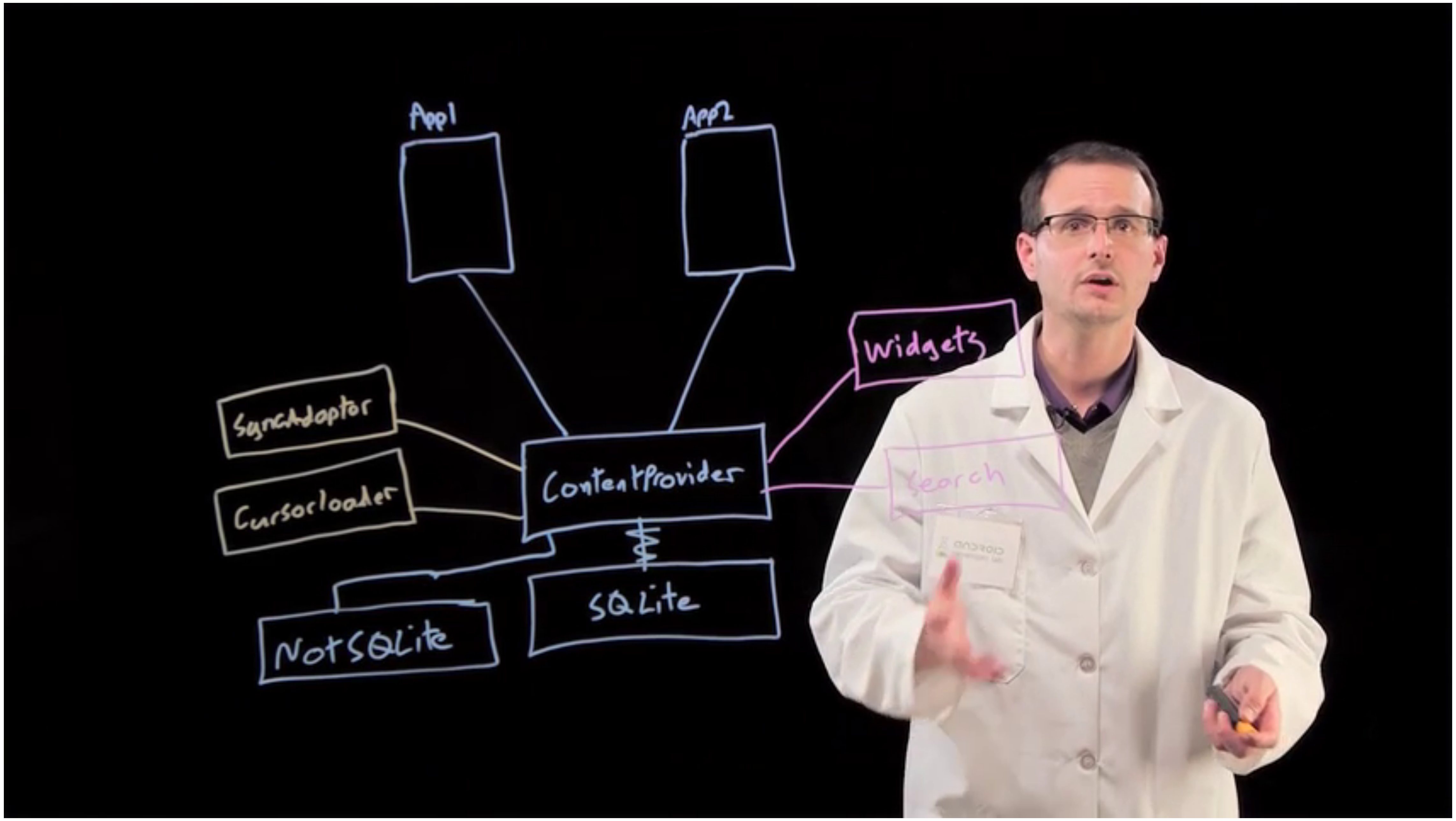
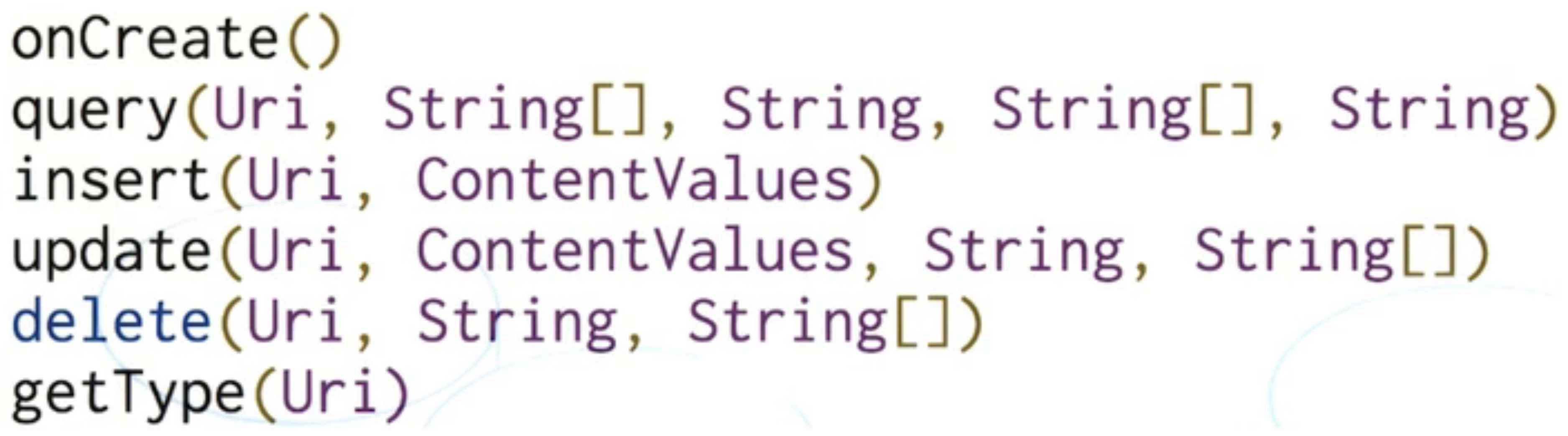
ContentProvider

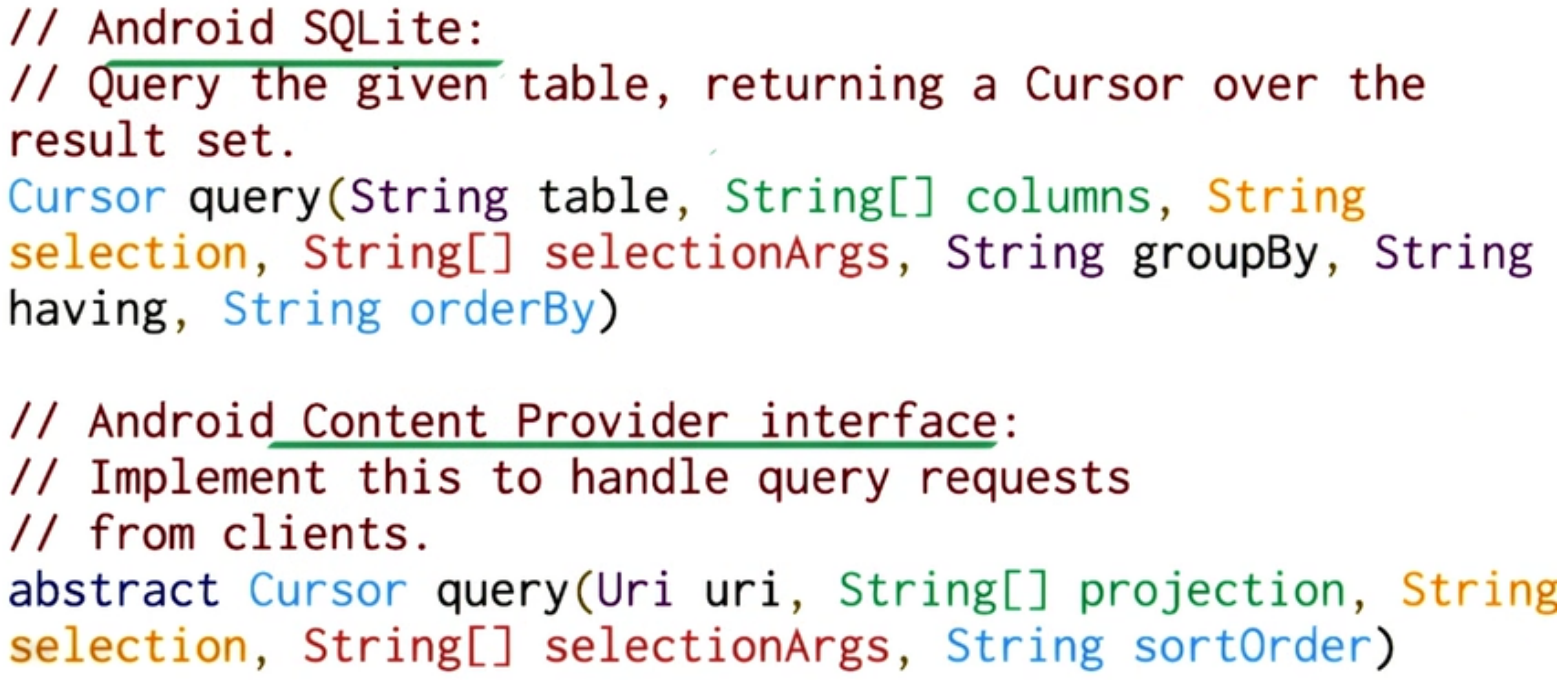

App1
App2
Widgets
SyncAdaptor
ContentProvider
Search
CursorLoader
NotSQLite
SQLite

Functions of a ContentProvider



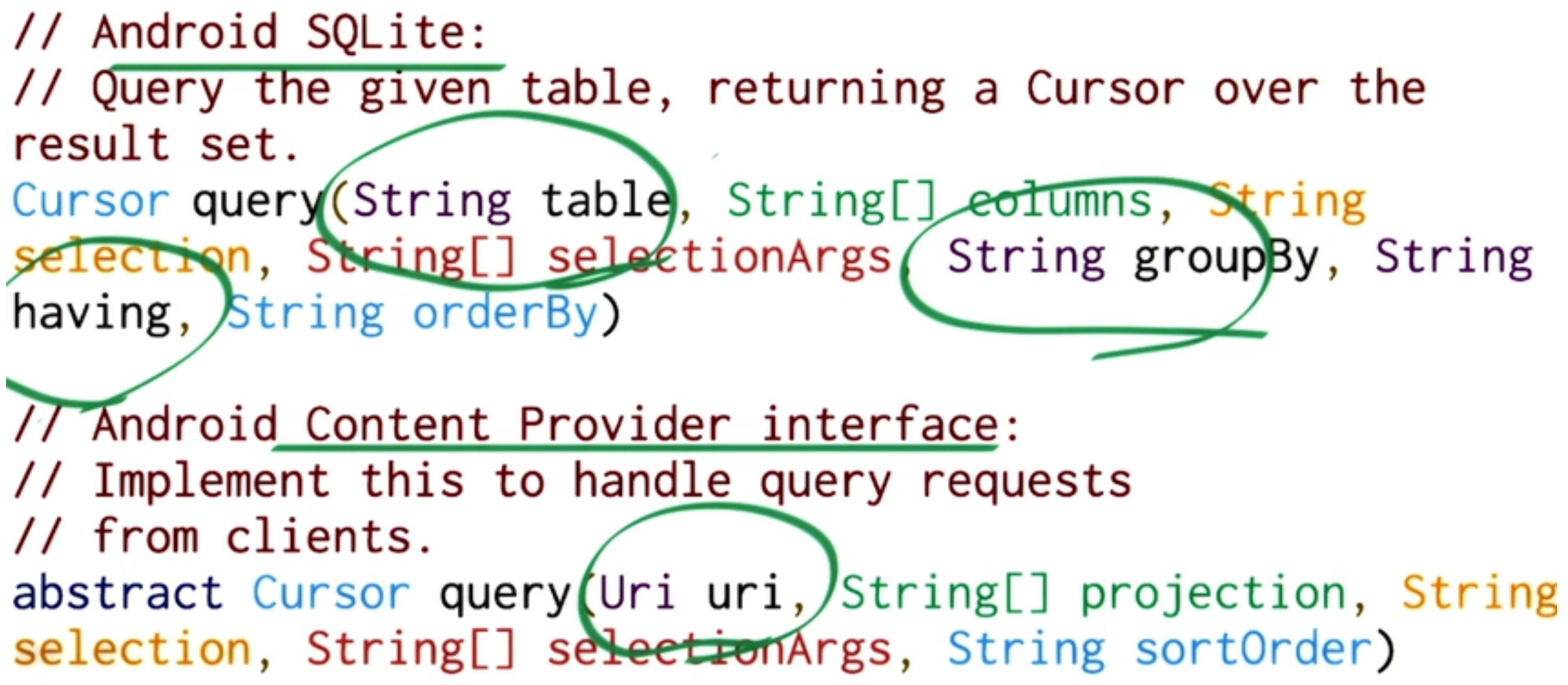
```
onCreate()
query(Uri, String[], String, String[], String)
insert(Uri, ContentValues)
update(Uri, ContentValues, String, String[])
delete(Uri, String, String[])
getType(Uri)
```

