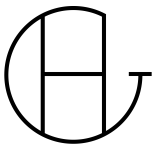


HoGent

Faculteit Bedrijf en Organisatie

Performantie van persistentiemogelijkheden in Android

Özgür Akin

Scriptie voorgedragen tot het bekomen van de graad van
Bachelor in de toegepaste informatica

Promotor:
Jens Buysse
Co-promotor:
Simon Raes

Academiejaar: 2015-2016

Tweede examenperiode

Faculteit Bedrijf en Organisatie

Performantie van persistentiemogelijkheden in Android

Özgür Akin

Scriptie voorgedragen tot het bekomen van de graad van
Bachelor in de toegepaste informatica

Promotor:
Jens Buysse
Co-promotor:
Simon Raes

Academiejaar: 2015-2016

Tweede examenperiode

Samenvatting

Vandaag de dag bestaan er veel applicaties, maar hoeveel daarvan blijven werken zonder internetverbinding? Tegenwoordig is het ondersteunen van offline werking in een applicatie geen luxe meer, maar een must-have. Om offline-support te voorzien binnen een applicatie, is er nood aan het gebruik van een database. Hierdoor zijn databases belangrijk binnen de IT-sector.

Er bestaan verschillende soorten databases, maar welke moet men gebruiken? Welke is het meest geschikt bij een bepaalde soort applicatie? De keuze van de database kan een grote invloed hebben op verschillende eigenschappen: performantie, opstartsnelheid, CPU-gebruik,.. Als de database deze eigenschappen op een negatieve manier beïnvloedt, kan dit tot gevolg hebben dat het aantal gebruikers van de mobiele applicatie zal verminderen. Ter beantwoording van de probleemstelling zijn volgende deelvragen geformuleerd met betrekking op de applicatie:

1. Wat is de invloed van de gekozen database op de opstartsnelheid? Vertraagt het gebruik van de gekozen database de opstartsnelheid van de applicatie, of heeft het helemaal geen invloed (in vergelijking met gebruik van andere databases)?
2. Wat is de invloed van de gekozen database op het CPU-gebruik? Een hoger CPU-gebruik zal zorgen voor meer batterijverbruik. Zal de applicatie bij gebruik van de gekozen database meer of juist minder CPU gebruiken (in vergelijking met gebruik van andere databases)?
3. Wat is de gemiddelde snelheid van de gekozen database bij het toevoegen van records aan de database?

Het onderzoek werd uitgevoerd op drie verschillende applicatieprofielen: weinig data (profiel 1), gemiddelde hoeveelheid data (profiel 2), veel data (profiel 3).

De verwachtingen waren dat Realm altijd de beste keuze zou zijn, behalve bij applicatieprofiel 1. Daar zou SharedPreferences de beste keuze moeten zijn, aangezien het speciaal ontwikkeld is voor kleine hoeveelheden simpele data. Het onderzoek heeft echter volgend resultaat opgeleverd:

- Profiel 1 - Weinig data : Realm
- Profiel 2 - Gemiddelde hoeveelheid data : Realm
- Profiel 3 - Veel data : SQLite

De details van het onderzoek zijn te vinden in het volgende deel van dit scriptie.

Voorwoord

Deze scriptie is geschreven ter afsluiting van de opleiding Toegepaste Informatica van de Hogeschool Gent.

Voor mijn scriptie heb ik van mijn docent Native Apps, Jens Buysse, het voorstel gekregen om de verschillende persistentiemogelijkheden binnen Android te vergelijken. Ik heb eerst wat research gedaan, om te zien of er reeds informatie beschikbaar was over dit onderwerp. Op het eerste zicht was er niet veel informatie over terug te vinden, maar ik wou mijn onderzoek toch over dit onderwerp doen. Ik heb dan besloten om drie soorten persistentiemogelijkheden te onderzoeken: SharedPreferences, GreenDAO en SQLite.

Ik heb dit onderwerp met mijn co-promotor besproken en hij heeft mij voorgesteld om er een vierde, redelijk recent uitgekomen persistentiemogelijkheid aan toe te voegen: Realm. Na de vastlegging van het definitieve onderwerp ben ik begonnen met het onderzoek, en hier ligt het resultaat.

Graag wil ik mijn promotor, Jens Buysse, bedanken voor de begeleiding tijdens het onderzoek, de snelle feedback en het voorstel om de persistentiemogelijkheden binnen Android te vergelijken. Tevens wil ik mijn co-promotor, Simon Raes (Android Developer bij In The Pocket), bedanken voor het voorstel om Realm toe te voegen aan het onderzoek als vierde persistentiemogelijkheid.

Inhoudsopgave

Hoofdstuk 1

Inleiding

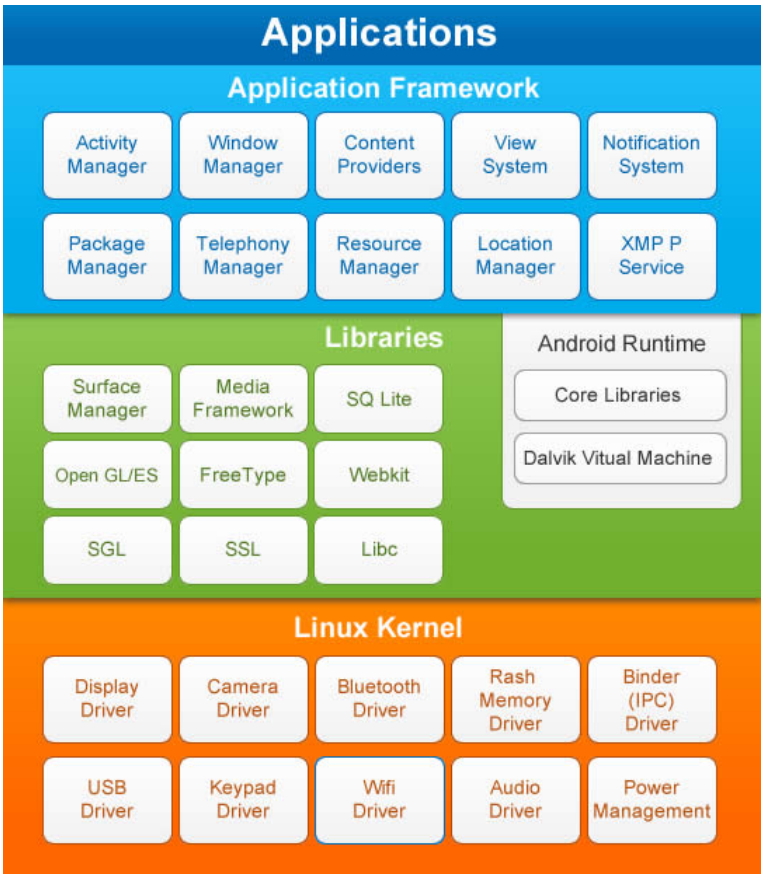
"Developers . . . cannot make assumptions about the constancy, quality, or even existence of an individual's network connection. Therefore offline support will be a crucial consideration for nearly every future modern application. Unfortunately, our experience shows that offline support is the mobile app feature continually underscoped by developers and over-simplified by stakeholders."(Michael Facemire, 2014)

Vandaag de dag bestaan er veel applicaties, maar hoeveel daarvan blijven werken zonder internetverbinding? Tegenwoordig is het ondersteunen van offline werking in een applicatie geen luxe meer, maar een must-have (Michael Facemire, 2014).

Laten we een voorbeeld bekijken: Persoon A heeft thuis bekeken wat hij/zij nog thuis heeft liggen, en heeft op basis daarvan een boodschappenlijst opgesteld met een applicatie B. Hierna gaat persoon A op stap naar de winkel en begint aan zijn/haar boodschappen. Tijdens de boodschappen heeft persoon A geen internetverbinding. Hij/Zij koopt een paar artikelen die op de boodschappenlijst van applicatie B staan, en vinkt deze zorgvuldig af. Na de boodschappen komt persoon A naar huis, de smartphone van deze persoon maakt verbinding met de Wifi thuis, en wat gebeurt er? Alle acties de persoon A heeft uitgevoerd op applicatie B zijn niet opgeslaan, omdat deze applicatie geen offline-support heeft. Hierdoor weet deze persoon niet meer wat hij/zij reeds gekocht heeft. Met als gevolg dat persoon A hoogstwaarschijnlijk zal op zoek gaan naar een applicatie die wel offline-support heeft.

Om offline-support te voorzien binnen een applicatie, is er nood aan het gebruik van een database. Hierdoor zijn databases belangrijk binnen de IT-sector. Nog belangrijker dan het gebruik van databases, is het gebruik van de juiste databases. Wat heeft een gebruiker aan een applicatie die offline-support heeft, maar wel heel traag werkt? Of wat heeft een gebruiker aan een applicatie die én offline-support heeft én snel werkt, maar enorm veel batterij verbruikt?

Figuur 1.1: Android architectuur (http://www.developer.com)



1.1 Probleemstelling en Onderzoeksvragen

De meeste mobiele applicaties, en dus ook die voor het Android platform, zijn genoodzaakt om minstens kleine hoeveelheden data lokaal op te slaan. Deze applicaties moeten dan bijvoorbeeld de status van de applicatie opslaan of bewaren, om de voortgang van de gebruiker niet te verliezen (AndroidDevelopers, 2016c). Er bestaan ook applicaties waar men niet enkel de status van de applicatie, maar ook andere gegevens moet opslaan. Om deze gegevens op te slaan binnen Android, is er nood aan een database. Meer bepaald om gegevens op te slaan die opnieuw gebruikt zullen worden, bijvoorbeeld contacten.

