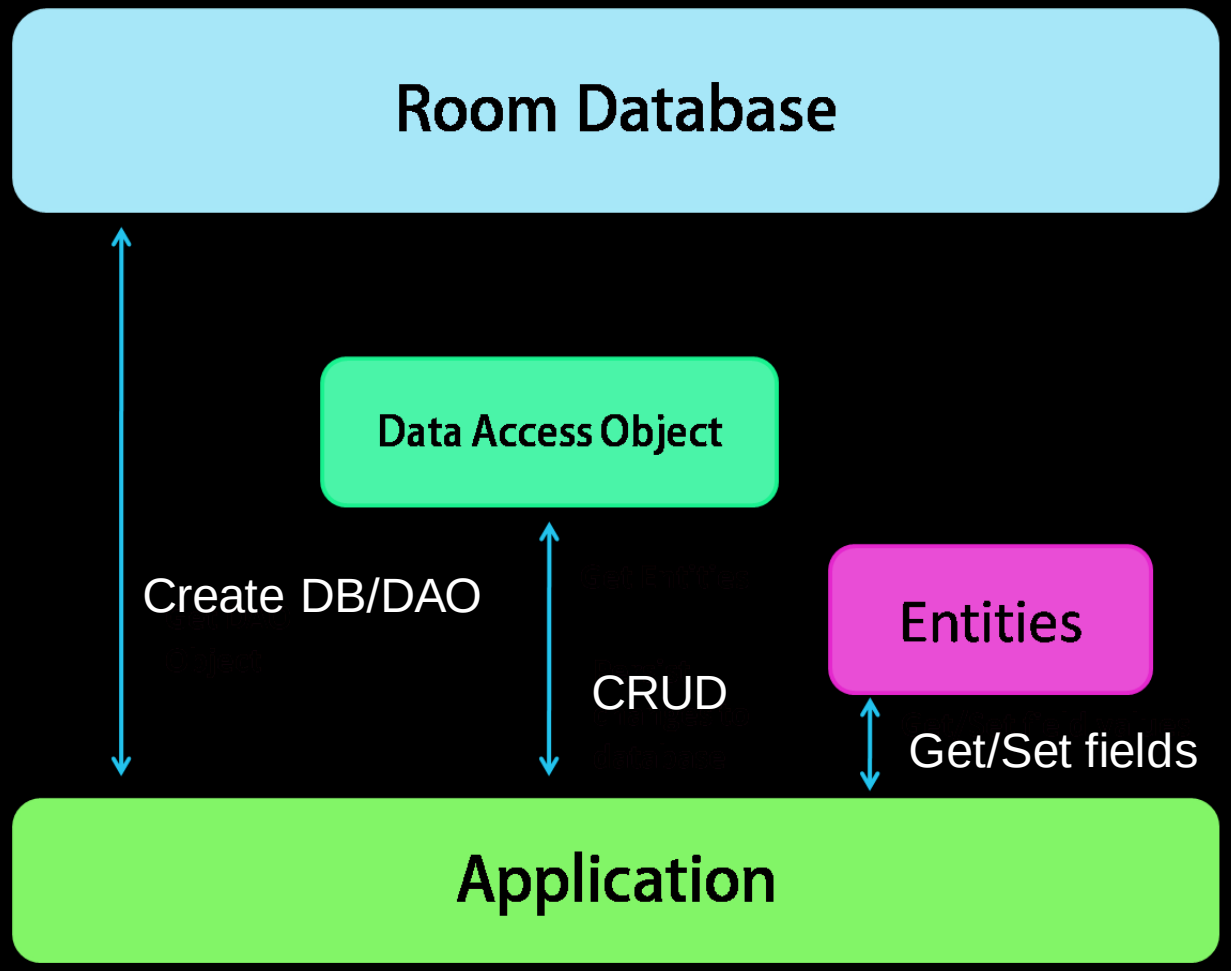
Datenpersistenz in Android mit Room – ein Einstieg.

