имеет довольно обширный набор API-методов, которые достаточно просто использовать. В этой статье мы рассмотрели лишь малую часть из них.

Ключевые слова: Язык программирование C#, .NET Framework 4.0, язык XML, Интернет – браузер, API-методы.

Zhumanbaeva A.M, Sambetbaeva A.K, Mirzakhmedova G.A

Development of desktop-application for the social network "VKontakte" in the programming language C #

Summary. The article describes the implementation of a simple desktop-application in C #. To develop desktop-applications used .NET Framework 4.0 and programming language C#. Social network "VKontakte" has a fairly extensive set of API-method that is simple enough to use. In this article, we looked at only a fraction of them.

Keywords: Programming language C#, .NET Framework 4.0, the language XML, Internet - browser, API-methods.

ӘОЖ (378.016.02:004.032.6:574)

Жуманбаева А.М., Самбетбаева А.К., Мирзахмедова Г.А.
(Әл-Фараби атындағы Қазақ Ұлттық университеті
Алматы қ, Қазақстан Республикасы, gulban84@mail.ru)

ANDROID ОЖ үшін SQLite ДЕРЕКТЕР ҚОРЫН ҚОЛДАНУ ЕРЕКШЕЛІКТЕРІ

Аннотация. Бүгінгі таңда Android операциялық жүйесі кең қолданысқа ие болып отыр. Android платформасының танымалдығы күн санап өсіп келе жатқандықтан оның қолданылу аясына қатысты мәселелер де өсіп келе жатыр. Олардың ішінде қысқа уақыт аралығында үлкен көлемді мәліметтерді өңдеуді қажет ететін мәселелер жатады. Android ОЖ мәліметтер қоры ретінде SQLite қолданылады. Бұл мақалада Android операциялық жүйесінде SQLite деректер қорымен жұмыс жасау кезінде кездесетін негізгі мәселелерді және оларды android.database.sqlite пакетінің көмегімен шешу жолдарын қарастырамыз.

Түйін сөздер: SQLite деректер қоры, android.database.sqlite пакеті, SQLiteOpenHelper, SQLiteDatabaseLockedException, PRAGMA, database is closed, corrupted database.

SQLite МҚ жұмыс істеуге арналған android.database.sqlite пакеті бар. Бірақ бұл пакет тек базамен жұмыс ісқол жеткізу мүмкіндігін жүзеге асыруды белгілейтін фреймворк болып табылмайды. Google іздеу жүйесі мәліметтер қорымен жұмыс істеуге арналған ешқандай әдістемелер бермейді. Ресми құжаттамаларда SQLite ("NotePad" және "SearchableDictionary") МҚ қолданылатын екі қарапайым мысалдар ғана келтірілген. Сондықтан программистер мәліметтер қорымен жұмыс істеуді жүзеге асыратын өзіндік жеке тәсілдерді ойлап шығарады, және нәтижесінде көптеген күрделі қиындықтар тудыратын болғандықтан бұл тәсілдер дұрыс болмай шығады.

Мәліметтер қорымен жұмыс істеуді жүзеге асыратын дұрыс тәсілдерді құрастыру өте қиын, олардың толыққанды құжаттандырылмауы және android.database.sqlite пакеттер класының қасиеттері мен ерекшеліктерінің ескерілмеуі негізгі проблемасы болып табылады. Мәліметтер қорымен жұмыс істеу барысында кездесетін келесідей қателіктер кездеседі:

- database is locked – мәліметтер қорына бірнеше ағындық жазбаларды енгізу барасында кездеседі.
- database is closed – бағдарламаның әртүрлі бөліктерін мәліметтер қорымен жұмыс істеу барысында пайда болуы мүмкін, мысалы, Activity және Service.
- corrupted database –мәліметтер қоры сақталған файл қолданушының немесе базаға мәліметтерді енгізу барысында жұмысты кенеттен тоқтану салдарынан бүлінген (телефонды өшіру, ОЖ қателіктер, кеңістіктің тарлығы, SD картадағы шалғайлық секторлар және т.б.) жағдайда пайда болады.
- Мәліметтер қорысен жұмыс жасау барысындағы өнімділіктің төмендігі ішкі блоктау, бәсекелік транзакция, көлемді қағазбастылық, пакеттік өңдеулердің жоқтығы салдарынан пайда болуы мүмкін.

