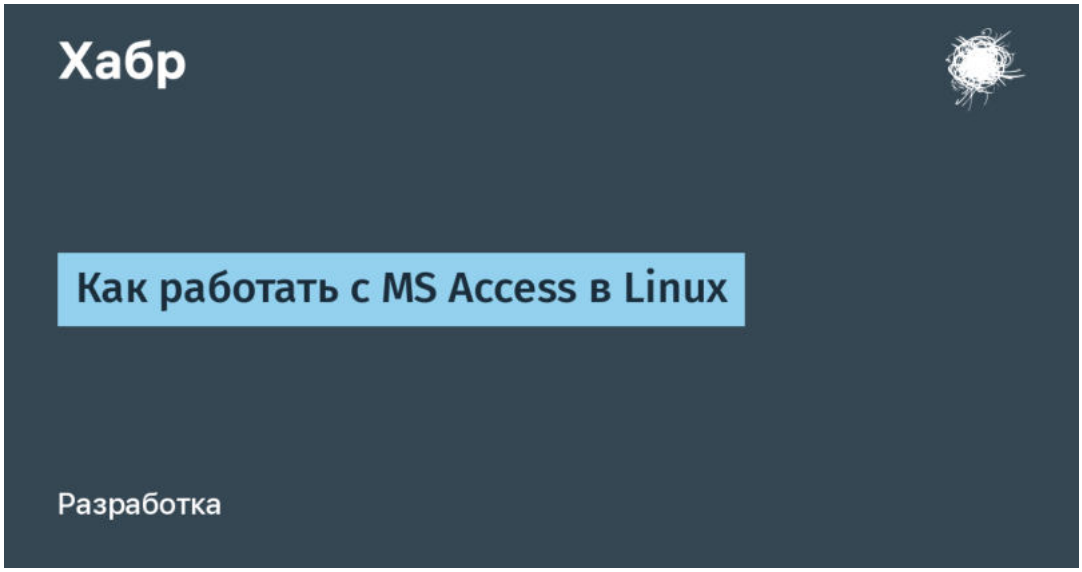
Как работать с MS Access в Linux

Loriowar ⋮⋮ 08.09.2016



Многие пользуются Аксесом… даже в продакшене… даже по сей день. Посему, случаются моменты, когда кому-то захочется подключиться к этой БД из какого-нибудь неожиданного места. Например с юниксового сервера. Конечно же, подключиться захочется не просто так, а для использования данных из Аксеса в веб-приложении. И, без всякого сомнения, появится желание использовать эти данные совместно с информацией из других, более современных БД.

Итак, я хочу описать несколько подходов к общению с существом, называемым MS Access. Посему, исходная задача такова: установить соединение с MS Access из Ruby on Rails приложения или из PostgreSQL (используя FDW) и получить доступ к данным, желательно, в реальном времени.

Ниже я постараюсь собрать всю информацию, относящуюся к вышеописанно задаче и попытаюсь описать нетривиальные случа и подводные табуретки. Надеюсь, это описание сэкономит время кому-нибудь… либо просто, в некоторой степени, позабавит уважаемую публику.

Сразу же tldr для тех кому важны только факты и мнение автора по этому вопросу.

Конвертация в CSV

Для начала, опишу простое рабочее решение. Оно гарантированно работает на Ubuntu 14.04. Должно работать на других дистрибутивах Linux. И не требует каких-либо ~~высших/сакральных~~ знаний, навыков и прочей магии.

Есть такая штука mdbtools. Ставится она очень просто:

```
sudo apt-get install mdbtools
```

Подробности о её зависимостях, ручной сборке, возможностях пакета и о многом другом можно найти на страничке GitHub'а.

Сей пакет предоставляет кучу разных инструментов для работы с Аксесом. Всю кучу рассматривать не будем, а остановимся на одном. Том самом, который умеет превращать mdb-файлы в csv:

```
mdb-export 'mdb-file' 'table-name' > result.csv
```

В результате получим csv-файл с содержимым указанной таблицы. Далее, сей файл можно подвергнуть всем мыслимым и немыслимым обработкам и истязаниям, потому что csv — это безумно простой и широко распространёный формат.

