



Mobil alkalmazásfejlesztés - Értesítés, Riasztás

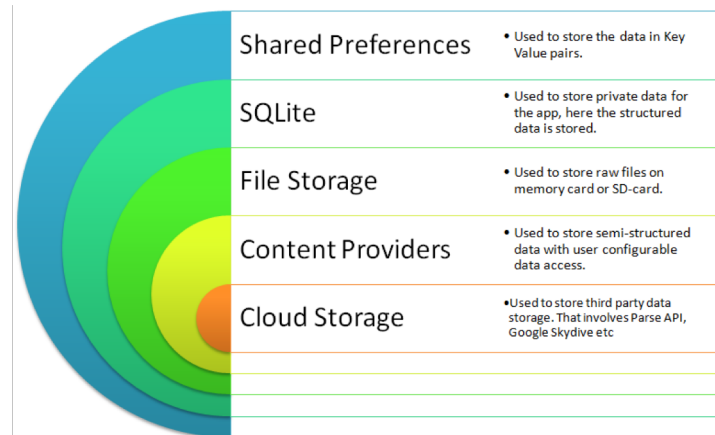
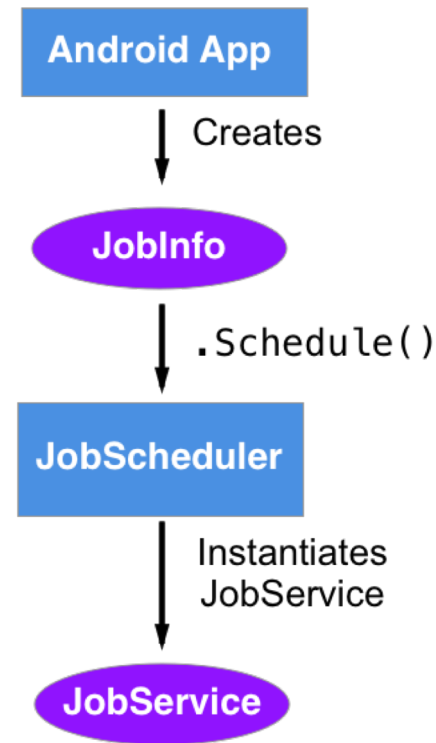
Dr. Bilicki Vilmos
Szoftverfejlesztés Tanszék

A fóliához felhasznált anyagok:

Google: Android Developer Fundamentals (Version 2), <https://developer.android.com/courses/fundamentals-training/overview-v2>

Áttekintés

- Hatékony adatátvitel
- Job Scheduler
 - JobService
 - JobInfo
 - JobScheduler
- Adattárolás
 - Android File System
 - Internal Storage
 - External Storage
 - SQLite Database
 - Other Storage Options



■ Tartalom

- Shared Preferences
 - Eseménykezelés
- SQLite Database
 - Lekérdezések
- ROOM
 - Entitások
 - DAO
 - Adatbázis



