(mBaseDir));
row.add(Root.COLUMN_AVAILABLE_BYTES, mBaseDir.getFreeSpace());
row.add(Root.COLUMN_ICON, R.drawable.ic_launcher);

return result;
}

```

Реалізація методу queryChildDocuments. Реалізація методу queryChildDocuments() повинна повертати об'єкт Cursor, який вказує на всі файли в заданому каталозі, використовуючи стовпці, визначені в DocumentsContract.Document.

Цей метод викликається, коли в інтерфейсі елемента вибору користувач вибирає кореневий каталог додатка. Метод отримує документи-нащадки каталогу на рівні, що нижче від кореневого. Його можна викликати на будь-якому рівні файлової ієрархії, а не тільки в кореновому каталозі. У наступному фрагменті коду створюється курсор з запитаними стовпцями. Потім в нього заноситься інформація про кожного найближчого нащадка батьківського каталогу. Нащадком може бути зображення, ще один каталог, загалом, будь-який файл:

```

@Override
public Cursor queryChildDocuments(String parentDocumentId, String[]
projection,
                                String sortOrder) throws
FileNotFoundException {

    final MatrixCursor result = new

```

```

        MatrixCursor(resolveDocumentProjection(projection));
final File parent = getFileForDocId(parentDocumentId);
for (File file : parent.listFiles()) {
    // Додає ім'я файлу відображення, MIME тип, розмір і так
дали.
    includeFile(result, null, file);
}
return result;
}

```

Реалізація методу queryDocument. Реалізація методу queryDocument() повинна повертати об'єкт Cursor, який вказує на заданий файл, використовуючи стовпці, визначені в DocumentsContract.Document.

Метод queryDocument() повертає ту ж інформацію, яку повертав queryChildDocuments(), але для конкретного файлу:

```

@Override
public Cursor queryDocument(String documentId, String[] projection)
throws
    FileNotFoundException {

    // Створює курсор із запитуваною проекцією або проекцією за замов-
чуванням.
    final MatrixCursor result = new
        MatrixCursor(resolveDocumentProjection(projection));
includeFile(result, documentId, null);
return result;
}

```

Реалізація методу openDocument. Необхідно реалізувати метод openDocument(), який повертає об'єкт ParcelFileDescriptor, що подає зазначений файл. Інші додатки зможуть скористатися повернутим об'єктом ParcelFileDescriptor для організації потоку даних. Система викликає цей метод, коли користувач вибирає файл, і клієнтський додаток запитує доступ нього, викликаючи метод openFileDescriptor(). Наприклад:

```

@Override
public ParcelFileDescriptor openDocument(final String documentId,
final String mode,
CancellationSignal signal) throws
    FileNotFoundException {
    Log.v(TAG, "OpenDocument, mode:" + mode);
    // Це нормально робити мережеві операції в цьому методі, щоб за-
вантажити документ,
    // до тих пір, як ви періодично перевіряти CancellationSignal.

```

```

Якщо у вас є
    // дуже великий файл для передачі по мережі, краще рішення може
    // бути труби або розетки (див ParcelFileDescriptor для
    допоміжних методів).

    final File file = getFileForDocId(documentId);

    final boolean isWrite = (mode.indexOf('W') != -1);
    if(isWrite) {
        // Прикріплення близького слухача, якщо документ відкритий в
        режимі запису.
        try {
            Handler handler = new
Handler(getContext().getMainLooper());
            return ParcelFileDescriptor.open(file, accessMode, han-
dler,
                new ParcelFileDescriptor.OnCloseListener() {
                    @Override
                    public void onClose(IOException e) {

                        // Прикріплення близького слухача, якщо документ
                        відкритий в режимі запису.
                        Log.i(TAG, "A file with id" +
                            documentId + "Has been closed! Time to" +
                            "Update the server.");
                    }
                });
        } catch (IOException e) {
            throw new FileNotFoundException("Failed to open document
with id"
                + documentId + "And mode" + mode);
        }
    } else {
        return ParcelFileDescriptor.open(file, accessMode);
    }
}

```

Безпека. Припустимо, що постачальник документів являє собою захищену паролем службу зберігання в хмарі, а додаток має переконатися, що користувач увійшов в систему, перш ніж він надасть йому доступ до файлів. Які заходи має вжити додаток, якщо користувач не виконав вхід? Рішення полягає в тому, щоб реалізація методу `queryRoots()` не повертала кореневих каталогів. Іншими словами, це повинен бути порожній кореневий курсор:

```

public Cursor queryRoots(String[] projection) throws
FileNotFoundException {
    ...
    // Якщо користувач не увійшов в систему, повертає порожній кореневої
    курсор. Це усуває OUR

```

```
// постачальник зі списку цілком.
if (!isUserLoggedIn()) {
    return result;
}
```

Наступний крок полягає у виклику методу `getContentResolver().notifyChange()`.

Пам'ятаєте об'єкт `DocumentsContract`? Скористаємося ним для створення відповідного URI. У наступному фрагменті коду система сповіщається про необхідність опитувати кореневі каталоги постачальника документів, коли змінюється статус входу користувача в систему. Якщо користувач не виконав вхід, метод `queryRoots()` поверне порожній курсор, як показано вище. Це гарантує, що документи постачальника будуть доступні тільки користувачам, що ввійшли в постачальник:

```
private void onLoginButtonClick() {
    loginOrLogout();
    getContentResolver().notifyChange(DocumentsContract
        .buildRootsUri(AUTHORITY), null);
}
```

Запитання для самоконтролю

1. Назвіть основні типи сховищ для даних, що доступні мобільному додатку на платформі Android.
2. Яким є призначення постачальника контактів?
3. Що таке локальна база даних SQLite?
4. Опишіть процес отримання даних за допомогою завантажувачів (Loaders).
5. Опишіть можливості при доступі до вбудованих баз даних на прикладі «Постачальника контактів».
6. Опишіть можливості платформи доступу до сховищ SAF.