How to convert SQLite SQL dump file to PostgreSQL?

Asked 14 years, 9 months ago Modified 8 months ago Viewed 162k times

152

For development I'm using SQLite database with production in PostgreSQL. I updated my local database with data and need to transfer a specific table to the production database.

Running `sqlite database .dump > /the/path/to/sqlite-dumpfile.sql`, SQLite outputs a table dump in the following format:

```
BEGIN TRANSACTION;
CREATE TABLE "courses_school" ("id" integer PRIMARY KEY, "department_count"
integer NOT NULL DEFAULT 0, "the_id" integer UNIQUE, "school_name"
varchar(150), "slug" varchar(50));
INSERT INTO "courses_school" VALUES(1,168,213,'TEST Name A',NULL);
INSERT INTO "courses_school" VALUES(2,0,656,'TEST Name B',NULL);
....
COMMIT;
```

How do I convert this into a PostgreSQL compatible dump file I can import into my production server?

postgresql sqlite migration

Share Improve this question Follow

edited Jun 20, 2023 at 16:18
user4157124
3,000 ● 19 ● 32 ● 48

asked Jan 3, 2011 at 2:33
DevX
1,934 ● 2 ● 15 ● 17

6 Well, that command did not work for me until I changed **sqlite** to **sqlite3** – Celal Ergün Mar 25, 2018 at 19:57

8 Answers

Sorted by: Highest score (default)

146

You should be able to feed that dump file straight into `psql`:

```
/path/to/psql -d database -U username -W < /the/path/to/sqlite-dumpfile.sql
```

If you want the `id` column to "auto increment" then change its type from "int" to "serial" in the table creation line. PostgreSQL will then attach a sequence to that column so that INSERTs with NULL ids will be automatically assigned the next available value. PostgreSQL will also not recognize `AUTOINCREMENT` commands, so these need to be removed.

You'll also want to check for `datetime` columns in the SQLite schema and change them to `timestamp` for PostgreSQL. (Thanks to Clay for pointing this out.)

If you have booleans in your SQLite then you could convert `1` and `0` to `1::boolean` and `0::boolean` (respectively) or you could change the boolean column to an integer in the schema section of the dump and then fix them up by hand inside PostgreSQL after the import.

If you have BLOBs in your SQLite then you'll want to adjust the schema to use `bytea`. You'll probably need to mix in some `decode` calls as well. Writing a quick'n'dirty copier in your favorite language might be easier than mangling the SQL if you a lot of BLOBs to deal with though.

As usual, if you have foreign keys then you'll probably want to look into `set constraints all deferred` to avoid insert ordering problems, placing the command inside the BEGIN/COMMIT pair.

Thanks to Nicolas Riley for the boolean, blob, and constraints notes.

If you have `` ` `` on your code, as generated by some SQLite3 clients, you need to remove them.

PostGRESQL also doesn't recognize `unsigned` columns, so you might want to drop that or add a custom-made constraint such as this:

```
CREATE TABLE tablename (
    ...
    unsigned_column_name integer CHECK (unsigned_column_name > 0)
);
```

While SQLite defaults null values to `''`, PostgreSQL requires them to be set as `NULL`.

The syntax in the SQLite dump file appears to be mostly compatible with PostgreSQL so you can patch a few things and feed it to `psql`. Importing a big pile of data through SQL INSERTs might take a while but it'll work.

Share Improve this answer Follow edited Sep 16, 2020 at 2:54 community wiki
11 revs, 7 users 62%
mu is too short

Sign up to request clarification or add additional context in comments.

11 Comments

Add a comment

Peter Eisentraut Over a year ago

No, you want to keep the transaction to avoid some overhead.

4 Reply

r03 Over a year ago

You can use to_timestamp() in the postgreSQL to convert a timestamp to a progreSQL timestamp

2 Reply •••

Jeff Hammerbacher Over a year ago

I'd note that the `integer` type in Sqlite defaults to 8 bytes, while the `integer` type in PostgreSQL is 4 bytes. To be safe, you may want to convert your `integer`-typed columns in Sqlite to the `bigint` type in PostgreSQL.

1 Reply •••

Saravanabalagi Ramachandran Over a year ago

Should remove `DELETE FROM sqlite_sequence;` and replace `INSERT INTO "sqlite_sequence" VALUES('locations',7);` with `ALTER SEQUENCE locations_id_seq RESTART WITH 8;` and it works...!

1 Reply •••

Jan Over a year ago

the sqlite3 output has table names with capitals which are quoted in the create table statement, but not in the insert into statements, so inserts fail. Another little gotcha

1 Reply •••

|

98

`pgloader`

I came across this post when searching for a way to convert an SQLite dump to PostgreSQL. Even though this post has an accepted answer (and a good one at that +1), I think adding this is important.

