Tài liệu PostgreSQL 9.0.13

Nhóm phát triển toàn cầu PostgreSQL

III. Quản trị máy chủ

&

IV. Các giao diện máy trạm

Dịch sang tiếng Việt: Lê Trung Nghĩa, letrungnghia.foss@gmail.com

Dịch xong: 12/11/2014

Bản gốc tiếng Anh:

http://www.postgresql.org/files/documentation/pdf/9.0/postgresql-9.0-A4.pdf

---

PostgreSQL 9.0.13 Documentation

The PostgreSQL Global Development Group

III. Server Administration

&

IV. Client Interfaces

Lời người dịch

Tài liệu PostgreSQL 9.0.13 của nhóm phát triển toàn cầu PostgreSQL xuất bản năm 2013, như trong phần LỜI NÓI ĐẦU ở bên dưới giới thiệu, gồm 7 phần chính, đánh số từ I tới VII và phần dành cho các phụ lục, được đánh số VIII, tổng cộng dài 2.364 trang.

Tiếp sau bản dịch đầu tiên đã được hoàn thành vào tháng 03/2014 với 2 phần đầu:

- Phần I: Sách chỉ dẫn, là một giới thiệu không chính thức cho những người mới sử dụng.
- Phần II: Ngôn ngữ SQL, viết về môi trường ngôn ngữ truy vấn SQL, bao gồm cả các dạng và các hàm dữ liệu, cũng như việc tinh chỉnh hiệu năng ở mức của người sử dụng. Mọi người sử dụng PostgreSQL đều nên đọc phần này.

Bản dịch sang tiếng Việt của Phần I và Phần II hiện có thể tải về từ địa chỉ:

https://www.dropbox.com/s/t5sj1z55ifjxeu8/postgresql-9.0-A4-Part-I-II-Vi-13032014.pdf

Bản dịch lần này gồm 2 phần tiếp sau cũng đã được dịch xong trước và đưa ra để các độc giả sử dụng sớm, bao gồm:

- Phần III mô tả sự cài đặt và quản trị máy chủ. Từng người mà quản lý một máy chủ PostgreSQL, dù là để sử dụng riêng hay vì những lý do khác, nên đọc phần này.
- Phần IV mô tả các giao diện lập trình cho các chương trình máy trạm PostgreSQL.

Chính vì tài liệu không được dịch xong hoàn toàn tất cả các phần cùng một lúc, nên trong quá trình sử dụng, có một số nội dung tham chiếu ở các phần từ Phần I đến Phần IV tới các phần còn lại của tài liệu các độc giả chỉ có thể xem nội dung ở bản gốc tiếng Anh (như theo đường dẫn ở bìa trước của tài liệu) mà chưa có phần dịch sang tiếng Việt. Rất mong được các độc giả thông cảm.

Hy vọng các phần tiếp sau sẽ được dịch sang tiếng Việt sớm.

Việc dịch thuật chắc chắn không khỏi có những lỗi nhất định, rất mong các bạn độc giả, một khi phát hiện được, xin liên lạc với người dịch và đóng góp ý kiến về cách chỉnh sửa để tài liệu sẽ ngày một có chất lượng tốt hơn, phục vụ cho các bạn độc giả và mọi người có nhu cầu được tốt hơn. Xin chân thành cảm ơn trước các bạn độc giả về các đóng góp chỉnh sửa đó.

Mọi thông tin đóng góp chỉnh sửa cho bản dịch tiếng Việt, xin vui lòng gửi vào địa chỉ thư điện tử:

letrungnghia.foss@gmail.com

Chúc các độc giả thành công!

Hà Nội, ngày 12/11/2014

Lê Trung Nghĩa

Mục lục

Lời nói đầu

Cuốn sách này là tài liệu chính thức của PostgreSQL. Nó đã được các lập trình viên và những người tình nguyện khác của PostgreSQL viết song song với sự phát triển của phần mềm PostgreSQL. Nó mô tả tất cả các chức năng mà phiên bản hiện hành của PostgreSQL chính thức hỗ trợ.

Để làm cho số lượng lớn các thông tin về PostgreSQL có khả năng quản lý được, cuốn sách này đã được tổ chức trong vài phần. Mỗi phần có mục đích cho từng lớp người sử dụng khác nhau, hoặc cho các giai đoạn khác nhau của người sử dụng đối với kinh nghiệm về PostgreSQL của họ:

- Phần I là một giới thiệu không chính thức cho những người mới sử dụng.
- Phần II viết về môi trường ngôn ngữ truy vấn SQL, bao gồm cả các dạng và các hàm dữ liệu, cũng như việc tinh chỉnh hiệu năng ở mức của người sử dụng. Mọi người sử dụng PostgreSQL đều nên đọc phần này.
- Phần III mô tả sự cài đặt và quản trị máy chủ. Từng người mà quản lý một máy chủ PostgreSQL, dù là để sử dụng riêng hay vì những lý do khác, nên đọc phần này.
- Phần IV mô tả các giao diện lập trình cho các chương trình máy trạm PostgreSQL.
- Phần V bao gồm các thông tin cho những người sử dụng cao cấp về các khả năng mở rộng của máy chủ. Các chủ đề bao gồm các dạng và hàm dữ liệu do người sử dụng định nghĩa.
- Phần VI bao gồm các thông tin tham chiếu về các lệnh SQL, các chương trình máy trạm và máy chủ. Phần này hỗ trợ các phần khác với các thông tin được sắp xếp theo lệnh hoặc chương trình.
- Phần VII bao gồm các thông tin hỗn hợp có thể được sử dụng cho các lập trình viên của PostgreSQL.

III. Quản trị máy chủ

Phần này đề cập tới các chủ đề mà một người quản trị cơ sở dữ liệu PostgreSQL quan tâm. Điều này bao gồm sự cài đặt phần mềm, thiết lập và cấu hình máy chủ, quản lý những người sử dụng các cơ sở dữ liệu, và duy trì các tác vụ. Bất kỳ ai mà chạy một máy chủ PostgreSQL, thậm chí cho sử dụng cá nhân, mà đặc biệt trong sản xuất, sẽ quen với các chủ đề được đề cập tới trong phần này.

Thông tin trong phần này được sắp xếp gần theo trật tự mà ở đó một người sử dụng mới sẽ đọc được nó. Nhưng các chương sẽ là khép kín và có thể được đọc một cách riêng rẽ nếu muốn. Thông tin trong phần này được trình bày theo cách ngắn gọn súc tích theo các đơn vị chủ đề. Các độc giả tìm kiếm một mô tả hoàn chỉnh một lệnh đặc biệt sẽ xem Phần VI.

Một ít chương đầu được viết sao cho chúng có thể được hiểu mà không cần có tri thức tiên quyết nào, nên những người sử dụng mới cần thiết lập máy chủ cho riêng mình có thể bắt đầu sự khai thác của họ với phần này. Các nội dung còn lại của phần này là về việc tinh chỉnh và quản lý; tài liệu giả thiết rằng độc giả đã quen với sử dụng chung hệ thống cơ sở dữ liệu PostgreSQL. Các độc giả được khuyến khích xem Phần I và Phần II để có các thông tin bổ sung.

Chương 15. Cài đặt từ mã nguồn

Chương này mô tả sự cài đặt PostgreSQL bằng việc sử dụng phân phối mã nguồn. (Nếu bạn đang cài đặt một phân phối được đóng gói trước rồi, như một gói RPM hoặc Debian, thì hãy bỏ qua chương này và thay vào đó hãy đọc các chỉ dẫn của gói đó).

15.1. Phiên bản ngắn gọn

```
./configure
gmake
su
gmake install
adduser postgres
mkdir /usr/local/pgsql/data
chown postgres /usr/local/pgsql/data
su - postgres
/usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
/usr/local/pgsql/bin/postgres -D /usr/local/pgsql/data >logfile 2>&1 &
/usr/local/pgsql/bin/createdb test
/usr/local/pgsql/bin/psql test
```

Phiên bản dài là phần còn lại của chương này.

15.2. Yêu cầu

Nói chung, một nền tảng tương thích với Unix hiện đại nên có khả năng chạy PostgreSQL. Các nền tảng mà đã nhận được việc kiểm thử đặc biệt vào thời điểm phát hành được liệt kê trong Phần 15.7 bên dưới. Trong thư mục con doc của phân phối sẽ có vài tài liệu Hỏi - Đáp thường gặp (FAQ) đặc thù nền tảng mà bạn có thể muốn tư vấn nếu bạn đang có trục trặc.

Các gói phần mềm sau được yêu cầu cho việc xây dựng PostgreSQL:

- GNU make được yêu cầu; các chương trình make khác sẽ không làm việc. GNU make thường được cài đặt với cái tên là gmake: tài liệu này sẽ luôn tham chiếu tới nó theo tên đó. (Trong một số hệ thống thì gmake là công cụ mặc định với tên make). Để kiểm tra gmake hãy gõ vào:

  ```
  gmake --version
  ```

  Được khuyến cáo hãy sử dụng phiên bản 3.79.1 hoặc sau này.
- Bạn cần một trình biên dịch ISO/ANSI C (ít nhất tuân thủ C89). Các phiên bản GCC gần đây được khuyến cáo, nhưng PostgreSQL được biết xây dựng bằng việc sử dụng một dải đa dạng rộng lớn các trình biên dịch từ các nhà bán hàng khác nhau.
- tar được yêu cầu để giải nén phân phối nguồn, hoặc có thể là cả gzip hoặc bzip2.
- Thư viện GNU Readline được sử dụng mặc định. Nó cho phép psql (trình biên dịch dòng lệnh SQL của PostgreSQL) nhớ từng lệnh bạn gõ vào, và cho phép bạn sử dụng các phím mũi tên để gọi lại và sửa các lệnh trước đó. Điều này rất hữu dụng và được khuyến cáo mạnh mẽ. Nếu bạn không muốn sử dụng nó thì bạn phải chỉ định lựa chọn --without-readline (không đọc dòng) để thiết lập cấu hình. Như một lựa chọn thay thế, bạn có thể thường sử dụng thư viện được cấp phép BSD là libedit, ban đầu được phát triển trên NetBSD. Thư viện

libedit là tương thích với GNU Readline và được sử dụng nếu libreadline không được thấy, hoặc nếu --with-libedit-preferred (với libedit được ưu tiên) được sử dụng như một lựa chọn cho configure (thiết lập cấu hình). Nếu bạn đang sử dụng một phát tán Linux được đóng gói rồi, hãy nhớ rằng bạn cần cả các gói readline và readline-devel, nếu chúng là tách biệt nhau trong phát tán của bạn.

- Thư viện nén zlib được sử dụng mặc định. Nếu bạn không muốn sử dụng nó thì bạn phải chỉ định lựa chọn --without-zlib (không có zlib) cho configure. Việc sử dụng lựa chọn này vô hiệu hóa sự hỗ trợ cho các kho lưu trữ được nén trong pg_dump và pg_restore.

Các gói sau là tùy chọn. Chúng không được yêu cầu trong cấu hình mặc định, nhưng chúng là cần thiết khi các lựa chọn nhất định được xây dựng được hỗ trợ, như được giải thích bên dưới:

- Để xây dựng ngôn ngữ lập trình máy chủ PL/Perl thì bạn cần một cài đặt đầy đủ Perl, bao gồm thư viện libperl và các tệp đầu đề. Vì PL/Perl sẽ là một thư viện được chia sẻ, nên thư viện libperl cũng phải là một thư viện được chia sẻ trong hầu hết các nền tảng. Điều này dường như sẽ là mặc định trong các phiên bản Perl gần đây, nhưng nó không phải thế trong các phiên bản trước đó, và trong bất kỳ trường hợp nào thì đó là sự lựa chọn của bất kỳ ai đã cài đặt Perl ở chỗ của bạn. Nếu bạn định thực hiện nhiều hơn so với sử dụng ngẫu nhiên PL/Perl, thì bạn nên đảm bảo rằng cài đặt Perl đã được xây dựng với lựa chọn usemultiplicity được bật (perl -V sẽ chỉ ra liệu điều này có là quan trọng hay không). Nếu bạn không có thư viện được chia sẻ nhưng bạn cần một thư viện, thì một thông điệp như thế này sẽ xuất hiện trong quá trình PostgreSQL được xây dựng để chỉ ra yếu tố này:

  ```
  *** Cannot build PL/Perl because libperl is not a shared library.
  *** You might have to rebuild your Perl installation. Refer to
  *** the documentation for details.
  ```

  (Nếu bạn không theo kết quả đầu ra trên màn hình thì bạn sẽ phải lưu ý rằng đối tượng thư viện PL/Perl, plperl.so hoặc tương tự, sẽ không được cài đặt). Nếu bạn thấy điều này, bạn sẽ phải xây dựng lại và cài đặt Perl bằng tay để có khả năng xây dựng PL/Perl. Trong quá trình cấu hình cho Perl, hãy yêu cầu một thư viện được chia sẻ.

- Để xây dựng ngôn ngữ lập trình máy chủ PL/Python, bạn cần một cài đặt Python với các tệp đầu đề và module distutils. Phiên bản được yêu cầu tối thiểu là Python 2.2. Python 3 được hỗ trợ nếu nó là phiên bản 3.1 hoặc sau này; mà hãy xem Phần 42.1 khi sử dụng Python 3.

  Vì PL/Python sẽ là một thư viện được chia sẻ, thư viện libpython cũng phải là một thư viện được chia sẻ trên hầu hết các nền tảng. Điều này không quan trọng trong một cài đặt Python mặc định. Nếu sau việc xây dựng và cài đặt PostgreSQL mà bạn có một tệp gọi là plpython.so (có khả năng với một mở rộng tệp khác), thì mọi điều sẽ là tốt. Nếu không, bạn sẽ thấy một lưu ý với nội dung sau:

  ```
  *** Cannot build PL/Python because libpython is not a shared library.
  *** You might have to rebuild your Python installation. Refer to
  *** the documentation for details.
  ```

  Điều đó có nghĩa là bạn phải xây dựng lại (một phần của) cài đặt Python của bạn để tạo ra

thư viện được chia sẻ này.

Nếu bạn có các vấn đề đó, hãy chạy Python 2.3 hoặc cấu hình sau này bằng việc sử dụng cờ --enable-shared. Trong một số hệ điều hành bạn không phải xây dựng một thư viện được chia sẻ, nhưng bạn sẽ phải thuyết phục hệ thống PostgreSQL đã xây dựng điều này. Hãy tư vấn Makefile trong thư mục src/pl/plpython để có các chi tiết.

- Để xây dựng ngôn ngữ thủ tục PL/Tcl, bạn tất nhiên cần một cài đặt Tcl. Nếu bạn đang sử dụng một phiên bản Tcl trước 8.4, hãy chắc chắn rằng nó từng được xây dựng mà không có sự hỗ trợ đa luồng.
- Để xúc tác cho sự Hỗ trợ Ngôn ngữ Bẩm sinh - NLS (Native Language Support), đó là, khả năng hiển thị các thông điệp của một chương trình trong một ngôn ngữ không phải tiếng Anh, bạn cần một triển khai Gettext API. Một vài hệ điều hành có điều này được xây dựng sẵn (như, Linux, NetBSD, Solaris), đối với các hệ điều hành khác bạn có thể tải về một gói bổ sung từ http://www.gnu.org/software/gettext/. Nếu bạn đang sử dụng một triển khai Gettext trong thư viện GNU C thì bạn sẽ cần thêm gói GNU Gettext cho một số chương trình tiện ích. Đối với bất kỳ trong số các triển khai khác nào, bạn sẽ không cần nó.
- Bạn cần Kerberos, OpenSSL, OpenLDAP, và/hoặc PAM, nếu bạn muốn hỗ trợ xác thực hoặc mã hóa bằng việc sử dụng các dịch vụ đó.

Nếu bạn đang xây dựng từ một cây Git thay vì việc sử dụng một gói nguồn mở được phát hành, hoặc nếu bạn muốn tiến hành phát triển máy chủ, bạn cũng cần các gói sau:

- GNU Flex và Bison sẽ là cần thiết để xây dựng từ một kiểm tra của Git, hoặc nếu bạn đã thay đổi các tệp định nghĩa trình phân tích cú pháp và trình quét thực sự. Nếu bạn cần chúng, hãy chắc chắn có Flex 2.5.31 hoặc sau này và Bison 1.875 hoặc sau này. Các chương trình lex và yacc khác có thể không được sử dụng.
- Perl 5.8 hoặc sau này là cần thiết để xây dựng từ một kiểm tra của Git, hoặc nếu bạn đã thay đổi các tệp đầu vào đối với bất kỳ bước xây dựng nào mà sử dụng các Perl script. Nếu xây dựng trong Windows thì bạn sẽ cần Perl trong bất kỳ trường hợp nào.

Nếu bạn cần có một gói GNU, bạn có thể thấy nó trong site nhân bản gương GNU cục bộ của bạn (xem http://www.gnu.org/order/ftp.html để có danh sách) hoặc ở http://ftp.gnu.org/gnu/.

Cũng hãy kiểm tra xem bạn có đủ không gian đĩa không. Bạn sẽ cần khoảng 100 MB cho cây nguồn trong quá trình biên dịch và khoảng 20 MB cho thư mục cài đặt. Một cụm cơ sở dữ liệu rỗng lấy mất khoảng 35 MB; các cơ sở dữ liệu lấy khoảng 5 lần lượng không gian mà một tệp văn bản thô với các dữ liệu y hệt có thể chiếm. Nếu bạn dự định chạy các kiểm tra sự hồi quy thì bạn tạm thời sẽ cần thêm tới 150 MB. Hãy sử dụng lệnh df để kiểm tra không gian trống của đĩa.

15.3. Lấy nguồn

Các nguồn của PostgreSQL 9.0.13 có thể có được bằng cách FTP một cách nặc danh từ ftp://ftp.postgresql.org/pub/source/v9.0.13/postgresql-9.0.13.tar.gz. Các lựa chọn tải về khác có thể thấy trên website của chúng tôi: http://www.postgresql.org/download/. Sau khi bạn đã có được tệp

đó, hãy giải nén nó:

```
gunzip postgresql-9.0.13.tar.gz
tar xf postgresql-9.0.13.tar
```

Điều này sẽ tạo ra một thư mục postgresql-9.0.13 dưới thư mục hiện hành với các nguồn của PostgreSQL. Hãy thay đổi về thư mục đó cho phần còn lại của thủ tục cài đặt.

Bạn cũng có thể có được nguồn đó trực tiếp từ kho kiểm soát phiên bản, xem Phụ lục H.

15.4. Nâng cấp

Các chỉ dẫn đó giả thiết là cài đặt đang tồn tại của bạn là nằm dưới thư mục /usr/local/pgsql, và rằng khu vực dữ liệu là trong /usr/local/pgsql/data. Hãy thay thế các đường dẫn của bạn cho phù hợp.

Định dạng lưu trữ dữ liệu nội bộ đó điển hình thay đổi theo mỗi phiên bản chính của PostgreSQL. Vì thế, nếu bạn đang nâng cấp một cài đặt đang tồn tại mà không có số phiên bản dạng "9.0.x", thì bạn phải sao lưu và phục hồi các dữ liệu của bạn. Nếu bạn đang nâng cấp từ PostgreSQL "9.0.x", thì phiên bản mới có thể sử dụng các tập dữ liệu hiện hành sao cho bạn sẽ bỏ qua các bước sao lưu và phục hồi bên dưới vì chúng là không cần thiết.

1. Nếu thực hiện một sao lưu, hãy chắc chắn là cơ sở dữ liệu của bạn không đang được cập nhật. Điều này không ảnh hưởng tới tính toàn vẹn của sao lưu, nhưng các dữ liệu được thay đổi có lẽ tất nhiên không được đưa vào. Nếu cần thiết, hãy sửa các quyền trong tệp /usr/local/pgsql/data/pg_hba.conf (hoặc tương đương) để không cho phép bất kỳ ai truy cập ngoài bạn ra.

   Để sao lưu cài đặt cơ sở dữ liệu của bạn, hãy gõ:

   ```
   pg_dumpall > outputfile
   ```

   Nếu bạn cần giữ lại các OID (khi đang sử dụng chúng như các khóa ngoại), sau đó hãy sử dụng lựa chọn -o khi chạy pg_dumpall.

   Để thực hiện sao lưu, bạn có thể sử dụng lệnh pg_dumpall từ phiên bản bạn hiện đang chạy. Tuy nhiên, để có được các kết quả tốt nhất, hãy thử sử dụng lệnh pg_dumpall từ PostgreSQL 9.0.13, vì phiên bản này chứa các sửa lỗi và các cải tiến so với các phiên bản cũ hơn. Trong khi khuyến cáo này có thể xem như là ngớ ngẩn vì bạn còn chưa cài đặt phiên bản mới, thì được khuyến cáo để cho phép nó nếu bạn có kế hoạch cài đặt một cách bình thường và truyền các dữ liệu đó sau này. Điều này cũng sẽ làm giảm thời gian chết.

2. Tắt máy chủ cũ:

   ```
   pg_ctl stop
   ```

   Trong các máy có PostgreSQL được bật lúc khởi động, có thể có một tệp khởi động sẽ hoàn tất thứ y hệt. Ví dụ, trong một máy Red Hat Linux người ta có thể thấy điều này sẽ làm việc:

   ```
   /etc/rc.d/init.d/postgresql stop
   ```

3. Nếu phục hồi từ sao lưu, hãy đổi tên hoặc xóa thư mục cài đặt cũ. Là ý tưởng tốt để đổi tên thư mục, thay vì xóa nó, trong trường hợp bạn có vấn đề và cần quay ngược về nó. Hãy nhớ trong đầu rằng thư mục đó có thể ngốn không gian đĩa đáng kể. Để đổi tên thư mục đó, hãy sử dụng một lệnh giống như thế này:

```
mv /usr/local/pgsql /usr/local/pgsql.old
```

4. Cài đặt phiên bản mới của PostgreSQL như được nêu trong Phần 15.5.
5. Tạo một cụm cơ sở dữ liệu mới nếu cần. Hãy nhớ rằng bạn phải thực thi các lệnh đó trong khi đã đăng nhập vào tài khoản người sử dụng cơ sở dữ liệu đặc biệt (mà bạn có rồi nếu bạn đang nâng cấp).
   ```
   /usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
   ```
6. Hãy phục hồi pg_hba.conf trước đó của bạn và bất kỳ sửa đổi nào của postgresql.conf.
7. Hãy khởi động máy chủ cơ sở dữ liệu, một lần nữa bằng việc sử dụng tài khoản người sử dụng cơ sở dữ liệu đặc biệt:
   ```
   /usr/local/pgsql/bin/postgres -D /usr/local/pgsql/data
   ```
8. Cuối cùng, hãy phục hồi các dữ liệu của bạn từ sao lưu với:
   ```
   /usr/local/pgsql/bin/psql -d postgres -f outputfile
   ```
   bằng việc sử dụng psql mới.

Thảo luận tiếp sẽ xuất hiện trong Phần 24.4, bao gồm cả các chỉ dẫn về cách mà cài đặt trước đó có thể tiếp tục chạy trong khi cài đặt mới được tiến hành.

15.5. Thủ tục cài đặt

1. Cấu hình

   Bước đầu tiên của thủ tục cài đặt là phải thiết lập cấu hình cho cây nguồn đối với máy của bạn và chọn các lựa chọn mà bạn muốn. Điều này được thực hiện bằng việc chạy script configure. Đối với một cài đặt mặc định thì đơn giản hãy gõ vào:
   ```
   ./configure
   ```
   Script này sẽ chạy một số các kiểm tra để xác định các giá trị cho hàng loạt các biến phụ thuộc hệ thống và dò tìm ra bất kỳ sự không minh bạch nào của hệ điều hành của bạn, và cuối cùng sẽ tạo ra vài tệp trong cây được xây dựng để ghi lại những gì nó thấy. Bạn cũng có thể chạy configure trong một thư mục bên ngoài cây nguồn, nếu bạn muốn giữ cho thư mục được xây dựng riêng rẽ. Thủ tục này còn được gọi là một xây dựng VPATH.

   Đây là cách làm:
   ```
   mkdir build_dir
   cd build_dir
   /path/to/source/tree/configure [options go here]
   gmake
   ```
   Cấu hình mặc định sẽ xây dựng máy chủ và các tiện ích, cũng như tất cả các ứng dụng và giao diện máy trạm đòi hỏi chỉ một trình biên dịch C. Tất cả các tệp sẽ được cài đặt theo mặc định trong /usr/local/pgsql.

   Bạn có thể tùy biến việc xây dựng và qui trình cài đặt đó bằng việc cung cấp một hoặc nhiều lựa chọn dòng lệnh sau đối với configure:

   --prefix=PREFIX

   > Hãy cài đặt tất cả các tệp trong thư mục PREFIX thay vì /usr/local/pgsql. Các tệp thực sự sẽ được cài đặt trong các thư mục con khác nhau; không tệp nào sẽ được cài đặt

trực tiếp vào thư mục PREFIX.

Nếu bạn có các nhu cầu đặc biệt, bạn cũng có thể tùy biến các thư mục con riêng rẽ với các lựa chọn sau. Tuy nhiên, nếu bạn để chúng với các mặc định của chúng, thì cài đặt đó sẽ có khả năng tái định vị, nghĩa là bạn có thể chuyển thư mục đó sau khi cài đặt. (Các vị trí man và doc sẽ không bị ảnh hưởng vì điều này).

Đối với các cài đặt có khả năng tái định vị, bạn có thể muốn sử dụng lựa chọn cấu hình --disable-rpath. Hơn nữa, bạn sẽ cần nói cho hệ điều hành cách để tìm các thư viện được chia sẻ.

--exec-prefix=EXEC-PREFIX

Bạn có thể cài đặt các tệp phụ thuộc kiến trúc dưới một tiền tố khác, EXEC-PREFIX, so với tiền tố PREFIX đã được thiết lập. Điều này có thể là hữu dụng để chia sẻ các tệp phụ thuộc kiến trúc giữa các máy chủ host. Nếu bạn bỏ qua điều này, thì sau đó EXEC-PREFIX được thiết lập ngang với PREFIX và cả 2 đều là phụ thuộc kiến trúc và các tệp độc lập sẽ được cài đặt trong cây y hệt, có thể là những gì bạn muốn.

--bindir=DIRECTORY

Chỉ định thư mục cho các chương trình có khả năng thực thi. Mặc định là EXEC-PREFIX/bin, thường có nghĩa là /usr/local/pgsql/bin.

--sysconfdir=DIRECTORY

Thiết lập thư mục đó cho các tệp cấu hình khác nhau, PREFIX/etc là mặc định.

--libdir=DIRECTORY

Thiết lập vị trí để cài đặt các thư viện và các module có khả năng tải động được. Mặc định là EXEC-PREFIX/lib.

--includedir=DIRECTORY

Thiết lập thư mục để cài đặt các tệp đầu đề C và C++. Mặc định là PREFIX/include.

--datarootdir=DIRECTORY

Thiết lập thư mục gốc root cho các dạng khác nhau các tệp dữ liệu chỉ đọc. Điều này chỉ thiết lập mặc định cho một số lựa chọn sau. Mặc định là PREFIX/share.

--datadir=DIRECTORY

Thiết lập thư mục cho các tệp dữ liệu chỉ đọc được các chương trình cài đặt. Mặc định là DATAROOTDIR. Lưu ý rằng điều này không có gì phải làm ở những nơi các tệp cơ sở dữ liệu của bạn sẽ được đặt.

--localedir=DIRECTORY

Thiết lập thư mục cho việc cài đặt các dữ liệu định vị, đặc biệt các tệp catalog giao dịch thông điệp. Mặc định là DATAROOTDIR/locale.

--mandir=DIRECTORY

Các trang man đi với PostgreSQL sẽ được cài đặt trong thư mục này, trong các thư mục con manx tương ứng của chúng. Mặc định là DATAROOTDIR/man.

--docdir=DIRECTORY

Thiết lập thư mục gốc root cho việc cài đặt các tệp tài liệu, ngoại trừ các trang "man". Điều này chỉ thiết lập mặc định cho các lựa chọn sau. Giá trị mặc định cho lựa chọn này là DATAROOTDIR/doc/postgresql.

--htmldir=DIRECTORY

Tài liệu có định dạng HTML cho PostgreSQL sẽ được cài đặt trong thư mục này. Mặc định là DATAROOTDIR.

Lưu ý: Sự chăm sóc đã được thực hiện để làm cho có khả năng cài đặt PostgreSQL ở các vị trí cài đặt được chia sẻ (như /usr/local/include) mà không có việc can thiệp với không gian tên của phần còn lại của hệ thống. Trước tiên, chuỗi "/postgresql" tự động được thêm vào datadir, sysconfdir, và docdir, trừ phi tên thư mục được mở rộng đầy đủ rồi có chứa chuỗi "postgres" hoặc "pgsql". Ví dụ, nếu bạn chọn /usr/local như là tiền tố, thì tài liệu sẽ được cài đặt trong /usr/local/doc/postgresql, nhưng nếu tiền tố là /opt/postgres, thì nó sẽ ở trong /opt/postgres/doc. Các tệp đầu đề C công cộng của các giao diện máy trạm sẽ được cài đặt trong includedir và là không gian tên sạch. Các tệp đầu đề nội bộ và các tệp đầu đề máy chủ sẽ được cài đặt trong các thư mục riêng dưới includedir. Xem tài liệu của từng giao diện cho thông tin về cách truy cập các tệp đầu đề của nó. Cuối cùng, một thư mục con riêng cũng sẽ được tạo ra, nếu phù hợp, dưới libdir đối với các module có khả năng tải động được.

--with-includes=DIRECTORIES

DIRECTORIES (các thư mục) là một danh sách các thư mục phân cách nhau bằng dấu hai chấm (:) sẽ được thêm vào danh sách mà trình biên dịch tìm kiếm các tệp đầu đề. Nếu bạn có các gói tùy chọn (như GNU Readline) được cài đặt ở một vị trí phi tiêu chuẩn, thì bạn phải sử dụng lựa chọn này và có thể cũng tương ứng với lựa chọn –with-libraries.

Ví dụ: --with-includes=/opt/gnu/include:/usr/sup/include.

--with-libraries=DIRECTORIES

DIRECTORIES (các thư mục) là một danh sách các thư mục để tìm kiếm các thư viện. bạn có thể sẽ phải sử dụng lựa chọn này (và tương ứng với lựa chọn --with-includes) nếu bạn có các gói được cài đặt ở các vị trí phi tiêu chuẩn.

Ví dụ: --with-libraries=/opt/gnu/lib:/usr/sup/lib.

--enable-nls[=LANGUAGES]

Kích hoạt Hỗ trợ Ngôn ngữ Bẩm sinh - NLS (Native Language Support), đó là, khả năng để hiển thị các thông điệp của một chương trình theo một ngôn ngữ khác với tiếng Anh. LANGUAGES (các ngôn ngữ) là một danh sách các lựa chọn được cách nhau bằng dấu trắng các mã ngôn ngữ mà bạn muốn được hỗ trợ, ví dụ --enable-nls='de fr'. (Sự giao nhau giữa danh sách của bạn và tập hợp các bản dịch thực sự được cung cấp sẽ được tính toán tự động). Nếu bạn không chỉ định một danh sách, thì tất cả các bản dịch sẵn sàng sẽ được cài đặt.

Để sử dụng lựa chọn này, bạn sẽ cần một triển khai Gettext API; xem ở trên.

--with-pgport=NUMBER

Thiết lập NUMBER như là số cổng mặc định cho máy chủ và các máy trạm. Mặc định là 5432. Cổng đó có thể luôn được thay đổi sau đó, nhưng nếu bạn chỉ định nó ở đây thì sau đó cả máy chủ và các máy trạm sẽ có cùng mặc định được biên dịch, nó có thể rất là thuận tiện. Thường lý do tốt duy nhất để chọn một giá trị không mặc định là nếu bạn định chạy nhiều máy chủ PostgreSQL trên cùng một máy.

--with-perl

Xây dựng ngôn ngữ PL/Perl phía máy chủ.

--with-python

Xây dựng ngôn ngữ Python phía máy chủ.

--with-tcl

Xây dựng ngôn ngữ PL/Tlc phía máy chủ.

--with-tclconfig=DIRECTORY

Tcl cài đặt tệp tclConfig.sh, nó chứa thông tin cấu hình cần thiết để xây dựng các module tương tác với Tcl. Tệp này thường được thấy tự động ở một vị trí được biết rõ, nhưng nếu bạn muốn sử dụng một phiên bản khác của Tcl thì bạn có thể chỉ định thư mục trong đó để tìm kiếm nó.

--with-gssapi

Được xây dựng với sự hỗ trợ cho xác thực GSSAPI. Trên nhiều hệ thống, hệ thống GSSAPI (thường một phần của cài đặt Kerberos) không được cài đặt ở một vị trí được mặc định tìm kiếm (như /usr/include, /usr/lib), nên bạn phải sử dụng các lựa chọn --with-includes và --with-libraries bổ sung thêm cho lựa chọn này. configure sẽ kiểm tra đối với các thư viện và tệp đầu đề được yêu cầu để chắc chắn rằng cài đặt Kerberos của bạn là đủ trước khi xử lý.

--with-krb5

Được xây dựng với sự hỗ trợ cho xác thực Kerberos 5. Trong nhiều hệ thống, hệ thống Kerberos không được cài đặt ở một vị trí được mặc định tìm thấy (như, /usr/include, /usr/lib), nên bạn phải sử dụng các lựa chọn --with-includes và --with-libraries bổ sung cho lựa chọn này. configure sẽ kiểm tra các tệp đầu đề và các thư viện được yêu cầu để chắc chắn rằng cài đặt Kerberos của bạn là đủ trước khi xử lý.

--with-krb-srvnam=NAME

Tên mặc định của dịch vụ Kerberos chính (còn được GSSAPI sử dụng). postgres là mặc định. Thường không có lý do để thay đổi điều này trừ phi bạn có một môi trường Windows, trong trường hợp đó nó phải được để thành chữ hoa POSTGRES.

--with-openssl

Được xây dựng với sự hỗ trợ cho các kết nối SSL (được mã hóa). Điều này đòi hỏi gói OpenSSL sẽ được cài đặt. configure sẽ kiểm tra đối với các thư viện và tệp đầu đề được yêu cầu để chắc chắn rằng cài đặt OpenSSL của bạn là đủ trước khi xử lý.

--with-pam

Được xây dựng với sự hỗ trợ của các module Xác thực được Cài cắm - PAM (Pluggable Authentication Modules).

--with-ldap

Được xây dựng với sự hỗ trợ của LDAP cho sự tra cứu tham số kết nối và xác thực (xem Phần 31.16 và Phần 19.3.7 để có thêm thông tin). Trên Unix, điều này đòi hỏi gói OpenLDAP phải được cài đặt. Trên Windows, thư viện mặc định WinLDAP được sử dụng. configure sẽ kiểm tra đối với các thư viện và tệp đầu đề được yêu cầu để chắc chắn rằng cài đặt OpenLDAP của bạn là đủ trước khi xử lý.

--without-readline

Ngăn chặn sử dụng thư viện Readline (và cả libedit). Lựa chọn này vô hiệu hóa việc soạn sửa dòng lệnh và lịch sử trong psql, nên nó không được khuyến cáo.

--with-libedit-preferred

Hỗ trợ sử dụng thư viện libedit có giấy phép BSD hơn là Readline có giấy phép GPL. Lựa chọn này chỉ quan trọng nếu bạn có cả 2 thư viện được cài đặt; mặc định trong trường hợp đó là để sử dụng Readline.

