29장. 룸

29.1 룸이란?

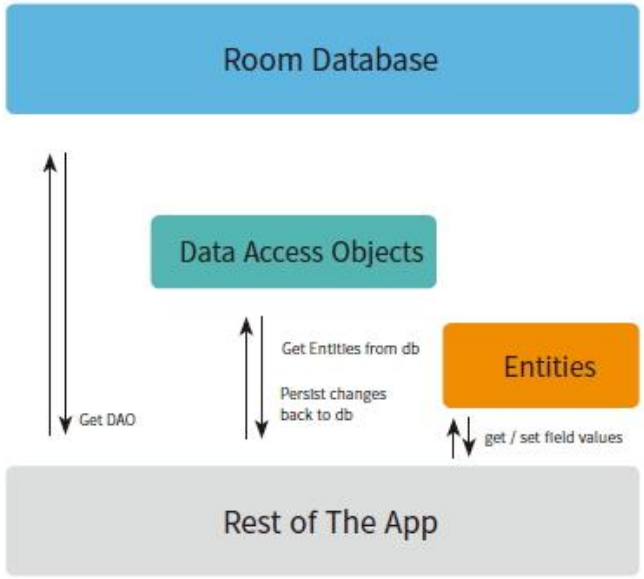
29.1.1. 룸 아키텍처

룸은 SQLite 추상화 라이브러리
SQLite를 이용하는 프로그램을 조금 더 추상화해서 여러 가지 도움을 주기 위한 라이브러리

Entity
데이터 구조를 표현하기 위한 클래스. DBMS에 이용되기 위한 데이터를 위한 클래스.
- @Entiry 어노테이션으로 표현되는 클래스
- 클래스 내에 @PrimaryKey, @ColumnInfo 등의 어노테이션으로 변수 선언

DAO
실제 DBMS를 위해 호출되는 함수를 선언하는 인터페이스나 추상 클래스.
인터페이스나 추상 클래스를 구현해 DBMS를 수행하는 코드는 자동으로 만 들어진다.
- @DAO 어노테이션으로 선언
- @Query, @Insert, @Delete 등의 어노테이션으로 함수 선언

Database
데이터베이스 이용을 위한 DAO 객체 획득 함수를 제공하는 클래스.
DAO 획득 함수는 추상 함 수로 정의하며, 데이터베이스를 이용하기 위해 가장 먼저 호출.
- @Database 어노테이션으로 만드는 클래스
- 추상 클래스로 작성
- Entity를 어노테이션 매개변수로 지정



29.1 룸이란?

29.1.2. 룸의 기초 이용법

```
implementation "android.arch.persistence.room:runtime:1.1.1"
annotationProcessor "android.arch.persistence.room:compiler:1.1.1"
// optional - RxJava support for Room
implementation "android.arch.persistence.room:rxjava2:1.1.1"
// optional - Guava support for Room
implementation "android.arch.persistence.room:guava:1.1.1"
implementation 'com.google.code.gson:gson:2.8.2'
```

29.1 룸이란?

Entiry 클래스 정의

```
@Entity
public class User {
@PrimaryKey
private int uid;
@ColumnInfo(name = "first_name")
private String firstName;
@ColumnInfo(name = "last_name")
private String lastName;
public User(int uid, String firstName, String lastName){
this.uid = uid;
this.firstName = firstName;
this.lastName = lastName;
}
public int getUid() { return uid; }
public void setUid(int uid) { this.uid = uid; }
public String getFirstName() { return firstName; }
public void setFirstName(String firstName) { this.firstName = firstName;}
public String getLastName() { return lastName; }
public void setLastName(String lastName) { this.lastName = lastName;}
}
```

29.1 룸이란?

DAO 정의

```
@Dao
public interface UserDAO {
@Query("SELECT * FROM user")
List<User> getAll();
@Insert
void insertAll(User... users);
}
```

Database 클래스

```
@Database(entities = {User.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {
public abstract UserDAO userDao();
}
```

29.1 룸이란?

