
Room
-KiniKu-
KodeV
Koding Akademi

Content

- Room

Room

- Dalam pembuatan aplikasi, kita akan sering bermain dengan database dan mengelolanya.
- Room menyediakan lapisan abstraksi di atas SQLite. Ini memungkinkan kita untuk mengakses database dengan lancar dan juga mampu memanfaatkan kekuatan dari SQLite secara maksimal.

Komponen ROOM

- **Database**: Berisi pemegang database. Berfungsi sebagai titik akses utama untuk melakukan koneksi ke database dari aplikasi. Sebuah kelas yang akan diberi anotasi dengan @Database harus memenuhi ketentuan berikut:
  - Harus menjadi kelas abstrak yang diberi turunan kelas RoomDatabase.
  - Sertakan daftar entitas yang berkaitan dengan database dalam anotasi.
  - Berisi metode abstrak yang memiliki 0 argumen dan mengembalikan kelas @Dao.

- **Entity**: Mempresentasikan tabel yang ada pada database Anda.
- **DAO**: Berisi metode yang digunakan untuk mengakses database.

- Contoh kode yang berisi konfigurasi database dengan satu entitas dan satu DAO:

```
@Entity
class User (
    @PrimaryKey
    var uid: Long,
    @ColumnInfo(name = "first_name")
    var firstName: String,
    @ColumnInfo(name = "last_name")
    var lastName: String
)
```

```
@Dao
interface UserDao {
    @Query("SELECT * FROM user")
    val getAll: List<User>

    @Insert
    fun insertAll(vararg users: User)

    @Delete
    fun delete(user: User)
    }
```

```
@Database(entities = [User::class], version = 1)
abstract class AppDatabase : RoomDatabase()
{
    abstract fun userDao(): UserDao
}
```

Komponen ROOM

- Setelah membuat berkas di atas, Anda mendapatkan instance dari database yang dibuat menggunakan kode berikut:

```
var db: AppDatabase = Room.databaseBuilder(applicationContext, AppDatabase::class.java, "database-name").build()
```

- Jika dibandingkan dengan API SQlite, Room lebih ringkas dan lebih mudah dalam melakukan maintenance. Kita bisa mengubah sebuah kelas POJO menjadi entitas dengan menambahkan anotasi @Entity dan menentukan atribut dari tabel tersebut seperti primary key-nya apakah boleh null atau tidak dll.

Latihan Room

- Seperti yang kita ketahui, Room adalah sebuah ORM (Object Relational Mapping) library. Dengan kata lain, Room akan memetakan obyek database ke obyek Java.
- Perbedaan antara library SQLite dan Room persistence antara lain:
  - Dalam kasus SQLite, tidak ada verifikasi query SQLite yang masih mentah pada saat waktu kompilasi. Tetapi di Room, ada validasi SQL pada saat waktu kompilasi.
  - Saat skema database berubah, kita perlu memperbarui query SQL yang sudah ada pada aplikasi sebelumnya secara manual.
  - Akan ada banyak kode boilerplate untuk mengkonversi query SQL dengan obyek data Java. Tapi dengan menggunakan Room, aplikasi mampu memetakan obyek database ke Java Object tanpa kode boilerplate.
  - Room dibangun untuk bekerja dengan LiveData dan RxJava untuk observasi data, sedangkan SQLite tidak.

Latihan ROOM

- Clone Repository https://github.com/KodeV-Academy/MyNoteApps.git
- Buatlah sebuah package baru dengan nama database, dan buat sebuah berkas Java/Kotlin baru dengan nama Note. Setelah terbentuk tambahkan annotation entity dan implementasikan Parcelable. Maka kelas Note akan menjadi seperti ini:

```
@Parcelize
@Entity
data class Note(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "id")
    var id: Int = 0,
    @ColumnInfo(name = "title")
    var title: String? = null,
    @ColumnInfo(name = "description")
    var description: String? = null,
    @ColumnInfo(name = "date")
    var date: String? = null
): Parcelable
```

-KiniKu-

KodeV
Koding Akademi

Latihan ROOM

- Buatlah satu kelas interface di dalam package database dengan nama NoteDao. Kelas ini nantinya digunakan untuk melakukan eksekusi quiring. Setelah terbentuk, tambahkan kodenya menjadi seperti ini:

```
@Dao
interface NoteDao {
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    fun insert(note: Note)

    @Update
    fun update(note: Note)

    @Delete
    fun delete(note: Note)

    @Query("SELECT * from note ORDER BY id ASC")
    fun getAllNotes(): LiveData<List<Note>>
}
```

-KiniKu-

KodeV
Koding Akademi

Latihan ROOM

- Selanjutnya buat kelas lagi di dalam package database dengan nama NoteRoomDatabase. Kelas ini akan digunakan untuk menginisialisasi database dalam aplikasi. Setelah terbentuk, tambahkan implementasi RoomDatabase, tambahkan annotation @Database dan ubah menjadi abstract class seperti berikut:

```
@Database(entities = [Note::class], version = 1)
abstract class NoteRoomDatabase : RoomDatabase() {

}
```

Latihan ROOM

- Setelah itu tambahkan kode berikut untuk membuat variabel global untuk Dao yang nanti akan dipanggil di kelas repository.

```
@Database(entities = [Note::class], version = 1)
abstract class NoteRoomDatabase : RoomDatabase() {
    abstract fun noteDao(): NoteDao
    companion object {
        @Volatile
        private var INSTANCE: NoteRoomDatabase? = null
        @JvmStatic
        fun getDatabase(context: Context): NoteRoomDatabase {
            if (INSTANCE == null) {
                synchronized(NoteRoomDatabase::class.java) {
                    INSTANCE = Room.databaseBuilder(context.applicationContext,
                            NoteRoomDatabase::class.java, "note_database")
                            .build()
                }
            }
            return INSTANCE as NoteRoomDatabase
        }
    }
}
```

-KiniKu-

KodeV
Koding Akademi

Latihan ROOM

- Selanjutnya buat package baru dengan nama repository, dan buat kelas di dalamnya dengan nama NoteRepository. Kelas ini berfungsi sebagai penghubung antara ViewModel dengan database atau resource data.

```
private val mNotesDao: NoteDao
    private val executorService = Executors.newSingleThreadExecutor()

    init {
        val db = NoteRoomDatabase.getDatabase(application)
        mNotesDao = db.noteDao()
    }

    fun getAllNotes() = mNotesDao.getAllNotes()

    fun insert(note: Note) {
        executorService.execute { mNotesDao.insert(note) }
    }

    fun delete(note: Note) {
        executorService.execute { mNotesDao.delete(note) }
    }

    fun update(note: Note) {
        executorService.execute { mNotesDao.update(note) }
    }
```

-KiniKu-

KodeV
Koding Akademi

Latihan ROOM

- Setelah itu, buatlah sebuah package baru dengan nama **ui**. Di dalamnya buat lagi 2 package baru dengan nama **insert** dan **main**. Setelah itu, pindahkan MainActivity ke package **main**. Ini untuk memudahkan kita memilah-milah fungsi dari masing masing kelas. Setelah itu kita bisa membuat sebuah activity baru dengan nama **NoteAddUpdateActivity**.

