24.1. SQL Dump

The idea behind this dump method is to generate a text file with SQL commands that, when fed back to the server, will recreate the database in the same state as it was at the time of the dump. PostgreSQL provides the utility program pg_dump for this purpose. The basic usage of this command is:

```
pg_dump dbname > outfile
```

As you see, pg_dump writes its results to the standard output. We will see below how this can be useful.

pg_dump is a regular PostgreSQL client application (albeit a particularly clever one). This means that you can do this backup procedure from any remote host that has access to the database. But remember that pg_dump does not operate with special permissions. In particular, it must have read access to all tables that you want to back up, so in practice you almost always have to run it as a database superuser.

To specify which database server pg_dump should contact, use the command line options `-h host` and `-p port`. The default host is the local host or whatever your `PGHOST` environment variable specifies. Similarly, the default port is indicated by the `PGPORT` environment variable or, failing that, by the compiled-in default. (Conveniently, the server will normally have the same compiled-in default.)

Like any other PostgreSQL client application, pg_dump will by default connect with the database user name that is equal to the current operating system user name. To override this, either specify the `-U` option or set the environment variable `PGUSER`. Remember that pg_dump connections are subject to the normal client authentication mechanisms (which are described in Chapter 21).

Dumps created by pg_dump are internally consistent, that is, the dump represents a snapshot of the database as of the time pg_dump begins running. pg_dump does not block other operations on the database while it is working. (Exceptions are those operations that need to operate with an exclusive lock, such as most forms of `ALTER TABLE`.)

> **Important:** If your database schema relies on OIDs (for instance as foreign keys) you must instruct pg_dump to dump the OIDs as well. To do this, use the `-o` command line option.

24.1.1. Restoring the dump

The text files created by pg_dump are intended to be read in by the psql program. The general command

form to restore a dump is

```
psql dbname < infile
```

where `infile` is what you used as `outfile` for the pg_dump command. The database `dbname` will not be created by this command, so you must create it yourself from `template0` before executing psql (e.g., with `createdb -T template0 dbname`). psql supports options similar to pg_dump's for specifying the database server to connect to and the user name to use. See the psql reference page for more information.

Before restoring a SQL dump, all the users who own objects or were granted permissions on objects in the dumped database must already exist. If they do not, then the restore will fail to recreate the objects with the original ownership and/or permissions. (Sometimes this is what you want, but usually it is not.)

By default, the psql script will continue to execute after an SQL error is encountered. You might wish to use the following command at the top of the script to alter that behaviour and have psql exit with an exit status of 3 if an SQL error occurs:

```
\set ON_ERROR_STOP
```

Either way, you will have an only partially restored database. Alternatively, you can specify that the whole dump should be restored as a single transaction, so the restore is either fully completed or fully rolled back. This mode can be specified by passing the `-1` or `--single-transaction` command-line options to psql. When using this mode, be aware that even the smallest of errors can rollback a restore that has already run for many hours. However, that might still be preferable to manually cleaning up a complex database after a partially restored dump.

The ability of pg_dump and psql to write to or read from pipes makes it possible to dump a database directly from one server to another, for example:

```
pg_dump -h host1 dbname | psql -h host2 dbname
```

> **Important:** The dumps produced by pg_dump are relative to `template0`. This means that any languages, procedures, etc. added via `template1` will also be dumped by pg_dump. As a result, when restoring, if you are using a customized `template1`, you must create the empty database from `template0`, as in the example above.

After restoring a backup, it is wise to run ANALYZE on each database so the query optimizer has useful statistics. An easy way to do this is to run `vacuumdb -a -z`; this is equivalent to running `VACUUM ANALYZE` on each database manually. For more advice on how to load large amounts of data into PostgreSQL efficiently, refer to Section 14.4.

24.1.2. Using pg_dumpall

pg_dump dumps only a single database at a time, and it does not dump information about roles or tablespaces (because those are cluster-wide rather than per-database). To support convenient dumping of the entire contents of a database cluster, the pg_dumpall program is provided. pg_dumpall backs up each database in a given cluster, and also preserves cluster-wide data such as role and tablespace definitions. The basic usage of this command is:

```
pg_dumpall > outfile
```

The resulting dump can be restored with psql:

```
psql -f infile postgres
```

(Actually, you can specify any existing database name to start from, but if you are reloading into an empty cluster then `postgres` should usually be used.) It is always necessary to have database superuser access when restoring a pg_dumpall dump, as that is required to restore the role and tablespace information. If you use tablespaces, be careful that the tablespace paths in the dump are appropriate for the new installation.

pg_dumpall works by emitting commands to re-create roles, tablespaces, and empty databases, then invoking pg_dump for each database. This means that while each database will be internally consistent, the snapshots of different databases might not be exactly in-sync.

24.1.3. Handling large databases

Since PostgreSQL allows tables larger than the maximum file size on your system, it can be problematic to dump such a table to a file, since the resulting file will likely be larger than the maximum size allowed by your system. Since pg_dump can write to the standard output, you can use standard Unix tools to work around this possible problem. There are several ways to do it:

Use compressed dumps. You can use your favorite compression program, for example gzip:

```
pg_dump dbname | gzip > filename.gz
```

Reload with:

```
gunzip -c filename.gz | psql dbname
```

or:

```
cat filename.gz | gunzip | psql dbname
```

Use `split`. The `split` command allows you to split the output into pieces that are acceptable in size to the underlying file system. For example, to make chunks of 1 megabyte:

```
pg_dump dbname | split -b 1m - filename
```

Reload with:

```
cat filename* | psql dbname
```

Use pg_dump's custom dump format. If PostgreSQL was built on a system with the zlib compression library installed, the custom dump format will compress data as it writes it to the output file. This will produce dump file sizes similar to using `gzip`, but it has the added advantage that tables can be restored selectively. The following command dumps a database using the custom dump format:

```
pg_dump -Fc dbname > filename
```

A custom-format dump is not a script for psql, but instead must be restored with pg_restore, for example:

```
pg_restore -d dbname filename
```

See the pg_dump and pg_restore reference pages for details.

For very large databases, you might need to combine `split` with one of the other two approaches.

Privacy Policy | About PostgreSQL
Copyright © 1996-2012 The PostgreSQL Global Development Group