5. Ветрова Ю.В., Васюткина Д.И., Ковалева Е.Г. Экономическая оценка ущерба от чрезвычайных ситуаций в учреждениях высшего профессионального образования. // «Современный взгляд на будущее науки». Сборник статей Международной научно-практической конференции. Научный центр «Аэтерна». 2014. С. 17-20.
6. Павленко А.В., Ковалева Е.Г., Радоуцкий В.Ю. Анализ подходов к оценке риска. // Вестник Белгородского государственного технологического университета им. В.Г. Шухова. 2015. №3. С. 106-109.
7. Северин Н.Н., Радоуцкий В.Ю., Ковалева Е.Г., Литвин М.В. Общая характеристика системы профессиональной подготовки сотрудников ГПС МЧС России. // Вестник Белгородского государственного технологического университета им. В.Г. Шухова. 2011. №2. С. 179-183.
8. Радоуцкий В.Ю., Шаптала В.Г. Методологические основы моделирования систем обеспечения комплексной безопасности ВУЗов // Вестник БГТУ им. В.Г. Шухова. 2008. №3. С. 64-66.
9. Шаптала В.Г., Радоуцкий В.Ю., Шаптала В.В. Системы информационной поддержки принятия управленческих решений при ликвидации последствий чрезвычайных ситуаций органами управления ВУЗа // Вестник БГТУ им. В.Г. Шухова. 2012. №4. С. 188-191.

УДК 004.9:519.688

Смольянов Андрей Григорьевич
к.ф.-м.н., зав. кафедрой
фундаментальной информатики
ФГБОУ ВПО «МГУ им. Н.П. Огарёва»,
г. Саранск, РФ
E-mail: mgutech@mail.ru
Ивановичев Вячеслав Валерьевич
студент 4 курса факультета математики
и информационных технологий
ФГБОУ ВПО «МГУ им. Н.П. Огарёва»,
г. Саранск, РФ
E-mail:try@svntech.ru

КРАТКИЙ ОБЗОР МОБИЛЬНОЙ СУБД REALM ДЛЯ GOOGLE ANDROID

Аннотация

В статье описывается одна из популярных систем управления базами данных (СУБД), предназначенная для поддержки информационной части мобильных приложений. В статье кратко описываются вводные сведения о СУБД, полезные для разработчика программного обеспечения.

Ключевые слова

Система управления базами данных, класс, объект, ключ, первичный ключ, транзакция, асинхронная транзакция.

Realm - это система управления базами данных (СУБД) для мобильных устройств. Она написана на C++, что делает ее одной из самых быстрых мобильных баз данных. Разработчики изначально ставили перед собой цель создать СУБД, которая должна отвечать следующим требованиям: 1) простота использования; 2) высокая скорость работы; 3) реляционность; 4) кросс-платформенность (Java, Swift & Objective C); 5) поддержка современных функций (шифрование, миграции, поддержка расширений и т. д.).

Большим преимуществом данной СУБД является широкое сообщество разработчиков на GitHub, Stackoverflow, Twitter, в котором можно найти ответы на многие интересующие вопросы.

Основной функционал СУБД будет рассмотрен позже на примере мобильного приложения для Android. Пока кратко обсудим начальные сведения о системе Realm, которые потребуются для решения практической задачи.

Моделью в Realm является класс, наследуемый от базового класса RealmObject. Поддерживаются типы данных полей: boolean, byte, short, ìnt, long, float, double, String, Date, byte[]. Обратим внимание на базовые классы: Boolean, Byte, Short, Integer, Long, Float, Double.

Перечислим аннотации для полей:

- @Required – признак обязательного заполнения поля;
- @Ignored – признак запрещения значения поля;
- @Index – признак поискового индекса для поля; заметим, что вставки реализуются медленнее, файл базы данных увеличивается в объеме, однако запросы реализуются быстрее. Рекомендуется добавлять это поле для быстрого поиска и чтения данных из БД. Для индексных полей поддерживаются типы String, byte, short, int, long, boolean, Date;
- @PrimaryKey - признак первичного ключа для поля; поддерживаются типы int, string, short, long. Использование типа string для поля с признаком первичного ключа подразумевает, что это поле является и индексом.

