Realm

MADE BY 한상엽

데이터베이스 기초

왜 데이터 베이스를 사용하는가??

앱이 종료될 때 데이터가 사라지는 것을 방지하기 위해 영구적인 보관이 필요하기 때문!

보관방법

1. SharedPreferences -> 앱의 설정정보 같이 간단한 정보 저장
2. 파일 입출력 api를 통해 기기 저장소에 직접 데이터 읽고 쓰기
3. SQL을 사용해 로컬 데이터베이스에 저장
4. 인터넷 서버에 저장

등등

데이터베이스 기초

가장 범용적이고 보편적인 방법

-> 데이터베이스를 사용

장점

- SharedPreferecnes 의 key-value 저장 형식 한계점 극복
- 파일 입출력 저장형식 설계 복잡성 해소
- 서버 구축 비용 절감

Realm 이란??

사전적 정의 : 영역, 왕국 등

발음 : 렐름이라고 부릅시다

Realm 이라는 객체 컨테이너
데이터베이스에서처럼 쿼리, 필터링, 상호 연결 관계지정 가능

기존의 데이터베이스는 객체를 복사하고 수정한 다음 그걸 다시 저장하는데
Realm은 실제 객체를 수정함

Realm

왜 Realm을 사용해야 하나요?

쉬운 사용

Realm은 SQLite을 기반으로 한 ORM이 *아닙니다*. 대신에 Realm은 쉬운 사용성과 속도를 위해 개발한 persistence 엔진을 사용합니다. 사용자 페이지에서 Realm을 시작하는데 몇 분이면 충분하고 몇 시간만에 앱을 새로 작성하고 몇 주의 업무 시간을 줄인 이야기를 확인할 수 있습니다.

크로스 플랫폼

Realm은 iOS와 Android 모두 지원합니다. Realm 파일은 플랫폼 간에 호환이 가능하고 Java, Swift, Objective-C에서 동일한 상위 모델을 사용할 수 있으며 두 플랫폼에서 유사한 비즈니스 로직을 사용할 수 있습니다.

