



**BINAR**  
ACADEMY

# Data Storage Using ORM

**Gold** - Chapter 4 - Topic 5

---

**Selamat datang di Chapter 4 Topic 5 *online course*  
Android Developer dari Binar Academy!**





# Hai teman-teman ✨

Pada topik sebelumnya, kita belajar tentang SharedPreference. Kali ini masih terkait dengan database, kita akan eksplor tentang teknik **Object Relational Mapping (ORM)** yang dapat membantu dalam mengatur hubungan aplikasi dengan database.

Kayak apa sih? Yuk langsung aja kita OTW!



**Detailnya, kita bakal bahas hal-hal berikut ini:**

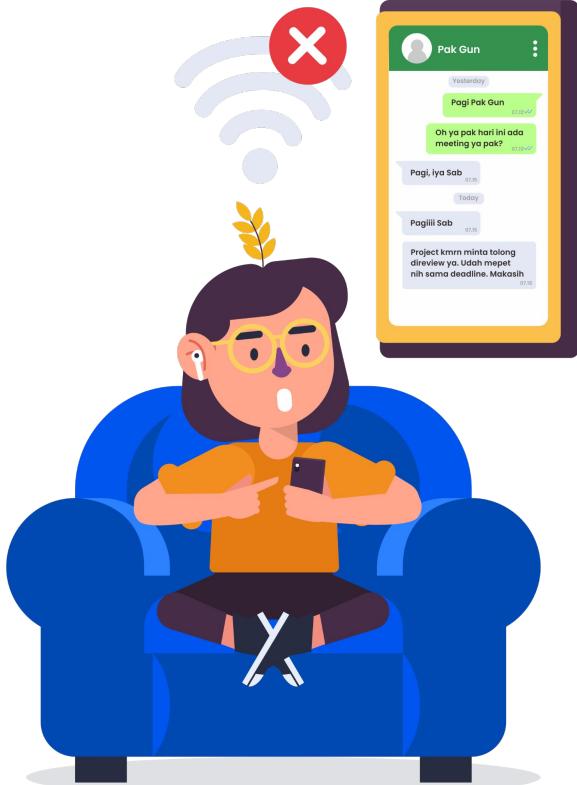
- Konsep Object Relational Mapping
- Pengenalan Library Room
- Penggunaan Library Room
- Library ORM Lainnya





Dalam menggunakan aplikasi chatting, kita pernah menemui peristiwa centang satu.

Kamu tahu nggak sih, itu adalah penerapan Object Relational Mapping (ORM)?



## Penggunaan ORM itu Tidak Asing loh!

Sini-sini, coba kita jabarin lagi. Ketika pesan diterima walaupun nggak terkoneksi ke internet? Apakah pesannya masih bisa dibuka? Atau nggak bisa karena kita sedang offline?

Jawabannya tentu bisa!

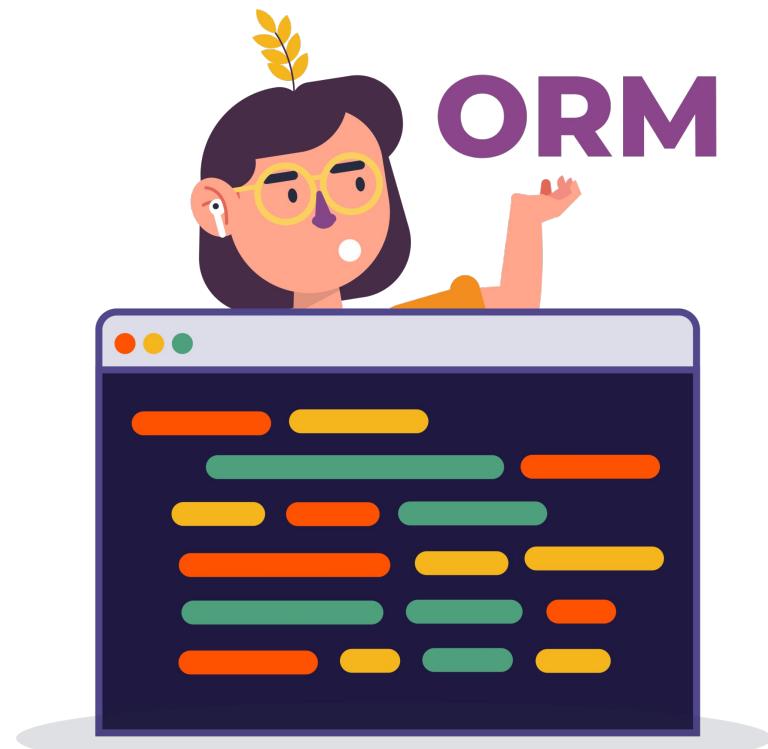
Mengapa? Karena ada sistem ORM yang ngebantu kita atur pesan-pesan WA kita tersimpan rapi antara dari server WA dengan penyimpanan lokal HP kita~

## Kenal lebih dekat dengan ORM

Object-Relational Mapping (ORM) adalah teknik yang memungkinkan kita **menggunakan query dan manipulasi data dari database menggunakan paradigma berorientasi object**.

Umumnya, pengaturan query dan penyimpanan data ini dilakukan di server dengan sistem bahasa SQL

Ketika membicarakan tentang ORM, kebanyakan orang merujuk ke **library yang mengimplementasikan teknik Object-Relational Mapping (ORM)**.

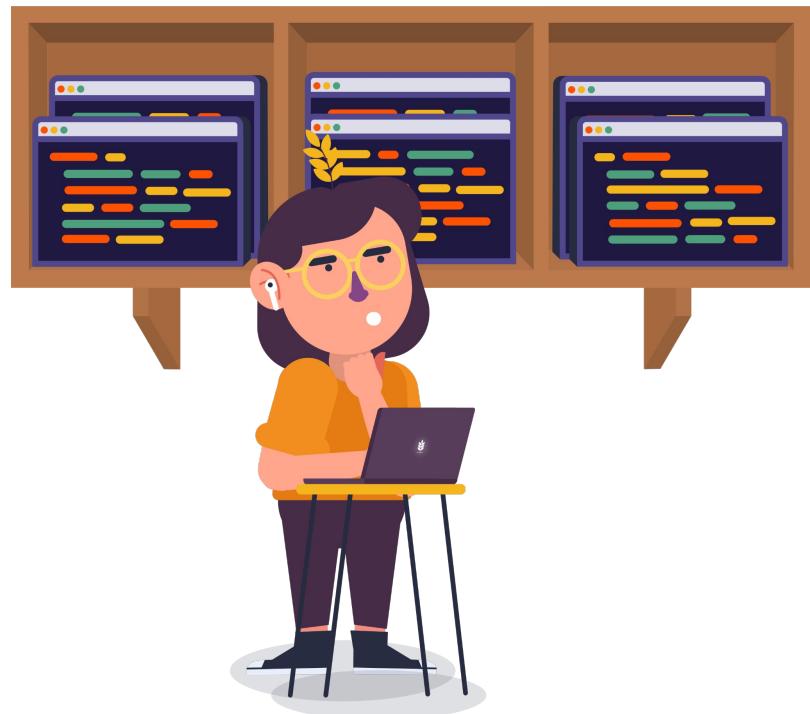


## Seperti apa Library ORM itu?

Library ORM adalah library yang benar-benar biasa ditulis dalam bahasa pilihan kita. Iya bahasa yang merangkum kode diperlukan untuk memanipulasi data.