Пусть в приложении требуется хранить заметки пользователей, т. е. рассмотрим две модели данных: пользователь и заметка. В папке под названием models будем хранить классы моделей Note и User. Первый – класс Note – будет являться моделью самой заметки, второй – класс User – моделью пользователя, создавшего эту заметку. Связь между моделью User и моделью Note будет иметь разновидность «один-ко-многим» (у пользователя может быть много заметок), а связь между Note и User – один-к-одному (у одной заметки – лишь один пользователь).

Связи между сущностями в Realm являются очень эффективными с точки зрения потребления памяти. Realm поддерживает связи: «один-к-одному», «один-ко-многим», «многие-к-одному», «многие-ко-многим».

Так как в языке Java объекты это ссылочные типы, а модели - это те же объекты, в Realm ничто не мешает им ссылаться на один и тот же объект. Для того чтобы удалить связную модель из объекта (на самом деле удалиться лишь ссылка на связный объект), достаточно присвоить null, при этом сам объект в Realm останется без изменений.

Приведем пример модели Note со связью «один-к-одному»:

```
public class Note extends RealmObject {
  @PrimaryKey
  private int id;       //Индекс
  private String title; //Заголовок заметки
  private Date date;    //Дата создания
  private User user;    //Пользователь, добавивший заметку
  public int getId() {
     return id;
  }
  public void setId(int id) {
     this.id = id;
  }
  public String getTitle() {
     return title;
  }
  public void setTitle(String title) {
     this.title = title
  public Date getDate() {
     return date;
  }
```

```
  public void setDate(Date date) {
    this.date = date;
  }
  public User getUser() {
    return user;
  }
  public void setUser(User user) {
    this.user = user;
  }
}
```

В случае связи «многие-ко-многим» ссылаемый объект будет содержать поле типа RealmList<Object>, который не отличается от Java-класса List.

Приведем пример модели User со связью типа «один-ко-многим»:

```
import io.realm.RealmList;
import io.realm.RealmObject;
import io.realm.annotations.PrimaryKey;

public class User extends RealmObject {
  @PrimaryKey
  private String name;          // Имя  уникальное
  private RealmList<Note> notes; //Связные заметки

  //геттеры и сеттеры
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
  public RealmList<Note> getNotes() {
    return this.notes;
  }
}
```

Для создания заметок следует изучить механизм добавления данных в базу, а также нужно добавить несколько пользователей для авторизации в приложении. У каждого пользователя, очевидно, будет свой набор заметок.

Получить экземпляр класса Realm просто. Для этого достаточен код:

```
Realm realm = Realm.getInstance(this)
```

При этом в конструктор класса передается контекст созданного приложения.

Создание и добавление пользователей можно реализовать двумя способами:

- непосредственно из Realm;
- отдельно от Realm создать объект, а затем, используя метод copyToRealm(), создать объект в базе данных.

Приведем примеры для обоих способов.

Пример. Способ-1.

```
realm.beginTransaction();
User user = realm.createObject(User.class);
user.setName("Александр");
```

```
User user2 = realm.createObject(User.class);
user2.setName("Вячеслав");
User user3 = realm.createObject(User.class);
user3.setName("Михаил");
realm.commitTransaction();
```

Пример. Способ-2.

```
User user = new User("Александр");
User user2 = new User("Вячеслав");
User user3 = new User("Михаил");
realm.beginTransaction();
User realmUser1 = realm.copyToRealm(user);
realm.commitTransaction();
```

Важно заметить, что СУБД использует механизм транзакций. Операцию записи нужно начинать с команды realm.beginTransaction(), а завершить командой realm.commitTransaction(). В случае неудачи следует вызвать функцию realm.cancelTransaction().

Используя метод realm.createObject(Note.class), нужно иметь в виду, что при создании объекта Realm присваивает полям значения по умолчанию. Эти значения соответствуют значениям по умолчанию в Java, т. е. для boolean - false, для int - 0 и т. д.