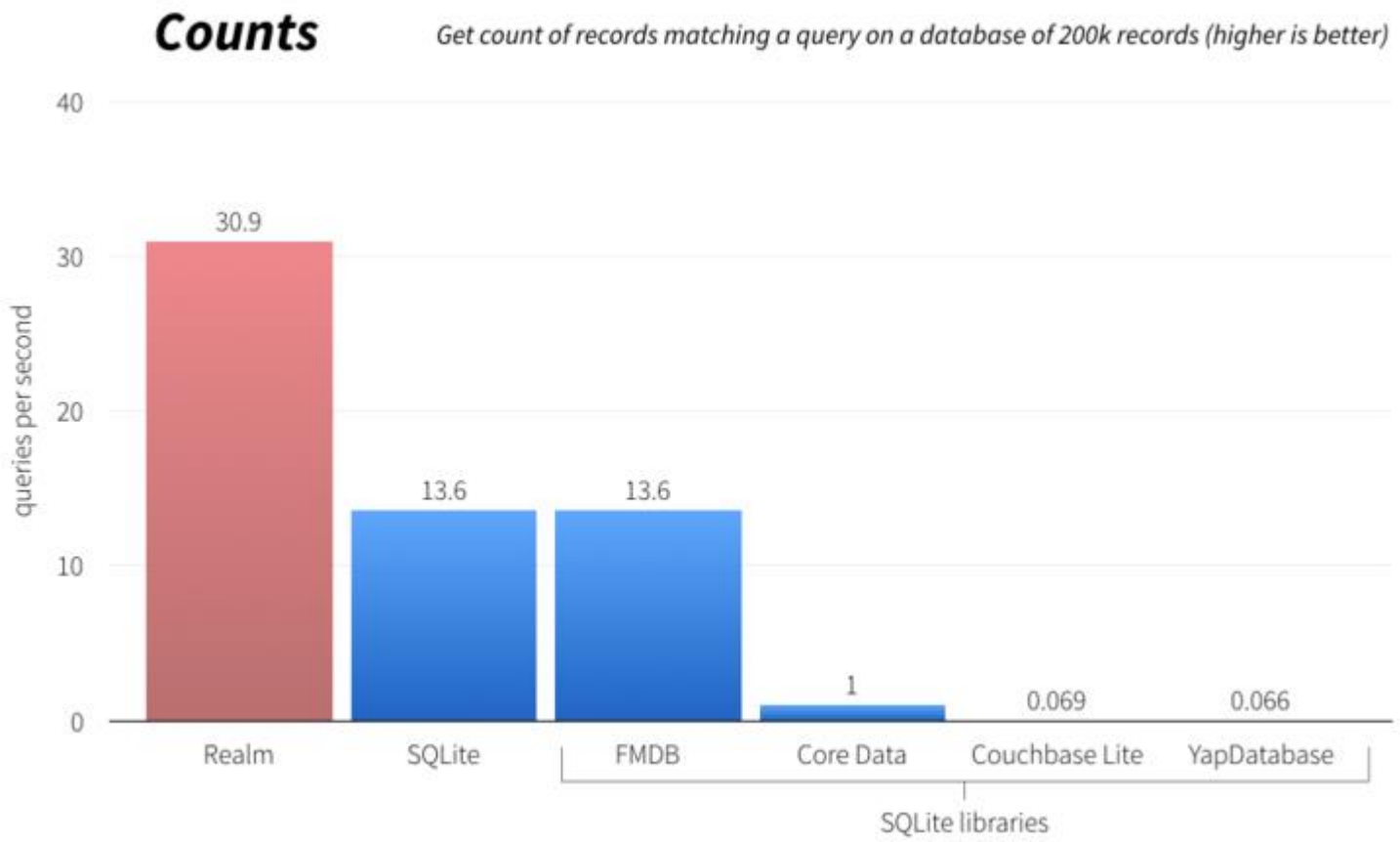
빠른 속도

zero-copy 디자인 덕분에 Realm은 SQLite를 기반으로 ORM을 구현했을 때보다 월등히 빠릅니다. iOS, Android 벤치마크를 확인하고 Twitter에서 Realm에 관련된 사용자들의 트윗을 확인해보세요.