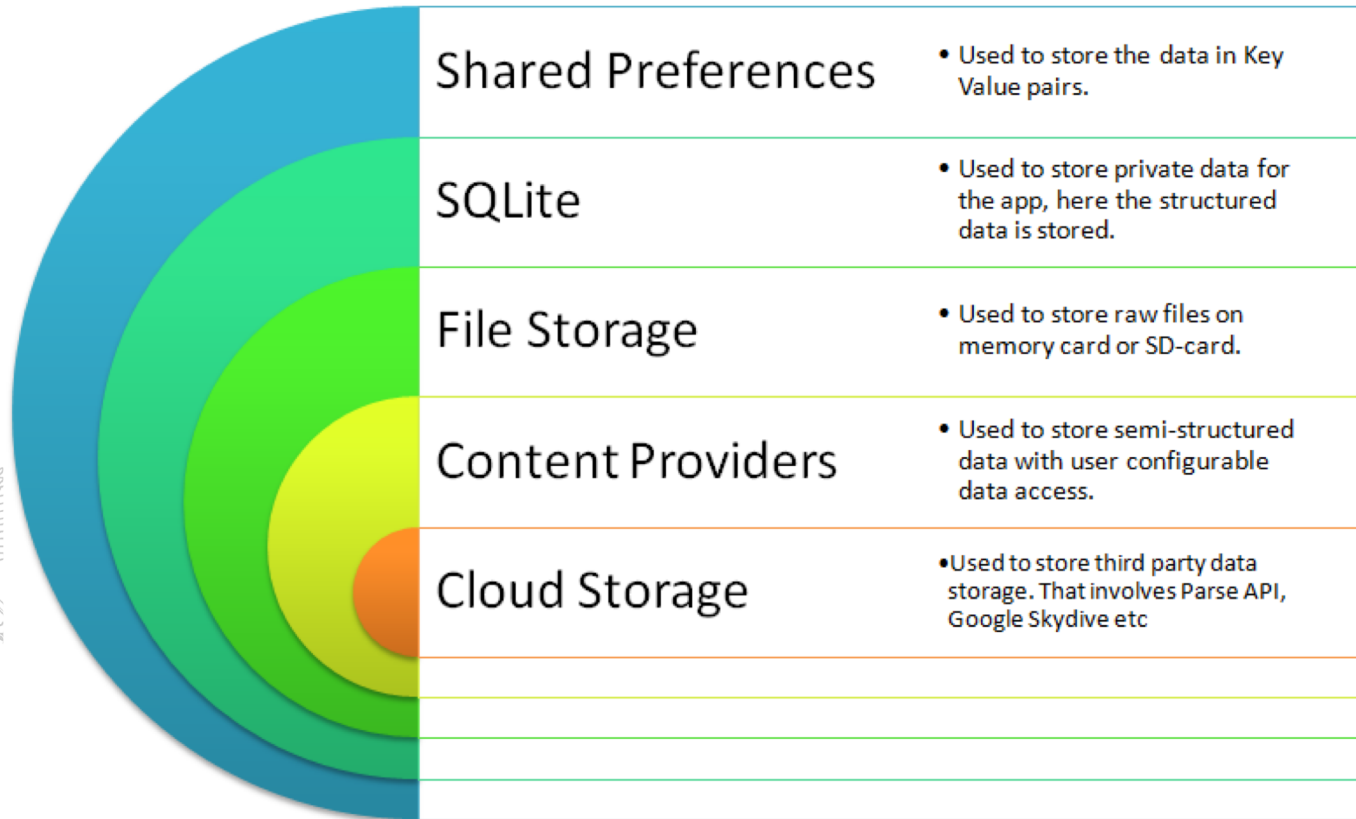
■ Shared Preferences

- Kevés, egyszerű adat név/érték alapú tárolása egy fájlban
- A SharedPreferences osztály megfelelő API-kat biztosít az adat kezeléséhez
- Elmenti az adatot onPause() esetén és betölti onCreate() esetén



■ Shared Preferences és Elmentett Példány Állapot

- Kevés név/érték pár alapú adat
- Csak az alkalmazás férhet hozzá



■ Shared Preferences vs. Elmentett Példány Állapot

- Adatot tárol a felhasználói viszonyok között, akkor is ha az alkalmazást leállítják, vagy az eszköz újraindul
- Az adatnak tartósan meg kell maadnia (pl.: felhasználó kedvenc beállításai, a játékban elért pontjai)
- A felhasználói beállítások kategória tárolására is használhatjuk
- Állapotot tart a felhasználói viszony alatt a különböző aktivitások között
- Az adatnak nem kell megmaradnia viszonyok között (pl.: adott activity-ben kiválasztott lap).
- Arra használják gyakran, hogy visszaállítsa az állapotot az eszköz forgatása után



■ Shared Preferences létrehozása

- Egy Shared Preferences fájl kell alkalmazásonként
- A nevét a csomagnév alapján hozzuk létre – egyedi és az alkalmazáshoz köthető
- A `getSharedPreferences()` `MODE` argumentuma csak a kompatibilitás miatt használandó —egyébként használjuk a `MODE_PRIVATE` a zárt eléréshez