Біз бұл мақалада жоғарыда атап кеткен мәселелердің пайда болу салдарларын және оларды шешу жолдарын қарастырамыз.

Мәліметтер қорын өңдеу барысында бағдарламаның кез келген бөлігі (сервис, presenter, widget …) SQLiteOpenHelper арқылы жүзеге асырылатын болғандықтан, SQLiteOpenHelper объектісі дұрыс жұмыс істеуі мәліметтер қорын құру барысында аса көңіл аударарлық жайттардың бірі болып табылады. Ең алдымен SQLite мәліметтер қорындағы бұғаттау файлдық деңгейде орындалғанын атап өткен жөн. Бұл әр түрлі ағындар мен оларды біріктіру барысындағы өзгерістег арналған бұғаттау жұмыстарына кепілдік бере алады. Сонымен қатар, мәліметтер қорын бірнеше ағындар бойынша оқуға, тек бір ағын бойынша ғана жазуға болады.

Мәліметтер қорына базамен біріктірілген екі немесе бірнеше ағындар арқылы да жазуға болады. Ал енді мәліметтер қоры тек бір ағынмен ғана жазуға болатын болса, онда келесідей екі нұсқа пайда болады:

- Егер сіз ақпараттарды бір қосылыстағы екі ағын арқылы жазып отырсаңыз, онда бір ағын екінші ағын жұмысын аяқтағанша өз жұмысын тоқтата тұрады.
- Ал егер сіз әр түрлі қосылыстағы екі ағын арқылы жазып отырсаңыз, онда қателіктер пайда болып, сіздің ақпараттарының мәліметтер қорына енгізілмейді, ал қосымшалар SQLiteDatabaseLockedException бұғаттау аясынан шығып қалады.

Олай болса қосымшалар үшін тек бір дана SQLiteOpenHelper объектісі болуы қажет, кері жағдайда SQLiteDatabaseLockedException қателігіне бой алдыруымыз мүмкін.

SQLiteOpenHelper объектісі сәйкесінше мәліметтерді жазуға және оқуға арналған getReadableDatabase() және getWritableDatabase() атты екі әдістен тұратыны белгілі. Бірақ, көп жағдайларда connection біреу болады. Шын мәнісінде бұлардың барлығы бір объект болып табылады:

```
SQLiteOpenHelper.getReadableDatabase()==SQLiteOpenHelper.
getWritableDatabase()
```

Яғни, мәліметтер қорындағы ақпараттарды қай әдіс арқылы оқитындығының ешқайндай айырмашылығы жоқ. Бірақ, SQLiteDatabase класының ішінде ең маңызды құжаттандырылмаған ерекшелігі бұл класта өзіндік бұғаттаушы mLock айнымалысы бар. Оқу және жазуға арналған SQLiteDatabase объектісі жалғыз болғандықтан SQLiteDatabase объектісі деңгейіндегі жазбалар бұғатталған жағдайда, сол ақпараттарды оқу да бұғатталады. Бұл үрдіс үлкен көлемді ақпараттарды жазу барасында айқын көрінеді.

SQLiteDatabase класының ішкі бұғаттаулардан бөлек ерекшеліктерінің бірі қолданылып отырған класс (API 11 дейін) транзакцияны тек exclusive transaction режимінде ғана құруға мүмкіндік береді. Осының салдарынан күрделі іс – әрекеттерді орындау барысында мәліметтер қорында могут возникать задержки. Мысалы, мәліметтер қорын іске қосқанда үлкен көлемді мәліметтерді (BLOB енетін ~7000 қатарды) көшіріп оларды базаға сақтау керек болсын делік. Егер мәліметтерді транзакцияда сақтасақ, онда мәліметтерді сақтау ~45сек уақытты алады, алайда барлық сұраныстар бұл уақыт аралығында бұғатталғандықтан қолданушы сұраныс жасай алмайды. Егер мәліметтерді қысқа ақпарат түрінде сақтайтын болсақ, онда процесстерді қайта өңдеу жұмысы ұзақ уақытты (~10-15 минут) алатын болады, ал қолданушы ешқандай қиындықсыз барлық уақытта қосымшада орындалатын кез келген амалдарды орындай алады. «Ағаштың екі ұшы» - бұл өте ыңғайлы әрі жылдам.