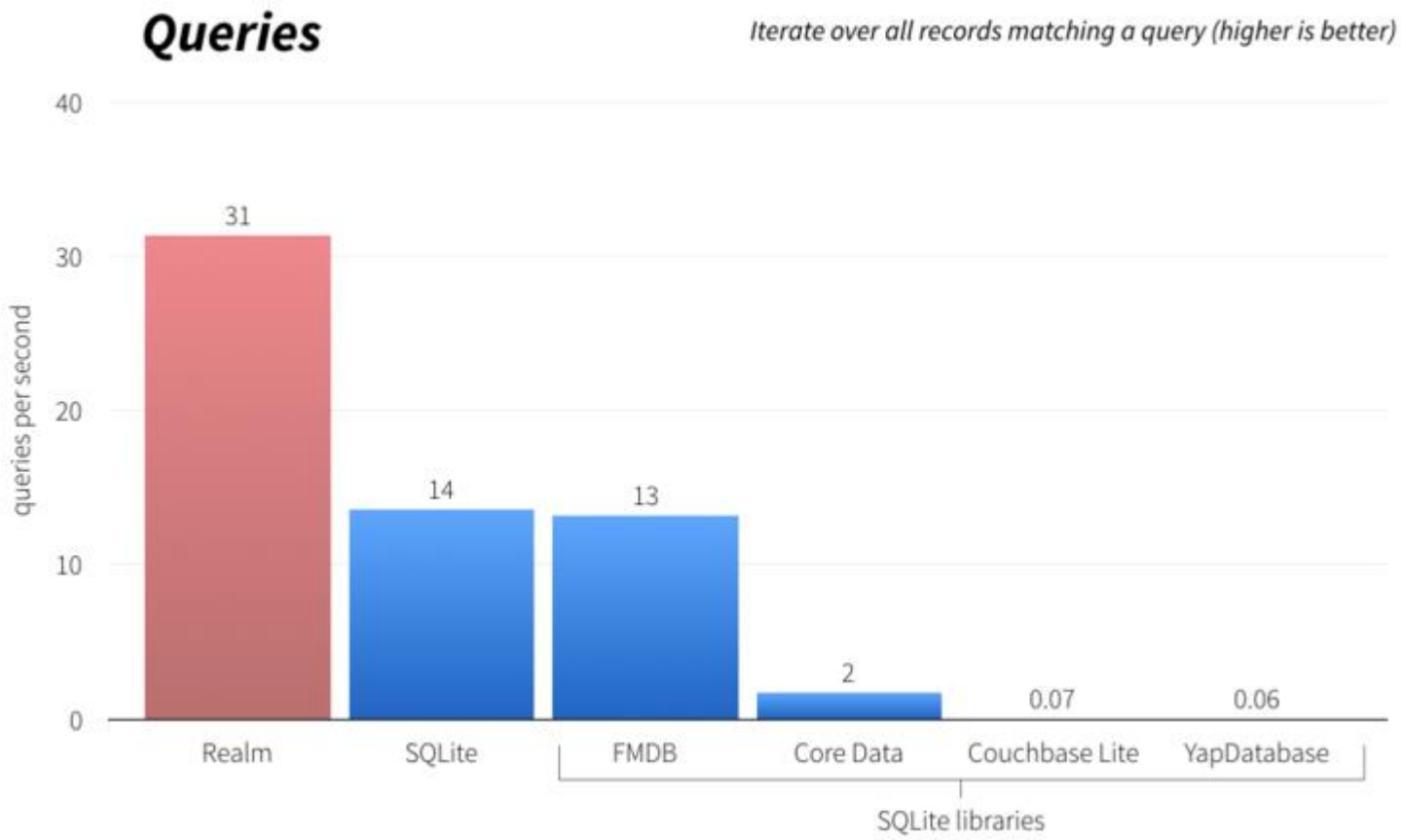
지원

Realm을 만드는 개발자로부터 빠르고 공식적인 답변을 얻을 수 있습니다. 사용자 그룹, 이메일, Github, StackOverflow, Twitter에서 질문을 받고 있습니다.