Johannes Quast

25. Mai 2021

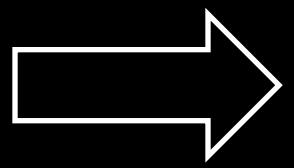
High-Level Architektur



Entities

Entities

```
data class LocalUser(
    val userId: Long,
    val username: String,
    val firstName: String,
    val lastName: String,
    var lastLogin: Long
)
```



```
@Entity
data class LocalUser(
    val userId: Long,
    val username: String,
    val firstName: String,
    val lastName: String,
    var lastLogin: Long
)
```

Entitys – Primärschlüssel

```
@Entity
data class LocalUser(
    @PrimaryKey(autoGenerate = true)
    val userId: Long,
    val remoteUserId: Long,
    val username: String,
    val endpoint: String,
    var lastLogin: Long
)
```

@PrimaryKey erlaubt es,
einen oder mehrere Primärschlüssel
festzulegen.

Entitys – Column Modifiers

```
@Entity
data class LocalUser(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "user_id")
    val userId: Long,

    @ColumnInfo(index = true)
    val username: String,

    val firstName: String,
    val lastName: String,

    @ColumnInfo(name = "last_login")
    var lastLogin: Long
)
```

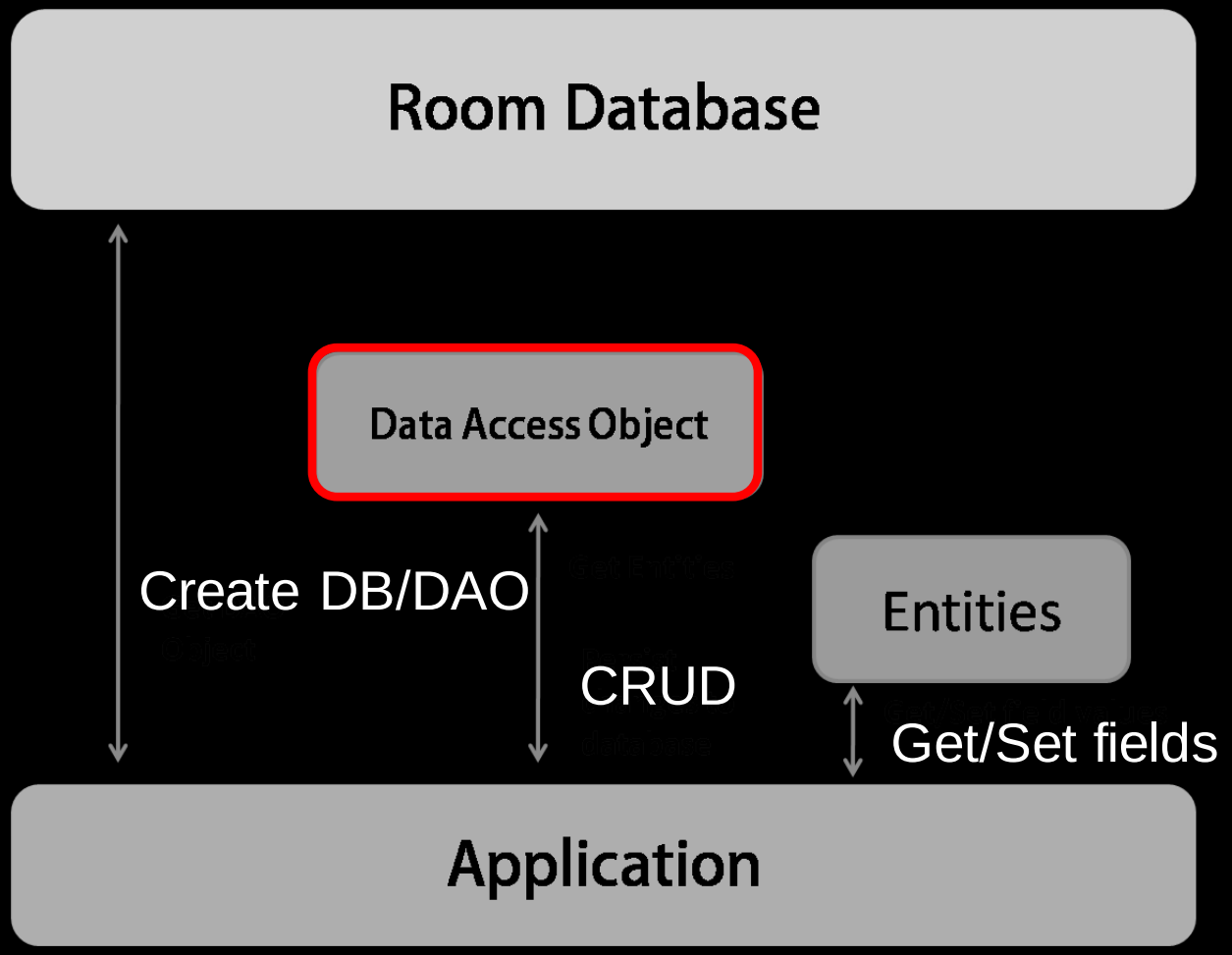
@ColumnInfo ermöglicht es, Spalten anzupassen.

