데이터베이스 마이그레이션은 데이터베이스 구조가 변경될 때 데이터를 안전하게 유지하면서 변경 사항을 적용하는 절차입니다. 예를 들어, 새로운 컬럼을 추가하거나 기존 컬럼을 수정하는 경우가 있습니다.

예제: 사용자 비밀번호 해시 컬럼 추가

1. 새로운 컬럼 추가하기

우리는 `User` 테이블에 `passwordHash`라는 새로운 컬럼을 추가하려고 합니다. 이 컬럼은 사용자의 비밀번호 해시 값을 저장할 것입니다.

2. 엔티티 클래스 변경

먼저, `User` 엔티티 클래스에 새로운 컬럼을 추가합니다:

kotlin 코드 복사

```kotlin
@Entity(tableName = "user")
data class User (
    @PrimaryKey val id: Int,
    val name: String,
    @ColumnInfo(name = "pwd_hash") val passwordHash: String
)
```

이 코드는 `User` 테이블에 `pwd_hash`라는 컬럼을 추가합니다.

3. 데이터베이스 버전 업데이트

이제 데이터베이스 버전을 업데이트해야 합니다. 버전 번호를 1에서 2로 변경합니다:

kotlin 코드 복사

```kotlin
@Database(
    entities = [User::class],
    version = 2
)
abstract class ApplicationDatabase: RoomDatabase() {
    abstract fun userDao(): UserDao
}
```

이 코드는 데이터베이스 버전을 2로 설정합니다.

4. 마이그레이션 정의

다음으로, 데이터베이스가 버전 1에서 2로 업그레이드될 때 실행될 코드를 정의합니다:

kotlin 코드 복사

```kotlin
internal val MIGRATION_1_2 = object : Migration(1, 2) {
    override fun migrate(database: SupportSQLiteDatabase) {
        database.execSQL(
            "ALTER TABLE user ADD COLUMN pwd_hash TEXT NOT NULL DEFAULT ''"
        )
    }
}
```

- `Migration(1, 2)`: 1에서 2로 버전이 변경될 때 실행되는 마이그레이션을 정의합니다.
- `migrate` 함수에서 SQL 명령어를 사용하여 `pwd_hash` 컬럼을 추가합니다.

5. 마이그레이션 등록

마지막으로, Room 데이터베이스 빌더에 마이그레이션을 등록합니다:

kotlin 코드 복사

```kotlin
companion object {
    private const val DATABASE_NAME = "app.db"

    @Volatile private var instance: ApplicationDatabase? = null

    fun getDatabase(context: Context): ApplicationDatabase =
        instance ?: synchronized(this) {
            instance ?: Room.databaseBuilder(
                context.applicationContext,
                ApplicationDatabase::class.java,
                DATABASE_NAME
            )
            .addMigrations(MIGRATION_1_2)
            .build()
            .also { instance = it }
        }
}
```

이 코드는 다음을 수행합니다:

1. 데이터베이스 이름을 `app.db`로 설정합니다.
2. `getDatabase` 함수는 싱글톤 패턴을 사용하여 데이터베이스 인스턴스를 반환합니다.
3. `Room.databaseBuilder`를 사용하여 데이터베이스를 빌드할 때 `addMigrations(MIGRATION_1_2)`를 호출하여 마이그레이션을 등록합니다.

요약

1. **엔티티 클래스 변경**: `User` 클래스에 새로운 컬럼을 추가합니다.
2. **데이터베이스 버전 업데이트**: `@Database` 어노테이션에서 버전을 2로 설정합니다.
3. **마이그레이션 정의**: 1에서 2로 버전이 변경될 때 실행될 SQL 명령어를 정의합니다.
4. **마이그레이션 등록**: Room 데이터베이스 빌더에 마이그레이션을 등록합니다.

이 과정을 통해 앱이 업데이트될 때 데이터베이스 구조를 안전하게 변경할 수 있습니다.