ORMLight ORM mapper za Android - kratak pregled

Da bi ispravno instalirali biblioteku potrebno je dodati naziv biblioteke u gradle.build fajl:

```
dependencies {
    ...
    compile 'com.j256.ormlite:ormlite-android:4.48'
}
```

Svaku tablu mapiramo na jednu klasu u bazi podataka. Da bi ispravno radilo moramo prvo da anotiramo sve atribute kao i samu klasu:

```
@DatabaseTable(tableName = User.TABLE_NAME_USERS)
public class User {

    public static final String TABLE_NAME_USERS = "users";

    public static final String FIELD_NAME_ID     = "id";
    public static final String FIELD_NAME_NAME   = "name";

    @DatabaseField(columnName = FIELD_NAME_ID, generatedId = true)
    private int mId;

    @DatabaseField(columnName = FIELD_NAME_NAME)
    private String mName;

    public User() {
        // Don't forget the empty constructor, needed by ORMLite.
    }

    /** Getters & Setters **/
    ...

}
```

Nikon definisanja table potrebno je da napravimo Dao objekat za svaku tabelu:

```
public Dao<User, Integer> getUserDao() throws SQLException {
    if (mUserDao == null) {
        mUserDao = getDao(User.class);
    }
    return mUserDao;
}
```

Takođe potrebno je da definišemo klasu koja će rukovati Dao objektima ali i ostvarivati konekciju ka bazi. Ova klasa zadužena je za kreiranje table, baze ali i za update baze:

```
public class DatabaseHelper extends OrmLiteSqliteOpenHelper {
    private static final String DATABASE_NAME    = "ormlite.db";
    private static final int    DATABASE_VERSION = 1;
    private Dao<User, Integer> mUserDao = null;

    public DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db, ConnectionSource connectionSource) {
        try {
            TableUtils.createTable(connectionSource, User.class);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, ConnectionSource connectionSource,
            int oldVersion, int newVersion) {
        try {
            TableUtils.dropTable(connectionSource, User.class, true);
            onCreate(db, connectionSource);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }

    /* User */

    public Dao<User, Integer> getUserDao() throws SQLException {
        if (mUserDao == null) {
            mUserDao = getDao(User.class);
        }
        return mUserDao;
    }

    @Override
    public void close() {
        mUserDao = null;
        super.close();
    }
}
```

Nikon kreiranja odgovarajucih klasa, potrebno je instancirati vezu ka bazi i Dao objekte:
DatabaseHelper helper = new DatabaseHelper(this);

```
Dao<User, Integer> userDao = null;
try {
   userDao = helper.getUserDao();
} catch (SQLException e) {
   e.printStackTrace();
}
```

Kreiranje objekata tj upis u tabelu:

```
User user = new User().setName("Mike");
try {
   userDao.create(user);
} catch (SQLException e) {
   e.printStackTrace();
}
```

Update zaps u bazi:

```
user.setName("Michael");
try {
   userDao.update(user);
} catch (SQLException e) {
   e.printStackTrace();
}
```

Brisanje zapisa iz tabele:

```
try {
   userDao.delete(user);
} catch (SQLException e) {
   e.printStackTrace();
}
```

Prikaz svih elemenata iz jedne tabele:

```
final List<User> users = userDao.queryForAll();
```

Dobijanje podataka o pojedinačnim zapisima:

```
User user = userDao.queryForId(userId);
```

Ako želimo da pišemo kompleksnije upite ili filtriramo sadržaj, na raspolaganju nam je poseba metoda queryBuilder:

```
userDao.queryBuilder()
   .where()
   .eq(User.FIELD_NAME_NAME, "Mike")
   .and() // :or()
   .eq(User.FIELD_NAME_EMAILS, "email@example.com")
   .query();
```

ORMLight podržava i veze između tabela:

One-to-one:

```
@DatabaseTable(tableName = "users")
public class User {

   public static final String FIELD_NAME_ROLE     = "role";
   …

   @DatabaseField(columnName = FIELD_NAME_ROLE, foreign = true, foreignAutoCreate = true,foreignAutoRefresh = true)
   private Role mRole;
   ...

}
```

One-to-many:

```
@DatabaseTable(tableName = User.TABLE_NAME_USERS)
public class User {

   public static final String FIELD_NAME_EMAILS = "emails";
   ...

   // One-to-many
   @ForeignCollectionField(columnName = FIELD_NAME_EMAILS, eager = true)
   private ForeignCollection<Email> mEmails;
   ...

   public ForeignCollection<Email> getEmails() {
      return mEmails;
   }

}
```

```
@DatabaseTable(tableName = Email.TABLE_NAME_EMAIL)
public class Email {

    public static final String TABLE_NAME_EMAIL = "emails";

    public static final String FIELD_NAME_ID    = "id";
    public static final String FIELD_NAME_EMAIL = "email";
    public static final String FIELD_NAME_USER  = "user";

    @DatabaseField(columnName = FIELD_NAME_ID, generatedId = true)
    private int mId;

    @DatabaseField(columnName = FIELD_NAME_EMAIL)
    private String mEmail;

     @DatabaseField(columnName = FIELD_NAME_USER, foreign = true, foreignAutoRefresh =
true)
    private User mUser;

    public Email() {
        // Don't forget the empty constructor, needed by ORMLite.
    }

    /** Getters & Setter **/
    ...

}
```

Many-to-many:

```
@DatabaseTable(tableName = Project.TABLE_NAME_PROJECTS)
public class Project {
    public static final String TABLE_NAME_PROJECTS = "projects";
    public static final String FIELD_NAME_ID   = "id";
    public static final String FIELD_NAME_NAME = "name";

    @DatabaseField(columnName = FIELD_NAME_ID, generatedId = true)
    private int mId;

    @DatabaseField(columnName = FIELD_NAME_NAME)
    private String mName;

    public Project() {
        // Don't forget the empty constructor, needed by ORMLite.
    }

    /** Getters & Setters **/
    ...
}
```

```
@DatabaseTable(tableName = UserProject.TABLE_NAME_USER_PROJECT)
public class UserProject {

    public static final String TABLE_NAME_USER_PROJECT = "user_project";

    public static final String FIELD_NAME_ID        = "id";
    public static final String FIELD_NAME_USER_ID   = "user_id";
    public static final String USER_NAME_PROJECT_ID = "project_id";

    @DatabaseField(columnName = FIELD_NAME_ID, generatedId = true)
    private int mId;

    @DatabaseField(foreign = true, columnName = FIELD_NAME_USER_ID)
    private User mUser;

    @DatabaseField(foreign = true, columnName = USER_NAME_PROJECT_ID)
    private Project mProject;

    public UserProject() {
        // Don't forget the empty constructor, needed by ORMLite.
    }

    /** Getters & Setters **/
    ...

}

@DatabaseTable(tableName = User.TABLE_NAME_USERS)
public class User {
    ...
    @SerializedName("id")
    @DatabaseField(columnName = FIELD_NAME_ID, id = true)
    private int mId;

    @SerializedName("name")
    @DatabaseField(columnName = FIELD_NAME_NAME)
    private String mName;

    ...

}
```

krater pregled komandi može da se pogleda na adresi:
https://www.jayway.com/2016/03/15/android-ormlite/