Zoals te zien is op figuur 1.1 maakt Android gebruik van **SQLite**, een open-source database. Het is een "built-in"component van Android. SQLite is een in-process library die een op zichzelf staand, zero-configuratie, transactionele SQL-database-engine im-

Figuur 1.2: GreenDAO structuur (GeenDAO, 2016)

plementeert zonder server. SQLite is de meest gebruikte database ter wereld met meer applicaties dan we kunnen rekenen, waaronder een aantal high-profile projecten. Een database in SQLite is een single disk file, waarbij het bestandsformaat cross-platform is. Een bestand (file) die op één machine gecreëerd is, kan gekopieerd en gebruikt worden op andere machines die een verschillend architectuur hebben (SQLite, 2001).

SQLite slaat de gegevens dus op in een file. Nadat de database is gecreëerd, wordt deze opgeslaan in een map op het Android toestel onder de naam */data/data/<package_name>/databases*. (Lee, 2012). SQLite ondersteunt drie data types: TEXT (gelijkaardig dan String in Java), INTEGER (gelijkaardig dan long in Java) en REAL (gelijkaardig dan double in Java). (Lee, 2012). Als men andere data types wil opslaan, moet men ze eerst converteren. Als men bijvoorbeeld data wil opslaan van het type boolean, kan men dit opslaan als de type INTEGER in de vorm: 1 voor "true", 0 voor "false".

Een ander manier om data op te slaan in Android, is het gebruik van **GreenDAO**. GreenDAO is een open source project om Android developers te helpen bij het opslaan van data in SQLite. Het schrijven van code in SQL en het ontleden van query resultaten zijn vervelende taken (GeenDAO, 2016). GreenDAO neemt dat deel van het werk over en zet Java objecten om naar database tabellen (zie figuur 1.2). GreenDAO is een ORM (Object Relational Mapping) tool voor Android. Het biedt een object-georiënteerde interface op de relationele database SQLite. ORM tools zoals GreenDAO doen herhalende taken voor u en bieden een eenvoudige interface voor de data die moet gepersisteerd worden (GeenDAO, 2016).

Sinds 14 september 2014 bestaat er een nieuwe manier om data op te slaan binnen Android: **Realm**. Realm is een mobiele database. Het is een vervanging voor SQLite en Core data. Het is geen ORM en maakt geen gebruik van SQLite. Het gebruikt een eigen manier om data te persisteren, gebaseerd op eenvoud en snelheid (Realm, 2014). Realm is ook cross-platform. Het ondersteunt zowel Android als iOS. Developers kunnen dus Realm bestanden moeiteloos delen. Op de site van Realm wordt beweerd dat Realm volgende eigenschappen bevat (Realm, 2014):

- **Mobile first:** Realm is de eerste database die van begin af aan gebouwd is voor gebruik binnen telefoons, tablets en wearables.
- **Eenvoudig:** Data wordt direct voorgesteld als objecten en is opvraagbaar aan de hand van query's.
- **Modern:** Realm ondersteunt thread-safety, relaties en encryptie.
- **Snel:** Realm is zelfs sneller dan raw SQLite op gemeenschappelijke operaties.

Bovenstaande databases zouden moeten gebruikt worden voor een gemiddelde hoeveelheid data die moet gepersisteerd worden. Als er daarentegen een relatief kleine hoeveelheid simpele data moet gepersisteerd worden, zou er gebruikt moeten gemaakt worden van **SharedPreferences**. Een SharedPreferences object verwijst naar een bestand met key-value paren en biedt eenvoudige methoden aan om deze te lezen en te schrijven (AndroidDevelopers, 2016d).

SQLite en SharedPreferences kunnen direct gebruikt worden binnen Android. Er moeten dus geen speciale dependencies voor toegevoegd worden. Om GreenDAO en Realm te gebruiken, moeten er daarentegen wel dependencies toegevoegd worden, anders kunnen deze niet gebruikt worden. Indien de nodige dependencies niet toegevoegd worden, zullen deze niet herkend worden door Android.

Er bestaan verschillende soorten databases, maar welke moet men gebruiken? Welke is het meest geschikt bij een bepaalde soort applicatie? De keuze van de database kan een grote invloed hebben op verschillende eigenschappen: performantie, opstartsnelheid, CPU-gebruik,.. Als de database deze eigenschappen op een negatieve manier beïnvloedt, kan dit tot gevolg hebben dat het aantal gebruikers van de mobiele applicatie zal verminderen. Ter beantwoording van de probleemstelling zijn volgende deelvragen geformuleerd met betrekking op de applicatie:

1. Wat is de invloed van de gekozen database op de opstartsnelheid? Vertraagt het gebruik van de gekozen database de opstartsnelheid van de applicatie, of heeft het helemaal geen invloed (in vergelijking met gebruik van andere databases)?
2. Wat is de invloed van de gekozen database op het CPU-gebruik? Een hoger CPU-gebruik zal zorgen voor meer batterijverbruik. Zal de applicatie bij gebruik van de gekozen database meer of juist minder CPU gebruiken (in vergelijking met gebruik van andere databases)?
3. Wat is de gemiddelde snelheid van de gekozen database bij het toevoegen van records aan de database?

Deze deelvragen worden achtereenvolgens beantwoord in hoofdstuk drie.

Hoofdstuk 2

Methodologie

2.1 Keuze van databases

Om te beginnen is er een onderzoek gedaan naar de reeds bestaande mogelijkheden om data te persisteren binnen een applicatie in Android. Het is belangrijk om nogmaals te vermelden dat dit onderzoek enkel de databases benadert binnen Android om data lokaal op te slaan op het device. De mogelijkheden om data op te slaan op een remote server over een netwerk, worden binnen dit onderzoek niet benaderd.

Uiteindelijk is een keuze gemaakt om volgende vier mogelijkheden voor opslag van data te onderzoeken: SQLite, GreenDAO, Realm en SharedPreferences.

2.2 Literatuurstudie

Na de keuze van de te onderzoeken mogelijkheden om data te persisteren, is een research gedaan naar deze mogelijkheden. Eerst is een research gedaan over SQLite, omdat het de standaard database is die binnen Android wordt gebruikt. Samen met SQLite is parallel een onderzoek gedaan naar GreenDao, want GreenDAO is een ORM tool voor Android die een object-georiënteerde interface biedt op de relationele database SQLite. GreenDao maakt dus gebruik van SQLite

Daarna is er een research gedaan naar een redelijk nieuwe methode om data op te slaan binnen Android, namelijk Realm. Volgende onderdelen zijn onderzocht: de architectuur, de werking en het gebruik ervan. Realm is de eerste database die enkel gericht is op het werken op smartphones, tablets en wearables, en maakt geen gebruik van SQLite. Het is wel net zoals SQLite een file-based database, maar heeft een eigen manier ontwikkeld om data op te slaan en op te halen.

Als laatst is er ook nog een research gedaan naar de werking en gebruik van SharedPreferences. Hieruit is gebleken dat SharedPreferences enkel zou moeten gebruikt

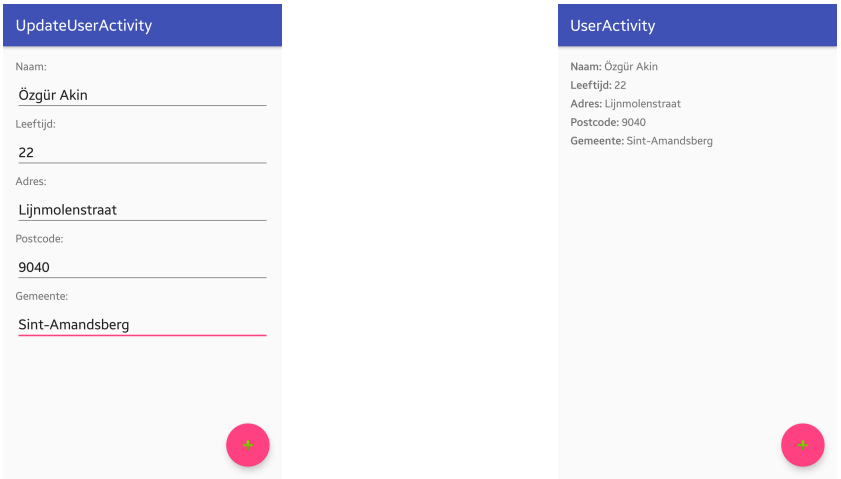
worden om een kleine hoeveelheid aan data op te slaan, aangezien deze gebruikt wordt om key-value paren op te slaan. Niet enkel is het moeilijker om een grote hoeveelheid aan data op te slaan met SharedPreferences, maar ook veel moelijker om de opgeslagen data op te halen. Hierdoor zal deze persistentiemogelijkheid enkel onderzocht worden binnen de applicatieprofiel waarbij weinig data moet gepersisteerd worden.

2.3 Voorbereiding op onderzoek

Na het verzamelen van de nodige informatie, is de voorbereiding op het onderzoek gestart. Eerst zijn er drie profielen van applicaties beschreven:

1. Een applicatie waarin **weinig data** moet opgeslaan worden. Hiervoor is een applicatie ontwikkeld, zoals te zien is op figuur 2.1, die de gegevens van de gebruiker vraagt zoals de naam, de leeftijd, het adres,.. Deze gegevens worden dan gepersisteerd en weergegeven op het scherm. Elke keer dat men de applicatie opent, ziet men de laatst ingevoerde gegevens. Deze gegevens kunnen altijd aangepast worden. Bij deze applicatie is het enkel mogelijk om de gegevens van één gebruiker op te slaan binnen dezelfde applicatie. Er kunnen dus geen gegevens van twee of meer gebruikers tegelijkertijd bewaard worden.

