

1. DAO 정의 시작

Creating Data Access Objects (DAO)

Room 라이브러리는 엔티티 정의를 더 쉽게 만들었지만, DAO(Data Access Objects)를 생성할 때 그 진정한 힘이 발휘됩니다. DAO는 테이블의 데이터를 다루는 객체로, SQLite 예제와 유사하게 데이터를 삽입, 삭제, 업데이트하는 데 사용됩니다.

Room의 컴파일러는 각 DAO를 구현하는 클래스를 생성하므로, DAO 정의는 인터페이스 선언으로 작성해야 합니다. 우리는 어노테이션을 사용하여 Room이 DAO 코드를 생성하도록 지시합니다. 몇몇 어노테이션을 사용하면 삽입, 삭제, 업데이트 메서드를 쉽게 작성할 수 있습니다.

다음은 삽입 및 삭제 작업을 제공하는 DAO의 전체 정의 예제입니다:

kotlin

코드 복사

```
@Dao
interface UserDao {
    @Insert
    fun insert(vararg users: User)

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

- `@Dao`: 이 인터페이스가 DAO임을 나타냅니다.
- `@Insert`: 삽입 메서드임을 나타냅니다.
- `fun insert(vararg users: User)`: 여러 `User` 객체를 삽입하는 메서드입니다.
- `@Delete`: 삭제 메서드임을 나타냅니다.
- `fun delete(user: User)`: 하나의 `User` 객체를 삭제하는 메서드입니다.

Room의 컴파일러가 빌드 시점에 구현을 생성하므로, 우리는 정의만 하면 됩니다.

2. 업데이트 기능 추가 및 쿼리

우리는 삽입, 삭제 기능 외에 업데이트 기능을 추가하고, 삭제 및 업데이트 메서드가 영향을 받은 행의 수를 반환하도록 정의를 확장할 수 있습니다.

kotlin


📄 코드 복사

```
@Dao
interface UserDao {
    @Insert
    fun insert(vararg users: User)

    @Delete
    fun delete(user: User): Int


    @Update
    fun update(vararg users: User): Int
}
```

- `@Update`: 업데이트 메서드임을 나타냅니다.
- `fun update(vararg users: User): Int`: 여러 `User` 객체를 업데이트하는 메서드로, 영향을 받은 행의 수를 반환합니다.

이제 DAO가 형성되었지만, 데이터베이스를 쿼리할 수 있어야 합니다. 쿼리는 `@Query` 어노테이션을 사용하여 함수 내에서 선언할 수 있습니다. 이 어노테이션은 SQL 명령을 값으로 받습니다. 

예를 들어, 모든 사용자를 가져오는 간단한 쿼리는 다음과 같습니다:


kotlin

 코드 복사

```
@Query("SELECT * FROM user")  
fun findAll(): List<User>
```

매개변수가 있는 쿼리도 사용할 수 있습니다. 예를 들어, 특정 레벨의 사용자를 가져오는 쿼리는 다음과 같습니다:

kotlin

 코드 복사

```
@Query("SELECT * FROM user WHERE level = :level")  
fun findAllByLevel(level: UserLevel): List<User>
```

3. 투영 및 조인 쿼리

투영(projection)은 자동으로 구현될 수 있습니다. 예를 들어, 테이블에서 모든 사용자 이름을 가져오려면 다음과 같이 작성합니다:

```
kotlin 코드 복사

data class UserName (
    @ColumnInfo(name = "user_name") val name: String
)

@Query("SELECT user_name FROM user")
fun findAllUserNames(): List<UserName>
```

조인(join)도 자동으로 매핑됩니다. 예를 들어, 사용자의 작업 로그를 유지하는 애플리케이션이 있다고 가정해 보겠습니다. 이 로그는 사용자가 무언가를 할 때마다 타임스탬프를 기록합니다.

```
kotlin 코드 복사

@Entity(tableName = "audit", primaryKeys = ["userId", "timestamp"])
data class AuditEntry (
    val userId: Int,
    val timestamp: Long
) ↓
```

```

data class FullAuditEntry(
    val userId: Int,
    val timestamp: Long,
    @ColumnInfo(name = "user_name") val userName: String,
    @ColumnInfo(name = "level") val userLevel: Int
)

@Dao
interface AuditDao {
    @Insert
    fun insert(vararg entries: AuditEntry)

    @Query("SELECT * FROM audit " +
        "INNER JOIN user ON audit.userId = user.id")
    fun findAllWithUserDetails(): List<FullAuditEntry>
}

```

여기서 우리는 `AuditEntry`와 `FullAuditEntry` 클래스를 정의하고, 이들 간의 조인을 수행하는 쿼리를 작성합니다. ↓

Room을 사용하여 데이터베이스와 상호작용하는 방법을 정리해보았습니다. DAO와 어노테이션을 통해 데이터베이스 작업을 쉽게 처리할 수 있으며, 이는 코드의 가독성과 유지보수성을 높이는 데 큰 도움이 됩니다.