Jadi, walaupun data-data yang digunakan dalam aplikasi diatur dalam sistem SQL, **kita nggak perlu lagi menggunakan bahasa pemrograman SQL untuk memanggil data tersebut.**

Nah, di topik ini kita bakal berinteraksi langsung dengan object dalam bahasa Kotlin~

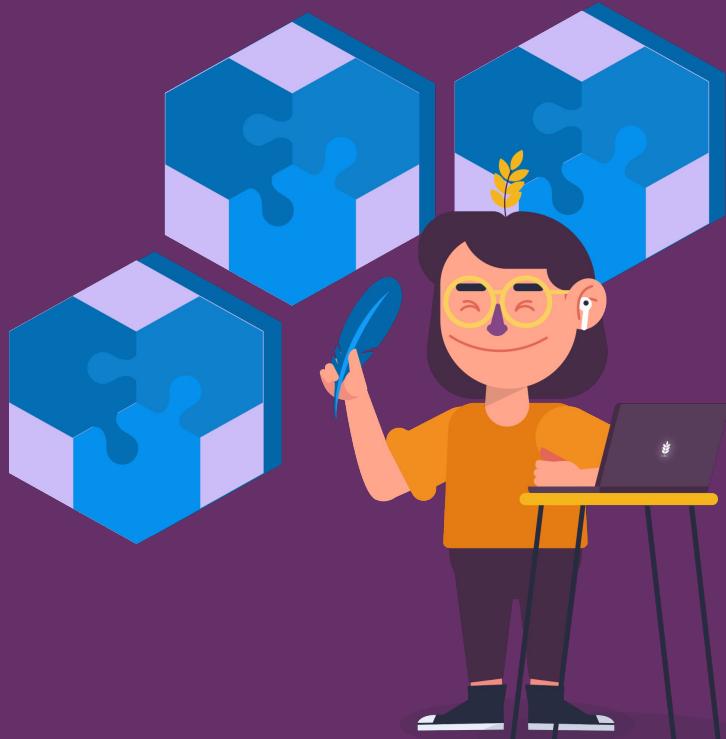




Voilà! Masih ngomongin library Android, salah satu library yang cukup terkenal menerapkan konsep ORM adalah ROOM.

Pada sesi setelah ini, kita akan perdalam apa itu ROOM, dan bagaimana cara menggunakannya.





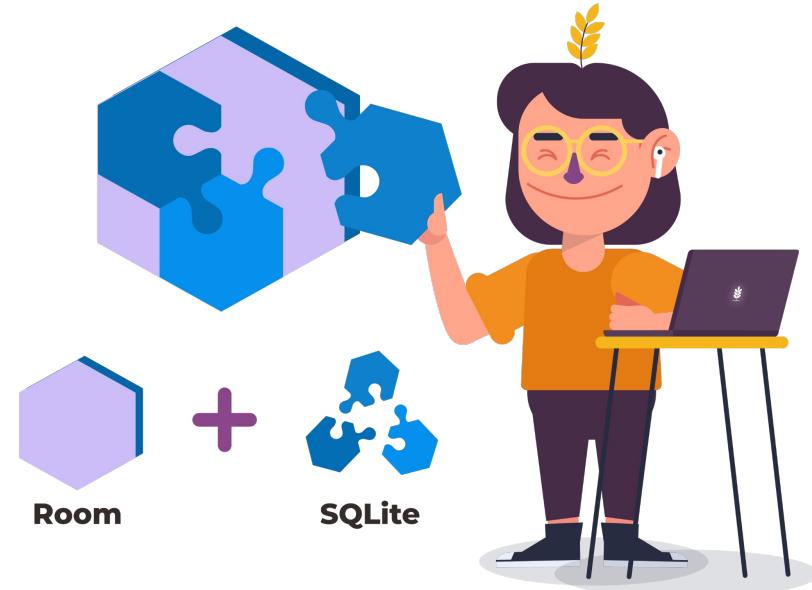
**Yes, betul. Abis ini kita bakal perdalam lagi nih pemahaman kita terhadap **ROOM**.**

**Inget ya jargon kita, tak kenal maka tak sayang~ Cuss lanjut ke bahasan berikutnya.**

## Apa itu Room?

Room adalah library andalan yang **menyediakan abstraction layer di atas SQLite** sehingga memungkinkan akses database yang lancar sambil memanfaatkan kekuatan penuh SQLite.

Room pada dasarnya adalah sebuah pembungkus di atas SQLite. Atau simpelnya gini deh.. di dalam sebuah ROOM digunakan teknologi SQLite.



Kenapa sih perlu mengenal ROOM? Jawabannya karena punya seribu satu khasiat. Berikut ini beberapa alasan kenapa kita pakai ROOM :

- Menawarkan pemeriksaan code sewaktu kompilasi.
- Cocok dengan LiveData, pemantauan langsung menggunakan LiveData.
- Mempermudah pengujian berbagai komponen di dalam Room.
- Mudah digunakan dan diimplementasikan.
- Mengurangi jumlah kode boilerplate.

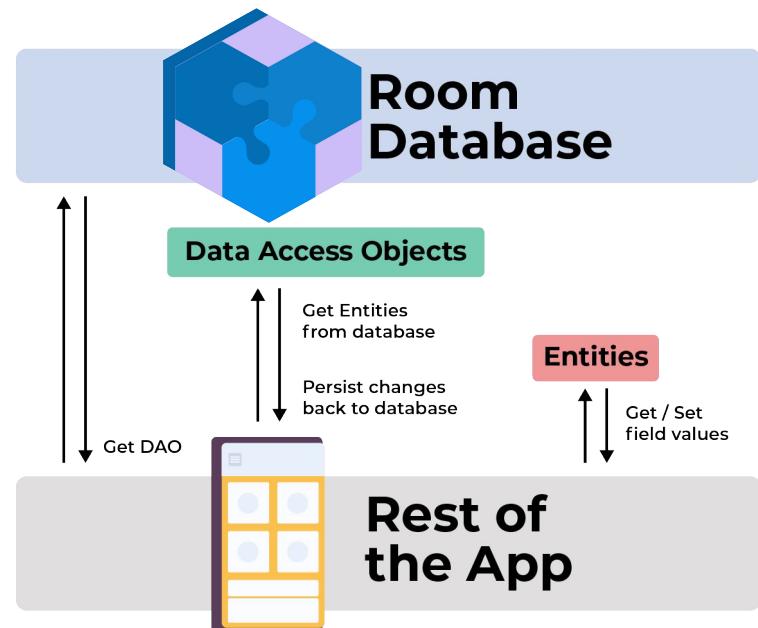




Kalau kamu sedari tadi udah melirik ke gambar disamping, kali ini kita bakal bahas itu. As you can see.. selain manfaat penggunaannya, ROOM juga punya komponen utama.

Coba kamu tebak dulu, kira-kira ada berapa komponen utama Room berdasarkan gambar disamping?

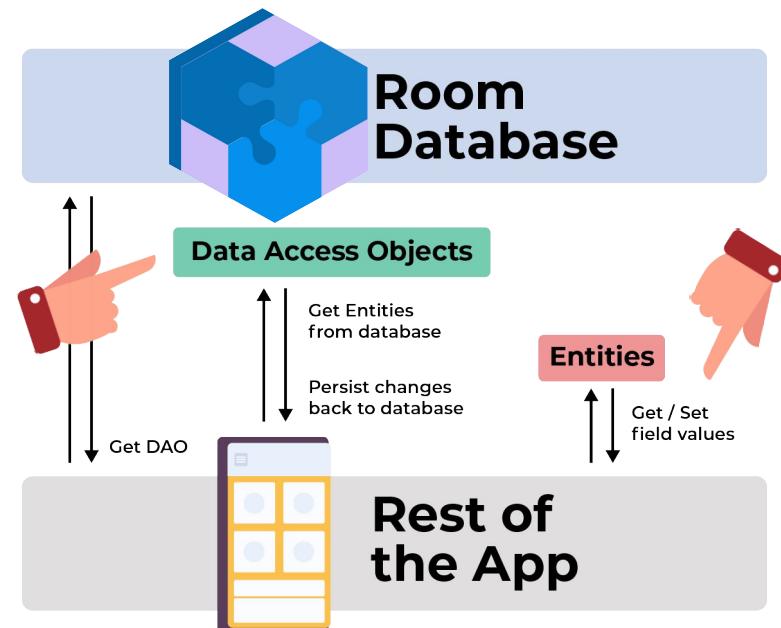
Jawabannya ada tiga komponen utama.



Pada dasarnya ada **3 komponen utama** di Room.