I started looking into the solutions here and realized that I was looking for a more automated method. I looked up the wiki docs:

https://wiki.postgresql.org/wiki/Converting_from_other_Databases_to_PostgreSQL

and discovered `pgloader`. Pretty cool application and it's relatively easy to use. You can convert the flat SQLite file into a usable PostgreSQL database. I installed from the `*.deb` and created a `command` file like this in a test directory:

```
load database
    from 'db.sqlite3'
    into postgresql:///testdb

with include drop, create tables, create indexes, reset sequences

set work_mem to '16MB', maintenance_work_mem to '512 MB';
```

like the docs state. I then created a `testdb` with `createdb`:

```
createdb testdb
```

I ran the `pgloader` command like this:

```
pgloader command
```

and then connected to the new database:

```
psql testdb
```

After some queries to check the data, it appears it worked quite well. I know if I had tried to run one of these scripts or do the stepwise conversion mentioned herein, I would have spent much more time.

To prove the concept I dumped this `testdb` and imported into a development environment on a production server and the data transferred over nicely.

Share Improve this answer Follow

edited Jun 20, 2020 at 9:12

Community Bot

1 • 1

answered May 30, 2015 at 9:30

nicorellius

4,143 • 5 • 56 • 81

5 Comments

Add a comment

silpol Over a year ago

Beware that (still supported) Ubuntu distributions might have outdated version - v2.x.y are already deprecated and don't actually work. v3.2.x might work but v3.2.3 is recommended. I have fetched v3.2.3 from bleeding edge and installed with *sudo dpkg -i <.deb file name>*, it had no problem with dependencies.

▲ 3 Reply •••

BenKoshy Over a year ago

I concur with @silpol - be sure to download the latest stable release and install using your fav package manager; for the "command" file this is just a text file called 'command' with no extension name (i.e. no need for .txt at the end of the file name) you don't need to put the file name in angular brackets; i had to change the search_parth of the psql database in order to see my data; pgloader works well and saved me a great deal of hassle

▲ 1 Reply •••

sveri Over a year ago

Unfortunately it does not work on windows.

▲ 0 Reply •••

user42723 Jan 15 at 13:32

Buggy program, it crashes when I tried it, version 3.6.7 (heap exhausted during garbage collection)

▲ 0 Reply •••

Alex Apr 2 at 6:19

I strongly recommend exporting your Postgres password as an env var `export PGPASSWORD=xxx` and then omitting it from the connection string. So just put e.g. `into postgresql://user@host:port/database` in the `pgloader` command file

0 Reply

35

The sequel gem (a Ruby library) offers data copying across different databases: http://sequel.jeremyevans.net/rdoc/files/doc/bin_sequel_rdoc.html#label-Copy+Databases

First install Ruby, then install the gem by running `gem install sequel`.

In case of sqlite, it would be like this: `sequel -C sqlite://db/production.sqlite3 postgres://user@localhost/db`

Update: In 2025 one seems to need to install sqlite gem before this can be run. Please consult the installation guide.

Share Improve this answer Follow edited Jan 19 at 8:04 answered Jul 20, 2015 at 16:16

lulalala
18k ● 15 ● 114 ● 179

12 Comments

Add a comment

hasufell Over a year ago

Absolutely, pgloader is messy, the GC seems to crash on huge databases: github.com/dimitri/pgloader/issues/962

3 Reply

Felix Over a year ago

Feel free to post your answer at stackoverflow.com/questions/6148421/… where I copied your answer. Then ping me and I will revoke my answer if you want the reps for it.

0 Reply

user42723 Jan 15 at 13:43

Doesn't work when I try it: `Error: Sequel::AdapterNotFound: LoadError: cannot load such file -- sqlite3`

0 Reply

lulalala Jan 16 at 7:35

@user42723 Can you try running `gem install sqlite3`?

0 Reply

user42723 Jan 16 at 12:46

@lulalala I tried that already, "ERROR: Failed to build gem native extension." The easiest for me was to just write some code with SELECT and INSERT queries.

0 Reply

|

26

You can use a one liner, here is an example with the help of sed command:

```
sqlite3 mjsqlite.db .dump | sed -e 's/INTEGER PRIMARY KEY AUTOINCREMENT/SERIAL
PRIMARY KEY/g;s/PRAGMA foreign_keys=OFF;//;s/unsigned big
int/BIGINT/g;s/UNSIGNED BIG INT/BIGINT/g;s/BIG INT/BIGINT/g;s/UNSIGNED
INT(10)/BIGINT/g;s/BOOLEAN/SMALLINT/g;s/boolean/SMALLINT/g;s/UNSIGNED BIG
INT/INTEGER/g;s/INT(3)/INT2/g;s/DATETIME/TIMESTAMP/g' | psql mypqdb mypguser
```

Share Improve this answer Follow edited Mar 2, 2022 at 6:57 answered Sep 18, 2014 at 23:30

develCuy
655 ● 7 ● 14

7 Comments

Add a comment

yetanothercoder Over a year ago
there is no replace for LONG type, e.g.

