



# PROGRAMOWANIE URZĄDZEŃ MOBILNYCH 2

## WYKŁAD 9

Bazy Danych ROOM:

- Podstawowe Komponenty: Entity, Dao, Database
- Operacje Asynchroniczne
- CRUD
- Integracja z MVVM

Jeżeli mamy konieczność zapisania **większej ilości danych** najczęściej korzystamy z **bazy danych**. W androidzie lokalnie możemy skorzystać z klasy **ROOM**, która jest warstwą abstrakcyjną nad wbudowanym silnikiem **SQLite**.

Jest to oprogramowanie, które *rozumie* i **wykonuje polecenia** napisane w języku **SQL**.

SQL to standardowy, **deklaracyjny język programowania** służący do komunikacji z relacyjnymi bazami danych.

Jeżeli mamy konieczność zapisania **większej ilości danych** najczęściej korzystamy z **bazy danych**. W androidzie lokalnie możemy skorzystać z klasy **ROOM**, która jest warstwą abstrakcyjną nad wbudowanym silnikiem **SQLite**.

Jest to oprogramowanie, które *rozumie* i **wykonuje polecenia** napisane w języku **SQL**.

SQL to standardowy, **deklaratywny język programowania** służący do komunikacji z relacyjnymi bazami danych.

CRUD - Cztery Podstawowe Operacje

## 1. C - Create -> INSERT

- Cel: Dodaje nowy wiersz (rekord) do tabeli.
- Składnia: `INSERT INTO tasks (title, is_done) VALUES ('Zrobić zakupy', false);`

## 2. R - Read -> SELECT

- Cel: Pobiera dane z tabeli.
- Składnia: `SELECT * FROM tasks WHERE is_done = true ORDER BY id DESC;`

## 3. U - Update -> UPDATE

- Cel: Modyfikuje istniejące wiersze w tabeli.
- Składnia SQL: `UPDATE tasks SET is_done = true WHERE id = 5;`

