Expand Projection V2 / deprecation

Fixed Bug P2 + Add Hotlist

STATUS UPDATE No update yet. Edit

DESCRIPTION yb...@google.com created issue #1 Apr 7, 2020 10:45AM

we've implemented expandProjection to support two goals: https://developer.android.com/jetpack/androidx/releases/room#compiler-options

- optimize queries not to fetch unused columns
- resolve conflicts in result columns when possible

The second goal is causing some issues since we are making assumptions on where query result values are coming from w/o fully understanding the query. Moreover, sqlite does actually have a functionality (`sqlite3_column_origin_name`) to tell us this information if possible but we cannot use it right now since we rely on jdbc.

Since it is not 100% reliable, we've not been able to turn expand projection on by default.

On the other hand, for the performance gains, seems like sqlite can already flatten a query so doing `select name, lastName from User` is not much different than `select name, lastName from (select * from User)` if the result pojo will only read `name` and `lastName`. (discussed w/ SQLite authors).

I think it makes sense for us to change our implementation such that:

a) wrap user query to explicitly list columns if there are unused values in the result set

b) isolate & deprecate expandProjection

c) implement column conflict resolution using `sqlite3_column_origin_name`.

A seems easy to do. Will give it a try.

B seems like it can be a problem if someone relied on the conflict resolution. We can start by isolating the code so that complexity added by expandProjection does not leak

C will require us to ship our own sqlite JNI layer instead of Xerial. Alternatively, we can try to get that into xerial and use the NativeDb interface. Shipping custom sqlite JNI might be useful in the future for other functionality as well (e.g. we can use authorization callbacks to detect accessed tables rather than parsing). We could possibly replace both antlr & xerial which would also allow us to always support latest sqlite version, or even a version defined by the application.

One issue we've discovered during evaluation is database views where we change the view to only reflect what is in the entity. This means, removing / adding expand projection will affect the schema verification. Would be nice to mitigate if we can but i don't see it as a blocker since `expandProjection` was an experimental feature. Unfortunately, in DAC this is not highlighted :/.

✓ Mentioned issues (2) ✓ Links (5) Hide all

Mentioned issues (2)

P2	expandProjection shows warnings " for things that Room already fixes automatically (b/140759491)."	ap...@ #8
P1	Room: "Parameter specified as non-null is null" for right part of multimap with LEFT JOIN "https://issuetracker.google.com/212279118"	el...@ #10

Links (5)

" ... implemented expandProjection to support two goals: https://developer.android.com/jetpack/androidx/releases/room#compiler-options"	yb...@ #1
"https://android-review.googlesource.com/1279514"	ap...@ #6
"https://android-review.googlesource.com/1287954"	ap...@ #7
"https://android-review.googlesource.com/1288456"	ap...@ #8
"https://android-review.googlesource.com/1287074"	ap...@ #9

COMMENTS All comments ↓ Oldest first

yb...@google.com <yb...@google.com> #2 Apr 7, 2020 10:48AM

actually if we wrap user query, we can also start using compile time defined column indices when reading the query instead of calling `columnIndex` since the result order is guaranteed when explicitly given in the query. We cannot remove the `columnIndex` support because of `@RawQuery` but might still be worth doing, especially if we add logic to resolve conflicting column names.

yb...@google.com <yb...@google.com> #3 Apr 7, 2020 10:50AM

WIP CL for the isolation. still needs work but the goal is to limit the extended parsing to the scope of expansion rather than base class ParsedQuery. This will avoid depending on antlr too much if we can replace it w/ sqlite's APIs.

Reporter	yb...@google.co
Type	Bug
Priority	P2
Severity	S2
Status	Fixed
Access	Default access Vie
Assignee	el...@google.cor
Verifier	--
Collaborators	
CC	ya...@google.com yb...@google.com
AOSP ID	--
Estimate	--
Found In	--
Targeted To	--
Verified In	--
In Prod	

Show 1 additional field

ya...@google.com <ya...@google.com> #4 Apr 7, 2020 10:14PM

> a) wrap user query to explicitly list columns if there are unused values in the result set
>
> b) isolate & deprecate expandProjection
>
> c) implement column conflict resolution using sqlite3_column_origin_name.

Will we still support scenario with column renaming like below?

```
@Entity data class User(@PrimaryKey val id: Long, val name: String, val teamId: Long)
@Entity data class Team(@PrimaryKey val id: Long, val name: String)

data class UserWithTeam(
  @Embedded val user: User
  @Embedded(prefix = "team_") val team: Team
)

@Query("SELECT * FROM User INNER JOIN Team AS team_ ON User.teamId = team_.id")
fun all(): UserWithTeam
```

The returned columns from the query are "id, name, teamId, id, name", and they have conflicting names.