--with-bonjour

Được xây dựng với sự hỗ trợ của Bonjour. Điều này đòi hỏi sự hỗ trợ của Bonjour trong hệ điều hành của bạn. Được khuyến cáo trên Mac OS X.

--with-ossp-uuid

Sử dụng thư viện[1] OSSP UUID khi xây dựng contrib/uuid-ossp. Thư viện đó cung cấp các hàm để tạo ra các UUID.

--with-libxml

Được xây dựng với libxml (xúc tác cho sự hỗ trợ của SQL/XML). Libxml phiên bản 2.6.23 hoặc sau này được yêu cầu cho tính năng này.

Libxml cài đặt một chương trình xml2-config có thể được sử dụng để dò tìm ra các lựa chọn trình biên dịch và trình liên kết được yêu cầu. PostgreSQL sẽ sử dụng nó tự động nếu thấy. Để chỉ định một cài đặt libxml ở một vị trí không thông thường, bạn có thể hoặc thiết lập biến môi trường XML2_CONFIG để chỉ tới chương trình xml2-config thuộc về cài đặt đó, hoặc sử dụng các lựa chọn --with-includes và --with-libraries.

--with-libxslt

Sử dụng libxslt khi xây dựng contrib/xml2. contrib/xml2 dựa vào thư viện này để thực hiện các biến đổi XSL của XML.

--disable-integer-datetimes

Vô hiệu hóa hỗ trợ cho lưu trữ số nguyên 64 bit đối với các dấu thời gian và các khoảng thời gian, và lưu trữ các giá trị ngày tháng thời gian (datetime) như là các số dấu chấm động thay vào đó. Lưu trữ ngày tháng thời gian theo dấu chấm động từng là mặc định trong các phiên bản PostgreSQL trước 8.4, nhưng nó bây giờ bị phản đối, vì nó không hỗ trợ độ chính xác micro giây cho dải đầy đủ các giá trị timestamp.

1 http://www.ossp.org/pkg/lib/uuid/

Tuy nhiên, lưu trữ ngày tháng thời gian dựa vào số nguyên đòi hỏi một dạng số nguyên 64 bit. Vì thế, lựa chọn này có thể được sử dụng khi không dạng nào như vậy là sẵn sàng, hoặc vì tính tương thích với các ứng dụng được viết cho các phiên bản trước của PostgreSQL. Xem Phần 8.5 để có thêm thông tin.

--disable-float4-byval

Vô hiệu hóa việc truyền các giá trị float4 "theo giá trị", làm cho chúng sẽ được truyền qua "theo tham chiếu" thay vào đó. Lựa chọn này gây tốn hiệu năng, nhưng có thể cần thiết cho tính tương thích với các hàm cũ do người sử dụng định nghĩa mà được viết trong C và sử dụng qui ước lời gọi "phiên bản 0". Giải pháp dài hạn tốt hơn là để cập nhật bất kỳ hàm nào như vậy để sử dụng qui ước lời gọi "phiên bản 1".

--disable-float8-byval

Vô hiệu hóa việc truyền các giá trị float8 "theo giá trị", làm cho chúng sẽ được truyền "theo tham chiếu" thay vào đó. Lựa chọn này gây tốn hiệu năng, nhưng có thể là cần thiết cho tính tương thích với các hàm cũ do người sử dụng định nghĩa mà được viết trong C và sử dụng qui ước lời gọi "phiên bản 0". Giải pháp dài hạn tốt hơn là cập nhật bất kỳ hàm nào như vậy để sử dụng qui ước lời gọi "phiên bản 1". Lưu ý rằng lựa chọn này ảnh hưởng không chỉ float8, mà còn int8 và một số dạng có liên quan như dấu thời gian timestamp. Trên các nền tảng 32 bit, --disable-float8-byval là mặc định và không được phép để lựa chọn –enable-float8-byval.

--with-segsize=SEGSIZE

Thiết lập kích cỡ phân đoạn, theo gigabyte (GB). Các bảng lớn được chia thành nhiều tệp hệ thống - điều hành, mỗi tệp kích cỡ bằng nhau đối với kích thước đoạn đó. Điều này tránh được các vấn đề với các giới hạn kích thước tệp tồn tại trong nhiều nền tảng. Kích thước đoạn mặc định, 1 GB, là an toàn trong tất cả các nền tảng được hỗ trợ. Nếu hệ điều hành của bạn có sự hỗ trợ "các tệp lớn" (hầu hết có hỗ trợ ngày nay), thì bạn có thể sử dụng một kích thước đoạn lớn hơn. Điều này có thể hữu dụng để làm giảm số lượng các trình mô tả tệp được sử dụng khi làm việc với các bảng rất lớn. Nhưng hãy cẩn thận không lựa chọn một giá trị lớn hơn so với giá trị được nền tảng của bạn hỗ trợ và các hệ thống tệp của bạn dự định sử dụng. Các công cụ khác bạn có thể muốn sử dụng, như tar, cũng có thể thiết lập các hạn chế về kích thước tệp khả dụng. Được khuyến cáo, dù không được yêu cầu tuyệt đối, rằng giá trị này phải là lũy thừa của 2. Lưu ý rằng việc thay đổi giá trị này đòi hỏi một initdb.

--with-blocksize=BLOCKSIZE

Thiết lập kích thước khối, theo kilobytes (KB). Đây là đơn vị lưu trữ và I/O (đầu vào/đầu ra) trong các bảng. Mặc định, 8 KB, là phù hợp cho hầu hết các tình huống; nhưng các giá trị khác có thể là hữu dụng trong những trường hợp đặc biệt. Giá trị đó phải là lũy thừa của 2 giữa 1 và 32 (KB). Lưu ý rằng việc thay đổi giá trị này đòi hỏi một initdb.

--with-wal-segsize=SEGSIZE

Thiết lập kích thước đoạn WAL, theo megabyte (MB). Đây là kích thước của từng tệp riêng rẽ trong lưu ký WAL. Có thể là hữu dụng để tinh chỉnh kích thước này để kiểm soát mức độ chi tiết của việc xuất ra các lưu ký WAL. Kích thước mặc định là 16 MB. Giá trị đó phải là lũy thừa của 2 giữa 1 và 64 (MB). Lưu ý rằng việc thay đổi giá trị này đòi hỏi một initdb.

--with-wal-blocksize=BLOCKSIZE

Thiết lập kích thước khối WAL, theo kilobytes (KB). Đây là đơn vị lưu trữ và I/O trong lưu ký WAL. Mặc định, 8 KB, là phù hợp cho hầu hết các tình huống; nhưng các giá trị khác có thể là hữu dụng trong các trường hợp đặc biệt. Giá trị đó phải là lũy thừa của 2 giữa 1 và 64 (KB). Lưu ý, việc thay đổi giá trị này đòi hỏi một initdb.

--disable-spinlocks

Cho phép xây dựng thành công thậm chí nếu PostgreSQL không có hỗ trợ khóa quay (spinlock) CPU cho nền tảng đó. Thiếu sự hỗ trợ khóa quay sẽ gây ra hiệu năng nghèo nàn; vì thế, lựa chọn này chỉ nên được sử dụng nếu xây dựng đó bỏ dở và thông báo cho bạn rằng nền tảng đó thiếu hỗ trợ khóa quay. Nếu lựa chọn này được yêu cầu để xây dựng PostgreSQL trong nền tảng của bạn, xin hãy báo cáo vấn đề đó cho các lập trình viên phát triển PostgreSQL.

--disable-thread-safety

Vô hiệu hóa dòng an toàn (thread-safety) các thư viện máy trạm. Điều này ngăn chặn các dòng đồng thời trong libpq và các chương trình ECPG khỏi việc kiểm soát an toàn các xử lý kết nối riêng của chúng.

--with-system-tzdata=DIRECTORY

PostgreSQL bao gồm cả cơ sở dữ liệu vùng thời gian của riêng nó, đòi hỏi các hoạt động ngày tháng và thời gian. Cơ sở dữ liệu vùng thời gian này trên thực tế là tương thích với cơ sở dữ liệu vùng thời gian "zoneinfo" được nhiều hệ điều hành cung cấp như FreeBSD, Linux và Solaris, nên có thể là thừa đề cài đặt nó một lần nữa. Khi lựa chọn này được sử dụng, cơ sở dữ liệu vùng thời gian do hệ thống cung cấp trong DIRECTORY được sử dụng thay vì một cơ sở dữ liệu được đưa vào trong phân phối nguồn của PostgreSQL.

DIRECTORY phải được chỉ định như một đường dẫn tuyệt đối. /usr/share/zoneinfo có khả năng là một thư mục trong một số hệ điều hành. Lưu ý rằng thủ tục cài đặt sẽ không dò tìm ra sự không khớp hoặc lỗi dữ liệu vùng thời gian. Nếu bạn sử dụng lựa chọn này, thì bạn được khuyến cáo chạy các kiểm thử đệ quy để kiểm tra các dữ liệu vùng thời gian mà bạn đã trỏ tới các công việc một cách đúng đắn với PostgreSQL.

Lựa chọn này chủ yếu nhằm vào các nhà phân phối gói nhị phân mà họ cũng biết hệ điều hành đích của họ. Ưu điểm chính của việc sử dụng lựa chọn này là gói PostgreSQL sẽ không cần phải được nâng cấp bất kỳ khi nào bất kỳ qui tắc thời gian tiết kiệm ánh sáng ban ngày của nhiều địa phương thay đổi. Ưu điểm khác là PostgreSQL có thể được liên biên dịch thẳng trực tiếp hơn nếu các tệp cơ sở dữ liệu

vùng thời gian không cần phải được xây dựng trong quá trình cài đặt.

--without-zlib

Ngăn cản sử dụng thư viện Zlib. Điều này vô hiệu hóa sự hỗ trợ cho các lưu trữ được nén trong pg_dump và pg_restore. Lựa chọn này chỉ có ý định cho những hệ thống hiếm thấy nơi mà thư viện này không sẵn sàng.

--enable-debug

Biên dịch tất cả các chương trình và thư viện với các biểu tượng gỡ lỗi. Điều này có nghĩa là bạn có thể chạy các chương trình trong một trình gỡ lỗi để phân tích các vấn đề. Điều này mở rộng kích thước các tệp thực thi được cài đặt một cách đáng kể, và trong các trình biên dịch không phải là GCC nó cũng thường vô hiệu hóa sự tối ưu hóa trình biên dịch, gây ra sự trì trệ. Tuy nhiên, có các biểu tượng sẵn sàng là cực kỳ hữu dụng để làm việc với bất kỳ vấn đề nào có thể nảy sinh. Hiện hành, lựa chọn này được khuyến cáo cho các cài đặt sản xuất chỉ nếu bạn sử dụng GCC. Nhưng bạn nên luôn có nó nếu đang tiến hành phát triển công việc hoặc chạy một phiên bản beta.

--enable-coverage

Nếu đang sử dụng GCC, tất cả các chương trình và thư viện được biên dịch với sự dàn phối kiểm thử mã bao trùm. Khi chạy, chúng sinh ra các tệp trong thư mục được xây dựng với các đo đếm bao trùm mã. Xem Phần 30.4 để có thêm thông tin. Lựa chọn này chỉ để sử dụng với GCC và khi tiến hành công việc phát triển.

--enable-profiling

Nếu sử dụng GCC, tất cả các chương trình và thư viện được biên dịch sao cho chúng có thể được tóm tắt lại. Lựa chọn này chỉ để sử dụng với GCC và khi tiến hành công việc phát triển.

--enable-cassert

Cho phép khẳng định các kiểm tra trong máy chủ, kiểm tra đối với nhiều điều kiện "không thể xảy ra". Điều này là có giá trị cho các mục đích phát triển mã, nhưng các kiểm tra có thể làm chậm máy chủ đáng kể. Hơn nữa, có các kiểm tra được bật sẽ không nhất thiết cải thiện được tính ổn định máy chủ của bạn! Các kiểm tra khẳng định sẽ không được phân loại cho tính khắc nghiệt, và vì thế những gì có thể là một lỗi tương đối vô hại vẫn sẽ dẫn tới việc khởi động lại máy chủ nếu nó làm bật dậy một khẳng định hỏng. Lựa chọn này không được khuyến cáo cho sử dụng trong sản xuất, nhưng bạn nên có nó cho công việc phát triển hoặc khi chạy phiên bản beta.

--enable-depend

Cho phép lần vết sự phụ thuộc một cách tự động. Với lựa chọn này, các makefiles được thiết lập sao cho tất cả các tệp đối tượng bị ảnh hưởng sẽ được xây dựng lại khi bất kỳ tệp đầu đề nào bị thay đổi. Điều này là hữu dụng nếu bạn đang tiến hành công việc phát triển, nhưng chỉ là lãng phí toàn bộ nếu bạn có ý định chỉ biên dịch một lần và cài đặt. Hiện tại, lựa chọn này chỉ làm việc với GCC.

--enable-dtrace

Biên dịch PostgreSQL với sự hỗ trợ cho công cụ lần vết động DTrace. Xem Phần 27.4 để có thêm thông tin.

Để trỏ tới chương trình dtrace, biến môi trường DTRACE có thể được thiết lập. Điều này sẽ thường là cần thiết vì dtrace điển hình được cài đặt trong /usr/sbin, nó có thể không nằm trong đường dẫn đó.

Các lựa chọn dòng lệnh thêm cho chương trình dtrace có thể được chỉ định trong biến môi trường DTRACEFLAGS. Trong Solaris, để đưa vào sự hỗ trợ của DTrace trong một tệp nhị phân 64 bit, bạn phải chỉ định DTRACEFLAGS="-64" để thiết lập cấu hình. Ví dụ, bằng việc sử dụng trình biên dịch GCC:

```
./configure CC='gcc -m64' --enable-dtrace DTRACEFLAGS='-64' ...
```

Bằng việc sử dụng trình biên dịch của Sun:

```
./configure CC='/opt/SUNWspro/bin/cc -xtarget=native64' --enable-dtrace DTRACEFLAGS='-
```

Nếu bạn thích hơn một trình biên dịch C khác với trình biên dịch mà configure chọn, thì bạn có thể thiết lập biến môi trường CC cho chương trình theo lựa chọn của bạn. Mặc định, configure sẽ chọn gcc nếu sẵn sàng, nếu không thì mặc định của nền tảng đó (thường là cc). Tương tự, bạn có thể ghi đè các cờ của trình biên dịch mặc định nếu cần thiết bằng biến CFLAGS.

Bạn có thể chỉ định các biến môi trường trong dòng lệnh configure, ví dụ:

```
./configure CC=/opt/bin/gcc CFLAGS='-O2 -pipe'
```

Đây là danh sách các biến quan trọng có thể được thiết lập theo cách này:

BISON

Chương trình Bison

CC

Trình biên dịch C

CFLAGS

Lựa chọn để truyền tới trình biên dịch C

CPP

Bộ vi xử lý trước C

CPPFLAGS

Các lựa chọn để truyền tới bộ vi xử lý trước C

DTRACE

Vị trí của chương trình dtrace

DTRACEFLAGS

Các lựa chọn để truyền tới chương trình dtrace

FLEX

Chương trình Flex

LDFLAGS

Các lựa chọn để sử dụng khi việc liên kết hoặc các thực thi hoặc là các thư viện được chia sẻ

LDFLAGS_EX

Các lựa chọn bổ sung cho việc liên kết chỉ các thực thi

LDFLAGS_SL

Các lựa chọn bổ sung cho việc liên kết chỉ các thư viện được chia sẻ

MSGFMT

Chương trình msgfmt cho sự hỗ trợ ngôn ngữ bẩm sinh

PERL

Đường dẫn đầy đủ tới trình biên dịch Perl. Điều này sẽ được sử dụng để xác định các phụ thuộc cho việc xây dựng PL/Perl.

PYTHON

Đường dẫn đầy đủ tới trình biên dịch Python. Điều này sẽ được sử dụng để xác định các phụ thuộc cho việc xây dựng PL/Python. Hơn nữa, hoặc Python 2 hoặc 3 được chỉ định ở đây (hoặc nếu không thì được chọn ngầm) xác định phương án nào của ngôn ngữ PL/Python sẽ trở nên sẵn sàng. Xem Phần 42.1 để có thêm thông tin.

TCLSH

Đường dẫn đầy đủ tới trình biên dịch Tcl. Điều này sẽ được sử dụng để xác định các phụ thuộc cho việc xây dựng PL/Tcl, và nó sẽ bị thay thế thành các script Tcl.

XML2_CONFIG

Chương trình xml2-config được sử dụng để định vị cài đặt libxml.

2. Xây dựng

Để bắt đầu xây dựng, hãy gõ:

```
gmake
```

(Hãy nhớ sử dụng GNU make). Sự xây dựng sẽ mất ít phút, phụ thuộc vào phần cứng của bạn. Dòng cuối cùng được hiển thị sẽ là:

```
All of PostgreSQL is successfully made. Ready to install.
```

(Tất cả PostgreSQL được làm thành công. Sẵn sàng để cài đặt).

Nếu bạn muốn xây dựng mọi điều có thể được xây, thì hãy đưa vào tài liệu (các trang HTML và man), và các module bổ sung (contrib), hãy gõ thay vào đó:

```
gmake world
```

Dòng cuối cùng được hiển thị sẽ là:

```
PostgreSQL, contrib and HTML documentation successfully made. Ready to install.
```

(PostgreSQL, tài liệu contrib và HTML được làm thành công. Sẵn sàng để cài đặt).

3. Các kiểm tra đệ quy

Nếu bạn muốn kiểm tra máy chủ mới được xây dựng trước khi bạn cài đặt nó, bạn có thể chạy các kiểm tra đệ quy ở thời điểm này. Các kiểm tra đệ quy là kiểm tra phù hợp để kiểm

tra rằng PostgreSQL chạy trong máy của bạn theo cách mà các lập trình viên kỳ vọng nó chạy. Hãy gõ vào:

```
gmake check
```

(Điều này sẽ không làm việc ở gốc root; hãy làm nó như một người sử dụng không có quyền ưu tiên). Chương 30 có các thông tin chi tiết về việc biên dịch các kết quả kiểm thử. Bạn có thể lặp lại kiểm thử này bất kỳ khi nào sau này bằng việc đưa ra lệnh y hệt.

4. Cài đặt các tệp

   Lưu ý: Nếu bạn đang nâng cấp một hệ thống đang tồn tại và sẽ cài đặt các tệp mới đè lên các tệp cũ, hãy chắc chắn sao lưu các dữ liệu của bạn và tắt máy chủ cũ trước khi xử lý, như được giải thích trong Phần 15.4 ở trên.

   Để cài đặt PostgreSQL hãy gõ vào:

   ```
   gmake install
   ```

   Điều này sẽ cài đặt các tệp vào các thư mục từng được chỉ định ở bước 1. Hãy chắc chắn rằng bạn có các quyền phù hợp để ghi vào khu vực đó. Bình thường bạn cần làm bước này như là gốc root. Như một lựa chọn thay thế, bạn có thể tạo các thư mục đích trước đó và dàn xếp các quyền phù hợp sẽ được trao.

   Để cài đặt tài liệu (các trang HTML và man), hãy gõ vào:

   ```
   gmake install-docs
   ```

   Nếu bạn xây dựng tổng thể ở trên, thì thay vào đó hãy gõ vào:

   ```
   gmake install-world
   ```

   Điều này cũng cài đặt tài liệu luôn.

   Bạn có thể sử dụng `gmake install-strip` thay cho `gmake install` để lấy các tệp thực thi và các thư viện khi chúng được cài đặt. Điều này sẽ tiết kiệm một vài không gian đĩa. Nếu bạn xây dựng bằng việc gỡ lỗi sự hỗ trợ, thì việc lấy này sẽ loại bỏ có hiệu quả việc gỡ lỗi sự hỗ trợ, nên nó chỉ nên được thực hiện nếu việc gỡ lỗi không còn cần thiết nữa. `install-strip` cố gắng làm một công việc hợp lý để tiết kiệm không gian, nhưng nó không có tri thức tốt về cách để lấy từng byte không cần thiết khỏi một tệp thực thi, nên nếu bạn muốn tiết kiệm tất cả không gian đĩa mà bạn có thể, thì bạn sẽ phải tiến hành công việc đó bằng tay.

   Cài đặt tiêu chuẩn đưa ra tất cả các tệp đầu đề cần thiết cho sự phát triển ứng dụng của các máy trạm cũng như cho sự phát triển chương trình ở phía máy chủ, như các dạng hàm dữ liệu tùy chỉnh được viết trong C. (Trước phiên bản PostgreSQL 8.0, một lệnh `gmake install-all-headers` riêng rẽ từng là cần thiết cho cái sau, nhưng bước này đã được xếp vào cài đặt tiêu chuẩn rồi).

   Cài đặt chỉ máy trạm: Nếu bạn muốn chỉ cài đặt các ứng dụng máy trạm và các thư viện giao diện, thì bạn có thể sử dụng các lệnh:

   ```
   gmake -C src/bin install
   gmake -C src/include install
   gmake -C src/interfaces install
   gmake -C doc install
   ```

   `src/bin` có một ít nhị phân để sử dụng chỉ cho máy chủ, nhưng chúng là nhỏ.

Việc đăng ký lưu ký sự kiện (eventlog) trên Windows: Để đăng ký một thư viện eventlog của Windows với hệ điều hành, hãy đưa ra lệnh này sau khi cài đặt:

```
regsvr32 pgsql_library_directory/pgevent.dll
```

Bỏ cài đặt: Để hủy cài đặt hãy sử dụng lệnh gmake uninstall. Tuy nhiên, điều này sẽ không loại bỏ bất kỳ thư mục nào được tạo ra.

Làm sạch: Sau cài đặt bạn có thể giải phóng không gian đĩa bằng việc loại bỏ các tệp xây dựng khỏi cây nguồn bằng lệnh gmake clean. Điều này sẽ giữ lại các tệp được chương trình configure tạo ra, sao cho bạn có thể xây dựng lại bất kỳ điều gì với gmake sau này. Để thiết lập lại cây nguồn ở tình trạng trong đó nó từng được phân phối, hãy sử dụng gmake distclean. Nếu bạn sẽ xây dựng cho vài nền tảng trong cùng cây nguồn thì bạn phải làm điều này và thiết lập cấu hình lại cho từng nền tảng. (Như một lựa chọn, hãy sử dụng một cây xây dựng riêng rẽ cho từng nền tảng, sao cho cây nguồn vẫn giữ không bị sửa đổi).

Nếu bạn thực hiện một xây dựng và sau đó phát hiện rằng các lựa chọn configure của bạn là sai, hoặc nếu bạn thay đổi bất kỳ điều gì mà configure nghiên cứu (ví dụ, các nâng cấp phần mềm), thì là ý tưởng tốt để tiến hành gmake distclean trước khi thiết lập cấu hình lại và xây dựng lại. Không có điều này, thì các thay đổi của bạn trong các lựa chọn cấu hình có thể không nhân giống ở từng nơi mà chúng cần phải làm.

15.6. Thiết lập sau cài đặt

15.6.1. Thư viện được chia sẻ

Trong một số hệ thống với các thư viện được chia sẻ bạn cần nói cho hệ thống biết cách để tìm các thư viện được chia sẻ mới được cài đặt. Các hệ thống trong đó điều này là không cần thiết bao gồm BSD/OS, FreeBSD, HP-UX, IRIX, Linux, NetBSD, OpenBSD, Tru64 UNIX (trước đây là Digital UNIX), và Solaris.

Phương pháp để thiết lập các giá trị đường dẫn tìm kiếm thư viện được chia sẻ là khác nhau giữa các nền tảng, nhưng phương pháp được sử dụng rộng rãi nhất là thiết lập biến môi trường LD_LIBRARY_PATH như sau: Trong các trình biên dịch shell Bourne (sh, ksh, bash, zsh):

```
LD_LIBRARY_PATH=/usr/local/pgsql/lib
export LD_LIBRARY_PATH
```

hoặc trong csh hoặc tcsh:

```
setenv LD_LIBRARY_PATH /usr/local/pgsql/lib
```

Thay thế /usr/local/pgsql/lib bằng bất kỳ thứ gì bạn thiết lập --libdir trong bước 1. Bạn nên đặt các lệnh đó vào một tệp khởi tạo shell như /etc/profile hoặc ~/.bash_profile.

Một số thông tin tốt lành về những khiếm khuyết có liên quan tới phương pháp này có thể thấy trong http://xahlee.org/UnixResource_dir/_/ldpath.html.

Trong một số hệ thống có thể ưu tiên thiết lập biến môi trường LD_RUN_PATH trước khi xây dựng.

Trong Cygwin, đặt thư mục thư viện trong PATH hoặc chuyển các tệp .dll vào thư mục bin. Nếu nghi ngờ, hãy tham chiếu tới các trang hướng dẫn hệ thống của bạn (có lex là ld.so hoặc rld). Nếu sau đó bạn có một thông điệp giống như:

```
psql: error in loading shared libraries
libpq.so.2.1: cannot open shared object file: No such file or directory
```

thì bước này là cần thiết. Đơn giản hãy quan tâm tới nó sau đó.

Nếu bạn ở trong BSD/OS, Linux, hoặc SunOS 4 và bạn có sự truy cập gốc root thì bạn có thể chạy:

```
/sbin/ldconfig /usr/local/pgsql/lib
```

(hoặc thư mục tương ứng) sau khi cài đặt cho phép trình kết nối thời gian chạy tìm các thư viện được chia sẻ nhanh hơn. Hãy tham chiếu tới trang hướng dẫn ldconfig để có thêm thông tin. Trong FreeBSD, NetBSD, và OpenBSD thì lệnh, thay vào đó, là:

```
/sbin/ldconfig -m /usr/local/pgsql/lib
```

Các hệ thống khác còn chưa biết có một lệnh tương đương hay không.

15.6.2. Biến môi trường

Nếu bạn đã cài đặt vào /usr/local/pgsql hoặc một số địa điểm khác mà không được tìm thấy đối với các chương trình theo mặc định, thì bạn nên bổ sung /usr/local/pgsql/bin (hoặc bất kỳ thứ gì bạn thiết lập cho --bindir ở bước 1) vào đường dẫn PATH của bạn. Nói một cách nghiêm túc, điều này không là cần thiết, nhưng nó sẽ làm cho sử dụng PostgreSQL tiện hơn nhiều.

Để làm điều này, hãy bổ sung thứ sau đây vào tệp khởi tạo shell của bạn, như ~/.bash_profile (hoặc /etc/profile, nếu bạn muốn nó tác động tới tất cả những người sử dụng):

```
PATH=/usr/local/pgsql/bin:$PATH
export PATH
```

Nếu bạn đang sử dụng csh hoặc tcsh, thì hãy sử dụng lệnh này:

```
set path = ( /usr/local/pgsql/bin $path )
```

Để cho phép hệ thống của bạn tìm thấy tài liệu man, bạn cần bổ sung các dòng như sau vào một tệp khởi tạo shell trừ phi bạn đã cài đặt vào một ví trí mà tìm thấy được theo mặc định:

```
MANPATH=/usr/local/pgsql/man:$MANPATH
export MANPATH
```

Các biến môi trường PGHOST và PGPORT chỉ định tới các ứng dụng máy trạm máy chủ host và cổng của máy chủ cơ sở dữ liệu, ghi đè các mặc định được biên dịch. Nếu bạn sẽ chạy các ứng dụng máy trạm ở xa thì là thuận tiện nếu mỗi người sử dụng có kế hoạch sử dụng cơ sở dữ liệu đó thiết lập PGHOST. Tuy nhiên, điều này không theo yêu cầu; các thiết lập có thể được giao tiếp thông qua các lựa chọn dòng lệnh tới hầu hết các chương trình máy trạm.

15.7. Nền tảng được hỗ trợ

Một nền tảng (đó là, sự kết hợp của một kiến trúc CPU và hệ điều hành) được xem xét được cộng đồng phát triển PostgreSQL hỗ trợ nếu mã có chứa các phần để làm việc trên nền tảng đó và nó gần đây được kiểm tra để xây dựng và truyền qua các kiểm tra đệ quy của nó trên nền tảng đó. Hiện hành, hầu hết việc kiểm tra tính tương thích nền tảng được thực hiện tự động bằng việc kiểm tra các máy trong trại xây dựng (Build Farm[2]) của PostgreSQL. Nếu bạn có quan tâm trong việc sử dụng PostgreSQL trên một nền tảng mà không được trình bày trong một trại xây dựng, mà trong đó mã làm việc hoặc có thể được làm để làm việc, thì bạn được khuyến khích mạnh mẽ để thiết lập máy

2 http://buildfarm.postgresql.org/

thành viên của một trại xây dựng sao cho tính tương thích liên tục có thể được đảm bảo.

Nói chung, PostgreSQL có thể được kỳ vọng làm việc trong các kiến trúc CPU đó: x86, x86_64, IA64, PowerPC, PowerPC 64, S/390, S/390x, Sparc, Sparc 64, Alpha, ARM, MIPS, MIPSEL, M68K, và PA-RISC. Sự hỗ trợ mã là sẵn sàng cho M32R, NS32K, và VAX, nhưng các kiến trúc đó không được biết có được kiểm thử hay chưa gần đây. Thường có khả năng để xây dựng trên một dạng CPU không được hỗ trợ bằng việc thiết lập cấu hình bằng --disable-spinlocks, nhưng hiệu năng sẽ là tồi.

PostgreSQL có thể được kỳ vọng làm việc trong các hệ điều hành: Linux (tất các các phát tán gần đây), Windows (Win2000 SP4 và sau này), FreeBSD, OpenBSD, NetBSD, Mac OS X, AIX, HP/UX, IRIX, Solarí, Tru64 Unix, và UnixWare. Các hệ thống giống Unix khác cũng có thể làm việc nhưng hiện còn chưa được kiểm thử. Trong hầu hết các trường hợp, tất cả các kiến trúc CPU được một hệ điều hành cho trước hỗ trợ sẽ làm việc. Hãy nhìn vào Phần 15.8 bên dưới để thấy liệu có thông tin đặc thù cho hệ điều hành của bạn hay không, đặc biệt nếu đang sử dụng một hệ thống cũ hơn.

Nếu bạn có các vấn đề về cài đặt trên một hệ thống mà được biết sẽ được hỗ trợ theo các kết quả của trại xây dựng gần đây, xin hãy báo cáo nó cho <pgsql-bugs@postgresql.org >. Nếu bạn có quan tâm trong việc chuyển PostgreSQL sang một nền tảng mới, <pgsql-hackers@postgresql.org > là chỗ phù hợp để thảo luận điều đó.

15.8. Lưu ý cho nền tảng đặc thù

Phần này ghi lại các vấn đề bổ sung có đặc thù nền tảng về sự cài đặt và thiết lập cấu hình của PostgreSQL. Hãy chắc chắn đọc các chỉ dẫn cài đặt, và đặc biệt là cả Phần 15.2 nữa. Hơn nữa, hãy kiểm tra Chương 30 về sự tương hợp các kết quả kiểm thử hồi quy.

Các nền tảng mà không được chuyển đổi ở đây không có các vấn đề cài đặt đặc thù nền tảng.

15.8.1. AIX

PostgreSQL làm việc được trên AIX, nhưng để làm cho nó được cài đặt đúng có thể là một thách thức. Các phiên bản AIX từ 4.3.3 tới 6.1 được xem là được hỗ trợ. Bạn có thể sử dụng GCC hoặc trình biên dịch bẩm sinh của IBM là xlc. Nói chung, việc sử dụng các phiên bản gần đây của AIX được biết là có làm việc.

Các mức sửa lỗi tối thiểu được khuyến cáo cho các phiên bản AIX được hỗ trợ là:

AIX 4.3.3
: Mức duy trì - ML (Maintainance Level) 11 + cụm sau ML 11

AIX 5.1
: Mức duy trì 9 + cụm sau ML 9

AIX 5.2
: Múc công nghệ 10 Service Pack 3

AIX 5.3
: Mức công nghệ 7

AIX 6.1
: Mức cơ bản

Để kiểm tra mức sửa lỗi hiện hành của bạn, hãy sử dụng oslevel -r trong AIX 4.3.3 tới AIX 5.2 ML 7, hoặc oslevel -s trong các phiên bản sau này.

Hãy sử dụng các cờ configure sau bổ sung cho riêng bạn nếu bạn đã cài đặt Readline hoặc libz trong /usr/local: --with-includes=/usr/local/include –with-libraries=/usr/local/lib.

15.8.1.1. Vấn đề của GCC

Trong AIX 5.3, đã có một số vấn đề làm cho PostgreSQL biên dịch và chạy có sử dụng GCC.

Bạn sẽ muốn sử dụng một phiên bản GCC sau 3.3.2, đặc biệt nếu bạn sử dụng một phiên bản được đóng gói sẵn trước. Chúng tôi đã thành công tốt với 4.0.1. Các vấn đề với các phiên bản trước đó dường như có nhiều điều hơn để làm theo cách mà IBM đã đóng gói GCC hơn là với các vấn đề thực sự của GCC, sao cho nếu bạn tự mình biên dịch GCC, thì bạn có lẽ sẽ thành công với một phiên bản GCC trước đó.

15.8.1.2. Socket miền Unix bị gẫy

AIX 5.3 có một vấn đề nơi mà sockaddr_storage không được xác định là đủ lớn. Trong phiên bản 5.3, IBM đã nâng kích thước sockaddr_un, cấu trúc địa chỉ đối với các sockets miền Unix, nhưng đã không gia tăng một cách tương ứng kích thước của sockaddr_storage. Kết quả của điều này là những cố gắng sử dụng các sockets miền Unix với PostgreSQL dẫn tới libpq tràn cấu trúc dữ liệu. Các kết nối TCP/IP làm việc OK, nhưng các sockets miền Unix thì không, chúng cản trở các kiểm tra đệ quy khi làm việc.

Vấn đề đó từng được báo cáo cho IBM, và được ghi lại như một báo cáo lỗi PMR29657. Nếu bạn nâng cấp lên mức duy trì 5300-03 hoặc sau này, thì điều đó sẽ bao gồm cả bản vá này rồi. Một sự khắc phục nhanh là tùy chỉnh _SS_MAXSIZE thành 1025 trong /usr/include/sys/socket.h. Trong cả 2 trường hợp, hãy biên dịch lại PostgreSQL một khi bạn có tệp đầu đề được sửa đúng.

15.8.1.3. Vấn đề địa chỉ Internet

PostgreSQL dựa vào hàm getaddrinfo của hệ thống để phân tích các địa chỉ IP trong listen_addresses , pg_hba.conf ... Các phiên bản cũ hơn của AIX đã có các loại lỗi trong hàm này. Nếu bạn có các vấn đề liên quan tới các thiết lập đó, thì việc nâng cấp lên mức sửa AIX phù hợp được chỉ ra ở trên sẽ quan tâm tới nó.

Một người sử dụng báo cáo:

Khi triển khai PostgreSQL phiên bản 8.1 trên AIX 5.3, chúng tôi theo định kỳ đã gặp các vấn đề nơi mà các trình thu thập số liệu thống kê có thể không làm việc thành công một cách “bí ẩn”. Điều này dường như là kết quả của hành vi không mong đợi trong triển khai IPv6. Nó giống như là PostgreSQL và IPv6 không cùng nhau hoạt động tốt được trên AIX 5.3.

Bất kỳ trong số các hành động nào sau đây sẽ “sửa” vấn đề đó.