Parameter des ContentProviders



```
// Android SQLite:
// Query the given table, returning a Cursor over the
result set.
Cursor query(String table, String[] columns, String
selection, String[] selectionArgs, String groupBy, String
having, String orderBy)

// Android Content Provider interface:
// Implement this to handle query requests
// from clients.
abstract Cursor query(Uri uri, String[] projection, String
selection, String[] selectionArgs, String sortOrder)
```

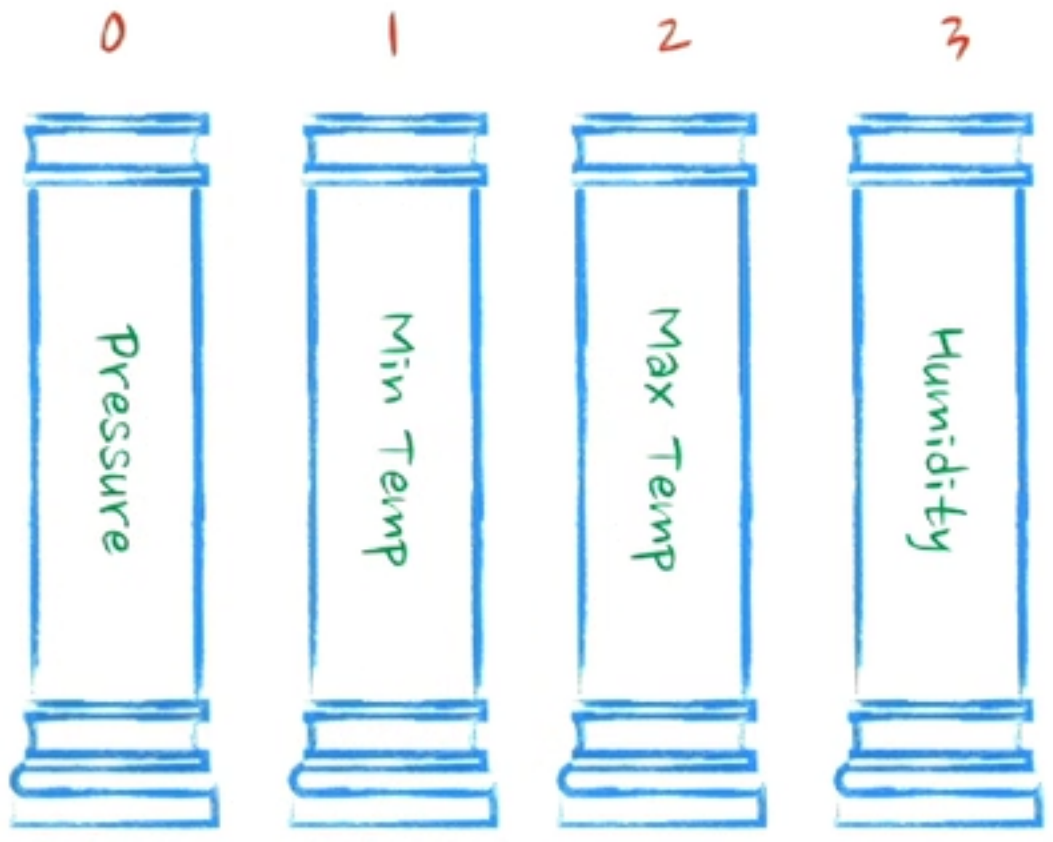
Die Parameter einer Content Provider - Query entsprechen nahezu exakt den Parametern für eine Datenbank - Query



```
// Android SQLite:
// Query the given table, returning a Cursor over the
result set.
Cursor query(String table, String[] columns, String
selection, String[] selectionArgs, String groupBy, String
having, String orderBy)

// Android Content Provider interface:
// Implement this to handle query requests
// from clients.
abstract Cursor query(Uri uri, String[] projection, String
selection, String[] selectionArgs, String sortOrder)
```

Anstelle eines Strings für eine Table wird ein Uri-Typ verwendet. Beachte ausserdem, dass es Parameter gibt - wie zB die GROUPing Funktionalität - die durch Content Provider nicht so einfach verarbeitet werden können.

Zugriff auf Spalten



```
float pressure =
    weatherCursor.getFloat(0);
float min = weatherCursor.getFloat(1);
float max = weatherCursor.getFloat(2);
float humidity =
    weatherCursor.getFloat(3);
```

```
int pressureIndex =
getColumnIndex(WeatherEntry.COLUMN_PRESSURE);
```

Der Zugriff auf die Spalten kann durch einen numerischen Index erfolgen. Die Verwendung einer eigenen getColumnIndex(…) - Funktion ist nicht notwendig

CONTRACT

THIS AGREEMENT, MADE THIS _____ DAY OF _____ DESCRIBES THE DATA TO BE EXCHANGED BETWEEN OUR DATA MODEL AND OUR VIEW, AGREED TO BY OUR CONTROLLER, [illegible]

[illegible]

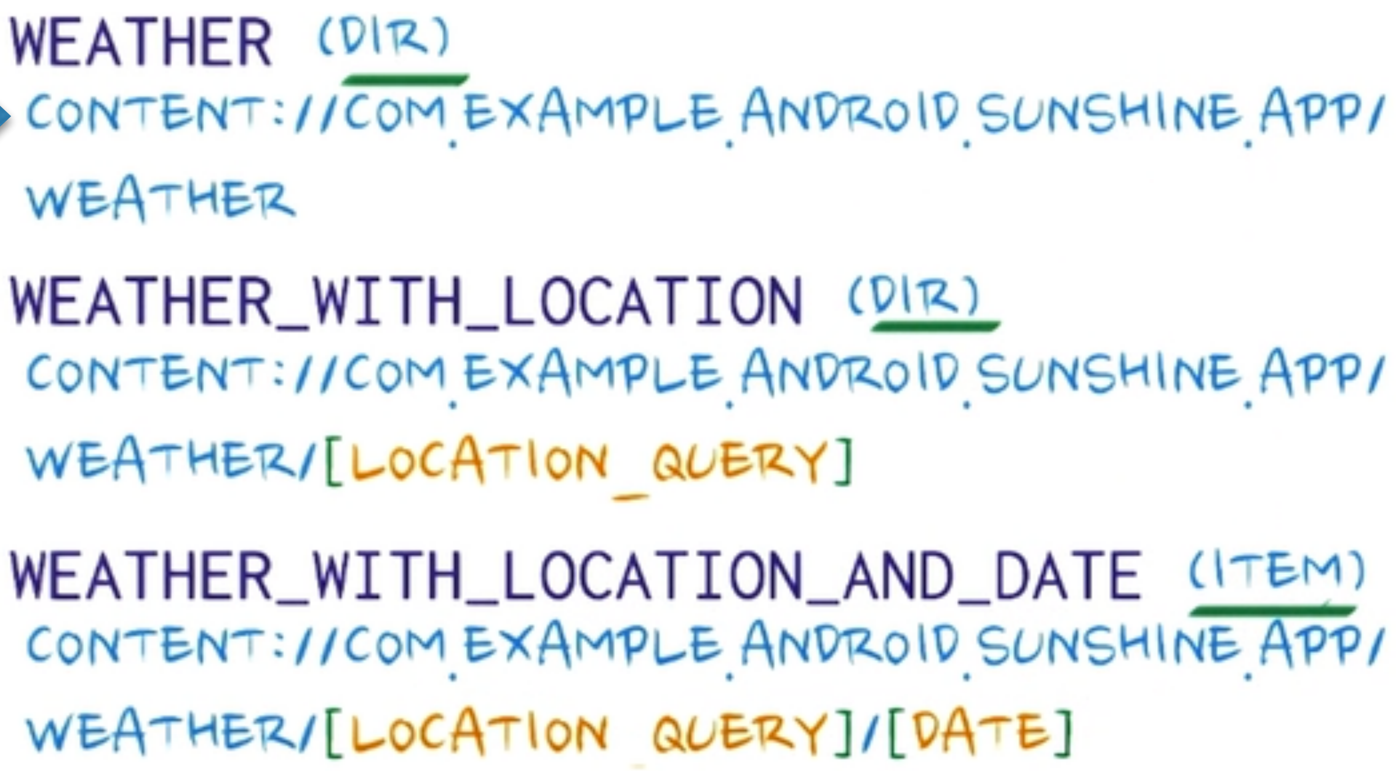
SIGNED

____
THE DATA MODEL

Der Contract wird sowohl für Database-Definitions als auch ContentProvider Definitions verwendet.

Bestandteile des Vertrages

Content authority



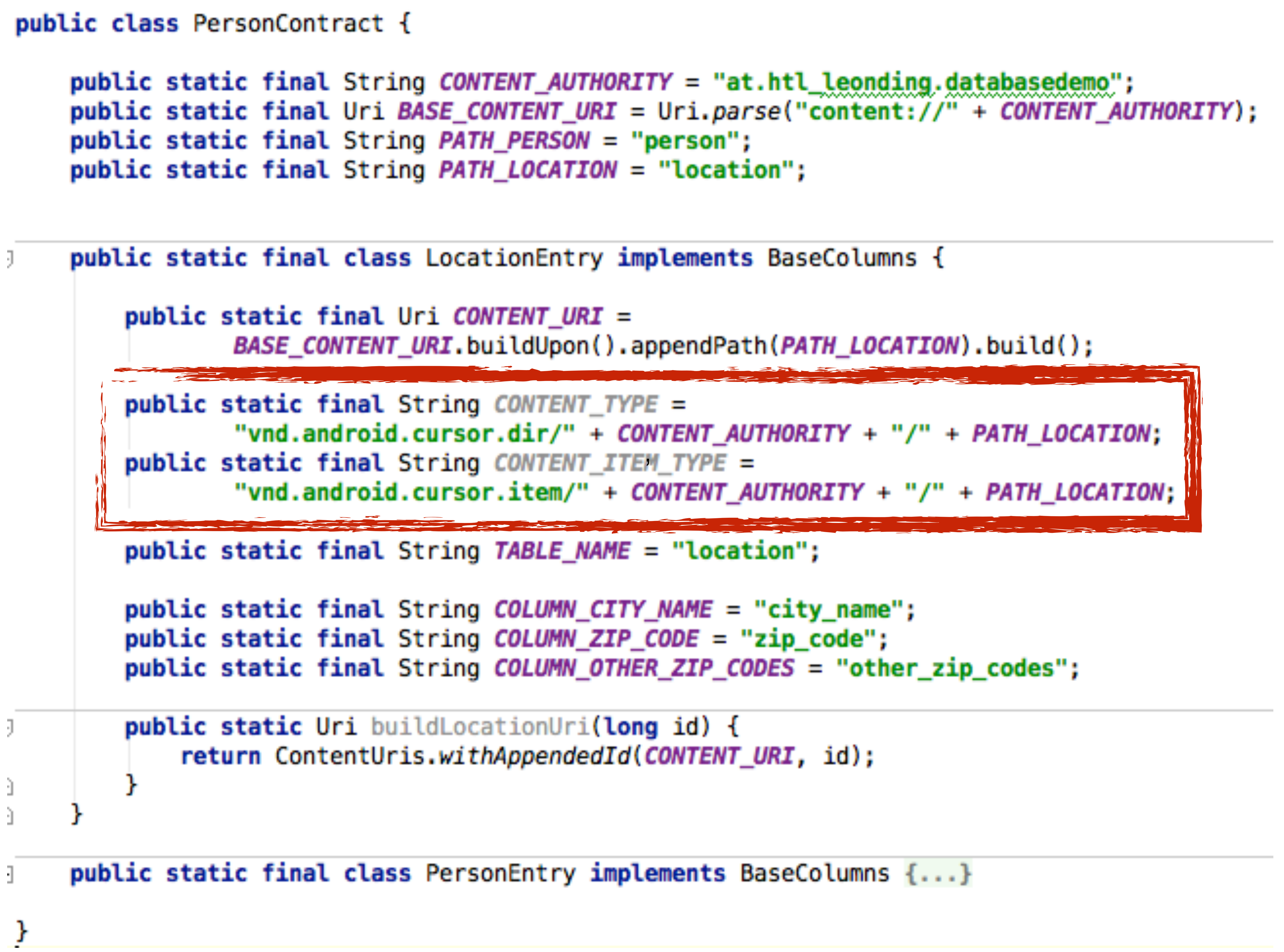
WEATHER (DIR)
CONTENT://COM.EXAMPLE.ANDROID.SUNSHINE.APP/
WEATHER

WEATHER_WITH_LOCATION (DIR)
CONTENT://COM.EXAMPLE.ANDROID.SUNSHINE.APP/
WEATHER/[LOCATION_QUERY]

WEATHER_WITH_LOCATION_AND_DATE (ITEM)
CONTENT://COM.EXAMPLE.ANDROID.SUNSHINE.APP/
WEATHER/[LOCATION_QUERY]/[DATE]