## 4. D - Delete -> DELETE

1. Cel: Usuwa wiersze z tabeli.
2. Składnia: `DELETE FROM tasks WHERE id = 5;`

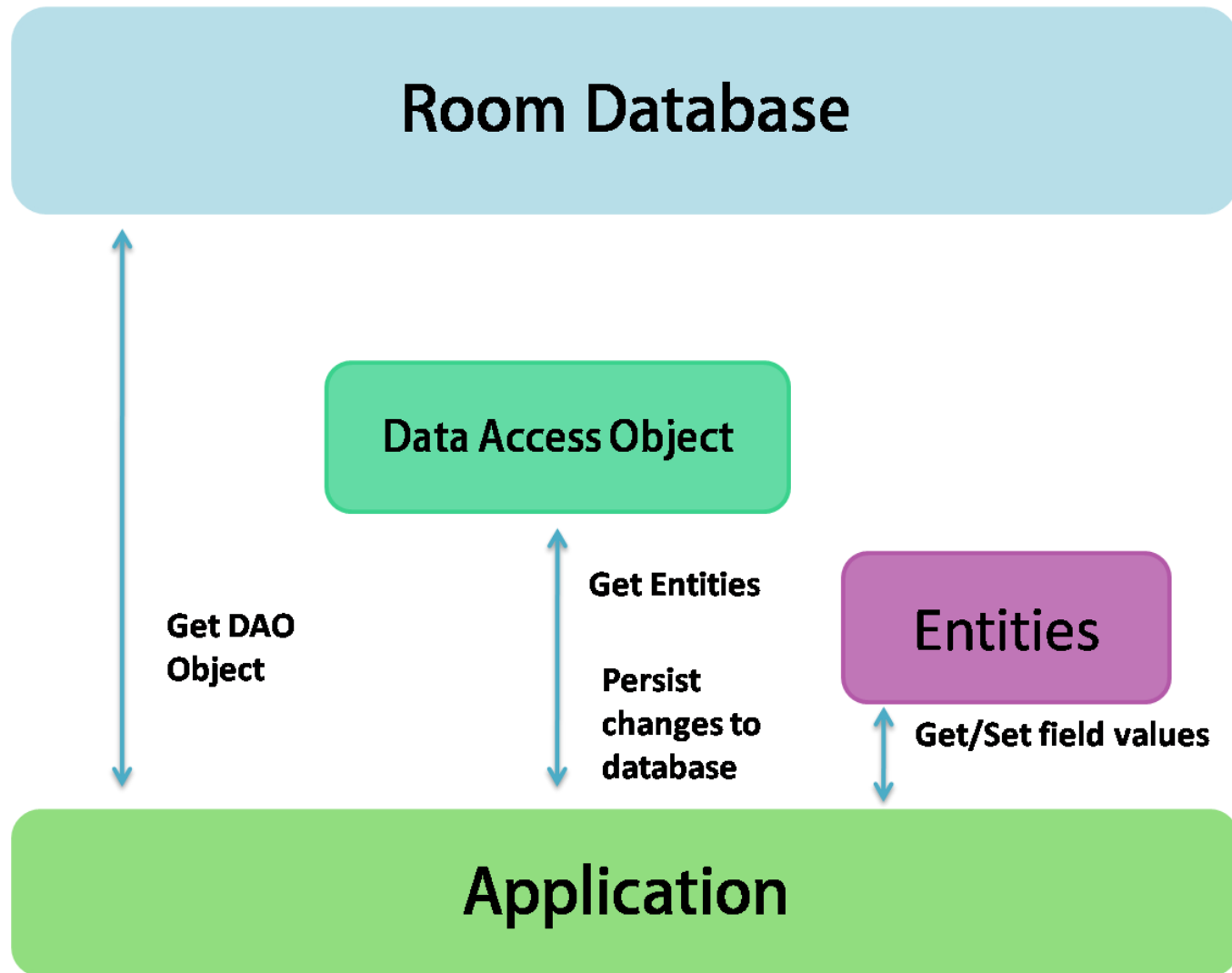
Jeżeli mamy konieczność zapisania **większej ilości danych** najczęściej korzystamy z **bazy danych**. W androidzie lokalnie możemy skorzystać z klasy **ROOM**, która jest warstwą abstrakcyjną nad wbudowanym silnikiem **SQLite**.

Jest to oprogramowanie, które *rozumie* i **wykonuje polecenia** napisane w języku **SQL**.

SQL to standardowy, **deklaratywny język programowania** służący do komunikacji z relacyjnymi bazami danych.

Główne cechy SQLite:

- **Działa jako biblioteka dostępna w postaci pliku w aplikacji:** że nie wymaga oddzielnego procesu – aplikacja może bezpośrednio komunikować się z bazą
- **Jest samowystarczalna:** cała baza danych jest przechowywana w jednym pliku – brak konieczności konfigurowania zasobów
- **Transakcyjność:** umożliwia grupowanie operacji bazodanowych jako pojedyncze, atomowe działania
- **Wsparcie dla standardowego SQL:** obsługuje większość standardowego języka SQL




Główne składniki biblioteki **ROOM**:

- **Entity**: reprezentuje tabelę w bazie danych. Każda klasa oznaczona adnotacją `@Entity` może być mapowana do jednej tabeli w bazie danych, pola klasy odpowiadają kolumnom tej tabeli
- **DAO**: definiuje metody dostępowe do danych w bazie. Definiujemy za pomocą adnotacji `@Dao`
- **Database**: Klasa bazowa reprezentująca bazę danych. Miejsce gdzie definiujemy wszystkie *encje*, które mają zostać użyte w aplikacji. Room automatycznie tworzy implementację bazy danych w oparciu o tą klasę

Definiuje **strukturę tabeli**. Jest to **zwykła klasa** data class.

To adnotacja z biblioteki Room, która oznacza, że ta **klasa definiuje tabelę** w bazie danych.



```
@Entity(tableName = "tasks")
data class Task(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val title: String,
    val isCompleted: Boolean = false
)
```

Definiuje **strukturę tabeli**. Jest to **zwykła klasa** data class.