- Setelah membuat Activity, kita akan buat terlebih dahulu kelas ViewModel sebagai penghubung antara Activity dengan Repository. Buatlah kelas di dalam package insert dengan nama **NoteAddUpdateViewModel**. Kemudian ubah dan tambahkan kodenya menjadi seperti ini:

```
class NoteAddUpdateViewModel(application: Application) : ViewModel() {

    private val mNoteRepository = NoteRepository(application)

    fun insert(note: Note) {
        mNoteRepository.insert(note)
    }
    fun update(note: Note) {
        mNoteRepository.update(note)
    }
    fun delete(note: Note) {
        mNoteRepository.delete(note)
    }
}
```

-KiniKu-

KodeV
Koding Akademi

Latihan ROOM

- Setelah itu, buat kelas ViewModel di package main dengan nama MainViewModel. Ubah dan tambahkan kode pada kelas tersebut menjadi seperti ini:

```
class MainViewModel(application: Application) : ViewModel() {
    private val mNoteRepository = NoteRepository(application)

    fun getAllNotes() = mNoteRepository.getAllNotes()
}
```

Latihan ROOM

- Sebelum melangkah lebih lanjut, buatlah package baru dengan nama helper. Package ini akan kita isi dengan beberapa kelas untuk membuat kode kita menjadi lebih mudah. Pertama buatlah kelas di dalamnya dengan nama DateHelper. Kelas ini berfungsi untuk mendapatkan waktu seperti tanggal, bulan, tahun dan jam. Setelah terbentuk, kita bisa ubah dan tambahkan kode pada kelas tersebut menjadi seperti ini:

```
object DateHelper {
    fun getCurrentDate(): String {
        val dateFormat = SimpleDateFormat("yyyy/MM/dd HH:mm:ss", Locale.getDefault())
        val date = Date()
        return dateFormat.format(date)
    }
}
```

Latihan ROOM

- Selanjutnya buat kelas baru dengan nama ViewModelFactory. Kelas ini berfungsi untuk menambahkan context ketika memanggil kelas ViewModel di dalam Activity. Setelah kelas tersebut terbentuk, ubah dan tambahkan kode pada kelas tersebut menjadi seperti ini:

```
@Suppress("UNCHECKED_CAST")
class ViewModelFactory private constructor(private val mApplication: Application) : ViewModelProvider.NewInstanceFactory() {
    companion object {
        @Volatile
        private var INSTANCE: ViewModelFactory? = null
        @JvmStatic
        fun getInstance(application: Application): ViewModelFactory {
            if (INSTANCE == null) {
                synchronized(ViewModelFactory::class.java) {
                    INSTANCE = ViewModelFactory(application)
                }
            }
            return INSTANCE as ViewModelFactory
        }
    }

    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        if (modelClass.isAssignableFrom(MainViewModel::class.java)) {
            return MainViewModel(mApplication) as T
        } else if (modelClass.isAssignableFrom(NoteAddUpdateViewModel::class.java)) {
            return NoteAddUpdateViewModel(mApplication) as T
        }
        throw IllegalArgumentException("Unknown ViewModel class: ${modelClass.name}")
    }
}
```

-KiniKu-

KodeV
Koding Akademi

Latihan ROOM

- Oke setelah itu, tambahkan kelas baru lagi di dalam package helper dan beri nama NoteDiffCallback untuk melakukan pengecekan apakah ada perubahan list note. Kelas ini nanti akan dipanggil di kelas adapter. Setelah itu, ubah dan tambahkan kode pada kelas tersebut menjadi seperti ini:

```
class NoteDiffCallback(private val mOldNoteList: List<Note>, private val mNewNoteList: List<Note>) :
    DiffUtil.Callback() {

    override fun getOldListSize(): Int {
        return mOldNoteList.size
    }

    override fun getNewListSize(): Int {
        return mNewNoteList.size
    }

    override fun areItemsTheSame(oldItemPosition: Int, newItemPosition: Int): Boolean {
        return mOldNoteList[oldItemPosition].id == mNewNoteList[newItemPosition].id
    }

    override fun areContentsTheSame(oldItemPosition: Int, newItemPosition: Int): Boolean {
        val oldEmployee = mOldNoteList[oldItemPosition]
        val newEmployee = mNewNoteList[newItemPosition]
        return oldEmployee.title == newEmployee.title && oldEmployee.description == newEmployee.description
    }

}
```

Latihan ROOM

- Selanjutnya kita buat menu yang akan digunakan di NoteAddUpdateActivity.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/action_delete"
        android:icon="@drawable/ic_clear"
        android:title="@string/delete"
        app:showAsAction="always" />
</menu>
```

Latihan ROOM

- Setelah resource menu sudah siap, bukalah NoteAddUpdateActivity. Tambahkan kode berikut untuk menginisialisasi view yang ada di layout-nya.

```
companion object {
      const val EXTRA_NOTE = "extra_note"
      const val EXTRA_POSITION = "extra_position"
      const val REQUEST_ADD = 100
      const val RESULT_ADD = 101
      const val REQUEST_UPDATE = 200
      const val RESULT_UPDATE = 201
      const val RESULT_DELETE = 301
      const val ALERT_DIALOG_CLOSE = 10
      const val ALERT_DIALOG_DELETE = 20
  }
  private var isEdit = false
  private var note: Note? = null
  private var position = 0
```

Latihan ROOM

- Selanjutnya hubungkan NoteAddUpdateViewModel dengan NoteAddUpdateActivity.

```
...

private lateinit var noteAddUpdateViewModel: NoteAddUpdateViewModel
private var _binding: ActivityNoteAddUpdateBinding? = null
private val binding get() = _binding
```

```
override fun onDestroy() {
        super.onDestroy()
        _binding = null
    }
```

```
override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        _binding = ActivityNoteAddUpdateBinding.inflate(layoutInflater)
        setContentView(binding?.root)

        noteAddUpdateViewModel = obtainViewModel(this@NoteAddUpdateActivity)
    }
```

```
private fun obtainViewModel(activity: AppCompatActivity): NoteAddUpdateViewModel {
        val factory = ViewModelFactory.getInstance(activity.application)
        return ViewModelProvider(activity, factory)[NoteAddUpdateViewModel::class.java]
    }
```

-KiniKu-

KodeV
Koding Akademi

Latihan ROOM

- Setelah itu, karena kelas ini berfungsi untuk menambahkan, memperbarui dan menghapus item, maka perlu masukkan kode berikut:

```
...
note = intent.getParcelableExtra(EXTRA_NOTE)
        if (note != null) {
            position = intent.getIntExtra(EXTRA_POSITION, 0)
            isEdit = true
        } else {
            note = Note()
        }

        val actionBarTitle: String
        val btnTitle: String
        if (isEdit) {
            actionBarTitle = getString(R.string.change)
            btnTitle = getString(R.string.update)
            if (note != null) {
                note?.let { note ->
                    binding?.edtTitle?.setText(note.title)
                    binding?.edtDescription?.setText(note.description)
                }
            }
        } else {
            actionBarTitle = getString(R.string.add)
            btnTitle = getString(R.string.save)
        }

        supportActionBar?.title = actionBarTitle
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
        binding?.btnSubmit?.text = btnTitle
```

Latihan ROOM

- Setelah itu, tambahkan aksi untuk button-nya seperti ini:

```
binding?.btnSubmit?.setOnClickListener {
        val title = binding?.edtTitle?.text.toString().trim()
        val description = binding?.edtDescription?.text.toString().trim()
        if (title.isEmpty()) {
            binding?.edtTitle?.error = getString(R.string.empty)
        } else if (description.isEmpty()) {
            binding?.edtDescription?.error = getString(R.string.empty)
        } else {
            note.let { note ->
                note?.title = title
                note?.description = description
            }
            val intent = Intent().apply {
                putExtra(EXTRA_NOTE, note)
                putExtra(EXTRA_POSITION, position)
            }
            if (isEdit) {
                noteAddUpdateViewModel.update(note as Note)
                setResult(RESULT_UPDATE, intent)
                finish()
            } else {
                note.let { note ->
                    note?.date = DateHelper.getCurrentDate()
                }
                noteAddUpdateViewModel.insert(note as Note)
                setResult(RESULT_ADD, intent)
                finish()
            }
        }
    }
```

-KiniKu-

KodeV
Koding Akademi

Latihan ROOM

- Setelah itu, tambahkan kode berikut untuk menghapus item dari database:

```
override fun onCreateOptionsMenu(menu: Menu): Boolean {
        if (isEdit) {
            menuInflater.inflate(R.menu.menu_form, menu)
        }
        return super.onCreateOptionsMenu(menu)
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        when (item.itemId) {
            R.id.action_delete -> showAlertDialog(ALERT_DIALOG_DELETE)
            android.R.id.home -> showAlertDialog(ALERT_DIALOG_CLOSE)
        }
        return super.onOptionsItemSelected(item)
    }

    override fun onBackPressed() {
        showAlertDialog(ALERT_DIALOG_CLOSE)
    }
```

Latihan ROOM

- Buat function showAlertDialog

```
private fun showAlertDialog(type: Int) {
        val isDialogClose = type == ALERT_DIALOG_CLOSE
        val dialogTitle: String
        val dialogMessage: String
        if (isDialogClose) {
            dialogTitle = getString(R.string.cancel)
            dialogMessage = getString(R.string.message_cancel)
        } else {
            dialogMessage = getString(R.string.message_delete)
            dialogTitle = getString(R.string.delete)
        }
        val alertDialogBuilder = AlertDialog.Builder(this)
        with(alertDialogBuilder) {
            setTitle(dialogTitle)
            setMessage(dialogMessage)
            setCancelable(false)
            setPositiveButton(getString(R.string.yes)) { _, _ ->
                if (!isDialogClose) {
                    noteAddUpdateViewModel.delete(note as Note)
                    val intent = Intent()
                    intent.putExtra(EXTRA_POSITION, position)
                    setResult(RESULT_DELETE, intent)
                }
                finish()
            }
            setNegativeButton(getString(R.string.no)) { dialog, _ -> dialog.cancel() }
        }
        val alertDialog = alertDialogBuilder.create()
        alertDialog.show()
    }
```

-KiniKu-

KodeV
Koding Akademi

Latihan ROOM

- Sebelum Anda mengubah kode yang ada di MainActivity, buatlah kelas baru untuk adapter dari list item di package main. Berilah nama NoteAdapter, dan tambahkan kode berikut:

```
class NoteAdapter internal constructor(private val activity: Activity) : RecyclerView.Adapter<NoteAdapter.NoteViewHolder>() {
    private val listNotes = ArrayList<Note>()

    fun setListNotes(listNotes: List<Note>) {
        val diffCallback = NoteDiffCallback(this.listNotes, listNotes)
        val diffResult = DiffUtil.calculateDiff(diffCallback)
        this.listNotes.clear()
        this.listNotes.addAll(listNotes)
        diffResult.dispatchUpdatesTo(this)
    }
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): NoteViewHolder {
        val binding = ItemNoteBinding.inflate(LayoutInflater.from(parent.context), parent, false)
        return NoteViewHolder(binding)
    }
    override fun onBindViewHolder(holder: NoteViewHolder, position: Int) {
        holder.bind(listNotes[position])
    }
    override fun getItemCount(): Int {
        return listNotes.size
    }
    inner class NoteViewHolder(private val binding: ItemNoteBinding) : RecyclerView.ViewHolder(binding.root) {
        fun bind(note: Note) {
            with(binding) {
                tvItemTitle.text = note.title
                tvItemDate.text = note.date
                tvItemDescription.text = note.description
                cvItemNote.setOnClickListener {
                    val intent = Intent(activity, NoteAddUpdateActivity::class.java)
                    intent.putExtra(NoteAddUpdateActivity.EXTRA_POSITION, adapterPosition)
                    intent.putExtra(NoteAddUpdateActivity.EXTRA_NOTE, note)
                    activity.startActivityForResult(intent, NoteAddUpdateActivity.REQUEST_UPDATE)
                }
            }
        }
    }
}
```

Latihan ROOM

- Terakhir, buka kelas MainActivity dan tambahkan kode-kode berikut:

```
private var _binding: ActivityMainBinding? = null
    private val binding get() = _binding
    private lateinit var noteAdapter: NoteAdapter

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        _binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding?.root)

        noteAdapter = NoteAdapter(this@MainActivity)

        binding?.rvNotes?.apply {
            layoutManager = LinearLayoutManager(this@MainActivity)
            setHasFixedSize(true)
            adapter = noteAdapter
        }
    }

    override fun onDestroy() {
        super.onDestroy()
        _binding = null
    }
```

-KiniKu-

KodeV
Koding Akademi

Latihan ROOM

- Setelah menginisialisasi RecyclerView, FloatingActionButton dan NoteAdapter, tambahkan kode di bawah ini untuk menghubungkan MainViewModel dengan MainActivity.

```
val mainViewModel = obtainViewModel(this@MainActivity)
mainViewModel.getAllNotes().observe(this, noteObserver)

...

private fun obtainViewModel(activity: AppCompatActivity): MainViewModel {
        val factory = ViewModelFactory.getInstance(activity.application)
        return ViewModelProvider(activity, factory)[MainViewModel::class.java]
}

    private val noteObserver = Observer<List<Note>> { noteList ->
        if (noteList != null) {
            noteAdapter.setListNotes(noteList)
        }
    }
```

Latihan ROOM

- Setelah itu, tambahkan kode untuk aksi pada fab dan tambahkan metode onActivityResult untuk menerima respon dari NoteAddUpdateActivity.