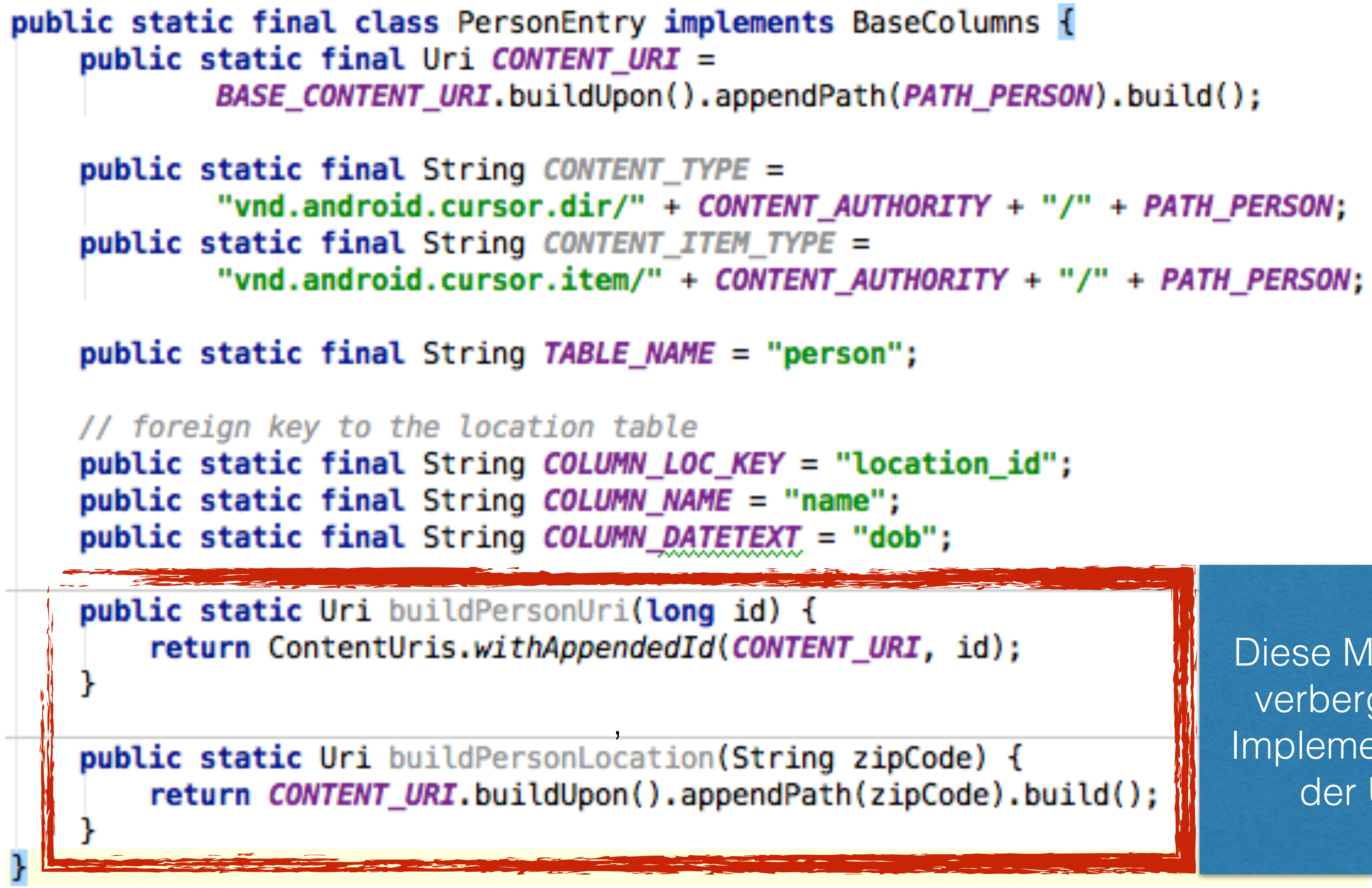
Die content authority gibt an, wie sich die URIs der App sich von anderen Apps unterscheiden (vgl. die DomainName einer Website). Meist wird hierfür der Package-Name verwendet

Der Vertrag (Contract) wird durch die URIs bestimmt. Manche URIs geben eine Liste zurück (DIR), andere URIs einzelne Objekte (ITEM)



```java
public class PersonContract {

    public static final String CONTENT_AUTHORITY = "at.htl_leonding.databasedemo";
    public static final Uri BASE_CONTENT_URI = Uri.parse("content://" + CONTENT_AUTHORITY);
    public static final String PATH_PERSON = "person";
    public static final String PATH_LOCATION = "location";


    public static final class LocationEntry implements BaseColumns {

        public static final Uri CONTENT_URI =
                BASE_CONTENT_URI.buildUpon().appendPath(PATH_LOCATION).build();

        public static final String CONTENT_TYPE =
                "vnd.android.cursor.dir/" + CONTENT_AUTHORITY + "/" + PATH_LOCATION;
        public static final String CONTENT_ITEM_TYPE =
                "vnd.android.cursor.item/" + CONTENT_AUTHORITY + "/" + PATH_LOCATION;

        public static final String TABLE_NAME = "location";

        public static final String COLUMN_CITY_NAME = "city_name";
        public static final String COLUMN_ZIP_CODE = "zip_code";
        public static final String COLUMN_OTHER_ZIP_CODES = "other_zip_codes";

        public static Uri buildLocationUri(long id) {
            return ContentUris.withAppendedId(CONTENT_URI, id);
        }
    }

    public static final class PersonEntry implements BaseColumns {...}

}
```

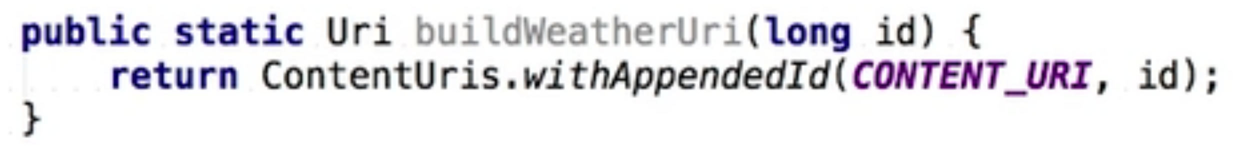
Uri-Builder und Decoder Functions



```
public static final class PersonEntry implements BaseColumns {
    public static final Uri CONTENT_URI =
            BASE_CONTENT_URI.buildUpon().appendPath(PATH_PERSON).build();

    public static final String CONTENT_TYPE =
            "vnd.android.cursor.dir/" + CONTENT_AUTHORITY + "/" + PATH_PERSON;
    public static final String CONTENT_ITEM_TYPE =
            "vnd.android.cursor.item/" + CONTENT_AUTHORITY + "/" + PATH_PERSON;

    public static final String TABLE_NAME = "person";

    // foreign key to the location table
    public static final String COLUMN_LOC_KEY = "location_id";
    public static final String COLUMN_NAME = "name";
    public static final String COLUMN_DATETEXT = "dob";

    public static Uri buildPersonUri(long id) {
        return ContentUris.withAppendedId(CONTENT_URI, id);
    }

    public static Uri buildPersonLocation(String zipCode) {
        return CONTENT_URI.buildUpon().appendPath(zipCode).build();
    }
}
```

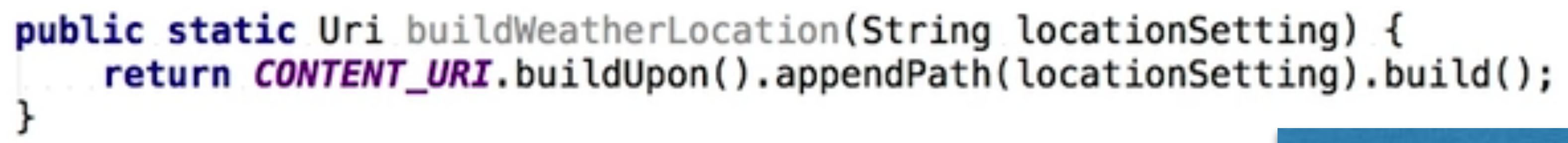
Diese Methoden verbergen die Implementierung der URIs

Queries



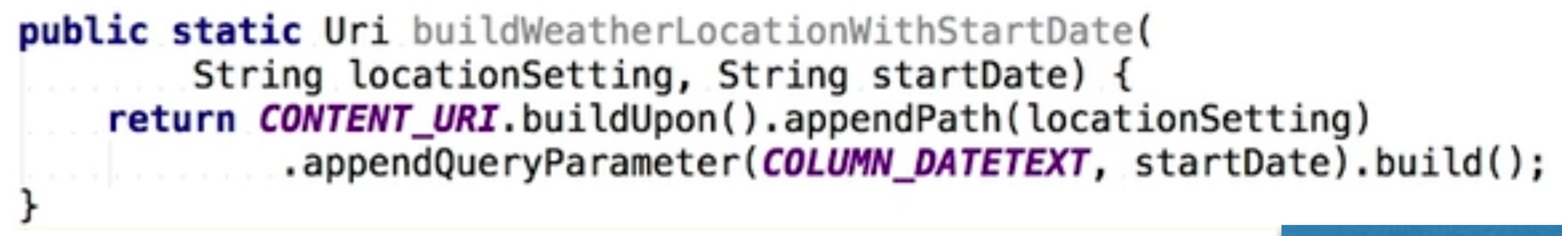
```java
public static Uri buildWeatherUri(long id) {
    return ContentUris.withAppendedId(CONTENT_URI, id);
}
```

Wird ein integer-PrimaryKey übergeben, kann die Methode withAppendedId(…) verwendet werden, um ein einzelnes Item zurückzugeben



```java
public static Uri buildWeatherLocation(String locationSetting) {
    return CONTENT_URI.buildUpon().appendPath(locationSetting).build();
}
```

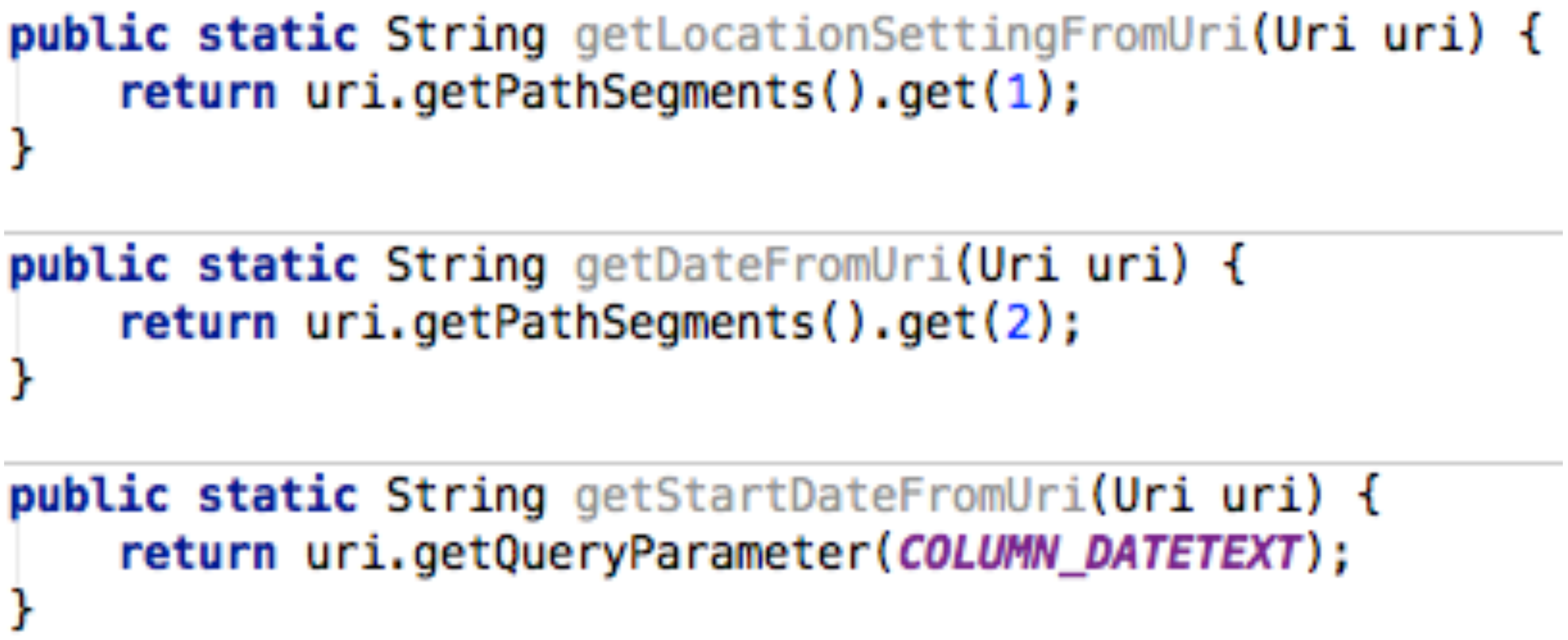
Ansonsten wird appendPath(…) verwendet



```java
public static Uri buildWeatherLocationWithStartDate(
        String locationSetting, String startDate) {
    return CONTENT_URI.buildUpon().appendPath(locationSetting)
            .appendQueryParameter(COLUMN_DATETEXT, startDate).build();
}
```

QueryParameter sind sehr nützlich, wenn man die Parametrisierung der Query einschränken möchte.