To adnotacja z biblioteki Room, która oznacza, że ta **klasa definiuje tabelę** w bazie danych.

**Określa** dokładną **nazwę** tej **tabeli** w bazie **SQLite**. Bez tego, Room **domyślnie** użyłby nazwy klasy (Task).

```
@Entity(tableName = "tasks")
data class Task(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val title: String,
    val isCompleted: Boolean = false
)
```



Definiuje **strukturę tabeli**. Jest to **zwykła klasa** data class.

To adnotacja z biblioteki Room, która oznacza, że ta **klasa definiuje tabelę** w bazie danych.

**Określa** dokładną **nazwę** tej **tabeli** w bazie **SQLite**. Bez tego, Room **domyślnie** użyłby nazwy klasy (Task).

Ta adnotacja oznacza, że właściwość **id** jest **kluczem głównym** tabeli. Klucz główny musi być **unikalny** dla **każdego wiersza** i służy do jego identyfikacji.

```
@Entity(tableName = "tasks")
data class Task(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val title: String,
    val isCompleted: Boolean = false
)
```

Definiuje **strukturę tabeli**. Jest to **zwykła klasa** data class.

To adnotacja z biblioteki Room, która oznacza, że ta **klasa definiuje tabelę** w bazie danych.

Ta adnotacja oznacza, że właściwość id jest **kluczem głównym** tabeli. Klucz główny musi być **unikalny** dla **każdego wiersza** i służy do jego identyfikacji.

Określa dokładną **nazwę** tej **tabeli** w bazie **SQLite**. Bez tego, Room **domyślnie** użyłby nazwy klasy (Task).

Automatycznie nadaje **nową, unikalną, rosnącą** wartość id dla **każdego nowego wiersza** dodawanego do tabeli

```
@Entity(tableName = "tasks")
data class Task(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val title: String,
    val isCompleted: Boolean = false
)
```

**Interfejs**, w którym definiujemy **wszystkie operacje**, jakie chcemy wykonywać **na danej tabeli**.

Ta adnotacja oznacza, że interfejs TaskDao jest obiektem dostępu do danych (Data Access Object). Room użyje tego interfejsu do **wygenerowania kodu**, który będzie **wykonywał zapytania** do bazy danych.

  
`@Dao``interface TaskDao {``1 Usage 1 Implementation``@Insert(onConflict = OnConflictStrategy.REPLACE)``suspend fun insertTask(task: Task)``1 Usage 1 Implementation``@Update``suspend fun updateTask(task: Task)``1 Implementation``@Delete``suspend fun deleteTask(task: Task)``1 Usage 1 Implementation``@Query(value = "SELECT * FROM tasks ORDER BY id DESC")``fun getAllTasksStream(): Flow<List<Task>>``}`

**Interfejs**, w którym definiujemy **wszystkie operacje**, jakie chcemy wykonywać **na danej tabeli**.

Ta adnotacja oznacza, że interfejs TaskDao jest obiektem dostępu do danych (Data Access Object). Room użyje tego interfejsu do **wygenerowania kodu**, który będzie **wykonywał zapytania** do bazy danych.

Metoda wstawiania nowych wierszy do tabeli

@Dao

interface TaskDao {

1 Usage 1 Implementation

@Insert(onConflict = OnConflictStrategy.REPLACE)

suspend fun insertTask(task: Task)

1 Usage 1 Implementation

@Update

suspend fun updateTask(task: Task)

1 Implementation

@Delete

suspend fun deleteTask(task: Task)

1 Usage 1 Implementation

@Query(value = "SELECT \* FROM tasks ORDER BY id DESC")

fun getAllTasksStream(): Flow<List<Task>>

}

**Interfejs**, w którym definiujemy **wszystkie operacje**, jakie chcemy wykonywać **na danej tabeli**.

Ta adnotacja oznacza, że interfejs TaskDao jest obiektem dostępu do danych (Data Access Object). Room użyje tego interfejsu do **wygenerowania kodu**, który będzie **wykonywał zapytania** do bazy danych.