Выполнение запросов в MS Access

Теперь более сложная задача: выполнить SQL-запрос, имея в руках mdb-файл и машину с чем-нибудь юниксовым. Нетрудно догадаться, что нужно поставить ещё пару пакетов и создать несколько конфигов.

Во-первых, понадобится ODBC. Это стандартное API для общения с БД. В Юниксе для этих целей существует unixODBC. Его установка очень проста:

```
sudo apt-get install unixodbc libmdbodbc1
```

Во втором пакете содержится *libmdbodbc.so*, который понадобится чуть ниже.

Следующим шагом нужно найти подходящий ODBC-драйвер для MS Access. Ближайшим доступным является драйвер из mdbtools. Далее, нужно поковыряться в конфигах: описать драйвер и объявить БД.

Драйвер описываем в **/etc/odbcinst.ini**:

```
[MDBTools]
Description     = MDBTools Driver
Driver          = libmdbodbc.so
Setup           = libmdbodbc.so
FileUsage       = 1
UsageCount      = 1
```

А БД объявляем в /etc/odbc.ini:

```
[testdb]
Description = test
Driver      = MDBTools
Database    = /opt/db/MS_Access.mdb
```

Стоит отметить, что в «Driver» нужно указывать имя драйвера, который описан в odbcinst.ini.

Больше про odbcinst.ini и odbc.ini можно найти здесь.

Итак, конфигурирование закончили. Теперь можно приступить к выполнению запросов. Для этих целей воспользуемся утилитой isql из пакета unixODBC:

```
isql testdb
```

Если всё сделано правильно, то должна появиться консоль для выполнения запросов:

```
SQL> SELECT * from "Раздел"
+------------+-----------------------------------------------------+
| Код        | Раздел                                              |
+------------+-----------------------------------------------------+
| 1          | Документация                                        |
| 2          | Сборочные единицы                                   |
| 3          | Детали                                              |
| 4          | Комплекты                                           |
+------------+-----------------------------------------------------+
SQLRowCount returns 4
4 rows fetched
```

На последок, стоит отметить, что существует аналог isql с поддержкой юникода. Он называется iusql.

Странности isql

Честно говоря, утилита isql довольно упорота. В ней куча ограничений на синтаксис и никакого дружелюбия и понимания пользователя. Например: поставил точку с запятой в конце выражения — получи ошибку и попробуй угадай из-за чего она. Никаких подсказок, советов и прочих прелестей современной разработки здесь нет. Это не PotgreSQL, который любезно скажет что вы ошиблись в выражении и предложит правильный вариант. Здесь вас просто пошлют и даже не сообщат причину. Посему, для хоть какого-то облегчения работы с isql была создана оболочка pyodbc-cli. С её помощью можно хоть как-то ослабить борьбу с isql и сосредоточиться на написании запросов.

Экзотические кодировки таблиц/колонок

Ходит много слухов о параметре 'Charset', который влияет на используемую коловую страницу. Вот пример использользования этого параметра:

```
[testdb]
Description = test
Driver = MDBTools
Database = /opt/db/MS_Access.mdb
Charset = CP1251
```

Влияние этого параметра на работу isql замечено не было.В isql я могу работать как с mdb-файлами, содержащими кириллицу, так и со обычными юникодовыми mdb-файлами. В это же время, утилита iusql вне зависимости от параметра 'Charset' выдавала много вопросительных знаков (вот примерно таких: ʃ) при работы с кириллическим mdb-файлом.

Альтернативы для isql

Альтернативой для isql является mdb-sql из пакета mdbtools. Для этой утилиты не нужны ini-файлы. Нужно просто натравить её на конкретный mdb-файл:

```
mdb-sql /opt/db/MS_Access.mdb
```

На все вопросы по использованию утилиты хорошо ответит man-страница. Единственная особенность: вышеупомянутый кириллический mdb-файл утилита проглотить не смогла. С юникодовыми файлами проблем не было.

Путь Ruby/Rails

Сейчас середина 2016 года, последний релиз MS Access был в 22 сентября 2015 года. Но вот незадача, последние работы над адаптером для ActiveRecord датированы 2008 годом. Поэтому, у меня, как это принято, две новости: хорошая и плохая.