- Xóa địa chỉ IPv6 đối với localhost:

  ```
  (as root)
  # ifconfig lo0 inet6 ::1/0 delete
  ```

- Loại bỏ IPv6 khỏi các dịch vụ net. Tệp /etc/netsvc.conf trên AIX tương ứng thô với /etc/nsswitch.conf trên Solaris/Linux. Mặc định, trên AIX, là:

hosts=local,bind

Thay thế thứ này bằng:

hosts=local4,bind4

để giải hoạt việc tìm kiếm các địa chỉ IPv6.

Cảnh báo

Đây thực sự là sự khắc phục cho các vấn đề có liên quan tới sự chưa chín muồi hỗ trợ IPv6, nó được cải thiện trông thấy trong quá trình các phát hành AIX 5.3. Nó đã làm việc với AIX phiên bản 5.3, nhưng không thể hiện là một giải pháp tao nhã cho vấn đề này. Được thông báo rằng sự khắc phục này không chỉ là không cần thiết, mà còn gây ra các vấn đề trong AIX 6.1, nơi mà sự hỗ trợ IPv6 đã trở nên chín muồi hơn.

15.8.1.4. Quản lý bộ nhớ

AIX có thể là hơi đặc biệt theo cách mà nó làm việc về quản lý bộ nhớ. Bạn có thể có một máy chủ với nhiều GB RAM tự do, nhưng vẫn bị hết bộ nhớ hoặc các lỗi không gian địa chỉ khi chạy các ứng dụng. Một ví dụ là việc createlang thất bại với các lỗi không thông thường. Ví dụ, chạy như là người chủ của cài đặt PostgreSQL:

```
-bash-3.00$ createlang plperl template1
createlang: language installation failed: ERROR: could not load library "/opt/dbs/pgsql748/
```

Chạy như một người không phải là chủ trong nhóm xử lý cài đặt PostgreSQL:

```
-bash-3.00$ createlang plperl template1
createlang: language installation failed: ERROR: could not load library "/opt/dbs/pgsql748/
```

Một ví dụ khác là các lỗi tràn bộ nhớ trong các lưu ký máy chủ PostgreSQL, với mỗi phân bổ bộ nhớ gần hoặc lớn hơn 256 MB hỏng.

Lý do tổng thể của tất cả các vấn đề đó là mô hình bộ nhớ và bit mặc định được tiến trình máy chủ sử dụng. Theo mặc định, tất cả các tệp nhị phân được xây dựng trên AIX là 32 bit. Điều này không phụ thuộc vào dạng phần cứng hoặc nhân khi sử dụng. Các tiến trình 32 bit đó bị giới hạn 4 GB bộ nhớ được đặt ra trong các đoạn 256 MB bằng việc sử dụng một trong ít mô hình đó. Mặc định cho phép ít hơn 256 MB trong đống khi nó chia sẻ một đoạn duy nhất với kho.

Trong trường hợp ví dụ createlang ở trên, hãy kiểm tra lệnh thiết lập mặt nạ (umask) và các quyền đối với các nhị phân của bạn trong cài đặt PostgreSQL. Các nhị phân có liên quan trong ví dụ đó từng là 32 bit và được cài đặt như là chế độ 750 thay vì 755. Vì các quyền đang được thiết lập theo cách này, chỉ người chủ sở hữu hoặc một thành viên của nhóm sở hữu chủ có thể tải được thư viện. Vì nó không phải là đọc được cho tất cả, trình tải đặt đối tượng vào đống tiến trình thay vì các đoạn thư viện được chia sẻ nơi mà nó có thể sẽ được đặt nếu khác.

Giải pháp "lý tưởng" cho điều này là hãy sử dụng một xây dựng PostgreSQL 64 bit, nhưng điều đó không luôn là thực tế, vì các hệ thống với các vi xử lý 32 bit có thể xây dựng, nhưng không chạy, các nhị phân 64 bit.

Nếu một nhị phân 32 bit là mong muốn, hãy thiết lập LDR_CNTRL thành MAXDATA=0xn0000000, trong đó 1<= n <=8, trước khi khởi động máy chủ PostgreSQL, và cố thử các giá trị khác và các thiết lập postgresql.conf để tìm một cấu hình làm việc thỏa mãn được. Sử dụng LDR_CNTRL này nói cho AIX rằng bạn muốn máy chủ có các bytes MAXDATA được thiết lập bên cạnh cho đống đó, được phân bổ theo các đoạn 256 MB. Khi bạn thấy một cấu hình làm việc được, thì ldedit có thể được sử dụng để sửa đổi các nhị phân sao cho chúng mặc định sử dụng kích thước đống mong muốn. PostgreSQL cũng có thể được xây dựng lại, truyền configure LDFLAGS="-Wl,-bmaxdata:0xn0000000" để đạt được hiệu ứng y hệt.

Đối với một xây dựng 64 bit, hãy thiết lập OBJECT_MODE thành 64 và truyền CC="gcc -maix64" và LDFLAGS="-Wl,-bbigtoc" tới configure. (Các lựa chọn cho xlc có thể khác). Nếu bạn bỏ qua xuất khẩu của OBJECT_MODE, thì xây dựng của bạn có thể hỏng với các lỗi của trình liên kết (linker). Khi OBJECT_MODE được thiết lập, nó nói cho các tiện ích được xây dựng của AIX như ar, as, và ld dạng các đối tượng nào mặc định cho việc điều khiển.

Mặc định, sự đệ trình quá (overcommit) không gian tạo trang có thể xảy ra. Trong khi chúng ta không thấy điều này xảy ra, thì AIX sẽ giết các tiến trình khi nó chạy hết bộ nhớ và sự đệ trình quá được truy cập. Thứ gần nhất với điều này mà chúng tôi đã thấy là hỏng sự rẽ nhánh vì hệ thống đã quyết định rằng đã không có đủ bộ nhớ cho tiến trình khác. Giống như các phần khác của AIX, phương pháp phân bổ không gian tạo trang và giết tràn bộ nhớ là có khả năng thiết lập cấu hình được trên cơ sở một hệ thống hoặc tiến trình nếu điều này trở thành một vấn đề.

Các tài liệu tham chiếu

- "Large Program Support[1]", *AIX Documentation: General Programming Concepts: Writing and Debugging Programs*.
- "Program Address Space Overview[2]", *AIX Documentation: General Programming Concepts: Writing and Debugging Programs*.
- "Performance Overview of the Virtual Memory Manager (VMM)[3]", *AIX Documentation: Performance Management Guide*.
- "Page Space Allocation[4]", *AIX Documentation: Performance Management Guide*.
- "Paging-space thresholds tuning[5]", *AIX Documentation: Performance Management Guide*.
- *Developing and Porting C and C++ Applications on AIX*[6], IBM Redbook.

15.8.2. Cygwin

PostgreSQL có thể được xây dựng bằng việc sử dụng Cygwin, một môi trường giống Linux cho Windows, nhưng phương pháp đó là không tốt so với xây dựng Windows bẩm sinh (xem Chương 16) và không còn được khuyến cáo.

1 http://publib.boulder.ibm.com/infocenter/pseries/topic/com.ibm.aix.doc/aixprggd/genprogc/lrg_prg_support.htm
2 http://publib.boulder.ibm.com/infocenter/pseries/topic/com.ibm.aix.doc/aixprggd/genprogc/address_space.htm
3 http://publib.boulder.ibm.com/infocenter/pseries/v5r3/topic/com.ibm.aix.doc/aixbman/prftungd/resmgmt2.htm
4 http://publib.boulder.ibm.com/infocenter/pseries/v5r3/topic/com.ibm.aix.doc/aixbman/prftungd/memperf7.htm
5 http://publib.boulder.ibm.com/infocenter/pseries/v5r3/topic/com.ibm.aix.doc/aixbman/prftungd/memperf6.htm
6 http://www.redbooks.ibm.com/abstracts/sg245674.html?Open

Khi xây dựng từ nguồn, hãy xử lý theo thủ tục cài đặt (nghĩa là ./configure; make ...), lưu ý các khác biệt đặc thù Cygwin sau đây:

- Thiết lập đường dẫn của bạn để sử dụng thư mục Cygwin bin trước các tiện ích của Windows. Điều này sẽ giúp ngăn cản các vấn đề khi biên dịch.
- Lệnh GNU make được gọi là make, không phải là gmake.
- Lệnh adduser không được hỗ trợ; hãy sử dụng ứng dụng quản lý người sử dụng phù hợp trong Windows NT, 2000, hoặc XP. Nếu không, hãy bỏ qua bước này.
- Lệnh su không được hỗ trợ; hãy sử dụng ssh để bắt chước su trên Windows NT, 2000, hoặc XP. Nếu không, hãy bỏ qua bước này.
- OpenSSL không được hỗ trợ
- Khởi động cygserver để hỗ trợ bộ nhớ được chia sẻ. Để làm điều này, hãy vào lệnh /usr/sbin/cygserver &. Chương trình này cần được chạy bất kỳ khi nào bạn khởi động máy chủ PostgreSQL hoặc khởi tạo một cụm cơ sở dữ liệu (initdb). Cấu hình cygserver mặc định có thể cần phải thay đổi (như, gia tăng SEMMNS) để ngăn cản PostgreSQL khỏi hỏng vì thiếu các tài nguyên hệ thống.
- Các kiểm tra đệ quy song song (make check) có thể tạo ra kiểm tra đệ quy giả bị hỏng vì tràn hàng đợi lưu ký ngược (backlog) listen(), là lý do các lỗi kết nối bị từ chối hoặc treo. Bạn có thể hạn chế số lượng các kết nối bằng việc sử dụng biến của make là MAX_CONNECTIONS:
  make MAX_CONNECTIONS=5 check
  (Trong một số hệ thống bạn có thể có tới khoảng 10 kết nối cùng một lúc).

Có khả năng cài đặt cygserver và máy chủ PostgreSQL như các dịch vụ Windows NT. Để có thông tin về cách làm điều này, xin tham chiếu tới tài liệu README được đưa vào trong gói nhị phân PostgreSQL trong Cygwin. Nó được cài đặt trong thư mục /usr/share/doc/Cygwin.

15.8.3. HP-UX

PostgreSQL 7.3+ sẽ làm việc trong các máy Series 700/800 PA-RISC chạy HP-UX 10.X hoặc 11.X, đưa ra các mức vá hệ thống phù hợp và các công cụ xây dựng. Ít nhất 1 lập trình viên thường xuyên kiểm tra trên HP-UX 10.20, và chúng tôi có các báo cáo các cài đặt thành công trên HP-UX 11.00 và 11.11.

Bên cạnh phân phối nguồn PostgreSQL, bạn sẽ cần GNU make (make của HP sẽ không làm việc), và cả GCC và trình biên dịch ANSI C đầy đủ của HP. Nếu bạn định xây dựng từ các nguồn của Git thay vì tệp nén rar của một phân phối, thì bạn cũng sẽ cần Flex (GNU lex) và Bison (GNU yacc). Chúng tôi cũng khuyến cáo chắc chắn bạn được cập nhật tốt với các bản vá HP. Tối thiểu, nếu bạn đang xây dựng các nhị phân 64 bit trên HP-UX 11.11 thì bạn có thể cần PHSS_30966 (11.11) hoặc một bản vá tiếp sau, nếu không initdb có thể treo:

PHSS_30966 s700_800 ld(1) và bản vá tích lũy các công cụ trình liên kết (linker)

Trên các nguyên tắc chung hiện tại bạn có các bản vá libc và ld/dld, cũng như các bản vá trình biên dịch nếu bạn đang sử dụng trình biên dịch C của HP. Xem các site hỗ trợ HP như http://itrc.hp.com

và ftp://usffs.external.hp.com/ cho các bản sao tự do các bản vá mới nhất của họ.

Nếu bạn đang xây dựng trên một máy PA-RISC 2.0 và muốn có các nhị phân 64 bit bằng việc sử dụng GCC, thì bạn phải sử dụng phiên bản GCC 64 bit. Các nhị phân GCC cho HP-UX PA-RISC và Itanium là sẵn sàng từ http://www.hp.com/go/gcc. Đừng quên lấy và cài đặt binutils cùng lúc.

Nếu bạn đang xây dựng trên máy PA-RISC 2.0 và muốn các nhị phân được biên dịch chạy trên các máy PA-RISC 1.1 thì bạn sẽ cần phải chỉ định +DAportable trong CFLAGS.

Nếu bạn đang xây dựng trên một máy HP-UX Itanium, bạn sẽ cần trình biên dịch HP ANSI C mới nhất với bản vá phụ thuộc của nó hoặc các bản vá sau này:

PHSS_30848 s700_800 HP C Compiler (A.05.57)
PHSS_30849 s700_800 u2comp/be/plugin library Patch

Nếu bạn có cả trình biên dịch C và GCC của HP, thì bạn có thể muốn chọn rõ ràng trình biên dịch để sử dụng khi bạn chạy configure:

./configure CC=cc

cho trình biên dịch C của HP, hoặc

./configure CC=gcc

cho GCC. Nếu bạn bỏ qua thiết lập này, thì sau đó thiết lập cấu hình sẽ lấy gcc nếu nó có cơ hội.

Vị trí đích cài đặt mặc định là /usr/local/pgsql, nên bạn có thể muốn thay đổi thứ gì đó trong /opt. Nếu thế, hãy sử dụng chuyển mạch --prefix cho configure.

Trong các kiểm thử đệ quy, có thể có một số khác biệt số trật tự thấp trong các kiểm thử theo địa lý, chúng khác nhau, phụ thuộc vào trình biên dịch và các phiên bản thư viện toán học nào bạn sử dụng. Bất kỳ lỗi nào khác cũng là lý do cho sự nghi ngờ.

15.8.4. IRIX

PostgreSQL từng được biết là chạy thành công trên các bộ vi xử lý MIPS r8000, r10000 (cả 2 là ip25 và ip27) và r12000 (ip35), chạy được với IRIX 6.5.5m, 6.5.12, 6.5.13 và 6.5.26 với các bộ biên dịch MIPSPro các phiên bản 7.30, 7.3.1.2m, 7.3 và 7.4.4m.

Bạn sẽ cần MIPSPro với trình biên dịch đầy đủ ANSI C. Có những vấn đề cố gắng xây dựng với GCC. Đây là một lỗi GCC nổi tiếng (không được sửa cho tới phiên bản 3.0) có liên quan tới việc sử dụng các hàm trả về các dạng cấu trúc nhất định. Lỗi này ảnh hưởng tới các hàm như inet_ntoa , inet_lnaof, inet_netof, inet_makeaddr và semctl. Được hỗ trợ để được sửa lỗi bằng việc ép mã liên kết với các hàm đó với libgcc, nhưng điều này còn chưa được kiểm thử.

Được biết rằng trình biên dịch MIPSPro phiên bản 7.4.1m tạo ra mã không đúng. Triệu chứng là "bản ghi điểm kiểm tra đầu tiên không hợp lệ" (invalid primary checkpoint record) khi cố khởi tạo cơ sở dữ liệu. Phiên bản 7.4.4m thì OK; tình trạng các phiên bản ở giữa còn chưa chắc chắn.

Có thể có một vấn đề biên dịch như sau:

```
cc-1020 cc: ERROR File = pqcomm.c, Line = 427
        The identifier "TCP_NODELAY" is undefined.
                if (setsockopt(port->sock, IPPROTO_TCP, TCP_NODELAY,
```

Một vài phiên bản bao gồm các định nghĩa TCP trong sys/xti.h, vì thế là cần thiết để thêm #include <sys/xti.h> vào trong src/backend/libpq/pqcomm.c và src/interfaces/libpq/fe-connect.c. Nếu bạn gặp điều này, hãy cho chúng tôi biết để chúng tôi có thể phát triển một bản sửa lỗi đúng phù hợp.

Trong các kiểm thử đệ quy, có thể có một số khác biệt số trật tự thấp trong các kiểm thử địa lý, phụ thuộc vào FPU nào bạn đang sử dụng. Bất kỳ lỗi nào khác cũng là lý do để nghi ngờ.

15.8.5. MinGW/Windows bẩm sinh

PostgreSQL for Windows có thể được xây dựng bằng việc sử dụng MinGW, một môi trường được xây dựng giống như Unix cho các hệ điều hành của Microsoft, hoặc bằng việc sử dụng bộ biên dịch Visual C++ của Microsoft. MinGW đã xây dựng các phương án sử dụng khác nhau mà hệ thống xây dựng thông thường đã mô tả trong chương này; các công việc được xây dựng bằng Visual C++ hoàn toàn khác và được mô tả trong Chương 16. Nó là một xây dựng bẩm sinh hoàn toàn và không sử dụng các phần mềm bổ sung như MinGW. Một trình cài đặt được làm rồi luôn sẵn sàng trên website chính của PostgreSQL.

Cổng Windows bẩm sinh đòi hỏi một phiên bản Windows 2000 hoặc sau này loại 32 bit hoặc 64 bit. Các hệ điều hành trước đó không có hạ tầng đủ (nhưng Cygwin có thể được sử dụng trên chúng). MinGW, công cụ được xây dựng giống như Unix, và MSYS, một bộ sưu tập các công cụ Unix theo yêu cầu để chạy các script shell giống như configure, có thể được tải về từ http://www.mingw.org/. Chúng đều không được yêu cầu để chạy các nhị phân được tạo ra; chúng chỉ cần thiết cho việc tạo ra các nhị phân đó.

Sau khi bạn đã cài đặt xong mọi thứ, được gợi ý là bạn chạy psql dưới CMD.EXE, khi bảng điều khiển (console) MSYS có các vấn đề về bộ nhớ tạm.

15.8.6. Máy chủ SCO OpenServer và SCO UnixWare

PostgreSQL có thể được xây dựng trên SCO UnixWare 7 và SCO OpenServer 5. Trên SCO OpenServer, bạn có thể sử dụng hoặc bộ công cụ phát triển OpenSer (OpenServer Development Kit) hoặc bộ công cụ phát triển vạn năng (Universal Development Kit). Tuy nhiên, một số việc vọc có thể là cần thiết, như được mô tả ở bên dưới.

15.8.6.1. Skunkware

Bạn nên định vị bảo sao đĩa CD SCO Skunkware của bạn. CD Skunkware được đưa vào với UnixWare 7 và các phiên bản hiện hành của OpenServer 5. Skunkware bao gồm các phiên bản sẵn sàng cài đặt của nhiều chương trình phổ biến và là sẵn sàng trên Internet. Ví dụ, gzip, gunzip, GNU Make, Flex, và Bison, tất cả đều được thêm vào. Đối với UnixWare 7.1, CD này bây giờ có nhãn là "Open License Software Supplement" (Bổ sung Phần mềm Giấy phép Mở). Nếu bạn không có đĩa CD này, thì các phần mềm đó là sẵn sàng ở http://www.sco.com/skunkware/.

Skunkware có các phiên bản khác nhau cho UnixWare và OpenServer. Hãy chắc chắn bạn cài đặt phiên bản đúng cho hệ điều hành của bạn, ngoại trừ như được lưu ý bên dưới.

Trên UnixWare 7.1.3 và hơn thế, trình biên dịch GCC được đưa vào trong UDK CD như là GNU Make.

15.8.6.2. GNU Make

Bạn cần sử dụng chương trình GNU Make, nó có trong đĩa CD Skunkware. Mặc định, nó cài đặt như là /usr/local/bin/make. Để tránh sự lúng túng với chương trình SCO make, bạn có thể muốn đổi tên SCO make thành make.

Như của UnixWare 7.1.3 và ở trên, chương trình GNU Make là một phần OSTK của UDK CD, và là trong /usr/gnu/bin/gmake.

15.8.6.3. Readline

Thư viện Readline có trên đĩa CD Skunkware. Nhưng nó không được đưa vào trong CD Skunkware UnixWare 7.1. Nếu bạn có các đĩa CD Skunkware UnixWare 7.0.0 hoặc 7.0.1, thì bạn có thể cài đặt nó từ đó. Nếu không, hãy thử http://www.sco.com/skunkware/.

Mặc định, Readline cài đặt vào /usr/local/lib and /usr/local/include. Tuy nhiên, chương trình configure của PostgreSQL sẽ không thấy nó ở đó mà không có trợ giúp. Nếu bạn đã cài đặt Readline, sau đó hãy sử dụng các lựa chọn sau cho configure:

```
./configure --with-libraries=/usr/local/lib --with-includes=/usr/local/include
```

15.8.6.4. Sử dụng UDK trên OpenServer

Nếu bạn đang sử dụng trình biên dịch của bộ công cụ phát triển vạn năng UDK trên OpenServer, thì bạn cần chỉ định các vị trí các thư viện UDK:

```
./configure --with-libraries=/udk/usr/lib --with-includes=/udk/usr/include
```

Hãy đặt chúng cùng với các lựa chọn Readline từ ở trên:

```
./configure --with-libraries="/udk/usr/lib /usr/local/lib" --with-includes="/udk/usr/include 363
```

15.8.6.5. Đọc các trang man của PostgreSQL

Mặc định, các trang man của PostgreSQL được cài đặt vào /usr/local/pgsql/man. Mặc định, UnixWare không tìm các trang man ở đó. Để có khả năng đọc chúng, bạn cần sửa đổi biến MANPATH trong /etc/default/man, ví dụ:

```
MANPATH=/usr/lib/scohelp/%L/man:/usr/dt/man:/usr/man:/usr/share/man:scohelp:/usr/local/man:/
```

Trên OpenServer, một số nghiên cứu thêm cần được đầu tư để làm cho các trang man hữu dụng, vì hệ thống man là khác một chút với các nền tảng khác. Hiện hành, PostgreSQL sẽ không cài đặt chúng hoàn toàn.

15.8.6.6. Các vấn đề C99 với bổ sung tính năng 7.1.1b

Đối với các trình biên dịch được phát hành trước trình biên dịch với OpenUNIX 8.0.8 (UnixWare 7.1.2), bao gồm cả bổ sung tính năng 7.1.1b (7.1.1b Feature Supplement), bạn có thể cần chỉ định -Xb trong CFLAGS hoặc biến môi trường CC. Chỉ số của điều này là một lỗi trong việc biên dịch tuplesort.c tham chiếu tới các hàm tại chỗ. Có thể đã có một sự thay đổi trong trình biên dịch 7.1.2 (8.0.0) hoặc hơn thế.

15.8.6.7. Tạo luồng trong UnixWare

Đối với việc tạo luồng, bạn phải sử dụng -Kpthread trong tất cả các chương trình sử dụng libpq.

-Kpthread sử dụng các lời gọi pthread_*, chúng chỉ sẵn sàng với cờ -Kpthread/-Kthread.

15.8.7. Solaris

PostgreSQL được hỗ trợ tốt trên Solaris. Hệ điều hành của bạn càng cập nhật, thì càng ít vấn đề bạn sẽ trải nghiệm; các chi tiết ở bên dưới.

Lưu ý rằng PostgreSQL được làm thành cụm với Solaris 10 (từ cập nhật 2). Các gói chính thức cũng sẵn sàng trên http://pgfoundry.org/projects/solarispackages/. Các gói cho các phiên bản Solaris cũ hơn (8, 9) bạn có thể có được từ http://www.sunfreeware.com/ hoặc http://www.blastwave.org/.

15.8.7.1. Công cụ được yêu cầu

Bạn có thể xây dựng bộ biên dịch của Sun hoặc GCC. Để tối ưu hóa mã tốt hơn, trình biên dịch của Sun được khuyến cáo mạnh mẽ trên kiến trúc SPARC. Chúng tôi đã nghe các báo cáo về các vấn đề khi sử dụng gcc 2.95.1; gcc 2.95.3 hoặc sau này được khuyến cáo. Nếu bạn đang sử dụng trình biên dịch của Sun, hãy thận trọng không chọn /usr/ucb/cc; hãy sử dụng /opt/SUNWspro/bin/cc.

Bạn có thể tải về Sun Studio từ http://developers.sun.com/sunstudio/downloads/. Nhiều công cụ GNU được tích hợp vào Solaris 10, hoặc chúng hiện diện trong đĩa đi theo Solaris. Nếu bạn thích các gói cho các phiên bản Solaris cũ, bạn có thể thấy các công cụ đó ở http://www.sunfreeware.com hoặc http://www.blastwave.org. Nếu bạn thích nguồn hơn, hãy xem ở: http://www.gnu.org/order/ftp.html.

15.8.7.2. Vấn đề với OpenSSL

Khi bạn xây dựng PostgreSQL với sự hỗ trợ của OpenSSL, bạn có thể có các lỗi biên dịch trong các tệp sau đây:

- src/backend/libpq/crypt.c
- src/backend/libpq/password.c
- src/interfaces/libpq/fe-auth.c
- src/interfaces/libpq/fe-connect.c

Điều này là vì một xung đột không gian tên giữa đầu đề tiêu chuẩn /usr/include/crypt.h và các tệp đầu để được OpenSSL cung cấp.

Việc nâng cấp cài đặt OpenSSL của bạn lên phiên bản 0.9.6a sửa lỗi này. Solaris 9 và cao hơn có một phiên bản OpenSSL mới hơn.

15.8.7.3. configure kêu chương trình kiểm thử hỏng

Nếu configure kêu về một chương trình kiểm thử hỏng, thì điều này có thể là trường hợp trình liên kết thời gian chạy (run-time linker) đang không có khả năng tìm thấy một vài thư viện, có khả năng là libz, libreadline hoặc một vài thư viện phi tiêu chuẩn khác như libssl. Để chỉ nó tới vị trí đúng, hãy thiết lập biến môi trường LDFLAGS trong dòng lệnh configure, như,

```
configure ... LDFLAGS="-R /usr/sfw/lib:/opt/sfw/lib:/usr/local/lib"
```

Xem trang ld man để có thêm thông tin.

15.8.7.4. Xây dựng 64 bit đôi khi bị hỏng

Về Solaris 7 và cũ hơn, phiên bản 64 bit của libc có một thủ tục lỗi vsnprintf, nó dẫn tới hỏng lõi bất thường trong PostgreSQL. Cách khắc phục đơn giản nhất được biết là ép PostgreSQL sử dụng phiên

bản vsnprintf của riêng mình thay vì sao chép thư viện. Để làm điều này, sau khi bạn chạy configure hãy sửa tệp được configure tạo ra: Trong src/Makefile.global, hãy thay đổi dòng

```
LIBOBJS =
```

thành

```
LIBOBJS = snprintf.o
```

(Có thể có những tệp khác được liệt kê rồi trong biến này. Trật tự không thành vấn đề). Sau đó xây dựng như bình thường.

15.8.7.5. Biên dịch vì hiệu năng tối ưu

Trên kiến trúc SPARC, Sun Studio được khuyến cáo mạnh mẽ cho sự biên dịch. Hãy thử sử dụng cờ tối ưu -xO5 để sinh ra các nhị phân nhanh hơn đáng kể. Đừng sử dụng bất kỳ cờ nào mà sửa hành vi các hoạt động dấu chấm động và việc xử lý errno (nghĩa là, -fast). Các cờ đó có thể làm nảy sinh một số hành vi phi tiêu chuẩn của PostgreSQL, ví dụ, trong việc tính toán ngày tháng/thời gian.

Nếu bạn không có lý do để sử dụng các nhị phân 64 bit trên SPARC, hãy ưu tiên hơn cho phiên bản 32 bit. Các hoạt động của phiên bản 64 bit là chậm hơn và các nhị phân 64 bit là chậm hơn so với các phương án 32 bit. Và mặt khác, mã 32 bit trong họ AMD64 CPU là không bẩm sinh, và điều đó giải thích vì sao mã 32 bit là chậm hơn đáng kể trong họ CPU này.

Một vài mẹo cho việc tinh chỉnh PostgreSQL và Solaris về hiệu năng có thể thấy tại http://www.sun.com/servers/coolthreads/tnb/applications_postgresql.jsp. Bài viết này ban đầu tập trung vào nền tảng T2000, nhưng nhiều khuyến cáo cũng là hữu dụng trên các phần cứng khác với Solaris.

15.8.7.6. Sử dụng DTrace để lần vết PostgreSQL

Vâng, việc sử dụng DTrace là có khả năng. Xem Phần 27.4 để có thêm thông tin. Bạn cũng có thể thấy nhiều thông tin hơn trong: http://blogs.sun.com/robertlor/entry/user_level_dtrace_probes_in. Nếu bạn thấy liên kết thực thi postgres bỏ qua với một thông điệp lỗi giống như:

```
Undefined                       first referenced
symbol                          in file
AbortTransaction                utils/probes.o
CommitTransaction               utils/probes.o
ld: fatal: Symbol referencing errors. No output written to postgres
collect2: ld returned 1 exit status
gmake: *** [postgres] Error 1
```

thì cài đặt DTrace của bạn là quá cũ không thể điều khiển các thăm dò trong các hàm tĩnh được. Bạn cần Solaris 10u4 hoặc mới hơn.

Chương 16. Cài đặt từ mã nguồn trên Windows

Được khuyến cáo rằng hầu hết những người sử dụng tải về phân phối nhị phân cho Windows, sẵn có như một gói cài đặt một nháy từ website của PostgreSQL. Việc xây dựng từ nguồn chỉ có ý định cho những người phát triển PostgreSQL hoặc các mở rộng.

Có vài cách thức khác nhau về việc xây dựng PostgreSQL trên Windows. Cách đơn giản nhất để xây dựng với các công cụ của Microsoft là cài đặt một phiên bản được bộ SDK nền tảng của Microsoft (Microsoft Platform SDK) hỗ trợ và sử dụng trình biên dịch có trong đó. Có khả năng xây dựng với đầy đủ bộ Microsoft Visual C++ 2005 hoặc 2008. Trong một số trường hợp đòi hỏi cài đặt Platform SDK bổ sung thêm cho trình biên dịch đó.

Cũng có khả năng xây dựng PostgreSQL bằng việc sử dụng các công cụ trình biên dịch GNU được MinGW cung cấp, hoặc sử dụng Cygwin cho các phiên bản cũ hơn của Windows.

Cuối cùng, thư viện truy cập máy trạm (libpq) có thể được xây dựng bằng Visual C++ 7.1 hoặc Borland C++ vì tính tương thích với các ứng dụng được liên kết tĩnh được xây dựng bằng các công cụ đó.

Việc xây dựng có sử dụng MinGW hoặc Cygwin sử dụng hệ thống xây dựng thông thường, xem Chương 15 và các lưu ý đặc biệt trong Phần 15.8.5 và Phần 15.8.2. Các xây dựng đó không thể tạo các nhị phân 64 bit. Cygwin không được khuyến cáo và chỉ nên được sử dụng cho các phiên bản Windows nơi mà xây dựng bẩm sinh không làm việc, như Windows98. MinGW chỉ được khuyến cáo nếu bạn đang xây dựng các module khác có sử dụng nó. Các nhị phân chính thức được xây dựng bằng việc sử dụng Visual Studio.

16.1. Xây dựng với Visual C++ hoặc Platform SDK

PostgreSQL có thể được xây dựng bằng việc sử dụng bộ biên dịch Visual C++ từ Microsoft. Các trình biên dịch đó có thể hoặc từ Visual Studio, Visual Studio Express hoặc một số phiên bản của Platform SDK. Nếu bạn không có một môi trường Visual Studio được thiết lập rồi, thì cách dễ nhất cho chúng ta để sử dụng các trình biên dịch trong Platform SDK tải về tự do được từ Microsoft.

PostgreSQL hỗ trợ các trình biên dịch từ Visual Studio 2005 và Visual Studio 2008. Khi sử dụng chỉ Platform SDK, hoặc khi xây dựng cho Windows 64bit, chỉ Visual Studio 2008 được hỗ trợ. Visual Studio 2010 còn chưa được hỗ trợ.

Khi xây dựng bằng việc sử dụng Platform SDK, các phiên bản 6.0 đến 7.0 của SDK được hỗ trợ. Các phiên bản cũ hơn hoặc mới hơn sẽ không làm việc. Đặc biệt, các phiên bản từ 7.0a và sau này sẽ không làm việc, vì chúng bao gồm các trình biên dịch từ Visual Studio 2010.

Các công cụ cho việc xây dựng sử dụng Visual C++, là trong thư mục src/tools/msvc. Khi xây dựng, hãy chắc chắn không có công cụ nào từ MinGW hoặc Cygwin hiện diện trong PATH hệ thống của chúng ta. Hơn nữa, hãy chắc chắn bạn có tất cả các công cụ được yêu cầu đối với Visual C++ có sẵn trong PATH. Trong Visual Studio, hãy khởi động dấu nhắc lệnh của Visual Studio (Visual Studio Command Prompt). Trong Platform SDK, hãy khởi động CMD shell được liệt kê dưới SDK trong thực đơn khởi tạo (Start Menu). Nếu bạn mong muốn xây dựng một phiên bản 64 bit, bạn phải sử

dụng phiên bản lệnh 64 bit, và ngược lại. Tất cả các lệnh nên được chạy từ thư mục src\tools\msvc.

Trước khi bạn xây dựng, bạn có thể cần phải sửa tệp config.pl để phản ánh bất kỳ lựa chọn cấu hình nào bạn muốn thay đổi, hoặc các đường dẫn tới bất kỳ các thư viện để sử dụng nào của bên thứ 3. Cấu hình hoàn chỉnh được xác định bằng việc đọc và phân tích tệp config_default.pl, và sau đó áp dụng bất kỳ thay đổi nào từ config.pl. Ví dụ, để chỉ định vị trí cài đặt Python của bạn, hãy đặt thứ sau đây trong config.pl:

```
$config->{python} = 'c:\python26';
```

Bạn chỉ cần chỉ định các tham số mà khác với những gì trong config_default.pl.

Nếu bạn cần thiết lập bất kỳ biến môi trường nào, hãy tạo một tệp gọi là buildenv.pl và đặt các lệnh theo yêu cầu ở đó. Ví dụ, để thêm đường dẫn cho cụm bison khi nó không có trong PATH, hãy tạo một tệp chứa:

```
$ENV{PATH}=$ENV{PATH} . ';c:\some\where\bison\bin';
```

16.1.1. Yêu cầu

Các sản phẩm bổ sung sau đây được yêu cầu để xây dựng PostgreSQL. Hãy sử dụng tệp config.pl để chỉ định các thư mục nào các thư viện là sẵn sàng trong đó.

Microsoft Platform SDK

> Được khuyến cáo rằng bạn nâng cấp lên phiên bản có sẵn mới nhất của Microsoft Platform SDK, có sẵn để tải về từ http://www.microsoft.com/downloads/.
>
> Bạn phải luôn đưa Windows Headers (đầu đề Windows) và các thư viện (Libraries) thành một phần của SDK. Nếu bạn cài đặt Platform SDK có bao gồm các trình biên dịch Visual C++, thì bạn không cần Visual Studio để xây.

ActiveState Perl

> ActiveState Perl được yêu cầu để chạy các script tạo xây dựng. MinGW hoặc Cygwin Perl sẽ không làm việc. Nó cũng phải hiện diện trong PATH. Các nhị phân có thể được tải về từ http://www.activestate.com (Lưu ý: phiên bản 5.8 hoặc sau này được yêu cầu, Phân phối Tiêu chuẩn [Standard Distribution] tự do là đủ).

Các sản phẩm bổ sung sau đây không được yêu cầu để khởi tạo, nhưng được yêu cầu để xây dựng gói hoàn chỉnh. Hãy sử dụng tệp config.pl để chỉ định các thư mục nào các thư viện là sẵn sàng ở đó.