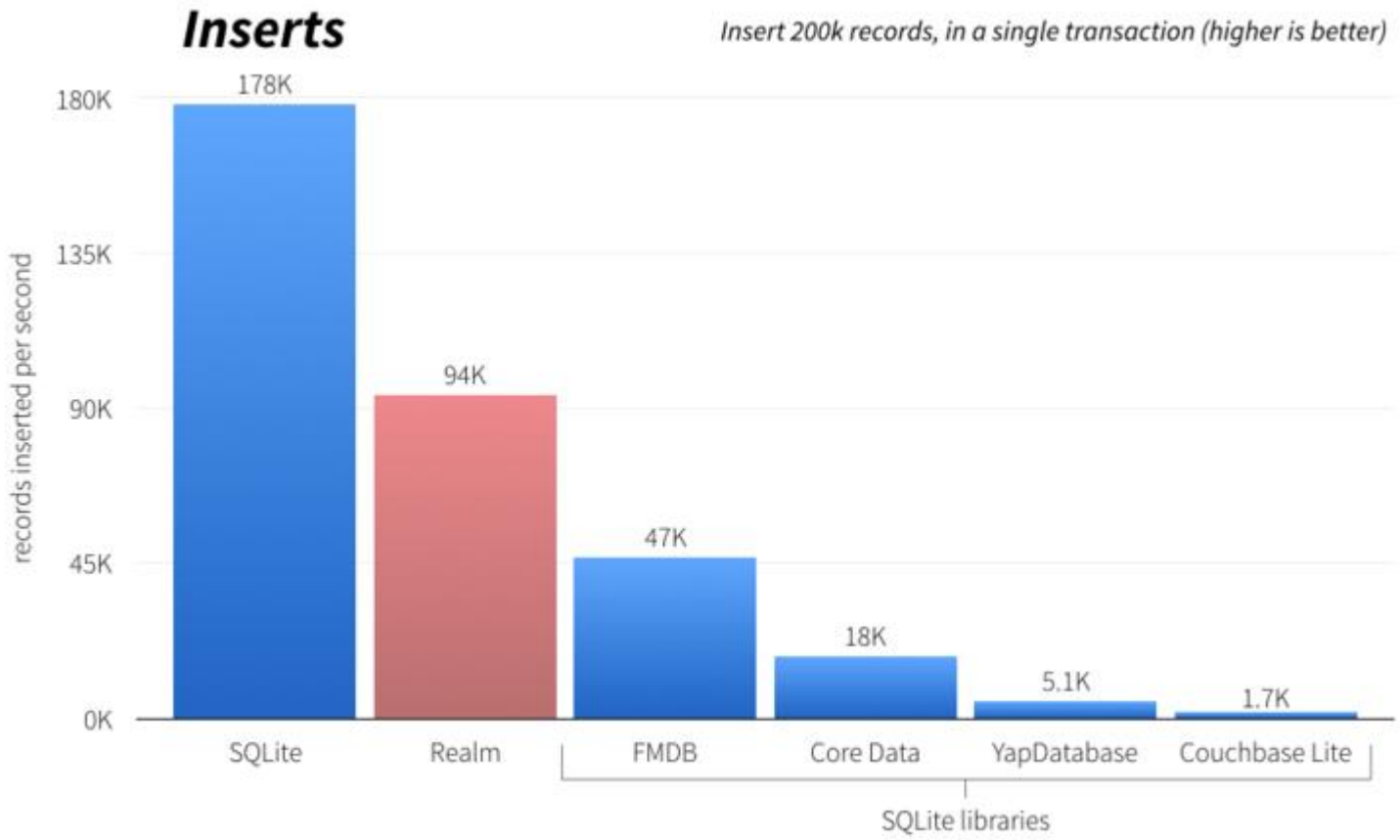
Realm


Counts
Get count of records matching a query on a database of 200k records (higher is better)
queries per second
40
30
20
10
0
30.9
13.6
13.6
1
0.069
0.066
Realm
SQLite
FMDB
Core Data
Couchbase Lite
YapDatabase
SQLite libraries

Realm


Queries
Iterate over all records matching a query (higher is better)
queries per second
40
30
20
10
0
31
14
13
2
0.07
0.06
Realm
SQLite
FMDB
Core Data
Couchbase Lite
YapDatabase
SQLite libraries

Realm


Inserts
Insert 200k records, in a single transaction (higher is better)
records inserted per second
180K
135K
90K
45K
0K
178K
94K
47K
18K
5.1K
1.7K
SQLite
Realm
FMDB
Core Data
YapDatabase
Couchbase Lite
SQLite libraries

SQL(Structured Query Language)