Начну с хорошей: существует odbc-rails и его реинкарнация activerecord-odbc-adapter.

А теперь плохая: как уже отмечалось выше, последние коммиты в репозиторий адаптера датированы 2008 годом и заявлена поддержка Rails и ActiveRecord версии один и два; посему, я не знаю как запустить его на Rails 3+ (и можно ли вообще это сделать). Причины моего незнания примерно следующие. Во-первых: у адаптера скверная документация (а скорее её отсутствие). А во-вторых: нет никакого желания лезть в исходники, разбираться и возвращать их к жизни. Так что если у вас достаточно знаний, опыта и времени — можете допилить и описать как этим пользоваться. Удачи вам в этом случае!

Ruby-ODBC

Раз с адаптером всё грустно, то можно посмотреть в другие стороны. Одна из сторон называется ruby-odbc.

Последнее обновление сего гема датировано 2011 годом, но, на текущий момент, он более-менее работает. Для установки гема нужно выполнить нехитрые действия:

```
sudo apt-get install unixodbc unixodbc-dev
gem install ruby-odbc
```

Без пакета unixodbc-dev компиляция native extension отвалится с ошибкой: ERROR: sql.h not found.

Далее, мы предположим, что ODBC в системе сконфигурировано (то есть присутствуют файлы **odbcinst.ini** и **odb.ini**). В этом случае можно открыть irb и сделать следующее:

```
001 > require 'odbc'
 => true
002 > client = ODBC.connect("testdb")
 => #<odbc::database:0x00000000e38d98>
003 > statement = client.prepare 'SELECT * FROM "Раздел"'
 => #<odbc::statement:0x00000000e11608 @_a="[]," @_h="{}," @_c0="{}," @_c1="{},"
@_c2="{}," @_c3="{}">
004 > statement.execute
 => #<odbc::statement:0x00000000e11608 @_a="[]," @_h="{}," @_c0="{}," @_c1="{},"
@_c2="{}," @_c3="{}">
005 > first_row = statement.fetch
 => [1,
"\xD0\x94\xD0\xBE\xD0\xBA\xD1\x83\xD0\xBC\xD0\xB5\xD0\xBD\xD1\x82\xD0\xB0\xD1\x86\xD0\xB8\xD1\x8F\x0(

006 > first_row[1].force_encoding("utf-8")
 => "Документация\u0000"
```

Больше информации о синтаксисе и о доступных командах гема ruby-odbc можно найти в директории ruby-odbc/test на GitHub'e.

Mdb gem

Сей гем предоставляет DSL'ку для работы с mdb-файлами. И она выглядит довольно мило. Но есть нюанс: гем — это просто Ruby-обёртка над вышеописанным mdbtools'ом. То есть, гем конвертирует mdb в csv и обрабатывает этот csv в памяти. Никакой магии и прямого обращения к БД.

Альтернатива для ODBC-драйвера

Существует коммерческая версия ODBC-драйвера для MS Access. Но не существует никакой фактической информации о нём. В оптимистичном варианте этот адаптер поможет с продвинутыми запросами в Access (драйвер из mdbtools много чего не умеет: нет LIMIT, GROUP, AS и тд). Но это только догадки. Что будет на самом деле можно узнать только купив его, либо взяв 14 дневный триал, который доступен после регистрации на сайте. Кроме этой информации не нашлось ни отзывов пользователей, ни каких-либо багрепортов, ни каких-либо упоминаний о том, что кто-то пользовался драйвером и он ему чем-то помог.

Путь PotgreSQL

Для Постгреса существует расширение OGR. Оно является частью GDAL. Который, в свою очередь, является огромной библиотекой по преобразованию растровых и векторных форматов геопространственных данных. Для наших текущих целей назначение библиотеки не имеет решительно никакого значения. Главное, что заявлено, что она умеет работать с mdb-форматом.

Установка

Для начала нужно поставить несколько зависимостей:

```
sudo apt-get install gdal-bin libgdal-dev
sudo apt-get install postgis postgresql-9.3-postgis-2.1
```

Сия команда потянет за собой тонну зависимостей… но это нормально. Первый набор пакетов для ogr_fdw, второй — для postgis.