■ getSharedPreferences()

```
private String sharedPrefFile =  
    "com.example.android.hellosharedprefs";  
  
mPreferences =  
    getSharedPreferences(sharedPrefFile,  
        MODE_PRIVATE);
```



■ Shared Preferences elmentése

- [SharedPreferences.Editor](#) interfész
- Elfedi a fájl műveleteket
- Amennyiben a kulcs már létezett akkor a put metódus átírja
- apply() aszinkron és biztonságos módon ment



■ SharedPreferences.Editor

```
@Override
protected void onPause() {
    super.onPause();
    SharedPreferences.Editor preferencesEditor =
        mPreferences.edit();
    preferencesEditor.putInt("count", mCount);
    preferencesEditor.putInt("color", mCurrentColor);
    preferencesEditor.apply();
}
```



■ Shared Preferences visszaállítása

- Az Activity-n belül az onCreate()-ban állítsuk vissza
- A get metódusok két argumentumot kaphatnak, egy kulcsot és egy alapértelmezett értéket amennyiben kulcs nincs meg
- Azért érdemes alapértelmezett értéket használni, mert ilyenkor nem kell megvizsgálni, hogy adott kulcs megvan-e



■ Adata lekérése az onCreate()-ban

```
mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);  
if (savedInstanceState != null) {  
    mCount = mPreferences.getInt("count", 1);  
    mShowCount.setText(String.format("%s", mCount));  
  
    mCurrentColor = mPreferences.getInt("color", mCurrentColor);  
    mShowCount.setBackgroundColor(mCurrentColor);  
  
    mNewText = mPreferences.getString("text", "");  
} else { ... }
```



■ Takarítás

- A clear() metódust kell meghívni a SharedPreferences.Editor –on és elfogadni a változásokat
- Amennyiben egyszerre használjuk a put és clear metódusokat a clear először hajtódik végre (apply)



■ clear()

```
SharedPreferences.Editor  
preferencesEditor =  
    mPreferences.edit();  
preferencesEditor.clear();  
preferencesEditor.apply();
```



■ Eseménykezelés

- SharedPreferences.OnSharedPreferencesChangeListener interfész megvalósítása
- Az eseménykezelő regisztrálása a registerOnSharedPreferencesChangeListener() segítségével
- A onResume() és onPause() hatására regisztráljuk és töröljük a regisztrációt
- Valósítsuk meg a onSharedPreferencesChanged() callback-et



■ Interfész és callback

```
public class SettingsActivity extends AppCompatActivity
    implements OnSharedPreferencesChangeListener { ...

    public void onSharedPreferencesChanged(
        SharedPreferences sharedPreferences, String key) {
        if (key.equals(MY_KEY)) {
            // Do something
        }
    }
}
```



■ Eseménykezelő létrehozása és regisztrálása

```
SharedPreferences.OnSharedPreferenceChangeListener listener =  
    new SharedPreferences.OnSharedPreferenceChangeListener() {  
        public void onSharedPreferenceChanged(  
            SharedPreferences prefs, String key) {  
                // Implement listener here  
            }  
    };  
prefs.registerOnSharedPreferenceChangeListener(listener);
```



■ SQLite szoftver könyvtár

SQL adatbázis motort valósít meg

- önhordó (nincs szükség más komponensre)
- szervermentes (nincs szükség szerver háttérre)
- nulla konfiguráció (nem kell konfigurálni az alkalmazásból)
- tranzakcionális (adott változások kötegek együtt történnek meg vagy sikertelen lesz a változás - atomiság)



■ SQL alap műveletek

- Sorok beszúrás
- Sorok törlése
- Adott sorokban értékek frissítése
- Adott feltételnek elegettevő sorok lekérése



■ SQL Query

- SELECT word, description
FROM WORD_LIST_TABLE
WHERE word="alpha"

