Room i SQLite

mgr inż. Stanisław Lota

Room

Room według dokumentacji technicznej Android zapewnia warstwę abstrakcji nad SQLite, aby umożliwić płynny dostęp do bazy danych, jednocześnie wykorzystując SQLite.

Ułatwia pracę z obiektami bazy SQLite w aplikacji, zmniejszając ilość standardowego kodu i weryfikując zapytania SQL w czasie kompilacji.

Room jest obecnie uważany za lepsze podejście do trwałości danych niż baza danych SQLite.

Room

Room został wydany przez Google w 2016. Co najważniejsze Room jest powiązany z technologią ORM (Object Relational Mapping).

Innymi słowy mapuje nasze obiekty bazy danych na obiekty Java. Mapowanie obiektowo-relacyjne (ORM) to sposób odwzorowania obiektowej architektury aplikacji do przechowywania obiektów które są częściej używane w kodzie aplikacji w tabelach relacyjnej bazy danych.

Zadaniem tej technologii jest odciążenie programisty od pracy nad bazą danych. W ORM nie operujemy na zapytaniach SQL.

Room

W Room cała komunikacja odbywa się automatycznie przy udziale technologii ORM.

Za pomocą adnotacji możemy definiować nasze bazy danych, tabele i operacje. Room automatycznie przetłumaczy te adnotacje na instrukcje/zapytania CRUD w SQLite, aby wykonać odpowiednie operacje w silniku bazy danych.

W skrajnych przypadkach nie jest wymagana nawet znajomość zapytań SQL, ponieważ programista operuje tylko na obiektach odwzorowujących strukturę tabel.

Room

Room składa się z trzech głównych składników pokoju:

Encja: Zamiast tworzyć tabelę SQLite, utworzymy Encję. Encja to nic innego jak klasa modelu z adnotacjami @Entity. Zmienne tej klasy to nasze kolumny, a klasa to nasza tabela.

Room

DAO: Interfejs zawierający metody dostępu do bazy danych. Jest opatrzony adnotacją @DAO.

DAO to skrót od Data Access Object(obiekt dostępu do danych). Jest to interfejs, który definiuje wszystkie operacje, które musimy wykonać w naszej bazie danych.

Room

Baza danych: Reprezentuje bazę danych. Jest to obiekt, który utrzymuje połączenie z bazą danych SQLite, a wszystkie operacje są wykonywane za jego pośrednictwem. Jest opatrzony adnotacją @Database.

Adnotacje

@Entity – Za pomocą tej adnotacji tworzymy model tabeli.

„tableName” – definiujemy nazwę tabeli. Gdy nazwa klasy jest taka sama jak nazwa tabeli, możemy ją zignorować.

„foreignKeys” – nazwy kluczy obcych (opcjonalne)

„indices” – lista indeksów w tabeli (opcjonalne)

„primaryKeys„ – klucze podstawowe Entity (może być puste, ale tylko gdy klasa ma pole z adnotacją @PrimaryKey)

„ColumnInfo” – umożliwia określenie niestandardowych informacji o kolumnie.

„Ignore” — pole nie będzie „utrwalone” przez Room

„Embeded” — pole zagnieżdżone. Można się do niego odwoływać bezpośrednio w zapytaniach SQL.

Zapytania

Zapytania „tworzymy” za pomocą metody bezpośredniej wraz z odpowiednią adnotacją.

@Query – to główna adnotacja używana w klasach DAO. Umożliwia wykonywanie operacji odczytu / zapisu w bazie danych. Każda metoda @Query jest weryfikowana w czasie kompilacji, więc jeśli wystąpi problem z zapytaniem, zamiast błędu w czasie wykonywania, wystąpi błąd kompilacji.

@Insert – Kiedy tworzymy metodę DAO i dodajesz do niej adnotacje @Insert, Room generuje implementację, która wstawia wszystkie parametry do bazy danych w jednej transakcji.

Zapytania

@Delete – Metoda z adnotacją @Delete usuwa z bazy danych zbiór jednostek podanych jako parametry. Używa kluczy podstawowych, aby znaleźć jednostki do usunięcia.

@Update – Metoda z adnotacją @Update modyfikuje zbiór jednostek podanych jako parametry w bazie danych. Używa zapytania, które pasuje do klucza podstawowego każdej jednostki.