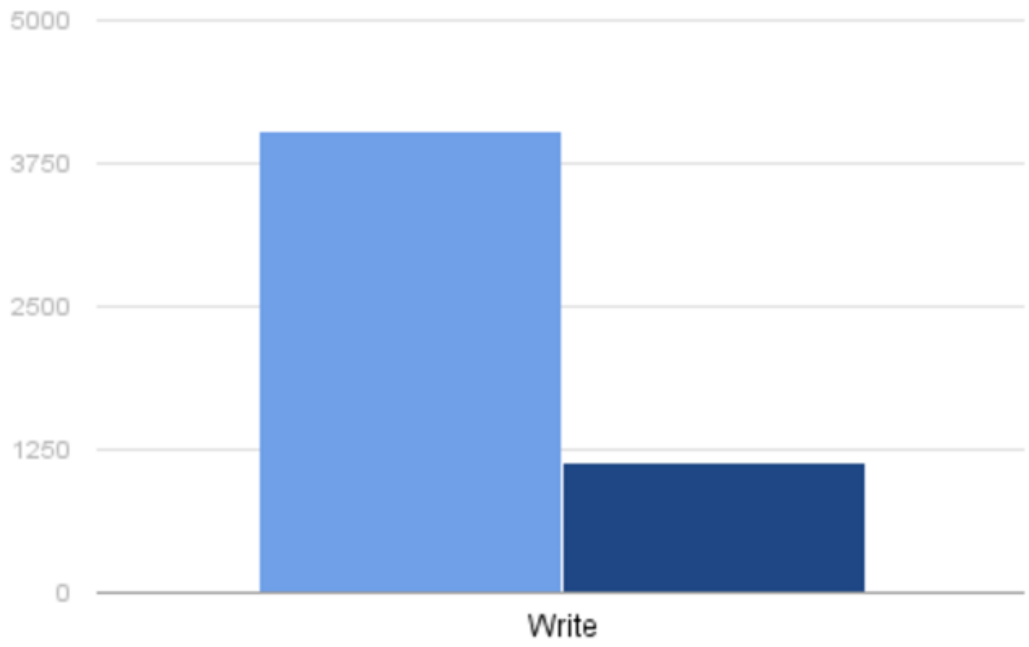
구조화 질의어

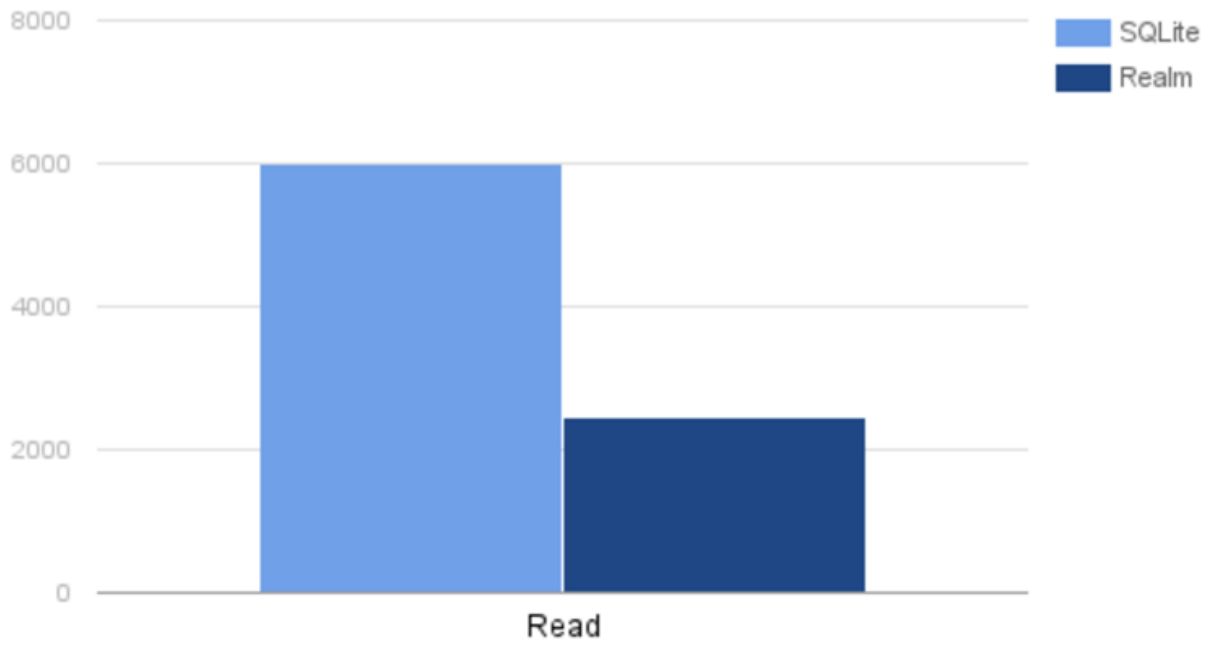
관계형 데이터베이스 관리 시스템의 데이터를 관리하기 위해 설계된 특수 목적의 프로그래밍 언어

Query : 질의, 데이터베이스에 내가 원하는 데이터가 있는지 물어보는 것.