Шаг второй: сбор pgsql-ogr-fdw из исходников. Вот небольшой мануал в стиле bash:

```
git clone git@github.com:pramsey/pgsql-ogr-fdw.git
cd pgsql-ogr-fdw

sudo apt-get install postgresql-server-dev-9.3
sudo apt-get install checkinstall

make
sudo checkinstall
```

Да, можно взять make install, но мы же не хотим, чтобы котики страдали. В появившемся диалоге от checkinstall нужно обязательно поправить параметр «version». Нужно сделать его в формате «числа разделённые точками» (например: '0.1.0'). Иначе, с дефолтными значениями, сборка пакета упадёт.

Шаг три: пойти и поставить расширения в Postgres:

```
CREATE EXTENSION ogr_fdw;
CREATE EXTENSION postgis;
```

Есть подозрение, что postgis тут лишний, но в Readme на GitHub сказано что нужны оба, посему оставлю этот вопрос пытливым читателям.

Шаг четыре: время создавать FDW. В ogr_fdw есть два возможных пути для работы с Access. Первый использует системный ODBC. Подробности об этом варианте можно найти здесь. Второй — более интересный, использует формат MDB из OGR, который предоставляет прямой доступ к файлу используя Jackcess. Подробности об этом варианте лежат тут. Ниже я опишу оба способа.

Напоследок, одно замечание: OGR — это чрезвычайно мощная штука; возможность работы с MS Access — это маленькая часть всего многообразия доступных форматов и, уважаемый читатель, может вполне резонно заявить, что это пальба из пушки по воробьям… но выбор не велик и кроме этой пушки никаких других орудий изыскать не удалось. И да, вот перечень всех поддерживаемых OGR'ом форматов.

Формат ORG ODBC

Этот подход использует системные настройки ODBC и работает по аналогии с вышеописанным osql и ruby-odbc, но внутри БД. Все доступные опции для инициализации FDW представлены на странице GDAL ODBC драйвера. Ниже я приведу лишь простой пример использования.

Собственно вот он:

```
postgres=# CREATE SERVER testdb_access
postgres-# FOREIGN DATA WRAPPER ogr_fdw
postgres-# OPTIONS(
postgres(#   datasource 'ODBC:testdb',
postgres(#   format 'ODBC');
CREATE SERVER
postgres=# CREATE FOREIGN TABLE access_sections (
postgres(#     "Код" decimal,
postgres(#     "Раздел" varchar)
postgres-#     SERVER testdb_access
postgres-#     OPTIONS (layer 'Раздел');
CREATE FOREIGN TABLE
postgres=# SELECT * FROM access_sections;
ERROR:  unable to connect to layer to "Раздел"
HINT:  Does the layer exist?
```

На сколько я понял из документации OGR, layer — в нашем случае, эквивалентен таблице БД.

Список всех layer'ов можно получить используя утилиту ogrinfo:

```
$ ogrinfo -al 'ODBC:testdb'
geometry_columns is not a table in this database
Got no result for 'SELECT f_table_name, f_geometry_column, geometry_type FROM
geometry_columns' command
```

```
INFO: Open of `ODBC:testdb'
      using driver `ODBC' successful.
```

Основываясь на этом сообщении, можно предположить, что всё работает, но целевая БД (то бишь mdb-файл), не содержит требуемого Geo-формата данных и OGR спотыкается об это досадное недоразумение. Я не знаю как отучить его принудительно проверять формат предоставленной БД. Но некоторые пишут, что сей подход замечательно работает под Windows. В общем, если вы знаете как образумить OGR ODBC и заставить его работать с произвольным mdb-файлом, пожалуйста, скажите об этом, не держите это знание в себе.

Отдельный вопрос: как PG будет работать с кирилическими (да и с любыми другими нелатинскими) названиями таблиц и колонок. С одной стороны — Postgres'y без разницы как называется таблица/столбец, обернуть их в двойные кавычки и хоть спец.символы можно использовать. С другой стороны: кто его знает, применимо ли это к FDW, а проверить на конкретном примере пока не получается.