Hilfsmethoden



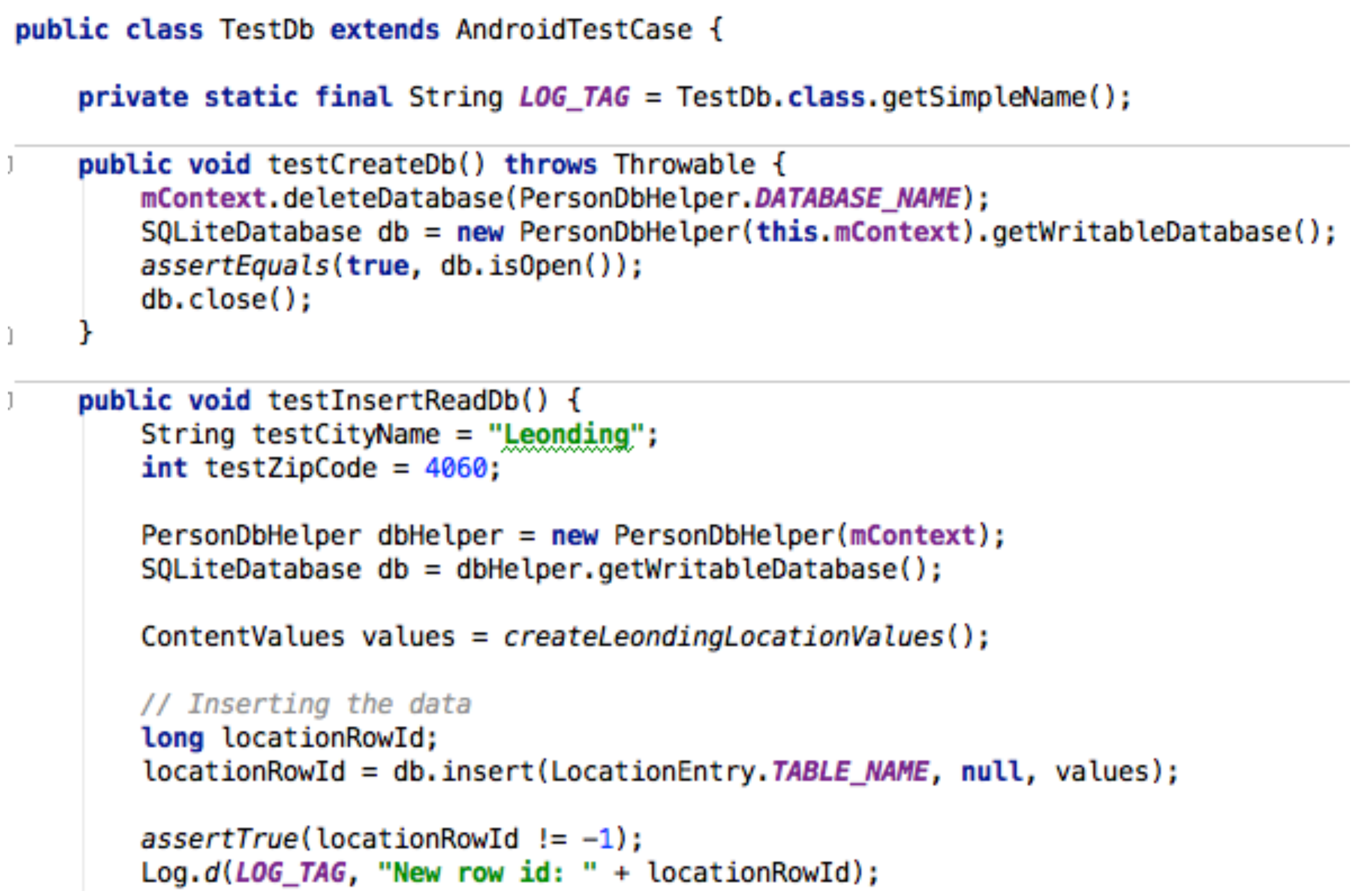
```
public static String getLocationSettingFromUri(Uri uri) {
    return uri.getPathSegments().get(1);
}
```

```
public static String getDateFromUri(Uri uri) {
    return uri.getPathSegments().get(2);
}
```

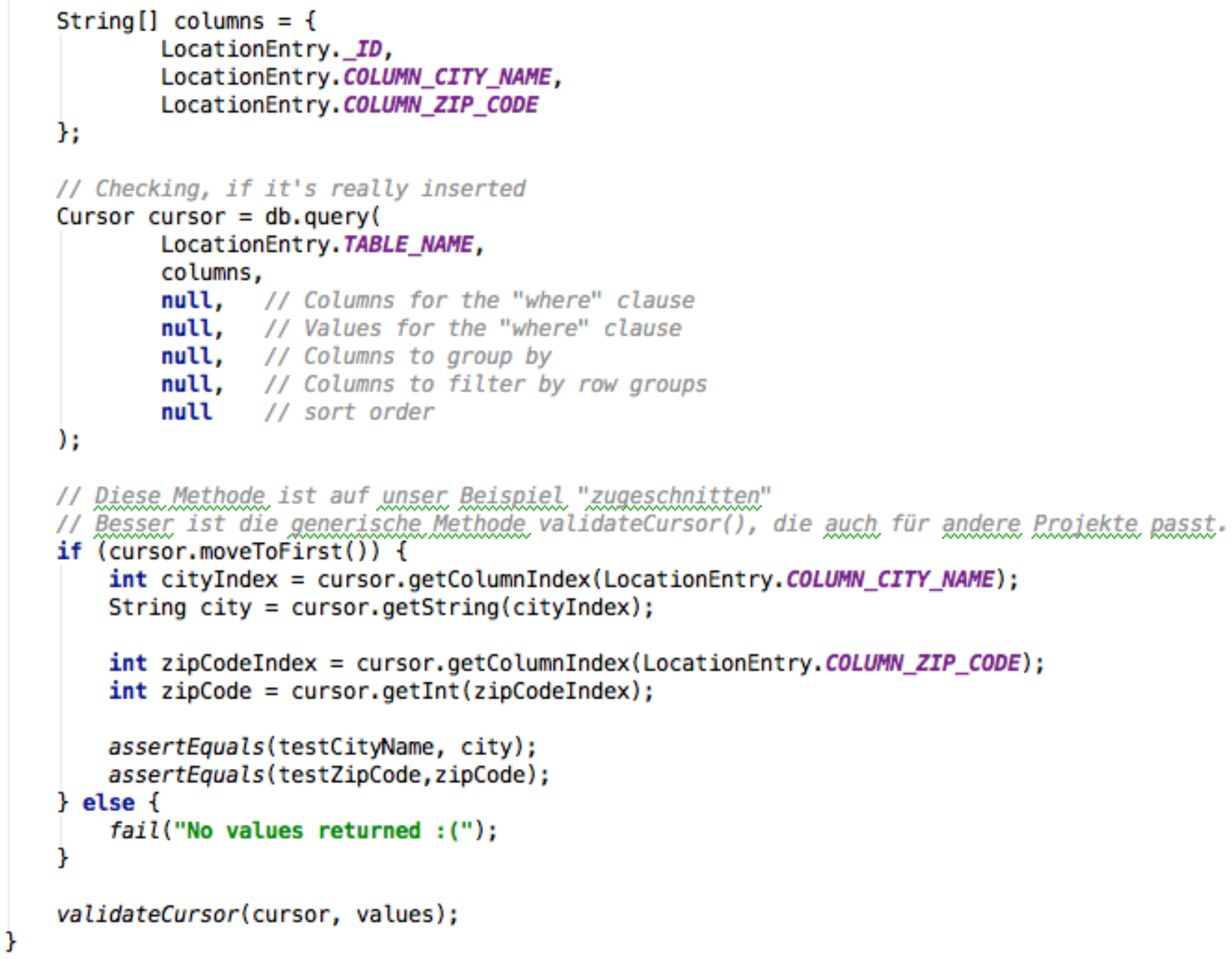
```
public static String getStartDateFromUri(Uri uri) {
    return uri.getQueryParameter(COLUMN_DATETEXT);
}
```

Diesen Methoden dienen ebenfalls zum Verbergen der Implementierung. Nur an einer Stelle ist so das Wissen über die innere Struktur der Datenressource erforderlich.

Unit-Testen der Datenbank

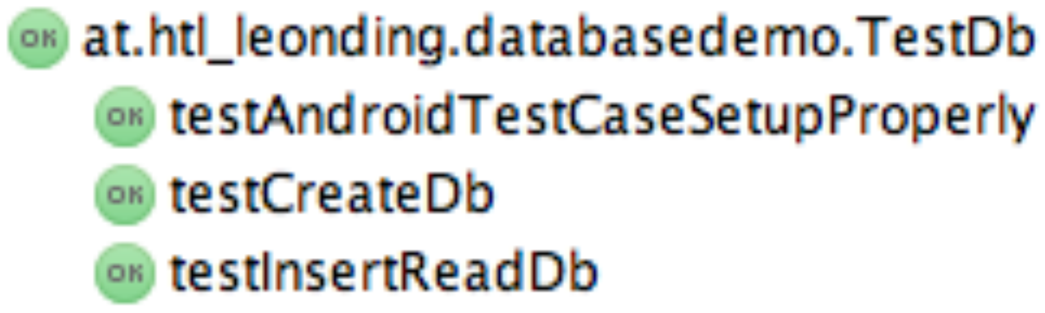


```java
public class TestDb extends AndroidTestCase {

    private static final String LOG_TAG = TestDb.class.getSimpleName();

    public void testCreateDb() throws Throwable {
        mContext.deleteDatabase(PersonDbHelper.DATABASE_NAME);
        SQLiteDatabase db = new PersonDbHelper(this.mContext).getWritableDatabase();
        assertEquals(true, db.isOpen());
        db.close();
    }

    public void testInsertReadDb() {
        String testCityName = "Leonding";
        int testZipCode = 4060;

        PersonDbHelper dbHelper = new PersonDbHelper(mContext);
        SQLiteDatabase db = dbHelper.getWritableDatabase();

        ContentValues values = createLeondingLocationValues();

        // Inserting the data
        long locationRowId;
        locationRowId = db.insert(LocationEntry.TABLE_NAME, null, values);

        assertTrue(locationRowId != -1);
        Log.d(LOG_TAG, "New row id: " + locationRowId);
```



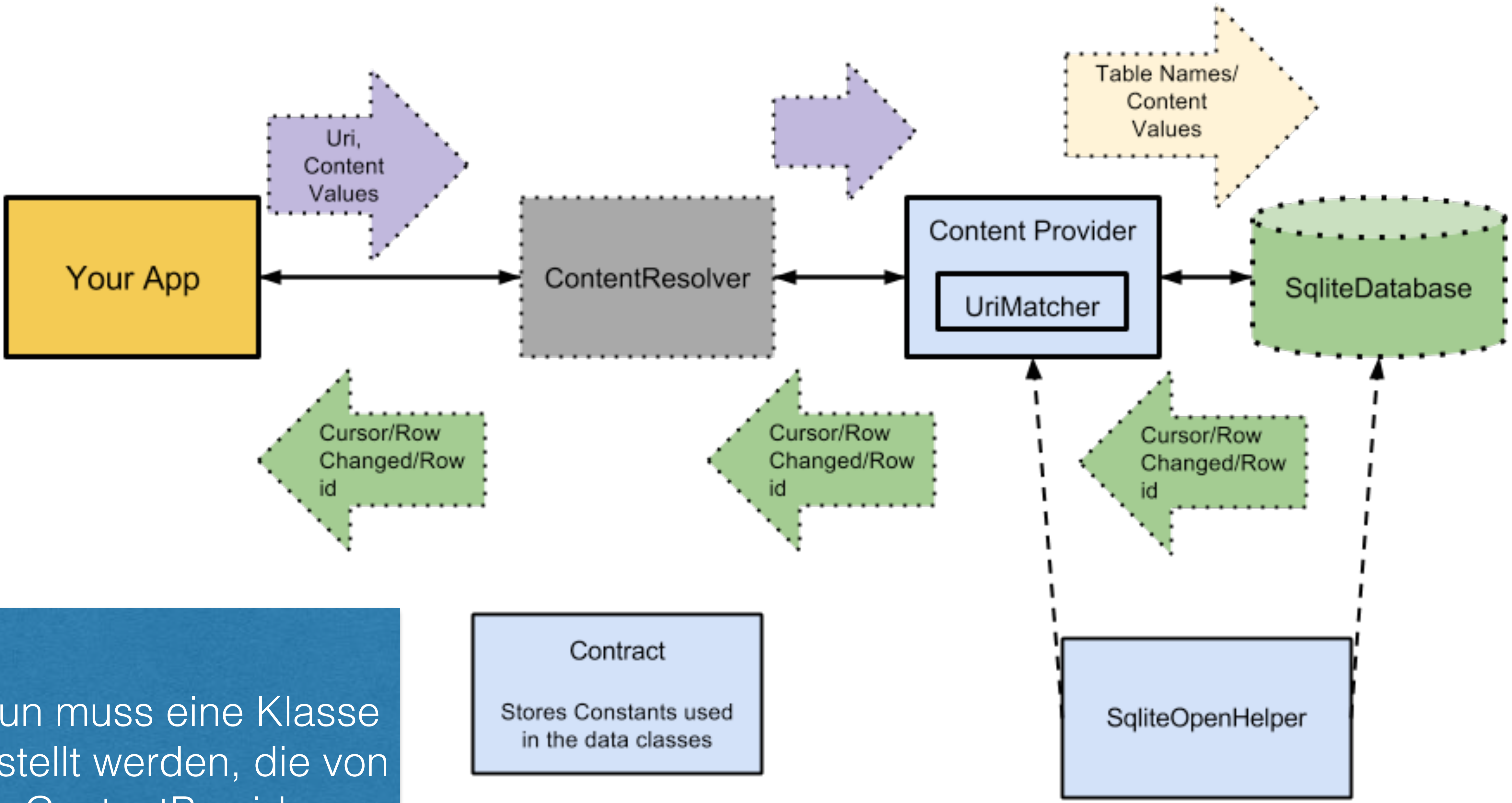
```java
    String[] columns = {
            LocationEntry._ID,
            LocationEntry.COLUMN_CITY_NAME,
            LocationEntry.COLUMN_ZIP_CODE
    };

    // Checking, if it's really inserted
    Cursor cursor = db.query(
            LocationEntry.TABLE_NAME,
            columns,
            null,   // Columns for the "where" clause
            null,   // Values for the "where" clause
            null,   // Columns to group by
            null,   // Columns to filter by row groups
            null    // sort order
    );

    // Diese Methode ist auf unser Beispiel "zugeschnitten"
    // Besser ist die generische Methode validateCursor(), die auch für andere Projekte passt.
    if (cursor.moveToFirst()) {
        int cityIndex = cursor.getColumnIndex(LocationEntry.COLUMN_CITY_NAME);
        String city = cursor.getString(cityIndex);

        int zipCodeIndex = cursor.getColumnIndex(LocationEntry.COLUMN_ZIP_CODE);
        int zipCode = cursor.getInt(zipCodeIndex);

        assertEquals(testCityName, city);
        assertEquals(testZipCode,zipCode);
    } else {
        fail("No values returned :(");
    }

    validateCursor(cursor, values);
}
```

```java
static void validateCursor(Cursor valueCursor, ContentValues expectedValues) {

    assertTrue(valueCursor.moveToFirst());

    Set<Map.Entry<String, Object>> valueSet = expectedValues.valueSet();
    for (Map.Entry<String, Object> entry : valueSet) {
        String columnName = entry.getKey();
        int idx = valueCursor.getColumnIndex(columnName);
        assertFalse(idx == -1);
        String expectedValue = entry.getValue().toString();
        assertEquals(expectedValue, valueCursor.getString(idx));
    }
}
```