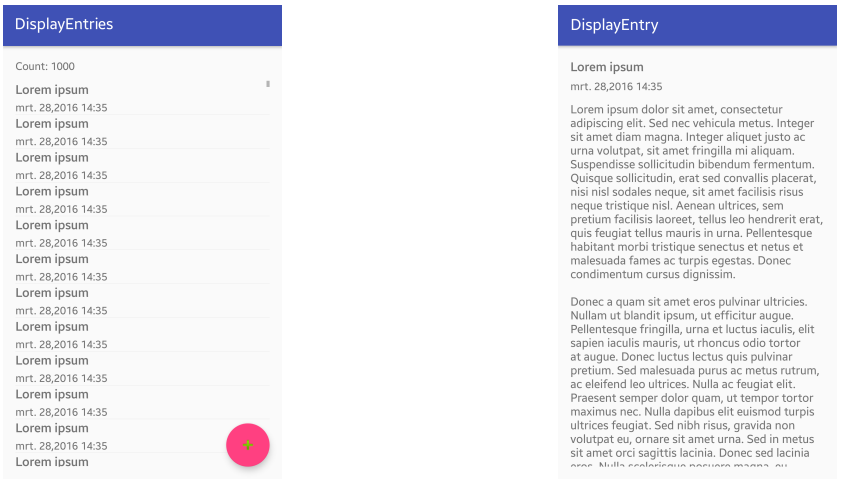
Figuur 2.1: Screenshots van de applicatie met weinig data



(a) Invoerscherm (b) Detailscherm

2. Een applicatie waarin een **gemiddelde hoeveelheid** data moet gepersisteerd worden. Hierbij denken we aan de opslag van een paar duizend records. Hiervoor is een dagboek applicatie ontwikkeld, die te zien is op figuur 2.2. Bij het openen

Figuur 2.2: Screenshots van de applicatie met een gemiddelde hoeveelheid data



(a) Lijst van entries

(b) Detailscherm van een entry

van de applicatie wordt er een scherm weergegeven met de lijst van alle entries. Bovenaan links wordt er ook weergegeven hoeveel entries er al in totaal bestaan. Wanneer er op een entry uit de lijst geklikt wordt, komt er een detailscherm tevoorschijn, die de details van de aangeklikte entry weergeeft.

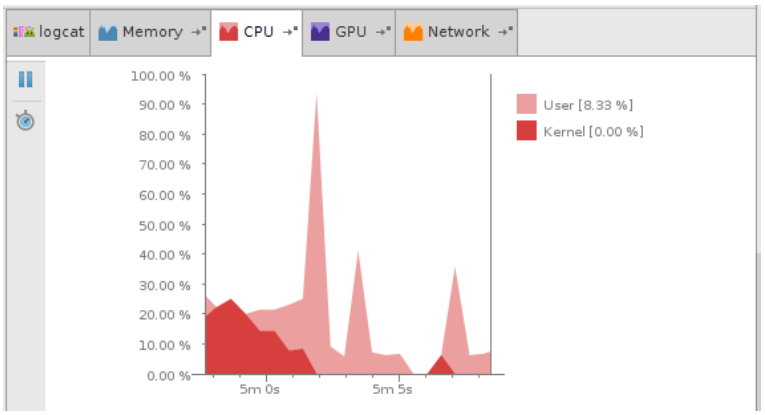
3. Een applicatie waarin **veel data** moet gepersisteerd worden. Hierbij denken we aan de opslag van honderd duizend records. Hiervoor is een dagboek applicatie ontwikkeld, die te zien is op figuur 2.2. Dit is dezelfde applicatie die gebruikt wordt bij het tweede profiel, maar dan met veel meer data die is gepersisteerd.

2.4 Gebruikte tools

2.4.1 Opstartsnelheid

Om de opstartsnelheid van de applicaties te meten, is de **monkeyrunner** tool gebruikt van Android. De monkeyrunner tool biedt een API voor het schrijven van programma's die een Android-apparaat of emulator besturen van buiten de Android code. Met het gebruik van deze tool, kan er een Python programma geschreven worden dat een Android applicatie of een test package installeert en deze runt. Met monkeyrunner kunnen er ook screenshots gemaakt worden van de applicatie terwijl het Python programma bezig is. Deze screenshots worden dan opgeslaan op het werkstation. Dit programma is bedoeld om applicaties en devices te testen op functioneel/ framework niveau, maar het kan ook gebruikt worden voor andere doeleinden (AndroidDevelopers, 2016b).

Figuur 2.3: CPU-Monitor van Android Studio (AndroidDevelopers, 2016a)



2.4.2 CPU-gebruik

Om het CPU-gebruik van de applicaties te meten, is de **CPU Monitor van Android Studio** gebruikt. Met de CPU Monitor van Android Studio kan eenvoudig het CPU-gebruik van de applicatie gecontroleerd worden. Het toont het CPU-gebruik in real time en geeft het percentage weer van het totale CPU-gebruik (AndroidDevelopers, 2016a).

2.4.3 Performantie

Om de performantie, meer bepaald de snelheid van de database bij insert operaties, te meten; is er gebruik gemaakt van Android studio zelf. Het is belangrijk te vermelden dat hier alle records in één transactie worden toegevoegd aan de database.

Hoofdstuk 3

Onderzoek

3.1 Manier van werken

In deze sectie wordt beschreven op welke manier data is gepersisteerd binnen de applicaties. Enkel de eerste applicaties waarin weinig data wordt opgeslaan, worden hier beschreven. De manier van persisteren verloopt analoog bij de andere applicaties die gebruikt zijn binnen dit onderzoek.

3.1.1 SharedPreferences

Zoals reeds vermeld is, wordt bij SharedPreferences data gepersisteerd in key-value paren. Binnen deze applicatie (waarin weinig data wordt gepersisteerd) werden de attributen (NAME, ADDRESS,..) gebruikt als key, en werd de erbij horende waarde opgeslaan als value. Hier is een deel van de code te zien, die toont op welke manier de gegevens worden gepersisteerd binnen de applicatie:

```
import android.content.Context;
import android.content.SharedPreferences;

public class PreferencesHelper {
   private static final String PREFERENCES_NAME = "userPrefs";
   private static final String NAME = "name";
   private static final String SAVED = "saved";
   private static final String AGE = "age";

   public static SharedPreferences getPreferences(final Context context){
      return context.getSharedPreferences(PREFERENCES_NAME,
          Context.MODE_PRIVATE);
   }

   public static void saveName(Context context, String name) {
      getPreferences(context)
         .edit()
         .putString(NAME, name)
         .apply();
      setUserSaved(context, true);
   }

   public static String getName(Context context) {
      return getPreferences(context).getString(NAME, "");
   }

   private static void setUserSaved(Context context, boolean saved) {
      getPreferences(context).edit().putBoolean(SAVED, saved).apply();
   }

   public static boolean isUserSaved(Context context){
      return getPreferences(context).getBoolean(SAVED, false);
   }

   public static void saveAge(Context context, int age) {
      getPreferences(context).edit().putInt(AGE, age).apply();
   }
}
```

3.1.2 GreenDAO

Bij gebruik van GreenDAO werd eerst een Java Module gecreëerd. Binnen deze module werd een klasse gemaakt, namelijk *UserDaoGenerator*, die de nodige code bevat voor het genereren van de klassen die nodig zijn om aan slag te kunnen gaan met GreenDAO: *DaoMaster, DaoSession, User, UserDao*. De code van de klasse *UserDaoGenerator* is hier te zien:

```
import de.greenrobot.daogenerator.DaoGenerator;
import de.greenrobot.daogenerator.Entity;
import de.greenrobot.daogenerator.Schema;

public class UserDaoGenerator {
   public static void main(String[] args) throws Exception {
      Schema schema = new Schema(1000, "de.greenrobot.daoexample");
      addNote(schema);
      new DaoGenerator().generateAll(schema, "../app/src/main/java");
   }

   private static void addNote(Schema schema) {
      Entity note = schema.addEntity("User");
      note.addIdProperty().autoincrement();
      note.addStringProperty("name");
      note.addIntProperty("age");
      note.addStringProperty("address");
      note.addIntProperty("zipcode");
      note.addStringProperty("city");
   }
}
```

Binnen de applicatie werd de nodige data dan gepersisteerd, gebruik makend van de gegenereerde klassen. Een voorbeeld daarvan is hier te zien:

```
   DaoMaster.DevOpenHelper helper = new DaoMaster.DevOpenHelper(this,
       "users-db", null);
   this.database = helper.getWritableDatabase();
   daoMaster = new DaoMaster(this.database);
   daoSession = this.daoMaster.newSession();
   userDao = this.daoSession.getUserDao();

   this.userDao.insert(user);
```