```
binding?.fabAdd?.setOnClickListener { view ->
        if (view.id == R.id.fab_add) {
            val intent = Intent(this@MainActivity, NoteAddUpdateActivity::class.java)
            startActivityForResult(intent, NoteAddUpdateActivity.REQUEST_ADD)
        }
    }

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (data != null) {
        if (requestCode == NoteAddUpdateActivity.REQUEST_ADD) {
            if (resultCode == NoteAddUpdateActivity.RESULT_ADD) {
                showSnackbarMessage(getString(R.string.added))
            }
        } else if (requestCode == NoteAddUpdateActivity.REQUEST_UPDATE) {
            if (resultCode == NoteAddUpdateActivity.RESULT_UPDATE) {
                showSnackbarMessage(getString(R.string.changed))
            } else if (resultCode == NoteAddUpdateActivity.RESULT_DELETE) {
                showSnackbarMessage(getString(R.string.deleted))
            }
        }
    }
}

private fun showSnackbarMessage(message: String) {
    Snackbar.make(binding?.root as View, message, Snackbar.LENGTH_SHORT).show()
}
```

-KiniKu-

KodeV
Koding Akademi

Latihan ROOM

- Run Aplikasi

-KiniKu-

KodeV
Koding Akademi

Latihan Room dalam Proyek Jetpack

- Bukalah aplikasi MyJetpack yang sudah kita sebelumnya
- Setelah itu, buka build.gradle level module: app dan tambahkan library berikut:

```
id 'kotlin-kapt'
```

```
implementation "androidx.room:room-runtime:2.4.2"
kapt "androidx.room:room-compiler:2.4.2"
```

-KiniKu-

KodeV
Koding Akademi

Latihan Room dalam Proyek Jetpack

- Kali ini, aplikasi akan mengimplementasikan Room dan Repository offline-online. Oleh karena itu kita harus menentukan skema yang perlu diterapkan. Hal pertama yang perlu dilakukan adalah menyiapkan entity yang akan digunakan sebagai tabel. Sebelumnya kita sudah membuat beberapa Entity, bukalah satu persatu dan ubah kode di dalamnya.
- Pindahkan GameEntity ke dalam package data > local > entity
- Lalu ubah GameEntity menjadi seperti berikut :

```
@Parcelize
@Entity(tableName = "tb_game")
data class GameEntity(
    @PrimaryKey
    @NonNull
    @ColumnInfo(name = "id")
    var id: Int,

    @ColumnInfo(name = "name")
    var name: String,

    @ColumnInfo(name = "released")
    var released: String,

    @ColumnInfo(name = "background_image")
    var background_image: String,

    @ColumnInfo(name = "rating")
    var rating: String,

    @ColumnInfo(name = "platforms")
    var platforms: String,

    @ColumnInfo(name = "genres")
    var genres: String,

    @ColumnInfo(name = "minimum")
    var minimum: String,

    @ColumnInfo(name = "recommended")
    var recommended: String,

    @ColumnInfo(name = "favorite")
    var favorite: Boolean = false,
): Parcelable
```

-KiniKu-

KodeV
Koding Akademi

Latihan Room dalam Proyek Jetpack

- Setelah membuat Entity-Entity yang dibutuhkan, kita perlu menyiapkan Dao. Buatlah package baru di dalam package local dengan nama room dan buat kelas baru di dalamnya dengan nama GamesDao. Setelah kelas tersebut terbentuk, masukkan kode berikut di dalamnya:

```
@Dao
interface GamesDao {

    @Query("SELECT * FROM tb_game")
    fun getLocalGames(): LiveData<List<GameEntity>>

    @Query("SELECT * FROM tb_game where favorite = 1")
    fun getFavoriteGame(): LiveData<List<GameEntity>>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insertGame(games: List<GameEntity>)

    @Update
    fun updateGame(games : GameEntity)
}
```

-KiniKu-

KodeV
Koding Akademi

Latihan Room dalam Proyek Jetpack

- Selanjutnya buatlah sebuah kelas Room Database yang berfungsi sebagai builder database dalam aplikasi. Klik kanan di package room → new → Kotlin/Java Class dan beri nama GameDatabase. Selanjutnya tambahkan kode berikut:

```
@Database(entities = [GameEntity::class], version = 1, exportSchema = false)
abstract class GameDatabase : RoomDatabase() {
    abstract fun gameDao(): GamesDao

    companion object {

        @Volatile
        private var INSTANCE: GameDatabase? = null

        fun getInstance(context: Context): GameDatabase =
            INSTANCE ?: synchronized(this) {
                Room.databaseBuilder(
                    context.applicationContext,
                    GameDatabase::class.java,
                    "Games.db"
                ).build().apply {
                    INSTANCE = this
                }
            }
    }
}
```

-KiniKu-

KodeV
Koding Akademi

Latihan Room dalam Proyek Jetpack

- Setelah membuat kelas-kelas yang dibutuhkan untuk Room, buatlah kelas LocalDataSource dan hubungkan GameDao ke LocalDataSource. Klik kanan package local → new → Kotlin/Java Class dan beri nama LocalDataSource. Bukalah kelas LocalDataSource dan ubah kode di dalamnya menjadi seperti ini:

```
class LocalDataSource private constructor(private val gamesDao: GamesDao) {

    companion object {
        private var INSTANCE: LocalDataSource? = null

        fun getInstance(gamesDao: GamesDao): LocalDataSource =
            INSTANCE ?: LocalDataSource(gamesDao)
    }

    fun getLocalGames(): LiveData<List<GameEntity>> = gamesDao.getLocalGames()

    fun getFavoriteGame(): LiveData<List<GameEntity>> = gamesDao.getFavoriteGame()

    fun insertGame(listGame: List<GameEntity>) = gamesDao.insertGame(listGame)

    fun updateGame(game: GameEntity, newState: Boolean) {
        game.favorite = newState
        gamesDao.updateGame(game)
    }
}
```

Latihan Room dalam Proyek Jetpack

- Selanjutnya, buatlah kelas-kelas pembantu di dalamnya. Buatlah kelas baru untuk memberitahukan simulasi NetworkAPI berhasil atau tidak. Klik kanan di bagian data.source.remote → new → Kotlin/Java Class. Berilah nama StatusResponse dan tambahkan kode berikut di dalamnya:

```
enum class StatusResponse {
    SUCCESS,
    EMPTY,
    ERROR
}
```

- Setelah membuat kelas StatusResponse, buatlah kelas kembali di package yang sama dengan nama ApiResponse. Tambahkanlah kode di dalamnya:

```
class ApiResponse<T>(val status: StatusResponse, val body: T, val message: String?) {
    companion object {
        fun <T> success(body: T): ApiResponse<T> = ApiResponse(StatusResponse.SUCCESS, body, null)

        fun <T> empty(msg: String, body: T): ApiResponse<T> = ApiResponse(StatusResponse.EMPTY, body, msg)

        fun <T> error(msg: String, body: T): ApiResponse<T> = ApiResponse(StatusResponse.ERROR, body, msg)
    }
}
```

Latihan Room dalam Proyek Jetpack

- Kita juga perlu mengubah kode yang ada di RemoteDataSource yang awalnya menggunakan Callback menjadi LiveData. Bukalah kelas tersebut dan sesuaikanlah kode di dalamnya menjadi seperti ini:

```
fun getGames(): LiveData<ApiResponse<ResponseGame>> {
        val responseGame = MutableLiveData<ApiResponse<ResponseGame>>()
        val client = getApiService().getGames("d084045ca6164bbeb97021752a930416", "10")
        client.enqueue(object : Callback<ResponseGame> {
            override fun onResponse(call: Call<ResponseGame>, response: Response<ResponseGame>) {
                if (response.isSuccessful) {
                    response.body()?.let {
                        responseGame.value = ApiResponse.success(it)
                    }
                } else {
                    Log.d(TAG, "onResponse: ${response.message()}")
                }
            }

            override fun onFailure(call: Call<ResponseGame>, t: Throwable) {
                Log.d(TAG, "onFailure: ${t.localizedMessage}")
            }
        })
        return responseGame
    }
```

-KiniKu-

KodeV
Koding Akademi

Latihan Room dalam Proyek Jetpack

- Selanjutnya, Anda perlu menambahkan tambahan kelas pembantu. Klik kanan di package utils → new → Kotlin/Java Class dan berilah nama AppExecutors. Tambahkan kode berikut di dalamnya:

```
class AppExecutors constructor(
    private val diskIO: Executor,
    private val networkIO: Executor,
    private val mainThread: Executor
) {

    companion object {
        private const val THREAD_COUNT = 3
    }

    constructor() : this(
        Executors.newSingleThreadExecutor(),
        Executors.newFixedThreadPool(THREAD_COUNT),
        MainThreadExecutor()
    )

    fun diskIO(): Executor = diskIO

    fun networkIO(): Executor = networkIO

    fun mainThread(): Executor = mainThread

    private class MainThreadExecutor : Executor {
        private val mainThreadHandler = Handler(Looper.getMainLooper())

        override fun execute(command: Runnable) {
            mainThreadHandler.post(command)
        }
    }
}
```

Latihan Room dalam Proyek Jetpack

- Selanjutnya, untuk kebutuhan NetworkBoundResource dengan GameRepository kita perlu menambahkan kelas tambahan. Buatlah package terlebih dahulu dan beri nama vo. Pacakge vo artinya adalah Value Object, yang isinya kelas-kelas pembungkus data yang digunakan untuk ui. Buatlah kelas baru di dalamnya dan beri nama Status. Tambahkan kode berikut di dalamnya:

```
enum class Status {
    SUCCESS,
    ERROR,
    LOADING
}
```

- Kelas Status digunakan sebagai indikator dari sukses, gagal dan loading. Selanjutnya tambahkan kelas di dalam package vo lagi dan berilah nama Resource. Tambahkan kode di dalamnya:

```
data class Resource<T>(val status: Status, val data: T?, val message: String?) {
    companion object {
        fun <T> success(data: T?): Resource<T> = Resource(Status.SUCCESS, data, null)

        fun <T> error(msg: String?, data: T?): Resource<T> = Resource(Status.ERROR, data, msg)

        fun <T> loading(data: T?): Resource<T> = Resource(Status.LOADING, data, null)
    }
}
```

-KiniKu-

KodeV
Koding Akademi

Latihan Room dalam Proyek Jetpack

- Saat ini LocalDataSource sudah bisa kita pakai untuk mengelola database local. Tahap selanjutnya adalah menghubungkan LocalDataSource dengan RemoteDataSource untuk Offline-Online. Buatlah kelas NetworkBoundResource di dalam package data->Source. Maka hasil kelas yang sudah Anda buat seperti ini:
- Selanjutnya implementasikan kelas-kelas yang dibuat sebelumnya ke NetworkBoundResource. Bukalah kelas tersebut dan tambahkan kode berikut di dalamnya:

Latihan Room dalam Proyek Jetpack (NetworkBoundResource)

```
abstract class NetworkBoundResource<ResultType, RequestType>(private val mExecutors: AppExecutors) {

    private val result = MediatorLiveData<Resource<ResultType>>()

    init {
        result.value = Resource.loading(null)

        @Suppress("LeakingThis")
        val dbSource = loadFromDB()

        result.addSource(dbSource) { data ->
            result.removeSource(dbSource)
            if (shouldFetch(data)) {
                fetchFromNetwork(dbSource)
            } else {
                result.addSource(dbSource) { newData ->
                    result.value = Resource.success(newData)
                }
            }
        }
    }
    protected fun onFetchFailed() {}

    protected abstract fun loadFromDB(): LiveData<ResultType>

    protected abstract fun shouldFetch(data: ResultType?): Boolean

    protected abstract fun createCall(): LiveData<ApiResponse<RequestType>>

    protected abstract fun saveCallResult(data: RequestType)
```

-KiniKu-

KodeV
Koding Akademi

Latihan Room dalam Proyek Jetpack (NetworkBoundResource)

```
private fun fetchFromNetwork(dbSource: LiveData<ResultType>) {

    val apiResponse = createCall()

    result.addSource(dbSource) { newData ->
      result.value = Resource.loading(newData)
    }
    result.addSource(apiResponse) { response ->
      result.removeSource(apiResponse)
      result.removeSource(dbSource)
      when (response.status) {
        StatusResponse.SUCCESS ->
          mExecutors.diskIO().execute {
            saveCallResult(response.body)
            mExecutors.mainThread().execute {
              result.addSource(loadFromDB()) { newData ->
                result.value = Resource.success(newData)
              }
            }
          }
        StatusResponse.EMPTY -> mExecutors.mainThread().execute {
          result.addSource(loadFromDB()) { newData ->
            result.value = Resource.success(newData)
          }
        }
        StatusResponse.ERROR -> {
          onFetchFailed()
          result.addSource(dbSource) { newData ->
            result.value = Resource.error(response.message, newData)
          }
        }
      }
    }
  }

  fun asLiveData(): LiveData<Resource<ResultType>> = result
```

-KiniKu-

KodeV
Koding Akademi

Latihan Room dalam Proyek Jetpack

- Selesai membuat kelas NetworkBoundResource, kita bisa memakainya di GameRepository untuk menyimpan data RemoteDataSource menjadi LocalDataSource.
- Sebelum mengubah GameRepository, bukalah GameDataSource dan ubahlah kode di dalamnya menjadi seperti ini:

```
interface GameDataSource {

    fun getGames(): LiveData<Resource<List<GameEntity>>>

   fun getFavoriteGame(): LiveData<List<GameEntity>>

   fun updateGame(game: GameEntity, newState: Boolean)

}
```

Latihan Room dalam Proyek Jetpack

- Selanjutnya sesuaikan kelas GameRepository:

```
class GameRepository private constructor(
    private val remoteDataSource: RemoteDataSource,
    private val localDataSource: LocalDataSource,
    private val appExecutors: AppExecutors
) :
    GameDataSource {

    companion object {
        @Volatile
        private var instance: GameRepository? = null

      fun getInstance(remoteData: RemoteDataSource, localData: LocalDataSource, appExecutors: AppExecutors): GameRepository
=
            instance ?: synchronized(this) {
                instance ?: GameRepository(remoteData, localData, appExecutors).apply {
                    instance = this
                }
            }   }

    override fun getGames(): LiveData<Resource<ResponseGame>> {
        TODO("Not yet implemented")
    }

    override fun getFavoriteGame(): LiveData<List<GameEntity>> {
        TODO("Not yet implemented")
    }

    override fun updateGame(game: GameEntity, newState: Boolean) {
        TODO("Not yet implemented")
    }

}
```

Latihan Room dalam Proyek Jetpack

- Karena Anda merubah constructor, maka Anda juga perlu memperbarui kode yang ada di kelas Injection. Bukalah kelas tersebut dan ubahlah kode di dalamnya:

```
object Injection {
    fun provideRepository(context: Context): GameRepository {

        val database = GameDatabase.getInstance(context)

        val remoteDataSource = RemoteDataSource.getInstance(JsonHelper(context))
        val localDataSource = LocalDataSource.getInstance(database.gameDao())
        val appExecutors = AppExecutors()

        return GameRepository.getInstance(remoteDataSource, localDataSource, appExecutors)
    }
}
```

Latihan Room dalam Proyek Jetpack

- Selanjutnya terapkan NetworkBoundResource di masing-masing fungsi yang ada di GameRepository

```
override fun getGames(): LiveData<Resource<List<GameEntity>>> {
        return object : NetworkBoundResource<List<GameEntity>, ResponseGame>(appExecutors) {

        }.asLiveData()
    }
```

```
override fun getGames(): LiveData<Resource<List<GameEntity>>> {
        return object : NetworkBoundResource<List<GameEntity>, ResponseGame>(appExecutors) {
            override fun loadFromDB(): LiveData<List<GameEntity>> {
                TODO("Not yet implemented")
            }

            override fun shouldFetch(data: List<GameEntity>?): Boolean {
                TODO("Not yet implemented")
            }

            override fun createCall(): LiveData<ApiResponse<ResponseGame>> {
                TODO("Not yet implemented")
            }

            override fun saveCallResult(data: ResponseGame) {
                TODO("Not yet implemented")
            }

        }.asLiveData()
    }
```

Latihan Room dalam Proyek Jetpack

- Selanjutnya terapkan NetworkBoundResource di masing-masing fungsi yang ada di GameRepository

```
override fun loadFromDB(): LiveData<List<GameEntity>> {
        return localDataSource.getLocalGames()
}
```

```
override fun shouldFetch(data: List<GameEntity>?): Boolean {
        return data == null || data.isEmpty()
}
```

```
override fun createCall(): LiveData<ApiResponse<ResponseGame>> {
       return remoteDataSource.getGames()
}
```

-KiniKu-

KodeV
Koding Akademi

Latihan Room dalam Proyek Jetpack

```
override fun saveCallResult(data: ResponseGame) {
        val listGame = ArrayList<GameEntity>()
        for (i in data.results) {
            val listPlatform = ArrayList<String>()
            val listGenre = ArrayList<String>()
            var recommended = ""
            var minimum = ""

            i.platforms.map {
                if (it.platform.name == "PC") {
                    minimum = it.requirements_en?.minimum.toString()
                    recommended = it.requirements_en?.recommended.toString()
                }
                listPlatform.add(it.platform.name)
            }

            i.genres.map {
                listGenre.add(it.name)
            }

            val game = GameEntity(
                i.id,
                i.name,
                i.released,
                i.background_image,
                i.rating.toString(),
                listPlatform.toString(),
                listGenre.toString(),
                minimum,
                recommended)

            listGame.add(game)
        }
        localDataSource.insertGame(listGame)
    }
```

Latihan Room dalam Proyek Jetpack

```
override fun getFavoriteGame(): LiveData<List<GameEntity>> =
        localDataSource.getFavoriteGame()
```

```
override fun updateGame(game: GameEntity, newState: Boolean) =
        appExecutors.diskIO().execute { localDataSource.updateGame(game, newState) }
```

-KiniKu-

KodeV
Koding Akademi

Latihan Room dalam Proyek Jetpack

- Anda sudah memperbarui GameRepository menjadi offline-online. Karena semua data dari GameRepository dibungkus dengan kelas Resource, maka Anda perlu memperbarui kode di tiap kelas ViewModel. Bukalah satu-persatu kelas ViewModel tersebut.

```
class GameViewModel(private val gameRepository: GameRepository): ViewModel() {

    fun getGames(): LiveData<Resource<List<GameEntity>>> = gameRepository.getGames()

    fun getFavoriteGame(): LiveData<List<GameEntity>> = gameRepository.getFavoriteGame()

    fun updateGame(game: GameEntity, newState: Boolean) = gameRepository.updateGame(game, newState)

}
```

Latihan Room dalam Proyek Jetpack

- Sesuaikan observe di GameFragment dengan menerapkan state LOADING, SUCCESS, ERROR dan pada GameAdapter ubah data DataGame menjadi GameEntity

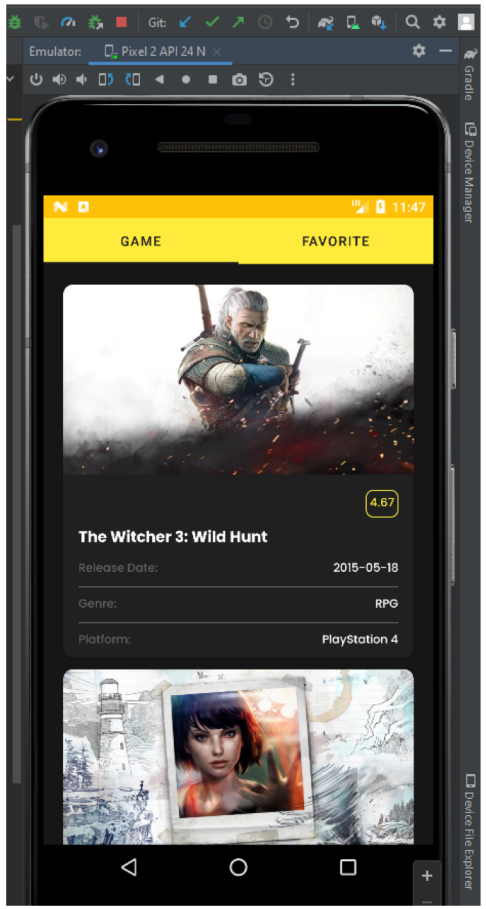
```
val gameAdapter = GameAdapter()
    viewModel.getGames().observe(viewLifecycleOwner) { response ->
        if (response != null) {
            when (response.status) {
                Status.LOADING -> {
                    binding.progressCircular.visibility = View.VISIBLE
                }
                Status.SUCCESS -> {
                    binding.progressCircular.visibility = View.GONE
                    response.data?.let {
                        gameAdapter.setData(it)
                    }