Формат ORG MDB

Сей подход основывается на Java-библиотеке Jackcess. Так как это Java и у неё свой богатый внутренний мир, то у этого подхода нет никаких связей с системным ODBC и, следовательно, проблемы с драйверами для MS Access для него чужды. Но есть другие особенности, которые опишу ниже.

Сразу предупрежу, что в силу «богатой» документации по всему описываемому процессу, отсутствию большого опыта с Java и некоторой монструозностью целевого пакета, рабочий вариант удалось собрать за 3 дня и ~20 полных пересборок пакета. Поэтому сразу скажу о некоторых вещах:

- этот подход годится только для работы с незашифрованными mdb-файлами (то бишь с файлами без пароля);
- так как это сборка пакета, то все нижеописанные зависимости, пути, версии и прочие атрибуты справедливы для моего конкретного случая и окружения, у вас всё может быть совсем не так и бездумно копипастить команды не рекомендуется.

Итак, всё ниженаписенное является более развёрнутой версией исходного официального описания GDAL ACCESS MDB database драйвера.

Во-первых: нужно поставить openjdk-6-jdk.

```
sudo apt-get install openjdk-6-jdk
```

После бегло вдумчивого чтения исходников GDAL'a, создалось ощущение, что он поддерживает и openjdk-7-jdk. Но у меня не получилось заставить его работать с 7-й версией.

Далее, понадобится libgdal-dev.

```
sudo apt-get install libgdal-dev
```

Здесь нужно запомнить версию пакета. Она напрямую связана с версией пакета GDAL. В моём случае это версия 1.10.1.

Примечание: поддержка формата mdb начинается с версии 1.9.0.

Ну и на последок, нужно снести пакет gdal-bin, так как его расширенную версию мы и собираемся собрать из исходников.

```
sudo apt-get remove gdal-bin
```

Во-вторых: нужно скачать несколько JAR-ов (древних и не очень), а именно: jackcess-1.2.2.jar, commons-lang-2.4.jar и commons-logging-1.1.1.jar; затем, положить их в lib/ext. В моём случае, полный путь до этой директории следующий: **/usr/lib/jvm/java-6-openjdk-amd64/jre/lib/ext**. Вышеозначенные версии JAR-ов можно найти внутри вот этой утилиты. Для меня, всё работает с любой более поздней версией commons-logging (1.*), с любой другой минорной версией commons-lang (2.*) и jackcess (1.*). Ошибки появлялись только при использовании следующей мажорной версии jaccess (2.1.4).

В-третьих: нужно скачать и сконфинурировать GDAL.

```
git clone git@github.com:OSGeo/gdal.git
cd gdal/gdal/
git checkout 1.10
```

Здесь нужно перейти в ветку, соответствующую версии пакета libgdal-dev, коий устанавливали в пункте номер раз. В противном случае собранный бинарник будет несовместим с библиотеками.

Далее нужно звать configure. Существует целых два способа вызова. Простой:

```
./configure --with-java=yes --with-jvm-lib-add-rpath=yes --with-mdb=yes
```

и с явным указанием путей:

```
./configure --with-java=/usr/lib/jvm/java-6-openjdk-amd64 \
            --with-jvm-lib=/usr/lib/jvm/java-6-openjdk-amd64/jre/lib/amd64/server \
            --with-jvm-lib-add-rpath=yes \
            --with-mdb=yes
```

Второй вариант может быть полезен если в системе присутствует несколько версий Java (например, openjdk-6-jdk и openjdk-7-jdk) или если первый вариант не дал желаемого результата.

После окончания работы configure нужно найти заветное слово 'yes' напротив формата MDB.

В-четвёртых: нужно изыскать чашечку чая/кофе или чего покрепче и запустить сборку пакета.

```
sudo checkinstall
```

Здесь нужно ответить на пару несложных вопросов и ждать. В моём случае ждать нужно было примерно 10 минут.

Здесь же нужно отметить, что пакет получится увесистым, около 300мб. Конечно же, можно выкинуть из него всё лишнее, собрать его руками и приблизиться к размеру пакета gdal-bin из репозитория (~900Kb), но это выходит за рамки повествования и, поэтому, описано не будет.

В-пятых: если что-то пошло не так, сборка пакета отвалилась, то гугл и светлый ум вам в помощь.