3.1.3 SQLite

Om gebruik te kunnen maken van SQLite werd er eerst een klasse *DatabaseHelper* gecreëerd die overerft van de klasse SQLiteOpenHelper. Dit is een helper dat nodig is om de creatie van de database en de versie ervan te beheren. Daarna is er ook een klasse *UserTable* gecreëerd die de nodige gegevens bevat om de tabel *Users* aan te maken binnen de database. De code van deze klasse is hier te zien:

```
public class UserTable {
   public static final String TABLE_NAME = "user";

   public static final String COLUMN_NAME = "name";
   public static final String COLUMN_AGE = "age";
   public static final String COLUMN_ADDRESS = "address";
   public static final String COLUMN_ZIPCODE = "zipcode";
   public static final String COLUMN_CITY = "city";

   private static final String CREATE_TABLE = "create table IF NOT
       EXISTS " + TABLE_NAME
      + " (_id integer primary key autoincrement, "
      + COLUMN_NAME + " text, "
      + COLUMN_AGE + " numeric, "
      + COLUMN_ADDRESS + " text, "
      + COLUMN_ZIPCODE + " numeric, "
      + COLUMN_CITY + " text "
      + ");";

   public static void onCreate(SQLiteDatabase db) {
      db.execSQL(CREATE_TABLE);
   }

   public static User constructFromDatabase(Cursor cursor) {
      User user = new User();
      user.setName(cursor.getString(cursor.getColumnIndex(COLUMN_NAME)));
      user.setAge(cursor.getInt(cursor.getColumnIndex(COLUMN_AGE)));
      user.setAdress(cursor.getString(cursor.getColumnIndex(COLUMN_ADDRESS)));
      user.setZipcode(cursor.getInt(cursor.getColumnIndex(COLUMN_ZIPCODE)));
      user.setCity(cursor.getString(cursor.getColumnIndex(COLUMN_CITY)));
      return user;
   }

   public static ContentValues getContentValues(User user) {
      ContentValues contentValues = new ContentValues();
```

```
        contentValues.put(COLUMN_NAME, user.getName());
        contentValues.put(COLUMN_AGE, user.getAge());
        contentValues.put(COLUMN_ADDRESS, user.getAdress());
        contentValues.put(COLUMN_ZIPCODE, user.getZipcode());
        contentValues.put(COLUMN_CITY, user.getCity());

        return contentValues;
    }
}
```

Deze twee klassen werden dan gebruikt om gegevens te kunnen persisteren in de database. Hier volgt een voorbeeld:

```
this.database = new DatabaseHelper.getWritableDatabase();
this.database.insert(UserTable.TABLE_NAME, null,
    UserTable.getContentValues(user));
```

3.1.4 Realm

Om gebruik te maken van Realm, heeft men de klasse waarvan objecten moeten gepersisteerd worden, laten overerven van de klasse *RealmObject*. Hier is een voorbeeld te zien (de getters en setters zijn weggelaten):

```
import io.realm.RealmObject;

public class User extends RealmObject
{
    private String name;
    private int age;
    private String adress;
    private int zipcode;
    private String city;

    //getters en setters
}
```

De nodige gegevens werden dan als volgt gepersisteerd:

```
RealmConfiguration realmConfiguration = new
    RealmConfiguration.Builder(this).build();
Realm realm = Realm.getInstance(realmConfiguration);

realm.beginTransaction();
realm.deleteAll();
realm.copyToRealm(user);
realm.commitTransaction();
```

3.2 Database en opstartsnelheid

3.2.1 Applicatie

Om de opstartsnelheid te meten bij profiel 1, is een applicatie gebruikt die enkele gegevens van de gebruiker bewaart, te zien op figuur 2.1. Dit applicatie is vier maal geschreven, telkens gebruik makend van een ander mogelijkheid om data te persisteren.

Om de opstartsnelheid bij profiel 2 en 3 te meten, is de dagboekapplicatie gebruikt.

3.2.2 Experiment

Bij dit experiment zijn twee aparte metingen gedaan: de opstartsnelheid van de applicatie bij de eerste installatie en de opstartsnelheid van de applicatie bij alle volgende keren dat de applicatie geopend wordt.

Om de opstartsnelheid te meten bij de **eerste opstart** is er gebruik gemaakt van de monkeyrunner tool van Android. Er werd een script geschreven die de opgegeven applicatie installeert op het device en deze terug verwijdert op het einde. Een deel van het script is hier te zien:

```
// sets a variable with the package's internal name
package = 'com.example.ozgur.SharedPref1'
// sets a variable with the name of an Activity in the package
activity = 'com.example.ozgur.SharedPref1.MainActivity'
// sets the name of the component to start
runComponent = package + '/' + activity
...
// Uninstalls the package
device.removePackage 'com.example.ozgur.SharedPref1'
```

Na het script een gewenst aantal keren uitgevoerd te hebben, in dit geval 30 keer, werd een commando uitgevoerd in de adb (Android Debug Bridge) shell van

Android Studio: *logcat -b events | grep am_activity_launch_time*. De output van dit commando is een lijst met alle activities die recent gestart zijn. Een item uit de lijst bevat volgende parameters: EventLogTags.AM_ACTIVITY_LAUNCH_TIME, userId, System.identityHashCode, shortComponentName, thisTime, totalTime. De opstartsnelheid is hier gelijk aan *thisTime*.

Om de opstartsnelheid te meten bij de volgende opstarten is er gebruik gemaakt van de adb shell. Er werd een script geschreven die 100 keer de applicatie start en daarna terug afsluit. Nadat de het script werd uitgevoerd werd opnieuw het commando *logcat -b events | grep am_activity_launch_time* uitgevoerd om op dezelfde manier de opstartsnelheden te kunnen zien.

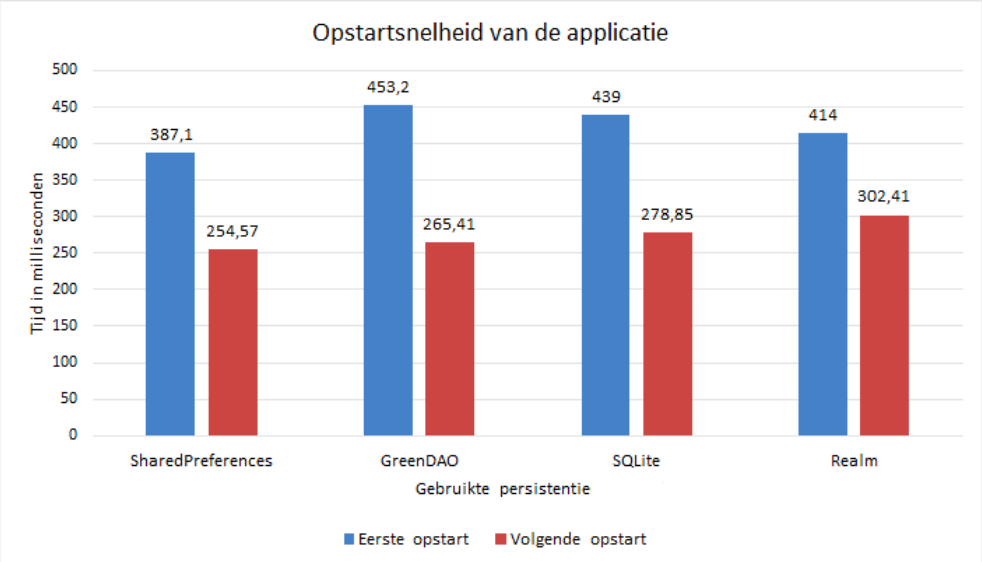
De output van het commando *logcat -b events | grep am_activity_launch_time* werd telkens werggeschreven naar een file, zodat de opstartsnelheden er konden uit gehaald worden om opnieuw weggeschreven te kunnen worden naar een andere file.

3.2.3 Resultaat

Profiel 1: Weinig data

De opstartsnelheden verschillen niet veel van elkaar zoals te zien op de tabellen 3.1 en 3.2. Er zit wel een opmerkelijk verschil tussen de opstartsnelheid bij eerste opstart en de opstartsnelheid bij cold start (volgende opstarten). Dit komt door het feit dat bij de installatie de database nog geïnitialiseerd moet worden. Bij de volgende keren moet dit niet meer gebeuren, en kan de database direct gebruikt worden.

Figuur 3.1: Opstartsnelheid van de applicatie



Tabel 3.1: Meetwaarden bij eerste opstart (weinig data)

	SharedPreferences	GreenDAO	SQLite	Realm
Gemiddelde (in ms)	387,10	453,20	439,00	414,00
Minimum (in ms)	332,00	383,00	372,00	353,00
Q1 (in ms)	381,50	413,00	422,00	371,25
Mediaan (in ms)	389,50	434,50	426,50	378,50
Q3 (in ms)	402,75	511,75	460,00	445,00
Maximum (in ms)	433,00	541,00	533,00	555,00
Standaardafwijking (in ms)	19,30	50,44	32,60	53,80
Aantal keren uitgevoerd	30	30	30	30

Tabel 3.2: Meetwaarden bij volgende opstart (weinig data)

	SharedPreferences	GreenDAO	SQLite	Realm
Gemiddelde (in ms)	254,57	265,41	278,85	302,41
Minimum (in ms)	198,00	206,00	201,00	222,00
Q1 (in ms)	225,75	246,75	240,00	280,00
Mediaan (in ms)	243,00	266,50	269,00	302,00
Q3 (in ms)	262,00	282,00	303,50	319,25
Maximum (in ms)	480,00	444,00	556,00	516,00
Standaardafwijking (in ms)	31,21	22,31	37,80	29,05
Aantal keren uitgevoerd	100	100	100	100

Profiel 2: Gemiddelde hoeveelheid data

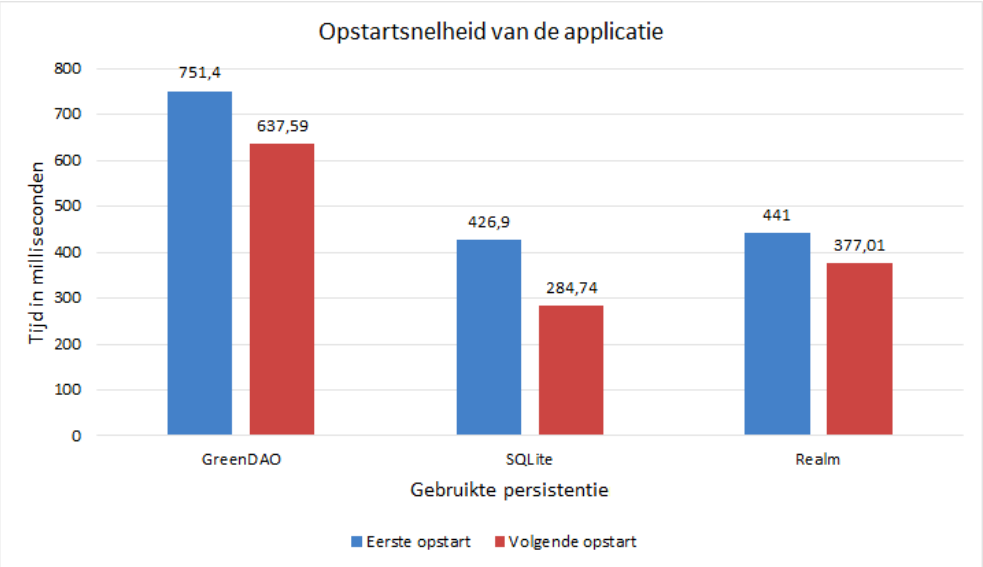
Bij de meting van de opstartsnelheid van de applicatie waarin er een gemiddelde hoeveelheid data in gepersisteerd is, werd SharedPreferences niet gebruikt.