Android операциялық жүйесінің API 11 интерфейсінен бастап шыққан жаңа нұсқаларында Google компаниясы SQLiteDatabase объектісімен байланысты мәселелерді түзете бастады, яғни келесідей әдістерді қосты:

beginTransactionNonExclusive() – "IMMEDIATE mode" –та транзакция құрады.

yieldIfContendedSafely() – өзге ағындар өз жұмыстарын уақытында орындаулары үшін транзакцияны уақытша тоқтатады

isDatabaseIntegrityOk() – мәліметтер қорының тұтастығына тексеру жұмыстарын жүргізеді.

SQLiteDatabase объектісінде кездесетін қателіктерді түзету үшін, ең алдымен бұғаттауды сөндіріп кез келген жағдайда мәліметте қорындағы ақпараттарды оқуға рұқсат беру қажет.

SQLiteDatabase.setLockingEnabled(false); сұраныстарға java класының (SQLite терминіндегі бұғаттау жұмыстарымен байланыссыз) логикалық деңгейінде ішкі бұғаттау жұмыстарын қолдануды болдырмайды.

SQLiteDatabase.execSQL (“PRAGMA read_uncommitted = true;”); мәліметтерді кештен оқуға мүмкіндік береді. Ақпараттардың мағынасына оқшаулау деңгейін өзгертеді. Бұл параметр әрбір қосылыстар үшін қайта орнатылуы қажет. Егер қосылыстар саны көп болса, онда осы команданы шақырған қосылыс үшін ғана амалдар орындалады.

SQLiteDatabase.execSQL(“PRAGMA synchronous=OFF”); Деректер қорына мәліметтерді «синхрондаусыз» жазу тәсілін өзгертеді. Бұл опцияны өшіретін болсақ, жүйе жұмысының кенеттен тоқтауы немесе электр қорегінің сөнуіне байланысты мәліметтер қоры зақымдалуы мүмкін. Бірақ, SQLite құжаттамасына сәйкес кейбір амалдар осы опцияның сөнуіне байланысты өз жұмысын 50 есе жылдам орындайды.

Өкінішке орай, Android операциялық жүйесі кез келген PRAGMA-ны қолдай бермейді, мысалы, Android операциялық жүйесі “PRAGMA locking_mode = NORMAL” және “PRAGMA journal_mode = OFF” және т.с.с. қолдамайды. PRAGMA мәліметтерін шақыру барысында қосымшада қателіктер пайда болуы мүмкін.

setLockingEnabled әдістемесіне арналған құжаттамаларда, бұл әдісті тек сіз барлық мәліметте қорымен байланысты жұмыстардың барлығы бір ағынмен орындалып тұрғанына сенімді болған жағдайда ғана қолдану ұсынылғаны жайында айтылған болатын. Ол үшін бір уақыт мезетінде тек бір ғана транзакция орындалатынына біз өзіміз кепілдік беруімізге тура келеді. Сонымен қатар үнсіздік бойынша орындалатын транзакцияның (exclusive transaction) орнына immediate transaction транзакциясын қолдану қажет. Android (API 11 төмен) операциялық жүйесінің ескі нұсқаларында java қаптамасы арқылы immediate transaction транзакциясын қолдану мүмкіндігі жоқ, бірақ SQLite деректер қоры бұл функционалды қолдайды. immediate mode-та транзакцияны инициализациялау үшін тікелей мәліметтер қорына келесі SQL сұраныстарын execSQL әдісі арқылы орындау қажет:

```
SQLiteDatabase.execSQL("begin immediate transaction");
```

Транзакцияны тікелей сұраныс арқылы инициализациялап отырғандықтан және оны сәйкесінше аяқтау қажет:

```
SQLiteDatabase.execSQL("commit transaction");
```

Енді біз қажетті мәліметтер типіндегі транзакцияны аяқтайтын және инициализациялайтын өзіндік TransactionManager класын жүзеге асыру керек. TransactionManager класының мақсаты өзгертілетін сұраныстардың (insert, update, delete, DDL сұраныстары) барлығы бір ағында орындалатынына кепілдік беру.

