SQLite 데이터베이스

재능대 컴퓨터소프트웨어학과
서연경교수

안드로이드 애플리케이션에서 SQLite 데이터베이스 사용

데이터 저장 및 관리

SQLite 데이터베이스란?

- SQLite는 경량화된 관계형 데이터베이스.
- 파일 기반의 데이터베이스
- 서버가 필요 없음
- 안드로이드에 내장

SQLite 데이터베이스의 특징

- 구조화된 데이터 저장
- 반복 가능하고 쿼리가 필요한 데이터 관리
- 다른 데이터 간의 관계 설정
- SQL 표준을 따름
- 단일 파일로 구성됨
- 애플리케이션과 함께 배포 가능
- 간편한 설치 및 사용
- 빠른 데이터 처리 속도
- 낮은 메모리 사용량

안드로이드에서의 SQLite

- SQLiteOpenHelper 클래스 사용
- onCreate() 및 onUpgrade() 메서드 구현
- 데이터베이스 생성 및 버전 관리
- CRUD (Create, Read, Update, Delete) 구현

SQLiteOpenHelper 클래스

-데이터베이스 생성 및 초기화 담당

-onCreate() : 데이터베이스가 처음 생성될 때 호출

-onUpgrade() : 데이터베이스 버전이 변경될 때 호출

```
class MyDatabaseHelper(context: Context) : SQLiteOpenHelper(context, DATABASE_NAME, null,
DATABASE_VERSION) {
    companion object {
        private const val DATABASE_NAME = "example.db"
        private const val DATABASE_VERSION = 1
    }

    override fun onCreate(db: SQLiteDatabase) {
        db.execSQL("CREATE TABLE example (_id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT);")
    }

    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
        db.execSQL("DROP TABLE IF EXISTS example")
        onCreate(db)
    }
}
```

CRUD 작업

- 데이터 삽입: insert()
- 데이터 조회: query(), rawQuery()
- 데이터 수정: update()
- 데이터 삭제: delete()

```
class MyDatabaseHelper(context: Context) : SQLiteOpenHelper(context, DATABASE_NAME, null,
DATABASE_VERSION) {
    companion object {
        private const val DATABASE_NAME = "example.db" // 데이터베이스 파일 이름
        private const val DATABASE_VERSION = 1 // 데이터베이스 버전
    }

    override fun onCreate(db: SQLiteDatabase) {
        // 데이터베이스가 처음 생성될 때 테이블을 생성하는 SQL 문
        db.execSQL("CREATE TABLE example (_id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT);")
    }

    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
        // 데이터베이스 버전이 변경될 때 호출되어 기존 테이블을 삭제하고 새로 생성
        db.execSQL("DROP TABLE IF EXISTS example")
        onCreate(db)
    }
}
```

```
override fun onCreate(db: SQLiteDatabase) {
    db.execSQL("""
        CREATE TABLE ${UserTable.TABLE_NAME} (
            ${UserTable.COLUMN_ID} INTEGER PRIMARY KEY,
            ${UserTable.COLUMN_NAME} TEXT
        )
    """.trimIndent())
}
```

```
override fun onUpgrade(db: SQLiteDatabase,
                       oldVersion: Int, newVersion: Int) {
    db.execSQL("DROP TABLE IF EXISTS ${UserTable.TABLE_NAME}")
    onCreate(db)
}
```

db.insert 기본 예

```
fun insertData(name: String) {
    val db = writableDatabase // 쓰기 가능한 데이터베이스 객체를 가져옴
    val values = ContentValues().apply {
        put("name", name) // 삽입할 데이터 설정
    }
    val newRowId = db.insert("example", null, values) // 데이터를 테이블에 삽입
    db.close() // 데이터베이스 연결 닫기
}
```

```kotlin
fun insertUser(user: User): Boolean {
    val userValues = ContentValues().apply {
        put(UserTable.COLUMN_ID, user.id)
        put(UserTable.COLUMN_NAME, user.name)
    }
    return writableDatabase.let {
        it.insert(UserTable.TABLE_NAME, null, userValues)
!= -1L
    }
}
```

db.update 기본 예

```
fun updateData(id: Long, newName: String) {
    val db = writableDatabase // 쓰기 가능한 데이터베이스 객체를 가져옴
    val values = ContentValues().apply {
        put("name", newName) // 수정할 데이터 설정
    }
    val selection = "_id = ?" // 수정할 행을 지정하는 조건
    val selectionArgs = arrayOf(id.toString()) // 조건에 해당하는 인자
    val count = db.update("example", values, selection, selectionArgs) // 데이터를 업데이트
    db.close() // 데이터베이스 연결 닫기
}
```

```kotlin
fun updateUser(user: User): Boolean {
    val userValues = ContentValues().apply {
        put(UserTable.COLUMN_NAME, user.name)
    }
    return writableDatabase.update(
            UserTable.TABLE_NAME, userValues,
            "${UserTable.COLUMN_ID}==?",
arrayOf(user.d.toString())
        ) > 0
}
```