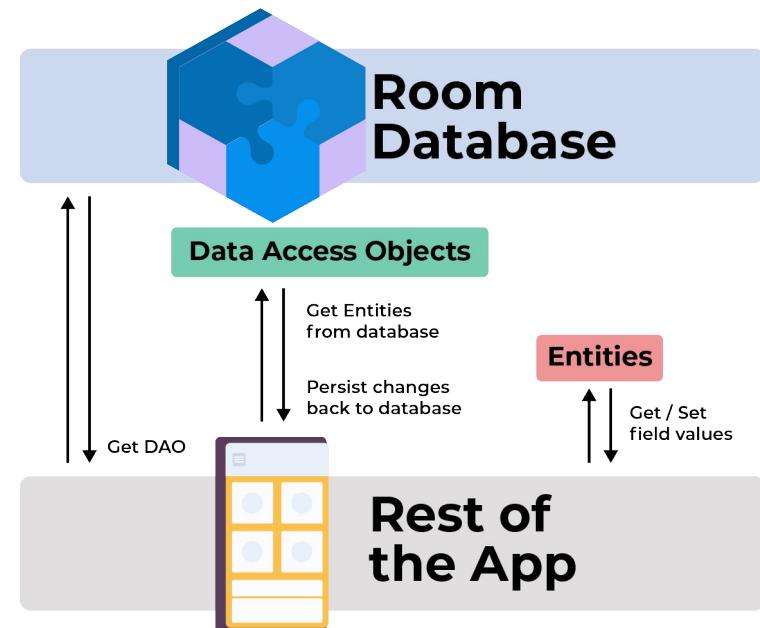
- **Entity** : Class Java atau Kotlin yang mewakili tabel dalam database.
- **DAO** : DAO adalah singkatan dari "DATA ACCESS OBJECT".

Ini pada dasarnya adalah interface, yang berisi method seperti `getData()` atau `storeData()` dll. Yang digunakan untuk **mengakses database**. Interface ini akan diimplementasikan oleh ROOM.



- **Database** : Ini adalah **class abstract yang meng-extends RoomDatabase**, ini adalah tempat kita mendefinisikan entitas (tabel) dan nomor versi dari database kita.

Ini berisi database dan berfungsi sebagai titik akses utama untuk koneksi yang mendasarinya.





**Widih, kereen~**

**Siapa yang tadi jawabannya betul semua?** 🙌

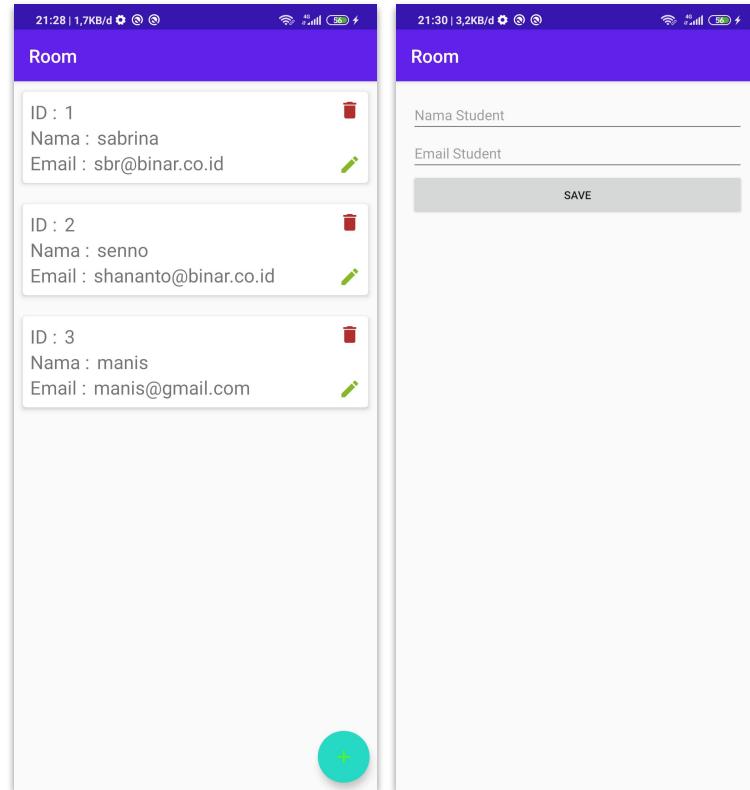
**Buat yang belum betul jangan sedih, karena abis ini kita coba recall sambil belajar cara menggunakan ROOM.**



### Biar Paham, kita bikin aplikasi yuk~

Kita bakalan bikin aplikasi yang memungkinkan CRUD (Create, Read, Update, dan Delete) terhadap database ROOM kita.

Ada delapan cara untuk bisa bikin aplikasi ini. Sambil kamu recall, sambil kamu lihat ya dalam proses pembuatannya akan melewati tiga komponen utama tadi.





## 1. Tambahkan Dependency ROOM

Pada file build.gradle (Module: app), tambahkan Dependency sehingga kurleb seperti pada gambar. Dalam koding file, kurang lebih dibutuhkan:

- **dependency Room** untuk mengolah Database Room,
- **Library Coroutines** untuk melakukan proses *Asynchronous* ketika mengolah database ROOM,
- **Library View tambahan** biar tampilannya enak dilihat.

Kalau udah beres, jangan lupa untuk klik **Sync Now** yang muncul pada bagian atas Editor lalu tunggu sampai proses Gradle Sync selesai.

```
apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-android-extensions'
apply plugin: 'kotlin-kapt'

android {
    //...
}

dependencies {
    implementation fileTree(dir: "libs", include: ["*.jar"])
    implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
    implementation 'androidx.core:core-ktx:1.1.0'
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'

    //lib coroutines
    implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.3.7'

    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'

    //Room Library
    implementation "androidx.arch.persistence.room:runtime:1.0.0"

    kapt "androidx.arch.persistence.room:compiler:1.0.0"

    //Library View Tambahan
    implementation 'androidx.recyclerview:recyclerview:1.1.0'
    implementation 'androidx.cardview:cardview:1.0.0'
    implementation 'com.google.android.material:material:1.1.0'
}
```



## 2. Membuat Class Entity

Sekarang, kita membutuhkan tabel untuk database kita dengan beberapa kolom tabel yang diperlukan.

Sebagai contoh kita membuat entity class bernama Student.kt sebagai Entity yang merepresentasikan spesifikasi dari Table Student.

```
@Entity  
@Parcelize  
data class Student(  
    @PrimaryKey(autoGenerate = true) var id: Int?,  
    @ColumnInfo(name = "nama") var nama: String,  
    @ColumnInfo(name = "email") var email: String  
) : Parcelable
```



Kalau kamu lihat di gambar, class ini punya beberapa annotation yang mungkin baru kita kenal seperti:

- **@Entity**

Menandai bahwa class ini adalah sebuah Entity yang akan diproses menjadi sebuah spesifikasi Table

- **@Parcelize**

Agar object class ini bisa dikirim melalui intent dengan parcelable. Jangan lupa extend juga ke class Parcelable ya~



```
@Entity  
@Parcelize  
data class Student(  
    @PrimaryKey(autoGenerate = true) var id: Int?,  
    @ColumnInfo(name = "nama") var nama: String,  
    @ColumnInfo(name = "email") var email: String  
) : Parcelable
```

- **@PrimaryKey**

Menandai atribut ini sebagai Primary Key pada Table. `autoGenerate = true` menandai primary key yang digunakan adalah sebuah Integer yang autoincrement (ID dimulai dari 1 untuk data pertama, 2 untuk data kedua, dan seterusnya).

- **@ColumnInfo**

Untuk menentukan informasi nama Kolom pada Table, diikuti oleh parameter name pada annotation tersebut.



```
@Entity  
@Parcelize  
data class Student(  
    @PrimaryKey(autoGenerate = true) var id: Int?,  
    @ColumnInfo(name = "nama") var nama: String,  
    @ColumnInfo(name = "email") var email: String  
) : Parcelable
```



Beberapa poin yang perlu diperhatikan:

- **Class harus dijelaskan pada Entity**

Ini adalah bagaimana ROOM mengidentifikasi setiap entitas yang kita buat sehingga tabel dibuat dengan Database terkait.