Metoda wstawiania nowych wierszy do tabeli

@Dao

interface TaskDao {

1 Usage 1 Implementation

@Insert(onConflict = OnConflictStrategy.REPLACE)

suspend fun insertTask(task: Task)

1 Usage 1 Implementation

@Update

suspend fun updateTask(task: Task)

1 Implementation

@Delete

suspend fun deleteTask(task: Task)

1 Usage 1 Implementation

@Query(value = "SELECT \* FROM tasks ORDER BY id DESC")

fun getAllTasksStream(): Flow<List<Task>>

}

Określa, co ma się stać, jeśli spróbujemy wstawić wiersz z **kluczem głównym**, który **już istnieje** w tabeli.

OnConflictStrategy.REPLACE mówi Room, aby w takim przypadku **zastąpił stary wiersz nowym**.

**Interfejs**, w którym definiujemy **wszystkie operacje**, jakie chcemy wykonywać **na danej tabeli**.

Ta adnotacja oznacza, że interfejs TaskDao jest obiektem dostępu do danych (Data Access Object). Room użyje tego interfejsu do **wygenerowania kodu**, który będzie **wykonywał zapytania** do bazy danych.

Metoda wstawiania nowych wierszy do tabeli

Metoda aktualizacji wierszy w tabeli

Metoda usuwania wierszy z tabeli

@Dao

interface TaskDao {

1 Usage 1 Implementation

@Insert(onConflict = OnConflictStrategy.REPLACE)

suspend fun insertTask(task: Task)

1 Usage 1 Implementation

@Update

suspend fun updateTask(task: Task)

1 Implementation

@Delete

suspend fun deleteTask(task: Task)

1 Usage 1 Implementation

@Query(value = "SELECT \* FROM tasks ORDER BY id DESC")

fun getAllTasksStream(): Flow<List<Task>>

}

Określa, co ma się stać, jeśli spróbujemy wstawić wiersz z **kluczem głównym**, który **już istnieje** w tabeli.

OnConflictStrategy.REPLACE mówi Room, aby w takim przypadku **zastąpił stary wiersz nowym**.

**Interfejs**, w którym definiujemy **wszystkie operacje**, jakie chcemy wykonywać **na danej tabeli**.

Ta adnotacja oznacza, że interfejs TaskDao jest obiektem dostępu do danych (Data Access Object). Room użyje tego interfejsu do **wygenerowania kodu**, który będzie **wykonywał zapytania** do bazy danych.

Metoda wstawiania nowych wierszy do tabeli

Metoda aktualizacji wierszy w tabeli

Metoda usuwania wierszy z tabeli

Pozwala na definiowanie własnych zapytań SQL

@Dao

interface TaskDao {

1 Usage 1 Implementation

@Insert(onConflict = OnConflictStrategy.REPLACE)

suspend fun insertTask(task: Task)

1 Usage 1 Implementation

@Update

suspend fun updateTask(task: Task)

1 Implementation

@Delete

suspend fun deleteTask(task: Task)

1 Usage 1 Implementation

@Query(value = "SELECT \* FROM tasks ORDER BY id DESC")

fun getAllTasksStream(): Flow<List<Task>>

Określa, co ma się stać, jeśli spróbujemy wstawić wiersz z **kluczem głównym**, który **już istnieje** w tabeli.

OnConflictStrategy.REPLACE mówi Room, aby w takim przypadku **zastąpił stary wiersz nowym**



**Interfejs**, w którym definiujemy **wszystkie operacje**, jakie chcemy wykonywać **na danej tabeli**.

Ta adnotacja oznacza, że interfejs TaskDao jest obiektem dostępu do danych (Data Access Object). Room użyje tego interfejsu do **wygenerowania kodu**, który będzie **wykonywał zapytania** do bazy danych.