ActiveState TCL

> Được yêu cầu cho việc xây dựng PL/TCL (Lưu ý: phiên bản 8.4 được yêu cầu, Phân phối Tiêu chuẩn tự do là đủ).

Bison và Flex

> Bison và Flex được yêu cầu để xây dựng từ Git, nhưng không được yêu cầu khi xây dựng từ một tệp phát hành. Lưu ý là chỉ Bison 1.875 hoặc các phiên bản 2.2 và sau này sẽ làm việc được. Hơn nữa, Flex phiên bản 2.5.31 hoặc sau này được yêu cầu. Bison có thể tải về từ http://gnuwin32.sourceforge.net, còn Flex từ http://www.postgresql.org/ftp/misc/winflex/.
>
> **Lưu ý**: Phân phối Bison từ GnuWin32 dường như có một lỗi làm cho Bison hoạt

động không đúng khi được cài đặt trong một thư mục với các chỗ trống trong tên, như vị trí mặc định trong các cài đặt tiếng Anh C:\Program Files\GnuWin32. Hãy xem xét cài đặt trong C:\GnuWin32 thay vào đó.

Diff

Diff được yêu cầu để chạy các kiểm thử đệ quy, và có thể được tải về từ http://gnuwin32.sourceforge.net.

Gettext

Gettext được yêu cầu để xây dựng với sự hỗ trợ của NLS, và có thể được tải về từ http://gnuwin32.sourceforge.net. Lưu ý rằng các nhị phân, các phụ thuộc và các tệp của lập trình viên tất cả đều cần thiết.

MIT Kerberos

Được yêu cầu để hỗ trợ xác thực Kerberos. MIT Kerberos có thể được tải về từ http://web.mit.edu/Kerberos/dist/index.html.

libxml2 and libxslt

Được yêu cầu để hỗ trợ XML. Các nhị phân có thể được tải về từ http://zlatkovic.com/pub/libxml hoặc nguồn từ http://xmlsoft.org. Lưu ý rằng libxml2 đòi hỏi iconv, nó là sẵn sàng từ cùng y hệt vị trí tải về.

openssl

Được yêu cầu để hỗ trợ SSL. Các nhị phân có thể tải về từ http://www.slproweb.com/products/Win32OpenSSL.html hoặc nguồn từ http://www.openssl.org.

ossp-uuid

Được yêu cầu để hỗ trợ UUID-OSSP (chỉ contrib). Nguồn có thể được tải về từ http://www.ossp.org/pkg/lib/uuid/.

Python

Được yêu cầu để xây dựng PL/Python. Các nhị phân có thể tải về từ http://www.python.org.

zlib

Được yêu cầu để hỗ trợ toàn diện trong pg_dump và pg_restore. Các nhị phân có thể được tải về từ http://www.zlib.net.

16.1.2. Cân nhắc đặc thù cho Windows 64 bit

PostgreSQL sẽ chỉ xây dựng cho kiến trúc x64 trên Windows 64 bit, không có hỗ trợ cho các vi xử lý Itanium.

Việc pha trộn các phiên bản 32 bit và 64 bit trong cùng cây xây dựng không được hỗ trợ. Hệ thống xây dựng sẽ tự động dò tìm ra nếu nó đang chạy trong một môi trường 32 bit hoặc 64 bit, và xây dựng PostgreSQL một cách tương xứng. Vì lý do này, là quan trọng để bắt đầu làm đúng lời nhắc lệnh trước khi xây dựng.

Để sử dụng thư viện của bên thứ 3 ở phía máy chủ như python và openssl, thư viện này cũng phải là 64 bit. Không có hỗ trợ cho việc tải một thư viện 32 bit vào một máy chủ 64 bit. Vài trong số các thư viện của bên thứ 3 mà PostgreSQL hỗ trợ chỉ có thể sẵn sàng trong các phiên bản 32 bit, trong trường hợp đó chúng không thể được sử dụng với PostgreSQL 64 bit.

16.1.3. Xây dựng

Để xây dựng tất cả PostgreSQL theo cấu hình phiên bản (mặc định), hãy chạy lệnh:

```
build
```

Để xây dựng tất cả PostgreSQL theo cấu hình gỡ lỗi, chạy lệnh:

```
build DEBUG
```

Để xây dựng chỉ một dự án duy nhất, ví dụ psql, hãy chạy lệnh:

```
build psql
build DEBUG psql
```

Để thay đổi cấu hình xây dựng mặc định cho gỡ lỗi, hãy đặt thứ sau đây vào tệp buildenv.pl:

```
$ENV{CONFIG}="Debug";
```

Cũng có khả năng xây dựng từ bên trong Visual Studio GUI. Trong trường hợp này, bạn cần chạy:

```
perl mkvcbuild.pl
```

từ dấu nhắc lệnh, và sau đó mở pgsql.sln được tạo ra (trong thư mục gốc root của cây nguồn) trong Visual Studio.

16.1.4. Làm sạch và cài đặt

Hầu hết thời gian, việc dõi theo sự phụ thuộc tự động trong Visual Studio sẽ điều khiển các tệp được thay đổi. Nhưng nếu từng có những thay đổi lớn, thì bạn có thể cần làm sạch sự cài đặt. Để làm điều này, đơn giản hãy chạy lệnh clean.bat, nó sẽ tự động làm sạch tất cả các tệp được tạo ra. Bạn cũng có thể chạy nó với tham số dist, trong trường hợp đó nó sẽ hành xử giống như make distclean và cũng loại bỏ các tệp đầu ra flex/bison.

Mặc định, tất cả các tệp được viết trong một thư mục con của các thư mục debug hoặc release. Để cài đặt các tệp đó bằng việc sử dụng sắp đặt bố trí tiêu chuẩn, và cũng tạo ra các tệp được yêu cầu để khởi tạo và sử dụng cơ sở dữ liệu, hãy chạy lệnh:

```
install c:\destination\directory
```

16.1.5. Chạy các kiểm thử đệ quy

Để chạy các kiểm thử đệ quy, hãy chắc chắn bạn đã hoàn tất xây dựng tất cả các phần theo yêu cầu trước. Hơn nữa, hãy chắc chắn rằng các DLL được yêu cầu để tải tất cả các phần của hệ thống (như các DLL của Perl và Python cho các ngôn ngữ thủ tục) đang hiện diện trong đường dẫn hệ thống. Nếu chúng không phải thế, hãy thiết lập nó qua tệp buildenv.pl. Để chạy các kiểm thử đó, hãy chạy một trong những lệnh sau từ thư mục src\tools\msvc:

```
vcregress check
vcregress installcheck
vcregress plcheck
vcregress contribcheck
```

Để thay đổi lịch trình được sử dụng (mặc định là song song), hãy nối nó vào dòng lệnh như:

```
vcregress check serial
```

Để có thêm thông tin về kiểm thử đệ quy, hãy xem Chương 30.

16.1.6. Xây dựng tài liệu

Việc xây dựng tài liệu PostgreSQL ở định dạng HTML đòi hỏi vài công cụ và tệp. Hãy tạo một thư mục gốc root cho tất cả các tệp đó, và lưu trữ chúng trong các thư mục con trong danh sách ở dưới:

OpenJade

Tải về từ http://sourceforge.net/projects/openjade/files/openjade/1.3.1/openjade-1_3_1-2-bin.zip/download và giải nén vào thư mục `openjade-1.3.1`.

DocBook DTD 4.2

Tải về từ http://www.oasis-open.org/docbook/sgml/4.2/docbook-4.2.zip và giải nén trong thư mục `docbook`.

DocBook DTD 1.79

Tải về từ http://sourceforge.net/projects/docbook/files/docbookdsssl/1.79/docbook-dsssl-1.79.zip/download và giải nén trong thư mục `docbook-dsssl-1.79`.

Các thực thể ký tự ISO

Tải về từ http://www.oasis-open.org/cover/ISOEnts.zip và giải nén trong thư mục con `docbook`.

Hãy sửa tệp `buildenv.pl`, và thêm một biến cho vị trí của thư mục gốc root, ví dụ:

```
$ENV{DOCROOT}='c:\docbook';
```

Để xây dựng tài liệu, hãy chạy lệnh `builddoc.bat`. Lưu ý rằng điều này thực sự sẽ chạy xây dựng 2 lần, để tạo ra các chỉ số (index). Các tệp HTML được tạo ra sẽ nằm trong `doc\src\sgml`.

16.2. Xây dựng libpq với Visual C++ hoặc Borland C++

Việc sử dụng Visual C++ 7.1 - 9.0 hoặc Borland C++ để xây dựng `libpq` chỉ được khuyến cáo nếu bạn cần một phiên bản với các cờ gỡ lỗi/phiên bản khác (debug/release) nhau, hoặc nếu bạn cần một thư viện tĩnh để liên kết trong một ứng dụng. Để sử dụng bình thường phương pháp MinGW hoặc Visual Studio hoặc Platform SDK được khuyến cáo.

Để xây dựng thư viện máy trạm `libpq` bằng Visual Studio 7.1 hoặc sau này, hãy thay đổi thành thư mục `src` và gõ lệnh:

```
nmake /f win32.mak
```

Để xây dựng một phiên bản 64 bit của thư viện máy trạm `libpq` bằng Visual Studio 8.0 hoặc sau này, hãy thay đổi thành thư mục `src` và gõ lệnh:

```
nmake /f win32.mak CPU=AMD64
```

Xem tệp `win32.mak` để có thêm thông tin về các biến được hỗ trợ.

Để xây dựng thư viện máy trạm `libpq` với Borland C++, hãy thay đổi thành thư mục `src` và gõ lệnh:

```
make -N -DCFG=Release /f bcc32.mak
```

16.2.1. Tệp được tạo ra

Các tệp sau sẽ được xây dựng:

interfaces\libpq\Release\libpq.dll

Thư viện mặt tiền có khả năng liên kết động

interfaces\libpq\Release\libpqdll.lib

Thư viện nhập để liên kết các chương trình của bạn với libpq.dll

interfaces\libpq\Release\libpq.lib

Phiên bản tĩnh của thư viện mặt tiền

Bình thường bạn không cần cài đặt bất kỳ tệp máy trạm nào. Bạn nên đặt tệp libpq.dll vào cùng thư mục như các ứng dụng của bạn thực thi tệp. Đừng cài đặt libpq.dll vào thư mục Windows, System hoặc System32 của bạn trừ phi tuyệt đối cần thiết. Nếu tệp này được cài đặt bằng việc sử dụng một chương trình thiết lập, thì nó sẽ được cài đặt với việc kiểm tra phiên bản bằng việc sử dụng tài nguyên VERSIONINFO được đưa vào trong tệp đó, để đảm bảo rằng một phiên bản mới hơn của thư viện đó không bị ghi đè.

Nếu bạn đang lên kế hoạch để tiến hành phát triển bằng việc sử dụng libpq trên máy này, bạn sẽ phải thêm các thư mục con src\include và src\interfaces\libpq của cây nguồn vào đường dẫn đưa vào trong các thiết lập trình biên dịch của bạn.

Để sử dụng thư viện đó, bạn phải thêm tệp libpqdll.lib vào dự án của bạn. (Trong Visual C++, chỉ cần nháy chuột phải vào dự án và chọn để thêm nó).

Chương 17. Thiết lập và vận hành máy chủ

Chương này thảo luận cách thiết lập và chạy máy chủ cơ sở dữ liệu và các tương tác của nó với hệ điều hành.

17.1. Tài khoản người sử dụng PostgreSQL

Như với bất kỳ daemon máy chủ nào có khả năng truy cập được tới thế giới bên ngoài, được khuyến cáo chạy PostgreSQL theo một tài khoản người sử dụng riêng rẽ. Tài khoản người sử dụng này chỉ nên nắm các dữ liệu mà được máy chủ đó quản lý, và nên không được chia sẻ với các daemon khác. (Ví dụ, bằng việc sử dụng người sử dụng nobody là một ý tưởng tồi). Không được khuyến cáo cài đặt các tệp thực thi được người sử dụng này sở hữu vì các máy bị tổn thương sau đó có thể sửa đổi các nhị phân của riêng chúng.

Để thêm một tài khoản người sử dụng Unix vào hệ thống của bạn, hãy tìm kiếm một lệnh useradd hoặc adduser. Tên người sử dụng là postgres thường được sử dụng, và được giả thiết qua cuốn sách này, nhưng bạn có thể sử dụng tên khác nếu bạn thích.

17.2. Tạo một cụm cơ sở dữ liệu

Trước khi bạn có thể làm bất kỳ điều gì, bạn phải khởi tạo một vùng lưu trữ cơ sở dữ liệu trên đĩa. Chúng tôi gọi điều này là một cụm cơ sở dữ liệu. (SQL sử dụng khái niệm cụm catalog). Một cụm cơ sở dữ liệu là một bộ sưu tập các cơ sở dữ liệu được một cài đặt duy nhất của một máy chủ cơ sở dữ liệu đang chạy quản lý. Sau khi khởi tạo, một cụm cơ sở dữ liệu sẽ có một cơ sở dữ liệu có tên là postgres, nó có nghĩa như là một cơ sở dữ liệu mặc định để sử dụng từ các tiện ích, những người sử dụng và các ứng dụng của các bên thứ 3. Bản thân máy chủ cơ sở dữ liệu không đòi hỏi cơ sở dữ liệu postgres phải tồn tại, nhưng nhiều chương trình tiện ích bên ngoài giả thiết nó tồn tại. Một cơ sở dữ liệu khác được tạo ra trong từng cụm trong quá trình khởi tạo được gọi là template1. Như cái tên của nó gợi ý, điều này sẽ được sử dụng như một mẫu template cho các cơ sở dữ liệu tiếp sau được tạo ra; nó không nên được sử dụng cho công việc thực sự. (Xem Chương 21 để có thông tin về việc tạo ra các cơ sở dữ liệu mới bên trong một cụm).

Trong các khái niệm hệ thống tệp, một cụm cơ sở dữ liệu sẽ là một thư mục duy nhất dưới đó tất cả các dữ liệu sẽ được lưu trữ. Chúng tôi gọi điều này là thư mục dữ liệu hoặc vùng dữ liệu. Hoàn toàn tùy bạn nơi nào bạn chọn lưu trữ các dữ liệu của bạn. Không có mặc định, dù các vị trí như /usr/local/pgsql/data hoặc /var/lib/pgsql/data là phổ biến. Để khởi tạo một cụm cơ sở dữ liệu, hãy sử dụng lệnh initdb, nó được cài đặt với PostgreSQL. Vị trí hệ thống tệp mong muốn của cụm cơ sở dữ liệu của bạn được lựa chọn -D chỉ ra, ví dụ:

```
$ initdb -D /usr/local/pgsql/data
```

Lưu ý rằng bạn phải thực thi lệnh này trong khi đã đăng nhập vào tài khoản người sử dụng PostgreSQL, nó được mô tả trong phần trước.

Mẹo: Như là lựa chọn thay thế cho lựa chọn -D, bạn có thể thiết lập biến môi trường PGDATA.

Như một lựa chọn thay thế, bạn có thể chạy initdb thông qua chương trình pg_ctl như sau:

```
$ pg_ctl -D /usr/local/pgsql/data initdb
```

Điều này có thể sẽ trực quan hơn nếu bạn đang sử dụng pg_ctl cho việc khởi động và dừng máy chủ (xem Phần 17.3), sao cho pg_ctl có thể là lệnh duy nhất bạn sử dụng cho việc quản lý cài đặt máy chủ cơ sở dữ liệu đó.

initdb sẽ cố tạo thư mục bạn chỉ định nếu nó không luôn tồn tại. Có khả năng là nó sẽ không có quyền để làm thế (nếu bạn đi theo khuyến cáo của chúng tôi và tạo ra một tài khoản không có các quyền ưu tiên). Trong trường hợp đó bạn nên tạo cho bạn thư mục (như gốc root) và thay đổi chủ sở hữu để trở thành người sử dụng PostgreSQL. Đây là cách mà điều này có thể được thực hiện:

```
root# mkdir /usr/local/pgsql/data
root# chown postgres /usr/local/pgsql/data
root# su postgres
postgres$ initdb -D /usr/local/pgsql/data
```

initdb sẽ từ chối chạy nếu thư mục dữ liệu trông giống như nó đã từng được khởi tạo.

Vì thư mục dữ liệu chứa tất cả các dữ liệu được lưu trữ trong cơ sở dữ liệu, là cơ bản rằng nó sẽ được đảm bảo an toàn từ sự truy cập không có quyền. initdb vì thế viện gọi các quyền truy cập từ từng người ngoại trừ người sử dụng PostgreSQL.

Dù vậy, trong khi các nội dung thư mục là an toàn, thì thiết lập xác thực máy trạm mặc định cho sẽ phép bất kỳ người sử dụng cục bộ nào kết nối tới cơ sở dữ liệu và thậm chí trở thành siêu người sử dụng (superuser) của cơ sở dữ liệu. Nếu bạn không tin cậy những người sử dụng cục bộ khác, thì chúng tôi khuyến cáo bạn sử dụng một trong các lựa chọn như initdb's -W, --pwprompt hoặc --pwfile để chỉ định một mật khẩu cho siêu người sử dụng cơ sở dữ liệu. Hơn nữa, hãy chỉ định -A md5 hoặc -A password sao cho chế độ xác thực trust mặc định không được sử dụng; hoặc sửa tệp pg_hba.conf được tạo ra sau khi chạy initdb, nhưng trước khi bạn khởi động máy chủ lần đầu tiên. (Các tiếp cận hợp lý khác bao gồm việc sử dụng xác thực ident hoặc các quyền hệ thống tệp để hạn chế các kết nối. Xem Chương 19 để có thêm thông tin).

initdb cũng khởi tạo bản địa mặc định cho cụm cơ sở dữ liệu. Thông thường, nó sẽ chỉ lấy các thiết lập bản địa trong môi trường và áp dụng chúng cho cơ sở dữ liệu được khởi tạo. Có khả năng chỉ định một bản địa khác cho cơ sở dữ liệu; nhiều thông tin hơn về điều đó có thể thấy trong Phần 22.1. Trật tự sắp xếp mặc định được sử dụng trong cụm cơ sở dữ liệu đặc biệt được initdb thiết lập, và trong khi bạn có thể tạo các cơ sở dữ liệu mới bằng việc sử dụng trật tự sắp xếp khác, thì trật tự được sử dụng trong các cơ sở dữ liệu mẫu template mà initdb tạo ra không thể bị thay đổi mà không bỏ và tạo lại chúng. Cũng có một tác động về hiệu năng cho việc sử dụng các bản địa khác so với C và POSIX. Vì thế, là quan trọng để làm cho sự lựa chọn này đúng đắn ngay từ lần đầu tiên.

initdb cũng thiết lập việc mã hóa tập hợp ký tự mặc định cho cụm cơ sở dữ liệu. Thông thường điều này nên được chọn để phù hợp với việc thiết lập bản địa. Xem Phần 22.2 để có thêm chi tiết.

17.2.1. Hệ thống tệp mạng

Nhiều cài đặt tạo ra các cụm cơ sở dữ liệu trong các hệ thống tệp mạng. Đôi khi điều này được thực hiện trực tiếp thông qua NFS, hoặc bằng việc sử dụng một thiết bị Lưu trữ Được gắn vào Mạng - NAS (Network Attached Storage) mà sử dụng NFS bên trong. PostgreSQL không làm gì đặc biệt cho các hệ thống tệp NFS, nghĩa là nó giả thiết NFS hành xử chính xác như các ổ đĩa được kết nối

cục bộ - DAS (Direct Attached Storage). Nếu các triển khai NFS máy trạm và máy chủ không có ngữ nghĩa tiêu chuẩn, thì điều này có thể gây ra các vấn đề về độ tin cậy (xem http://www.time-travellers.org/shane/papers/NFS_considered_harmful.html). Đặc biệt, việc ghi có sự trễ (không đồng bộ) vào máy chủ NFS có thể gây ra các vấn đề về độ tin cậy; nếu có khả năng, hãy kích hoạt (mount) các hệ thống tệp NFS một cách đồng bộ (không có việc tạo bộ nhớ tạm thời - caching) để tránh điều này. Hơn nữa, việc kích hoạt mềm NFS không được khuyến cáo. (Các Mạng Cục bộ Lưu trữ - SAN [Storage Area Network]) sử dụng một giao thức truyền thông mức thấp hơn là NFS).

17.3. Khởi tạo máy chủ cơ sở dữ liệu

Trước khi bất kỳ ai có thể truy cập được cơ sở dữ liệu, bạn phải khởi động máy chủ cơ sở dữ liệu. Chương trình máy chủ cơ sở dữ liệu được gọi là postgres. Chương trình postgres phải biết nơi đâu để tìm dữ liệu mà nó được hỗ trợ để sử dụng. Điều này được thực hiện với lựa chọn -D. Vì thế, cách đơn giản nhất để khởi động máy chủ là:

```
$ postgres -D /usr/local/pgsql/data
```

điều sẽ để máy chủ chạy ở phần mặt tiền (foreground). Điều này phải được thực hiện trong khi đã đăng nhập vào tài khoản người sử dụng PostgreSQL. Không có -D, máy chủ sẽ cố sử dụng thư mục dữ liệu được biến môi trường PGDATA đặt tên. Nếu biến đó cũng không được cung cấp, thì sẽ hỏng.

Thông thường là tốt hơn để khởi động postgres ở phần phụ trợ (background). Đối với điều này, hãy sử dụng cú pháp shell thông thường của Unix:

```
$ postgres -D /usr/local/pgsql/data >logfile 2>&1 &
```

Là quan trọng để lưu trữ đầu ra stdout và stderr của máy chủ ở đâu đó, như được nêu ở trên. Nó sẽ giúp kiểm toán các mục đích và chuẩn đoán các vấn đề. (Xem Phần 23.3 cho một thỏa luận tỉ mỉ hơn về việc điều khiển tệp lưu ký).

Chương trình postgres cũng lấy một số lựa chọn dòng lệnh khác. Để có thêm thông tin, hãy xem trang tham chiếu postgres và Chương 18 bên dưới.

Cú pháp shell này có thể trở thành nặng nề nhanh chóng. Vì thế chương trình của trình đóng gói pg_ctl được cung cấp để đơn giản hóa một số tác vụ. Ví dụ:

```
pg_ctl start -l logfile
```

sẽ khởi động máy chủ ở phần phụ trợ (background) và đặt đầu ra vào tệp lưu ký được đặt tên. Lựa chọn -D có cùng y hệt ý nghĩa ở đây như đối với postgres. pg_ctl cũng có khả năng dùng máy chủ đó.

Thông thường, bạn sẽ muốn khởi động máy chủ cơ sở dữ liệu khi máy tính khởi động. Các script tự khởi động (Autostart) là đặc thù theo hệ điều hành. Có một ít được phân tán với PostgreSQL trong thư mục contrib/start-scripts. Việc cài đặt một script sẽ yêu cầu các quyền ưu tiên của gốc root.

Các hệ thống khác nhau có các qui ước khác nhau cho việc khởi động các daemons vào thời điểm khởi động. Nhiều hệ thống có một tệp /etc/rc.local or /etc/rc.d/rc.local. Các hệ thống khác sử dụng các thư mục rc.d. Bất kể điều gì bạn làm, máy chủ phải được chạy từ tài khoản người sử dụng PostgreSQL và không phải là từ gốc root hoặc từ bất kỳ người sử dụng nào khác. Vì thế bạn có lẽ nên hình thành các lệnh của bạn bằng việc sử dụng su postgres -c '...'. Ví dụ:

```
su postgres -c 'pg_ctl start -D /usr/local/pgsql/data -l serverlog'
```

Đây là một ít các gợi ý đặc thù hệ điều hành. (Trong từng trường hợp hãy chắc chắn sử dụng thư mục cài đặt và tên người sử dụng phù hợp khi chúng ta chỉ ra các giá trị chung).

- Đối với FreeBSD, hãy nhìn vào tệp contrib/start-scripts/freebsd trong phân phối nguồn của PostgreSQL.
- Trong OpenBSD, hãy thêm các dòng sau vào tệp /etc/rc.local:

```
if      [ -x /usr/local/pgsql/bin/pg_ctl -a -x /usr/local/pgsql/bin/postgres ]; then
        su -l postgres -c '/usr/local/pgsql/bin/pg_ctl start -s -l /var/postgresql/log -D /
        echo -n ' postgresql'
fi
```

- Trên các hệ thống Linux, hoặc thêm:

  ```
  /usr/local/pgsql/bin/pg_ctl start -l logfile -D /usr/local/pgsql/data
  ```

  vào /etc/rc.d/rc.local hoặc xem xét tệp contrib/start-scripts/linux trong phân phối nguồn của PostgreSQL.
- Trong NetBSD, hãy sử dụng hoặc các scripts khởi động của FreeBSD hoặc Linux, phụ thuộc vào sự ưu tiên.
- Trong Solaris, hãy tạo một tệp gọi là /etc/init.d/postgresql mà có chứa dòng sau đây:

  ```
  su - postgres -c "/usr/local/pgsql/bin/pg_ctl start -l logfile -D /usr/local/pgsql/data"
  ```

  Rồi thì, tạo một liên kết biểu tượng cho nó trong /etc/rc3.d as S99postgresql.

Trong khi máy chủ đang chạy, thì PID của nó được lưu trữ trong tệp postmaster.pid trong thư mục dữ liệu. Điều này được sử dụng để ngăn chặn nhiều cài đặt máy chủ khỏi việc chạy trong cùng y hệt thư mục dữ liệu và cũng có thể được sử dụng cho việc tắt máy chủ.

17.3.1. Hỏng khởi động máy chủ

Có vài lý do chung máy chủ có thể hỏng lúc khởi động. Hãy kiểm tra tệp lưu ký của máy chủ, hoặc khởi động nó bằng tay (không có tái định tuyến đầu ra tiêu chuẩn hoặc lỗi tiêu chuẩn) và xem thông điệp lỗi nào xuất hiện. Bên dưới chúng ta giải thích một số thông điệp lỗi phổ biến chi tiết hơn.

```
LOG: could not bind IPv4 socket: Address already in use
HINT: Is another postmaster already running on port 5432? If not, wait a few seconds and
FATAL: could not create TCP/IP listen socket
```

Điều này thường có nghĩa đúng như những gì nó gợi ý: bạn đã cố khởi động một máy chủ khác ở cùng một cổng nơi mà cổng đó đang chạy rồi. Tuy nhiên, nếu thông điệp lỗi của nhân không phải là Address already in use (Địa chỉ đang được sử dụng rồi) hoặc một vài phương án của lỗi này, thì có lẽ sẽ là một vấn đề khác. Ví dụ, việc cố gắng khởi động một máy chủ trên một số cổng được giữ lại có thể dẫn tới thứ gì đó như thế này:

```
$ postgres -p 666
LOG: could not bind IPv4 socket: Permission denied
HINT: Is another postmaster already running on port 666? If not, wait a few seconds and
FATAL: could not create TCP/IP listen socket
```

Một thông điệp giống như:

```
FATAL: could not create shared memory segment: Invalid argument
DETAIL: Failed system call was shmget(key=5440001, size=4011376640, 03600).
```

có thể có nghĩa là giới hạn nhân của bạn về kích thước bộ nhớ được chia sẻ là nhỏ hơn so với khu

vực làm việc mà PostgreSQL đang cố tạo ra (4011376640 byte trong ví dụ này). Hoặc nó có thể có nghĩa rằng bạn hoàn toàn không có sự hỗ trợ bộ nhớ được chia sẻ kiểu System-V được thiết lập cấu hình trong nhân của bạn. Như một sự khắc phục tạm thời, bạn có thể cố khởi động máy chủ với một số bộ nhớ tạm buffer (shared_buffers) nhỏ hơn bình thường. Cuối cùng bạn sẽ muốn tái thiết lập cấu hình cho nhân của bạn để làm gia tăng kích thước bộ nhớ chia sẻ được phép. Bạn cũng có thể thấy thông điệp này khi cố gắng khởi động nhiều máy chủ trên cùng một máy tính, nếu tổng không gian của chúng được yêu cầu vượt quá giới hạn của nhân.

Một lỗi giống như:

```
FATAL: could not create semaphores: No space left on device
DETAIL: Failed system call was semget(5440126, 17, 03600).
```

không có nghĩa là bạn đã dùng hết không gian đĩa. Nó có nghĩa là giới hạn nhân của bạn về số lượng các sự ra hiệu của System V là nhỏ hơn so với số lượng mà PostgreSQL muốn tạo ra. Như ở trên, bạn có thể có khả năng khắc phục vấn đề đó bằng việc khởi tạo máy chủ bằng một số lượng được giảm các kết nối được phép (max_connections), nhưng cuối cùng bạn sẽ muốn làm tăng giới hạn của nhân.

Nếu bạn có một lỗi "illegal system call" (lời gọi hệ thống không hợp lệ), thì có khả năng là bộ nhớ được chia sẻ hoặc các sự ra hiệu hoàn toàn không được hỗ trợ trong nhân của bạn. Trong trường hợp đó lựa chọn duy nhất của bạn là tái thiết lập cấu hình cho nhân để kích hoạt các tính năng đó.

Các chi tiết về việc thiết lập cấu hình cho các phương tiện dễ dàng của System V IPC được đưa ra trong Phần 17.4.1.

17.3.2. Các vấn đề kết nối máy trạm

Dù các điều kiện lỗi có khả năng ở phía máy trạm là hoàn toàn khác nhau và phụ thuộc vào ứng dụng, thì một ít trong số chúng có thể có liên quan trực tiếp tới cách mà máy chủ đã được khởi động. Các điều kiện khác so với chúng được chỉ ra bên dưới sẽ được làm thành tài liệu với ứng dụng máy trạm tương ứng.

```
psql:   could not connect to server: Connection refused
        Is the server running on host "server.joe.com" and accepting
        TCP/IP connections on port 5432?
```

Đây là hỏng hóc chung "I couldn't find a server to talk to" (Tôi không thể tìm thấy máy chủ để nói với nó). Nó trông giống như ở trên khi giao tiếp TCP/IP có ý định. Một sai lầm chung là quên thiết lập cấu hình cho máy chủ để cho phép các kết nối TCP/IP.

Như một lựa chọn, bạn sẽ có điều này khi cố giao tiếp socket miền Unix với một máy chủ cục bộ:

```
psql:   could not connect to server: No such file or directory
        Is the server running locally and accepting
        connections on Unix domain socket "/tmp/.s.PGSQL.5432"?
```

Dòng cuối cùng là hữu dụng trong việc kiểm tra hợp lệ rằng máy trạm đang cố gắng kết nối tới đúng vị trí. Nếu trong thực tế không có máy chủ chạy ở đó, thì thông điệp lỗi nhân điển hình sẽ hoặc là Connection refused (Kết nối bị từ chối) hoặc No such file or directory (Không có tệp hoặc thư mục như vậy), như được minh họa. (Là quan trọng để nhận thức được rằng Connection refused trong ngữ cảnh này không có nghĩa là máy chủ có yêu cầu kết nối của bạn và đã từ chối nó. Trường hợp

đó sẽ tạo ra một thông điệp khác, như được chỉ ra trong Phần 19.4). Các thông điệp lỗi khác như Connection timed out (Hết thời gian kết nối) có thể chỉ ra các vấn đề cơ bản hơn, như thiếu tính kết nối mạng.

17.4. Quản lý các tài nguyên của nhân

Một cài đặt lớn PostgreSQL có thể nhanh chóng làm kiệt quệ các giới hạn tài nguyên hệ điều hành khác nhau. (Trên một số hệ thống, các mặc định tài nguyên là thấp tới mức bạn thậm chí không cần một cài đặt "lớn"). Nếu bạn đã gặp dạng vấn đề này rồi, hãy đọc tiếp.

17.4.1. Bộ nhớ được chia sẻ và các sự ra hiệu

Bộ nhớ được chia sẻ và các sự ra hiệu được hợp tác tham chiếu tới như là "System V IPC" (cùng với các hàng đợi thông điệp, chúng là không phù hợp cho PostgreSQL). Hầu hết tất cả các hệ điều hành hiện đại đưa ra các tính năng đó, nhưng nhiều trong số chúng không bắt phải bật chúng hoặc có kích cỡ đủ theo mặc định, đặc biệt như RAM sẵn sàng và các đòi hỏi các ứng dụng cơ sở dữ liệu gia tăng. (trên Windows, PostgreSQL đưa ra sự triển khai thay thế của riêng nó đối với các phương tiện đó, vì thế hầu hết phần này có thể bị xem nhẹ).

Sự thiếu hụt hoàn toàn các phương tiện đó thường được nêu với một lỗi lời gọi hệ thống không hợp pháp khi máy chủ khởi động. Trong trường hợp đó không có lựa chọn thay thế nhưng để thiết lập cấu hình cho nhân của bạn. PostgreSQL sẽ không làm việc mà không có chúng. Tuy nhiên, tình huống này là hiếm trong các hệ điều hành hiện đại.

Khi PostgreSQL vượt quá một trong những giới hạn IPC cứng khác nhau, thì máy chủ sẽ từ chối khởi động và sẽ để lại một thông điệp lỗi chỉ định, mô tả vấn đề đó và cách làm gì với nó. (Xem Phần 17.3.1). Các tham số nhân phù hợp được đặt tên nhất quán khắp các hệ thống khác nhau; Bảng 17-1 đưa ra một tổng quan. Tuy nhiên, các phương pháp để thiết đặt chúng là khác nhau. Các gợi ý cho một số nền tảng được đưa ra bên dưới.

Bảng 17-1. Các tham số System V IPC

Tên	Mô tả	Các giá trị hợp lý
SHMMAX	Kích thước tối đa đoạn bộ nhớ được chia sẻ (byte)	ít nhất vài MB (xem văn bản)
SHMMIN	Kích thước tối đa đoạn bộ nhớ được chia sẻ (byte)	1
SHMALL	Tổng lượng bộ nhớ được chia sẻ sẵn sàng (byte hoặc trang)	nếu là byte, y hệt như SHMMAX; nếu là trang, ceil(SHMMAX/PAGE_SIZE)
SHMSEG	Số tối đa các đoạn bộ nhớ được chia sẻ theo từng tiến trình	chỉ 1 đoạn là cần thiết, nhưng mặc định là cao hơn nhiều
SHMMNI	Số tối đa các đoạn bộ nhớ được chia sẻ toàn hệ thống	giống SHMSEG cộng thêm chỗ cho các ứng dụng khác
SEMMNI	Số tối đa các mã định danh sự ra hiệu (như, các tập hợp)	ít nhất ceil((max_connections + autovacuum_max_workers + 4) / 16)
SEMMNS	Số tối đa các sự ra hiệu toàn hệ thống	ceil((max_connections + autovacuum_max_workers + 4) / 16) * 17 cộng thêm chỗ cho các ứng dụng khác
SEMMSL	Số tối đa các sự ra hiệu cho từng tập hợp	ít nhất 17
SEMMAP	Số lượng khoản đầu vào trong bản đồ sự ra hiệu	xem văn bản

Tên	Mô tả	Các giá trị hợp lý
SEMVMX	Giá trị tối đa của sự ra hiệu	ít nhất 1000 (Mặc định thường là 32767; không thay đổi trừ phi cần thiết)

Tham số bộ nhớ được chia sẻ quan trọng nhất là SHMMAX, kích cỡ cực đại, theo byte, của một đoạn bộ nhớ được chia sẻ. Nếu bạn có một thông điệp lỗi từ shmget giống như "Invalid argument" (Đối số không hợp lệ), có khả năng là giới hạn này đã bị vượt quá. Kích cỡ của đoạn bộ nhớ được chia sẻ được yêu cầu thường khác nhau phụ thuộc vào vài tham số cấu hình của PostgreSQL, như được chỉ ra trong Bảng 17-2. (Bất kỳ thông điệp lỗi nào bạn có thể có được sẽ bao gồm kích cỡ chính xác của yêu cầu phân bổ bị hỏng). Như một giải pháp tạm thời, bạn có thể hạ thấp một số thiết lập đó để tránh hỏng. Trong khi có khả năng để làm cho PostgreSQL chạy được với SHMMAX nhỏ hơn 2MB, thì bạn cần cân nhắc nhiều hơn vì hiệu năng chấp nhận được. Các thiết lập mong muốn là cỡ hàng trăm MB cho tới vài GB.

