УДК 004.451.7.031.43

ОСОБЕННОСТИ ЖУРНАЛЬНЫХ РЕЖИМОВ РАБОТЫ БАЗЫ ДАННЫХ SQLITE ПРИ ИСПОЛЬЗОВАНИИ В КАЧЕСТВЕ ХРАНИЛИЩА CORE DATA

В.В. НИКОЛАЕНКО, И.Н. ЦЫРЕЛЬЧУК

Белорусский государственный университет информатики и радиоэлектроники
ул. П. Бровки, 6, г. Минск, 220013, Республика Беларусь
uladzimir.nikalayenka@gmail.com

При разработке iOS приложений зачастую при реализации каких-то задач используется база данных. В iOS SDK есть отличное решение CoreData, которое поддерживает различные виды файлов, используемых непосредственно в качестве хранилища данных. Одним из этих хранилищ может являться sqlite база данных, которая представляет собой один файл-хранилище. Однако существуют различные режимы работы и доступа к данным в этом файле, которые могут повлиять на работоспособность приложения.

Ключевые слова: iOS, sqlite, CoreData, CocoaTouch, iPhone.

Существует 6 журнальных режимов работы с базой данных SQLite: DELETE, TRUNCATE, PERSIST, MEMORY, WAL, режим выключенного журнала. По умолчанию в iOS используется один из них. В iOS 6 используется DELETE, в iOS 7 – WAL. Данное изменение при обновлении системы может повлиять на работоспособность приложений из-за своих различий в работе с файлом журнала.

При использовании DELETE при проведении транзакции (записи в базу данных) в месте хранения .sqlite файла создается вспомогательный файл с расширением .sqlite-journal. Именно в этот файл и записываются новые данные, одновременно в файле .sqlite-shm помечаются страницы базы данных, которые были подвергнуты изменениям. Сам файл .sqlite на время записи остается нетронутым (рис. 1). После завершения транзакций все данные синхронизируются в файл базы данных, а файл журнала удаляется. Подобным образом гарантируется целостность базы данных.

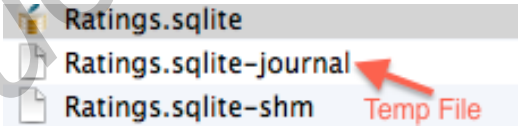

Рис. 1. Работа с журналом в режиме DELETE

В последней версии операционной системы режим по умолчанию был изменен с DELETE на WAL. В WAL режиме файл журнала (расширение .sqlite-wal) существует не только при проведении транзакции. Он создается при открытии базы данных и остается на все время работы.

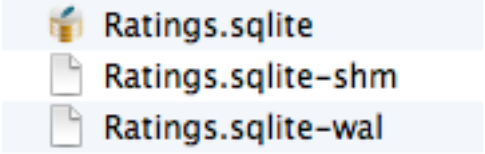

Рис. 2. Работа с журналом в режиме WAL

Принцип работы схож с DELETE – все транзакции записываются в файл журнала, в файле схемы помечаются страницы, которые были изменены. Отличаются механизмы синхронизации файла журнала в базу данных. В WAL режиме журнал синхронизируется в двух случаях: если файл журнала достиг размера в 1000 страниц БД или при открытии базы данных. Операция синхронизации журнала называется checkpoint. Исходя из этого можно утверждать, что фактически база данных в режиме WAL хранится в нескольких файлах: файле БД и вспомогательных файлах. Это помогает реализовать одновременное чтение/запись в БД [1].

Так же необходимо упомянуть другие особенности WAL режима:

1. Если файл базы данных был открыт в WAL режиме, то он останется в нем до тех пор пока базе данных не будет принудительно изменен режим работы с БД через pragma journal_mode.
2. Если файл журнала будет синхронизирован в другую базу данных, то она будет повреждена. При открытии такой базы будет сгенерирована исключительная ситуация SQLITE error code: 11 ‘database disk image is malformed’.

В некоторых приложения реализовано обновление «по воздуху» базы данных, путем полной замены файла .sqlite. В случае iOS 7, в котором используется WAL режим существует риск повредить обновленную базу данных, если в файле журнала будут какие-либо изменения и он будет синхронизирован в нее (рис 3).

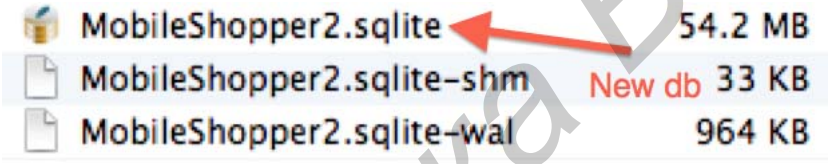

Рис. 3. Синхронизация файла журнала в другой файл базы данных

Это значит что при разработке необходимо учитывать данные особенности. Например, изменить режим работы с .sqlite файлом на уровне CoreData можно при создании persistentStore с параметрами. В качестве параметра нужно указать по ключу NSSQlitePragmasOption необходимый режим работы с базой данных (рис 4).

```
NSDictionary *storeOptions = @{ NSMigratePersistentStoresAutomaticallyOption: @(NO),
                                NSInferMappingModelAutomaticallyOption: @(NO),
                                NSSQLitePragmasOption : @{@"journal_mode" : @"DELETE"}
                                };
NSPersistentStore *persistentStore = [self.persistentStoreCoordinator
                                      addPersistentStoreWithType:NSSQLiteStoreType
                                      configuration:nil URL:storeURL
                                      options:storeOptions error:error];
```

Рис. 4. Инициализация persistentStore
с указанием параметров режима работы .sqlite файла

Еще одним решением подобного рода проблем является принудительная операция checkpoint. Она может быть вызвана путем открытия базы данных. На уровне CoreData это инициализация persistentStore.

Список литературы

1. Write Ahead Logging. [Электронный ресурс]. – Режим доступа: http://www.sqlite.org/wal.html. - Дата доступа: 20.12.2013.