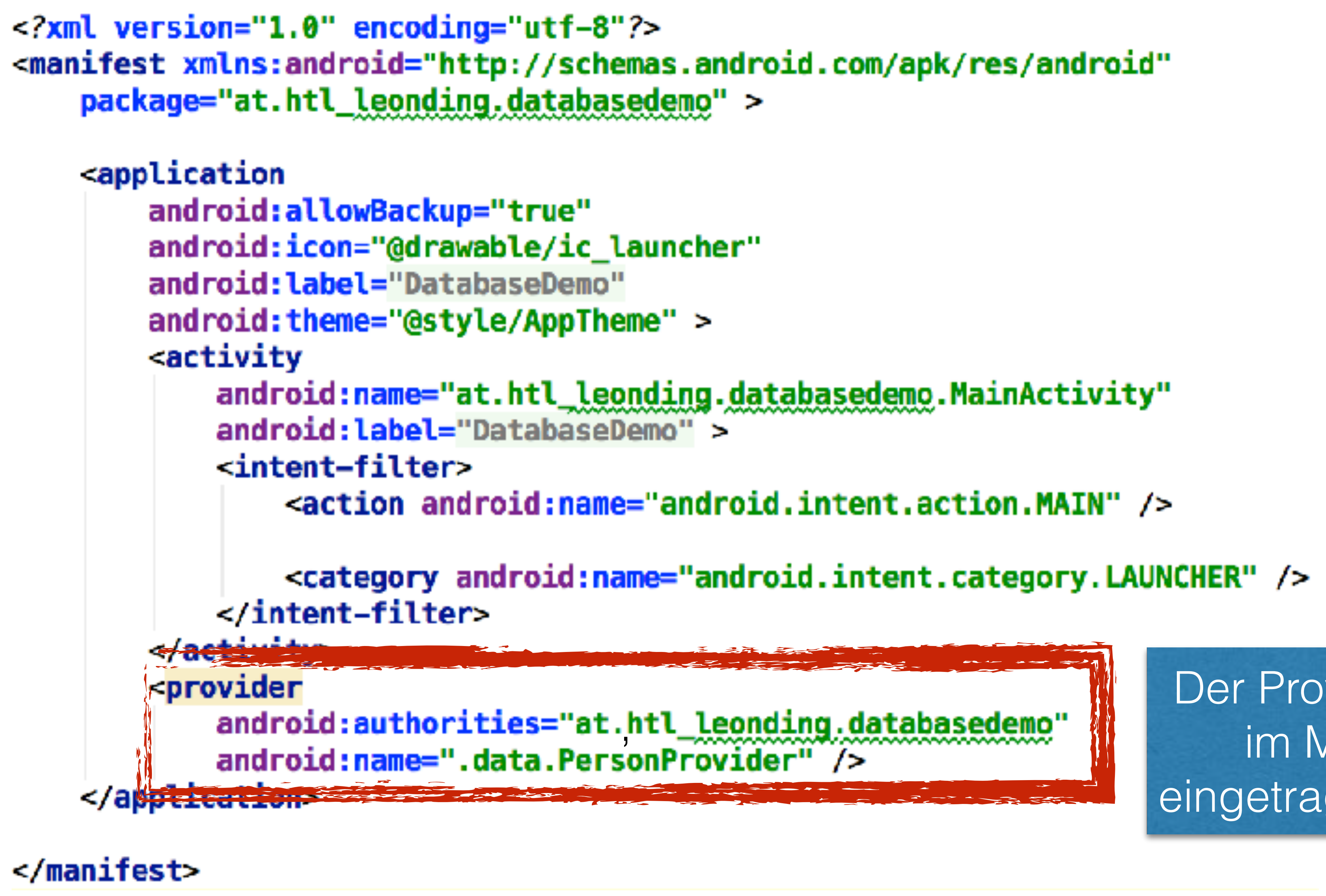
```
OK at.htl_leonding.databasedemo.TestDb
    OK testAndroidTestCaseSetupProperly
    OK testCreateDb
    OK testInsertReadDb
```

Big Picture



Nun muss eine Klasse erstellt werden, die von ContentProvider abgeleitet ist

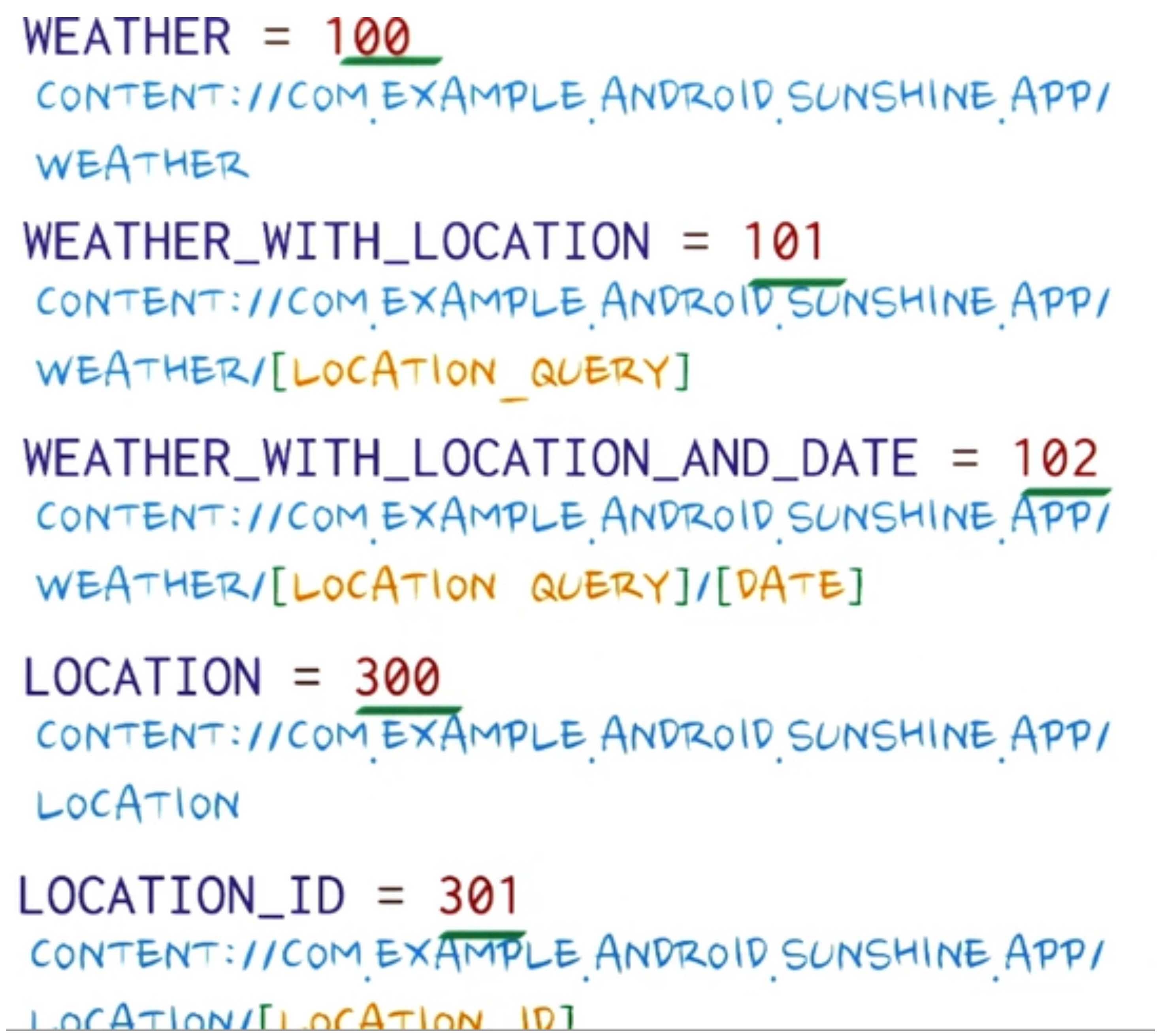
AndroidManifest.xml



```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="at.htl_leonding.databasedemo" >

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="DatabaseDemo"
        android:theme="@style/AppTheme" >
        <activity
            android:name="at.htl_leonding.databasedemo.MainActivity"
            android:label="DatabaseDemo" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <provider
            android:authorities="at.htl_leonding.databasedemo"
            android:name=".data.PersonProvider" />
    </application>

</manifest>
```

Der Provider muss im Manifest eingetragen werden

Uri-Types



WEATHER = 100

CONTENT://COM.EXAMPLE.ANDROID.SUNSHINE.APP/WEATHER

WEATHER_WITH_LOCATION = 101

CONTENT://COM.EXAMPLE.ANDROID.SUNSHINE.APP/WEATHER/[LOCATION_QUERY]

WEATHER_WITH_LOCATION_AND_DATE = 102

CONTENT://COM.EXAMPLE.ANDROID.SUNSHINE.APP/WEATHER/[LOCATION QUERY]/[DATE]

LOCATION = 300

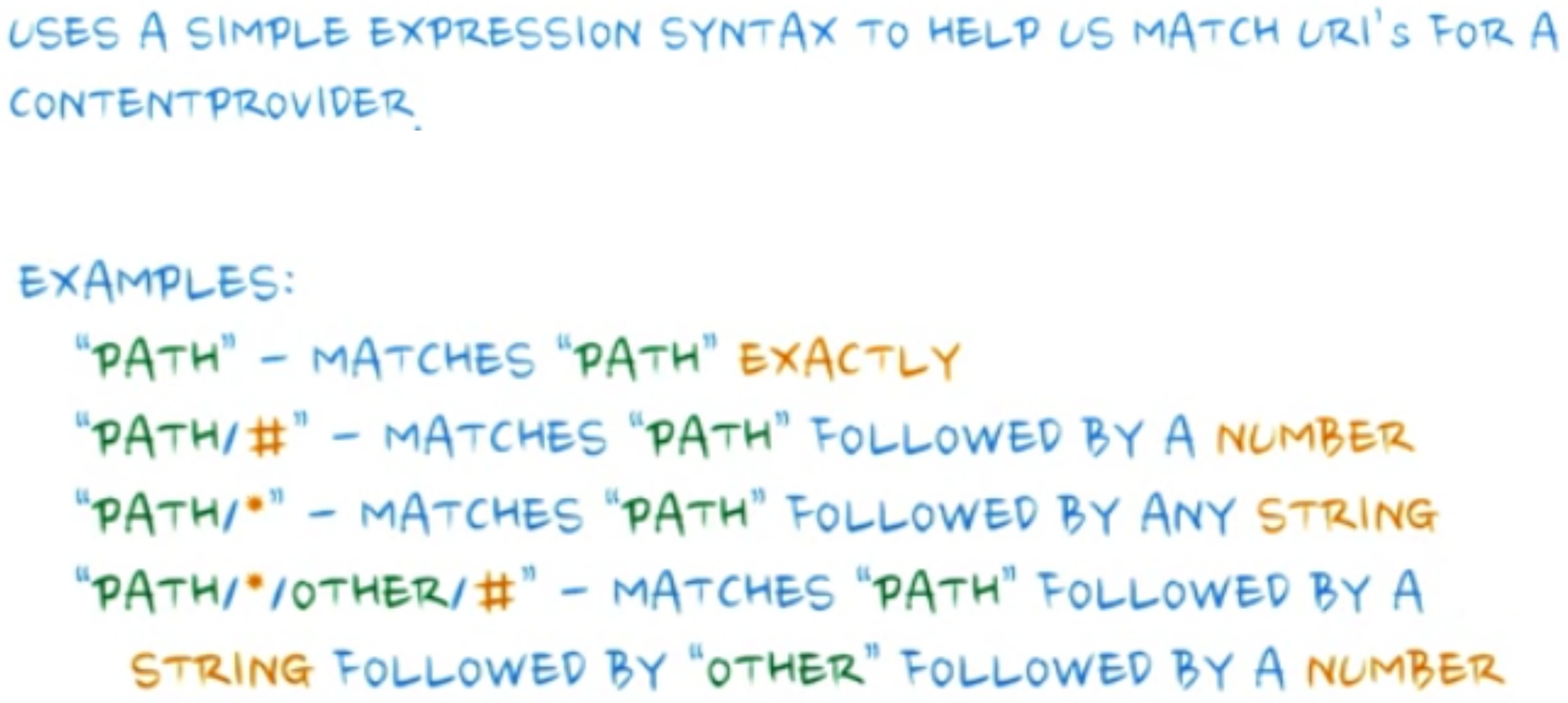
CONTENT://COM.EXAMPLE.ANDROID.SUNSHINE.APP/LOCATION

LOCATION_ID = 301

CONTENT://COM.EXAMPLE.ANDROID.SUNSHINE.APP/LOCATION/[LOCATION_ID]

Die Funktionalität eines ContentProviders basiert auf den unterschiedlichen Uri-Typen. Jedem Uri-Type wird eine Konstante mit einem Integer-Wert zugewiesen

Uri-Matcher



USES A SIMPLE EXPRESSION SYNTAX TO HELP US MATCH URI'S FOR A CONTENTPROVIDER.