In the current implementation, we rename returned columns based on the match between the table alias and the embedded prefix, so the query results in "SELECT User.id AS id, User.name AS name, User.teamId AS teamId, Team.id AS team_id, Team.name AS team_name FROM ...". It seems to me that we can't use this kind of renaming if we wrap the base query.

---

yb...@google.com <yb...@google.com> #5 Apr 8, 2020 01:40AM

yes we cannot use that. the problem is that, the alias matching we make does not really know the column comes from that table.

If we wrap, the order of columns in the response will be pre-defined at compile time (independent of migrations) hence we can pull values from their indices w/o renaming result column names. That'll require shipping custom JNI layer though so not an easy task.

---

ap...@google.com <ap...@google.com> #6 Apr 17, 2020 08:34AM

Project: platform/frameworks/support
Branch: androidx-master-dev

commit 71103042400340f114a0d431a56fc3a12522a47d
Author: Yigit Boyar <yboyar@google.com>
Date: Mon Apr 06 17:28:21 2020

Isolate expand projection related code

Expand projection makes some assumptions which are bound to go wrong unless
we fully parse the query. On the other hand, optimization part can easily
be achieved by wrapping the query.

This CL tries to limit the access of expand-projection related changes (which
failry complicated the parser, necessarily).

For the isolation, this CL reverts the SQLParser and introduces
ExpandableSQLParser for backwards compatibility.
The Query rewriting logic now goes through a QueryRewriter API that we
can eventually swap with the new rewriter.

Even though this CL duplicates code, I thought it is better so that we
don't carry over extra information for ParsedQueries. The downside is
though, if expand projection is on, we'll be parsing the queries twice.

Bug: 153387066
Test: existing tests are passing.
Test: kotlin and java tests now run w/ and w/o expand projection

Change-Id: I28c5370d0eb97b80e0498b7c4783032038e892f1

M room/compiler/src/main/kotlin/androidx/room/parser/ParsedQuery.kt
M room/compiler/src/main/kotlin/androidx/room/parser/SqlParser.kt
A room/compiler/src/main/kotlin/androidx/room/parser/expansion/ExpandableParsedQuery.kt
A room/compiler/src/main/kotlin/androidx/room/parser/expansion/ExpandableSqlParser.kt
M room/compiler/src/main/kotlin/androidx/room/parser/expansion/ProjectionExpander.kt
M room/compiler/src/main/kotlin/androidx/room/processor/Context.kt
M room/compiler/src/main/kotlin/androidx/room/processor/DaoProcessor.kt
M room/compiler/src/main/kotlin/androidx/room/processor/DatabaseProcessor.kt
M room/compiler/src/main/kotlin/androidx/room/processor/QueryMethodProcessor.kt
A room/compiler/src/main/kotlin/androidx/room/processor/QueryRewriter.kt
M room/compiler/src/main/kotlin/androidx/room/verifier/DatabaseVerifier.kt
M room/compiler/src/main/kotlin/androidx/room/writer/QueryWriter.kt
M room/compiler/src/test/data/daoWriter/output/ComplexDao.java
A room/compiler/src/test/kotlin/androidx/room/parser/ExpandableSqlParserTest.kt

M room/compiler/src/test/kotlin/androidx/room/parser/SqlParserTest.kt
M room/compiler/src/test/kotlin/androidx/room/processor/BaseDaoTest.kt
M room/compiler/src/test/kotlin/androidx/room/processor/DaoProcessorTest.kt
M room/compiler/src/test/kotlin/androidx/room/processor/DatabaseProcessorTest.kt
M room/compiler/src/test/kotlin/androidx/room/processor/ProjectionExpanderTest.kt
M room/compiler/src/test/kotlin/androidx/room/processor/QueryMethodProcessorTest.kt
M room/compiler/src/test/kotlin/androidx/room/solver/query/QueryWriterTest.kt
M room/compiler/src/test/kotlin/androidx/room/testing/TestProcessor.kt
M room/compiler/src/test/kotlin/androidx/room/testing/test_util.kt
M room/compiler/src/test/kotlin/androidx/room/writer/DaoWriterTest.kt
M room/integration-tests/kotlintestapp/build.gradle
M room/integration-tests/testapp/build.gradle

https://android-review.googlesource.com/1279514

ap...@google.com <ap...@google.com> #7 Apr 17, 2020 08:34AM

Project: platform/frameworks/support
Branch: androidx-master-dev

commit 79f1e93f2ed962cdc8ac101557f1482c49f457ae
Author: Yigit Boyar <yboyar@google.com>
Date: Thu Apr 16 13:07:16 2020

Move shared SQL parser code into a helper class

Bug: 153387066
Test: existing tests
Change-Id: I93925036fcb829eaf68aa051306214dc51bb6a23