Realm vs. SQLite

Write 20회, Read 20회 수행하는데 필요한 시간





Realm vs. SQLite

Write 20회, Read 20회 수행하는데 필요한 시간

	SQLite	Realm	성능 향상
Write	4039ms	1142ms	3.5x
Read	6010ms	2450ms	2.5x

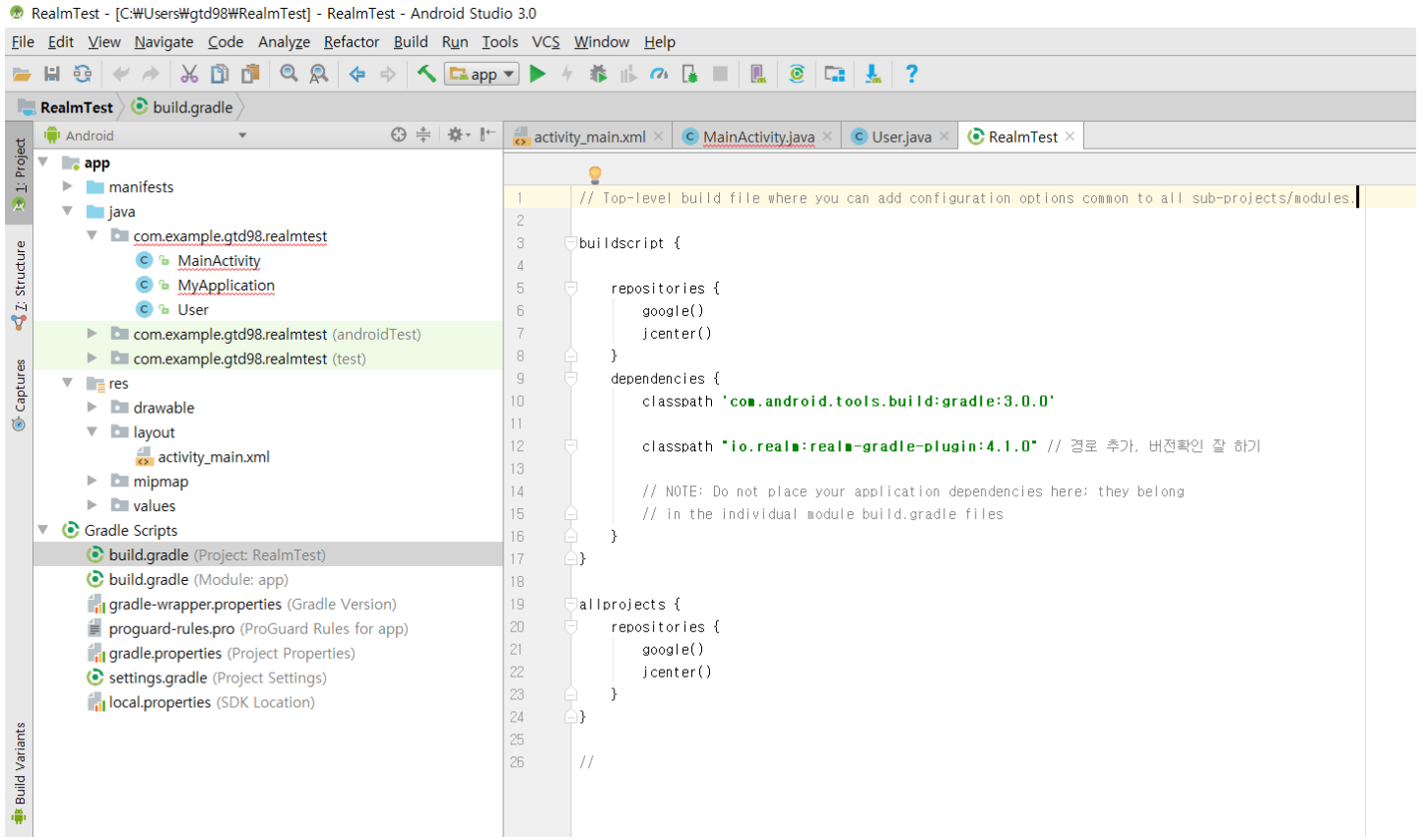
11.	11.	↓ 10.	SQLite	Relational DBMS	112.76	+0.77	+0.76
56.	↑ 57.	↑ 59.	Realm	Relational DBMS	4.53	+0.18	+1.24

Realm을 사용해보자!