"database is closed" проблемасы. Бір Activity базасымен SQLiteOpenHelper объектісі арқылы жұмыс істеу барысында мәліметтер қоры ашық тұрған Activity арқылы ашылуы, ал жабу үшін жабық тұрған Activity арқылы жабылуы қажет. Егер бір мезгілде бірнеше Activity, бірнеше Service мәліметтер қорымен жұмыс істеп тұрса және мәліметтердің бір бөлігін ContentProvider қол жетімді етсе, онда: «мәліметтер қорымен байланысты қай уақытта ашу керек, және қай уақытта жабу керек?» секілді сұрақ туындауы мүмкін. Егер байланысты әрбір сұраныстан соң ашып жауып отырсақ, онда мәліметтер қорына жүгіну жылдамдығы төмендейді, ал егер байланысты қосымшаларды іске қосқан сәтте ашып, шығу барысында жабатын болсақ, онда біздің қосымшадан қай уақытта шығатытымыз белгісіз болар еді (ал егер сервис немесе провайдер қолданылып тұрған болса – онда тек Application.onTerminate() әдісі ғана қалады). Бірақ бұл әдістердің барлығы үлкен көлемді мәліметтер қорымен жұмыс істеу барысында қисынсыз болып табылады. Ал келесі шарттар орындалғанда мәліметтер қорымен байланыс автоматты түрде тоқтатылады:

- Егер бірнеше Activity бір – бірінен тәуелсіз жаңа қосылыстарды ашатын болса, онда жоғарыда “database is locked” бөлігінде атап кеткен қателіктерге жол берілуі мүмкін.
- Егер мәліметтер қорымен байланысты қосымшаларды іске қосу барысында ашып, және Application.onTerminate() барысында жабатын болсақ, онда мәліметтер қорымен байланыс Cursor.getCount() немесе Cursor.onMove() әдістерін кезекті іске қосбарысында жабылуы мүмкін. Егер сәйкесінше осы кластарды толығымен қарастыратын болсақ, онда шарттардың айқын амалдар нәтижесінде dbclose() әдісі шақыратын SQLiteDatabase.onAllReferencesReleased() әдісі шақырылады.

Енді осы мәселерді шешу жолдарын қарастырайық:

1. Мәліметтер қорымен жұмыс істеу барысында мәліметтер қорының ашық не жабық екенін тексеріп отыру қажет, ал егер мәліметтер қоры жабық болса оны қайта ашу қажет.

```
public synchronized SQLiteDatabase getReadableDatabase()
{SQLiteDatabase db;
try {
db = super.getReadableDatabase();
}
catch (SQLiteException e) {
Log.d(Constants.DEBUG_TAG, e.getMessage());
db = reopenDatabase(dbFile.getAbsolutePath(), null);
}
return db;
```

Бұл әдісте айқын кемшіліктер бар – егер біз мәліметтер қорымен жұмыс істеп, сонан соң сілтемесін ашық тұрған нұсқасына сақтап, оны SQLiteDatabase.getReadableDatabase() әдісін шақырмай қолданатын болсақ онда бұл әдіс жүзеге аспайды.

2. Деректер қорына жалған сілтемені қосып, мәліметтер қоры өз жұмыс н тоқтатқанша қолданылады:

```
SQLiteClosable.acquireReference();
```

Мұндай жағдайда қолдан құрылған сілтемелерді алдын ала өшіріп мәліметтер қорын өздігінен жабу керек. Бірақ сілтемелер саны нөлге тең болуы мүмкін, сондықтан сілтемелер санын үздіксіз тексеріп қажет болған жағдайда толықтырып отыру қажет. Бірақ бұл әдіс те аса тиімді болып табылмайды.