Secara default, ROOM membuat kolom untuk setiap *fields*, tetapi kita dapat menghindari ini untuk beberapa fields dengan menggunakan annotation `@ignore`.

```
● ● ●  
@Entity  
@Parcelize  
data class Student(  
    @PrimaryKey(autoGenerate = true) var id: Int?,  
    @ColumnInfo(name = "nama") var nama: String,  
    @ColumnInfo(name = "email") var email: String  
) : Parcelable
```



- **Setiap Entitas, harus menetapkan setidaknya 1 primary key.**  
Kita harus membuat anotasi fields dengan anotasi @PrimaryKey.
- **Secara default, Room menggunakan nama class sebagai nama tabel**  
Kita bisa memberi nama khusus dengan menggunakan properti **tableName**.

```
@Entity  
@Parcelize  
data class Student(  
    @PrimaryKey(autoGenerate = true) var id: Int?,  
    @ColumnInfo(name = "nama") var nama: String,  
    @ColumnInfo(name = "email") var email: String  
) : Parcelable
```



Hmm.. bisa dibilang, dalam membuat class Entity ini ada banyak hal yang bisa kita lakukan seperti relasi antar entitas atau membuat **nested object** atau beberapa *primary keys*.

```
@Entity  
@Parcelize  
data class Student(  
    @PrimaryKey(autoGenerate = true) var id: Int?,  
    @ColumnInfo(name = "nama") var nama: String,  
    @ColumnInfo(name = "email") var email: String  
) : Parcelable
```



### 3. Membuat DAO

Sekarang, kita membuat **DAO (Data Access Object)** yang akan digunakan untuk mengelola database.

Class ini berupa Class Interface. Pada Class ini, terdapat beberapa method abstract yang disesuaikan dengan kebutuhan kita.

Namun, setidaknya untuk kali ini kita buat method untuk melakukan CRUD (Create, Read, Update, dan Delete).



```
@Dao
interface StudentDao {
    @Query("SELECT * FROM Student")
    fun getAllStudent(): List<Student>

    @Insert(onConflict = REPLACE)
    fun insertStudent(student: Student): Long

    @Update
    fun updateStudent(student: Student): Int

    @Delete
    fun deleteStudent(student: Student): Int
}
```



Kuy kita ketahui gimana class ini bisa melakukan CRUD terhadap Table Student :

- **Read** :

Method `getAllStudent` yang memiliki return `List<Student>` akan mengembalikan daftar student dari hasil Query yang didefinisikan dalam annotation `@Query`.



```
@Dao
interface StudentDao {
    @Query("SELECT * FROM Student")
    fun getAllStudent(): List<Student>

    @Insert(onConflict = REPLACE)
    fun insertStudent(student: Student): Long

    @Update
    fun updateStudent(student: Student): Int

    @Delete
    fun deleteStudent(student: Student): Int
}
```



- **Create** :

Method `insertStudent` membutuhkan parameter object dari `Student` sebagai data yang akan diinput ke dalam Table. Me-return Long sebagai ID dari data yang baru diinput.

Ditandai dengan annotation `@Insert` dengan parameter `onConflict = REPLACE` untuk menangani jika terdapat ID yang sama, maka data baru akan menggantikan data yang lama.



```
@Dao
interface StudentDao {
    @Query("SELECT * FROM Student")
    fun getAllStudent(): List<Student>

    @Insert(onConflict = REPLACE)
    fun insertStudent(student: Student):Long

    @Update
    fun updateStudent(student: Student):Int

    @Delete
    fun deleteStudent(student: Student):Int
}
```



- **Update** :

Method `updateStudent`. Membutuhkan parameter object `Student` sebagai data baru yang akan diupdate.

Me-return `Int` sebagai jumlah data yang berhasil diubah. Ditandai dengan annotation `@Update`.



```
@Dao
interface StudentDao {
    @Query("SELECT * FROM Student")
    fun getAllStudent(): List<Student>

    @Insert(onConflict = REPLACE)
    fun insertStudent(student: Student):Long

    @Update
    fun updateStudent(student: Student):Int

    @Delete
    fun deleteStudent(student: Student):Int
}
```



- **Delete** :

Method `deleteStudent`. Membutuhkan object Student sebagai parameter untuk mengetahui data mana yang akan dihapus.

Me-return Int sebagai jumlah data yang berhasil dihapus. Method ini ditandai dengan annotation `@Delete`.



```
@Dao
interface StudentDao {
    @Query("SELECT * FROM Student")
    fun getAllStudent(): List<Student>

    @Insert(onConflict = REPLACE)
    fun insertStudent(student: Student):Long

    @Update
    fun updateStudent(student: Student):Int

    @Delete
    fun deleteStudent(student: Student):Int
}
```



Beberapa poin yang perlu diperhatikan dalam class DAO:

- Untuk berinteraksi atau mengatakan akses, kita **memerlukan interface yang dijelaskan dengan DAO**. Ini adalah cara agar fungsi Room berjalan.

Setiap DAO mencakup method yang menawarkan akses abstract ke database aplikasi kita.



```
@Dao
interface StudentDao {
    @Query("SELECT * FROM Student")
    fun getAllStudent(): List<Student>

    @Insert(onConflict = REPLACE)
    fun insertStudent(student: Student): Long

    @Update
    fun updateStudent(student: Student): Int

    @Delete
    fun deleteStudent(student: Student): Int
}
```



- Dengan mengakses database menggunakan DAO alih-alih membuat query atau request langsung, kita dapat **memisahkan berbagai komponen arsitektur database kita**.
- DAO memungkinkan kita untuk dengan mudah **memainkan akses database sebagai aplikasi pengujian kita**.



```
@Dao
interface StudentDao {
    @Query("SELECT * FROM Student")
    fun getAllStudent(): List<Student>

    @Insert(onConflict = REPLACE)
    fun insertStudent(student: Student): Long

    @Update
    fun updateStudent(student: Student): Int

    @Delete
    fun deleteStudent(student: Student): Int
}
```



- DAO dapat **berupa interface atau class abstract**. Dalam kasus class abstract, secara opsional dapat memiliki constructor yang mengambil RoomDatabase sebagai satu-satunya parameter.
- ROOM membuat setiap implementasi DAO pada waktu kompilasi.



```
@Dao
interface StudentDao {
    @Query("SELECT * FROM Student")
    fun getAllStudent(): List<Student>

    @Insert(onConflict = REPLACE)
    fun insertStudent(student: Student):Long

    @Update
    fun updateStudent(student: Student):Int

    @Delete
    fun deleteStudent(student: Student):Int
}
```



Kita dapat melakukan jauh lebih banyak dengan DAO, seperti mengirimkan parameter ke dalam query, mengembalikan subsets dari kolom, atau mengirimkan kumpulan argumen dan banyak lagi.



```
@Dao
interface StudentDao {
    @Query("SELECT * FROM Student")
    fun getAllStudent(): List<Student>

    @Insert(onConflict = REPLACE)
    fun insertStudent(student: Student): Long

    @Update
    fun updateStudent(student: Student): Int

    @Delete
    fun deleteStudent(student: Student): Int
}
```



### 4. Membuat Database Class

Akhirnya, kita membutuhkan DatabaseClass, yang ditandai dengan annotation @Database. Class Database **membuat pengelompokan logical antara interface DAO.**

```
@Database(entities = [Student::class],version = 1)
abstract class StudentDatabase: RoomDatabase() {
    abstract fun studentDao(): StudentDao

    companion object{
        private var INSTANCE: StudentDatabase? = null

        fun getInstance(context: Context): StudentDatabase? {
            if(INSTANCE == null){
                synchronized(StudentDatabase::class){
                    INSTANCE = Room.databaseBuilder(context.applicationContext,
                        StudentDatabase::class.java,"Student.db").build()
                }
            }
            return INSTANCE
        }

        fun destroyInstance(){
            INSTANCE = null
        }
    }
}
```



Class Ini juga mendefinisikan nomor versi yang diperlukan, yang digunakan untuk melacak dan mengimplementasikan migrasi database.

Pada annotation @Database, dibutuhkan parameter entities yang memuat Entity atau Table apa saja yang akan dimuat dalam Database. Parameter ini bisa berupa array.