Zoals te zien is op figuur 3.2 en de tabellen 3.3 en 3.4, neemt het opstarten van de applicatie meer tijd in beslag wanneer GreenDAO gebruikt wordt als persistentiemogelijkheid. Het meeste tijd kruipt hier in het ophalen van de data bij het opstarten van de applicatie. Zoals in het begin vermeld werd, is GreenDAO een ORM-laag op SQLite. Dit zorgt ervoor dat het opstarten van de applicatie bij gebruik van GreenDAO een vertraging oploopt in vergelijking met de opstartsnelheid van bij gebruik van SQLite.

Zoals te zien is op de tabellen 3.3 en 3.4, is er geen opmerkelijk groot verschil tussen de meetwaarden van SQLite. Het verschil is verwaarloosbaar klein.

Met deze data kan niet gezegd worden dat de opstartsnelheid het snelst is bij gebruik van een bepaalde persistentiemogelijkheid, omdat de meetwaarden bij SQLite en Realm heel dicht bij elkaar liggen. Er kan wel gezegd worden dat de opstartsnelheid van de applicatie het traagst is bij gebruik van GreenDAO.

Figuur 3.2: Opstartsnelheid van de applicatie



Tabel 3.3: Meetwaarden bij eerste opstart (gemiddelde hoeveelheid data)

	GreenDAO	**SQLite**	**Realm**
Gemiddelde (in ms)	751,40	426,90	441,00
Minimum (in ms)	597,00	358,00	331,00
Q1 (in ms)	687,00	384,75	431,75
Mediaan (in ms)	759,50	406,00	450,50
Q3 (in ms)	804,00	467,75	458,00
Maximum (in ms)	867,00	525,00	495,00
Standaardafwijking (in ms)	67,12	48,48	27,80
Aantal keren uitgevoerd	30	30	30

Tabel 3.4: Meetwaarden bij volgende opstart (gemiddelde hoeveelheid data)

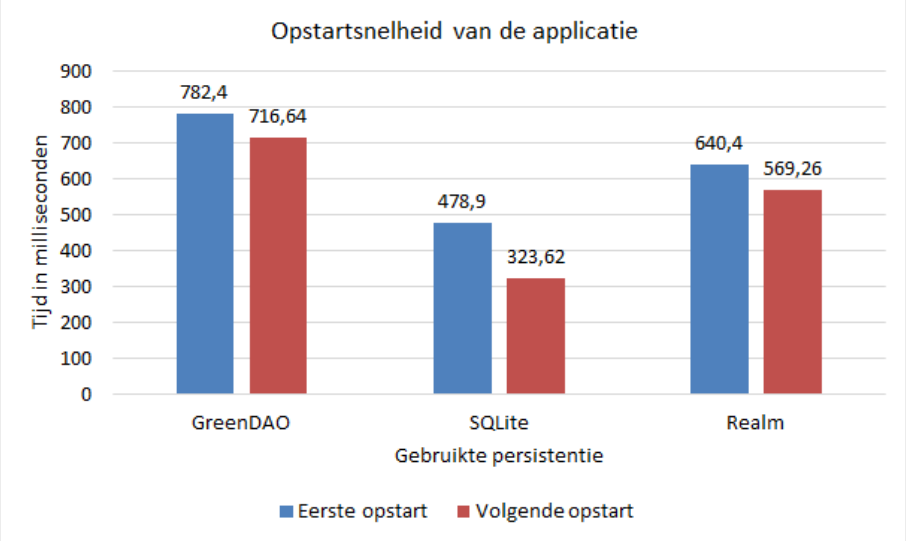
	GreenDAO	SQLite	Realm
Gemiddelde (in ms)	637,59	284,74	377,01
Minimum (in ms)	540,00	194,00	299,00
Q1 (in ms)	572,75	242,25	350,00
Mediaan (in ms)	647,00	277,00	381,50
Q3 (in ms)	685,25	314,50	405,25
Maximum (in ms)	767,00	548,00	457,00
Standaardafwijking (in ms)	54,09	44,49	28,45
Aantal keren uitgevoerd	100	100	100

Profiel 3: Veel data

Het resultaat van dit onderzoek verschilt niet veel van het resultaat van het onderzoek van profiel 2.

De opstartsnelheid van de applicatie bij gebruik van GreenDAO is opnieuw het traagst, maar het verschil tussen de meetwaarden van SQLite en Realm is groter geworden. De gemiddelde opstartsnelheid bij gebruik van SQLite is sneller dan de gemiddelde opstartsnelheid bij gebruik van Realm.

Figuur 3.3: Opstartsnelheid van de applicatie



Tabel 3.5: Meetwaarden bij eerste opstart (veel data)

	GreenDAO	**SQLite**	**Realm**
Gemiddelde (in ms)	782,40	478,90	640,40
Minimum (in ms)	643,00	412,00	428,00
Q1 (in ms)	714,75	427,75	495,50
Mediaan (in ms)	792,50	455,00	687,50
Q3 (in ms)	852,75	484,00	780,75
Maximum (in ms)	898,00	641,00	807,00
Standaardafwijking (in ms)	72,52	55,26	137,12
Aantal keren uitgevoerd	30	30	30

Tabel 3.6: Meetwaarden bij volgende opstart (veel data)

	GreenDAO	**SQLite**	**Realm**
Gemiddelde (in ms)	716,64	323,62	569,26
Minimum (in ms)	545,00	214,00	280,00
Q1 (in ms)	667,50	258,50	480,00
Mediaan (in ms)	725,00	294,00	567,50
Q3 (in ms)	749,50	367,75	667,25
Maximum (in ms)	951,00	744,00	761,00
Standaardafwijking (in ms)	52,23	76,00	92,18
Aantal keren uitgevoerd	100	100	100

3.3 Database en CPU-gebruik

3.3.1 Applicatie

Om het CPU-gebruik van de applicatie te meten bij profiel 1 is opnieuw de applicatie gebruikt met weinig data, te zien op figuur 2.1.

Om het CPU-gebruik van de applicatie te meten bij profiel 2 en 3, is een dagboekapplicatie gebruikt, zie figuur 2.1. De dagboek entries die toegevoegd worden, bevatten allemaal dezelfde gegevens (titel, inhoud en datum):

```
public static final String SAMPLE_TITLE = "Lorem ipsum";

public static final String CONTENT = "Lorem ipsum dolor sit amet,
    consectetur adipiscing elit. Sed nec vehicula metus. Integer sit
    amet diam magna. Integer aliquet justo ac urna volutpat, sit amet
    fringilla mi aliquam. Suspendisse sollicitudin bibendum fermentum.
    Quisque sollicitudin, erat sed convallis placerat, nisi nisl sodales
    neque, sit amet facilisis risus neque tristique nisl. Aenean
    ultrices, sem pretium facilisis laoreet, tellus leo hendrerit erat,
    quis feugiat tellus mauris in urna. Pellentesque habitant morbi
    tristique senectus et netus et malesuada fames ac turpis egestas.
    Donec condimentum cursus dignissim.

Donec a quam sit amet eros pulvinar ultricies. Nullam ut blandit ipsum,
    ut efficitur augue. Pellentesque fringilla, urna et luctus iaculis,
    elit sapien iaculis mauris, ut rhoncus odio tortor at augue. Donec
    luctus lectus quis pulvinar pretium. Sed malesuada purus ac metus
    rutrum, ac eleifend leo ultrices. Nulla ac feugiat elit. Praesent
    semper dolor quam, ut tempor tortor maximus nec. Nulla dapibus elit
    euismod turpis ultrices feugiat. Sed nibh risus, gravida non
    volutpat eu, ornare sit amet urna. Sed in metus sit amet orci
    sagittis lacinia. Donec sed lacinia eros. Nulla scelerisque posuere
    magna, eu aliquet massa semper sit amet.";
```

Dit applicatie is drie maal geschreven, telkens gebruik makend van een ander mogelijkheid om data te persisteren. De persistentiemogelijkheid SharedPreferences is hier niet onderzocht geweest, omdat deze enkel bedoeld is voor opslag van simpele data in kleine hoeveelheid.

3.3.2 Experiment

Bij dit experiment is het CPU-gebruik gemeten. Eerst werd er een analyse gedaan om te zien wanneer het CPU-gebruik maximaal is. Tijdens de analyse werd geconcludeerd dat

het CPU-gebruik maximaal is, wanneer er records worden toegevoegd aan de database.

Daarom is er enkel onderzoek gedaan op het CPU-gebruik op het moment dat er records worden toegevoegd aan de database. Het aantal records dat werd toegevoegd, was telkens hetzelfde per profiel.