В-шестых: если всё прошло хорошо, то после checkinstall пакет должен был автоматически установиться и теперь нужно проверить, действительно ли полученные бинарники поддерживают формат mdb:

```
$ ogrinfo --formats | grep MDB
-> "MDB" (readonly)
```

Если в выводе ogrinfo информации про mdb не нашлось, то перейдите к началу этой секции, перечитайте мануалы, посмотрите на зависимости, параметры системы, фазу луны и прочие атрибуты которые могут повлиять на компиляцию и итоговый бинарник, и попробуйте пересобрать всё это хозяйство ещё раз.

Если же команда и вывод совпали, то всё хорошо и самая мутная часть позади. Теперь ogrinfo может работать с mdb-файлами и предоставлять информацию об их содержимом:

```
$ ogrinfo /opt/db/test-database.mdb
INFO: Open of `/opt/db/test-database.mdb'
      using driver `MDB' successful.
1: closeouts
2: economics
```

В-седьмых: теперь можно настраивать FDW в Postgres. Вот небольшой скрипт с примером этого действа:

```
postgres=# CREATE SERVER acc
    FOREIGN DATA WRAPPER ogr_fdw
    OPTIONS (
      datasource '/opt/db/test-database.mdb',
      format 'MDB' );
CREATE SERVER
```

```
postgres=# CREATE FOREIGN TABLE economics(
    ID integer)
    SERVER acc
    OPTIONS(layer 'economics');
CREATE FOREIGN TABLE
postgres=# SELECT * FROM economics;
 id
----
  1
  2
  3
  4
  5
(5 rows)
```

И, в общем-то, всё. В заключении этой секции скажу пару слов про «шифрованные» mdb-файлы.

Если FDW не может вытащить данные из Access, а ogrinfo ругается следующим образом:

```
Exception in thread "main"
com.healthmarketscience.jackcess.UnsupportedCodecException: Decoding not supported.
Please choose a CodecProvider which supports reading the current database encoding.
        at
com.healthmarketscience.jackcess.DefaultCodecProvider$UnsupportedHandler.decodePage(DefaultCodecProv
```

то, скорее всего, у вас запароленный mdb-файл. В этом случае стоит посмотреть на FAQ из Jaccess и задуматься о допиле драйвера OGR Access. На сколько я понял, существует проект Jackcess Encrypt. Сей проект предоставляет CryptCodecProvider, который, в свою очередь, предоставляет реализацию интерфейса CodecProvider для Jackess и поддерживает некоторые форматы шифрования mdb-файлов. Но, к несчастью, текущий драйвер от GDAL никак не умеет работать с Jackcess Encrypt и, следовательно, никак не поддерживает шифрованные файлы. Так что, есть хорошее направление для работы в стане опенсорса.

Прочие FDW

Список всех существующих FDW для Postgres можно найти на официальной вики-странице. Там есть ZhengYang/odbc_fdw, в котором последний коммит датирован 2011 годом. И CartoDB/odbc_fdw, который активно развивается и поддерживает Postgres 9.5+. Так что выбор невелик.

Заключение

Работать с MS Access больно… вдвойне больно если нужно это делать под Linux. Так что сразу добрый совет: вытащите из аксеса данные в любую современную БД и избавитесь от вагона проблем. Если вытащить не получается, то работайте с аксесом в Windows. Там есть нормальный драйвер, предоставляемый Microsoft «из коробки», о стыковке Access и Postgres в Windows есть хоть какие-то статьи и примеры настройки и вообще продукты одной и той же фирмы, обычно, хорошо работают друг с другом. Если же и этой возможности нет, то у вас снова два выхода: превращать всё в CSV и работать с ним или пытаться напрямую обратиться к mdb-файлу. Первый вариант простой, работает «из коробки» и особых навыков не требует. Второй вариант гораздо сложнее, требует времени, нервов, прямых рук, имеет набор ограничений, подводных камней и прочих неприятных вещей. Посему, выбирайте с умом.

Ссылки

- Блог некой девушки SARA SAFAVI про GDAL/OGR под Ubuntu
- Stackoverflow про MS Access под Ubuntu
- Тема на OpenNet
- и много много ссылок, любезно предоставленных корпорацией добра Google