EXAMPLES:

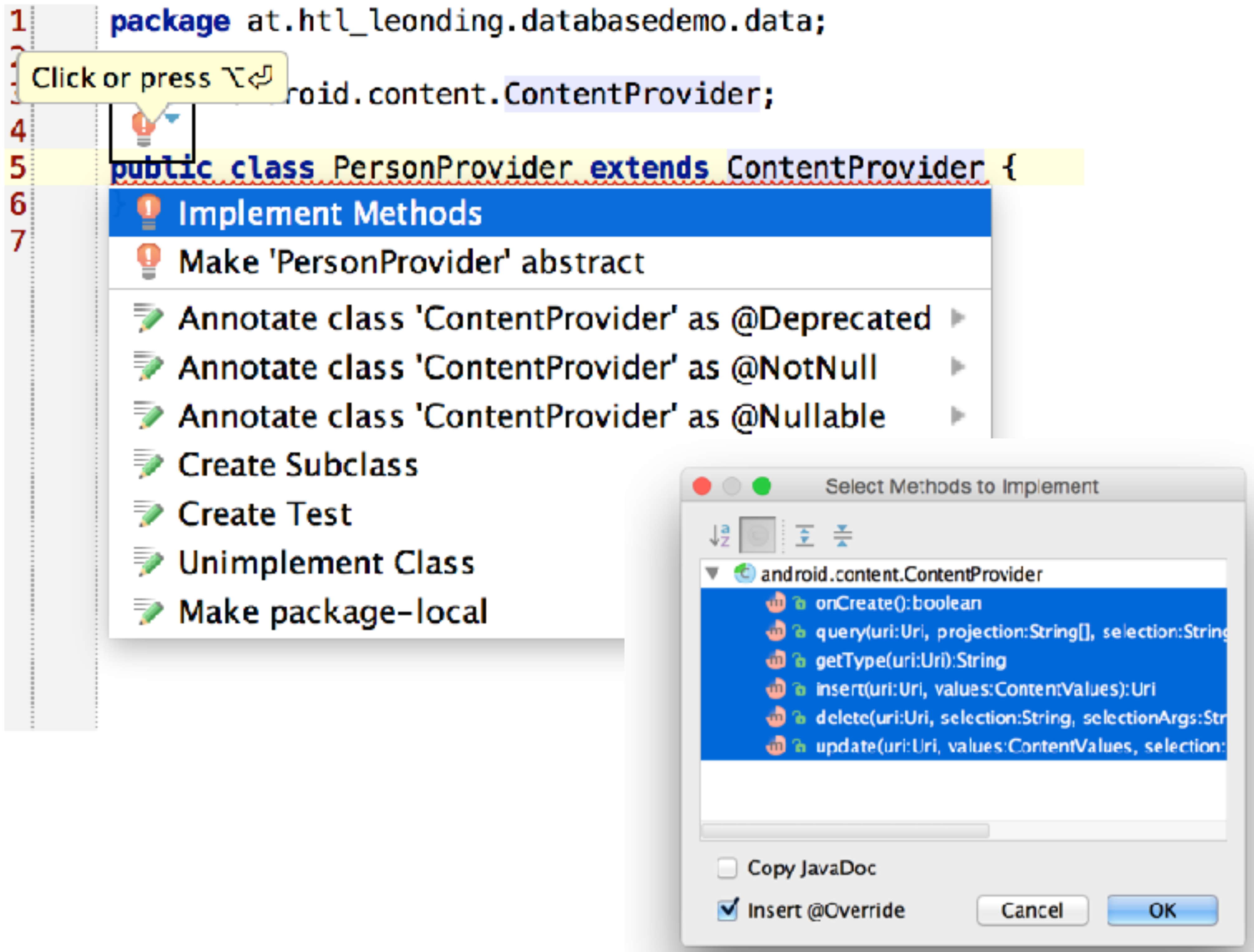
"PATH" – MATCHES "PATH" EXACTLY

"PATH/#" – MATCHES "PATH" FOLLOWED BY A NUMBER

"PATH/*" – MATCHES "PATH" FOLLOWED BY ANY STRING

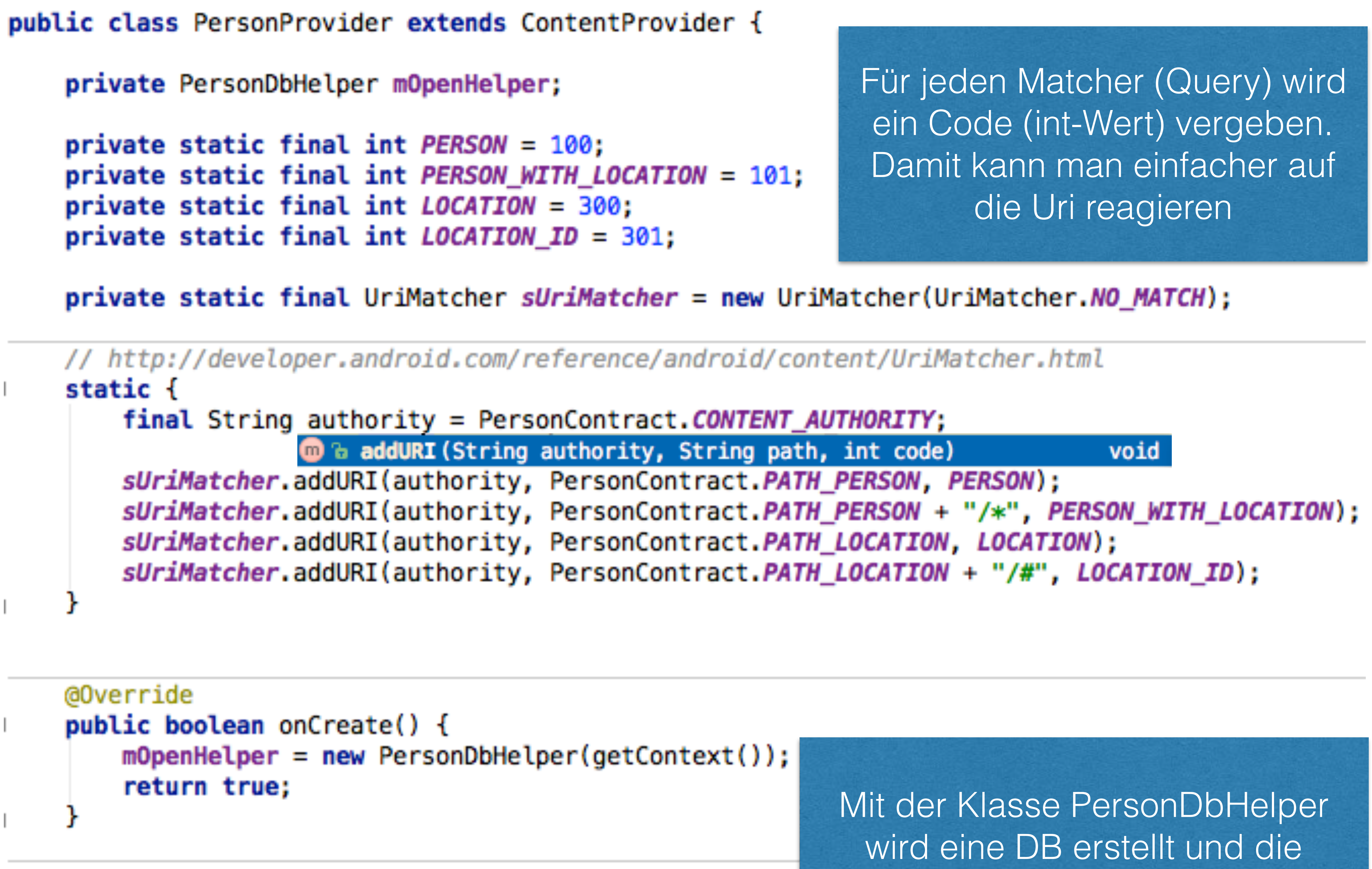
"PATH/*/OTHER/#" – MATCHES "PATH" FOLLOWED BY A STRING FOLLOWED BY "OTHER" FOLLOWED BY A NUMBER

ContentProvider



```
package at.htl_leonding.databasedemo.data;

import android.content.ContentProvider;

public class PersonProvider extends ContentProvider {
```

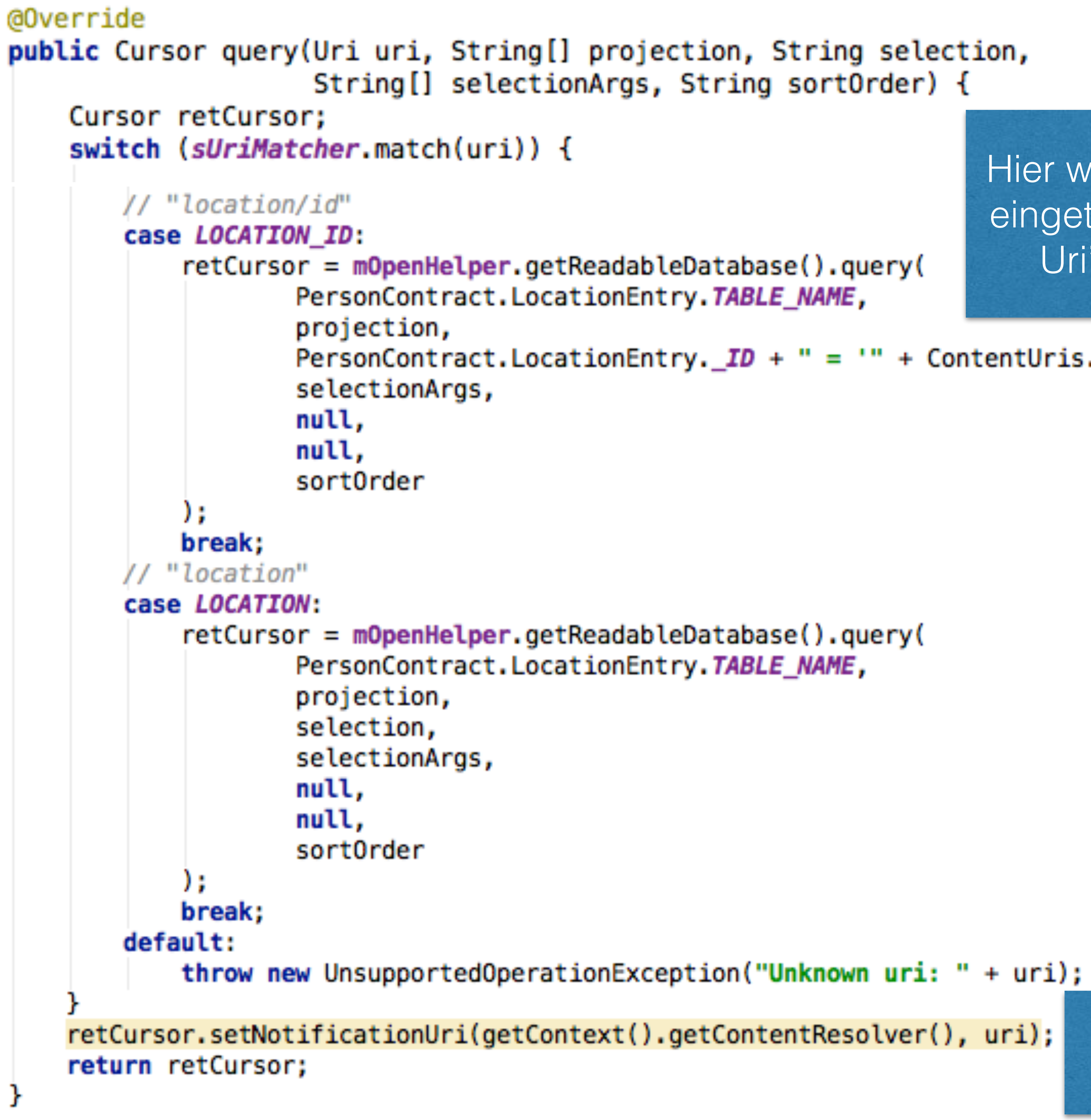
Click or press ⌥⏎
Implement Methods
Make 'PersonProvider' abstract
Annotate class 'ContentProvider' as @Deprecated
Annotate class 'ContentProvider' as @NotNull
Annotate class 'ContentProvider' as @Nullable
Create Subclass
Create Test
Unimplement Class
Make package-local
Select Methods to Implement
android.content.ContentProvider
onCreate():boolean
query(uri:Uri, projection:String[], selection:String
getType(uri:Uri):String
insert(uri:Uri, values:ContentValues):Uri
delete(uri:Uri, selection:String, selectionArgs:Str
update(uri:Uri, values:ContentValues, selection:
Copy JavaDoc
Insert @Override
Cancel
OK



```java
public class PersonProvider extends ContentProvider {

    private PersonDbHelper mOpenHelper;

    private static final int PERSON = 100;
    private static final int PERSON_WITH_LOCATION = 101;
    private static final int LOCATION = 300;
    private static final int LOCATION_ID = 301;

    private static final UriMatcher sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);

    // http://developer.android.com/reference/android/content/UriMatcher.html
    static {
        final String authority = PersonContract.CONTENT_AUTHORITY;
                          addURI (String authority, String path, int code)         void
        sUriMatcher.addURI(authority, PersonContract.PATH_PERSON, PERSON);
        sUriMatcher.addURI(authority, PersonContract.PATH_PERSON + "/*", PERSON_WITH_LOCATION);
        sUriMatcher.addURI(authority, PersonContract.PATH_LOCATION, LOCATION);
        sUriMatcher.addURI(authority, PersonContract.PATH_LOCATION + "/#", LOCATION_ID);
    }


    @Override
    public boolean onCreate() {
        mOpenHelper = new PersonDbHelper(getContext());
        return true;
    }
```