Room 이용

```
public class MainActivity extends AppCompatActivity {
UserDAO dao;
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
AppDatabase db = Room.databaseBuilder(getApplicationContext(),
AppDatabase.class, "database-name").build();

dao = db.userDao();
……
}

class InsertThread extends Thread {
……
@Override
public void run() {
dao.insertAll(users);
}
}

class SelectAllThread extends Thread {
……
List<User> results = dao.getAll();
for(User user: results){
Log.d("kkang", user.getUid() + "," + user.getFirstName() + "," + user.getLastName());
}
}
```

29.2 룸 엔티티

29.2.1. Entity 정의

Entity는 DBMS에 이용되는 데이터의 구조를 표현하기 위한 클래스

```
@Entity
public class User2 {
@PrimaryKey
public int id;
public String firstName;
public String lastName;
@Ignore
Bitmap picture;

public User2(int id, String firstName, String lastName){
this.id=id;
this.firstName = firstName;
this.lastName = lastName;
}
}
```

29.2 룸 엔티티

- Entity 클래스는 @Entity 어노테이션으로 선언한다.
- Entity 클래스의 데이터를 저장하기 위한 테이블이 자동으로 만들어진다.
- 테이블 이름이 곧 클래스명이며 대소문자는 무시된다.
- Entity 클래스 내의 변수에 해당하는 칼럼(column)이 만들어진다.
- 기본으로 변수명과 같은 이름의 칼럼과 대응하며 대소문자를 구분한다.
- @PrimaryKey 어노테이션으로 기본 키(Primary Key)를 지정할 수 있다.
- 변수 중 데이터베이스와 상관없이 이용하려는 변수는@ Ignore 어노테이션을 추가한다.
- 생성자는 자유롭게 추가할 수 있다.
- 데이터베이스에 대응하는 변수는 public이나 private으로 선언하면 게터/세터 함수를 추가해야 한다.

@PrimaryKey
Entity 클래스의 식별자 변수에 선언되는 어노테이션

```
@PrimaryKey(autoGenerate = true)
public int id;
```

```
@Entity(primaryKeys = {"firstName", "lastName"})
public class User2 {
@NonNull
public String firstName;
@NonNull
public String lastName;
}
```

29.2 룸 엔티티

@Entity(tableName = "users")
Entity에 대응하는 테이블명은 기본으로 클래스명을 따르지만 @ Entity 어노테이션에 tableName 속성으로 테이블명을 지정하여 변경가능.

@ColumnInfo(name = "first_name")
테이블의 칼럼명은 기본으로 Entity 클래스의 변수명을 따르지만 원한다면 @ColumnInfo(name = "first_name")을 이용하여 변수명과 다른 칼럼명을 지정

@Entiry(indices = { })
indices 속성으로 @Entity에 인덱스 정보를 설정

```
@Entity(
tableName = "tb_user",
indices = {
@Index("name"),
@Index(value = {"last_name", "address"}, unique = true)})
```

29.2 룸 엔티티

29.2.2. 관계 설정
룸에서는 foreignKeys 속성을 이용해 다른 Entiry 클래스와의 관계를 설정

```
@Entity(foreignKeys = @ForeignKey(entity = User2.class,
parentColumns = "id",
childColumns = "user_id"))
public class Book {
@PrimaryKey
public int bookId;
public String title;
@ColumnInfo(name = "user_id")
public int userId;
}
```

부모 테이블의 데이터가 삭제되거나 수정되었을 때 자식 테이블의 데이터가 어떻게 되어야 하는지 를 명시
@ ForeignKey (onDelete=CASCADE)로 설정

- CASCADE: 부모 행이 삭제되거나 수정되면 관련된 모든 자식 행을 삭제하거나 수정한다.
- NO_ACTION: 부모 키가 삭제되거나 수정되었더라도 자식에는 아무 일도 발생하지 않는다 .
- RESTRICT: 만약 자식 쪽에 부모 키로 매핑된 데이터가 있다면 부모의 삭제 및 수정을 금지한다.
- SET_DEFAULT: 부모 쪽에서 삭제되거나 수정되면 자식 키 칼럼을 디폴트 값으로 설정한다.
- SET_NULL: 부조 쪽이 삭제되거나 수정되면 자식 키 칼럼을n ull로 설정한다.