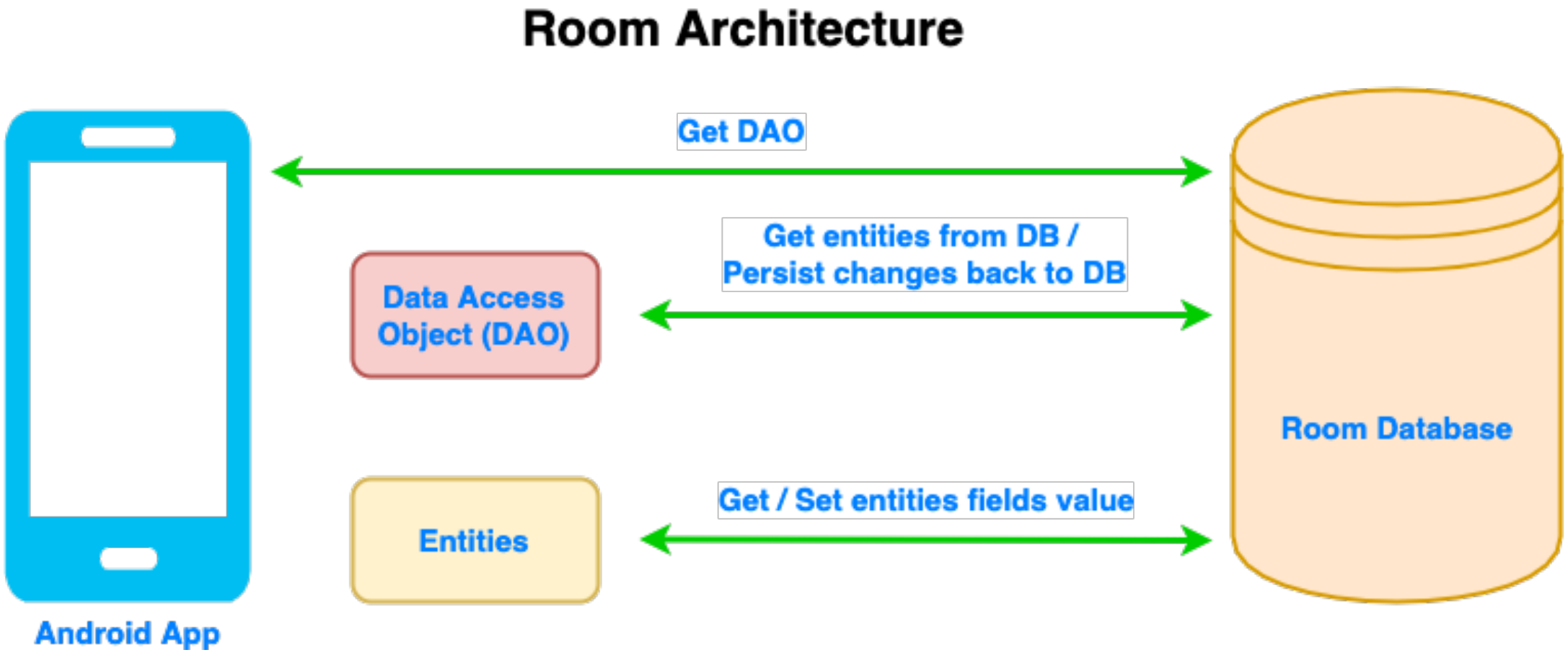
DAO

Zawiera „właściciela” bazy danych (eng. database holder) i służy jako główny punkt dostępu do połączenia z relacyjnymi danymi aplikacji.

Ta klasa musi spełniać następujące warunki:

- Klasa abstrakcyjna, którą rozszerza RoomDatabase.
- Zawiera listę Entities skojarzonych z bazą danych.
- Zawierają metodę abstrakcyjną, która niema argumentów i zwraca klasę oznaczoną adnotacją @DAO.

Room – schemat architektury



SQLite

SQLite jest bez serwerową, relacyjną, lekką bazą danych. Znajduje się w dokładnie jednych pliku tekstowym. Bardzo często wybierana jest jako baza dla aplikacji mobilnych na iOS oraz Android jak również w małych aplikacjach okienkowych.