Für jeden Matcher (Query) wird ein Code (int-Wert) vergeben. Damit kann man einfacher auf die Uri reagieren

Mit der Klasse PersonDbHelper wird eine DB erstellt und die Tabellen werden angelegt

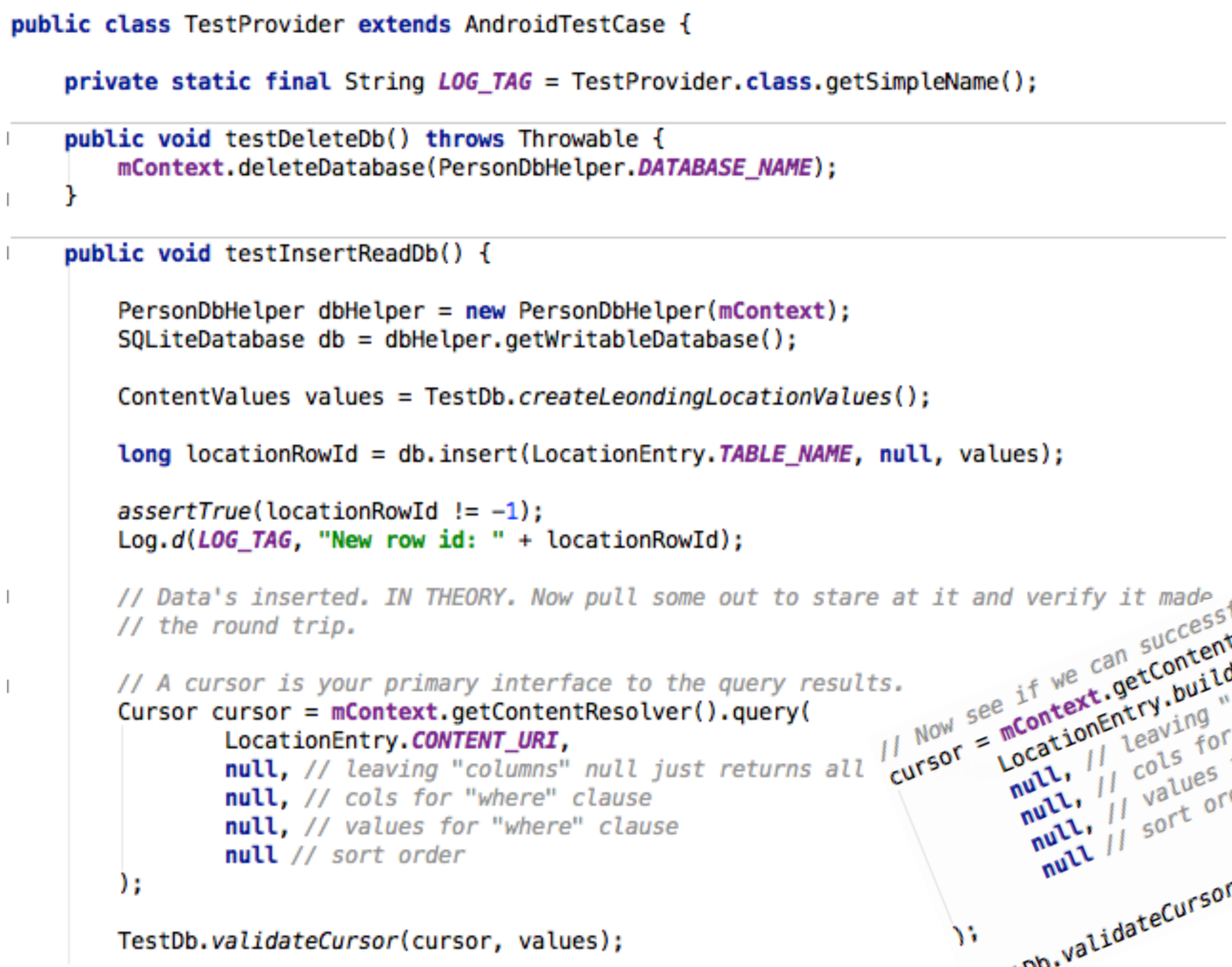


```java
@Override
public Cursor query(Uri uri, String[] projection, String selection,
                    String[] selectionArgs, String sortOrder) {
    Cursor retCursor;
    switch (sUriMatcher.match(uri)) {

        // "location/id"
        case LOCATION_ID:
            retCursor = mOpenHelper.getReadableDatabase().query(
                    PersonContract.LocationEntry.TABLE_NAME,
                    projection,
                    PersonContract.LocationEntry._ID + " = '" + ContentUris.parseId(uri) + "'",
                    selectionArgs,
                    null,
                    null,
                    sortOrder
            );
            break;
        // "location"
        case LOCATION:
            retCursor = mOpenHelper.getReadableDatabase().query(
                    PersonContract.LocationEntry.TABLE_NAME,
                    projection,
                    selection,
                    selectionArgs,
                    null,
                    null,
                    sortOrder
            );
            break;
        default:
            throw new UnsupportedOperationException("Unknown uri: " + uri);
    }
    retCursor.setNotificationUri(getContext().getContentResolver(), uri);
    return retCursor;
}
```

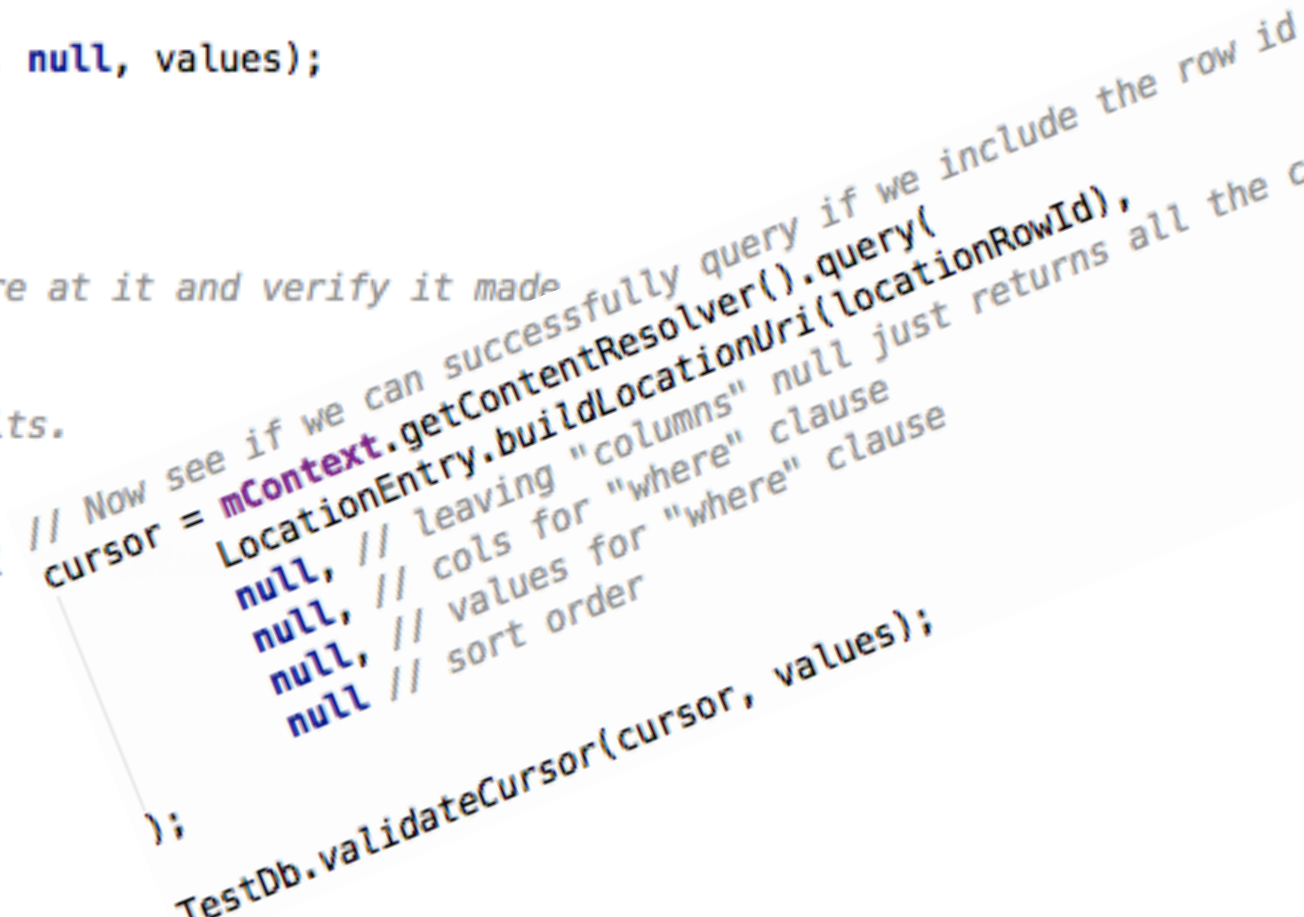
Hier werden sämtliche Uri's eingetragen. Die PERSON-Uri's fehlen hier noch

Observer für Datenänderungen

Unit-Test für ContentProvider

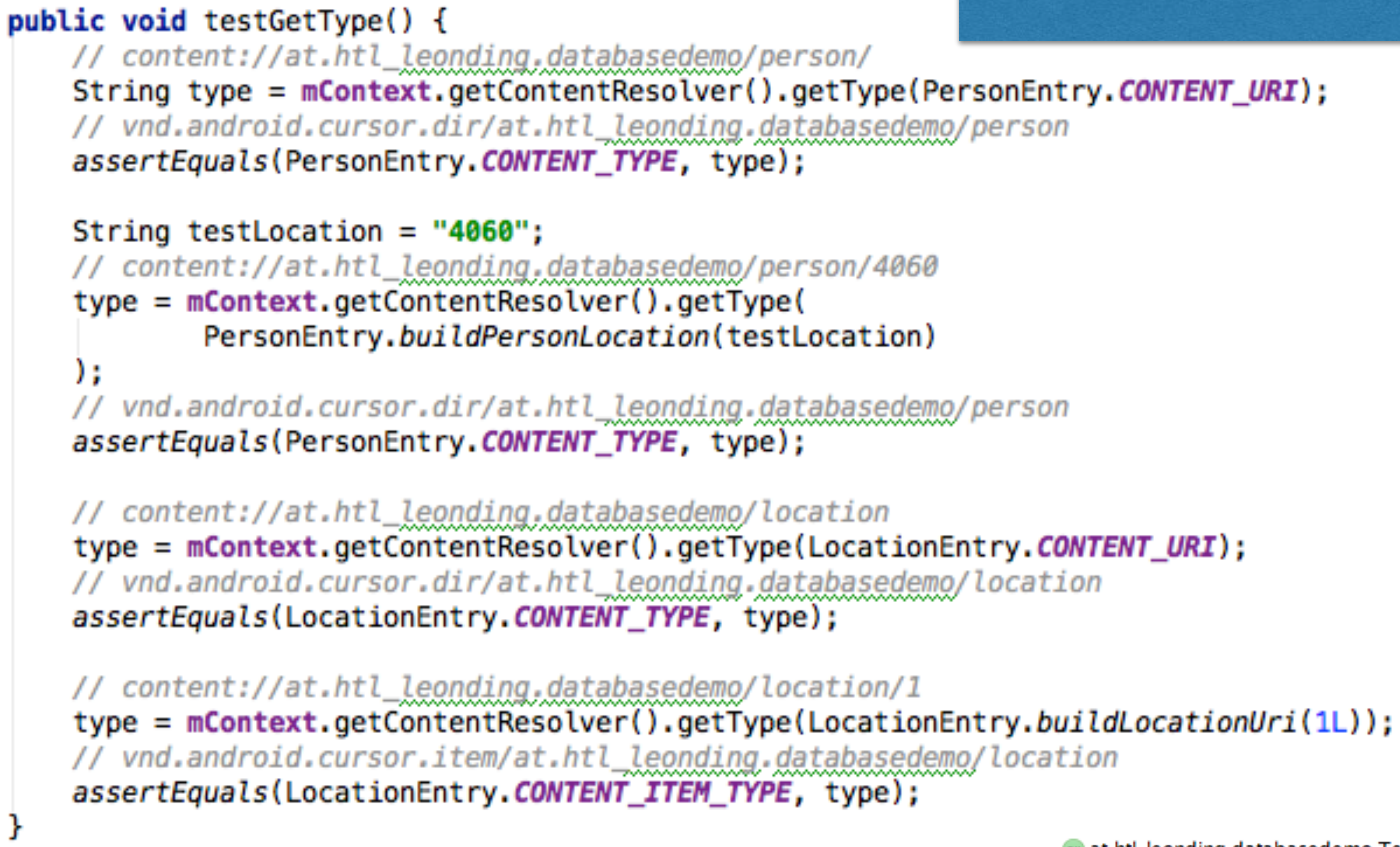


```
public class TestProvider extends AndroidTestCase {

    private static final String LOG_TAG = TestProvider.class.getSimpleName();

    public void testDeleteDb() throws Throwable {
        mContext.deleteDatabase(PersonDbHelper.DATABASE_NAME);
    }

    public void testInsertReadDb() {

        PersonDbHelper dbHelper = new PersonDbHelper(mContext);
        SQLiteDatabase db = dbHelper.getWritableDatabase();

        ContentValues values = TestDb.createLeondingLocationValues();

        long locationRowId = db.insert(LocationEntry.TABLE_NAME, null, values);

        assertTrue(locationRowId != -1);
        Log.d(LOG_TAG, "New row id: " + locationRowId);

        // Data's inserted. IN THEORY. Now pull some out to stare at it and verify it made
        // the round trip.

        // A cursor is your primary interface to the query results.
        Cursor cursor = mContext.getContentResolver().query(
                LocationEntry.CONTENT_URI,
                null, // leaving "columns" null just returns all
                null, // cols for "where" clause
                null, // values for "where" clause
                null // sort order
        );

        TestDb.validateCursor(cursor, values);
```