Általános

- SELECT columns
FROM table
WHERE column="value"



■ AND, ORDER BY, LIMIT

```
SELECT _id FROM WORD_LIST_TABLE  
WHERE word="alpha" AND definition LIKE  
"%art%" ORDER BY word DESC LIMIT 1
```

- **AND, OR**—feltételek
- **ORDER BY**—ASC , DESC
- **LIMIT**—adott számú eredmény



■ Minta lekérdezések



1	SELECT * FROM WORD_LIST_TABLE	Az egész táblát
2	SELECT word, definition FROM WORD_LIST_TABLE WHERE _id > 2	[["alpha", "particle"]]

WORD_LIST_TABLE

_id	word	definition
1	"alpha"	"first letter"
2	"beta"	"second letter"
3	"alpha"	"particle"

■ Minta lekérdezések



3	SELECT _id FROM WORD_LIST_TABLE WHERE word="alpha" AND definition LIKE "%art%"	[["3"]]
4	SELECT * FROM WORD_LIST_TABLE ORDER BY word DESC LIMIT 1	(_id) [["3", "alpha", "particle"]]

WORD_LIST_TABLE		
_id	word	definition
1	"alpha"	"first letter"
2	"beta"	"second letter"
3	"alpha"	"particle"

■ Minta lekérdezések



5	<pre>SELECT * FROM WORD_LIST_TAB LE LIMIT 2,1</pre>	Egy elemet ad vissza a második helytől számítva. [["2", "beta", "second letter"]]
---	---	--

WORD_LIST_TABLE		
_id	word	definition
1	"alpha"	"first letter"
2	"beta"	"second letter"
3	"alpha"	"particle"

■ rawQuery()

```
String query = "SELECT * FROM WORD_LIST_TABLE";  
rawQuery(query, null);
```

```
query = "SELECT word, definition FROM  
WORD_LIST_TABLE WHERE _id> ? ";
```

```
String[] selectionArgs = new String[]{"2"}  
rawQuery(query, selectionArgs);
```



■ query()



```
SELECT * FROM  
WORD_LIST_TABLE  
WHERE word="alpha"  
ORDER BY word ASC  
LIMIT 2,1;
```

Returns:

```
[[ "alpha",  
  "particle" ]]
```

```
String table = "WORD_LIST_TABLE"  
String[] columns = new String[]{"*"};  
String selection = "word = ?"  
String[] selectionArgs = new  
String[]{"alpha"};  
String groupBy = null;  
String having = null;  
String orderBy = "word ASC"  
String limit = "2,1"  
  
query(table, columns, selection,  
selectionArgs, groupBy, having, orderBy,  
limit);
```

■ Kurzorok

A lekérdezések kurzor objektumot adnak vissza

A Cursor egy interfész amely lehetővé teszi a lekérdezés eredményének véletlen hozzáférésű írását/olvasását

⇒ Egy mutató a tábla sorra



■ Tranzakció

Egy logikai munkamenet műveleteinek összessége.

Négy fontos tulajdonság:

- atomiság
- konzisztencia
- elkülönítés
- tartósság



■ Tranzakció szintjei

id	name	age
1	Joe	20
2	Jill	25



■ Dirty read – Piszkos adatok olvasása



Transaction 1	Transaction 2
BEGIN;	BEGIN;
SELECT age FROM users WHERE id = 1; <i>/* will read 20 */</i>	
	UPDATE users SET age = 21 WHERE id = 1; <i>/* No commit here */</i>
SELECT age FROM users WHERE id = 1; <i>/* will read 21 */</i>	

■ Non-repeatable Reads– Nem megismételhető olvasás

Transaction 1	Transaction 2
BEGIN;	BEGIN;
SELECT * FROM users WHERE id = 1;	
	UPDATE users SET age = 21 WHERE id = 1;
	COMMIT;
SELECT * FROM users WHERE id = 1; <i>/* will read 21 */</i>	