Selain itu, juga terdapat parameter version untuk menetapkan nomor versi dari database

```
@Database(entities = [Student::class],version = 1)
abstract class StudentDatabase: RoomDatabase() {
    abstract fun studentDao(): StudentDao

    companion object{
        private var INSTANCE: StudentDatabase? = null

        fun getInstance(context: Context): StudentDatabase? {
            if(INSTANCE == null){
                synchronized(StudentDatabase::class){
                    INSTANCE = Room.databaseBuilder(context.applicationContext,
                        StudentDatabase::class.java,"Student.db").build()
                }
            }
            return INSTANCE
        }

        fun destroyInstance(){
            INSTANCE = null
        }
    }
}
```



Beberapa hal yang perlu diperhatikan class Database:

- Kita harus **mengikuti design pattern singleton** ketika membuat instance object AppDatabase.  
Maksudnya hanya ada 1 Object database yang akan digunakan berkali-kali selama aplikasi masih aktif.
- Class harus **abstract** dan **harus extend RoomDatabase**.

```
@Database(entities = [Student::class],version = 1)
abstract class StudentDatabase: RoomDatabase() {
    abstract fun studentDao(): StudentDao

    companion object{
        private var INSTANCE: StudentDatabase? = null

        fun getInstance(context: Context): StudentDatabase? {
            if(INSTANCE == null){
                synchronized(StudentDatabase::class){
                    INSTANCE = Room.databaseBuilder(context.applicationContext,
                        StudentDatabase::class.java,"Student.db").build()
                }
            }
            return INSTANCE
        }

        fun destroyInstance(){
            INSTANCE = null
        }
    }
}
```



Jika kita mencoba menjalankan dengan database yang dibuat di atas, aplikasi kita akan mengalami sedikit lag karena operasi yang dilakukan ada di **main thread**.

Secara default, ROOM melakukan pengecekan terhadap hal ini dan tidak mengizinkan operasi berada di main thread karena dapat membuat UI kita sedikit lag.

```
@Database(entities = [Student::class],version = 1)
abstract class StudentDatabase: RoomDatabase() {
    abstract fun studentDao(): StudentDao

    companion object{
        private var INSTANCE: StudentDatabase? = null

        fun getInstance(context: Context): StudentDatabase? {
            if(INSTANCE == null){
                synchronized(StudentDatabase::class){
                    INSTANCE = Room.databaseBuilder(context.applicationContext,
                        StudentDatabase::class.java,"Student.db").build()
                }
            }
            return INSTANCE
        }

        fun destroyInstance(){
            INSTANCE = null
        }
    }
}
```



Kita dapat menghindarinya dengan menggunakan `AsyncTask`, `Handler`, atau `Rxjava` dengan io scheduler atau opsi lain yang menempatkan operasi pada thread lainnya.

Namun pada project ini, kita akan menggunakan **Kotlin Coroutine** untuk menangani proses AsyncTask dalam mengelola database.

```
@Database(entities = [Student::class],version = 1)
abstract class StudentDatabase: RoomDatabase() {
    abstract fun studentDao(): StudentDao

    companion object{
        private var INSTANCE: StudentDatabase? = null

        fun getInstance(context: Context): StudentDatabase? {
            if(INSTANCE == null){
                synchronized(StudentDatabase::class){
                    INSTANCE = Room.databaseBuilder(context.applicationContext,
                        StudentDatabase::class.java,"Student.db").build()
                }
            }
            return INSTANCE
        }

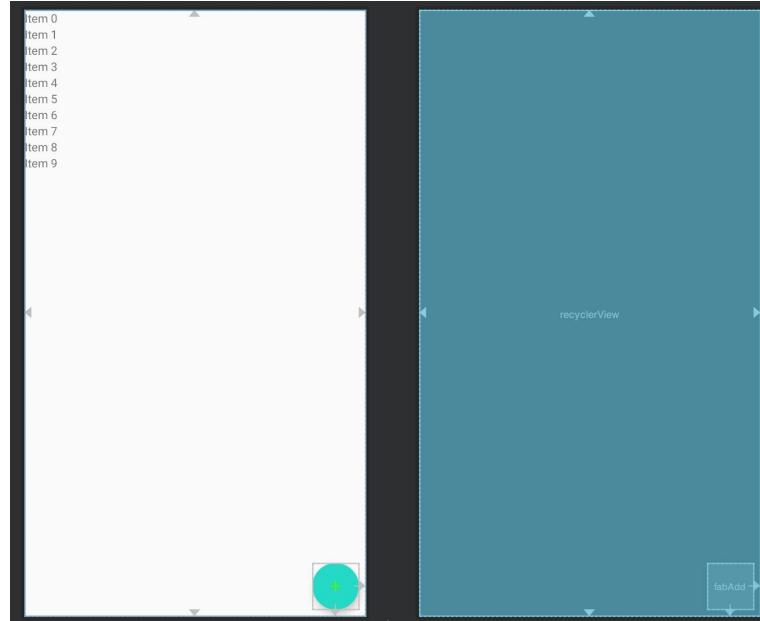
        fun destroyInstance(){
            INSTANCE = null
        }
    }
}
```



### 5. Desain Tampilan Main Activity

Desain Layout XML MainActivity kita sehingga desainnya sama seperti pada Gambar.

Kita hanya membutuhkan **RecyclerView** untuk **menampilkan daftar Students** dan **FloatingActionButton** untuk menambahkan data dengan berpindah ke Activity lainnya.





## 6. Membuat Adapter

Sebelum coding Main Activity, kita siapkan dulu Adapter untuk menampilkan daftar data student ke RecyclerView.

Hayoo.. siapa yang masih inget sama materi **RecyclerView** yang udah kita bahas?

```
class StudentAdapter(val listStudent : List<Student>) : RecyclerView.Adapter<StudentAdapter.ViewHolder>() {  
    class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView)  
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {  
        val itemView = LayoutInflater.from(parent.context)  
            .inflate(R.layout.student_item,parent,false)  
        return ViewHolder(itemView)  
    }  
    override fun getItemCount(): Int {  
        return listStudent.size  
    }  
    override fun onBindViewHolder(holder: ViewHolder, position: Int) {  
        holder.itemView.tvID.text = listStudent[position].id.toString()  
        holder.itemView.tvNama.text = listStudent[position].nama  
        holder.itemView.tvEmail.text = listStudent[position].email  
        holder.itemView.ivEdit.setOnClickListener {  
            val intentKeEditActivity = Intent(it.context,  
                EditActivity::class.java)  
            intentKeEditActivity.putExtra("student",listStudent[position])  
            it.context.startActivity(intentKeEditActivity)  
        }  
        holder.itemView.ivDelete.setOnClickListener {  
            AlertDialog.Builder(it.context).setPositiveButton("Ya") { p0, p1 ->  
                val mDb = StudentDatabase.getInstance(holder.itemView.context)  
                GlobalScope.async {  
                    val result = mDb?.studentDao()?.deleteStudent(listStudent[position])  
                    (holder.itemView.context as MainActivity).runOnUiThread {  
                        if (result!=0){  
                            Toast.makeText(it.context,"Data ${listStudent[position].nama} berhasil  
dihapus",Toast.LENGTH_LONG).show()  
                        }else{  
                            Toast.makeText(it.context,"Data ${listStudent[position].nama} Gagal  
dihapus",Toast.LENGTH_LONG).show()  
                        }  
                    }  
                    (holder.itemView.context as MainActivity).fetchData()  
                }  
            }.setNegativeButton("Tidak")  
            { p0, p1 ->  
                p0.dismiss()  
            }  
            .setMessage("Apakah Anda Yakin ingin menghapus data  
${listStudent[position].nama}").setTitle("Konfirmasi Hapus").create().show()  
        }  
    }  
}
```



Yepp, pada adapter ini kita juga set aksi ketika menekan tombol Edit agar berpindah ke Activity yang berfungsi untuk mengupdate data.