Metoda wstawiania nowych wierszy do tabeli

Metoda aktualizacji wierszy w tabeli

Metoda usuwania wierszy z tabeli

Pozwala na definiowanie własnych zapytań SQL

@Dao

interface TaskDao {

1 Usage 1 Implementation

@Insert(onConflict = OnConflictStrategy.REPLACE)

suspend fun insertTask(task: Task)

1 Usage 1 Implementation

@Update

suspend fun updateTask(task: Task)

1 Implementation

@Delete

suspend fun deleteTask(task: Task)

1 Usage 1 Implementation

@Query(value = "SELECT \* FROM tasks ORDER BY id DESC")

fun getAllTasksStream(): Flow<List<Task>>

Określa, co ma się stać, jeśli spróbujemy wstawić wiersz z **kluczem głównym**, który **już istnieje** w tabeli.

OnConflictStrategy.REPLACE mówi Room, aby w takim przypadku **zastąpił stary wiersz nowym**

Funkcja nie jest zawieszająca, ponieważ zwraca Flow. Room będzie **automatycznie** emitował **nową, zaktualizowaną** listę zadań do tego **strumienia** za każdym razem, gdy dane w tabeli tasks ulegną jakiegokolwiek **zmianie**.



Abstrakcyjna klasa, która łączy wszystkie encje i DAO.

Adnotacja Room oznaczająca, że ta klasa jest **głównym punktem dostępowym do bazy danych**.

```
@Database(entities = [Task::class], version = 1, exportSchema = false)
abstract class AppDatabase : RoomDatabase() {
    1 Usage 1 Implementation
    abstract fun taskDao(): TaskDao
    4 Usages
    companion object {
        2 Usages
        @Volatile
        private var INSTANCE: AppDatabase? = null
        1 Usage
        fun getDatabase(context: Context): AppDatabase {
            return INSTANCE ?: synchronized(lock = this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    class = AppDatabase::class.java,
                    name = "task_database_db"
                ).build()
                INSTANCE = instance
                instance
            }
        }
    }
}
```

Abstrakcyjna klasa, która łączy wszystkie encje i DAO.

Zawiera informację o wszystkich encjach (tabelach) w bazie danych

Adnotacja Room oznaczająca, że ta klasa jest **głównym punktem dostępowym do bazy danych**.

```
@Database(entities = [Task::class], version = 1, exportSchema = false)
abstract class AppDatabase : RoomDatabase() {
    1 Usage 1 Implementation
    abstract fun taskDao(): TaskDao
    4 Usages
    companion object {
        2 Usages
        @Volatile
        private var INSTANCE: AppDatabase? = null
        1 Usage
        fun getDatabase(context: Context): AppDatabase {
            return INSTANCE ?: synchronized( lock = this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    class = AppDatabase::class.java,
                    name = "task_database_db"
                ).build()
                INSTANCE = instance
                instance
            }
        }
    }
}
```

Wyłącza eksportowanie schematu bazy do pliku JSON

Określa **numer wersji bazy danych**. Jest to kluczowe dla mechanizmu migracji.

Abstrakcyjna klasa, która łączy wszystkie encje i DAO.

Zawiera informację o wszystkich encjach (tabelach) w bazie danych

Adnotacja Room oznaczająca, że ta klasa jest **głównym punktem dostępowym do bazy danych**.

Klasa bazy danych musi być abstrakcyjna, ponieważ Room sam **wygeneruje** jej konkretną **implementację w tle**.

```
@Database(entities = [Task::class], version = 1, exportSchema = false)
abstract class AppDatabase : RoomDatabase() {
    1 Usage 1 Implementation
    abstract fun taskDao(): TaskDao
    4 Usages
    companion object {
        2 Usages
        @Volatile
        private var INSTANCE: AppDatabase? = null
        1 Usage
        fun getDatabase(context: Context): AppDatabase {
            return INSTANCE ?: synchronized( lock = this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    class = AppDatabase::class.java,
                    name = "task_database_db"
                ).build()
                INSTANCE = instance
                instance
            }
        }
    }
}
```

Wyłącza eksportowanie schematu bazy do pliku JSON

Określa **numer wersji bazy danych**. Jest to kluczowe dla mechanizmu migracji.