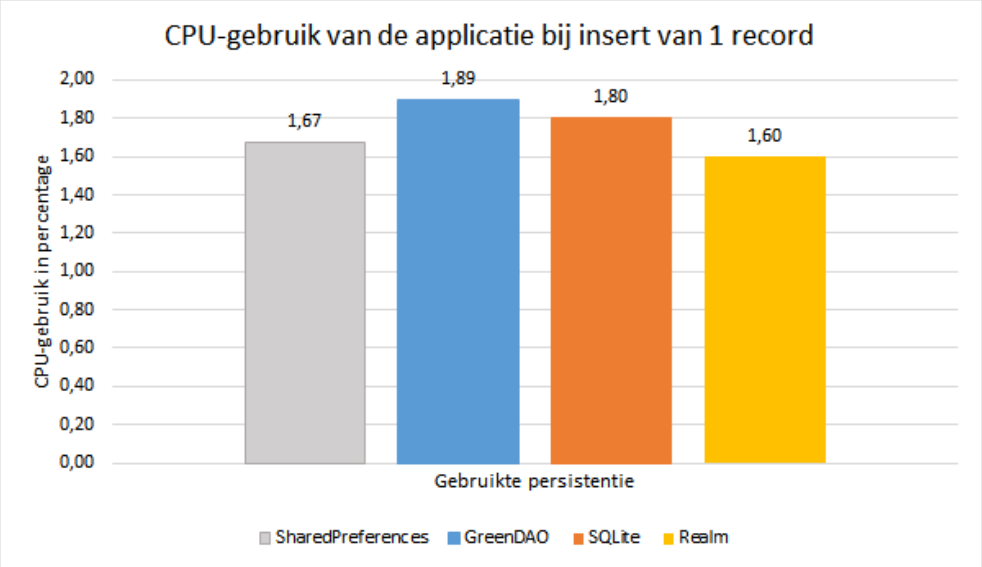
3.3.3 Resultaat

Profiel 1: Weinig data

Bij dit experiment is het CPU-gebruik gemeten bij het opslaan van de gegevens van een gebruiker. Een voorbeeld is te zien op figuur 2.1. Er werden telkens dezelfde gegevens ingevoerd, om sverschillen in de metingen te voorkomen. Het verschil in grootte van de gegevens die moeten gepersisteerd worden, zou immers kunnen zorgen voor verschillen in CPU-gebruik.

Op figuur 3.4 is het resultaat te zien van 30 metingen. Het verschil tussen het gemiddeld CPU-gebruik van de verschillende persistentiemogelijkheden is relatief klein, maar toch is het gemiddelde CPU-gebruik bij Realm het laagst.

Figuur 3.4: CPU-gebruik van de applicatie



Tabel 3.7: Meetwaarden CPU-gebruik

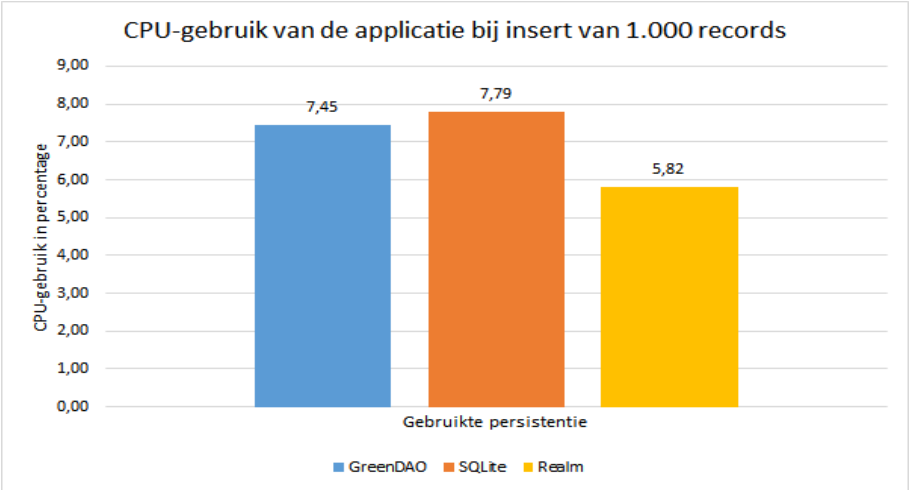
	SharedPreferences	GreenDAO	SQLite	Realm
Gemiddelde (in %)	1,67	1,89	1,80	1,60
Minimum (in %)	1,09	1,31	1,10	1,12
Q1 (in %)	1,55	1,65	1,56	1,34
Mediaan (in %)	1,77	1,82	1,78	1,58
Q3 (in %)	1,81	2,06	2,06	1,78
Maximum (in %)	2,23	2,67	2,68	2,27
Standaardafwijking (in %)	0,23	0,28	0,33	0,24
Aantal keren uitgevoerd	30	30	30	30

Profiel 2: Gemiddelde hoeveelheid data

Bij dit experiment is het CPU-gebruik gemeten bij het opslaan van entries. Er werden 1.000 records toegevoegd aan de database in een simpele transactie. Alle records bevatten dezelfde gegevens: titel, datum en inhoud. Zo werden verschillen door grootte van objecten geëlimineerd.

Op figuur 3.5 is het resultaat te zien van 30 metingen. Het verschil tussen het gemiddeld CPU-gebruik van de applicatie bij gebruik van SQLite en GreenDAO is relatief klein, maar bij gebruik van Realm gebruikt de applicatie minder CPU.

Figuur 3.5: CPU-gebruik van de applicatie



Tabel 3.8: Meetwaarden CPU-gebruik

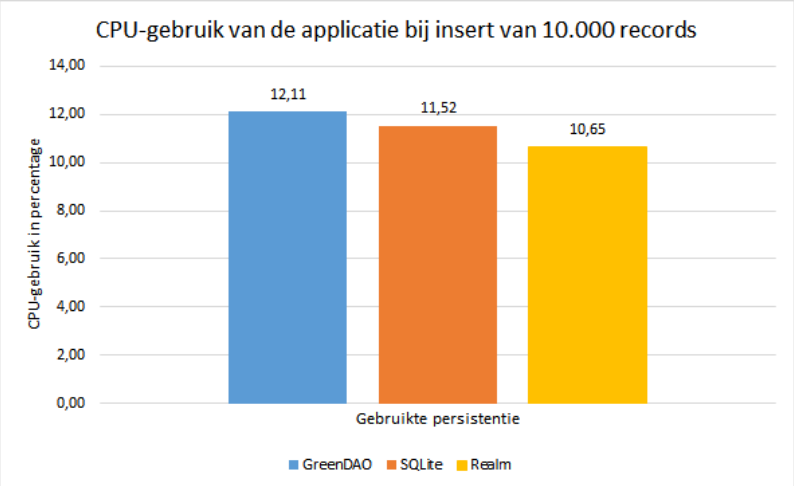
	GreenDAO	SQLite	Realm
Gemiddelde (in %)	7,45	7,79	5,82
Minimum (in %)	4,18	4,04	3,79
Q1 (in %)	6,07	6,91	4,89
Mediaan (in %)	7,43	7,88	5,69
Q3 (in %)	8,72	8,93	6,41
Maximum (in %)	11,94	11,26	10,34
Standaardafwijking (in %)	1,60	1,27	0,99
Aantal keren uitgevoerd	30	30	30

Profiel 3: Veel data

Bij dit experiment is het CPU-gebruik gemeten bij het opslaan van entries. Er werden 10.000 records toegevoegd aan de database in een simpele transactie. Alle records bevatten dezelfde gegevens: titel, datum en inhoud. Zo werden verschillen door grootte van objecten geëlimineerd.

Op figuur 3.6 is het resultaat te zien van 30 metingen. Het verschil tussen het gemiddeld CPU-gebruik van de applicatie bij gebruik van de verschillende persistentiemogelijkheden is relatief klein.

Figuur 3.6: CPU-gebruik van de applicatie



Tabel 3.9: Meetwaarden CPU-gebruik

	GreenDAO	**SQLite**	**Realm**
Gemiddelde (in %)	12,11	11,52	10,65
Minimum (in %)	10,38	10,58	7,68
Q1 (in %)	11,48	11,12	9,67
Mediaan (in %)	12,05	11,43	10,66
Q3 (in %)	12,70	11,78	11,83
Maximum (in %)	13,56	12,96	13,36
Standaardafwijking (in %)	0,71	0,44	1,15
Aantal keren uitgevoerd	30	30	30

3.4 Database en performantie

3.4.1 Applicatie

Om de performantie op vlak van inserts te meten, is opnieuw de dagboekapplicatie gebruikt, zie figuur 2.1. Hier is de persistentiemogelijkheid SharedPreferences ook buiten beschouwing gelaten, omdat deze enkel bedoeld is voor simpele data in kleine hoeveelheid.

3.4.2 Experiment

Bij dit experiment is de snelheid gemeten bij het invoeren van een bepaald aantal records in de database. Het aantal records verschilt per profiel. Het is belangrijk om te vermelden dat alle records toegevoegd zijn aan de database in een simpele transactie. De manier van werken wordt weergegeven per gebruikte persistentiemogelijkheid.