환경 세팅

Project build.gradle

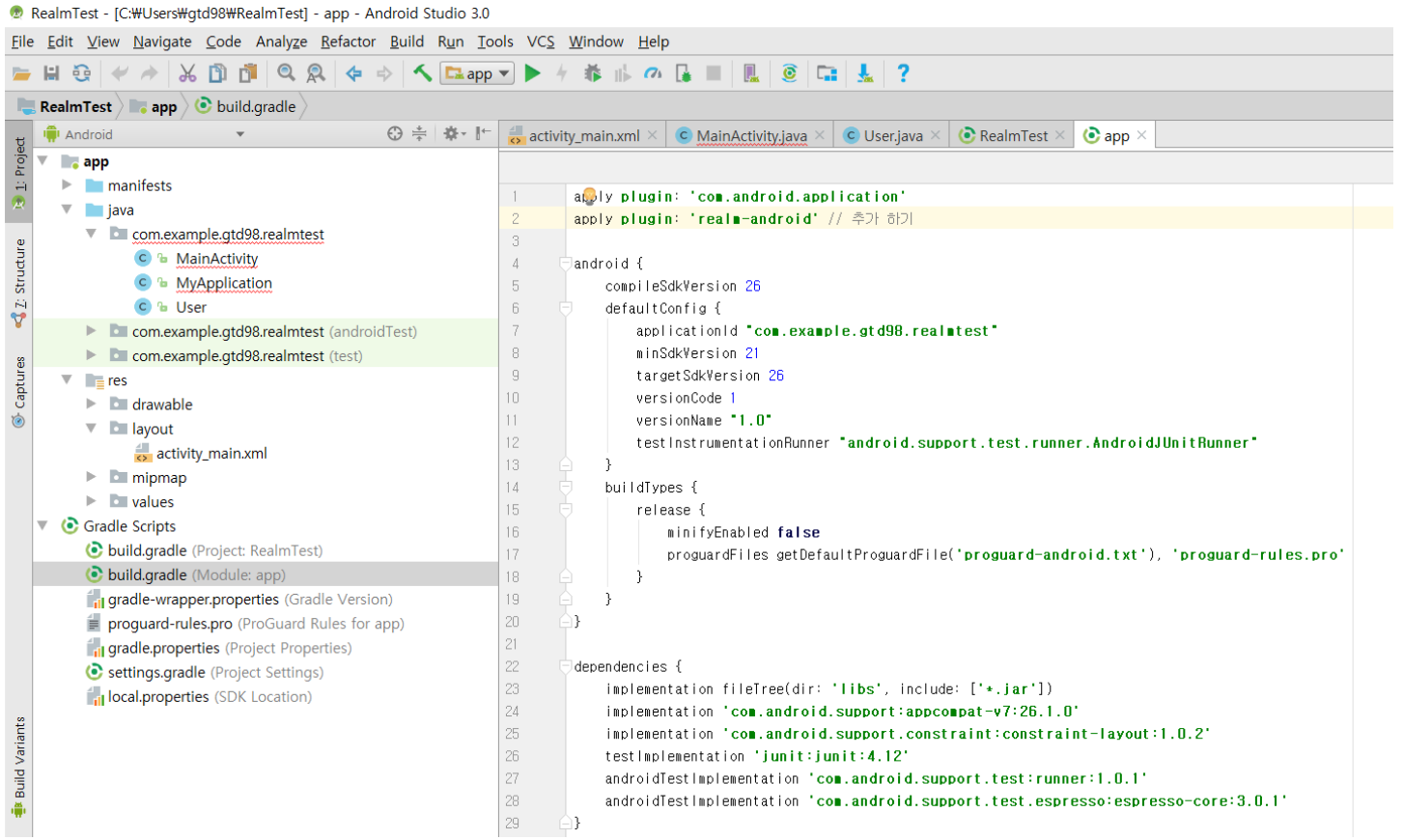
classpath 추가해주기



환경 세팅

app build.gradle에

apply plugin 추가



RealmObject를 상속받는 모델

RealmObject를 상속받는 모델을 만들어 주면 이 모델의 형식으로 Realm에 추가 가능하다

```
public class MemberTest extends RealmObject {
    @PrimaryKey     //고유값
    private int id;
    @Required       // Null 값을 가질 수 없다.
    private String name;
    @Index          // 검색 편함
    private int age;

    //getter 와 setter들
    public int getId() {
        return id;
    }
```

Android Annotation

“@”

ex. @Override

Annotation을 쓰는 이유 :

긴 코드설명을 간략하게 표시 가능

메서드나 멤버 변수를 사용할 때 필요한 규약을 걸어놓을 수 있음 -> 이대로 안 쓰면 컴파일 에러남

Realm Annotation

@Ignore : 필드가 디스크에 저장되지 않게 한다. 예를 들어 사용자의 입력이 모델의 필드보다 많을 때, 필요 이상의 데이터필드를 만들지 않는다.

@PrimaryKey : 쉽게 말해 고유값. 하나의 데이터 필드는 유일한 PrimaryKey를 가진다.

@Index : 검색을 용이하게 해줌.

@Required : Null 값을 가지지 못하게 함. RealmList 형은 암묵적으로 Required 취급한다.

데이터 쓰기

데이터 쓰기에서 읽기 오퍼레이션은 암묵적으로 진행됨

모든 쓰기 업무(추가, 수정, 삭제) 반드시 트랜젝션 안에서 진행되어야함

Transaction : 업무

RealmTransaction은 종류가 3가지

- realm.beginTransation()
- realm.commitTransaction()
- realm.cancelTransaction()

Transaction()

beginTransaction과 commitTransaction 사이에서 객체를 추가하거나 갱신

쓰기 트랜잭션은 스레드 안전을 보장하기 위해 사용합니다.

```
// Realm 인스턴스를 얻습니다
Realm realm = Realm.getDefaultInstance();

realm.beginTransaction();

//... 객체를 여기에서 추가하거나 갱신합니다 ...

realm.commitTransaction();
```

Transaction()

cancelTransaction() 있으면 커밋하지 않고 쓰기를 취소한다.