3. Әрбір сұраныстан соң мәліметтер қорын ашып және жауып отыру керек. Бір – бірінен тәуелсіз жазба қосылыстарын құру мүмкіндігі бар болғандықтан қателіктер болуы мүмкін сондықтан бұл аса әдіс сенімді тәсіл болып табылмайды. Егер бұл әдісті тек оқу үшін қолдансақ, онда қателіктер кездеспейді, бірақ мәліметтер қоры қосымшасының жұмысын едәуір төмендетеді.

4. Мәліметтер қорына қолжетімділік үшін ContentProvider әдісін қолдану қажет. Және тек бір провайдерді қолданған жөн, осы провайдерге шексіз Uri қосуға болғандықтан оны жүзеге асыру оңай болады. ContentProvider провайдері мәліметтер қорының қалып – күйіне өзі тексеріп реттеп отырады, және операциялық жүйе жабылған сәтте ол ескі провайдерлерді компьютер жадынан өшіріп, өажет болған жағдайда қайта қалпына келтіреді.

"corrupted database" проблемасы. Android телефондарында қосымшаларға өте аз орын бөлінеді. Қосымшалардың дерлік барлығы ақпараттарды сақтау үшін мәліметтер қорын қолданады, егер ақпараттар сақталатын мәліметтер қоры өте үлкен болса оларды SD картада сақтаған жөн. Android операциялық жүйесінің (2.2 және одан төмен) ескі нұсқаларында мәліметтер қорын SQLiteOpenHelper объектісінің стандартты құралдары арқылы SD құруға мүмкіндік бермейді, бірақ ORMLite – тағы AndroidConnectionSource объектісін қолдану арқылы бұл қиындықтарды айналып өтуге болады.

Қолданушыға көрініп тұрған ақпараттардың барлығы өшірілуі мүмкін екенін естен шығармау өажет. Қолданушы мәліметтер қоры сақталған файлды өшіруі немесе өзге де тәсілдер арқылы осы файлды зақымдауы, SD картаны телефоннан шығаруы мүмкін. Бірақ мәліметтер қоры тек қолданушы арқылы зақымдалмайды. Телефон бұл – сенімді қоректендіргіші бар құрылғы, ақпараттардың бір бөлігі жазылмауы мүмкін, мәліметтердің қоры ақпараттарды жүктеу кезінде немесе алдын ала орнатылған мәліметтер қорын қолдану барысында немесе осы секілді көптеген жағдайларда зақымдалуы мүмкін.

Егер программист әзірлеуші мәліметтер қорын қайта қалпына келтіру алгоритмін құрастырмаған жағдайда Android операциялық жүйесі өзі қайтадан мәліметтер қорын құрастырады. Мәліметтер қорын қайта қалпына келтіруге болатын тәсілдер де бар. Мәліметтер қорын қайта қалпына елтіруге болатын ең қарапайым тәсіл – ол қол жетімді кестелерге сұраныс жасау жаңа

мәліметер қорын құру болып табылады. Бірақ ең қолайлысы “VACUUM” командасын орындау – бұл әдіс мәліметтер өорын өайта құрады және мәлімметердің басым бөлігін қайта қалпына келтіреді.

Көп жағдайда мәліметтер қорының қосымшасын алдын ала орнатылған мәліметтер арқылы құру қажет болады. ол дайын мәліметтер қорын жинақтап, raw папкасында сақтау қажет, ал мәліметтер қорының қосымшасын орнату барысында мәліметтер қоры өздігінен құрылғыға көшіріледі. Мәліметтер қоры құрылған файлды raw папкасына орналастырған жөн. Assets папкасы архививтеп сығуға мүмкіндік беретін болғандықтан бұл папкада мәліметтер қорын сақтау ыңғайлы болып келеді, бірақ бұл папкадан көлемі 1 мб – тан артық мәліметті көшіре алмаймыз, сондықтан бұл папканы көлемі 1мб болатын бірнеше файлдарға бөлуге тура келеді. Ең маңыздысы мәліметтер қорын барлық уақытта ең кішкентай нұсқадағы эмуляторда жинақтау қажет. Себебі, егер алдын ала құрастырылған мәліметтер қорын Android операциялық жүйесінің Android 2.3 нұсқасы үшін жинақтайтын болсақ, онда бұл мәліметтер қорын Android 2.2 нұсқасына орнатқанда “corrupted database” қателігі пайда болады, бірақ Android 2.3 және одан жоғары нұсқаларында жұмыс жасаған өте ыңғайлы болып келеді.