Entitys – Alternative über @Entity

```
@Entity(
    primaryKeys = [
        "user_id"
    ],
    indices = [
        Index("username")
    ]
)
data class LocalUser(
    @ColumnInfo(name = "user_id")

    val userId: Long,
    val username: String,
    val firstName: String,
    val lastName: String,

    @ColumnInfo(name = "last_login")
    var lastLogin: Long
)
```

@Entity kann Dinge auch übersichtlicher gestalten.

DAOs


Room Database
Data Access Object
Create DB/DAO
CRUD
Entities
Get/Set fields
Application

DAO

```
@Dao
interface LocalUserDao {
    ...
}
```

Hinweis für Room, dass es sich hierbei um ein DAO handelt.

Interface für alle CRUD Operationen.

Liste von **abstrakten Methoden**.

DAO am Beispiel der User-Tabelle

```
@Dao
interface LocalUserDao {
    fun getAllUsers(): List<LocalUser>
    fun createUser(localUser: LocalUser)
    fun updateUser(localUser: LocalUser)
    fun deleteUser(localUser: LocalUser)
}
```

DAO am Beispiel der User-Tabelle

```
@Dao
interface LocalUserDao {

    @Query("SELECT * FROM LocalUser")
    fun getAllUsers(): List<LocalUser>

    @Insert
    fun createUser(localUser: LocalUser)

    @Update
    fun updateUser(localUser: LocalUser)

    @Delete
    fun deleteUser(localUser: LocalUser)
}
```

DAO am Beispiel der User-Tabelle

```
@Dao
interface LocalUserDao {
    ...

    @Query("SELECT * FROM LocalUser WHERE user_id = (:id)")
    fun getUserById(id: Long): LocalUser?
}
```

DAO: Einschub zu LiveData

```
@Dao
interface LocalUserDao {

    @Query("SELECT * FROM LocalUser")
    fun getAllUsers(): List<LocalUser>

    @Insert
    fun createUser(localUser: LocalUser)

    @Update
    fun updateUser(localUser: LocalUser)

    @Delete
    fun deleteUser(localUser: LocalUser)
}
```

DAO: Einschub zu LiveData

```
@Dao
interface LocalUserDao {

    @Query("SELECT * FROM LocalUser")
    fun getAllUsers(): LiveData<List<LocalUser>>

    @Insert
    fun createUser(localUser: LocalUser)

    @Update
    fun updateUser(localUser: LocalUser)

    @Delete
    fun deleteUser(localUser: LocalUser)
}
```

```
userDao.getAllUsers().observe(<livecycleOwner>) {
    userListe ->
        // Verarbeite aktualisierte Liste von Usern
}
```

DAO: Einschub zu LiveData – Negativbeispiel

```
@Dao
interface LocalUserDao {
    ...

    @Query("SELECT * FROM LocalUser WHERE user_id = (:id)")
    fun getUserById(id: Long): LiveData<LocalUser?>
}
```

Beziehungen

1:n Beziehung zwischen einem User und vielen Dateien

```
@Entity
data class LocalUser(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "user_id")
    val userId: Long,

    @ColumnInfo(index = true)
    val username: String,

    val firstName: String,
    val lastName: String,

    @ColumnInfo(name = "last_login")
    var lastLogin: Long
)
```