GreenDAO

- Profiel 1:

```
    this.userDao.insert(user);
```

- Profiel 2 en 3:

```
database.beginTransaction();
try
{
   for (int i = 0; i < 1000; i++)
   {
      this.entryDao.insert(entry);
   }
   database.setTransactionSuccessful();
} catch (Exception ex){}
finally
{
   database.endTransaction();
}
```

SQLite

- Profiel 1:

```
this.database.insert(UserTable.TABLE_NAME, null,
    UserTable.getContentValues(user));
```

- Profiel 2 en 3:

```
database.beginTransaction();
try
{
   for (int i = 0; i < 1000; i++)
   {
      this.adapter.insert(values);
   }
   database.setTransactionSuccessful();
}catch (Exception ex){}
finally
{ database.endTransaction(); }
```

Realm

- Profiel 1:

```
realm.copyToRealm(user);
```

- Profiel 2 en 3: Bij Realm kan er gebruik gemaakt worden van de methode *realm.executeTransaction()*. Deze methode zal zelf de transactie starten en eindigen, en annuleren indien er iets fout loopt (Realm, 2016).

```
realm.executeTransaction(new Realm.Transaction() {
   @Override
   public void execute(Realm realm)
   {
      for (int i = 0; i < 1000; i++)
      {
         realm.copyToRealm(entry);
      }
   }
});
```

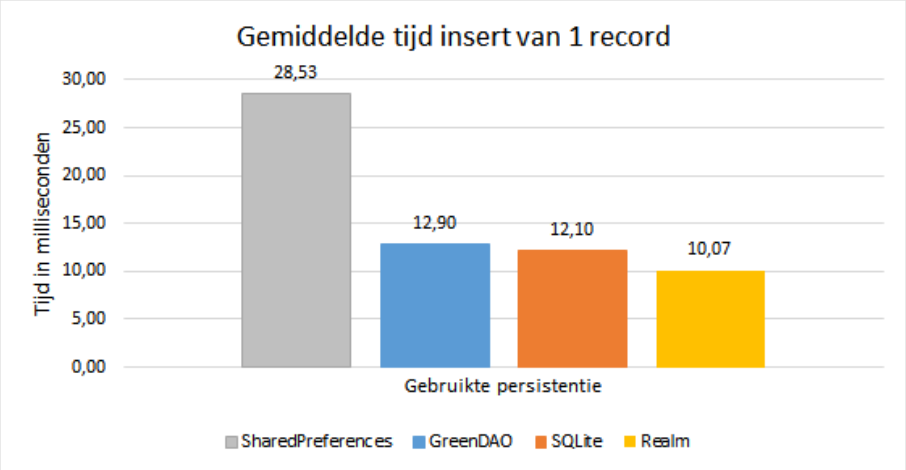
3.4.3 Resultaat

Profiel 1: Weinig data

Op figuur 3.7 is te zien dat de gemiddelde tijd voor een insert bij gebruik van SharedPreferences meer dan het dubbele gemiddelde tijd van een insert in beslag neemt bij gebruik van de andere databases.

De meetwaarden bij gebruik van de andere drie persistentiemogelijkheden liggen dict bij elkaar, maar bij gebruik van Realm is de gemiddelde tijd voor een insert toch wat minder.

Figuur 3.7: Performantie van de applicatie bij inserts



Tabel 3.10: Meetwaarden bij performantie

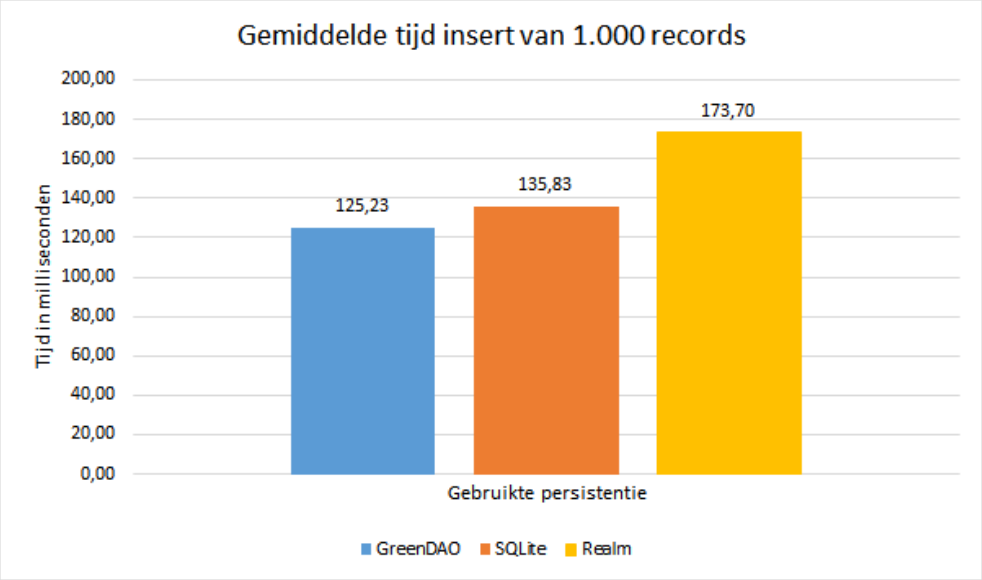
	SharedPreferences	GreenDAO	SQLite	Realm
Gemiddelde (in ms)	28,53	12,90	12,10	10,07
Minimum (in ms)	19,00	8,00	8,00	7,00
Q1 (in ms)	23,00	10,25	11,00	9,00
Mediaan (in ms)	29,00	12,00	12,00	9,50
Q3 (in ms)	32,75	14,00	13,75	11,00
Maximum (in ms)	41,00	24,00	17,00	15,00
Standaardafwijking (in ms)	5,56	2,74	1,81	1,49
Aantal keren uitgevoerd	30	30	30	30

Profiel 2: Gemiddelde hoeveelheid data

Op figuur 3.8 is te zien dat de gemiddelde tijd voor een insert van 1.000 records bij gebruik van Realm opmerkelijk meer is dan de gemiddelde tijd van een insert van 1.000 records bij gebruik van de andere databases.

De meetwaarden bij gebruik van SQLite en GreenDAO liggen dicht bij elkaar, maar GreenDAO scoort iets beter.

Figuur 3.8: Performantie van de applicatie bij inserts



Tabel 3.11: Meetwaarden bij performantie

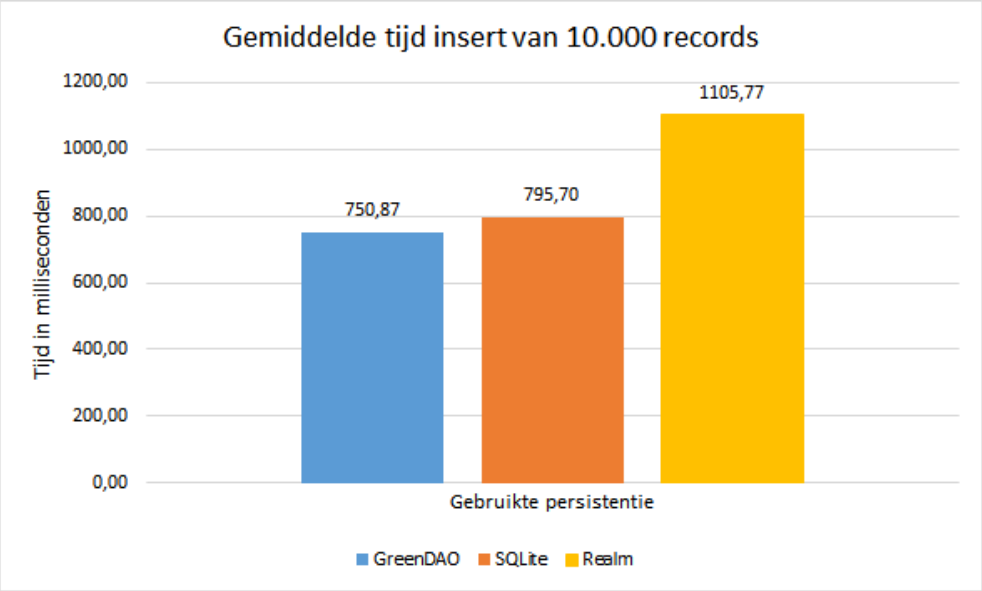
	GreenDAO	SQLite	Realm
Gemiddelde (in ms)	125,23	135,83	173,70
Minimum (in ms)	107,00	114,00	143,00
Q1 (in ms)	112,25	122,00	159,00
Mediaan (in ms)	120,00	136,00	172,00
Q3 (in ms)	138,00	144,00	186,75
Maximum (in ms)	169,00	167,00	230,00
Standaardafwijking (in ms)	13,46	11,97	16,95
Aantal keren uitgevoerd	30	30	30

Profiel 3: Veel data

Op figuur 3.8 is te zien dat het resultaat van dit onderzoek gelijkaardig is als het resultaat van het onderzoek met een gemiddelde hoeveelheid data.

De meetwaarden van Realm liggen aanzienlijk hoger dan de meetwaarden van GreenDAO en SQLite. GreenDAO scoort alweer iets beter dan SQLite.