■ Phantom Reads – Fantom olvasás



Transaction 1	Transaction 2
BEGIN;	BEGIN;
SELECT COUNT(*) FROM users WHERE age BETWEEN 10 AND 30; <i>/* will return 5 */</i>	
	INSERT INTO users(id,name,age) VALUES (3, 'Bob', 27);
	COMMIT;
SELECT COUNT(*) FROM users WHERE age BETWEEN 10 AND 30; <i>/* will return 6 */</i>	

■ SQL elkülönítési szintek

1. Read uncommitted: mindhármát megengedi.
2. Read committed: megengedi a fantom és a nem megismételhető olvasását.
3. Repeatable read: csak a fantom olvasást engedi meg
4. Serializable: nem enged meg hibát



■ Room áttekintés

A Room egy robosztus SQL objektum áthidaló könyvtár

- SQLite Android kódot generál
- Egyszerű API-t biztosít



■ A Room elemei

- **Entity:** Az adatbázis sémáját adja meg

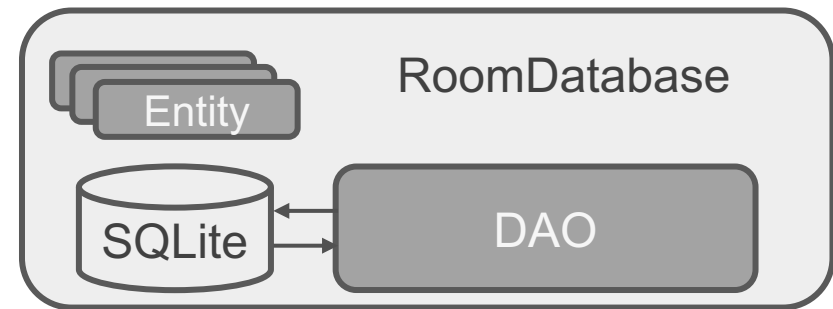
- **DAO:** Database Access Object

Az adatbázis írás/olvasás műveleteit adja meg.

Database:

Az adatbázis tartalmazza.

Adatbázisok létrehozása, hozzájuk csatlakozás

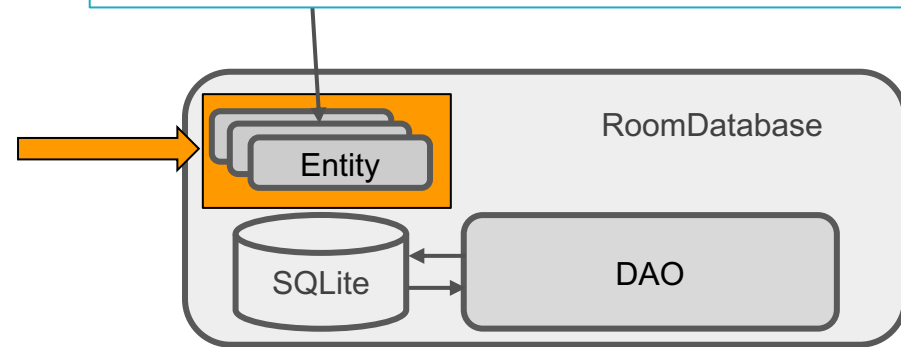


■ Entity

- Egyed példány = egy sor az adatbázisban
- POJO osztályok segítségével definiáljuk
- 1 példány = 1 sor
- Tag változó = oszlop név



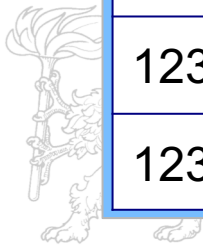
```
public class Person {  
    private int uid;  
    private String firstName;  
    private String lastName;  
}
```



■ Egyed példány = sor a táblában

```
public class Person {  
    private int uid;  
    private String firstName;  
    private String lastName;  
}
```

uid	firstName	lastName
12345	Aleks	Becker
12346	Jhansi	Kumar



■ Entitások annotálása

@Entity

```
public class Person {
```

```
    @PrimaryKey (autoGenerate=true)
```

```
    private int uid;
```

```
    @ColumnInfo(name = "first_name")
```

```
    private String firstName;
```

```
    @ColumnInfo(name = "last_name")
```

```
    private String lastName;
```

```
    // + getters and setters if variables are private.
```

```
}
```