Một số hệ thống cũng có một giới hạn về tổng lượng bộ nhớ được chia sẻ trong hệ thống đó (SHMALL). Phải chắc chắn điều này là đủ lớn cho PostgreSQL cộng với bất kỳ ứng dụng nào khác mà đang sử dụng các đoạn bộ nhớ được chia sẻ. Lưu ý rằng SHMALL được đo đếm theo các trang thay vì theo các byte trong nhiều hệ thống.

Ít có khả năng gây ra các vấn đề là kích cỡ tối thiểu cho các đoạn bộ nhớ được chia sẻ (SHMMIN), nó sẽ nằm vào khoảng 500 KB cho PostgreSQL (thường chỉ là 1). Số lượng tối đa các đoạn cho toàn bộ hệ thống (SHMMNI) hoặc theo từng tiến trình (SHMSEG) có lẽ không gây ra vấn đề trừ phi hệ thống của bạn nhờ chúng thiết lập về 0.

PostgreSQL sử dụng một sự ra hiệu cho từng kết nối được phép (max_connections) và tiến trình của trình công việc (worker) tự động hút chân không được phép (autovacuum_max_workers), trong các tập hợp 16. Từng tập hợp như vậy cũng sẽ có một sự ra hiệu thứ 17 có chứa một "số ma thuật" (magic number), dò tìm sự va chạm với các tập hợp sự ra hiệu được các ứng dụng khác sử dụng. Số lượng tối đa các sự ra hiệu trong hệ thống được SEMMNS thiết lập, nó sau đó phải ít nhất là cao như max_connections cộng với autovacuum_max_workers, cộng với một sự dư thừa cho từng trong số 16 kết nối được phép cộng với các trình công việc (xem công thức trong Bảng 17-1). Tham số SEMMNI xác định giới hạn về số lượng các tập hợp sự ra hiệu có thể tồn tại trong hệ thống ở một thời điểm. Vì thế tham số này phải là ít nhất ceil((max_connections + autovacuum_max_workers + 4) / 16). Việc hạ thấp số lượng các kết nối được phép là một sự khắc phục tạm thời cho các hỏng hóc, chúng thường được nêu một cách lẫn lộn "Không còn chỗ trong thiết bị" (No space left on device), từ hàm semget.

Trong một số trường hợp cũng có thể cần thiết để tăng SEMMAP tới ít nhất theo trình tự của SEMMNS.

Tham số này xác định kích cỡ của bản đồ tài nguyên sự ra hiệu, trong đó từng khối kề nhau của các sự ra hiệu có sẵn cần một đầu vào. Khi một tập hợp sự ra hiệu được giải phóng thì hoặc là nó được bổ sung cho một đầu vào đang tồn tại nằm cạnh khối được giải phóng đó hoặc nó được đăng ký dưới một đầu vào bản đồ mới. Nếu bản đồ đó đầy, thì các sự ra hiệu được giải phóng đó sẽ mất (cho tới khi được khởi động lại). Sự phân mảnh không gian ra hiệu có thể qua thời gian dẫn tới ít hơn các

sự ra hiệu có sẵn so với nó đáng phải có.

Tham số SEMMSL xác định có bao nhiêu sự ra hiệu có thể có trong một tập hợp, phải ít nhất là 17 đối với PostgreSQL.

Các thiết lập khác nhau khác có liên quan tới "hoãn sự ra hiệu" (semaphore undo), như SEMMNU và SEMUME, không ảnh hưởng tới PostgreSQL.

AIX

Ít nhất như trong phiên bản 5.1, là không nhất thiết phải làm bất kỳ cấu hình đặc biệt nào cho các tham số như vậy như SHMMAX, khi dường như điều này được thiết lập cấu hình để cho phép tất cả bộ nhớ sẽ được sử dụng như bộ nhớ được chia sẻ. Đó là dạng cấu hình được sử dụng phổ biến cho các cơ sở dữ liệu khác như DB/2.

Tuy nhiên, có lẽ sẽ là cần thiết để sửa đổi thông tin ulimit tổng thể trong /etc/security/limits, như các giới hạn cứng mặc định cho các kích cỡ tệp (fsize) và số lượng các tệp (nofiles) có lẽ là quá thấp.

BSD/OS

Bộ nhớ được chia sẻ. Mặc định, chỉ 4 MB bộ nhớ được chia sẻ được hỗ trợ. Hãy nhớ trong đầu rằng bộ nhớ được chia sẻ không theo trang; nó bị khóa trong RAM. Để tăng lượng bộ nhớ được chia sẻ được hệ thống của bạn hỗ trợ, hãy thêm thứ gì đó giống như sau vào tệp cấu hình nhân của bạn:

```
options "SHMALL=8192"
options "SHMMAX=\(SHMALL*PAGE_SIZE\)"
```

SHMALL được đo đếm theo 4 KB trang, nên một giá trị 1024 đại diện cho 4MB bộ nhớ được chia sẻ. Vì thế điều ở trên làm tăng vùng bộ nhớ được chia sẻ tối đa lên 32 MB. Đối với những người đang chạy phiên bản 4.3 hoặc sau này, bạn có lẽ cũng cần tăng KERNEL_VIRTUAL hơn mặc định 248. Một khi tất cả những thay đổi đã được thực hiện, hãy biên dịch lại nhân, và khởi động lại.

Các sự ra hiệu (Semaphores). Bạn có lẽ cũng sẽ muốn gia tăng số lượng các sự ra hiệu; tổng cộng hệ thống mặc định 60 sẽ chỉ cho phép khoảng 50 kết nối PostgreSQL. Hãy thiết lập các giá trị mà bạn muốn trong tệp cấu hình nhân của bạn, như:

```
options "SEMMNI=40"
options "SEMMNS=240"
```

FreeBSD

Các thiết lập mặc định chỉ phù hợp cho các cài đặt nhỏ (ví dụ, SHMMAX mặc định là 32 MB). Các thay đổi có thể được thực hiện qua các giao diện sysctl hoặc loader. Các tham số sau có thể được thiết lập bằng việc sử dụng sysctl:

```
$ sysctl -w kern.ipc.shmall=32768
$ sysctl -w kern.ipc.shmmax=134217728
$ sysctl -w kern.ipc.semmap=256
```

Để các thiết lập đó ổn định khi khởi động, hãy sửa /etc/sysctl.conf.

Các thiết lập sự ra hiệu còn lại sẽ là chỉ đọc cho tới khi nào sysctl được quan tâm, nhưng có

thể bị thay đổi trước khi khởi động bằng việc sử dụng dấu nhắc loader:

```
(loader) set kern.ipc.semmni=256
(loader) set kern.ipc.semmns=512
(loader) set kern.ipc.semmnu=256
```

Tương tự chúng có thể được lưu lại giữa các lần khởi động trong /boot/loader.conf.

Bạn có thể cũng muốn thiết lập cấu hình cho nhân của bạn để khóa bộ nhớ được chia sẻ trong RAM và ngăn cản nó khỏi bị phân trang thành bộ nhớ hoán đổi (swap). Điều này có thể được hoàn thành bằng việc sử dụng sysctl thiết lập kern.ipc.shm_use_phys.

Nếu đang chạy trong các jail của FreeBSD và đang kích hoạt security.jail.sysvipc_allowed của sysctl, thì các postmaster (các bưu tá) đang chạy trong các jail khác nhau nên được những người sử dụng các hệ điều hành khác nhau chạy. Điều này cải thiện an toàn vì nó ngăn cản những người sử dụng không phải gốc root khỏi việc can thiệp bằng bộ nhớ được chia sẻ hoặc các sự ra hiệu trong các jail khác nhau, và nó cho phép PostgreSQL IPC làm sạch mã để vận hành đúng phù hợp. (Trong FreeBSD 6.0 và sau này mã làm sạch IPC không dò tìm đúng các tiến trình trong các jail khác, ngăn cản việc chạy của các bưu tá trên cùng cổng trong các jail khác).

Các phiên bản FreeBSD trước 4.0 làm việc giống OpenBSD (xem bên dưới).

NetBSD

Trong NetBSD 5.0 và sau này, các tham số IPC có thể được tinh chỉnh bằng việc sử dụng sysctl, ví dụ:

```
$ sysctl -w kern.ipc.shmmax=16777216
```

Để các thiết lập này ổn định nhất quán đối với sự khởi động máy, hãy sửa /etc/sysctl.conf.

Bạn cũng có thể muốn thiết lập cấu hình cho nhân của bạn khóa bộ nhớ được chia sẻ trong RAM và ngăn trở nó khỏi bị phân trang thành bộ nhớ hoán đổi (swap). Điều này có thể được hoàn thành bằng việc sử dụng sysctl thiết lập kern.ipc.shm_use_phys.

Các phiên bản NetBSD trước 5.0 làm việc như OpenBSD (bên dưới), ngoại trừ là các tham số sẽ được thiết lập với từ khóa options chứ không phải option.

OpenBSD

Các lựa chọn SYSVSHM và SYSVSEM cần phải được bật khi nhân được biên dịch. (Chúng là mặc định). Kích cỡ cực đại của bộ nhớ được chia sẻ được lựa chọn SHMMAXPGS (theo các trang) xác định. Thứ sau đây chỉ ra một ví dụ về cách để thiết lập các tham số khác nhau:

```
option  SYSVSHM
option  SHMMAXPGS=4096
option  SHMSEG=256

option  SYSVSEM
option  SEMMNI=256
option  SEMMNS=512
option  SEMMNU=256
option  SEMMAP=256
```

Bạn cũng có thể muốn thiết lập cấu hình cho nhân của bạn để khóa bộ nhớ được chia sẻ vào RAM và ngăn cản nó khỏi bị phân trang thành bộ nhớ hoán đổi swap. Điều này có thể được

hành thành bằng việc sử dụng sysctl thiết lập kern.ipc.shm_use_phys.

HP-UX

Các thiết lập mặc định có xu hướng đáp ứng đủ cho các cài đặt thông thường. Trên HP-UX 10, mặc định cho SEMMNS là 128, nó có lẽ là quá thấp đối với các site cơ sở dữ liệu lớn hơn.

Các tham số IPC có thể được thiết lập trong Quản lý Quản trị Hệ thống - SAM (System Administration Manager) dưới Kernel Configuration → Configurable Parameters. Hãy chọn Tạo một Nhân Mới (Create A New Kernel) khi bạn làm xong.

Linux

Kích thước đoạn cực đại mặc định là 32 MB, nó chỉ phù hợp cho các cài đặt PostgreSQL rất nhỏ. Kích thước tổng tối đa mặc định là 2097152 trang. Một trang hầu như luôn là 4096 byte ngoại trừ trong các cấu hình nhân không bình thường với "các trang khổng lồ" (sử dụng PAGE_SIZE để kiểm tra hợp lệ). Điều đó tạo một giới hạn mặc định 8 GB, thường là đủ, nhưng không luôn đủ.

Các thiết lập kích cỡ bộ nhớ được chia sẻ có thể được thay đổi thông qua giao diện sysctl. Ví dụ, để cho phép 16 GB:

```
$ sysctl -w kernel.shmmax=17179869184
$ sysctl -w kernel.shmall=4194304
```

Hơn nữa các thiết lập đó có thể được giữ lại giữa các lần khởi động lại máy trong tệp /etc/sysctl.conf.

Làm như thế được khuyến cáo cao độ.

Các phân phối cũ có thể không có chương trình sysctl, nhưng những thay đổi tương ứng có thể được thực hiện bằng việc điều khiển hệ thống tệp /proc:

```
$ echo 17179869184 >/proc/sys/kernel/shmmax
$ echo 4194304 >/proc/sys/kernel/shmall
```

Các mặc định còn lại khá rộng lượng về kích cỡ, và thường không đòi hỏi những thay đổi.

MacOS X

Phương pháp được khuyến cáo cho việc thiết lập cấu hình bộ nhớ được chia sẻ trong OS X là để tạo ra một tệp có tên là /etc/sysctl.conf, chứa các chỉ định biến như:

```
kern.sysv.shmmax=4194304
kern.sysv.shmmin=1
kern.sysv.shmmni=32
kern.sysv.shmseg=8
kern.sysv.shmall=1024
```

Lưu ý rằng trong một số phiên bản OS X, tất cả 5 tham số bộ nhớ được chia sẻ phải được thiết lập trong /etc/sysctl.conf, nếu không thì các giá trị đó sẽ bị bỏ qua.

Hãy nhận thức được rằng các phát hành gần đây của OS X bỏ qua các ý định thiết lập SHMMAX thành một giá trị mà sẽ không phải là một bội số nhân xác của 4096.

SHMALL được đo đếm theo các trang 4 KB trong nền tảng này.

Trong các phiên bản cũ hơn của OS X, bạn sẽ cần khởi động lại để những thay đổi trong các tham số bộ nhớ được chia sẻ có hiệu lực. Từ 10.5 có khả năng thay đổi tất cả ngoại trừ

SHMMNI ngay tại chỗ, bằng việc sử dụng sysctl. Nhưng vẫn còn tốt nhất là thiết lập các giá trị được ưu tiên của bạn thông qua /etc/sysctl.conf, sao cho các giá trị sẽ được giữ qua các lần khởi động lại máy.

Tệp /etc/sysctl.conf là duy nhất được đề cập tới trong OS X 10.3.9 và sau này. Nếu bạn đang chạy một phiên bản trước 10.3.x, thì bạn phải sửa tệp /etc/rc và thay đổi các giá trị trong các lệnh sau:

```
sysctl -w kern.sysv.shmmax
sysctl -w kern.sysv.shmmin
sysctl -w kern.sysv.shmmni
sysctl -w kern.sysv.shmseg
sysctl -w kern.sysv.shmall
```

Lưu ý rằng /etc/rc thường bị ghi đè bằng các bản cập nhật của OS X, vì thế bạn nên kỳ vọng phải làm lại các sửa đổi đó sau từng bản cập nhật .

Trong OS X 10.2 và trước đó, hãy sửa các lệnh đó trong tệp /System/Library/StartupItems/SystemTuning/SystemTuning.

SCO OpenServer

Trong cấu hình mặc định, chỉ 512 KB bộ nhớ được chia sẻ cho từng đoạn là được phép. Để tăng thiết lập đó, trước hết hãy thay đổi thư mục /etc/conf/cf.d. Để hiển thị giá trị hiện hành của SHMMAX, hãy chạy:

```
./configure -y SHMMAX
```

Để thiết lập một giá trị mới cho SHMMAX, hãy chạy:

```
./configure SHMMAX=value
```

trong đó value là giá trị mới mà bạn muốn sử dụng (theo byte). Sau khi thiết lập SHMMAX, hãy xây dựng lại nhân:

```
./link_unix
```

và khởi động lại.

Solaris

Ít nhất trong phiên bản 2.6, kích thước tối đa mặc định của một đoạn bộ nhớ được chia sẻ là quá thấp đối với PostgreSQL. Các thiết lập phù hợp có thể được thay đổi trong /etc/system, ví dụ:

```
set shmsys:shminfo_shmmax=0x2000000
set shmsys:shminfo_shmmin=1
set shmsys:shminfo_shmmni=256
set shmsys:shminfo_shmseg=256

set semsys:seminfo_semmap=256
set semsys:seminfo_semmni=512
set semsys:seminfo_semmns=512
set semsys:seminfo_semmsl=32
```

Bạn cần phải khởi động lại để các thay đổi có hiệu lực.

Xem thêm http://sunsite.uakom.sk/sunworldonline/swol-09-1997/swol-09-insidesolaris.html để có thông tin về bộ nhớ được chia sẻ trong Solaris.

UnixWare

Trong UnixWare 7, kích thước cực đại cho các đoạn bộ nhớ được chia sẻ chỉ là 512 KB trong cấu hình mặc định. Để hiển thị giá trị hiện hành của SHMMAX, hãy chạy:

```
/etc/conf/bin/idtune -g SHMMAX
```

nó hiển thị các giá trị hiện hành, mặc định và tối đa. Để thiết lập một giá trị mới cho SHMMAX, hãy chạy:

```
/etc/conf/bin/idtune SHMMAX value
```

trong đó value là giá trị mới mà bạn muốn sử dụng (theo byte). Sau khi thiết lập SHMMAX, hãy xây dựng lại nhân:

```
/etc/conf/bin/idbuild -B
```

và khởi động lại.

Bảng 17-2. Sử dụng bộ nhớ được chia sẻ của PostgreSQL

Sử dụng	Bộ nhớ được chia sẻ tiệm cận theo số byte được yêu cầu (như đối với 8.3)
Các kết nối	(1800 + 270 * max_locks_per_transaction) * max_connections
Trình công việc tự động hút chân không	(1800 + 270 * max_locks_per_transaction) * autovacuum_max_workers
Các giao dịch được chuẩn bị	(770 + 270 * max_locks_per_transaction) * max_prepared_transactions
Các bộ nhớ tạm đĩa được chia sẻ	(block_size + 208) * shared_buffers
Các bộ nhớ tạm WAL	(wal_block_size + 8) * wal_buffers
Các yêu cầu không gian cố định	770 kB

17.4.2. Các giới hạn tài nguyên

Các hệ điều hành giống Unix ép các dạng hạn chế tài nguyên khác nhau có thể can thiệp bằng sự vận hành máy chủ PostgreSQL của bạn. Đặc biệt quan trọng là các giới hạn về số các tiến trình theo từng người sử dụng, số các tệp mở theo tiến trình, và lượng bộ nhớ sẵn có cho từng tiến trình.

Từng trong số đó có một giới hạn "cứng" và "mềm". Giới hạn mềm là những gì thực sự được tính tới nhưng nó có thể bị người sử dụng thay đổi cho tới giới hạn cứng. Giới hạn cứng chỉ có thể bị người sử dụng gốc root thay đổi. Lời gọi hệ thống setrlimit có trách nhiệm cho việc thiết lập các tham số đó. Lệnh có sẵn của trình biên dịch (shell) ulimit (Bourne shells) hoặc limit (csh) được sử dụng để kiểm soát các giới hạn tài nguyên từ dòng lệnh.

Trong các hệ thống có xuất xứ từ BSD thì tệp /etc/login.conf kiểm soát tập hợp các giới hạn tài nguyên khác nhau trong quá trình đăng nhập. Hãy xem tài liệu hệ điều hành để có các chi tiết. Các tham số phù hợp là maxproc, openfiles và datasize. Ví dụ:

```
default:\
...
        :datasize-cur=256M:\
        :maxproc-cur=256:\
        :openfiles-cur=256:\
...
```

(-cur là giới hạn mềm. Hãy nối thêm -max để thiết lập giới hạn cứng).

Các nhân cũng có thể có các giới hạn rộng khắp hệ thống trong một số tài nguyên.

- Trong Linux /proc/sys/fs/file-max xác định số cực đại các tệp mở mà nhân sẽ hỗ trợ. Nó có thể bị thay đổi bằng việc viết một số khác vào tệp đó hoặc bằng việc bổ sung thêm một chỉ định trong /etc/sysctl.conf. Giới hạn cực đại các tệp theo từng tiến trình được cố định vào thời điểm nhân (kernel) được biên dịch; xem /usr/src/linux/Documentation/proc.txt để có thêm thông tin.

Máy chủ PostgreSQL sử dụng một tiến trình theo kết nối sao cho bạn sẽ đưa ra ít nhất càng nhiều tiến trình được phép cho các kết nối càng tốt, bổ sung cho những gì bạn cần cho phần còn lại của hệ thống của bạn. Điều này thường không phải là vấn đề nhưng nếu bạn chạy vài máy chủ trong một máy thì mọi điều đó có thể sẽ chặt chẽ.

Giới hạn mặc định đối với các tệp mở thường được thiết lập về các giá trị "thân thiện về mặt xã hội" mà cho phép nhiều người sử dụng cùng tồn tại trong một máy mà không có việc sử dụng một miếng nhỏ các tài nguyên nào không phù hợp của hệ thống. Nếu bạn chạy nhiều máy chủ trong một máy thì điều này có thể là những gì bạn muốn, nhưng trong các máy chủ chuyên tâm thì bạn có thể muốn đặt ra giới hạn này.

Mặt khác, một số hệ thống cho phép các tiến trình riêng rẽ để mở số lượng lớn các tệp; nếu nhiều hơn một ít các tiến trình làm thế thì giới hạn rộng khắp hệ thống có thể dễ dàng vượt qua được. Nếu bạn thấy điều này xảy ra, và bạn không muốn điều chỉnh giới hạn rộng khắp hệ thống, thì bạn có thể thiết lập tham số cấu hình max_files_per_process của PostgreSQL để giới hạn sử dụng các tệp mở.

17.4.3. Ủy thác quá bộ nhớ Linux

Trong Linux 2.4 và sau này, hành xử của bộ nhớ ảo mặc định không là tối ưu cho PostgreSQL. Vì cách mà nhân triển khai bộ nhớ bị ủy thác quá mức, nên nhân có thể làm kết thúc postmaster (tiến trình chủ của máy chủ) của PostgreSQL nếu các đòi hỏi bộ nhớ hoặc của PostgreSQL hoặc của tiến trình khác làm cho hệ thống tiêu hết bộ nhớ ảo.

Nếu điều này xảy ra, thì bạn sẽ thấy một thông điệp nhân trông giống thế này (hãy tư vấn tài liệu và cấu hình hệ thống của bạn về nơi để tìm kiếm một thông điệp như vậy):

```
Out of Memory: Killed process 12345 (postgres).
```

Điều này chỉ ra rằng tiến trình postgres đã bị dừng vì sức ép về bộ nhớ. Dù các kết nối cơ sở dữ liệu hiện hành sẽ tiếp tục vận hành bình thường, thì không kết nối mới nào sẽ được chấp nhận nữa cả. Để phục hồi, PostgreSQL sẽ cần phải được khởi động lại.

Một cách để tránh vấn đề này là chạy PostgreSQL trong một máy nơi mà bạn có thể chắc chắn rằng các tiến trình khác sẽ không làm cho máy chạy hết bộ nhớ. Nếu bộ nhớ là chặt chẽ, thì việc gia tăng không gian hoán đổi của hệ điều hành có thể giúp tránh được vấn đề này, vì kẻ hủy diệt bộ nhớ (OOM) chỉ được triệu hồi khi bộ nhớ vật lý và không gian hoán đổi bị dùng hết.

Nếu bản thân PostgreSQL là lý do của việc hệ thống dùng hết bộ nhớ, thì bạn có thể tránh vấn đề đó bằng việc thay đổi cấu hình của bạn. Trong một số trường hợp, nó có thể giúp để làm giảm các tham số cấu hình có liên quan tới bộ nhớ, đặc biệt là shared_buffers và work_mem. Trong các trường hợp

khác, vấn đề có thể xảy ra bằng việc cho phép quá nhiều kết nối tới bản thân máy chủ cơ sở dữ liệu. Trong nhiều trường hợp, có thể tốt hơn để làm giảm max_connections và thay vào đó hãy sử dụng phần mềm kết nối kéo từ bên ngoài.

Trong Linux 2.6 và sau này, có khả năng sửa đổi hành vi của nhân sao cho nó sẽ không "ủy thác quá" bộ nhớ. Dù thiết lập này sẽ không ngăn được trình diệt OOM (OOM killer[1]) khỏi việc được triệu gọi cùng nhau, thì nó sẽ làm giảm cơ hội đáng kể và vì thế dẫn tới hành vi hệ thống lành mạnh hơn. Điều này được thực hiện bằng việc lựa chọn chế độ ủy thác quá ngặt nghèo thông qua sysctl:

```
sysctl -w vm.overcommit_memory=2
```

hoặc đặt một khoản đầu vào trong /etc/sysctl.conf. Bạn cũng có thể muốn sửa đổi thiết lập có liên quan vm.overcommit_ratio. Để có thêm chi tiết, xem tệp Documentation/vm/overcommit-accounting.

Một tiếp cận khác có thể được sử dụng với hoặc không với việc sửa đổi vm.overcommit_memory, sẽ thiết lập giá trị đặc thù tiến trình oom_adj cho tiến trình postmaster về -17, bằng cách đó đảm bảo nó sẽ không bị trình diệt OOM nhằm vào. Cách đơn giản nhất để làm điều này là hãy thực thi:

```
echo -17 > /proc/self/oom_adj
```

trong script khởi tạo của postmaster ngay trước khi triệu gọi postmaster. Hãy lưu ý rằng hành động này phải được thực hiện như là gốc root, nếu không nó sẽ không có tác dụng; vì thế một script khởi tạo do root sở hữu là nơi dễ nhất để làm điều này. Nếu bạn làm điều này, thì bạn cũng có thể muốn xây dựng PostgreSQL với -DLINUX_OOM_ADJ=0 được thêm vào cho CPPFLAGS. Điều đó sẽ làm cho các tiến trình con của postmaster chạy với giá trị thông thường 0 của oom_adj, sao cho trình diệt OOM vẫn có thể nhằm vào chúng khi cần.

> **Lưu ý**: Một số nhân Linux 2.4 của các nhà bán hàng được nêu sẽ có các phiên bản sớm của nhân 2.6 cho tham số sysctl ủy thác quá. Tuy nhiên, việc thiết lập vm.overcommit_memory về 2 trong một nhân 2.4 mà không có mã phù hợp sẽ làm cho mọi điều tồi tệ hơn, chứ không tốt hơn. Được khuyến cáo rằng bạn kiểm tra kỹ mã nguồn của nhân thực tế (xem hàm vm_enough_memory trong tệp mm/mmap.c) để kiểm tra những gì được hỗ trợ trong nhân của bạn trước khi bạn cố thử điều này trong một cài đặt nhân 2.4. Sự hiện diện của tệp tài liệu overcommit-accounting sẽ không được lấy như là bằng chứng rằng tính năng đó là có. Nếu có nghi ngờ, hãy tư vấn với một chuyên gia về nhân hoặc người bán cho bạn nhân đó.

17.5. Tắt máy chủ

Có vài cách để tắt máy chủ cơ sở dữ liệu. bạn kiểm soát dạng tắt máy bằng việc gửi các tín hiệu khác nhau tới tiến trình chủ postgres.

SIGTERM

> Đây là chế độ Tắt máy Thông minh (Smart Shutdown). Sau khi nhận được SIGTERM, máy chủ sẽ không cho phép các kết nối mới, nhưng để các phiên đang tồn tại kết thúc công việc của chúng một cách bình thường. Nó tắt máy chỉ sau khi tất cả các phiên kết thúc. Nếu máy chủ đang ở chế độ sao lưu trực tuyến, thì nó sẽ tiếp tục chờ đợi cho tới khi chế độ sao lưu trực tuyến không còn hoạt động nữa. Trong khi chế độ sao lưu còn hoạt động, thì các kết nối

1 http://lwn.net/Articles/104179/

mới vẫn sẽ được cho phép, nhưng chỉ đối với các siêu người sử dụng (superuser) (ngoại lệ này cho phép một siêu người sử dụng kết nối để kết thúc chế độ sao lưu trực tuyến). Nếu máy chủ đang phục hồi khi một sự tắt máy thông minh được yêu cầu, thì sự phục hồi và sự nhân bản dòng sẽ bị dừng chỉ sau khi tất cả các phiên thông thường đã kết thúc.

SIGINT

Đây là chế độ Tắt máy Nhanh (Fast Shutdown). Máy chủ không cho phép các kết nối mới và gửi tất cả các tiến trình đang tồn tại tới SIGTERM, nó sẽ làm cho chúng bỏ qua các giao dịch hiện hành của chúng và thoát ra một cách đúng đắn. Nó sau đó chờ cho tất cả các tiến trình máy chủ thoát ra và cuối cùng tắt máy. Nếu máy chủ ở chế độ sao lưu trực tuyến, thì chế độ sao lưu sẽ được kết thúc, làm cho sự sao lưu thành vô dụng.

SIGQUIT

Đây là chế độ Tắt máy Ngay lập tức (Immediate Shutdown). Tiến trình postgres chủ sẽ gửi một SIGQUIT tới tất cả các tiến trình con và thoát ra ngay lập tức, không cho việc tắt máy đúng đắn. Các tiến trình con có khả năng thoát ra ngay lập tức khi nhận được SIGQUIT. Điều này sẽ dẫn tới sự phục hồi (bằng việc lặp lại lưu ký WAL) khi lần tiếp sau khởi động. Điều này chỉ được khuyến cáo trong các trường hợp khẩn cấp.

Chương trình pg_ctl đưa ra một giao diện thuận tiện cho việc gửi các tín hiệu đó để tắt máy chủ.

Như một lựa chọn thay thế, bạn có thể gửi tín hiệu đó trực tiếp bằng việc sử dụng kill trong các hệ thống không phải Windows. PID của tiến trình postgres có thể thấy được bằng việc sử dụng chương trình ps, hoặc từ tệp postmaster.pid trong thư mục cơ sở dữ liệu. Ví dụ, để tắt máy nhanh:

```
$ kill -INT ‘head -1 /usr/local/pgsql/data/postmaster.pid‘
```

Quan trọng: Tốt nhất không sử dụng SIGKILL để tắt máy chủ. Làm như vậy sẽ ngăn chặn được máy chủ khỏi việc nhả bộ nhớ được chia sẻ và các cờ đánh tín hiệu, chúng có thể sau đó phải được thực hiện bằng tay trước khi một máy chủ mới có thể được khởi động. Hơn nữa, SIGKILL sẽ giết tiến trình postgres mà không để cho nó trễ tín hiệu đối với các tiến trình con của nó, vì thế cũng sẽ là cần thiết để giết các tiến trình con bằng tay.

Để kết thúc một phiên riêng rẽ trong khi cho phép các phiên khác tiếp tục, hãy sử dụng pg_terminate_backend() (xem Bảng 9-55) hoặc gửi một tín hiệu SIGTERM tới tiến trình con có liên quan tới phiên đó.

17.6. Ngăn chặn việc đánh lừa máy chủ

Trong khi máy chủ đang chạy, không có khả năng đối với một người sử dụng độc hại chiếm chỗ máy chủ cơ sở dữ liệu thông thường. Tuy nhiên, khi máy chủ bị tắt, có khả năng đối với một người sử dụng cục bộ lừa gạt máy chủ thông thường bằng việc khởi động máy chủ của riêng họ. Máy chủ lừa gạt có thể đọc các mật khẩu và các truy vấn được các máy trạm gửi tới, nhưng có thể không trả về bất kỳ dữ liệu nào vì thư mục PGDATA có thể vẫn còn là an toàn vì các quyền của thư mục đó. Việc lừa gạt là có khả năng vì bất kỳ người sử dụng nào cũng có thể khởi động một máy chủ cơ sở dữ liệu; một máy trạm không thể nhận diện được một máy chủ không hợp lệ trừ phi nó được thiết lập cấu hình đặc biệt.

Cách đơn giản nhất để ngăn chặn việc lừa gạt đối với các kết nối cục bộ local là hãy sử dụng thư mục khe cắm miền (domain socket) Unix (unix_socket_directory) mà có quyền ghi chỉ cho một người sử dụng cục bộ tin cậy. Điều này ngăn chặn một người sử dụng độc hại khỏi việc tạo ra tệp khe cắm của riêng anh ta trong thư mục đó. Nếu bạn quan tâm tới một số ứng dụng thì vẫn có thể tham chiếu /tmp cho tệp khe cắm và vì thế sẽ có khả năng bị tổn thương đối với việc lừa gạt, trong khi khởi động hệ điều hành sẽ tạo ra một liên kết biểu tượng /tmp/.s.PGSQL.5432 chỉ tới tệp khe cắm được tái định vị. Bạn cũng có thể cần sửa đổi script làm sạch /tmp để ngăn chặn loại bỏ liên kết biểu tượng đó.

Để ngăn chặn việc lừa gạt trong các kết nối TCP, giải pháp tốt nhất là hãy sử dụng các chứng thực SSL và hãy chắc chắn rằng các máy trạm kiểm tra chứng thực của máy chủ đó. Để làm điều này, máy chủ phải được thiết lập cấu hình để chỉ chấp nhận các kết nối hostssl (Phần 19.1) và có các tệp SSL server.key (khóa) và server.crt (chứng thực). Máy trạm TCP phải kết nối bằng việc sử dụng sslmode=verify-ca hoặc verify-full và có tệp chứng thực gốc root phù hợp được cài đặt (Phần 31.1).

17.7. Lựa chọn mã hóa

PostgreSQL đưa ra vài mức mã hóa, và đưa ra sự mềm dẻo trong việc bảo vệ các dữ liệu khỏi bị lộ ra vì ăn cắp máy chủ cơ sở dữ liệu, các quản trị viên vô lương tâm, và các mạng không an toàn. Sự mã hóa cũng có thể được yêu cầu để đảm bảo an toàn cho các dữ liệu nhạy cảm như các hồ sơ y tế hoặc các giao dịch tài chính.

Mã hóa lưu trữ mật khẩu

Mặc định, các mật khẩu người sử dụng cơ sở dữ liệu được lưu trữ như các hàm băm MD5, nên người quản trị không thể xác định mật khẩu thực được chỉ định cho người sử dụng đó. Nếu mã hóa MD5 được sử dụng cho xác thực máy trạm, thì mật khẩu không được mã hóa thậm chí không bao giờ hiện diện tạm thời trên máy chủ vì máy trạm mã hóa nó bằng MD5 trước khi gửi qua mạng.

Mã hóa cho các cột đặc thù

Thư viện pgcrypto hàm contrib cho phép các trường nhất định sẽ được mã hóa khi được lưu trữ. Điều này là hữu dụng nếu chỉ một số dữ liệu là nhạy cảm. Máy trạm cung cấp khóa mã hóa và dữ liệu đó được giải mã trên máy chủ và sau đó được gửi cho máy trạm.

Dữ liệu được mã hóa đó và khóa giải mã có trên máy chủ một thời gian ngắn khi nó đang được giải mã và được giao tiếp giữa máy chủ và máy trạm. Điều này làm hiện diện một thời điểm ngắn nơi mà dữ liệu và các khóa có thể bị can thiệp từ ai đó với sự truy cập hoàn chỉnh tới máy chủ cơ sở dữ liệu, như quản trị viên hệ thống chẳng hạn.

Mã hóa một phần dữ liệu

Trên Linux, mã hóa có thể làm ở lớp đỉnh của một hệ thống tệp bằng việc sử dụng một "thiết bị lặp ngược" (loopback device). Điều này cho phép phân vùng của toàn bộ hệ thống tệp sẽ được mã hóa trên đĩa, và được hệ điều hành giải mã. Trên FreeBSD, tiện ích tương đương được gọi là Mã hóa Đĩa Dựa vào GEOM - gbde (GEOM Based Disk Encryption), và nhiều

hệ điều hành khác hỗ trợ chức năng này, bao gồm cả Windows.