Abstrakcyjna klasa, która łączy wszystkie encje i DAO.

Zawiera informację o wszystkich encjach (tabelach) w bazie danych

Adnotacja Room oznaczająca, że ta klasa jest **głównym punktem dostępowym do bazy danych**.

Klasa bazy danych musi być abstrakcyjna, ponieważ Room sam **wygeneruje** jej konkretną **implementację w tle**.

Deklaruje abstrakcyjną funkcję, która zwraca instancję TaskDao. Implementację konkretną dostarczy ROOM

```
@Database(entities = [Task::class], version = 1, exportSchema = false)
abstract class AppDatabase : RoomDatabase() {
    1 Usage 1 Implementation
    abstract fun taskDao(): TaskDao
    4 Usages
    companion object {
        2 Usages
        @Volatile
        private var INSTANCE: AppDatabase? = null
        1 Usage
        fun getDatabase(context: Context): AppDatabase {
            return INSTANCE ?: synchronized( lock = this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    class = AppDatabase::class.java,
                    name = "task_database_db"
                ).build()
                INSTANCE = instance
                instance
            }
        }
    }
}
```

Wyłącza eksportowanie schematu bazy do pliku JSON

Określa **numer wersji bazy danych**. Jest to kluczowe dla mechanizmu migracji.

Abstrakcyjna klasa, która łączy wszystkie encje i DAO.

Zawiera informację o wszystkich encjach (tabelach) w bazie danych

Adnotacja Room oznaczająca, że ta klasa jest **głównym punktem dostępowym do bazy danych**.

Klasa bazy danych musi być abstrakcyjna, ponieważ Room sam **wygeneruje** jej konkretną **implementację w tle**.

Deklaruje abstrakcyjną funkcję, która zwraca instancję TaskDao. Implementację konkretną dostarczy ROOM

Adnotacja zapewniająca, że wartość zmiennej INSTANCE będzie **zawsze aktualna i widoczna** dla wszystkich wątków w aplikacji.

```
@Database(entities = [Task::class], version = 1, exportSchema = false)
abstract class AppDatabase : RoomDatabase() {
    1 Usage 1 Implementation
    abstract fun taskDao(): TaskDao
    4 Usages
    companion object {
        2 Usages
        @Volatile
        private var INSTANCE: AppDatabase? = null
        1 Usage
        fun getDatabase(context: Context): AppDatabase {
            return INSTANCE ?: synchronized( lock = this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    class = AppDatabase::class.java,
                    name = "task_database_db"
                ).build()
                INSTANCE = instance
                instance
            }
        }
    }
}
```

Wyłącza eksportowanie schematu bazy do pliku JSON

Określa **numer wersji bazy danych**. Jest to kluczowe dla mechanizmu migracji.

Abstrakcyjna klasa, która łączy wszystkie encje i DAO.

Zawiera informację o wszystkich encjach (tabelach) w bazie danych

Adnotacja Room oznaczająca, że ta klasa jest **głównym punktem dostępowym do bazy danych**.

Klasa bazy danych musi być abstrakcyjna, ponieważ Room sam **wygeneruje** jej konkretną **implementację w tle**.