                }
                Status.ERROR -> {
                    binding.progressCircular.visibility = View.GONE
                    Toast.makeText(context, "Terjadi kesalahan", Toast.LENGTH_SHORT).show()
                }
            }
        }
    }
```

-KiniKu-

KodeV
Koding Akademi

Latihan Room dalam Proyek Jetpack

- Sampai sini jalankan terlebih dahulu project Anda, maka hasilnya akan seperti ini:



DB Browser for SQLite - D:\Games.db

File Edit View Tools Help

New Database | Open Database | Write Changes | Revert Changes | Open Project | Save Project | Attach Database | Close Database

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: tb_game — Filter in any column

	id	name	released	background_image	rating	platforms	genres	minimum	recommended	favorite
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	3328	The Witcher 3: Wild Hunt	2015-05-18	https://media.rawg.io/media/games/...	4.67	PlayStation 4	RPG	null	null	0
2	3439	Life is Strange	2015-01-29	https://media.rawg.io/media/games/...	4.11	Xbox One	Adventure	null	null	0
3	3498	Grand Theft Auto V	2013-09-17	https://media.rawg.io/media/games/...	4.48	PlayStation 5	Adventure	null	null	0
4	4062	BioShock Infinite	2013-03-26	https://media.rawg.io/media/games/fc1/...	4.38	Xbox One	Shooter	null	null	0
5	4200	Portal 2	2011-04-18	https://media.rawg.io/media/games/...	4.62	Xbox One	Puzzle	null	null	0
6	4291	Counter-Strike: Global Offensive	2012-08-21	https://media.rawg.io/media/games/...	3.57	PlayStation 3	Shooter	null	null	0
7	5286	Tomb Raider (2013)	2013-03-05	https://media.rawg.io/media/games/...	4.06	PlayStation 3	Adventure	null	null	0
8	5679	The Elder Scrolls V: Skyrim	2011-11-11	https://media.rawg.io/media/games/7cf/...	4.42	PlayStation 3	RPG	null	null	0
9	12020	Left 4 Dead 2	2009-11-17	https://media.rawg.io/media/games/d58/...	4.09	Xbox 360	Shooter	null	null	0
10	13536	Portal	2007-10-09	https://media.rawg.io/media/games/7fa/...	4.51	Android	Puzzle	4.4 and up	null	0

-KiniKu-

KodeV Koding Akademi

Latihan Room dalam Proyek Jetpack (Add Data to Favorite)

- Hapus pemanggilan data di FavoriteFragment terlebih dahulu
- Tambahkan FloatingActionButton untuk memasukan game ke dalam list favorite

```
<com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/fab_favorite"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="end|bottom"
        android:layout_margin="16dp"
        android:contentDescription="@string/add_to_favorite"
        android:src="@drawable/ic_baseline_favorite_selector"
        app:backgroundTint="@color/colorSoftBlack"
        app:tint="@color/white" />
```

Latihan Room dalam Proyek Jetpack (Add Data to Favorite)

- Selanjutnya pada DetailGameActivity, buat inisiasi data dan isFavorite untuk menampung nilai

```
private lateinit var data: GameEntity
private var isFavorite = false
```

- Ubah data yang ada di dalam function populateView() menjadi seperti ini

```
private fun populateView() {
        data = intent.getParcelableExtra(EXTRA_DATA)!!
        binding.apply {
            ...
            isFavorite = data.favorite
            populateFabButton(isFavorite)
        }
    }
```

- Tambahkan function populateFabButton() untuk perubahan state di FloatingActionButton

```
private fun populateFabButton(favorite: Boolean) {
        if (favorite)
            binding.fabFavorite.setImageResource(R.drawable.ic_baseline_favorite)
        else
            binding.fabFavorite.setImageResource(R.drawable.ic_baseline_favorite_border)
    }
```

-KiniKu-

KodeV
Koding Akademi

Latihan Room dalam Proyek Jetpack (Add Data to Favorite)

- Inisiasikan viewmodel didalam method onCreate()

```
val factory = ViewModelFactory.getInstance(this@DetailGameActivity)
val viewModel = ViewModelProvider(this@DetailGameActivity, factory)[GameViewModel::class.java]
```

- Lalu tambahkan setOnClickListener pada FloatingActionButton

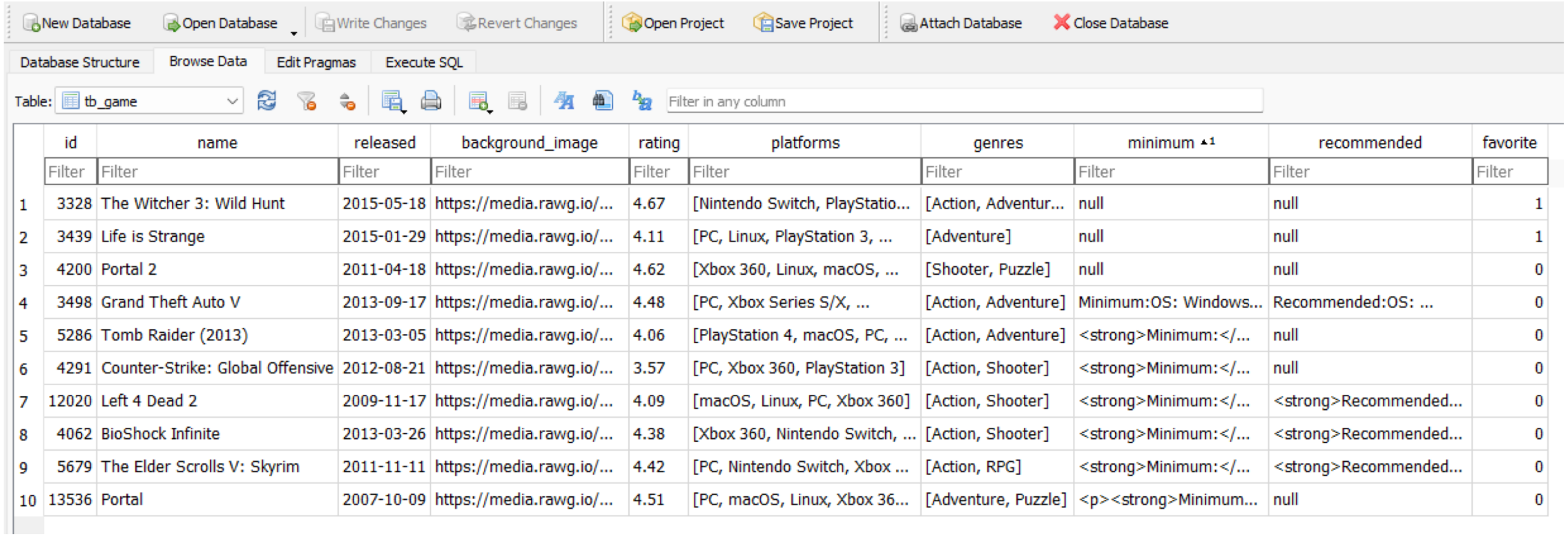
```
binding.fabFavorite.setOnClickListener {
        viewModel.updateGame(data, !isFavorite)
        populateFabButton(!isFavorite)
        if (isFavorite)
            Snackbar.make(it, "Berhasil Dihapus dari Favorit", Snackbar.LENGTH_SHORT).show()
        else
            Snackbar.make(it, "Berhasil Ditambah ke Favorit", Snackbar.LENGTH_SHORT).show()
    }
```

-KiniKu-

KodeV
Koding Akademi

Latihan Room dalam Proyek Jetpack (Add Data to Favorite)

- Jalankan aplikasi dan lihat perubahan datanya pada database menggunakan SQLite Browser



	id	name	released	background_image	rating	platforms	genres	minimum	recommended	favorite
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	3328	The Witcher 3: Wild Hunt	2015-05-18	https://media.rawg.io/...	4.67	[Nintendo Switch, PlayStatio...	[Action, Adventur...	null	null	1
2	3439	Life is Strange	2015-01-29	https://media.rawg.io/...	4.11	[PC, Linux, PlayStation 3, ...	[Adventure]	null	null	1
3	4200	Portal 2	2011-04-18	https://media.rawg.io/...	4.62	[Xbox 360, Linux, macOS, ...	[Shooter, Puzzle]	null	null	0
4	3498	Grand Theft Auto V	2013-09-17	https://media.rawg.io/...	4.48	[PC, Xbox Series S/X, ...	[Action, Adventure]	Minimum:OS: Windows...	Recommended:OS: ...	0
5	5286	Tomb Raider (2013)	2013-03-05	https://media.rawg.io/...	4.06	[PlayStation 4, macOS, PC, ...	[Action, Adventure]	Minimum:</...	null	0
6	4291	Counter-Strike: Global Offensive	2012-08-21	https://media.rawg.io/...	3.57	[PC, Xbox 360, PlayStation 3]	[Action, Shooter]	Minimum:</...	null	0
7	12020	Left 4 Dead 2	2009-11-17	https://media.rawg.io/...	4.09	[macOS, Linux, PC, Xbox 360]	[Action, Shooter]	Minimum:</...	Recommended...	0
8	4062	BioShock Infinite	2013-03-26	https://media.rawg.io/...	4.38	[Xbox 360, Nintendo Switch, ...	[Action, Shooter]	Minimum:</...	Recommended...	0
9	5679	The Elder Scrolls V: Skyrim	2011-11-11	https://media.rawg.io/...	4.42	[PC, Nintendo Switch, Xbox ...	[Action, RPG]	Minimum:</...	Recommended...	0
10	13536	Portal	2007-10-09	https://media.rawg.io/...	4.51	[PC, macOS, Linux, Xbox 36...	[Adventure, Puzzle]	<p>Minimum...	null	0

-KiniKu-

KodeV
Koding Akademi

Latihan Room dalam Proyek Jetpack (Show Data from Favorite)

- Selanjutnya kita tinggal menampilkan data list favorite game ke dalam FavoriteFragment()
- Inisiasikan viewmodel dan recyelerview nya terlebih dahulu lalu panggil fungsi getFavoriteGame()