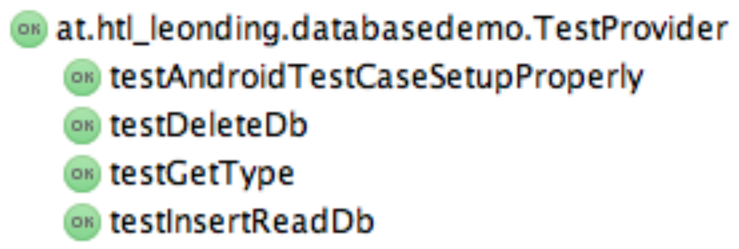


```
// Now see if we can successfully query if we include the row id
cursor = mContext.getContentResolver().query(
        LocationEntry.buildLocationUri(locationRowId),
        null, // leaving "columns" null just returns all the c
        null, // cols for "where" clause
        null, // values for "where" clause
        null // sort order
);

TestDb.validateCursor(cursor, values);
```

Hier wird der Rückgabetyp (DIR oder ITEM) geprüft

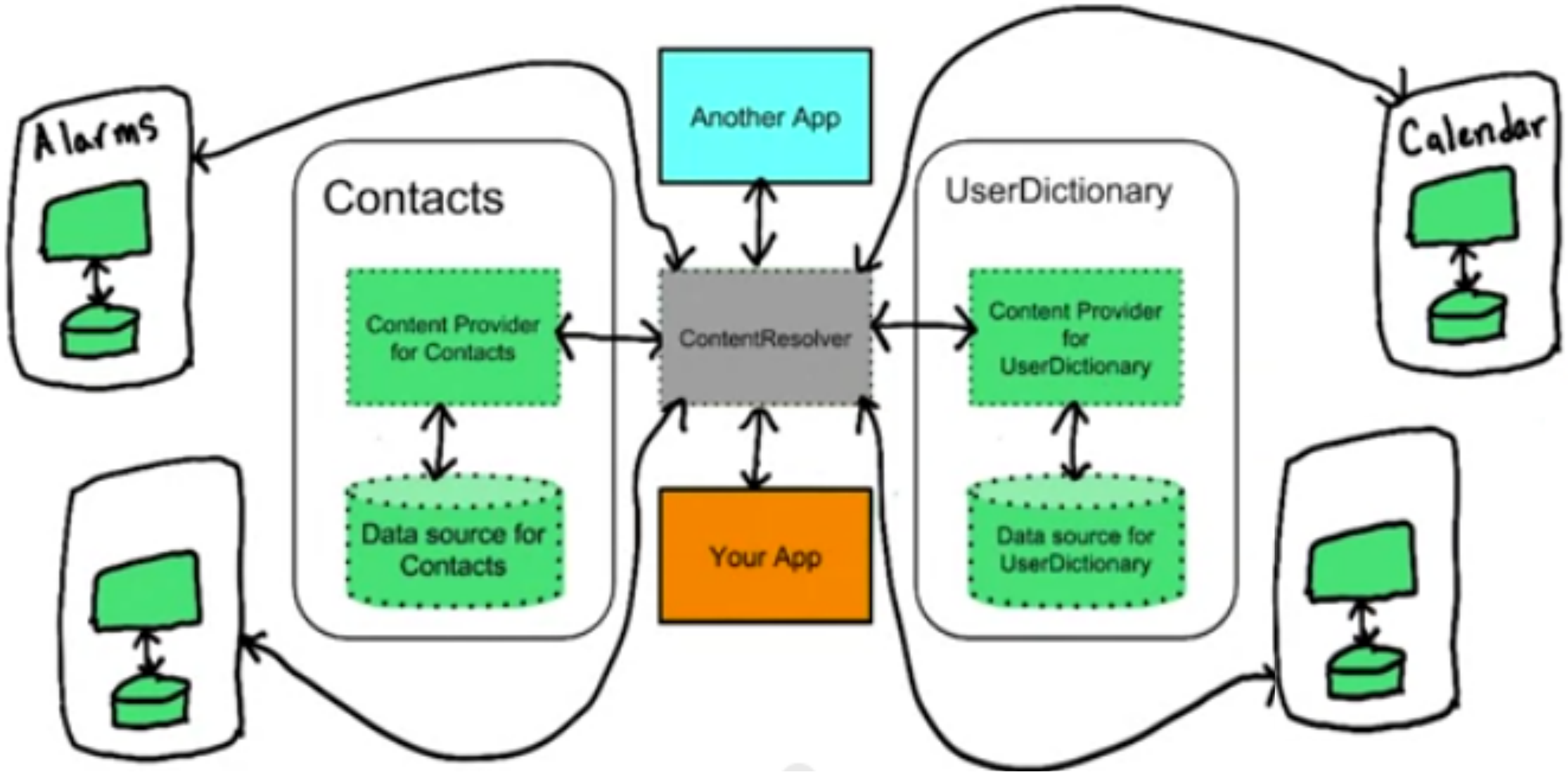


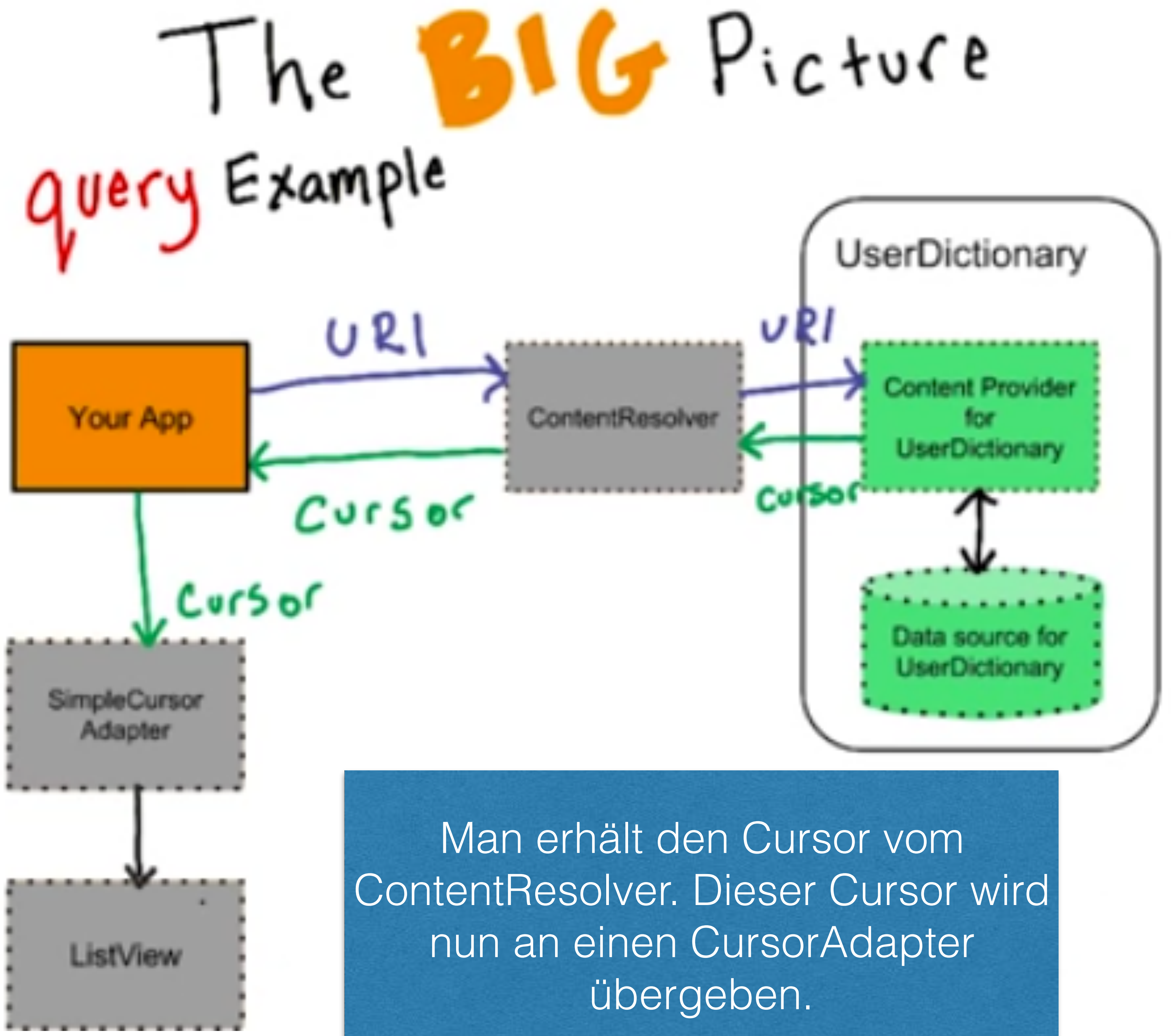
```java
public void testGetType() {
    // content://at.htl_leonding.databasedemo/person/
    String type = mContext.getContentResolver().getType(PersonEntry.CONTENT_URI);
    // vnd.android.cursor.dir/at.htl_leonding.databasedemo/person
    assertEquals(PersonEntry.CONTENT_TYPE, type);

    String testLocation = "4060";
    // content://at.htl_leonding.databasedemo/person/4060
    type = mContext.getContentResolver().getType(
            PersonEntry.buildPersonLocation(testLocation)
    );
    // vnd.android.cursor.dir/at.htl_leonding.databasedemo/person
    assertEquals(PersonEntry.CONTENT_TYPE, type);

    // content://at.htl_leonding.databasedemo/location
    type = mContext.getContentResolver().getType(LocationEntry.CONTENT_URI);
    // vnd.android.cursor.dir/at.htl_leonding.databasedemo/location
    assertEquals(LocationEntry.CONTENT_TYPE, type);

    // content://at.htl_leonding.databasedemo/location/1
    type = mContext.getContentResolver().getType(LocationEntry.buildLocationUri(1L));
    // vnd.android.cursor.item/at.htl_leonding.databasedemo/location
    assertEquals(LocationEntry.CONTENT_ITEM_TYPE, type);
}
```



```
OK at.htl_leonding.databasedemo.TestProvider
    OK testAndroidTestCaseSetupProperly
    OK testDeleteDb
    OK testGetType
    OK testInsertReadDb
```

ContentResolver


Alarms
Contacts
Content Provider for Contacts
Data source for Contacts
Another App
ContentResolver
Your App
UserDictionary
Content Provider for UserDictionary
Data source for UserDictionary
Calendar



Man erhält den Cursor vom ContentResolver. Dieser Cursor wird nun an einen CursorAdapter übergeben.

https://www.udacity.com/course/viewer#!/c-ud258/l-3372188753/m-3381368821

What is the Content Resolver?

The Content Resolver is the single, global instance in your application that provides access to your (and other applications') content providers. The Content Resolver behaves exactly as its name implies: it accepts requests from clients, and resolves these requests by directing them to the content provider with a distinct authority. To do this, the Content Resolver stores a mapping from authorities to Content Providers. This design is important, as it allows a simple and secure means of accessing other applications' Content Providers.

The Content Resolver includes the CRUD (create, read, update, delete) methods corresponding to the abstract methods (insert, delete, query, update) in the Content Provider class. The Content Resolver does not know the implementation of the Content Providers it is interacting with (nor does it need to know); each method is passed an URI that specifies the Content Provider to interact with.

What is the Content Provider?

Whereas the Content Resolver provides an abstraction from the application's Content Providers, Content Providers provides an abstraction from the underlying data source (i.e. a `SQLite database`). They provide mechanisms for defining data security (i.e. by enforcing read/write permissions) and offer a standard interface that connects data in one process with code running in another process.

Content Providers provide an interface for publishing and consuming data, based around a simple URI addressing model using the `content:// schema`. They enable you to decouple your application layers from the underlying data layers, making your application data-source agnostic by abstracting the underlying data source.

Source: http://www.androiddesignpatterns.com/2012/06/content-resolvers-and-content-providers.html

http://stackoverflow.com/questions/17567326/how-does-getcontentresolver-work