Deklaruje abstrakcyjną funkcję, która zwraca instancję TaskDao. Implementację konkretną dostarczy ROOM

Adnotacja zapewniająca, że wartość zmiennej INSTANCE będzie **zawsze aktualna i widoczna** dla wszystkich wątków w aplikacji.

```
@Database(entities = [Task::class], version = 1, exportSchema = false)
abstract class AppDatabase : RoomDatabase() {
    1 Usage 1 Implementation
    abstract fun taskDao(): TaskDao
    4 Usages
    companion object {
        2 Usages
        @Volatile
        private var INSTANCE: AppDatabase? = null
        1 Usage
        fun getDatabase(context: Context): AppDatabase {
            return INSTANCE ?: synchronized( lock = this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    class = AppDatabase::class.java,
                    name = "task_database_db"
                ).build()
                INSTANCE = instance
                instance
            }
        }
    }
}
```

Wyłącza eksportowanie schematu bazy do pliku JSON

Określa **numer wersji bazy danych**. Jest to kluczowe dla mechanizmu migracji

przechowuje jedyną instancję bazy danych



Abstrakcyjna klasa, która łączy wszystkie encje i DAO.

Zawiera informację o wszystkich encjach (tabelach) w bazie danych

Adnotacja Room oznaczająca, że ta klasa jest **głównym punktem dostępowym do bazy danych**.

Klasa bazy danych musi być abstrakcyjna, ponieważ Room sam **wygeneruje** jej konkretną **implementację w tle**.

Deklaruje abstrakcyjną funkcję, która zwraca instancję TaskDao. Implementację konkretną dostarczy ROOM

Adnotacja zapewniająca, że wartość zmiennej INSTANCE będzie **zawsze aktualna i widoczna** dla wszystkich wątków w aplikacji.

```
@Database(entities = [Task::class], version = 1, exportSchema = false)
abstract class AppDatabase : RoomDatabase() {
    1 Usage 1 Implementation
    abstract fun taskDao(): TaskDao
    4 Usages
    companion object {
        2 Usages
        @Volatile
        private var INSTANCE: AppDatabase? = null
        1 Usage
        fun getDatabase(context: Context): AppDatabase {
            return INSTANCE ?: synchronized( lock = this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    class = AppDatabase::class.java,
                    name = "task_database_db"
                ).build()
                INSTANCE = instance
                instance
            }
        }
    }
}
```

Wyłącza eksportowanie schematu bazy do pliku JSON

Określa **numer wersji bazy danych**. Jest to kluczowe dla mechanizmu migracji

przechowuje **jedyną instancję bazy danych**

Tworzy **blok synchronizowany**. Gwarantuje, że **kod wewnątrz** niego **nie zostanie wykonany** przez **dwa wątki jednocześnie**

Abstrakcyjna klasa, która łączy wszystkie encje i DAO.

Zawiera informację o wszystkich encjach (tabelach) w bazie danych

Adnotacja Room oznaczająca, że ta klasa jest **głównym punktem dostępowym do bazy danych**.

Klasa bazy danych musi być abstrakcyjna, ponieważ Room sam **wygeneruje** jej konkretną **implementację w tle**.

Deklaruje abstrakcyjną funkcję, która zwraca instancję TaskDao. Implementację konkretną dostarczy ROOM

Adnotacja zapewniająca, że wartość zmiennej INSTANCE będzie **zawsze aktualna i widoczna** dla wszystkich wątków w aplikacji.

Konstruktor bazy danych Room.

Kontekst aplikacji

Klasa bazowa

```
@Database(entities = [Task::class], version = 1, exportSchema = false)
abstract class AppDatabase : RoomDatabase() {
    1 Usage 1 Implementation
    abstract fun taskDao(): TaskDao
    4 Usages
    companion object {
        2 Usages
        @Volatile
        private var INSTANCE: AppDatabase? = null
        1 Usage
        fun getDatabase(context: Context): AppDatabase {
            return INSTANCE ?: synchronized( lock = this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    class = AppDatabase::class.java,
                    name = "task_database_db"
                ).build()
                INSTANCE = instance
            }
        }
    }
}
```

Wyłącza eksportowanie schematu bazy do pliku JSON

Określa **numer wersji bazy danych**. Jest to kluczowe dla mechanizmu migracji

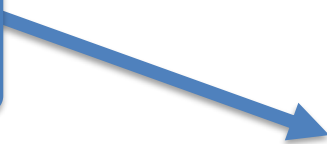
przechowuje jedyną instancję bazy danych