Cơ chế này ngăn chặn các dữ liệu không được mã hóa khỏi bị đọc từ các ổ đĩa nếu các ổ đĩa hoặc toàn bộ máy tính bị ăn cắp. Điều này không bảo vệ chống lại được các cuộc tấn công trong khi hệ thống tệp được kích hoạt, vì khi được kích hoạt, hệ điều hành đưa ra một kiểu nhìn dữ liệu không được mã hóa. Tuy nhiên, để kích hoạt hệ thống tệp, bạn cần một số cách cho khóa mã hóa sẽ được truyền tới hệ điều hành, và đôi khi khóa đó được lưu trữ ở đâu đó trên máy chủ mà kích hoạt đĩa đó.

Mã hóa các mật khẩu khắp một mạng

Phương pháp xác thực bằng MD5 mã hóa nhân đôi mật khẩu trên máy trạm trước khi gửi nó tới máy chủ. Trước hết nó mã hóa bằng MD5 dựa vào tên người sử dụng, và sau đó mã hóa dựa vào một phần ngẫu nhiên được máy chủ gửi khi kết nối cơ sở dữ liệu được thiết lập. Đây chính là giá trị được mã hóa gấp đôi được gửi qua mạng tới máy chủ. Mã hóa gấp đôi không chỉ ngăn chặn mật khẩu khỏi bị phát hiện, mà nó cũng ngăn chặn kết nối khác khỏi việc sử dụng cùng y hệt mật khẩu được mã hóa để kết nối tới máy chủ cơ sở dữ liệu sau này.

Mã hóa dữ liệu khắp một mạng

Các kết nối SSL mã hóa tất cả các dữ liệu được gửi qua mạng: mật khẩu, các truy vấn, và dữ liệu được trả về. Tệp pg_hba.conf cho phép các quản trị hệ thống chỉ định các máy chủ nào có thể sử dụng các kết nối (máy chủ) nào không được mã hóa và đòi hỏi các kết nối SSL được mã hóa (hostssl). Hơn nữa, các máy trạm có thể chỉ định là chúng kết nối tới các máy chủ chỉ qua SSL. Stunnel hoặc SSH cũng có thể được sử dụng để mã hóa các cuộc truyền.

Xác thực máy chủ SSL

Có khả năng cho cả máy chủ và máy trạm cung cấp các xác thực SSL cho nhau. Điều này cần một vài cấu hình thêm ở từng phía, nhưng điều này đưa ra sự kiểm tra hợp lệ nhận diện mạnh hơn so với chỉ sử dụng các mật khẩu. Nó ngăn chặn một máy tính khỏi việc giả vờ là máy chủ đủ dài để đọc mật khẩu được máy trạm gửi đi. Nó cũng giúp ngăn chặn các cuộc tấn công "người chặn giữa đường", nơi mà một máy tính nằm giữa máy chủ và máy trạm giả vờ làm máy chủ để đọc và truyền tất cả dữ liệu giữa máy chủ và máy trạm đó.

Mã hóa ở phía máy trạm

Nếu người quản trị hệ thống máy chủ không thể tin cậy được, thì cần thiết đối với máy trạm phải mã hóa dữ liệu; cách này, các dữ liệu không được mã hóa sẽ không bao giờ xuất hiện trên máy chủ cơ sở dữ liệu. Dữ liệu được mã hóa trên máy trạm trước khi được gửi tới máy chủ, và kết quả cơ sở dữ liệu phải được giải mã ở phía máy trạm trước khi được sử dụng.

17.8. Kết nối TCP/IP an toàn với SSL

PostgreSQL có sự hỗ trợ bẩm sinh cho việc sử dụng các kết nối SSL để mã hóa các giao tiếp truyền thông máy chủ/máy trạm để gia tăng sự an toàn. Điều này đòi hỏi là OpenSSL được cài đặt trên cả các máy chủ và máy trạm và sự hỗ trợ trong PostgreSQL được kích hoạt vào thời điểm xây dựng (xem Chương 15).

Với sự hỗ trợ SSL được biên dịch sẵn, máy chủ PostgreSQL có thể được khởi động với SSL được kích hoạt bằng việc thiết lập tham số ssl ở on (bật) trong postgresql.conf. Máy chủ sẽ nghe cả các kết nối thông thường và SSL trên cùng một cổng TCP, và sẽ thương lượng với bất kỳ máy trạm nào đang kết nối về việc liệu có sử dụng SSL hay không. Mặc định, điều này là sự lựa chọn của máy trạm; xem Phần 19.1 về cách để thiết lập máy chủ để yêu cầu sử dụng SSL cho một vài hoặc tất cả các kết nối.

PostgreSQL đọc tệp cấu hình OpenSSL rộng khắp hệ thống. Mặc định, tệp này được đặt tên là openssl.cnf và nằm trong thư mục được openssl version -d nêu. Mặc định này có thể bị việc thiết lập biến môi trường OPENSSL_CONF ghi đè vào tên của tệp cấu hình mong muốn.

OpenSSL hỗ trợ một dải rộng lớn các thuật toán mật mã và xác thực, với độ mạnh khác nhau. Trong khi một danh sách mật mã có thể được chỉ định trong tệp cấu hình của OpenSSL, thì bạn có thể chỉ định các mật mã một cách đặc biệt để máy chủ cơ sở dữ liệu sử dụng bằng việc sửa đổi ssl_ciphers trong postgresql.conf.

> **Lưu ý**: Có khả năng có xác thực mà không cần mã hóa hoàn toàn bằng việc sử dụng các mật mã NULL-SHA hoặc NULL-MD5. Tuy nhiên, một người chặn giữa đường có thể đọc và truyền các giao tiếp truyền thông giữa máy chủ và máy trạm. Hơn nữa, toàn bộ mã hóa là tối thiểu so với toàn bộ xác thực. Vì các lý do đó nên các mật mã NULL không được khuyến cáo.

Để khởi động trong chế độ SSL, các tệp server.crt và server.key phải tồn tại trong thư mục dữ liệu của máy chủ. Các tệp đó nên gồm chứng thực và khóa riêng của máy chủ, một cách tương ứng. Trong các hệ thống Unix, quyền trong server.key phải vô hiệu hóa bất kỳ sự truy cập nào tới thế giới hoặc nhóm; hãy lưu trữ điều này bằng lệnh chmod 0600 server.key. Nếu khóa riêng được bảo vệ bằng một mật khẩu, thì máy chủ sẽ nhắc phải vào mật khẩu đó và sẽ không khởi động cho tới khi nào mật khẩu đó được nhập vào.

Trong một số trường hợp, chứng thực của máy chủ có thể được một cơ quan chứng thực "trung gian" ký, hơn là một cơ quan mà những máy trạm trực tiếp tin cậy. Để sử dụng một chứng thực như vậy, hãy nối thêm chứng thực của cơ quan ký đó vào tệp server.crt, sau đó là chứng thực của cơ quan cha, và cứ tiếp tục như vậy tới một cơ quan "gốc root" mà được các máy trạm tin cậy. Chứng thực gốc root sẽ được đưa vào trong từng trường hợp nơi mà server.crt có nhiều hơn 1 chứng thực.

17.8.1. Sử dụng các chứng thực máy trạm

Để yêu cầu máy trạm cung cấp một chứng thực tin cậy được, hãy đặt các chứng thực của các cơ quan chứng thực (CA) mà bạn tin cậy vào tệp root.crt trong thư mục dữ liệu, và thiết lập tham số clientcert về 1 trên (các) dòng phù hợp của hostssl trong pg_hba.conf. Một chứng thực sau đó sẽ được yêu cầu từ máy trạm trong khi khởi động kết nối SSL. (Xem Phần 31.17 để có mô tả về cách thiết lập các chứng thực trên máy trạm). Máy chủ sẽ kiểm tra hợp lệ chứng thực máy trạm có được ký từ một trong những các cơ quan chứng thực tin cậy hay không. Các khoản của Danh sách Thu hồi Chứng thực - CRL (Certificate Revocation List) cũng được kiểm tra liệu tệp root.crl có tồn tại hay không. (Xem http://h71000.www7.hp.com/DOC/83final/BA554_90007/ch04s02.html để có các sơ đồ chỉ cách sử dụng chứng thực SSL).

Lựa chọn clientcert trong pg_hba.conf là sẵn sàng cho tất cả các phương pháp xác thực, nhưng chỉ cho các hàng được chỉ định như là hostssl. Khi clientcert không được chỉ định hoặc không được thiết lập về 0, thì máy chủ vẫn sẽ kiểm tra hợp lệ các chứng thực máy trạm được thể hiện đối với root.crt nếu tệp đó tồn tại - mà nó sẽ không khăng khăng rằng một chứng thực máy trạm sẽ được thể hiện.

Lưu ý rằng root.crt liệt kê các CA mức đỉnh mà sẽ được cân nhắc tin cậy cho việc ký các chứng thực máy trạm. Về nguyên tắc thì nó không cần liệt kê CA mà đã ký chứng thực máy chủ, dù trong hầu hết các trường hợp CA đó cũng có thể được tin cậy cho các chứng thực máy trạm.

Nếu bạn đang thiết lập các chứng thực máy trạm, thì bạn có thể muốn sử dụng phương pháp xác thực cert, sao cho các chứng thực kiểm tra xác thực của người sử dụng cũng như cung cấp an toàn kết nối. Xem Phần 19.3.9 để có thêm các chi tiết.

17.8.2. Sử dụng tệp máy chủ SSL

Các tệp server.key, server.crt, root.crt và root.crl chỉ được kiểm tra trong quá trình khởi động máy chủ; nên bạn phải khởi động lại máy chủ để những thay đổi trong chúng có hiệu lực.

Bảng 17-3. Sử dụng tệp máy chủ SSL

Tệp	Nội dung	Hiệu ứng
server.crt	chứng thực máy chủ	gửi tới máy trạm để chỉ ra nhận diện máy chủ
server.key	khóa riêng của máy chủ	chứng minh chứng thực máy chủ đã được chủ nhân gửi đi; không chỉ ra chủ nhân của chứng thực là tin cậy được
root.crt	các cơ quan chứng thực tin cậy	kiểm tra chứng thực máy trạm được một cơ quan chứng thực tin cậy ký
root.crl	các chứng thực bị các cơ quan chứng thực thu hồi	chứng thực máy trạm phải không nằm trong danh sách này

17.8.3. Tạo một chứng thực tự ký

Để tạo một chứng thực tự ký nhanh chóng cho máy chủ, hãy sử dụng lệnh OpenSSL sau:

```
openssl req -new -text -out server.req
```

Hãy điền các thông tin mà openssl yêu cầu. Hãy chắc chắn bạn vào tên máy chủ như là "Tên phổ biến" (Common Name); mật khẩu thách thức có thể để trống. Chương trình sẽ tạo ra một khóa được mật khẩu bảo vệ; nó sẽ không chấp nhận một mật khẩu ít hơn 4 ký tự. Để loại bỏ mật khẩu (nếu phải khi bạn muốn tự động khởi động máy chủ), hãy chạy các lệnh:

```
openssl rsa -in privkey.pem -out server.key
rm privkey.pem
```

Vào mật khẩu cũ để mở khóa cho khóa đang tồn tại. Bây giờ hãy làm:

```
openssl req -x509 -in server.req -text -key server.key -out server.crt
```

để biến chứng thực thành một chứng thực tự ký và sao chép khóa và chứng thực đó tới nơi mà máy chủ sẽ tìm chúng. Cuối cùng hãy làm:

```
chmod og-rwx server.key
```

vì máy chủ sẽ từ chối tệp nếu các quyền của nó là tự do hơn so với điều này. Để có thêm chi tiết về cách để tạo khóa riêng và chứng thực của máy chủ của bạn, hãy tham chiếu tới tài liệu OpenSSL.

Một chứng thực tự ký có thể được sử dụng cho việc kiểm thử, nhưng một chứng thực được một cơ quan chứng thực (CA) ký (hoặc một trong các CA toàn cầu hoặc địa phương) sẽ được sử dụng trong việc tạo đó sao cho các máy trạm có thể kiểm tra hợp lệ định danh của máy chủ. Nếu tất cả các máy trạm là cục bộ đối với tổ chức, thì việc sử dụng một CA địa phương được khuyến cáo.

17.9. Các kết nối TCP/IP an toàn với các đường hầm SSH

Có khả năng sử dụng SSH để mã hóa kết nối mạng giữa các máy trạm và một máy chủ PostgreSQL. Được làm đúng, điều này sẽ cung cấp một kết nối mạng an toàn đúng, thậm chí cho các máy trạm không có khả năng SSL.

Trước hết hãy chắc chắn rằng một máy chủ SSH đang chạy đúng trên cùng y hệt máy như là máy chủ PostgreSQL và bạn có thể đăng nhập bằng việc sử dụng ssh như một vài người sử dụng. Sau đó bạn có thể thiết lập một đường hầm an toàn với một lệnh giống thế này từ máy trạm:

```
ssh -L 63333:localhost:5432 joe@foo.com
```

Số đầu trong đối số -L, 63333, là số cổng của kết thúc đường hầm của bạn; nó có thể là bất kỳ cổng không được sử dụng nào. (IANA giữ lại các cổng từ 49152 tới 65535 cho các mục đích sử dụng riêng). Số thứ hai, 5432, là kết thúc ở xa của đường hầm: số cổng mà máy chủ của bạn đang sử dụng. Tên hoặc địa chỉ IP giữa các số cổng là máy chủ host với máy chủ cơ sở dữ liệu mà bạn sẽ kết nối tới, như được thấy từ máy chủ host mà bạn đang đăng nhập vào, nó là foo.com trong ví dụ này. Để kết nối tới máy chủ cơ sở dữ liệu bằng việc sử dụng đường hầm này, bạn hãy kết nối tới cổng 63333 trên máy cục bộ:

```
psql -h localhost -p 63333 postgres
```

Đối với máy chủ cơ sở dữ liệu thì nó sau đó sẽ trông như là bạn thực sự là người sử dụng joe trong máy chủ host foo.com đang kết nối tới localhost theo ngữ cảnh đó, và nó sẽ sử dụng thủ tục xác thực bất kỳ từng được thiết lập cấu hình cho các kết nối từ người sử dụng này và máy chủ host. Lưu ý rằng máy chủ đó sẽ không nghĩ kết nối đó là được mã hóa bằng SSL, vì trong thực tế nó không được mã hóa giữa máy chủ SSH và máy chủ PostgreSQL. Điều này sẽ không đặt ra bất kỳ rủi ro an toàn thêm nào miễn là chúng là trên cùng y hệt máy đó.

Để đường hầm được thiết lập thành công thì bạn phải được phép kết nối qua ssh như là joe@foo.com, hệt như bạn đã định sử dụng ssh để tạo một phiên làm việc của một máy đầu cuối.

Bạn cũng nên phải thiết lập cổng chuyển tiếp như

```
ssh -L 63333:foo.com:5432 joe@foo.com
```

nhưng sau đó máy chủ cơ sở dữ liệu sẽ coi kết nối đó như sẽ đi vào trong giao diện foo.com của nó, mà nó không được mở bằng thiết lập mặc định listen_addresses = 'localhost'. Điều này thường không phải là những gì bạn muốn.

Nếu bạn phải "nhảy" tới máy chủ cơ sở dữ liệu qua một số máy chủ host đăng nhập, thì một thiết lập có khả năng có thể trông giống như thế này:

```
ssh -L 63333:db.foo.com:5432 joe@shell.foo.com
```

Lưu ý rằng cách này thì kết nối từ shell.foo.com tới db.foo.com sẽ không được đường hầm SSH mã hóa. SSH đưa ra một ít khả năng cấu hình khi mạng bị hạn chế theo các cách thức khác nhau. Xin

tham chiếu tới tài liệu SSH để có các chi tiết.

> **Mẹo**: Vài ứng dụng khác tồn tại mà có thể cung cấp các đường hầm an toàn bằng việc sử dụng một thủ tục tương tự theo khái niệm đối với thủ tục vừa được mô tả.

Chương 18. Cấu hình máy chủ

Có nhiều tham số cấu hình ảnh hưởng tới hành vi của hệ thống cơ sở dữ liệu. Trong phần đầu của chương này, chúng tôi mô tả cách thiết lập các tham số cấu hình. Các phần tiếp sau thảo luận từng tham số đó một cách chi tiết.

18.1. Thiết lập các tham số

Tất cả các tên tham số là phân biệt chữ hoa chữ thường. Mỗi tham số lấy một giá trị của 1 trong 5 dạng: Boolean, integer (số nguyên), floating point (điểm thập phân), string (chuỗi) hoặc enum (đánh số). Các giá trị boolean có thể được viết như là on, off, true, false, yes, no, 1, 0 (tất cả đều có phân biệt chữ hoa chữ thường) hoặc bất kỳ tiền tố rõ ràng nào của chúng.

Một số thiết lập chỉ định giá trị bộ nhớ hoặc thời gian. Từng trong số đó có một đơn vị ngầm định, nó hoặc là kilobytes, hoặc các khối (thường là 8 kylobytes), mili giây, giây, hoặc phút. Các đơn vị mặc định có thể được thấy bằng việc tham chiếu tới pg_settings.unit. Để thuận tiện, một đơn vị khác cũng có thể được chỉ định một cách rõ ràng. Các đơn vị bộ nhớ hợp lệ là KB (kilobyte), MB (megabyte) và GB (gigabyte); các đơn vị thời gian hợp lệ là ms (mili giây), s (giây), min (phút), h (giờ) và d (ngày). Lưu ý rằng bội số của các đơn vị bộ nhớ là 1024, chứ không phải là 1.000.

Các tham số của dạng "đánh số" được chỉ định theo cách y hệt như các tham số chuỗi, nhưng bị giới hạn tới một tập hợp có giới hạn các giá trị. Các giá trị được phép có thể thấy được từ pg_settings.enumvals. Các giá trị tham số enum là có phân biệt chữ hoa chữ thường.

Một cách để thiết lập các tham số đó là hãy sửa tệp postgresql.conf, nó được giữ bình thường trong thư mục dữ liệu. (Một bản sao mặc định được cài đặt ở đó khi thư mục bó các cơ sở dữ liệu được khởi tạo). Một ví dụ của những gì tệp này có thể trông giống là:

```
# This is a comment
log_connections         = yes
log_destination         = 'syslog'
search_path             = '"$user", public'
shared_buffers          = 128MB
```

Một tham số được chỉ định theo từng dòng. Dấu bằng giữa tên và giá trị là tùy ý. Dấu trắng là vô nghĩa và các dòng trắng sẽ bị bỏ qua. Các dấu thăng (#) chỉ định phần còn lại của dòng như một chú giải. Các giá trị tham số mà không phải là các mã định danh đơn giản hoặc các số phải nằm trong các dấu ngoặc đơn. Để nhúng một dấu ngoặc đơn vào một giá trị tham số, hãy viết hoặc 2 dấu ngoặc kép (được ưa thích hơn) hoặc dấu chéo ngược.

Bổ sung thêm vào các thiết lập tham số, tệp postgresql.conf có thể có các lệnh bao gồm (include), nó chỉ định một tệp khác phải đọc và xử lý giống như nó đã được chèn vào trong tệp cấu hình ở thời điểm này. Các lệnh bao gồm đơn giản giống như sau:

```
include 'filename'
```

Nếu tên tệp không phải là đường dẫn tuyệt đối, thì nó được lấy như là tương đối cho thư mục có chứa tệp cấu hình tham chiếu. Các lệnh bao gồm có thể lồng nhau.

Tệp cấu hình được đọc lại bất cứ khi nào tiến trình chính của máy chủ nhận được một tín hiệu SIGHUP (nó được gửi đi dễ nhất bằng pg_ctl reload). Tiến trình chính của máy chủ cũng sinh ra tín

hiệu này cho tất cả các tiến trình hiện đang chạy của máy chủ sao cho các phiên làm việc đang tồn tại cũng có được giá trị mới đó. Như một sự lựa chọn, bạn có thể gửi tín hiệu đó tới một tiến trình của một máy chủ một cách trực tiếp. Một vài tham số chỉ có thể được thiết lập khi máy chủ khởi động; bất kỳ thay đổi nào đối với các đầu vào của chúng trong tệp cấu hình đều sẽ bị bỏ qua cho tới khi máy chủ được khởi động lại.

Cách thứ 2 để thiết lập các tham số cấu hình đó là để chúng như một lựa chọn dòng lệnh cho lệnh postgres, như:

```
postgres -c log_connections=yes -c log_destination='syslog'
```

Các lựa chọn dòng lệnh ghi đè lên bất kỳ thiết lập có xung đột nào trong postgresql.conf. Lưu ý rằng điều này có nghĩa là bạn sẽ không có khả năng thay đổi giá trị khi đang làm bằng việc soạn sửa postgresql.conf, vì thế trong khi phương pháp dòng lệnh có thể là thuận tiện, thì nó cũng có thể lấy đi của bạn sự mềm dẻo sau này.

Một cách ngẫu nhiên, là hữu dụng để trao một lựa chọn dòng lệnh cho chỉ một phiên làm việc đặc biệt. Biến môi trường PGOPTIONS có thể được sử dụng cho mục đích này ở phía máy trạm:

```
env PGOPTIONS='-c geqo=off' psql
```

(Điều này làm việc cho bất kỳ ứng dụng máy trạm nào dựa vào libpq, chứ không chỉ psql). Lưu ý rằng điều này sẽ không làm việc đối với các tham số cố định khi máy chủ được khởi động hoặc chúng phải được chỉ định trong postgresql.conf.

Hơn nữa, có khả năng chỉ định một tập hợp các thiết lập tham số cho một người sử dụng hoặc một cơ sở dữ liệu. Bất kỳ khi nào một phiên làm việc được khởi tạo, các thiết lập mặc định cho người sử dụng và cơ sở dữ liệu có liên quan sẽ được tải lên. Các lệnh ALTER USER và ALTER DATABASE, một cách tương ứng, sẽ được sử dụng để thiết lập cấu hình cho các thiết lập đó. Các thiết lập theo từng cơ sở dữ liệu sẽ ghi đè bất kỳ thứ gì nhận được từ dòng lệnh postgres hoặc tệp cấu hình, và tới lượt nó sẽ bị các thiết lập theo người sử dụng ghi đè; cả 2 sẽ bị các thiết lập theo phiên làm việc ghi đè.

Một vài tham số có thể bị thay đổi trong các phiên làm việc riêng rẽ của SQL với lệnh SET, ví dụ:

```
SET ENABLE_SEQSCAN TO OFF;
```

Nếu SET được phép, thì nó ghi đè lên tất cả các nguồn giá trị khác đối với tham số đó. Một vài tham số không thể bị thay đổi qua SET: ví dụ, nếu chúng kiểm soát hành vi mà không thể bị thay đổi mà không có việc khởi động lại toàn bộ máy chủ PostgreSQL. Hơn nữa, một số sửa đổi tham số SET hoặc ALTER đòi hỏi quyền của siêu người sử dụng (superuser).

Lệnh SHOW cho phép kiểm tra các giá trị hiện hành của tất cả các tham số.

Bảng ảo pg_settings (được mô tả trong Phần 45.55) cũng cho phép hiển thị và cập nhật các tham số thời gian chạy của phiên làm việc. Nó là tương đương với SHOW và SET, nhưng có thể thuận tiện hơn để sử dụng vì nó có thể được liên kết với các bảng khác, hoặc được chọn từ việc sử dụng bất kỳ điều kiện lựa chọn mong muốn nào. Nó cũng có nhiều thông tin hơn về các giá trị nào sẽ được phép đối với các tham số đó.

18.2. Vị trí tệp

Ngoài tệp postgresql.conf được nhắc tới ở trên, PostgreSQL sử dụng 2 tệp cấu hình được chỉnh bằng

tay khác, chúng kiểm soát sự xác thực máy trạm (sử dụng chúng được thảo luận trong Chương 19). Mặc định, tất cả 3 tệp cấu hình được lưu trữ trong thư mục dữ liệu của bó cơ sở dữ liệu. Các tham số được mô tả trong phần này cho phép các tệp cấu hình được đặt ở những nơi khác nữa. (Làm như vậy có thể dễ dàng quản trị hơn. Đặc biệt thường dễ dàng hơn để đảm bảo rằng các tệp cấu hình đó được sao lưu đúng khi chúng được giữ tách biệt nhau).

data_directory (string)

Chỉ định thư mục sử dụng để lưu trữ các dữ liệu. Tham số này chỉ có thể được thiết lập khi máy chủ khởi động.

config_file (string)

Chỉ định tệp cấu hình máy chủ chính (thường được gọi là postgresql.conf). Tham số này chỉ có thể được thiết lập trong dòng lệnh postgres.

hba_file (string)

Chỉ định tệp cấu hình cho xác thực dựa vào máy chủ host (thường được gọi là pg_hba.conf). Tham số này chỉ có thể được thiết lập khi máy chủ khởi động.

ident_file (string)

Chỉ định tệp cấu hình cho việc ánh xạ tên người sử dụng ở Phần 19.2 (thường được gọi là pg_ident.conf). Tham số này chỉ có thể được thiết lập khi máy chủ khởi động.

external_pid_file (string)

Chỉ định tên tệp mã tiến trình bổ sung (PID) mà máy chủ sẽ tạo ra để sử dụng từ các chương trình quản trị máy chủ. Tham số này chỉ có thể được thiết lập khi máy chủ khởi động.

Trong một cài đặt mặc định, không tham số nào ở trên được thiết lập rõ ràng rồi. Thay vào đó, thư mục dữ liệu được lựa chọn dòng lệnh -D hoặc biến môi trường PGDATA chỉ định, và các tệp cấu hình đó tất cả sẽ được thấy bên trong thư mục dữ liệu đó.

Nếu bạn muốn giữ các tệp cấu hình ở những nơi khác với thư mục dữ liệu, thì lựa chọn dòng lệnh -D hoặc biến môi trường PGDATA phải chỉ ra thư mục chứa các tệp cấu hình đó, và biến data_directory phải được thiết lập trong postgresql.conf (hoặc trong dòng lệnh) để chỉ ra nơi mà thư mục dữ liệu thực sự được đặt. Lưu ý rằng data_directory sẽ ghi đè -D và PGDATA đối với vị trí của thư mục dữ liệu, nhưng không ghi đè đối với vị trí các tệp cấu hình.

Nếu bạn muốn, bạn có thể chỉ định tên và vị trí các tệp cấu hình một cách riêng rẽ bằng việc sử dụng các tham số config_file, hba_file và/hoặc ident_file. config_file chỉ có thể được chỉ định trong dòng lệnh postgres, nhưng các tham số khác có thể được thiết lập bên trong tệp cấu hình chính. Nếu tất cả 3 tham số cộng với data_directory được thiết lập rõ ràng, thì không cần thiết phải chỉ định -D hoặc PGDATA.

Khi thiết lập bất kỳ tham số nào trong số đó, thì một đường dẫn tương đối sẽ được hiểu là thư mục mà ở đó postgres được khởi tạo.

18.3. Kết nối và xác thực

18.3.1. Thiết lập kết nối

listen_addresses (string)

Chỉ định (các) địa chỉ TCP/IP mà ở đó máy chủ sẽ nghe các kết nối từ các ứng dụng máy trạm. Giá trị đó có dạng một danh sách tách bạch nhau bằng dấu phẩy các tên máy chủ host và/hoặc các địa chỉ IP số. Khoản đầu vào đặc biệt * tương ứng với tất cả các giao diện IP sẵn có. Nếu danh sách này rỗng, thì máy chủ không nghe trên bất kỳ giao diện IP nào cả, trong trường hợp đó chỉ các khe cắm (sockets) miền Unix có thể được sử dụng để kết nối với nó. Giá trị mặc định là localhost, nó cho phép chỉ các kết nối "ngược về nguồn" (loopback) được thực hiện. Trong khi xác thực máy trạm (Chương 19) cho phép kiểm soát tốt ai có thể truy cập máy chủ, thì listen_addresses kiểm soát các giao diện nào chấp nhận các ý định kết nối, chúng có thể giúp ngăn chặn các yêu cầu kết nối độc hại lặp đi lặp lại trong các giao diện mạng không an toàn. Tham số này chỉ có thể được thiết lập khi máy chủ khởi động.

port (integer)

Cổng TCP mà máy chủ nghe; mặc định là 5432. Lưu ý rằng số cổng y hệt được sử dụng cho tất cả các địa chỉ IP mà máy chủ nghe. Tham số này chỉ có thể được thiết lập khi máy chủ khởi động.

max_connections (integer)

Xác định số lượng cực đại các kết nối đồng thời tới máy chủ cơ sở dữ liệu. Mặc định thường là 100 kết nối, nhưng có thể là ít hơn nếu các thiết lập nhân (kernel) của bạn sẽ không hỗ trợ nó (như được xác định trong initdb). Tham số này chỉ có thể được thiết lập khi máy chủ khởi động.

Việc tăng tham số này có thể làm cho PostgreSQL yêu cầu nhiều bộ nhớ hoặc cờ tín hiệu được chia sẻ của System V hơn so với cấu hình mặc định mà hệ điều hành của bạn cho phép. Xem Phần 17.4.1 để có thông tin về cách tinh chỉnh các tham số đó, nếu cần.

Khi chạy một máy chủ dự phòng sẵn (standby), bạn phải thiết lập tham số này về giá trị y hệt hoặc cao hơn so với máy chủ chính (master). Nếu không, các truy vấn sẽ không được phép trong máy chủ dự phòng đó.

superuser_reserved_connections (integer)

Xác định số lượng các "rãnh" (slot) kết nối được dự trữ cho các kết nối của các siêu người sử dụng (superuser) PostgreSQL. Nhiều nhất có max_connections kết nối có thể hoạt động cùng một lúc. Bất kỳ khi nào số các kết nối đồng thời tích cực ít nhất là max_connections trừ đi superuser_reserved_connections, thì các kết nối mới sẽ chỉ được chấp nhận cho các siêu người sử dụng, và không kết nối nhân bản mới nào sẽ được chấp nhận.

Giá trị mặc định là 3 kết nối. Giá trị đó phải ít hơn giá trị của max_connections. Tham số này chỉ có thể thiết lập khi máy chủ khởi động.

unix_socket_directory (string)

Chỉ định thư mục khe cắm (socket) miền Unix mà ở đó máy chủ sẽ nghe các kết nối từ các

ứng dụng máy trạm. Mặc định thường là /tmp, nhưng có thể được thay đổi vào lúc xây dựng. Tham số này chỉ có thể được thiết lập khi máy chủ khởi động.

Hơn nữa bản thân tệp socket có tên là .s.PGSQL.nnnn trong đó nnnn là số cổng của máy chủ, một tệp thông thường có tên là .s.PGSQL.nnnn.lock sẽ được tạo ra trong thư mục unix_socket_directory. Không tệp nào bị loại bỏ bằng tay cả.

Tham số này là không phù hợp trong Windows, nó không có các khe cắm miền Unix.

unix_socket_group (string)

Thiết lập nhóm sở hữu khe cắm miền Unix. (người sử dụng sở hữu khe cắm đó luôn là người sử dụng mà khởi động máy chủ đó). Kết hợp với tham số unix_socket_permissions , điều này có thể được sử dụng như một cơ chế kiểm soát truy cập bổ sung cho các kết nối miền Unix. Mặc định đây là chuỗi rỗng, nó sử dụng nhóm mặc định của người sử dụng máy chủ. Tham số này chỉ có thể thiết lập khi máy chủ khởi động.

Tham số này không phù hợp trong Windows, nó không có các khe cắm miền Unix.

unix_socket_permissions (integer)

Thiết lập các quyền truy cập khe cắm miền Unix. Các khe cắm miền Unix sử dụng tập hợp các quyền của hệ thống tệp Unix thông thường. Giá trị tham số đó được kỳ vọng sẽ là một chế độ số được chỉ định ở định dạng được các lời gọi hệ thống chmod và umask chấp nhận. (Để sử dụng định dạng thông thường cơ số 8 thì các số phải bắt đầu bằng một số 0 [zero]).

Các quyền mặc định là 0777, nghĩa là bất kỳ ai cũng có thể kết nối. Các lựa chọn thay thế hợp lý là 0770 (chỉ người sử dụng và nhóm, xem thêm unix_socket_group) và 0700 (chỉ người sử dụng). (Lưu ý là đối với một khe cắm miền Unix, chỉ có quyền ghi là đáng lưu ý, vì thế không có điểm nào trong việc thiết lập hoặc thu hồi các quyền đọc hoặc thực thi).

Cơ chế kiểm soát truy cập này là độc lập với cơ chế được mô tả trong Chương 19.

Tham số này chỉ có thể được thiết lập khi máy chủ khởi động.

Tham số này là không phù hợp trong Windows, nó không có các khe cắm miền Unix.

bonjour (boolean)

Cho phép quảng cáo sự tồn tại của máy chủ thông qua Bonjour. Mặc định là tắt (off). Tham số này chỉ có thể được thiết lập khi máy chủ khởi động.

bonjour_name (string)

Chỉ định tên dịch vụ Bonjour. Tên máy tính được sử dụng nếu tham số này được thiết lập về chuỗi rỗng " (nó là mặc định). Tham số này bị bỏ qua nếu máy chủ đã không được biên dịch với sự hỗ trợ của Bonjour. Tham số này chỉ có thể được thiết lập khi máy chủ khởi động.

tcp_keepalives_idle (integer)

Chỉ định số giây trước khi gửi một gói giữ cho sống trong một kết nối mà nếu khác là nhàn rỗi. Giá trị 0 sử dụng mặc định của hệ thống. Tham số này chỉ được hỗ trợ trong các hệ thống mà hỗ trợ các ký hiệu TCP_KEEPIDLE hoặc TCP_KEEPALIVE, và trong Windows; trong các hệ thống khác, nó phải bằng 0 (zero). Trong các phiên làm việc được kết nối qua một khe

cắm miền Unix, tham số này bị bỏ qua và luôn được đọc như là zero.

> **Lưu ý**: Trong Windows, một giá trị 0 sẽ thiết lập tham số này thành 2 giờ, vì Windows không đưa ra cách thức để đọc giá trị mặc định của hệ thống.

tcp_keepalives_interval (integer)

Chỉ định số giây giữa việc gửi giữ sống trong một kết nối nhàn rỗi. Giá trị 0 sử dụng mặc định của hệ thống. Tham số này chỉ được hỗ trợ trong các hệ thống mà hỗ trợ ký hiệu TCP_KEEPINTVL, và trong Windows; trong các hệ thống khác, nó phải bằng zero. Trong các phiên làm việc được kết nối qua một khe cắm miền Unix, tham số này bị bỏ qua và luôn được đọc như là zero.

> **Lưu ý**: Trong Windows, một giá trị 0 sẽ thiết lập tham số này thành 2 giờ, vì Windows không đưa ra cách thức để đọc giá trị mặc định của hệ thống.

tcp_keepalives_count (integer)

Chỉ định số các gói giữ sống để gửi trong một kết nối mà nếu không là nhàn rỗi. Giá trị 0 sử dụng mặc định của hệ thống. Tham số này chỉ được hỗ trợ trong các hệ thống mà hỗ trợ ký hiệu TCP_KEEPCNT; trong các hệ thống khác, nó phải là zero. Trong các phiên làm việc được kết nối qua một khe cắm miền Unix, thì tham số này bị bỏ qua và luôn được đọc như là zero.

> **Lưu ý**: Tham số này không được hỗ trợ trong Windows, và phải là zero.