Nggak hanya itu, kita set juga tombol Delete agar menampilkan dialog konfirmasi dan melakukan aksi penghapusan data.

```
class StudentAdapter(val listStudent : List<Student>) : RecyclerView.Adapter<StudentAdapter.ViewHolder>() {  
  
    class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView)  
  
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {  
        val itemView = LayoutInflater.from(parent.context)  
            .inflate(R.layout.student_item,parent,false)  
        return ViewHolder(itemView)  
    }  
  
    override fun getItemCount(): Int {  
        return listStudent.size  
    }  
  
    override fun onBindViewHolder(holder: ViewHolder, position: Int) {  
        holder.itemView.tvID.text = listStudent[position].id.toString()  
        holder.itemView.tvNama.text = listStudent[position].nama  
        holder.itemView.tvEmail.text = listStudent[position].email  
  
        holder.itemView.ivEdit.setOnClickListener {  
            val intentKeEditActivity = Intent(it.context,  
                EditActivity::class.java)  
  
            intentKeEditActivity.putExtra("student",listStudent[position])  
            it.context.startActivity(intentKeEditActivity)  
        }  
  
        holder.itemView.ivDelete.setOnClickListener {  
            AlertDialog.Builder(it.context).setPositiveButton("Ya") { p0, p1 ->  
                val mDb = StudentDatabase.getInstance(holder.itemView.context)  
  
                GlobalScope.async {  
                    val result = mDb?.studentDao()?.deleteStudent(listStudent[position])  
  
                    (holder.itemView.context as MainActivity).runOnUiThread {  
                        if (result!=0){  
                            Toast.makeText(it.context,"Data ${listStudent[position].nama} berhasil  
dihapus",Toast.LENGTH_LONG).show()  
                        }else{  
                            Toast.makeText(it.context,"Data ${listStudent[position].nama} Gagal  
dihapus",Toast.LENGTH_LONG).show()  
                        }  
                    }  
  
                    (holder.itemView.context as MainActivity).fetchData()  
                }  
            }.setNegativeButton("Tidak")  
            { p0, p1 ->  
                p0.dismiss()  
            }  
            .setMessage("Apakah Anda Yakin ingin menghapus data  
${listStudent[position].nama}").setTitle("Konfirmasi Hapus").create().show()  
        }  
    }  
}
```



### 7. Coding Class ‘Main Activity’

Pada class Main Activity, kita membuat code untuk mendapatkan semua data yang ada di Table Student dalam database lalu menampilkannya daftar student tersebut dengan RecyclerView.

Jangan lupa juga untuk menambahkan aksi intent pada FloatingActionButton untuk berpindah ke Activity lain yang bisa menambahkan data.

```
● ● ●

class MainActivity : AppCompatActivity() {
    private var mDB : StudentDatabase? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        mDB = StudentDatabase.getInstance(this)
        recyclerView.layoutManager = LinearLayoutManager(this, LinearLayoutManager.VERTICAL, false)
        fetchData()
        fabAdd.setOnClickListener {
            val keActivityAdd = Intent(this, AddActivity::class.java)
            startActivity(keActivityAdd)
        }
    }

    override fun onResume() {
        super.onResume()
        fetchData()
    }

    fun fetchData(){
        GlobalScope.launch {
            val listStudent = mDB?.studentDao()?.getAllStudent()

            runOnUiThread{
                listStudent?.let {
                    val adapter = StudentAdapter(it)
                    recyclerView.adapter = adapter
                }
            }
        }
    }

    override fun onDestroy() {
        super.onDestroy()
        StudentDatabase.destroyInstance()
    }
}
```



## 8. Coding Class 'Add Activity'

Pada class **Add Activity**, kita membuat code untuk menambahkan data ke Database ROOM.

Dalam proses insert-nya, kita menggunakan Coroutine dengan syntax **GlobalScope.async** untuk menjalankan proses Asynchronous agar aplikasi nggak nge-lag.

```
● ● ●

class AddActivity : AppCompatActivity() {

    var mDb: StudentDatabase? = null
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_add)

        mDb = StudentDatabase.getInstance(this)

        btnSave.setOnClickListener {
            val objectStudent = Student(
                null,
                etNamaStudent.text.toString(),
                etEmailStudent.text.toString()
            )

            GlobalScope.async {
                val result = mDb?.studentDao()?.insertStudent(objectStudent)
                runOnUiThread {
                    if(result != 0.toLong()){
                        //sukses
                        Toast.makeText(this@AddActivity,"Sukses menambahkan
${objectStudent.nama}",Toast.LENGTH_LONG).show()
                    }else{
                        //gagal
                        Toast.makeText(this@AddActivity,"Gagal menambahkan
${objectStudent.nama}",Toast.LENGTH_LONG).show()
                    }
                    finish()
                }
            }
        }
    }
}
```



### 9. Coding Class ‘Edit Activity’

Pada class Edit Activity, kita membuat code untuk menerima object Student yang akan diedit melalui intent.

Lalu menggunakan data tersebut sebagai patokan data Student mana yang akan diubah datanya.

Mirip dengan proses insert, kita menggunakan Coroutine dengan syntax GlobalScope.async untuk menjalankan proses Asynchronous agar aplikasi nggak nge-lag.

```
class EditActivity : AppCompatActivity() {
    var mDb: StudentDatabase? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_edit)

        mDb = StudentDatabase.getInstance(this)

        val objectStudent = intent.getParcelableExtra<Student>("student")

        etNamaStudent.setText(objectStudent.nama)
        etEmailStudent.setText(objectStudent.email)

        btnSave.setOnClickListener {
            objectStudent.nama = etNamaStudent.text.toString()
            objectStudent.email = etEmailStudent.text.toString()

            GlobalScope.async {
                val result = mDb?.studentDao()?.updateStudent(objectStudent)

                runOnUiThread {
                    if(result!=0){
                        Toast.makeText(this@EditActivity,"Sukses mengubah ${objectStudent.nama}",
                        Toast.LENGTH_LONG).show()
                    }else{
                        Toast.makeText(this@EditActivity,"Gagal mengubah ${objectStudent.nama}",
                        Toast.LENGTH_LONG).show()
                    }
                }

                finish()
            }
        }
    }
}
```



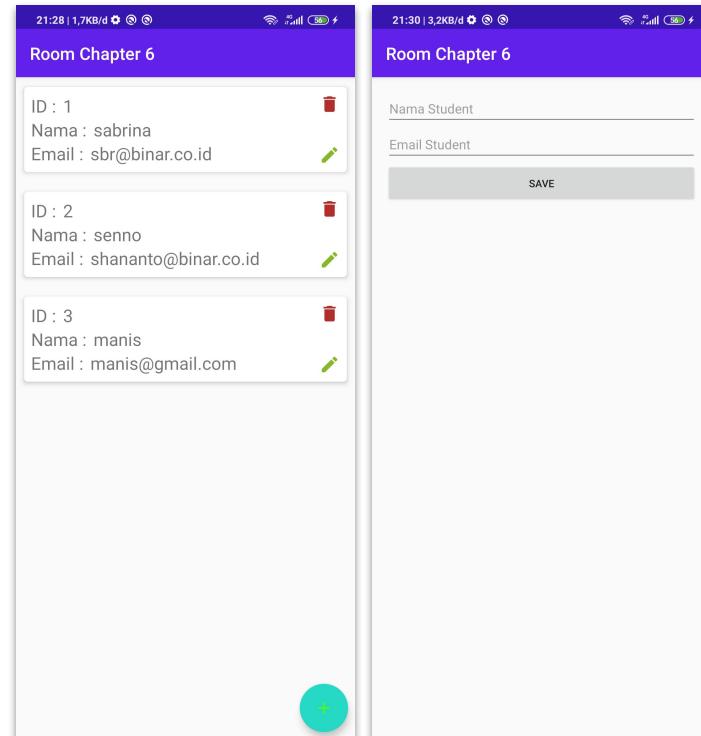
# Selesai!

Yeay! Sudah selesai aktivitas melakukan CRUD pada database kita~

Silakan untuk dicoba melakukan operasi CRUD (Create, Read, Update, dan Delete)!!.

Eh bentar, ada yang ketinggalan? Ternyata error? Relax... Code secara keseluruhan, bisa kamu lihat di :

<https://github.com/sennohananto/Room-Chapter-6>



The image displays two side-by-side screenshots of a mobile application interface titled "Room Chapter 6".

**Left Screenshot (Data View):** Shows a list of three student records. Each record includes an ID, name, and email, along with edit and delete icons.

| ID | Nama    | Email                |
|----|---------|----------------------|
| 1  | sabrina | sbr@binar.co.id      |
| 2  | senno   | shananto@binar.co.id |
| 3  | manis   | manis@gmail.com      |

**Right Screenshot (Form View):** Shows a form for adding a new student. It has fields for "Nama Student" and "Email Student", and a "SAVE" button.