Tworzy **blok synchronizowany**. Gwarantuje, że **kod wewnątrz** niego **nie zostanie wykonany** przez **dwa wątki jednocześnie**

Nazwa pliku



Klasa TaskRepository przyjmuje TaskDao jako **zależność** w swoim konstruktorze.



```
class TaskRepository(private val taskDao: TaskDao) {  
    1 Usage  
    val allTasks: Flow<List<Task>> = taskDao.getAllTasksStream()  
    1 Usage  
    suspend fun insert(task: Task) {  
        taskDao.insertTask(task)  
    }  
    1 Usage  
    suspend fun update(task: Task) {  
        taskDao.updateTask(task)  
    }  
}
```

```
class TaskViewModel(private val repository: TaskRepository) : ViewModel() {  
    1 Usage  
    val tasks: StateFlow<List<Task>> = repository.allTasks  
        .stateIn(  
            scope = viewModelScope,  
            started = SharingStarted.WhileSubscribed(stopTimeoutMillis = 5000),  
            initialValue = emptyList()  
        )  
    1 Usage  
    fun addTask(title: String) {  
        if (title.isNotBlank()) {  
            viewModelScope.launch {  
                repository.insert(Task(title = title))  
            }  
        }  
    }  
    1 Usage  
    fun toggleTaskCompletion(task: Task) {  
        viewModelScope.launch {  
            repository.update(task = task.copy(isCompleted = !task.isCompleted))  
        }  
    }  
}
```

```
@OptIn( ...markerClass = ExperimentalMaterial3Api::class)
@Composable
fun TaskScreen() {
    // Pobranie Application context do fabryki
    val application = LocalContext.current.applicationContext as Application
    val viewModel: TaskViewModel = viewModel(factory = TaskViewModelFactory(application))
    val tasks by viewModel.tasks.collectAsStateWithLifecycle()
    var newTaskTitle by remember { mutableStateOf( value = "" ) }

    Scaffold(...) { padding ->
        Column(modifier = Modifier.padding( paddingValues = padding ).fillMaxSize() ) { ... }
    }
}
```

Każda zmiana w bazie danych (nawet z innego miejsca w aplikacji) **automatycznie** przepływa przez wszystkie warstwy i odświeża UI.

1. **Czym jest Room?:** To biblioteka, która działa jako warstwa abstrakcji nad wbudowaną w Androida bazą danych **SQLite**. Upraszcza pracę z bazą, zamieniając skomplikowane operacje na **proste wywołania funkcji w Kotlinie** i zapewnia sprawdzanie poprawności zapytań SQL już w czasie kompilacji.
2. **Główne Komponenty Room:** Architektura Room opiera się na **trzech głównych elementach**, które definiujemy za pomocą adnotacji:
  - **@Entity:** Klasa data class, która definiuje tabelę w bazie danych, jej kolumny oraz klucz główny (@PrimaryKey).
  - **@Dao (Data Access Object):** Interfejs, w którym definiujemy wszystkie operacje na bazie danych, używając adnotacji takich jak @Insert, @Update, @Delete oraz @Query do własnych zapytań SQL.
  - **@Database:** Abstrakcyjna klasa, która jest głównym kontenerem całej bazy. Łączy w sobie wszystkie encje i DAO, a także zarządza wersjonowaniem.
3. **Reaktywność z Flow:** Kluczową zaletą Room jest jego integracja z korutinami. Metody w DAO mogą być oznaczone jako suspend dla jednorazowych operacji lub mogą zwracać Flow. Zwrócenie Flow<List<Task>> sprawia, że UI może **reaktywnie obserwować** zmiany w bazie danych i aktualizować się automatycznie, bez dodatkowego kodu.
4. **Wzorzec Singleton:** Instancja bazy danych (AppDatabase) jest tworzona jako **singleton**, aby zapewnić, że w całej aplikacji istnieje tylko jedno połączenie z bazą, co jest kluczowe dla wydajności i spójności danych.