■ @Entity annotáció

```
@Entity(tableName = "word_table")
```

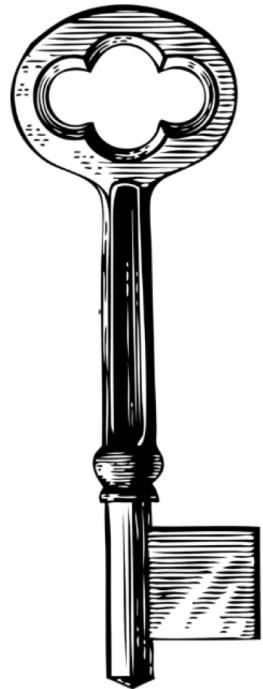
- Minden @Entity példány egy sort jelent a táblában
- Amennyiben más tábla nevet szeretnénk akkor it tudjuk megadni



■ @PrimaryKey annotáció

@PrimaryKey (autoGenerate=true)

- Az **Entity** osztálynak kell egy elsődleges kulcs mező
- Leht generálni is egyedi azonosítókat



■ @NonNull annotáció

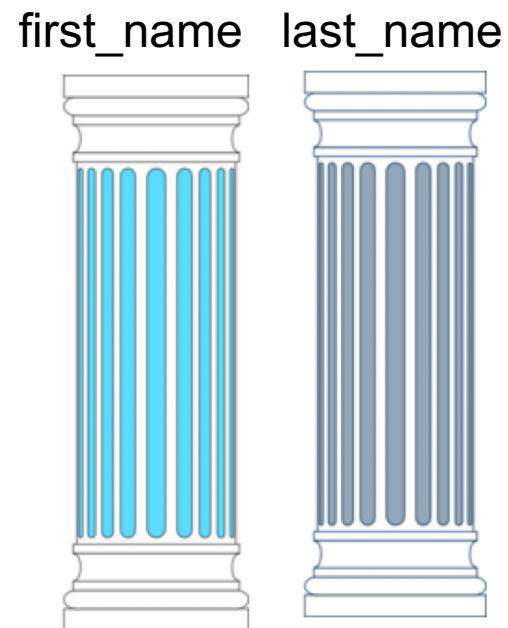
@NonNull

- Azt jelöli, hogy egy paraméter, mező vagy visszatérési érték nem lehet null értékű
- A kötelező mezőkre alkalmazzuk
- Az elsődleges kulcsra kötelező a @NonNull



■ @ColumnInfo annotáció

```
@ColumnInfo(name =  
"first_name")  
private String firstName;  
  
@ColumnInfo(name =  
"last_name")  
private String lastName;
```