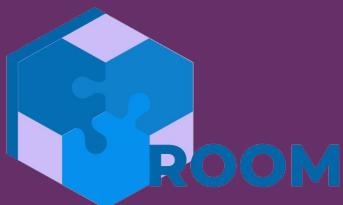
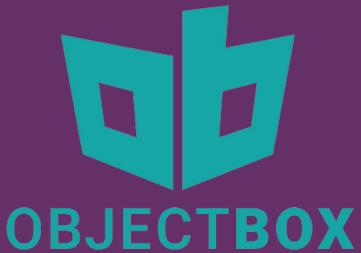
### Referensi Tambahan nih!

Masih penasaran dengan library ROOM?

Silahkan cek video berikut untuk memperkaya skill dan pengetahuan kita~



[Klik disini untuk Link Video](#)



Mantep niih, kita udah bisa pakai Room~

Eitss.. tunggu dulu, usut punya usut Library ORM itu nggak cuma Room aja loh!

Kira-kira ada apa lagi ya di Library ORM itu?



## Ada dua library yang cukup populer untuk ORM selain Room~

Di antaranya yang cukup terkenal adalah:

- ObjectBox
- REALM

Lantas apa saja sih yang membedakan mereka? Dan idealnya yang mana nih yang digunakan untuk menerapkan ORM? Yuk kita simak di pembahasan berikutnya~

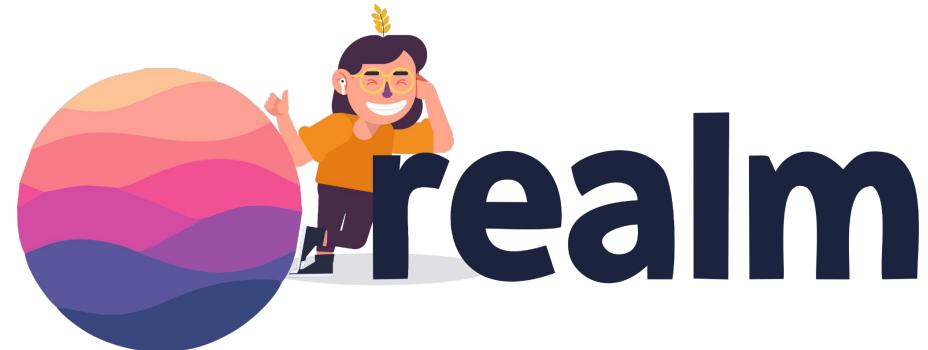


## Library 'Realm'

Realm adalah library dengan sistem basis data non-relasional pada mobile. Library ini menyimpan objek Java langsung ke dalam memory (dalam format khusus agar memori optimal).

Sejak 2011, Realm sudah menjadi pilihan utama developers.

Library ini menawarkan simplicitas, membutuhkan sedikit usaha untuk persiapan, dan punya dokumentasi yang baik dalam pengembangan dan penggunaannya.

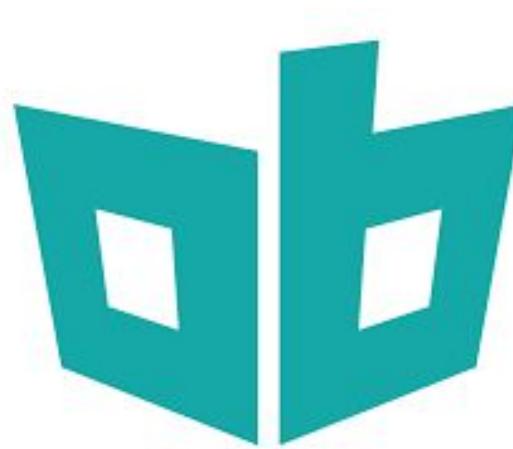




### Library 'Object Box'

Library ini terbilang baru dibandingkan library lainnya.

Dibuat oleh GreenRobot, library ini menggunakan database mobile NoSQL yang telah dioptimalkan kinerjanya.





Beberapa keuntungan yang dimiliki library ini:

- Menyimpan object yang kompleks membutuhkan sedikit usaha.
- Memiliki fungsi bawaan untuk memilih dan menyortir data
- Menawarkan kecepatan dan performa yang luar biasa, bahkan melebihi library lain.





|                     | ROOM  | REALM  | ObjectBox   |
|---------------------|---|--|---|
| Penjelasan          | Menyediakan abstraction layer di atas SQLite, dan merupakan dari bagian dari Android Jetpack, jadi sudah jelas ini dibuat oleh Google Developers. | Sistem basis data non-relasional pada mobile. Menyimpan objek Java langsung ke dalam memory (dalam format khusus agar memori optimal). | Library yang menggunakan database mobile NoSQL yang telah dioptimalkan kinerjanya. Dibuat oleh GreenRobot |
| Support oleh Google | Disupport langsung oleh Google  | Tidak disupport oleh Google  | Tidak disupport oleh Google (cont.)   |

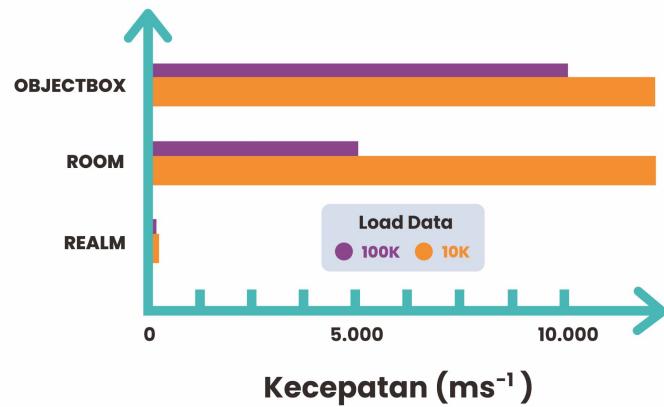


|  | ROOM   | REALM   | ObjectBox   |
|--|--|---|---|
| Menyimpan object yang kompleks               | Lebih mudah saat bekerja dengan data yang sederhana, tetapi membutuhkan waktu tambahan untuk mengimplementasikan Relasi Antar Object (RAT) | Menyimpan object yang kompleks membutuhkan sedikit usaha. | Menyimpan object yang kompleks membutuhkan sedikit usaha. |
| Mengambil data yang banyak di segala kondisi | Toolkit yang hebat untuk pembuatan multi-level query. Membutuhkan pemahaman tentang SQL.   | Memiliki fungsi bawaan untuk memilih dan menyortir data.  | Memiliki fungsi bawaan untuk memilih dan menyortir data.  |
| Menambah beban aplikasi                      | 50KB   | 3-4MB tergantung dari arsitektur handphone.               | 3-4MB tergantung dari arsitektur handphone.               |

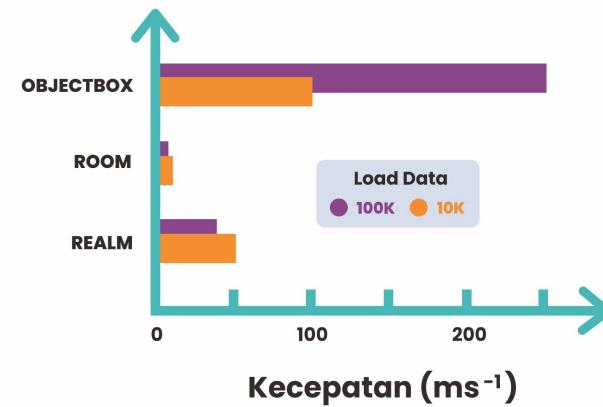
Kita coba lihat tabel perbandingan dari speed yang diberikan, ketika menggunakan ketiga library tadi.

Pada bagian sebelah kiri merupakan waktu (ms) yang digunakan.

## ACCESS



## INSERT



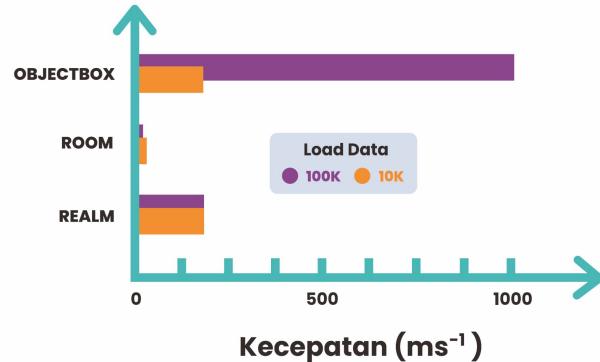
[References Link](#)

Hasilnya menarik bukan?