```
val factory = ViewModelFactory.getInstance(requireActivity())
val viewModel = ViewModelProvider(this@FavoriteFragment, factory)[GameViewModel::class.java]
```

```
val favoriteAdapter = FavoriteAdapter()
        binding.rvFavorite.apply {
            setHasFixedSize(true)
            adapter = favoriteAdapter
        }
```

```
viewModel.getFavoriteGame().observe(viewLifecycleOwner) {
            favoriteAdapter.setData(it)
            binding.progressCircular.visibility = View.GONE
        }
```

-KiniKu-

KodeV
Koding Akademi

Latihan Room dalam Proyek Jetpack (Show Data from Favorite)

- Buatkan function ketika tombol Unfavorite, Share, dan View ditekan

```
favoriteAdapter.onItemFavoriteClick = {
        viewModel.updateGame(it, !it.favorite)
        Snackbar.make(requireView(), "Berhasil Dihapus dari Favorit", Snackbar.LENGTH_SHORT).show()
    }
```

```
favoriteAdapter.onItemShareClick = {
        val mimeType = "text/plain"
        ShareCompat.IntentBuilder
            .from(requireActivity())
            .setType(mimeType)
            .setChooserTitle("Mainkan Game ${it.name} Ini Sekarang.")
            .setText(resources.getString(R.string.share_text, it.name))
            .startChooser()
    }
```

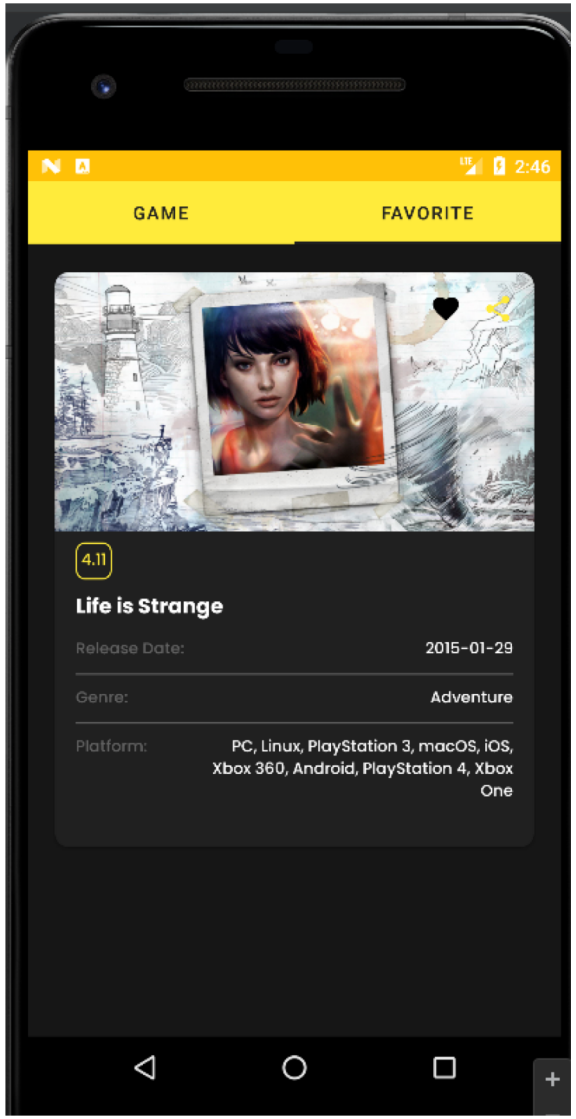
```
favoriteAdapter.onItemClick = {
        val intent = Intent(requireContext(), DetailGameActivity::class.java)
        intent.putExtra(DetailGameActivity.EXTRA_DATA, it)
        requireActivity().startActivity(intent)
    }
```

-KiniKu-

KodeV
Koding Akademi

Latihan Room dalam Proyek Jetpack (Show Data from Favorite)

- Jalankan aplikasi dan lihat pada tab Favorite



-KiniKu-

KodeV
Koding Akademi

-KiniKu-

KodeV
Koding Akademi