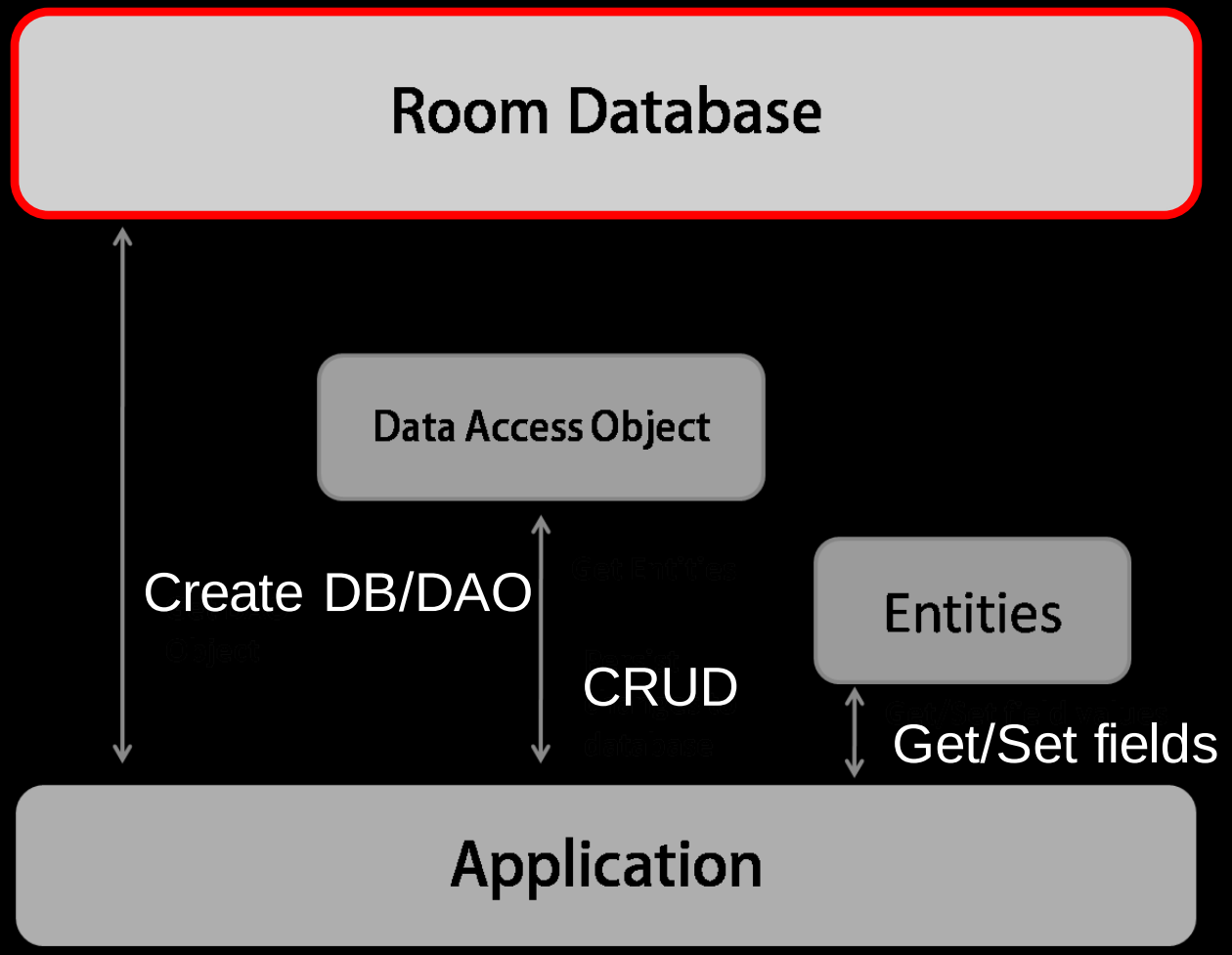
```
@Entity
data class LocalFile(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "file_id")
    val fileId: Long,

    @ColumnInfo(name = "user_id")
    val userId: Long,

    @ColumnInfo(name = "file_name")
    val fileName: String
)
```

1:n Beziehung zwischen einem User und vielen Dateien

```
data class UserWithFiles(
    @Embedded val user: LocalUser,

    @Relation(
        parentColumn = "user_id",
        entityColumn = "user_id„
    )
    val files: List<LocalFile>
)
```

```
@Dao
interface LocalUserDao {
    ...

    @Transaction
    @Query("SELECT * FROM LocalUser")
    fun getUsersWithFiles(): List<UserWithFiles>

}
```

Datenbank


Room Database
Data Access Object
Create DB/DAO
CRUD
Entities
Get/Set fields
Application

Datenbank zusammensetzen

```
@Database(
    entities = [LocalUser::class, LocalFile::class],
    version = 1
)
abstract class MyCoolLocalDatabase : RoomDatabase() {

    abstract fun localUserDao(): LocalUserDao
    abstract fun localFileDao(): LocalFileDao
}
```

```
val database = Room.databaseBuilder(
    context,
    MyCoolLocalDatabase::class.java,
    "MyCoolDBName"
).build()

val allUsers = database.localUserDao().getAll()
```

Live Coding

Fragen?

???