0 Reply

silpol Over a year ago
one more item could be added `sed -e 's/DATETIME/TIMESTAMP/g'`

2 Reply

Purplejacket Over a year ago
`sed -e 's/TINYINT(1)/SMALLINT/g'` -- and for a comparison of all the data types see stackoverflow.com/questions/1942586/…

2 Reply

Cerin Over a year ago
This doesn't work. I get the error `ERROR:  syntax error at or near "AUTOINCREMENT"`

0 Reply

AstraSerg Over a year ago
Replace `' | sed -e '` with `;` :)

2 Reply

|

1. Dump SQLite database to JSON:

```
python3 manage.py dumpdata > data.json
```

2. Change connection from SQLite to PostgreSQL.
3. Create tables without migration:

```
python3 manage.py migrate --run-syncdb
```

4. Open Django shell, then exclude `ContentType` data:

```
python3 manage.py shell
from django.contrib.contenttypes.models import ContentType
ContentType.objects.all().delete()
quit()
```

5. Load data:

```
python3 manage.py loaddata data.json
```

Share Improve this answer Follow

edited Sep 29, 2024 at 3:42

user4157124
3,000 ● 19 ● 32 ● 48

answered Nov 22, 2020 at 21:54

Kusal Thiwanka
339 ● 3 ● 5

4 Comments

Add a comment


giveJob Over a year ago
for large GB JSON files huge ram required

1 Reply •••


ValheruBorn Over a year ago
Thanks a million for this. Why should content types be excluded?

0 Reply •••


JessieinAg Jan 18 at 21:52
Thank you so much! This is the perfect solution for Django projects.

0 Reply •••


texnic May 25 at 15:45
Insteaf of `> data.json`, one should use `-o data.json`. Otherwise cannot import back (dumping adds a text line infront of JSON).

0 Reply •••

pgloader work wonders on converting database in sqlite to postgresql.

2

Here's an example on converting a local sqlitedb to a remote PostgreSQL db:

pgloader **sqlite.db** postgresql://**username**:**password**@**hostname**/**dbname**

Share Improve this answer Follow

answered Sep 4, 2018 at 22:00

kouichi

106 ● 8

1 Comment

Add a comment

Cerin Over a year ago

Pgloader is terribly buggy and unreliable. It immediately crashes with the error `KABOOM! Control stack exhausted (no more space for function call frames).`

3 Reply

1

I have tried editing/regexping the sqlite dump so PostgreSQL accepts it, it is tedious and prone to error.

What I got to work really fast:

First recreate the schema on PostgreSQL without any data, either editing the dump or if you were using an ORM you may be lucky and it talks to both back-ends (sqlalchemy, peewee, ...).

Then migrate the data using pandas. Suppose you have a table with a bool field (which is 0/1 in sqlite, but must be t/f in PostgreSQL)

```
def int_to_strbool(df, column):
    df = df.replace({column: 0}, 'f')
    df = df.replace({column: 1}, 't')
    return df

#def other_transform(df, column):
#...

conn = sqlite3.connect(db)
df = pd.read_sql(f'select * from {table_name}', conn)

df = int_to_strbool(df, bool_column_name)
#df = other_transform(df, other_column_name)

df.to_csv(table_name + '.csv'), sep=',', header=False, index=False)
```

This works like a charm, is easy to write, read and debug each function, unlike (for me) the regular expressions.

Now you can try to load the resulting csv with PostgreSQL (even graphically with the admin tool), with the only caveat that you must load the tables with foreign keys after you have loaded the tables with the corresponding source keys. I did not have the case of a circular dependency, I guess you can suspend temporarily the key checking if that is the case.

Share Improve this answer Follow

answered May 8, 2019 at 6:22

agomcas

705 ● 5 ● 12

Comments

Add a comment

0

install `pgloader`:

```
sudo apt-get install pgloader
```

then:

```
pgloader sqlite:///path/to/sqlite/database.sqlite
postgresql://username:password@hostname:port/databasename
```

that's it!

Share Improve this answer Follow

answered Apr 24, 2024 at 10:09

Mohammad Amin Eskandari

384 ● 5 ● 8

Comments

Add a comment

Start asking to get answers

Find the answer to your question by asking.

Ask question

Explore related questions

postgresql **sqlite** **migration**

See similar questions with these tags.