Сұраныстарды орындау жылдамдығы көптеген факторлар арқылы дестеленеді, бірақ олардың ең маңыздысы мәліметтер қорының құрылымын және сұранысты тиімділеу болып табылады. Сұраныстарды тиімділеу үшін ғаламтор арқылы көшіріп алуға болатын көптеген стандартты әдістер бар, сондықтан SQLite деректер қорына арналған тиімділеу тәсілдерінің ерекшеліктерін атап өтейік. Көлемі 1 мб-тан асып кететін мәліметтерді немесе 1000 қатардан артық символды шығаратын сұраныстарды жазу қажет емес. Барлық уақытта limit операторын қалданып отыру қажет. Егер сұраныстар 1000 қатардан артық символды шығаратын, онда сізге сол туралы ескерту келеді, немесе қосымша қызметінің төмендеуіне алып келеді, бұл құрылғы жадысындағы бос орындар мен құрылғының өзіне тәуелді болады. егер ұзын тізімдерді сипаттап бейнелеу қажет болса, оларды келесідей екі жолмен шешуге болады:

• Тізімдерге бөлшектеп сұраныс жасап, оларды android.database.CursorJoiner әдісінің көмегімен біріктіруге болады.

• Интерфейсте авто толықтырушы (тізімді үстемелеу) тізімді жүзеге асыру қажет.

Бір сұраныс екі бөлек сұраныстарға қарағанда едәуір жылдам өызмет етеді. Ең дұрысы 1 сұранысты орындап, join әдісін қолдану. Where операторы бойынша таңдалатын қатарлардың декарттық көбейтіндісіне тап болмау үшін join әдісінің шектік тәртібін сақтау қажет.

Егер мәліметер қорында қандайда бір өзгерістер жасау қажет болса оны транзакцияда орындаған жөн. Бұл тек мәліметтердің тұтастығына ғана кепілдік беріп қоймай, сонымен қатар тапсырманының жүзеге асуын жылдамдатады. Кез келген мәліметтер өорына өзгерістер енгізетін болсақ, мәліметтер қоры құрылған файлдың жанынан өзгерістер енгізу файлы автоматты түрде құрылады. Егер сіз 100 insert командасын орындасаңыз, онда 100 рет мәліметтер файлы құрылады да өзгеріс файлы жойылады, ал егер сіз осы амалды транзакцияда орындасаңыз, онда мәліметтер файл тек бір рет қана құрылады. Егер сіз алдын ала құрылған мәліметтер арқылы кесте құру қажет болса, онда амалдардың орындалу уақытын жылдамдату үшін INSERT AS SELECT (INSERT командасын жеке қолдануға болмайды) командасын қолданыңыз. Егер сіз мәліметтер қорынан үлкен көлемде ақпараттарды бірден алған болсаңыз, және осы сұраныс жиі қаталанатын болса, онда **SQLiteDatabase.releaseMemory()** арқылы жадыдағы өажетсіз ақпараттарды өшіріңіз. Where операторында алдымен қарапайым шарттарды жазу қажет. Мысалы:

```
SELECT * FROM tablename WHERE col1 LIKE '%string%' AND col2 = 123456
```

командасы келесі командаға қарағанда 3 – 4 есе баяу жұмыс жасайды:

```
SELECT * FROM tablename WHERE col2=123456 AND col1 LIKE '%string%'
```

Кестелерді дұрыстап индексациялау сқраныстардың жұмысын 5 – 7 есе жылдамдатады. Ең алдымен join әдісі арқылы орындалатын өрістерді, содан соң іздеу жүргізілетін өрістерді индексациялау қажет. Индекстің жұмыс бағытын көрсету ең тиімді тәсіл болып табылады, мысалы:

```
CREATE INDEX index_name ON table_name (column_name ASC).
```

Үлкен кестелерде іздеу командасымен қатар FTS3 командасын қолданған жөн, бұл команда кестелер бойыншпа текстік іздеу жұмысын жылдамдатады. іздеу жұмыстары үшін LIKE командасының орнына MATCH командасын қолданған жөн, бірақ үндеместік заңы бойынша MATCH тіртұтас сөздерді іздеу командасы болып табылады.