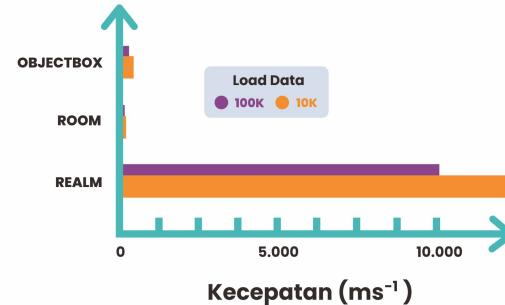
Dengan uji coba CRUD, diketahui bahwa ObjectBox berhasil memenangkan kompetisi!

Meski terbilang baru, ObjectBox menjadi yang tercepat dibandingkan library Realm dan ROOM.

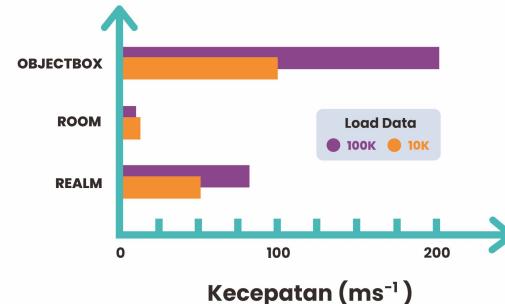
## DELETE



## LOAD



## UPDATE





### Jadi, Library mana yang terbaik?

Kita dapat melihat bahwa library manapun yang kita pilih untuk digunakan, masing-masing memiliki kelebihannya masing-masing.

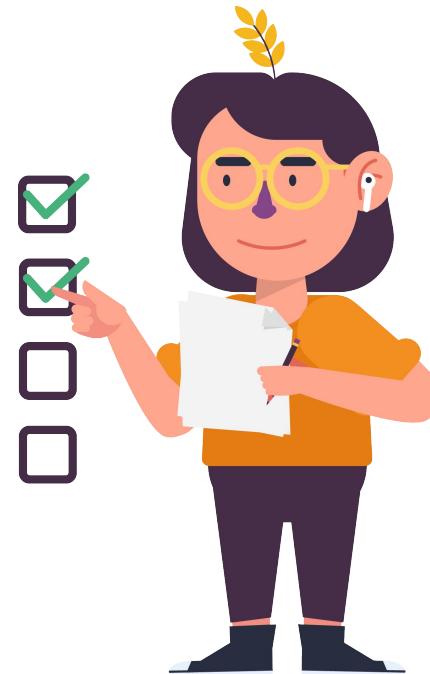
Jika kita mencari yang **cepat dan efisiensi**, **ObjectBox** adalah jawabannya.

Namun, jika kita dibatasi oleh **ukuran aplikasi**, tentu saja kita akan berurusan dengan SQL dan jawabannya sudah jelas adalah menggunakan **Room**.





Di sisi lain, **Realm** mungkin bukanlah yang tercepat atau yang terkecil dari segi library, tetapi mereka **menawarkan solusi yaitu kestabilan library, bebas bug** yang sudah terbukti lebih dari 7 tahun dan improvement yang mereka lakukan tanpa kita sadari.



# Saatnya kita Quiz!





## 1. Berikut ini adalah beberapa pernyataan yang tepat terkait ORM, yaitu...

- A. Merupakan teknik yang memungkinkan kita memanipulasi data dari query dalam Android Studio
- B. Memiliki beberapa library yang bisa digunakan, seperti ROOM, PandaSQL, dan ObjectBox
- C. Membuat kita mampu mengubah display layout data dari portrait menjadi landscape, begitu juga sebaliknya



## 1. Berikut ini adalah beberapa pernyataan yang tepat terkait ORM, yaitu...

- A. Merupakan teknik yang memungkinkan kita memanipulasi data dari query dalam Android Studio
- B. Memiliki beberapa library yang bisa digunakan, seperti ROOM, PandaSQL, dan ObjectBox
- C. Membuat kita mampu mengubah display layout data dari portrait menjadi landscape, begitu juga sebaliknya

**Object-Relational Mapping (ORM) adalah teknik yang memungkinkan kita menggunakan query dan memanipulasi data dari database menggunakan paradigma berorientasi object.**

**Ketika membicarakan tentang ORM, kebanyakan orang merujuk ke library yang mengimplementasikan teknik Object-Relational Mapping (ORM).**



- 2. Library yang menyediakan abstraction layer di atas SQLite untuk memungkinkan akses database yang lancar sambil memanfaatkan kekuatan penuh SQLite ...**
- A. ORM
  - B. ROOM
  - C. SQLite



**2. Library yang menyediakan abstraction layer di atas SQLite untuk memungkinkan akses database yang lancar sambil memanfaatkan kekuatan penuh SQLite ...**

- A. ORM
- B. ROOM
- C. SQLite

**Betul!. Jawabannya Room.**

**Selain Room, juga ada library DAO lainnya yang bisa kita coba, yaitu :  
GreenDAO, Realm, dan ObjectBOX.**



### 3. Berikut adalah komponen-komponen utama dalam room, kecuali...

- A. Entity, DAO, SQLite
- B. SQLite, Entity, Database
- C. Entity, DAO, Database



### 3. Berikut adalah komponen-komponen utama dalam room, kecuali...

- A. Entity, DAO, SQLite
- B. SQLite, Entity, Database
- C. Entity, DAO, Database

**Benar!** Entity adalah Class Java atau Kotlin yang mewakili tabel dalam database.  
DAO adalah interface yang berisi method untuk mengakses database.  
Database adalah class abstract yang meng-extends RoomDatabase,



#### **4. Jika dibandingkan dengan library, kelebihan ROOM adalah dibawah ini, kecuali ...**

- A. Memiliki kecepatan pemrosesan CRUD terbaik
- B. Merupakan library resmi yang di support oleh Google
- C. Membutuhkan kapasitas memori lebih kecil dibanding library lainnya



#### 4. Jika dibandingkan dengan library, kelebihan library Room adalah dibawah ini, kecuali ...

- A. Memiliki kecepatan pemrosesan CRUD terbaik
- B. Merupakan library resmi yang di support oleh Google
- C. Membutuhkan kapasitas memori lebih kecil dibanding library lainnya

Benar!. Jawabannya adalah A.

Meski disupport penuh oleh Google, beberapa library rekanannya ada yang jauh lebih cepat loh! Dengan catatan membutuhkan memori lebih besar dari Room



## 5. Paradigma apakah yang digunakan pada Object-Relational Mapping (ORM)?

- A. Orientasi Object
- B. Tidak Menggunakan Paradigma
- C. Struktural



## 5. Paradigma apakah yang digunakan pada Object-Relational Mapping (ORM)?

- A. Orientasi Object
- B. Tidak Menggunakan Paradigma
- C. Struktural

**Benar!** Paradigma yang digunakan dalam ORM adalah Paradigma Berorientasi Object, yang mana struktur database dan table dalam Room diproyeksikan sedemikian rupa dengan pendekatan Class dan Object.

## Referensi dan bacaan lebih lanjut~

1. [What is an ORM, how does it work, and how should I use one? - Stack Overflow](#)
2. [An Introduction to Android ROOM](#)
3. [Android Architecture Components — Room and Kotlin | by ritesh singh | MindOrks | Medium](#)
4. [Using Room Database | Android Jetpack | by Ashish Rawat | MindOrks | Medium](#)
5. [Save data in a local database using Room | Android Developers](#)
6. [Pengantar Room dan Flow](#)
7. [Realm, ObjectBox or Room. Which one is for you? | by Rado Yankov | Dev Labs](#)





**Nah, selesai sudah pembahasan kita di Chapter 4 Topic 5 ini.**

**Daan, Topik ini juga mengakhiri bahasan Chapter 4 kita.**

**Selanjutnya, kita akan bahas Challenge yang akan kita kerjakan sebagai penutup Chapter ini~**



# Terima Kasih!



Chapter ✓

completed