29.2 룸 엔티티

자식 부분이 추가되거나 수정될 수 있는데 이때 추가되거나 수정되는 데이터가 부모 테이블의 칼럼 데이 터와 맞지 않아서 발생하는 충돌 문제를 어떻게 처리할 것인가의 설정
@Insert (onConflict = REPLACE)처럼 함수의 어노테이션을 설정

- ABORT: 트랜잭션 처리를 중단한다.
- FAIL: 트랜잭션을 실패(fail) 처리한다.
- IGNORE: 충돌을 무시하고 계속 진행한다.
- REPLACE: 변경 시 충돌이 나면 과거 데이터를 유지하고 계속 진행한다 .
- ROLLBACK: 트랜잭션을 롤백(rollback)한다.

@Embedded
두 Entity 클래스의 데이터가 하 나의 테이블에 모두 저장되게 하고 싶을 때

```
@Entity(tableName = "tb_user")
public class User3 {
…
@Embedded
public Address address;
}
```

29.2 룸 엔티티

29.2.3. 컬렉션 타입 데이터 처리
@TypeConverter 어노테이션을 이용 컬렉션 타입의 데이터 처리 클래스 명시

```
@Entity(tableName = "tb_user")
public class User3 {
…
@TypeConverters(Converters.class)
public ArrayList<Book3> books;
}
```

```
public class Converters {
@TypeConverter
public static ArrayList<Book3> fromString(String value) {
Type listType = new TypeToken<ArrayList<Book3>>() {}.getType();
return new Gson().fromJson(value, listType);
}

@TypeConverter
public static String fromArrayList(ArrayList<Book3> list) {
Gson gson = new Gson();
String json = gson.toJson(list);
return json;
}
}
```

29.3 DAOs

29.3.1. DAO 메서드 정의
DAO 클래스는 인터페이스나 추상 클래스로 작성하며 추상 함수에 정의된 어노테이션 정보를 보고 실제 데이 터베이스에 접근하는 클래스를 자동으로 만들어 주는 구조

삽입(Insert)
데이터베이스에 데이터를 저장하는 함수는 @Insert 어노테이션으로 정의

```
@Dao
public interface UserDAO4 {
@Insert
long insertUser(User4 uses);
@Insert
public void insertBothUsers(User4 user1, User4 user2);
@Insert
public long[] insertUsersList(List<User4> users);
}
```

29.3 DAOs

갱신(update)
데이터베이스에 데이터를 갱신하는 함수는 @Update 어노테이션으로 정의

```
@Update
public void updateUsers(User4... users);
```

삭제(delete)
데이터베이스에 데이터를 삭제하는 함수는 @Delete 어노테이션으로 정의

```
@Delete
public void deleteUsers(User4... users);
```

29.3 DAOs

29.3.2. 질의문을 이용한 데이터 조회
@Query 어노테이션에는 질의문을 직접 작성
주로 select 문을 담기 위해 이용되지만, insert, update 등의 다른 질의문 가능

```
@Dao
public interface UserDAO {
@Query("SELECT * FROM user5")
List<User5> getAll();
@Query("UPDATE user5 SET firstName = :value WHERE id = :id")
int updateName(String value, int id);
}
```

29.4 데이터베이스 스키마 변경

스키마를 변경하려면 우선 AppDatabase 클래스의 어노테이션 정보에 version 값을 변경
변경될 때 무 엇을 할지를 개발자가 직접 정의
Migration 클래스를 상속받아 migrate ( ) 함수를 재정의한 클래스를 준비

```
@Database(entities = {User6.class}, version = 2)
public abstract class AppDatabase6 extends RoomDatabase {

static final Migration MIGRATION_1_2 = new Migration(1, 2) {
@Override
public void migrate(SupportSQLiteDatabase database) {
database.execSQL("ALTER TABLE user6 "
+ " ADD COLUMN address TEXT");
}
};
public abstract UserDAO6 userDao();
}
```

```
AppDatabase6 db = Room.databaseBuilder(getApplicationContext(),
AppDatabase6.class, "test1")
.addMigrations(MIGRATION_1_2)
.build();
```

Step by Step 실습 – 29-1. 룸

- 그레이들 작업
- AndroidManifest.xml
- ItemModel.java
- AppDatabase.java
- ArticleDAO.java
- MyViewModel.java