ӘДЕБИЕТ

1. Колисниченко Д.Н. Программирование для Android. Самоучитель // БХВ – Петербург, 2012г.
2. Google. Android 2.3. Руководство пользователя// Google Inc, 2010г.
3. В.Г. Олифер, Н.Ф. Олифер. Сетевые операционные системы // Питер, 2009г.
4. Браин Хардн, Билл Филлипс. Программирование под Android // Питер, 2014г.

REFERENCES:

1. Kolisnichenko D.N. Programmirovaniye dlya Android. Samouchitel' // BKHV – Peterburg, 2012g.
2. Google. Android 2.3. Rukovodstvo pol'zovatelya// Google Inc, 2010g.
3. V.G. Olifer, N.F. Olifer. Setevyye operatsionnyye sistemy // Piter, 2009g.
4. Brain Khardn, Bill Fillips. Programmirovaniye pod Android // Piter, 2014g.

Жуманбаева А.М., Самбетбаева А.К., Мирзахмедова Г.А.

Android ОЖ үшін SQLite деректер қорын қолдану ерекшеліктері

Түйіндеме. Мақалада Android операциялық жүйесінде SQLite деректер қорымен жұмыс жасау кезінде кездесетін негізгі мәселелерді қарастырылды. Бірақ өкінішке орай API интерфейсінде көптеген кемшіліктер кездесіп жатады, ал мәселелерді шешу үшін қажетті құжаттамалар саны өте аз, сонымен қатар Android операциялық жүйесімен жұмыс істеу барысында жүйенің өзінде көптеген қателіктер пайда болады. Android API интерфейсінің жаңа нұсқалары шыққан сайын бұл интерфейсте жұмыс істеу оңай әрі ыңғайлы, көптеген қателіктер түзетіліп, құжаттамалар кеңейтіліп келеді.

Түйін сөздер: SQLite деректер қоры, android.database.sqlite пакеті, SQLiteOpenHelper, SQLiteDatabaseLockedException, PRAGMA, database is closed, corrupted database.

Жуманбаева А.М., Самбетбаева А.К., Мирзахмедова Г.А.

Особенности использование для ОС Android базу данных SQLite

Резюме. В статье рассмотрена основные проблемы при работе с SQLite в Android. К сожалению, в API еще существует очень много пробелов, для решения ряда вопросов отсутствует необходимая документация, а также в процессе работы периодически выявляются ошибки в самой системе. Но радует тот факт, что с каждой новой версией Android API становится все гибче и полнее, ошибки исправляются, а документация расширяется.

Ключевые слова: Базы данных SQLite, пакет android.database.sqlite, SQLiteOpenHelper, SQLiteDatabaseLockedException, PRAGMA, database is closed, corrupted database.

Zhumanbaeva A.M, Sambetbaeva A.K, Mirzakhmedova G.A

Features for use on Android SQLite database

Summary. The article discusses the main challenges when working with SQLite in Android. Unfortunately, the API still a lot of gaps to address a number of issues lack the necessary documentation, as well as in the process periodically to detect errors in the system. But encouraged by the fact that with each new version of Android API is becoming more flexible and better, errors are corrected and expanded documentation.

Keywords: Databases SQLite, package android.database.sqlite, SQLiteOpenHelper, SQLiteDatabaseLockedException, PRAGMA, database is closed, corrupted database.

УДК 004.312.22

Nigay A.M.
(International Information Technologies University, Kazakhstan, Almaty,
extranam88@gmail.com)

LOGIC GATES AS WIREWORLD CELLULAR AUTOMATON PATTERNS

Abstract. One of the unconventional computing methods that can potentially supersede transistors is the computation with cellular automata. One of such cellular automata is the Wireworld. It is possible to construct logic gates, and thus, computing devices, inside Wireworld. Multiple designs of logic gates and other components exist,