Figuur 3.9: Performantie van de applicatie bij inserts



Tabel 3.12: Meetwaarden bij performantie

	GreenDAO	SQLite	Realm
Gemiddelde (in ms)	750,87	795,70	1105,77
Minimum (in ms)	702,00	734,00	979,00
Q1 (in ms)	739,75	771,25	1056,25
Mediaan (in ms)	750,00	794,00	1112,00
Q3 (in ms)	763,00	814,00	1156,50
Maximum (in ms)	792,00	859,00	1233,00
Standaardafwijking (in ms)	15,27	23,30	54,32
Aantal keren uitgevoerd	30	30	30

3.5 Kost voor de ontwikkelaar

3.5.1 SharedPreferences

Om SharedPreferences te kunnen gebruiken, moet er niet veel aan het project worden toegevoegd. Een key-value paar persisteren of de value opvragen aan de hand van de key, gebeurt als volgt:

- **Opvragen:**

```
context.getSharedPreferences(PREF_NAME, Context.MODE_PRIVATE)
   .getString(NAME, "");
```

- **Persisteren:**

```
context.getSharedPreferences(PREF_NAME, Context.MODE_PRIVATE)
   .edit()
   .putString(NAME, name)
   .commit();
```

3.5.2 GreenDAO

Om GreenDAO te kunnen gebruiken binnen het project, moeten er enkele stappen gevolgd worden:

1. Maak een nieuwe Java module aan binnen het project
2. Voeg de nodige dependency toe in de build.gradle file

```
dependencies {
   compile fileTree(dir: 'libs', include: ['*.jar'])
   compile('de.greenrobot:DaoGenerator:1.3.0')
}
```

3. Maak een nieuwe klasse aan in het Java module, met de beschrijving van de nodige schema's

```
public class DiaryDaoGenerator {
   public static void main(String[] args)throws Exception {
      Schema schema = new Schema(1000,
          "de.greenrobot.daoexample");
      addEntry(schema);
```

```
            new DaoGenerator().generateAll(schema,
                "../app/src/main/java");
        }

        private static void addEntry(Schema schema) {
            Entity note = schema.addEntity("Entry");
            note.addIdProperty().autoincrement();
            note.addStringProperty("title");
            note.addLongProperty("date");
            note.addStringProperty("content");
        }
    }
```

4. Run de aangemaakte klasse

Data ophalen of persisteren gebeurt dan als volgt:

- **Ophalen**

```
this.cursor = this.database.query(this.entryDao.getTablename(),
    this.entryDao.getAllColumns(), null, null, null, null, null);

if (cursor.moveToFirst())
{
    do
    {
        String title =
            cursor.getString(cursor.getColumnIndex(Constants.TITLE_NAME));
        String content =
            cursor.getString(cursor.getColumnIndex(Constants.CONTENT_NAME));

        long date =
            cursor.getLong(cursor.getColumnIndex(Constants.DATE_NAME));

        Entry entry = new Entry(null, title, date, content);

        this.entries.add(entry);
    }
    while (cursor.moveToNext());
}
```

- **Persisteren**

```
SQLiteDatabase database = entryDao.getDatabase();
database.beginTransaction();

try
{
   this.entryDao.insert(entry);
   database.setTransactionSuccessful();
}catch (Exception ex)
{
}finally
{
   database.endTransaction();
}
```

3.5.3 SQLite

Om SQLite te gebruiken binnen een project, moeten er ook enkele stappen gevolgd worden:

1. Eerst wordt een klasse aangemaakt die overerft van de klasse SQLiteOpenHelper, waarin onder andere de query wordt uitgetypt om de tabel(len) aan te maken

```
public class DatabaseHelper extends SQLiteOpenHelper{
   private static final String CREATE_TABLE = "CREATE TABLE "
       + Constants.TABLE_NAME + " (" +
   Constants.KEY_ID + " integer primary key autoincrement, " +
   Constants.TITLE_NAME + " text not null, " +
   Constants.CONTENT_NAME + " text not null, " +
   Constants.DATE_NAME + " long);";
   ...
}
```

2. ContentProvider toevoegen aan het project

```
public class DiaryContentProvider extends ContentProvider{
   private Database database;
   private static final UriMatcher uriMatcher;
   private static final int DIARIES = 1;

   public static final String AUTHORITY =
       "com.example.ozgur.diary";
```

```
    public static final Uri CONTENT_URI =
        Uri.parse("content://" + AUTHORITY
    + "/" + Constants.TABLE_NAME);
    ...
}
```

Het ophalen of persisteren van data gebeurt dan als volgt:

- **Ophalen:** Het ophalen van data gebeurt op een analoge manier als bij GreenDAO.
- **Persisteren:** Het persisteren van data gebeurt ook op een analoge manier als bij GreenDAO. Het enige verschil is dat je ContentValues moet doorgeven om een insert uit te voeren, in plaats van een object.

```
ContentValues values = new ContentValues();
values.put(Constants.TITLE_NAME, Constants.SAMPLE_TITLE);
values.put(Constants.CONTENT_NAME, Constants.SAMPLE_CONTENT);
values.put(Constants.DATE_NAME, System.currentTimeMillis());
```

3.5.4 Realm

Om Realm te gebruiken binnen het project moeten volgende stappen uitgevoerd worden:

1. Dependency toevoegen aan de build.gradle(project) file

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath "io.realm:realm-gradle-plugin:1.0.0"
    }
}
```

2. Dependency toevoegen aan de build.gradle(app) file

```
apply plugin: 'realm-android'
```

Het ophalen of persisteren van data gebeurt dan als volgt:

- **Ophalen:**

```
RealmConfiguration realmConfiguration = new
    RealmConfiguration.Builder(this).build();
Realm realm = Realm.getInstance(realmConfiguration);

return realm.where(DiaryEntry.class).findAll();
```

- **Persisteren:**

```
RealmConfiguration realmConfiguration = new
    RealmConfiguration.Builder(this).build();
Realm realm = Realm.getInstance(realmConfiguration);

realm.executeTransaction(new Realm.Transaction() {
   @Override
   public void execute(Realm realm) {
      realm.copyToRealm(entry);
   }
});
```

Hoofdstuk 4

Conclusie

4.1 Samenvattende tabel en reflectie over resultaat

4.1.1 Profiel 1: Weinig data

Tabel 4.1: Samenvatting van alle gegevens over applicatieprofiel 1

	SharedPreferences	**GreenDAO**	**SQLite**	**Realm**
Gemiddelde opstartsnelheid (in ms)	378,10	453,20	439,00	414,00
Gemiddelde CPU-gebruik (in %)	1,67	1,89	1,80	1,60
Gemiddelde snelheid van insert (in ms)	28,53	12,90	12,10	10,70

De verschillen tussen de gemiddelde opstartsnelheden zijn verwaarloosbaar klein, zoals te zien op tabel 4.1. Op vlak van CPU-gebruik en performantie (inserts) is Realm toch beter. Het is aangeraden om voor een applicatie waarin weinig data moet gepersisteerd worden, Realm te gebruiken.

Voor dit onderzoek uitgevoerd werd, werd er gedacht dat SharedPreferences de beste keuze zou zijn om mee te werken binnen een applicatie waarin een kleine hoeveelheid data moet gepersisteerd worden. Uit het onderzoek is gebleken dat Realm beter is dan SharedPreferences, behalve op vlak van opstartsnelheid. Het verschil tussen de gemiddelde opstartsnelheden is echter verwaarloosbaar klein, waardoor toch beter Realm gebruikt moet worden.

4.1.2 Profiel 2: Gemiddeldde hoeveelheid data

Tabel 4.2: Samenvatting van alle gegevens over applicatieprofiel 2

	GreenDAO	SQLite	Realm
Gemiddelde opstartsnelheid (in ms)	637,59	284,74	377,01
Gemiddelde CPU-gebruik (in %)	7,45	7,79	5,82
Gemiddelde snelheid van insert (in ms)	125,23	135,83	173,70

De verschillen tussen de gemiddelde opstartsnelheden zijn verwaarloosbaar klein bij SQLite en Realm, maar bij GreenDAO zien we toch wel dat het bijna dubbel zo veel is in vergelijking met de andere gemiddelde opstartsnelheden. Op vlak van CPU-gebruik liggen de gemiddeldes van GreenDAO en SQLite dicht bij elkaar. Het gemiddelde CPU-gebruik van Realm is gemiddeld 2% minder dan de andere gemiddeldes. Op vlak van inserts scoren GreenDAO en SQLite iets beter dan Realm, maar alweer is het verschil verwaarloosbaar klein. De combinatie van deze drie factoren en de kost voor de ontwikkelaar, zorgen ervoor dat Realm alweer aangeraden is voor dit applicatieprofiel.

Voor dit onderzoek uitgevoerd werd, werd er gedacht dat Realm de beste keuze zou zijn om mee te werken binnen een applicatie waarin een gemiddelde hoeveelheid data moet gepersisteerd worden. Uit het onderzoek is gebleken dat Realm ook een goede keuze is, zoals verwacht.

4.1.3 Profiel 3: Veel data

Tabel 4.3: Samenvatting van alle gegevens over applicatieprofiel 3

	GreenDAO	SQLite	Realm
Gemiddelde opstartsnelheid (in ms)	716,64	323,62	569,26
Gemiddelde CPU-gebruik (in %)	12,11	11,52	10,65
Gemiddelde snelheid van insert (in ms)	750,87	795,90	1105,77

Zoals te zien is op tabel 4.3, scoort SQLite beter bij alle drie de factoren. De combinatie van deze drie factoren, zorgen ervoor dat SQLite aangeraden is voor dit applicatieprofiel.

Voor dit onderzoek uitgevoerd werd, werd gedacht dat Realm de beste keuze zou zijn om mee te werken binnen een applicatie waarin een grote hoeveelheid data moet gepersisteerd worden. Uit het onderzoek is gebleken dat SQLite een goede keuze is.

Bibliografie

AndroidDevelopers (2016a). Cpu monitor. Geraadpleegd op: 17/04/2016.

AndroidDevelopers (2016b). monkeyrunner. Geraadpleegd op: 17/04/2016.

AndroidDevelopers (2016c). Saving data. Geraadpleegd op: 16/04/2016.

AndroidDevelopers (2016d). Saving key-value sets. Geraadpleegd op: 16/04/2016.

GeenDAO (2016). Greendao: the superfast android orm for sqlite. Geraadpleegd: 13/03/2016.

Lee, S. (2012). Creating and using databases for android applications. *International Journal of Database Theory and Application*, 5(2):99–106.

Michael Facemire, Jeffrey S. Hammond, C. M. E. W. (2014). The offline mobile challenge. *Tackling Mobile Apps' Most Important And Difficult Feature*.

Realm (2014). Realm is a mobile database hundreds of millions of people rely on. Geraadpleegd: 13/03/2016.

Realm (2016). Realm java 0.90.1. Geraadpleegd: 13/03/2016.

SQLite (2001). About sqlite. Geraadpleegd: 29/02/2016.

Lijst van figuren

Lijst van tabellen