- Itt tudjuk az oszlop nevét átírni

■ Getter, setter

Az adatbázisban tárolt mezőknek

- publikusnak

vagy

- "getter" metódussal bírónak kell lennie



■ Viszonyok

Az entitások közötti
viszonyokat a
`@Relation` annotációval
adhatjuk meg

A lekérdezések a
viszonyokat is lekérlik

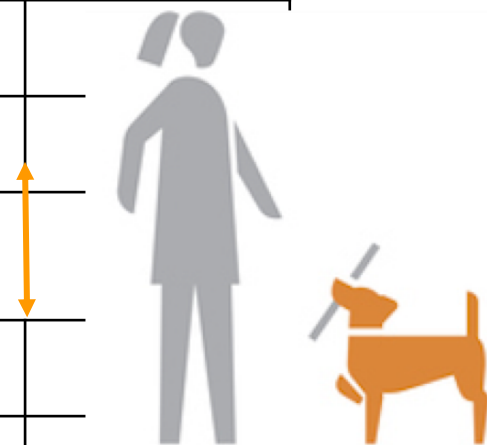


users table

id	
name	
pet	

pets table

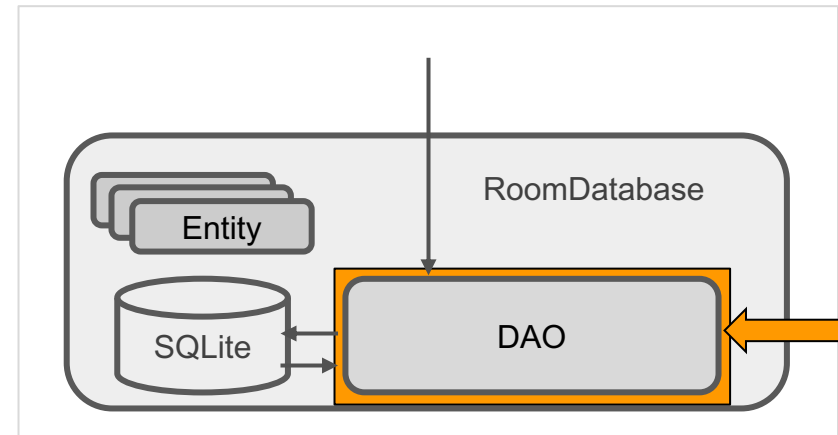
id	
name	
owner	



■ Data access object



A Room persistence library-hoz szükséges a DAO megadása



■ Data access object

- A DAO metódusok absztrakt hozzáférést specifikálnak
- Az adatforrások az entitás objektumok
- A DAO kötelezően interface vagy abstract osztály
- A Room keretrendszer a DAO segítségével generál API-t



■ Minta DAO

@Dao

public interface WordDao {

@Insert

void insert(Word word);

@Update

public void updateWords(Word... words);



■ Minta lekérdezések

```
@Query("DELETE FROM word_table")  
void deleteAll();
```

```
@Query("SELECT * from word_table ORDER BY word ASC")  
List<Word> getAllWords();
```

```
@Query("SELECT * FROM word_table WHERE word LIKE :word ")  
public List<Word> findWord(String word);
```



■ Room adatbázis létrehozása

- A RoomDatabase-ból származó publikus, absztrakt osztályt kell létrehoznunk
- @Database-ként annotáljuk
- Deklaráljuk az entitásokat és a verziót

```
@Database(entities = {Word.class}, version = 1)
```

```
public abstract class WordRoomDatabase extends RoomDatabase
```



Room osztály minta

```
@Database(entities = {Word.class}, version = 1)
public abstract class WordRoomDatabase
    extends RoomDatabase {

    public abstract WordDao wordDao();

    private static WordRoomDatabase INSTANCE;

    // ... create instance here

}
```

*Entitás átlatl def.
DB séma*

*DAO az
adatbázsihoz*

*Singelton
adatbázis*



Room adatbázis létrehozás minta

```
static WordRoomDatabase getDatabase(final Context context) {  
    if (INSTANCE == null) {  
        synchronized (WordRoomDatabase.class) {  
            if (INSTANCE == null) {  
                INSTANCE = Room.databaseBuilder(  
                    context.getApplicationContext(),  
                    WordRoomDatabase.class, "word_database")  
                        .addCallback(sOnOpenCallback)  
                        .fallbackToDestructiveMigration()  
                        .build();  
            }  
        }  
    }  
    return INSTANCE;  
}
```

Van-e már létező
DB



■ DB inicializálás onOpen callback-re

```
private static RoomDatabase.Callback sOnOpenCallback =  
    new RoomDatabase.Callback(){  
        @Override  
        public void onOpen (@NonNull SupportSQLiteDatabase db){  
            super.onOpen(db);  
            initializeData();  
        }  
    };
```



■ Tartalom

- Shared Preferences
 - Esménykezelés
- SQLite Database
 - Lekérdezések
- ROOM
 - Entitások
 - DAO
 - Adatbázis



A következő előadás tartalma

1. Alkalmazás architektúra
2. LiveData és ViewModel
3. Android összefoglaló