18.3.2. An toàn và xác thực

authentication_timeout (integer)

Thời gian tối đa để hoàn thành xác thực máy trạm, theo giây. Nếu máy có thể là máy trạm đã chưa hoàn tất giao thức xác thực trong thời gian nhiều này, thì máy chủ sẽ đóng kết nối đó. Điều này ngăn chặn các máy trạm bị treo khỏi việc chiếm đoạt một kết nối không xác định. Mặc định là 1 phút. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc dòng lệnh máy chủ.

ssl (boolean)

Cho phép các kết nối SSL. Đọc Phần 17.8 trước khi sử dụng điều này. Mặc định là tắt (off). Tham số này chỉ có thể được thiết lập khi khởi động máy chủ. Giao tiếp SSL chỉ có thể với các kết nối TCP/IP.

ssl_renegotiation_limit (integer)

Chỉ định có bao nhiêu dữ liệu có thể chảy qua một kết nối SSL được mã hóa trước khi sự tái thương lượng các khóa phiên sẽ diễn ra. Sự tái thương lượng làm giảm các cơ hội của một kẻ tấn công khỏi việc thực hiện phân tích mật mã khi lượng giao thông lớn có thể được kiểm tra, nhưng nó cũng mang theo hậu quả lớn cho hiệu năng. Tổng giao thông được gửi và nhận sẽ được sử dụng để kiểm tra giới hạn này. Nếu tham số này được thiết lập về 0, thì sự tái thương thảo bị vô hiệu hóa. Mặc định là 512 MB.

> **Lưu ý**: Các thư viện SSL trước tháng 11/2009 là không an toàn khi sử dụng sự tái thương thảo SSL, vì một chỗ bị tổn thương trong giao thức SSL. Như một sự sửa lỗi đối với chỗ bị tổn thương này, một vài nhà bán hàng đã xuất các thư viện SSL không

có khả năng tiến hành tái thương thảo. Nếu bất kỳ thư viện nào đang được sử dụng ở máy chủ hoặc máy trạm, thì sự tái thương thảo SSL sẽ bị vô hiệu hóa.

ssl_ciphers (string)

Chỉ định một danh sách các mật mã SSL để sử dụng trong các kết nối an toàn. Xem trang chỉ dẫn openssl để có danh sách các mật mã được hỗ trợ. Tham số này là không sẵn có trừ phi máy chủ được biên dịch có hỗ trợ cho SSL.

password_encryption (boolean)

Khi một mật khẩu được chỉ định trong CREATE USER hoặc ALTER USER mà không viết ENCRYPTED hoặc UNENCRYPTED, thì tham số này xác định liệu mật khẩu đó có được mã hóa hay không. Mặc định là bật (on) (mã hóa mật khẩu).

krb_server_keyfile (string)

Thiết lập vị trí tệp khóa máy chủ Kerberos. Xem Phần 19.3.5 hoặc Phần 19.3.3 để có các chi tiết. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

krb_srvname (string)

Thiết lập tên dịch vụ Kerberos. Xem Phần 19.3.5 để có các chi tiết. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

krb_caseins_users (boolean)

Thiết lập liệu các tên người sử dụng của Kerberos và GSSAPI sẽ có được đối xử dạng phân biệt chữ hoa chữ thường hay không. Mặc định là tắt (off) (có phân biệt). Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

db_user_namespace (boolean)

Tham số này cho phép các tên người sử dụng theo từng cơ sở dữ liệu. Mặc định là tắt (off). Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hay trong dòng lệnh máy chủ.

Nếu nó là bật (on), thì bạn nên tạo những người sử dụng như là username@dbname. Khi username được một máy trạm kết nối cho qua, thì @ và tên cơ sở dữ liệu được nối thêm vào tên người sử dụng đó và tên người sử dụng đặc biệt của cơ sở dữ liệu đó được máy chủ đó tra cứu. Lưu ý là khi bạn tạo những người sử dụng có các tên chứa @ trong môi trường SQL, bạn sẽ cần đưa tên người sử dụng đó vào trong các dấu ngoặc kép.

Với tham số này được bật (on), bạn vẫn có thể tạo những người sử dụng toàn thể thông thường. Đơn giản hãy nối @ vào khi chỉ định tên người sử dụng ở máy trạm, như joe@. Ký tự @ sẽ được xóa bỏ trước khi tên người sử dụng đó được máy chủ tra cứu.

db_user_namespace làm cho sự trình bày tên người sử dụng của máy trạm và máy chủ khác nhau. Các kiểm tra xác thực luôn được thực hiện với tên người sử dụng máy chủ nên các phương pháp phải được thiết lập cấu hình cho tên người sử dụng máy chủ, chứ không phải máy trạm. Vì md5 sử dụng tên người sử dụng như là muối trong cả máy trạm và máy chủ, nên md5 không thể được sử dụng với db_user_namespace.

Lưu ý: Tính năng này có ý định như một biện pháp tạm thời cho tới khi một giải

pháp hoàn chỉnh được thấy. Khi đó, lựa chọn này sẽ bị loại bỏ.

18.4. Tiêu xài tài nguyên

18.4.1. Bộ nhớ

shared_buffers (integer)

Thiết lập lượng bộ nhớ mà máy chủ cơ sở dữ liệu sử dụng cho các bộ nhớ đệm (buffer) được chia sẻ. Mặc định thường là 32 MB, nhưng có thể ít hơn nếu các thiết lập nhân của bạn sẽ không hỗ trợ nó (như được xác định trong quá trình initdb). Thiết lập này phải ít nhất là 128 KB. (Các giá trị không mặc định của BLCKSZ làm thay đổi cực tiểu). Tuy nhiên, các thiết lập cao hơn đáng kể so với cực tiểu thường cần tới cho hiệu năng tốt. Tham số này chỉ có thể được thiết lập khi máy chủ khởi động.

Nếu bạn có một máy chủ cơ sở dữ liệu chuyên tâm với 1GB RAM hoặc nhiều hơn, một giá trị hợp lý ban đầu cho shared_buffers là 25% bộ nhớ trong hệ thống của bạn. Có một vài tải công việc nơi mà thậm chí các thiết lập lớn cho shared_buffers là có hiệu quả, nhưng vì PostgreSQL cũng dựa vào bộ nhớ lưu tạm (cache) của hệ điều hành, nên có lẽ không là một sự phân bổ hơn so với 40% lượng RAM cho shared_buffers sẽ làm việc tốt hơn so với một lượng nhỏ hơn. Các thiết lập lớn hơn cho shared_buffers thường đòi hỏi một sự gia tăng tương ứng trong checkpoint_segments, để lan truyền ra tiến trình ghi các số lượng lớn các dữ liệu mới hoặc bị thay đổi qua một khoảng thời gian dài hơn.

Trong các hệ thống với ít hơn 1GB RAM, một tỷ lệ phần trăm nhỏ hơn của RAM là phù hợp, nên để lại chỗ phù hợp cho hệ điều hành. Hơn nữa, trong Windows, các giá trị lớn cho shared_buffers sẽ không hiệu quả. Bạn có thể thấy các kết quả tốt hơn bằng việc giữ cho thiết lập khá thấp và sử dụng bộ nhớ tạm thời (cache) của hệ điều hành nhiều hơn, thay vào đó. Dải hữu dụng cho shared_buffers trong các hệ thống Windows thường là từ 64MB tới 512MB.

Làm gia tăng tham số này có thể làm cho PostgreSQL yêu cầu nhiều bộ nhớ chia sẻ System V hơn so với cấu hình mặc định của hệ điều hành của bạn cho phép. Xem Phần 17.4.1 để có thông tin về cách tinh chỉnh các tham số đó, nếu cần.

temp_buffers (integer)

Thiết lập số lớn nhất bộ nhớ đệm được từng phiên làm việc của cơ sở dữ liệu sử dụng. Có các bộ nhớ đệm phiên cục bộ chỉ được sử dụng để truy cập tới các bảng tạm thời. Mặc định là 8MB. Thiết lập đó có thể bị thay đổi trong các phiên làm việc riêng rẽ, nhưng chỉ trước khi sử dụng đầu tiên các bảng tạm trong phiên làm việc đó; các cố gắng sau đó để thay đổi giá trị này sẽ không có hiệu quả trong phiên làm việc đó.

Phiên làm việc sẽ phân bổ các bộ nhớ tạm như cần có cho tới giới hạn được temp_buffers đưa ra. Chi phí của việc thiết lập một giá trị lớn trong các phiên làm việc mà không thực sự cần nhiều bộ nhớ tạm chỉ là một ký hiệu bộ nhớ tạm, hoặc khoảng 64 byte, cho từng sự tăng dần trong temp_buffers. Tuy nhiên nếu một bộ nhớ tạm thực sự được sử dụng thì 8192 byte

bổ sung sẽ được tiêu dùng cho nó (hoặc nói chung, BLCKSZ byte).

max_prepared_transactions (integer)

Thiết lập số tối đa các giao dịch có thể nằm trong tình trạng "được chuẩn bị" cùng một lúc (xem CHUẨN BỊ GIAO DỊCH). Việc thiết lập tham số này về zero (là mặc định) sẽ vô hiệu hóa tính năng giao dịch được chuẩn bị. Tham số này chỉ có thể được thiết lập khi máy chủ khởi động.

Nếu bạn không lên kế hoạch sử dụng các giao dịch được chuẩn bị, thì tham số này sẽ được thiết lập về zero để ngăn chặn ngẫu nhiên tạo ra các giao dịch được chuẩn bị. Nếu bạn đang sử dụng các giao dịch được chuẩn bị, thì bạn có lẽ sẽ muốn max_prepared_transactions ít nhất sẽ là lớn như max_connections, sao cho mỗi phiên làm việc có thể có một giao dịch được chuẩn bị đang chờ.

Việc gia tăng tham số này có thể làm cho PostgreSQL yêu cầu nhiều bộ nhớ được chia sẻ System V hơn so với cấu hình mặc định hệ điều hành của bạn cho phép. Xem Phần 17.4.1 để có thông tin về cách tinh chỉnh các tham số đó, nếu cần.

Khi chạy một máy chủ nhàn rỗi, bạn phải thiết lập tham số này về giá trị y hệt hoặc cao hơn so với máy chủ chính (master). Nếu không, các truy vấn sẽ không được phép trong máy chủ nhàn rỗi.

work_mem (integer)

Chỉ định lượng bộ nhớ sẽ được các hoạt động sắp xếp nội bộ và các bảng băm sử dụng trước khi ghi các tệp tạm thời của đĩa. Giá trị đó mặc định là 1MB. Lưu ý rằng đối với một truy vấn phức tạp, vài hoạt động sắp xếp hoặc băm có thể đang chạy song song; một hoạt động sẽ được phép sử dụng lượng bộ nhớ như giá trị này chỉ định trước khi nó bắt đầu ghi dữ liệu vào các tệp tạm thời. Hơn nữa, vài phiên làm việc đang chạy có thể đang thực hiện các hoạt động đó một cách đồng thời. Vì thế tổng bộ nhớ được sử dụng có thể là nhiều lần giá trị của work_mem; cần thiết để giữ yếu tố này trong đầu khi chọn giá trị đó. Các hoạt động sắp xếp sẽ được sử dụng cho ORDER BY, DISTINCT và các liên kết trộn. Các bảng băm sẽ được sử dụng trong các liên kết băm, tổng hợp dựa vào băm, và việc xử lý các truy vấn con IN dựa vào băm.

maintenance_work_mem (integer)

Chỉ định lượng cực đại bộ nhớ sẽ được các hoạt động duy trì sử dụng, như VACUUM, CREATE INDEX, và ALTER TABLE ADD FOREIGN KEY. Nó mặc định 16MB. Vì chỉ một trong các hoạt động đó có thể được thực thi ở một thời điểm của một phiên làm việc cơ sở dữ liệu, và một cài đặt thường không có nhiều chúng chạy một cách đồng thời, nên sẽ là an toàn để thiết lập giá trị này lớn hơn đáng kể so với work_mem. Các thiết lập lớn hơn có thể cải thiện hiệu năng đối với việc hút và việc phục hồi các hỏng hóc cơ sở dữ liệu.

Lưu ý rằng khi sự hút chân không tự động chạy, cho tới các thời gian autovacuum_max_workers thì bộ nhớ này có thể được phân bổ, nên hãy cẩn thận không thiết lập giá trị mặc định quá cao.

max_stack_depth (integer)

Chỉ định độ sâu an toàn tối đa của hàng đợi thực thi của máy chủ. Thiết lập lý tưởng cho tham số này là giới hạn kích cỡ thực được nhân ép tuân thủ (được thiết lập bằng ulimit -s hoặc tương đương nội bộ), ít hơn ngưỡng an toàn khoảng 1MB. Ngưỡng an toàn là cần thiết vì độ sâu hàng đợi không được kiểm tra theo từng thủ tục trong máy chủ, mà chỉ trong các thủ tục tiềm tàng đệ quy chính như đánh giá biểu thức. Thiết lập mặc định là 2MB, nó là nhỏ dè dặt và có lẽ không mạo hiểm đổ vỡ. Tuy nhiên, nó có thể là quá nhỏ để cho phép thực thi các hàm phức tạp. Chỉ các siêu người sử dụng mới có thể thay đổi thiết lập này.

Thiết lập max_stack_depth cao hơn giới hạn nhân thực ngụ ý là một hàm đệ quy được đưa ra có thể làm hỏng một tiến trình riêng rẽ ở phần phụ trợ (backend). Trong các nền tảng nơi mà PostgreSQL có thể xác định giới hạn nhân, thì máy chủ sẽ không cho phép biến này sẽ được thiết lập về một giá trị không an toàn. Tuy nhiên, không phải tất cả các nền tảng đưa ra thông tin, nên cảnh báo được khuyến khích trong việc lựa chọn một giá trị.

18.4.2. Sử dụng tài nguyên của nhân

max_files_per_process (integer)

Thiết lập số cực đại các tệp mở cùng một lúc được phép đối với từng tiến trình con của máy chủ. Mặc định là 1.000 tệp. Nếu nhân đang ép một giới hạn theo từng tiến trình an toàn, thì bạn không cần lo lắng về thiết lập này. Nhưng trong một số nền tảng (ấy là, hầu hết các hệ thống BSD), nhân sẽ cho phép các tiến trình riêng rẽ mở nhiều tệp hơn so với hệ thống có thể thực sự hỗ trợ nếu nhiều tiến trình tất cả đều cố mở nhiều tệp đó. Nếu bạn tự thấy việc khi xem những thất bại của "quá nhiều tệp được mở", thì hãy cố gắng giảm thiết lập này. Tham số này chỉ có thể được thiết lập khi máy chủ khởi động.

shared_preload_libraries (string)

Biến này chỉ định một hoặc nhiều thư viện được chia sẻ để được tải lên trước khi máy chủ khởi động. Ví dụ, '$libdir/mylib' có thể làm cho mylib.so (hoặc trên vài nền tảng, mylib.sl) sẽ được tải lên trước từ thư mục thư viện tiêu chuẩn của cài đặt. Tất cả các tên thư viện được biến đổi sang chữ thường trừ phi có các dấu ngoặc kép. Nếu nhiều hơn một thư viện sẽ được tải lên, thì hãy tách bạch các tên của chúng bằng dấu phẩy. Tham số này chỉ có thể được thiết lập khi máy chủ khởi động.

Các thư viện ngôn ngữ thủ tục của PostgreSQL có thể được tải lên trước theo cách này, điển hình bằng việc sử dụng cú pháp '$libdir/plXXX' nơi mà XXX là pgsql, perl, tcl, hoặc python.

Bằng việc tải lên trước một thư viện được chia sẻ, thời gian khởi tạo thư viện tránh được khi thư viện lần đầu tiên được sử dụng. Tuy nhiên, thời gian để khởi tạo từng tiến trình mới của máy chủ có thể gia tăng một chút, thậm chí nếu tiến trình đó không bao giờ sử dụng thư viện đó. Vì thế tham số này chỉ được khuyến cáo cho các thư viện sẽ được sử dụng trong hầu hết các phiên làm việc.

Lưu ý: Trong Windows, việc tải lên trước một thư viện khi máy chủ khởi động sẽ không làm giảm thời gian được yêu cầu để khởi động từng tiến trình mới; mỗi tiến

trình của máy chủ sẽ tải lên tất cả các thư viện được tải lên trước. Tuy nhiên, các thư viện tải lên trước được chia sẻ vẫn là hữu dụng trong các máy chủ host Windows vì một số thư viện được chia sẻ có thể cần thực hiện các hoạt động nhất định mà chỉ diễn ra khi chủ sự (postmaster) khởi động (ví dụ, một thư viện được chia sẻ có thể cần dự trữ các khóa nhẹ hoặc bộ nhớ được chia sẻ và bạn không thể làm điều đó sau khi chủ sự đã khởi động.

Nếu một thư viện được chỉ định không được tìm thấy, thì máy chủ sẽ không khởi động được.

Mỗi thư viện PostgreSQL được hỗ trợ có một "khối ma thuật" được kiểm tra để đảm bảo tính tương thích. Vì lý do này, các thư viện không phải PostgreSQL không thể được tải lên theo cách này.

18.4.3. Hút chân không trễ dựa vào chi phí

Trong khi thực thi các lệnh VACUUM và ANALYZE, hệ thống duy trì một bộ đếm nội bộ theo dõi chi phí ước tính của các hoạt động I/O khác nhau được thực hiện. Khi chi phí cộng dồn đạt tới một giới hạn (được vacuum_cost_limit chỉ định), thì tiến trình thực hiện hoạt động đó sẽ ngủ trong một khoảng thời gian ngắn, như được vacuum_cost_delay chỉ định. Sau đó nó sẽ khởi động lại bộ đếm và tiếp tục thực thi.

Ý định của tính năng này là để cho phép các quản trị viên giảm tác động của I/O đối với các lệnh đó lên hoạt động cùng một lúc của cơ sở dữ liệu. Có nhiều tình huống nơi mà không quan trọng duy trì các lệnh như VACUUM và ANALYZE kết thúc nhanh chóng; tuy nhiên, thường rất quan trọng rằng các lệnh đó không can thiệp đáng kể bằng khả năng của hệ thống thực hiện các hoạt động khác của cơ sở dữ liệu. Độ trễ vacuum dựa vào chi phí đưa ra một cách thức cho những người quản trị để đạt được điều này.

Tính năng này mặc định bị vô hiệu hóa đối với các lệnh VACUUM được đưa ra bằng tay. Để kích hoạt nó, hãy thiết lập biến vacuum_cost_delay về một giá trị không phải zero.

vacuum_cost_delay (integer)

Độ dài thời gian, theo mili giây, mà tiến trình sẽ ngủ khi giới hạn chi phí bị vượt quá. Giá trị mặc định là zero, nó vô hiệu hóa tính năng trễ vacuum dựa vào chi phí. Các giá trị dường như cho phép việc vacuum dựa vào chi phí. Lưu ý là trong nhiều hệ thống, quyết định trễ ngủ tích cực là 10 mili giây; việc thiết lập vacuum_cost_delay về một giá trị không phải là bội số của 10 có thể có kết quả y hệt như việc thiết lập nó về bội số của 10 cao hơn tiếp theo.

Khi sử dụng vacuum dựa vào chi phí, các giá trị phù hợp cho vacuum_cost_delay thường rất nhỏ, có lẽ là 10 hoặc 20 mili giây. Việc tinh chỉnh sự tiêu dùng tài nguyên của vacuum được thực hiện tốt nhất bằng việc thay đổi các tham số chi phí vacuum khác.

vacuum_cost_page_hit (integer)

Chi phí ước tính cho việc vacuum một bộ nhớ đệm (buffer) được thấy trong bộ nhớ tạm (cache) được chia sẻ. Nó đại diện cho chi phí để khóa kho bộ nhớ tạm, tra bảng băm được chia sẻ và quét nội dung của trang. Giá trị mặc định là 1.

vacuum_cost_page_miss (integer)

Chi phí ước tính cho việc vacuum một bộ nhớ tạm mà phải đọc được từ đĩa. Điều này thể hiện nỗ lực để khóa kho bộ nhớ đệm, tra cứu bảng băm, đọc khối mong muốn từ đĩa và quét nội dung của nó. Giá trị mặc định là 10.

vacuum_cost_page_dirty (integer)

Chi phí ước tính được lấy khi vacuum sửa một khối mà trước đó từng là làm sạch. Nó thể hiện I/O thêm được yêu cầu để san bằng khối bẩn ra khỏi đĩa một lần nữa. Giá trị mặc định là 20.

vacuum_cost_limit (integer)

Chi phí cộng dồn mà sẽ làm cho tiến trình vacuum ngủ. Giá trị mặc định là 200.

Lưu ý: Có các hoạt động nhất định mà giữ các khóa sống còn và vì thế sẽ hoàn chỉnh nhanh nhất có thể. Độ trễ vacuum dựa vào chi phí không xảy ra trong các hoạt động như vậy. Vì thế có khả năng là chi phí cộng dồn đó cao hơn nhiều so với giới hạn được chỉ định. Để tránh các trễ dài vô dụng trong các trường hợp như vậy, trễ thực sự được tính như là vacuum_cost_delay * accumulated_balance/ vacuum_cost_limit với cực đại của vacuum_cost_delay * 4.

18.4.4. Trình ghi nền (Background Writer)

Có một tiến trình máy chủ tách biệt được gọi là người ghi nền (background writer), chức năng của nó là để đưa ra việc ghi của các bộ nhớ đệm được chia sẻ "bẩn" (mới hoặc được sửa đổi). Nó ghi các bộ nhớ đệm được chia sẻ sao cho các tiến trình của máy chủ giữ các truy vấn của người sử dụng hiếm khi hoặc không bao giờ cần phải chờ đợi một sự ghi xảy ra. Tuy nhiên, người ghi nền gây ra một sự gia tăng tổng thể trong tải I/O, vì trong khi một trang bị bẩn lặp đi lặp lại có thể nếu khác chỉ được ghi một lần cho từng khoảng thời gian kiểm tra dấu, thì người ghi nền có thể ghi nó vài lần giống như là nó bị bẩn trong khoảng thời gian y hệt đó. Các tham số được thảo luận trong phần con này có thể được sử dụng để tinh chỉnh hành vi cho các nhu cầu cục bộ.

bgwriter_delay (integer)

Chỉ định sự trễ giữa các vòng hoạt động đối với người ghi nền. Trong từng vòng người ghi đưa các các cuộc ghi cho một số bộ nhớ đệm bẩn (kiểm tra được bằng các tham số sau). Nó sau đó ngủ trong bgwriter_delay mili giây, và lặp lại. Giá trị mặc định là 200 mili giây. Lưu ý là trong nhiều hệ thống, quyết định hiệu quả của các trễ ngủ là 10 mili giây; việc thiết lập bgwriter_delay về một giá trị không phải là bội số của 10 có thể có các kết quả y hệt như việc thiết lập nó về bội số của 10 cao hơn tiếp theo. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh của máy chủ.

bgwriter_lru_maxpages (integer)

Trong từng vòng, không nhiều hơn nhiều bộ nhớ đệm này sẽ được người ghi nền ghi. Việc thiết lập điều này về zero sẽ vô hiệu hóa việc ghi nền (ngoại trừ vì hoặc động của điểm kiểm tra). Giá trị mặc định là 100 bộ nhớ đệm. Tham số này chỉ có thể được thiết lập trong tệp

postgresql.conf hoặc trong dòng lệnh của máy chủ.

bgwriter_lru_multiplier (floating point)

Số lượng các bộ nhớ đệm bẩn được ghi trong từng vòng dựa vào số lượng các bộ nhớ đệm mới từng là cần thiết đối với các tiến trình của máy chủ trong các vòng gần đây. Nhu cầu trung bình gần đây được nhân với bgwriter_lru_multiplier để tới được một ước tính số lượng các bộ nhớ đệm mà sẽ là cần thiết trong vòng tiếp sau. Các bộ nhớ đệm bẩn được ghi cho tới khi có nhiều bộ nhớ đệm sử dụng lại được, sạch sẵn sàng. (Tuy nhiên, không lớn hơn bgwriter_lru_maxpages bộ nhớ đệm sẽ được ghi theo từng vòng). Vì thế, một thiết lập của 1.0 đại diện cho một chính sách "vừa đúng lúc" của việc ghi chính xác số lượng các bộ nhớ đệm được dự đoán trước là cần thiết. Các giá trị lớn hơn đưa ra một vài miếng đệm đối với các nhánh theo yêu cầu, trong khi các giá trị nhỏ hơn có chủ ý để lại các cuộc ghi sẽ được các tiến trình máy chủ thực hiện. Mặc định là 2.0. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

Các giá trị nhỏ hơn của bgwriter_lru_maxpages và bgwriter_lru_multiplier làm giảm tải I/O thêm do người ghi nền gây ra, nhưng làm cho nó có khả năng hơn là các tiến trình máy chủ sẽ phải đưa ra các cuộc ghi cho bản thân chúng, làm trễ các truy vấn tương tác.

18.4.5. Hành xử không đồng bộ

effective_io_concurrency (integer)

Thiết lập số lượng các hoạt động I/O của đĩa đồng thời xảy ra mà PostgreSQL kỳ vọng có thể được thực thi cùng lúc. Việc tăng giá trị này sẽ làm tăng số lượng các hoạt động I/O mà bất kỳ phiên làm việc riêng rẽ nào của PostgreSQL cũng cố khởi tạo song song. Dải được phép từ 1 tới 1.000, hoặc zero để vô hiệu hóa sự đưa ra các yêu cầu I/O không đồng bộ.

Một điểm khởi đầu tốt cho thiết lập này là số lượng các ổ riêng rẽ tạo thành một dải RAID 0 hoặc gương soi RAID 1 đang được sử dụng cho cơ sở dữ liệu. (Đối với RAID 5 thì ổ chẵn lẻ [parity] sẽ không được tính). Tuy nhiên, nếu cơ sở dữ liệu thường bận rộn với nhiều truy vấn được đưa ra trong các phiên làm việc đồng thời, thì các giá trị thấp hơn có thể là đủ để giữ cho đống đĩa bận rộn. Một giá trị cao hơn cần thiết để giữ cho các đĩa bận rộn sẽ chỉ làm tăng tổng chi phí dư thừa của CPU.

Đối với các hệ thống kỳ lạ hơn, như lưu trữ dựa vào bộ nhớ hoặc một đống RAID mà có giới hạn vì băng rộng của bus, thì giá trị đúng có thể là số đường I/O sẵn sàng. Một vài thử nghiệm có thể là cần thiết để tìm ra giá trị tốt nhất.

I/O không đối xứng phụ thuộc vào một hàm posix_fadvise có hiệu quả, mà việc vận hành một vài hệ thống còn thiếu. Nếu hàm đó không tồn tại thì việc thiết lập tham số này về bất kỳ giá trị nào khác zero sẽ gây ra một lỗi. Trong một số hệ điều hành (như Solaris), hàm đó tồn tại nhưng không thực sự làm bất kỳ điều gì.

18.5. Ghi trước lưu ký

Xem Phần 29.4 để có các chi tiết về WAL và tinh chỉnh điểm kiểm tra (checkpoint).

18.5.1. Thiết lập

wal_level (enum)

wal_level xác định có bao nhiêu thông tin được ghi vào WAL. Giá trị mặc định là minimal, nó chỉ ghi thông tin cần thiết để phục hồi từ một hỏng hóc hoặc tắt tức thì. archive thêm việc lưu ý được yêu cầu cho việc lưu trữ WAL, và hot_standby bổ sung thêm thông tin được yêu cầu để chạy các truy vấn chỉ đọc trong một máy chủ nhàn rỗi. Tham số này chỉ có thể được thiết lập khi máy chủ khởi động.

Ở mức minimal, việc lưu ký bằng WAL của một vài đống hoạt động có thể an toàn bỏ qua, nó có thể làm cho các hoạt động đó nhanh hơn nhiều (xem Phần 14.4.7). Các hoạt động mà ở đó sự tối ưu hóa này có thể được áp dụng bao gồm:

CREATE TABLE AS
CREATE INDEX
CLUSTER
COPY vào các bảng từng được tạo ra hoặc bị cắt ngắn bớt trong cùng giao dịch y hệt

Nhưng WAL tối thiểu không có đủ thông tin để tái xây dựng lại các dữ liệu từ một sao lưu cơ sở và các lưu ký WAL, nên hoặc mức archive hoặc hot_standby phải được sử dụng để cho phép lưu trữ WAL (archive_mode) và sắp xếp sự nhân bản.

Ở mức hot_standby, thông tin y hệt được lưu ký như với archive, cộng với thông tin cần thiết để tái xây dựng tình trạng chạy các giao dịch từ WAL. Để cho phép các truy vấn chỉ đọc trong một máy chủ nhàn rỗi, wal_level phải được thiết lập về hot_standby ngay từ đầu, và hot_standby phải được kích hoạt ở chế độ nhàn rỗi (standby). Được cho là có ít sự khác biệt đo đếm được trong hiệu năng giữa việc sử dụng các mức hot_standby và archive, nên ý kiến phản hồi được chào đón nếu bất kỳ ảnh hưởng nào tới sản xuất được lưu ý.

fsync (boolean)

Nếu tham số này được bật, thì máy chủ PostgreSQL sẽ cố gắng chắc chắn rằng các bản cập nhật được ghi một cách vật lý tới đĩa, bằng việc đưa ra các lời gọi hệ thống fsync() hoặc các phương pháp tương đương khác (xem wal_sync_method). Điều này đảm bảo rằng bó cơ sở dữ liệu có thể phục hồi về một tình trạng ổn định sau một sự hỏng hóc hệ điều hành hoặc phần cứng.

Trong khi tắt đi fsync thường có lợi cho hiệu năng, thì điều này có thể làm cho hỏng dữ liệu không thể phục hồi trong trường hợp mất điện hoặc hỏng hệ thống. Vì thế chỉ được khuyến cáo tắt fsync nếu bạn có thể dễ dàng tái tạo được toàn bộ cơ sở dữ liệu của bạn từ các dữ liệu bên ngoài.

Các ví dụ về các hoàn cảnh an toàn cho việc tắt fsync bao gồm việc tải ban đầu bó cơ sở dữ liệu mới từ một tệp sao lưu, việc sử dụng một bó cơ sở dữ liệu cho việc xử lý một bó các dữ liệu sau đó cơ sở dữ liệu sẽ được bỏ đi và được tái tạo lại, hoặc cho một sự nhái cơ sở dữ liệu chỉ đọc mà nó sẽ được tái tạo thường xuyên và không được sử dụng cho việc chuyển đổi sang dự phòng (failover). Chỉ một mình phần cứng chất lượng là không đủ chứng minh cho việc tắt fsync.

Trong nhiều tình huống, việc tắt synchronous_commit đối với các giao dịch không sống còn có thể đưa ra nhiều lợi ích tiềm tàng cho hiệu năng của việc tắt fsync, không có các rủi ro hỏng dữ liệu kèm theo.

fsync chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ. Nếu bạn tắt tham số này, cũng hãy cân nhắc tắt full_page_writes.

synchronous_commit (boolean)

Chỉ định liệu đề xuất giao dịch sẽ có chờ hay không cho các bản ghi của WAL sẽ được ghi vào đĩa trước khi lệnh đó trả về một dấu hiệu "thành công" cho máy trạm. Mặc định, và an toàn, thiết lập này là bật (on). Khi tắt (off) có thể có độ trễ ở giữa khi sự thành công được ghi nhận đối với máy trạm hoặc khi giao dịch được thực sự đảm bảo là an toàn đối với sự hỏng hóc của một máy chủ. (Sự trễ cực đại là gấp 3 lần wal_writer_delay). Không giống như fsync, việc thiết lập tham số này về tắt (off) không tạo ra bất kỳ rủi ro nào đối với sự không ổn định của cơ sở dữ liệu: một sự hỏng hóc hệ điều hành hoặc cơ sở dữ liệu có thể làm cho một vài giao dịch được cho là đã được thực hiện gần đây bị mất, nhưng tình trạng của cơ sở dữ liệu sẽ vẫn là y hệt dường như các giao dịch đó đã bị loại bỏ một cách sạch sẽ. Vì thế việc tắt synchronous_commit có thể là lựa chọn thay thế hữu dụng khi hiệu năng là quan trọng hơn sự chắc chắn chính xác về tính lâu bền của một giao dịch. Xem Phần 29.3 để thảo luận thêm.

Tham số này có thể bị thay đổi bất kỳ lúc nào; hành vi đó đối với bất kỳ một giao dịch nào cũng được xác định bằng việc thiết lập có hiệu quả khi nó thực hiện. Vì thế có khả năng, và hữu dụng, để có một vài giao dịch thực hiện đồng bộ và vài giao dịch khác không đồng bộ. Ví dụ, để tiến hành một giao dịch duy nhất đa tình trạng hãy thực hiện không đồng bộ khi mặc định là ngược lại, đưa SET LOCAL synchronous_commit VỀ OFF bên trong giao dịch đó.

wal_sync_method (enum)

Phương pháp được sử dụng cho việc ép WAL cập nhật hết tới đĩa. Nếu fsync là tắt thì thiết lập này là không phù hợp, vì các cập nhật tệp WAL sẽ không bị ép hết hoàn toàn. Các giá trị có thể là:

- open_datasync (ghi các tệp WAL với lựa chọn open() cho O_DSYNC)
- fdatasync (gọi fdatasync() ở từng đệ trình thực hiện)
- fsync (gọi fsync() ở từng đệ trình thực hiện)
- fsync_writethrough (gọi fsync() ở từng đệ trình thực hiện, ép ghi qua bất kỳ bộ nhớ tạm ghi đĩa nào)
- open_sync (ghi các tệp WAL bằng lựa chọn open() cho O_SYNC)

Các lựa chọn open_ * cũng sử dụng O_DIRECT nếu sẵn có. Không phải tất cả các lựa chọn đó là sẵn có trong tất cả các nền tảng. Mặc định là phương pháp đầu tiên trong danh sách ở trên mà nó được nền tảng đó hỗ trợ, ngoại trừ fdatasync là mặc định trong Linux. Mặc định không nhất thiết là lý tưởng; có thể cần thiết phải thay đổi thiết lập này hoặc các khía cạnh

khác của cấu hình hệ thống của bạn để tạo ra một cấu hình an toàn tránh hỏng hóc hoặc đạt được hiệu năng tối ưu. Các khía cạnh đó được thảo luận trong Phần 29.1. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

full_page_writes (boolean)

Khi tham số này được bật, máy chủ PostgreSQL sẽ ghi toàn bộ nội dung của từng trang đĩa tới WAL trong quá trình sửa đổi đầu tiên trang đó sau một điểm kiểm tra (checkpoint). Điều này là cần thiết vì một trang ghi điều đó là trong tiến trình khi một hỏng hóc hệ điều hành có thể chỉ một phần được hoàn tất, dẫn tới một trang trên đĩa mà có chứa một sự pha trộn giữa các dữ liệu cũ và mới. Dữ liệu thay đổi mức hàng thường được lưu trữ trong WAL sẽ không đủ để phục hồi hoàn toàn một trang như vậy trong quá trình phục hồi sau hỏng hóc. Việc lưu trữ ảnh của toàn bộ trang đó đảm bảo rằng trang đó có thể được phục hồi đúng, nhưng với cái giá của việc làm gia tăng lượng dữ liệu phải được ghi vào WAL. (Vì WAL luôn làm lại từ đầu cho tới một điểm kiểm tra, là đủ để thực hiện điều này trong quá trình của sự thay đổi đầu tiên của từng trang sau một điểm kiểm tra. Vì thế, một cách để làm giảm chi phí của các cuộc ghi toàn bộ trang là hãy gia tăng các tham số khoảng thời gian của điểm kiểm tra).