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

// ...

realm.cancelTransaction();
```

Transaction()

트랜잭션 중 Exception을 만나면 Realm 자체가 오류가 발생하면 그 트랜잭션의 데이터만 잃게 된다.

Exception을 잡고 앱을 실행하기 위해 트랜잭션을 취소하는 것이 중요하다.

executeTransaction() -> 이용하면 자동으로 해준다.

트랜잭션 블록

수동으로 `realm.beginTransaction()`, `realm.commitTransaction()`, `realm.cancelTransaction()` 을 관리하는 대신에 자동으로 begin/commit을 관리하고 에러가 발생했을 때 cancel 하도록 지원하는 realm.executeTransaction 메서드를 사용할 수 있습니다.

```
realm.executeTransaction(new Realm.Transaction() {
        @Override
        public void execute(Realm realm) {
                User user = realm.createObject(User.class);
                user.setName("John");
                user.setEmail("john@corporation.com");
        }
});
```

데이터 읽기

Query로 진행

John이나 Peter라는 이름을 가진 모든 사용자를 질의하기 위해서 다음과 같이합니다.

```
// Build the query looking at all users:
RealmQuery<User> query = realm.where(User.class);

// 질의 조건을 추가합니다
query.equalTo("name", "John");
query.or().equalTo("name", "Peter");

// 질의를 수행합니다
RealmResults<User> result1 = query.findAll();

// 같은 일들을 한 번에 합니다 ("Fluent interface"):
RealmResults<User> result2 = realm.where(User.class)
                                  .equalTo("name", "John")
                                  .or()
                                  .equalTo("name", "Peter")
                                  .findAll();
```

필터링

where() 메서드로 시작