Blogs

Told you, we love sharing!

Category

Subscribe to (

SQLite Locking and Transaction Handling in Android

ANDROID (HTTP://WWW.TOTHENEW.COM/BLOG/CATEGORY/TECHNOLOGY/ANDROID/)

19 / APR / 2015 BY AKHILESH DUBEY (HTTP://WWW.TOTHENEW.COM/BLOG/AUTHOR/AKHILESH-DUBEYINTELLIGRAPE-COM/) 3 COMMENTS (HTTP://WWW.TOTHENEW.COM/BLOG/SQLITE-LOCKING-AND-TRANSACTION-HANDLING-IN-ANDROID/#COMMENTS)

Share this blog

Email Twitter Facebook LinkedIn Google+

SQLite locking concept comes when we access database from multiple threads. What things should we keep in our mind while designing the database in android (http://www.tothenew.com/mobile-android-application-development-services), let's see.

firstly we create a helper class which extends SQLiteHelper class:

```
1 public class DatabaseHelper extends SQLiteOpenHelper { ... }
```

Now lets we have multiple thread which are writing the data in database like this.

```
//thread-1
DatabaseHelper helper = new DatabaseHelper(getApplicationContext());
SQLiteDatabase database = helper.getWritableDatabase();
database.insert(.....);
database.close();

//thread-2
DatabaseHelper helper = new DatabaseHelper(getApplicationContext());
SQLiteDatabase database = helper.getWritableDatabase();
database.insert(.....);
database.close();
```

Now, when we run this code, we may get following exception:

```
android.database.sqlite.SQLiteDatabaseLockedException: database is
locked (code 5)
```

why it actually occurred?

From **SQLite FAQ**:
Multiple processes can have the same database open at the same time.
But only one process can be making changes to the database at any moment in time.

It means, In multithreading environment, multiple thread can read database but single thread can write to database. i.e. in our case, we are creating many helper instance and one helper instance is responsible for creating one database connection and if we make multiple helper instance then multiple database connection we will have.

Android uses java locking mechanism for Sqlite database to keep access serialized. So, if multiple threads have one database instance, and we are using one helper instance shared between all the threads, so all of our database access code is serial which is managed internally by the SQLite database.

Now, Let's make previous class singleton, so that at any point of time we have only one instance of helper class and database as well.

```
public class DatabaseHelper extends SQLiteOpenHelper {
     private static DatabaseHelper sInstance;
     public static synchronized DatabaseHelper getInstance(Context
context){
          // Use the application context, which will ensure that you
don't accidentally leak an Activity's context.
          if (sInstance == null) {
                sInstance = new
DatabaseHelper(context.getApplicationContext());
          }
          return sInstance;
     }

     private DatabaseHelper(Context context) {
          super(context, DATABASE_NAME, null, DATABASE_VERSION);
     }
}
```

Now, we made our helper class singleton and share same helper object to every threads like this.

```
//thread-1
DatabaseHelper helper =
DatabaseHelper.getInstance(getApplicationContext());
SQLiteDatabase database = helper.getWritableDatabase();
database.insert(".....");
database.close();

//thread-2
DatabaseHelper helper =
DatabaseHelper.getInstance(getApplicationContext());
SQLiteDatabase database = helper.getWritableDatabase();
database.insert(".....");
database.close();
```

Now, when we run this code, we may get another exception:

```
java.lang.IllegalStateException: attempt to re-open an already-closed
object: SQLiteDatabase
```

We get this exception because we call close() database in between the operation. Actually database open/close is file handling operation done inside SQLite database. When we write to database it actually flushed to disk immediately. Android implements the reference count for each operation and whenever we do anything with database, it increments that count by 1, and when that operation done successfully, it decrements the count by 1. When all operations are done, Android closes the database.

There was a method called setLockingEnabled(boolean lockingEnabled) which was deprecated from API level 16. If you set lockingEnabled false you could perform database operations with multiple thread which is not thread safe. Now this method does nothing. Do not use it. So we can make sure that if we use one helper instance shared with multiple thread and we don't call close() on database then SQlite database handle everything, we don't have to worry about.

SQLite Transaction:

Each SQL statement has a new transaction started for it. This is very expensive, since it requires reopening, writing to, and closing the journal file for each statement. This can be avoided by wrapping sequences of SQL statements in a single transaction.