На практике удобно использовать метод realm.executeTransaction(), который автоматически будет завершать транзакцию при наличии ошибок, либо при успешном выполнении.

```
realm.executeTransaction(new Realm.Transaction() {
  @Override
  public void execute(Realm realm) {
            User user = realm.createObject(User.class);
            user.setName("Александр");
User user2 = realm.createObject(User.class);
  user2.setName("Вячеслав");
User user3 = realm.createObject(User.class);
  user3.setName("Михаил");
  }
});
```

Если запись в базу данных будет занимать длительное время, то UI поток приложения может «зависнуть». Чтобы этого не случилось в Realm, существуют асинхронные транзакции, которые выносят выполнение добавления данных в отдельный поток, который будет работать в фоновом режиме и при необходимости уведомит пользователя о завершении работы транзакции:

```
realm.executeTransaction(new Realm.Transaction() {
@Override
public void execute(Realm bgRealm) {
        User user = realm.createObject(User.class);
             user.setName("Александр");
         User user2 = realm.createObject(User.class);
              user2.setName("Вячеслав");
         User user3 = realm.createObject(User.class);
              user3.setName("Михаил");
        }
        }, new Realm.Transaction.Callback() {// Данный код вызовется при удачном или неудачном
завершении транзакции
         @Override
         public void onSuccess() {
         }
```

```
        @Override
        public void onError(Exception e) {
                }
    });
```

Для остановки транзакции воспользуемся кодом:

```
    public void onStop () {
    super.onStop();
  if (transaction != null && !transaction.isCancelled()) {
    transaction.cancel();
  }
    }
```

При завершении работы с Activity нужно остановить выполнение транзакции и этот код окажется очень важным: выход из Activity во время выполнения асинхронной транзакции с сообщением в UI поток может привести к вызову исключения и аварийному завершению приложения.

Пример добавления пользователей в методе onCreate(), созданного Activity. Сначала проверяется наличие записей в базе с помощью запроса double userCount = realm.where(User.class).count(), если количество равно 0, то записываем пользователей и их заметки в Realm.

```
    Realm realm;
    RealmAsyncTask transaction;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      realm = Realm.getInstance(this);
      double userCount = realm.where(User.class).count();
      if (userCount == 0) {
        transaction = realm.executeTransaction(new Realm.Transaction() {
          @Override
          public void execute(Realm bgRealm) {
            Realm realm =
                Realm.getInstance(getApplicationContext());
            //Создаем пользователей
            User alex = realm.createObject(User.class);
            alex.setName("Александр");
            User vyacheslav = realm.createObject(User.class);
            vyacheslav.setName("Вячеслав");
            User michael = realm.createObject(User.class);
            michael.setName("Михаил");

            //Создаем заметки Александра
            RealmList<Note> alexNotes = new RealmList<Note>();
            Note note1 = realm.createObject(Note.class);
            note1.setId(1);
            note1.setTitle("Заметка 1 Александра");
            note1.setDate(new Date());
            note1.setUser(alex);
            alexNotes.add(note1);

            Note note2 = realm.createObject(Note.class);
            note2.setId(2);
            note2.setTitle("Заметка 2 Александра");
```

```
        note2.setDate(new Date());
        note2.setUser(alex);
        alexNotes.add(note2);
        alex.setNotes(alexNotes);

        //Создаем заметки Вячеслава
        RealmList<Note> sNotes = new RealmList<Note>();
        note1 = realm.createObject(Note.class);
        note1.setId(3);
        note1.setTitle("Заметка 1 Вячеслава");
        note1.setDate(new Date());
        note1.setUser(vyacheslav);
        sNotes.add(note1);

        note2 = realm.createObject(Note.class);
        note2.setId(4);
        note2.setTitle("Заметка 2 Вячеслава");
        note2.setDate(new Date());
        note2.setUser(vyacheslav);
        sNotes.add(note2);

        vyacheslav.setNotes(sNotes);

        //Создаем заметку с автором Михаил
        Note note = new Note();
        note.setId(5);
        note.setDate(new Date());
        note.setUser(michael);
        note.setTitle("Заметка 1 Михаила");
        realm.copyToRealm(note);
    }
}, new Realm.Transaction.Callback() {
    @Override
    public void onSuccess() {

        Toast.makeText(getApplicationContext(),
            "Данные успешно добавлены",
            Toast.LENGTH_LONG).show();
    }
    @Override
    public void onError(Exception e) {
        e.printStackTrace();
        Toast.makeText(getApplicationContext(),
        "Не удалось добавить данные",
        Toast.LENGTH_LONG).show();
    }
});
```