A room/compiler/src/main/kotlin/androidx/room/parser/SingleQuerySqlParser.kt
M room/compiler/src/main/kotlin/androidx/room/parser/SqlParser.kt
M room/compiler/src/main/kotlin/androidx/room/parser/expansion/ExpandableSqlParser.kt

https://android-review.googlesource.com/1287954

ap...@google.com <ap...@google.com> #8 Apr 21, 2020 12:33PM

Project: platform/frameworks/support
Branch: androidx-master-dev

commit bdde5a1a970ddc9007b28de4aa29d60ffa588f08
Author: Yigit Boyar <yboyar@google.com>
Date: Thu Apr 16 16:47:05 2020

Re-factor how errors are dismissed when query is re-written

This CL changes how we handle errors/warnings if query is
re-written.
There was a bug in expandProjection where we would report warnings
for things that Room already fixes automatically (b/140759491).
The solution to that problem (I7659002e5e0d1ef60fc1af2a625c0c36da0664d8)
solved it by deferring validating of columns until after re-write
decision is made. Unfortunately, this required changing PojoRowAdapter
to have a dummy mapping until it is validating, make it hard to use
as it does have a non-null mapping which is not useful.

This CL partially reverts that change and instead rely on the log
deferring logic we have in Context. This way, we don't need to break
the stability of PojoRowAdapter while still having the ability to
drop warnings that room fixes. This will also play nicer when we
have different query re-writing options that can use more information
about the query results.

Bug: 153387066
Bug: 140759491
Test: existing tests pass
Change-Id: I2ec967c763d33d7a3ff02c1a13c6953b460d1e5f

M room/compiler/src/main/kotlin/androidx/room/log/RLog.kt
M room/compiler/src/main/kotlin/androidx/room/processor/QueryMethodProcessor.kt
M room/compiler/src/main/kotlin/androidx/room/solver/TypeAdapterStore.kt
M room/compiler/src/main/kotlin/androidx/room/solver/query/result/PojoRowAdapter.kt

https://android-review.googlesource.com/1288456

ap...@google.com <ap...@google.com> #9 May 6, 2020 05:42AM

Project: platform/frameworks/support
Branch: androidx-master-dev

commit c88bdb3289ecb053e0a1f6888a9e205360d4e088
Author: Yigit Boyar <yboyar@google.com>

Date: Wed Apr 15 17:40:37 2020

Introduce RemoveUnusedColumns annotation

This annotation can be used when a Cursor mismatch happens where
the query returns more columns than the ones that will be used in
the response object.

This is a convenience for developers who can still write * projection
for future proofing while not paying the cost of pulling unused
columns from the database.

It can be used along with a @Query, @Dao or @Database annotation and
will scope the functionality similar to the TypeConverters.

When enabled, query is wrapped in a
SELECT <usedColumns> FROM (<originalQuery>).

SQLite automatically flattens this query so it does not become a perf
problem in the SQLite side either.

Bug: 153387066
Test: QueryMethodProcessorTest, RemoveUnusedColumnsTest

Change-Id: Ibdb7637b1731de155b7fec2783020e8857936d4c

```
M      room/common/api/2.3.0-alpha01.txt
M      room/common/api/current.txt
M      room/common/api/public_plus_experimental_2.3.0-alpha01.txt
M      room/common/api/public_plus_experimental_current.txt
M      room/common/api/restricted_2.3.0-alpha01.txt
M      room/common/api/restricted_current.txt
A      room/common/src/main/java/androidx/room/RewriteQueriesToDropUnusedColumns.java
A      room/compiler/src/main/kotlin/androidx/room/parser/optimization/RemoveUnusedColumnQueryRewriter.kt
M      room/compiler/src/main/kotlin/androidx/room/processor/Context.kt
M      room/compiler/src/main/kotlin/androidx/room/processor/ProcessorErrors.kt
M      room/compiler/src/main/kotlin/androidx/room/processor/QueryMethodProcessor.kt
M      room/compiler/src/main/kotlin/androidx/room/vo/Warning.kt
M      room/compiler/src/test/kotlin/androidx/room/processor/QueryMethodProcessorTest.kt
A      room/compiler/src/test/kotlin/androidx/room/processor/RemoveUnusedColumnsTest.kt
```

https://android-review.googlesource.com/1287074

---

el...@google.com <el...@google.com> Jan 8, 2021 06:02AM

Reassigned to el...@google.com.

---

el...@google.com <el...@google.com> #10 Aug 26, 2022 04:20AM

Marked as fixed.

Closing this bug as this feature has now been deprecated. In Room 2.5.0.alpha-02 we added an algorithm for resolving duplicate columns:
https://issuetracker.google.com/issues/212279118