Nie wspiera zarządzania użytkownikami, ani nie jest przeznaczona do równoległego jej użytkowania. Przez co staje się idealną bazą danych w momencie kiedy potrzebujemy prostej, relacyjnej bazy danych, znajdującej się w pakiecie aplikacji.

SQLite

Na przechowywanych w bazie danych możemy wykonać operacje typu CRUD jak dodawanie nowych danych, aktualizowanie, odczytywanie i usuwanie tych danych.

SQLite jest bazą offline, która jest przechowywana lokalnie na urządzeniu użytkownika i nie musimy tworzyć żadnego połączenia, aby połączyć się z tą bazą danych takich jak JDBC, ODBC ani żadnych innych połączeń zewnętrznych, tak jak to należy zrobić w aplikacjach Java czy Python.

SQLite

Lite w SQLite oznacza „lekkość” pod względem potrzebnych zasobów, niezbędnej konfiguracji oraz administrowania bazą danych.

Do obsługi bazy danych stosuje się język SQL (ang. Structure Query Language). Język SQL został zaprojektowany do komunikacji z bazami danych i w odróżnieniu od innych języków składa się tylko z kilku słów.

Zadaniem języka SQL jest proste i wydajne odczytywanie i zapisywanie informacji w bazie. Za pomocą SQL pomimo jego prostoty można budować bardzo złożone zapytania opierające się na relacjach między tabelami.

Kluczowe cechy SQLite

Samowystarczalny (wymaga minimalnego wsparcia ze strony systemu operacyjnego lub biblioteki zewnętrznej).

Bez serwerowy (zwykle RDBMS, taki jak MySQL, PostgreSQL itp., do działania, wymaga procesu serwera). Baza danych SQLite jest zintegrowana z aplikacją. Aplikacje współpracują z bazą danych SQLite, odczytując i zapisując bezpośrednio z plików bazy danych na dysku).

Zerowa konfiguracja (dzięki architekturze bez serwerowej nie trzeba „instalować” oprogramowania SQLite przed jego użyciem). Nie ma procesu serwera, który trzeba konfigurować, uruchamiać i zatrzymywać).

Transakcyjny (wszystkie transakcje w SQLite są w pełni zgodne z ACID).

Dlaczego warto skorzystać z Room?

Weryfikacja zapytań SQL w czasie kompilacji.

Każde @Query i @Entity jest sprawdzane w czasie kompilacji, co chroni aplikację przed awariami w czasie wykonywania i nie tylko sprawdza jedyną składnię, ale także brakujące tabele.

Kod wzorca.

Łatwa integracja z innymi komponentami architektury (takimi jak LiveData).

Główne problemy z użyciem SQLite to

Nie ma weryfikacji w czasie kompilacji nieprzetworzonych zapytań SQL. Na przykład, jeśli napiszemy zapytanie SQL z nieprawidłową nazwą kolumny, która nie istnieje w rzeczywistej bazie danych, spowoduje to wyjątek w czasie wykonywania i nie można przechwycić tego problemu w czasie kompilacji.

W miarę zmiany schematu należy ręcznie zaktualizować kwerendy SQL, których dotyczy problem. Proces ten może być czasochłonny i podatny na błędy.

Musimy użyć dużej ilości standardowego kodu do konwersji między zapytaniami SQL a obiektami danych Java (POJO).

Różnica między SQLite i Room

W przypadku SQLite nie ma weryfikacji w czasie kompilacji surowych zapytań SQLite. W Room istnieje walidacja SQL w czasie kompilacji.

Do konwersji między zapytaniami SQL a obiektami danych Java należy użyć wielu standardowych kodów. Ale Room mapuje nasze obiekty bazy danych na obiekt Java bez standardowego kodu.

W miarę zmiany schematu należy ręcznie zaktualizować kwerendy SQL, których dotyczy problem. Room rozwiązuje ten problem.

Room jest zbudowany do pracy z LiveData i RxJava, podczas gdy SQLite nie.

Zadanie do wykonania

https://developer.android.com/codelabs/kotlin-android-training-room-database#0

https://developer.android.com/codelabs/kotlin-android-training-coroutines-and-room#0