For instance, before begin with transaction let's see how we perform SQL operation without transaction-

```
SQLiteDatabase database = helper.getWritableDatabase();

for (Address address : addressList) {
ContentValues values = new ContentValues();
values.put(fieldAddressId, address.getId());
values.put(fieldAddressName, address.getName());
database.insert(tableName, null, values);
}
```

Now, there are three basic method for transaction handling in android i.e.

```
database.beginTransaction();
database.setTransactionSuccessful();
database.endTransaction();
```

Let's see how the transaction handling done in android by this example:

```
SQLiteDatabase db = helper.getWritableDatabase();

db.beginTransaction();

String sql = "INSERT INTO " + ADDRESS_TABLE + "
(house_id,house_name)" + " VALUES(?,?)";

SQLiteStatement insert = db.compileStatement(sql);

try {
     for (Address address : addressList) {
          insert.bindString(1, address.getHouseId());
          insert.bindString(2, address.getHouseName());
          insert.execute();
          insert.clearBindings();
     }
} finally {
     db.endTransaction();
}
db.setTransactionSuccessful();
db.endTransaction();
db.close();
```

Tag -

Android (Http://Www.Tothenew.Com/Blog/Tag/Android/)

Locking (Http://Www.Tothenew.Com/Blog/Tag/Locking/) SQLite (Http://Www.Tothenew.Com/Blog/Tag/Sqlite/)

Transaction (Http://Www.Tothenew.Com/Blog/Tag/Transaction/)

Whitepaper
Building Mobile Applications with MBaaS

Download Whitepaper (http://insights.tothenew.com/mobility-building-mobile-applications-with-mbaas)

COMMENTS (3)

Oankredibili Jahn -October 11, 2017 at 6:13 pm (http://www.tothenew.com/blog/sqlite-locking-and-transaction-handling-in-android/comment-page-1/#comment-189929)

The transaction example doesn't make any sense to me.
There should not be any code between "beginTransaction()" and "try" – if there's a crash the transaction remains open. Maybe there's a rollback eventually in the future but not immediately like in the finally block if transaction wasn't set successful.
"db.setTransactionSuccessful();" has to be in the try block right behind the loop,
"db.endTransaction();" only belongs to the finally block. Not after it.
Does it work at all if you end a transaction twice?
I would not close a db when I didn't open it.

Reply (/Blog/Sqlite-Locking-And-Transaction-Handling-In-Android/?Replytocom=189929#Respond)

riya -April 6, 2017 at 11:53 am (http://www.tothenew.com/blog/sqlite-locking-and-transaction-handling-in-android/comment-page-1/#comment-178911)

nice blog

Reply (/Blog/Sqlite-Locking-And-Transaction-Handling-In-Android/?Replytocom=178911#Respond)

AndroidNames -July 22, 2016 at 9:43 am (http://www.tothenew.com/blog/sqlite-locking-and-transaction-handling-in-android/comment-page-1/#comment-170060)

Great tutorial! Working directly with SQLite in big project really kills me. A lot of bugs occur when there are many different data types.

Reply (/Blog/Sqlite-Locking-And-Transaction-Handling-In-Android/?Replytocom=170060#Respond)

LEAVE A COMMENT -

Name *

Email *

Comment

Submit

by
Akhilesh Dubey (http://www.tothenew.com/blog/author/akhilesh-dubeyintelligrape-com/)

Akhilesh is a Software Engineer with many years of experience in Android app development. He is hard-working, quick-learner and innovative developer. He has experience in a variety of applications including Video, Music, E-Commerce, Automobiles and Banking, as well as extensive experience in Java. He is a self-motivated and techno freak guy who loves to step up to new challenges. He loves to play Table Tennis when not sitting in front of his laptop.

YOU MAY ALSO LIKE

Batch update performance enhancements using SQL withBatch() (http://www.tothenew.com/blog/batch-update-performance-enhancements-using-sql-withbatch/)

Perform any DB operation in an isolated transaction (http://www.tothenew.com/blog/perform-any-db-operation-in-an-isolated-transaction/)

Creating Android Application with Groovy (http://www.tothenew.com/blog/creating-android-application-with-groovy/)

FOLLOW US ON

(https://www.linkedin.com/company/tothenew)

Subscribe to our Blog

Get latest articles straight to your inbox

Subscribe Now

Who We Are +

What We Do +

Knowledge +

Contact Us +

Connect With Us