db.delete 기본 예

```
fun deleteData(id: Long) {
    val db = writableDatabase // 쓰기 가능한 데이터베이스 객체를 가져옴
    val selection = "_id = ?" // 삭제할 행을 지정하는 조건
    val selectionArgs = arrayOf(id.toString()) // 조건에 해당하는 인자
    val deletedRows = db.delete("example", selection, selectionArgs) // 데이터를 삭제
    db.close() // 데이터베이스 연결 닫기
}
```

```
1. fun removeUser(userId: Int): Int =
2.     writableDatabase
3.         .delete(UserTable.TABLE_NAME, "${UserTable.COLUMN_ID}==?",
4.             arrayOf(userId.toString()))
```

db.query 기본 예

```
fun getAllData(): List<String> {
    val dataList = mutableListOf<String>() // 결과를 저장할 리스트
    val db = readableDatabase // 읽기 가능한 데이터베이스 객체를 가져옴
    val cursor = db.query("example", null, null, null, null, null, null) // 테이블의 모든 데이터를 조회

    with(cursor) {
        while (moveToNext()) { // 다음 행으로 이동
            val name = getString(getColumnIndexOrThrow("name")) // "name" 열의 데이터 가져오기
            dataList.add(name) // 리스트에 추가
        }
        close() // Cursor 닫기
    }
    db.close() // 데이터베이스 연결 닫기
    return dataList // 결과 리스트 반환
}
```

```
fun query(
    table: String,        // 테이블 이름
    columns: Array<String>?,   // 조회할 열 목록 (null이면 모든 열을 조회)
    selection: String?,        // WHERE 절 (null이면 모든 행을 조회)
    selectionArgs: Array<String>?, // WHERE 절의 플레이스홀더에 해당하는 값 (null이면 플레이스
    groupBy: String?,          // GROUP BY 절 (null이면 그룹화하지 않음)
    having: String?,           // HAVING 절 (null이면 HAVING 절을 사용하지 않음)
    orderBy: String?,          // ORDER BY 절 (null이면 정렬하지 않음)
    limit: String? = null      // LIMIT 절 (null이면 모든 행을 반환)
): Cursor
```

```
val cursor = db.query(
    "example",    // table: "example" 테이블에서 데이터를 조회합니다.
    null,         // columns: null은 모든 열을 조회합니다.
    null,         // selection: null은 모든 행을 조회합니다.
    null,         // selectionArgs: WHERE 절의 플레이스홀더 값이 없음을 의미합니다.
    null,         // groupBy: null은 그룹화하지 않음을 의미합니다.
    null,         // having: null은 HAVING 절을 사용하지 않음을 의미합니다.
    null          // orderBy: null은 정렬하지 않음을 의미합니다.
)
```

```kotlin
fun query(
    table: String,          // 테이블 이름
    columns: Array<String>?,    // 조회할 열 목록 (null이면 모든 열을 조회)
    selection: String?,         // WHERE 절 (null이면 모든 행을 조회)
    selectionArgs: Array<String>?, // WHERE 절의 플레이스홀더에 해당하는 값 (null이면 플레이스
    groupBy: String?,           // GROUP BY 절 (null이면 그룹화하지 않음)
    having: String?,            // HAVING 절 (null이면 HAVING 절을 사용하지 않음)
    orderBy: String?,           // ORDER BY 절 (null이면 정렬하지 않음)
    limit: String? = null       // LIMIT 절 (null이면 모든 행을 반환)
): Cursor
```

```kotlin
val cursor = db.query(
    "example",                  // 테이블 이름
    arrayOf("name", "age"),     // 조회할 열 목록
    "age > ?",                  // WHERE 절
    arrayOf("20"),              // WHERE 절의 플레이스홀더 값
    null,                       // GROUP BY 절 (null이면 그룹화하지 않음)
    null,                       // HAVING 절 (null이면 HAVING 절을 사용하지 않음)
    "name ASC"                  // ORDER BY 절
)
```

```
1. fun listUsers(): List<User> =
2.     readableDatabase.let {db ->
3.         db.query(UserTable.TABLE_NAME, arrayOf(UserTable.COLUMN_ID,
4.                 UserTable.COLUMN_NAME), null, null, null, null,        null)
5.             .use {cursor ->
6.                 List(cursor.count) {index ->
7.                     cursor.moveToPosition(index)
8.                     User(
9.                         cursor.getInt(cursor.getColumnIndex(
10.                                 UserTable.COLUMN_ID)),
11.                         cursor.getString(cursor.getColumnIndex(
12.                                 UserTable.COLUMN_NAME))
13.                     )
14.                 }
15.             }
16.     }
```