Việc tắt tham số này làm tăng tốc hoạt động bình thường, nhưng có thể dẫn tới hoặc hỏng dữ liệu không thể phục hồi, hoặc hỏng dữ liệu âm thầm, sau một hỏng hóc hệ thống. Các rủi ro đó là tương tự như với việc tắt fsync, dù nhỏ hơn, và nó sẽ được tắt chỉ dựa vào các hoàn cảnh y hệt được khuyến cáo cho tham số đó.

Việc tắt tham số này không ảnh hưởng tới sử dụng lưu trữ WAL đối với sự phục hồi đúng thời điểm - PITR (Point-In-Time Recovery) (xem Phần 24.3).

Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ. Mặc định là bật (on).

wal_buffers (integer)

Lượng bộ nhớ được sử dụng trong bộ nhớ được chia sẻ cho các dữ liệu WAL. Mặc định là 64KB. Việc thiết lập chỉ cần đủ lớn để duy trì lượng dữ liệu WAL được một giao dịch điển hình sinh ra, vì dữ liệu đó được ghi hết vào đĩa ở từng đệ trình thực hiện giao dịch. Tham số này chỉ có thể được thiết lập khi máy chủ khởi động.

Việc gia tăng tham số này có thể làm cho PostgreSQL yêu cầu nhiều bộ nhớ được chia sẻ System V hơn so với cấu hình mặc định của hệ điều hành của bạn cho phép. Xem Phần 17.4.1 để có thông tin về cách tinh chỉnh các tham số đó, nếu cần.

wal_writer_delay (integer)

Chỉ định sự trễ giữa các vòng hoạt động cho người ghi WAL. Trong từng vòng thì người ghi sẽ đẩy WAL vào đĩa. Nó sau đó sẽ ngủ trong wal_writer_delay mili giây, và lặp lại. Giá trị mặc định là 200 mili giây. Lưu ý là trong nhiều hệ thống, quyết định trễ ngủ có hiệu quả là 10 mili giây; việc thiết lập wal_writer_delay về giá trị mà không là bội số của 10 có thể có các kết quả y hệt như việc thiết lập nó về bội số của 10 cao hơn tiếp theo. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh của máy chủ.

commit_delay (integer)

Trễ thời gian giữa việc ghi một bản ghi được thực hiện tới bộ nhớ đệm WAL và đẩy bộ nhớ đệm đó hết vào đĩa, trong vài mili giây. Một sự trễ không phải zero có thể cho phép nhiều giao dịch sẽ được thực hiện với chỉ một lời gọi hệ thống fsync(), nếu tải hệ thống là đủ cao mà các giao dịch bổ sung thêm trở thành sẵn sàng để thực hiện bên trong khoảng thời gian được đưa ra đó. Nhưng sự trễ đó chỉ là uổng công nếu không có các giao dịch nào khác trở nên sẵn sàng để thực hiện. Vì thế, sự trễ đó chỉ được thực hiện nếu ít nhất commit_siblings các giao dịch khác là tích cực tức thì mà một tiến trình máy chủ đã ghi bản ghi thực hiện của nó. Mặc định là zero (không có trễ).

commit_siblings (integer)

Số lượng tối thiểu các giao dịch mở đồng thời để yêu cầu trước việc thực thi trễ commit_delay. Một giá trị lớn hơn làm cho nó có khả năng xảy ra hơn là ít nhất một giao dịch khác sẽ trở nên sẵn sàng để thực hiện trong quá trình đối với khoảng trễ đó. Mặc định là 5 giao dịch.

18.5.2. Điểm kiểm tra (checkpoint)

checkpoint_segments (integer)

Số lượng tối đa các phân đoạn tệp lưu ký giữa các điểm kiểm tra tự động WAL (từng đoạn thường là 16MB). Mặc định là 3 đoạn. Việc gia tăng tham số này có thể làm gia tăng lượng thời gian cần thiết cho sự phục hồi hỏng hóc. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc hoặc trong dòng lệnh máy chủ.

checkpoint_timeout (integer)

Thời gian tối đa giữa các điểm kiểm tra tự động WAL, theo giây. Mặc định là 5 phút. Việc gia tăng tham số này có thể làm gia tăng lượng thời gian cần thiết cho phục hồi hỏng hóc. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

checkpoint_completion_target (floating point)

Chỉ định đích của sự hoàn tất điểm kiểm tra, như một phần của tổng thời gian giữa các điểm kiểm tra. Mặc định là 0.5. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

checkpoint_warning (integer)

Hãy viết một thông điệp vào lưu ký máy chủ nếu các điểm kiểm tra được tạo ra bằng việc điền các tệp của đoạn các điểm kiểm tra xảy ra gần nhau hơn so với nhiều giây này (nó gợi ý rằng checkpoint_segments phải được nâng lên). Mặc định là 30 giây. Zero vô hiệu hóa việc cảnh báo đó. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

18.5.3. Lưu trữ

archive_mode (boolean)

Khi archive_mode được kích hoạt, các phân đoạn hoàn chỉnh của WAL được gửi tới kho lưu trữ bằng việc thiết lập archive_command. archive_mode và archive_command là các biến tách

bạch sao cho archive_command có thể được thay đổi mà không rời khỏi chế độ lưu trữ. Tham số này chỉ có thể được thiết lập khi máy chủ khởi động. wal_level phải được thiết lập về archive hoặc hot_standby để kích hoạt archive_mode.

archive_command (string)

Lệnh biên dịch shell để thực thi cho lưu trữ một phân đoạn tệp hoàn chỉnh WAL. Bất kỳ %p nào trong chuỗi đó cũng được tên đường dẫn của tệp lưu trữ thay thế, và bất kỳ %f nào cũng chỉ được tên tệp đó thay thế. (Tên đường dẫn là tương đối cho thư mục làm việc của máy chủ, như, thư mục dữ liệu của bó đó). Hãy sử dụng %% để nhúng một ký tự thực sự % vào lệnh đó. Điều quan trọng đối với lệnh đó để trả về tình trạng thoát ra zero chỉ nếu nó thành công. Xem Phần 24.3.1 để có thêm thông tin.

Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ. Nó bị bỏ qua trừ phi archive_mode được kích hoạt khi máy chủ khởi động. Nếu archive_command là một chuỗi rỗng (mặc định) trong khi archive_mode được kích hoạt, việc lưu trữ WAL bị vô hiệu hóa tạm thời, nhưng máy chủ vẫn tiếp tục tích lũy các tệp của phân đoạn WAL với kỳ vọng rằng một lệnh sẽ sớm được đưa ra. Việc thiết lập archive_command về một lệnh mà không làm gì ngoài việc trả về đúng (true), như /bin/true (REM trên Windows), sẽ vô hiệu hóa một cách có hiệu quả việc lưu trữ, nhưng cũng phá vỡ chuỗi tệp WAL cần thiết cho sự phục hồi lưu trữ, vì thế nó sẽ chỉ được sử dụng trong các hoàn cảnh bất thường.

archive_timeout (integer)

archive_command chỉ được triệu hồi đối với các phân đoạn hoàn chỉnh WAL. Vì thế, nếu máy chủ của bạn sinh ra một chút giao thông WAL (hoặc có các giai đoạn chậm chạp nơi mà nó làm thế), thì có thể có một sự trễ lâu giữa sự hoàn tất một giao dịch và việc ghi an toàn trong kho lưu trữ. Để hạn chế cách mà các dữ liệu cũ không được lưu trữ có thể xảy ra, bạn có thể thiết lập archive_timeout để ép máy chủ đó chuyển sang tệp phân đoạn WAL mới một cách định kỳ. Khi tham số này lớn hơn zero, máy chủ sẽ chuyển tới một tệp phân đoạn mới bất kỳ khi nào nhiều giây này đã trôi qua kể từ khi chuyển tệp phân đoạn cuối cùng, và từng có bất kỳ hoạt động cơ sở dữ liệu nào, bao gồm cả một điểm kiểm tra độc nhất. (Việc gia tăng checkpoint_timeout sẽ làm giảm các điểm kiểm tra không cần thiết trong một hệ thống nhàn rỗi). Lưu ý rằng các tệp được lưu trữ mà sẽ bị đóng sớm vì một sự chuyển bị ép buộc vẫn có độ dài y hệt như các tệp hoàn toàn đầy đủ. Vì thế, sẽ là không khôn ngoan để sử dụng một archive_timeout rất ngắn - nó sẽ làm tăng nhanh kho lưu trữ của bạn, các thiết lập archive_timeout trong khoảng một phút vì thế thường là hợp lý. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

18.5.4. Nhân bản dòng

Các thiết lập này kiểm soát hành vi của tính năng nhân bản dòng (streaming replication) được xây dựng sẵn. Các tham số đó có thể được thiết lập trong máy chủ chính mà sẽ gửi các dữ liệu nhân bản tới một hoặc nhiều hơn các máy chủ nhàn rỗi.

max_wal_senders (integer)

Chỉ định số lượng tối đa các kết nối đồng thời từ các máy chủ nhàn rỗi (như, số lượng tối đa các tiến trình của người gửi WAL đang chạy đồng thời). Mặc định là zero, nghĩa là sự nhân bản bị vô hiệu hóa. Các tiến trình của người gửi WAL tính tới tổng số các kết nối, nên tham số đó không thể được thiết lập cao hơn max_connections. Tham số này chỉ có thể được thiết lập khi máy chủ khởi động. wal_level phải được thiết lập về archive hoặc hot_standby để cho phép các kết nối từ các máy chủ nhàn rỗi.

wal_sender_delay (integer)

Chỉ định sự trễ giữa các vòng hoạt động đối với các tiến trình của người gửi WAL. Trong từng vòng thì người gửi WAL sẽ gửi đi bất kỳ WAL nào được tích lũy kể từ vòng cuối cùng tới máy chủ nhàn rỗi. Nó sau đó ngủ trong wal_sender_delay mili giây, và lặp lại. Giá trị mặc định là 200 mili giây. Lưu ý là trong nhiều hệ thống, quyết định có hiệu quả các trễ ngủ là 10 mili giây; việc thiết lập wal_sender_delay về một giá trị không phải là bội số của 10 có thể có các kết quả y hệt như việc thiết lập nó về bội số của 10 cao hơn tiếp theo. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

wal_keep_segments (integer)

Chỉ định số tối thiểu các phân đoạn tệp lưu ký trong quá khứ được lưu giữ trong thư mục pg_xlog, trong trường hợp một máy chủ nhàn rỗi cần lấy chúng cho nhân bản dòng. Mỗi phân đoạn thường là 16MB. Nếu một máy chủ nhàn rỗi được kết nối tới máy chủ chính tụt lại đằng sau hơn wal_keep_segments phân đoạn, thì máy chủ chính đó có thể loại bỏ một phân đoạn WAL vẫn còn cần thiết đối với sự nhàn rỗi đó, trong trường hợp đó kết nối nhân bản sẽ bị kết thúc. (Tuy nhiên, máy chủ nhàn rỗi đó có thể phục hồi bằng việc lấy phân đoạn đó từ lưu trữ, nếu việc lưu trữ WAL đang được sử dụng).

Điều này chỉ thiết lập số lượng tối thiểu các phân đoạn nằm trong pg_xlog; hệ thống có thể cần giữ lại nhiều phân đoạn hơn cho lưu trữ hoặc phục hồi WAL từ một điểm kiểm tra. Nếu wal_keep_segments là zero (mặc định), thì hệ thống không giữ bất kỳ phân doạn dư thừa nào vì các mục đích nhàn rỗi, và số lượng các phân đoạn WAL cũ sẵn có cho các máy chủ nhàn rỗi là một hàm vị trí của điểm kiểm tra và tình trạng trước đó của việc lưu trữ WAL. Tham số này không có tác động lên các điểm khởi động lại. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

vacuum_defer_cleanup_age (integer)

Chỉ định số lượng các giao dịch theo đó các bản cập nhật VACUUM và HOT sẽ hoãn làm sạch các phiên bản hàng chết. Mặc định là các giao dịch zero, nghĩa là các phiên bản hàng chết có thể bị loại bỏ cành nhanh càng tốt, đó là, ngay khi chúng không còn nhìn thấy được nữa đối với bất kỳ giao dịch mở nào. Bạn có thể muốn thiết lập điều này về một giá trị không bằng zero trong một máy chủ chính mà đang hỗ trợ nóng các máy chủ nhàn rỗi, như được mô tả trong Phần 25.5. Điều này cho phép nhiều thời gian hơn cho các truy vấn trong máy chủ nhàn rỗi để hoàn tất mà không xảy ra xung đột vì việc sớm làm sạch các hàng. Tuy nhiên, vì giá trị đó được đo đếm theo số lượng các giao dịch được ghi xảy ra trong máy chủ chính, là

khó để đoán trước cần thêm bao nhiêu thời gian sẽ được làm cho sẵn sàng cho các truy vấn nhàn rỗi. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

18.5.5. Máy chủ nhàn rỗi

Các thiết lập này kiểm soát hành vi của một máy chủ nhàn rỗi mà nó sẽ nhận các dữ liệu nhân bản.

hot_standby (boolean)

Chỉ định liệu có hay không bạn có thể kết nối và chạy các truy vấn trong khi phục hồi, như được mô tả trong Phần 25.5. Giá trị mặc định là tắt (off). Tham số này chỉ có thể được thiết lập khi máy chủ khởi động. Nó chỉ có hiệu quả trong quá trình phục hồi lưu trữ hoặc trong chế độ nhàn rỗi.

max_standby_archive_delay (integer)

Khi Hot Standby hoạt động, tham số này xác định máy chủ nhàn rỗi sẽ chờ bao lâu trước khi hoãn các truy vấn nhàn rỗi mà xung đột với các khoản đầu vào WAL sắp được áp dụng, như được mô tả trong Phần 25.5.2. max_standby_archive_delay áp dụng khi dữ liệu WAL đang được đọc từ lưu trữ WAL (và vì thế không hiện hành). Mặc định là 30 giây. Các đơn vị là mili giây nếu không được chỉ định. Một giá trị -1 cho phép sự nhàn rỗi chờ bất tận đối với các truy vấn xung đột để hoàn tất. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

Lưu ý rằng max_standby_archive_delay không phải là y hệt như độ dài thời gian tối đa một truy vấn có thể chạy trước khi hoãn; thay vào đó nó là tổng thời gian tối đa được phép áp dụng bất kỳ một dữ liệu nào của phân đoạn WAL. Vì thế, nếu một truy vấn đã gây ra sự trễ đáng kể trước đó trong phân đoạn WAL, thì các truy vấn xung đột tiếp sau sẽ có ít hơn nhiều thời gian cần thiết.

max_standby_streaming_delay (integer)

Khi Hot Standby hoạt động, tham số này xác định máy chủ nhàn rỗi sẽ chờ bao lâu trước khi hoãn các truy vấn nhàn rỗi mà xung đột với các khoản đầu vào WAL sắp được áp dụng, như được mô tả trong Phần 25.5.5. max_standby_streaming_delay áp dụng khi dữ liệu WAL đang nhận được qua nhân bản dòng. Mặc định là 30 giây. Các đơn vị sẽ là mili giây nếu không được chỉ định. Một giá trị -1 cho phép sự nhàn rỗi chờ bất tận đối với các truy vấn xung đột để hoàn tất. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

Lưu ý rằng max_standby_streaming_delay không phải là y hệt như độ dài thời gian cực đại một truy vấn có thể chạy trước khi hoãn; thay vào đó nó là tổng thời gian tối đa được phép để áp dụng dữ liệu WAL một khi nó đã nhận được từ máy chủ chính. Vì thế, nếu một truy vấn từng gây ra sự trễ đáng kể, thì các truy vấn xung đột sau đó sẽ có ít hơn nhiều thời gian cần thiết cho tới khi máy chủ nhàn rỗi giành được một lần nữa.

18.6. Lên kế hoạch truy vấn

18.6.1. Cấu hình phương pháp cho trình lập kế hoạch

Các tham số cấu hình đưa ra một phương pháp thô ảnh hưởng tới các kế hoạch truy vấn được trình tối ưu truy vấn chọn. Nếu kế hoạch mặc định được trình tối ưu chọn cho một truy vấn đặc biệt nào đó không là tối ưu, thì một giải pháp tạm thời sẽ sử dụng một trong những tham số cấu hình đó để ép trình tối ưu chọn một kế hoạch khác. Các cách thức tốt hơn để cải thiện chất lượng các kế hoạch được chọn của trình tối ưu bao gồm việc tinh chỉnh các hằng số chi phí của trình lên kế hoạch (xem Phần 18.6.2), bằng việc chạy bằng tay ANALYZE, làm gia tăng giá trị tham số cấu hình default_statistics_target, và làm gia tăng lượng số liệu thống kê được thu thập cho các cột đặc thù bằng việc sử dụng ALTER TABLE SET STATISTICS.

enable_bitmapscan (boolean)

Kích hoạt hoặc giải hoạt sử dụng các dạng kế hoạch quét bitmap của trình lên kế hoạch truy vấn. Mặc định là bật (on).

enable_hashagg (boolean)

Kích hoạt hoặc giải hoạt sử dụng các dạng kế hoạch tổng hợp được băm của trình lên kế hoạch truy vấn. Mặc định là bật (on).

enable_hashjoin (boolean)

Kích hoạt hoặc giải hoạt sử dụng các dạng kế hoạch ghép nối băm (hash-join) của trình lên kế hoạch truy vấn. Mặc định là bật (on).

enable_indexscan (boolean)

Kích hoạt hoặc giải hoạt sử dụng các dạng kế hoạch quét chỉ số của trình lên kế hoạch truy vấn. Mặc định là bật (on).

enable_material (boolean)

Kích hoạt hoặc giải hoạt sử dụng vật chất hóa của trình lên kế hoạch truy vấn. Không có khả năng ép hoàn toàn sự vật chất hóa, nhưng việc tắt biến này sẽ ngăn chặn trình lên kế hoạch khỏi việc chèn các nút vật chất hóa ngoại trừ trong các trường hợp nơi mà nó được yêu cầu để sửa cho đúng. Mặc định là bật (on).

enable_mergejoin (boolean)

Kích hoạt hoặc giải hoạt sử dụng các dạng trộn ghép nối của trình lên kế hoạch truy vấn. Mặc định là bật (on).

enable_nestloop (boolean)

Kích hoạt hoặc giải hoạt sử dụng các kế hoạch liên kết vòng lặp lồng nhau của trình lên kế hoạch truy vấn. Không có khả năng để ép toàn bộ các kết nối lặp lồng nhau, nhưng việc tắt biến này sẽ vô hiệu hóa trình lên kế hoạch khỏi việc sử dụng một thứ như vậy nếu các phương pháp khác là có sẵn. Mặc định là bật (on).

enable_seqscan (boolean)

Kích hoạt hoặc giải hoạt sử dụng các dạng kế hoạch quét tuần tự của trình lên kế hoạch truy vấn. Không có khả năng ép toàn bộ các vụ quét tuần tự, nhưng việc tắt biến này sẽ vô hiệu hóa trình lên kế hoạch khỏi việc sử dụng một thứ như vậy nếu các biện pháp khác là sẵn có. Mặc định là bật (on).

enable_sort (boolean)

Kích hoạt hoặc giải hoạt sử dụng các bước sắp xếp rõ ràng của trình lên kế hoạch truy vấn. Không có khả năng để ép toàn bộ các sắp xếp rõ ràng, nhưng việc tắt biến này sẽ vô hiệu hóa trình lên kế hoạch khỏi việc sử dụng một thứ như vậy nếu các biện pháp khác là sẵn có. Mặc định là bật (on).

enable_tidscan (boolean)

Kích hoạt hoặc giải hoạt sử dụng kế hoạch quét TID của trình lên kế hoạch truy vấn. Mặc định là bật (on).

18.6.2. Hằng số chi phí của trình lên kế hoạch

Các biến chi phí được mô tả trong phần này được đo đếm trong một phạm vi tùy ý. Chỉ các giá trị tương đối của chúng là có vấn đề, vì thế việc mở rộng hoặc thu hẹp phạm vi cho tất cả chúng theo cùng y hệt một yếu tố sẽ không làm thay đổi trong các lựa chọn của trình lên kế hoạch. Mặc định, các biến chi phí đó dựa vào chi phí của việc lấy được các trang tuần tự; đó là, seq_page_cost thường được thiết lập về 1.0 và các biến chi phí khác được thiết lập với tham chiếu tới đó. Nhưng bạn có thể sử dụng một phạm vi khác nếu bạn thích hơn, như thời gian thực thi thực tế theo mili giây trong một máy đặc thù.

Lưu ý: Không may, không có phương pháp được xác định tốt cho việc xác định các giá trị lý tưởng đối với các biến chi phí. Chúng được ứng xử tốt nhất như là các trung bình đối với toàn bộ sự pha trộn các truy vấn mà một cài đặt cụ thể sẽ nhận được. Điều này có nghĩa là việc thay đổi chúng trên cơ sở của chỉ một ít các thí điểm là rất rủi ro.

seq_page_cost (floating point)

Thiết lập ước lượng chi phí của trình lên kế hoạch để lấy về một trang đĩa mà là một phần của một loạt các lấy về tuần tự nhau. Mặc định là 1.0. Giá trị này có thể bị ghi đè đối với một không gian bảng đặc thù bằng việc thiết lập tham số không gian bảng với tên y hệt (xem ALTER TABLESPACE).

random_page_cost (floating point)

Thiết lập ước tính chi phí của trình lên kế hoạch đối với trang đĩa được lấy về không tuần tự. Mặc định là 4.0. Giá trị này có thể bị ghi đè đối với một không gian đĩa đặc biệt bằng việc thiết lập tham số không gian đĩa với tên y hệt (xem ALTER TABLESPACE).

Việc làm giảm giá trị này tương ứng với seq_page_cost sẽ làm cho hệ thống thích các quét chỉ số hơn; việc gia tăng nó sẽ làm cho các quét chỉ số trông khá đắt hơn. Bạn có thể tăng hoặc giảm cả 2 giá trị đó cùng nhau để thay đổi tầm quan trọng các chi phí I/O của đĩa một cách tương đối so với các chi phí CPU, điều được mô tả bằng các tham số sau đây.

Mẹo: Dù hệ thống sẽ để cho bạn thiết lập random_page_cost ít hơn so với seq_page_cost, không cảm thấy một cách vật lý để làm như vậy. Tuy nhiên, việc thiết lập chúng ngang bằng nhau là có ý nghĩa nếu cơ sở dữ liệu hoàn toàn được giữ trong RAM, vì trong trường hợp đó không có thiệt hại đối với việc động chạm tới các trang ngoài sự tuần tự đó. Hơn nữa, trong một cơ sở dữ liệu được nhớ tạm nặng thì bạn nên giảm cả 2 giá trị tương ứng với các tham số của CPU, vì chi phí của việc lấy một trang có rồi trong RAM là nhỏ hơn nhiều so với nó có thể là bình thường.

cpu_tuple_cost (floating point)

Thiết lập ước lượng chi phí của trình lên kế hoạch đối với việc xử lý từng hàng trong quá trình một truy vấn. Mặc định là 0.01.

cpu_index_tuple_cost (floating point)

Thiết lập ước lượng chi phí của trình lên kế hoạch đối với việc xử lý từng khoản đầu vào chỉ số trong quá trình một sự quét chỉ số. Mặc định là 0.005.

cpu_operator_cost (floating point)

Thiết lập ước lượng chi phí của trình lên kế hoạch đối với việc xử lý từng toán tử hoặc hàm được thực thi trong quá trình một truy vấn. Mặc định là 0.0025.

effective_cache_size (integer)

Thiết lập giả thiết của trình lên kế hoạch về kích thước có hiệu quả của bộ nhớ tạm của đĩa mà có sẵn cho một truy vấn duy nhất. Đây là yếu tố trong các ước tính chi phí của việc sử dụng một chỉ số; một giá trị cao hơn làm cho có khả năng nhiều hơn các quét chỉ số sẽ được sử dụng, một giá trị thấp hơn làm cho có khả năng hơn các quét tuần tự sẽ được sử dụng. Khi thiết lập tham số này thì bạn nên cân nhắc cả các bộ nhớ đệm được chia sẻ của PostgreSQL và phần của bộ nhớ tạm trên đĩa cho nhân mà sẽ được các tệp dữ liệu của PostgreSQL sử dụng. Hơn nữa, hãy tính tới số các truy vấn cùng đồng thời được kỳ vọng trong các bảng khác nhau, vì chúng sẽ phải chia sẻ không gian sẵn có. Tham số này không có ảnh hưởng lên kích cỡ bộ nhớ được chia sẻ được PostgreSQL phân bổ, cũng không dự trữ bộ nhớ tạm trên đĩa cho nhân; nó chỉ được sử dụng cho các mục đích ước tính. Mặc định là 128 MB.

18.6.3. Trình tối ưu hóa truy vấn di truyền

Trình tối ưu hóa truy vấn di truyền - GEQO (Genetic Query Optimizer) là một thuật toán làm kế hoạch truy vấn bằng việc sử dụng tìm kiếm tối ưu (heuristic searching). Điều này làm giảm thời gian lên kế hoạch đối với các truy vấn phức tạp (các truy vấn liên kết nhiều quan hệ), với chi phí sản xuất các kế hoạch đôi khi yếm thế hơn so với các các chi phí được tìm ra bằng thuật toán tìm kiếm vét cạn thông thường. Để có thêm thông tin, xem Chương 50.

geqo (boolean)

Kích hoạt và giải hoạt sự tối ưu hóa truy vấn di truyền. Đây là mặc định. Thường tốt nhất không tắt nó trong sản xuất; biến geqo_threshold đưa ra sự kiểm soát mịn hơn đối với GEQO.

geqo_threshold (integer)

Sử dụng tối ưu hóa truy vấn di truyền để lên kế hoạch các truy vấn với ít nhất nhiều khoản FROM này tham gia. (Lưu ý rằng một cấu trúc FULL OUTER JOIN tính như chỉ một khoản FROM). Mặc định là 12. Đối với các truy vấn đơn giản hơn thường là tốt nhất để sử dụng trình lên kế hoạch thông thường, tìm kiếm vết cạn, nhưng đối với các truy vấn với nhiều bảng thì tìm kiếm vét cạn sẽ rất lâu, thường lâu hơn cả sự thiệt hại của việc thực thi một kế hoạch gần tối ưu. Vì thế, một ngưỡng trong kích cỡ của truy vấn là một cách thức thuận lợi để quản lý sử dụng GEQO.

geqo_effort (integer)

Các kiểm soát lựa chọn thỏa hiệp giữa thời gian lên kế hoạch và chất lượng kế hoạch của truy vấn trong GEQO. Biến này phải là một số nguyên trong dải từ 1 tới 10. Giá trị mặc định là 5. Các giá trị lớn hơn làm gia tăng thời gian bỏ ra để lên kế hoạch truy vấn, nhưng cũng làm gia tăng khả năng một kế hoạch truy vấn có hiệu quả sẽ được chọn.

geqo_effort thực sự không làm bất kỳ điều gì trực tiếp; nó chỉ được sử dụng để tính toán các giá trị mặc định cho các biến khác mà có ảnh hưởng tới hành vi của GEQO (được mô tả bên dưới). Nếu bạn thích hơn, bạn có thể thiết lập các tham số khác bằng tay.

geqo_pool_size (integer)

Các kiểm soát kích cỡ kho được GEQO sử dụng, đó là số các cá nhân trong tập hợp di truyền. Nó ít nhất phải là 2, và các giá trị hữu dụng thường là 100 tới 1.000. Nếu nó được thiết lập về zero (thiết lập mặc định) thì một giá trị phù hợp sẽ được chọn dựa vào geqo_effort và số lượng các bảng trong truy vấn đó.

geqo_generations (integer)

Các kiểm soát mà các số phát sinh được GEQO sử dụng, đó là số lặp đi lặp lại của thuật toán. Nó phải ít nhất là 1, và các giá trị hữu dụng trong cùng dãy y hệt như kích thước kho. Nếu nó được thiết lập về zero (thiết lập mặc định) thì một giá trị phù hợp sẽ được chọn dựa vào geqo_pool_size.

geqo_selection_bias (floating point)

Các kiểm soát khuynh hướng lựa chọn được GEQO sử dụng. Khuynh hướng lựa chọn là sức ép lựa chọn bên trong tập hợp. Các giá trị có thể từ 1.50 tới 2.00; giá trị sau là mặc định.

geqo_seed (floating point)

Các kiểm soát mà giá trị ban đầu của bộ sinh số ngẫu nhiên được GEQO sử dụng để lựa chọn các đường dẫn ngẫu nhiên qua không gian tìm kiếm trật tự chung. Giá trị đó có thể trải từ zero (mặc định) tới 1. Việc biến đổi giá trị đó làm thay đổi tập hợp các đường dẫn tham gia được khai thác, và có thể làm cho một đường dẫn tốt hơn hoặc tồi hơn được tìm thấy.

18.6.4. Các lựa chọn khác của trình lên kế hoạch

default_statistics_target (integer)

Thiết lập mục tiêu số liệu thống kê mặc định cho các cột của bảng mà không có đích đặc thù cột được thiết lập qua ALTER TABLE SET STATISTICS. Các giá trị lớn hơn làm gia tăng thời gian

cần thiết để thực hiện ANALYZE, nhưng có thể cải thiện chất lượng của các ước tính của trình lên kế hoạch. Mặc định là 100. Để có thêm thông tin về sử dụng các số liệu thống kê của trình lên kế hoạch truy vấn PostgreSQL, hãy tham chiếu tới Phần 14.2.

constraint_exclusion (enum)

Các kiểm soát sử dụng các ràng buộc bảng của trình lên kế hoạch truy vấn để tối ưu hóa các truy vấn. Các giá trị được phép của constraint_exclusion là bật - on (kiểm tra các ràng buộc cho tất cả các bảng), tắt - off (không bao giờ kiểm tra các ràng buộc), và một phần - partition (kiểm tra các ràng buộc đối với các bảng con thừa kế và các truy vấn con UNION ALL). partition là thiết lập mặc định. Nó thường được sử dụng với sự kế thừa và các bảng được phân vùng để cải thiện hiệu năng.

Khi tham số này cho phép nó đối với một bảng nhất định, thì trình lên kế hoạch sẽ so sánh các điều kiện truy vấn với các ràng buộc CHECK của bảng đó, và bỏ qua việc quét các bảng mà đối với chúng các điều kiện đó xung đột với các ràng buộc đó. Ví dụ:

```
CREATE TABLE parent(key integer, ...);
CREATE TABLE child1000(check (key between 1000 and 1999)) INHERITS(parent);
CREATE TABLE child2000(check (key between 2000 and 2999)) INHERITS(parent);
...
SELECT * FROM parent WHERE key = 2400;
```

Với ngoại lệ ràng buộc được kích hoạt, thì SELECT này sẽ không quét hoàn toàn child1000, khi cải thiện hiệu năng.

Hiện hành, ngoại trừ ràng buộc chỉ được kích hoạt ngầm định cho các trường hợp thường được sử dụng để triển khai việc phân vùng bảng. Việc bật nó cho tất cả các bảng đặt ra chi phí tổng thể bổ sung của việc lên kế hoạch mà hoàn toàn lưu ý thấy được trong các truy vấn đơn giản, và rất thường xuyên không có được lợi ích cho các truy vấn đơn giản. Nếu bạn không có các bảng được phân vùng thì bạn có thể thích hơn để tắt nó hoàn toàn.

Tham chiếu Phần 5.9.4 để có thêm thông tin về sử dụng ngoại lệ ràng buộc và phân vùng.

cursor_tuple_fraction (floating point)

Thiết lập ước tính phân số của trình lên kế hoạch đối với các hàng của một con trỏ sẽ được truy xuất. Mặc định là 0.1. Các giá trị nhỏ hơn của thiết lập này có xu hướng sử dụng các kế hoạch “khởi động nhanh” đối với các con trỏ, chúng sẽ truy xuất một ít hàng đầu tiên nhanh chóng trong khi có lẽ mất nhiều thời gian để lấy tất cả các hàng. Các giá trị lớn hơn đặt ra sự nhấn mạnh hơn vào tổng thời gian được ước tính. Ở thiết lập tối đa 1.0, các con trỏ được lên kế hoạch chính xác giống như các truy vấn thông thường, chỉ cân nhắc tới tổng thời gian được ước tính và không cân nhắc tới việc các hàng đầu tiên có thể được phân phối thế nào.

from_collapse_limit (integer)

Trình lên kế hoạch sẽ trộn các truy vấn con vào các truy vấn phía trên nếu danh sách sinh ra FROM có thể không có nhiều hơn so với nhiều khoản này. Các giá trị nhỏ hơn làm giảm thời gian lên kế hoạch nhưng có thể có các kế hoạch truy vấn yếm thế hơn. Mặc định là 8. Để có thêm thông tin, hãy xem Phần 14.3.

Việc thiết lập giá trị này về ngưỡng geqo_threshold hoặc nhiều hơn có thể làm bật dậy sử dụng trình lên kế hoạch GEQO, tạo thành các kế hoạch không tối ưu. Xem Phần 18.6.3.

join_collapse_limit (integer)

Trình lên kế hoạch sẽ viết lại các cấu trúc JOIN rõ ràng (ngoại trừ FULL JOIN s) thành các danh sách của các khoản FROM trong đó bất kỳ khi nào một danh sách của không nhiều hơn số lượng các khoản này có thể tạo ra. Các giá trị nhỏ hơn làm giảm thời gian lên kế hoạch nhưng có thể lấy các kế hoạch truy vấn kém.

Mặc định, biến này được thiết lập y hệt như from_collapse_limit, nó phù hợp cho hầu hết các sử dụng. Việc thiết lập nó về 1 ngăn ngừa bất kỳ việc tái lập trật tự nào của JOIN rõ ràng. Vì thế, trật tự ghép nối rõ ràng được chỉ định trong truy vấn đó sẽ là trật tự thực tế theo đó các quan hệ được kết nối. Vì trình lên kế hoạch truy vấn không phải luôn chọn trật tự kết nối tối ưu, nên những người sử dụng tiên tiến có thể chọn để thiết lập tạm thời biến này về 1, và sau đó chỉ định trật tự liên kết mà họ mong muốn rõ ràng. Để có thêm thông tin, xem Phần 14.3.

Việc thiết lập giá trị này về ngưỡng geqo_threshold hoặc nhiều hơn có thể làm bật dậy sử dụng trình lên kế hoạch GEQO, tạo ra các kế hoạch không tối ưu. Xem Phần 18.6.3.

18.7. Báo cáo và lưu ký lỗi

18.7.1. Lưu ký ở đâu

log_destination (string)

PostgreSQL hỗ trợ vài phương pháp lưu ký các thông điệp máy chủ, bao gồm stderr, csvlog và syslog. Trên Windows, eventlog cũng được hỗ trợ. Hãy thiết lập tham số này vào một danh sách các đích đến lưu ý mong muốn được phân cách nhau bằng các dấu phẩy. Mặc định là để lưu ký chỉ tới stderr. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

Nếu csvlog được đưa vào trong log_destination, thì các khoản đầu vào lưu ký là đầu ra trong định dạng "giá trị được phân cách nhau bằng dấu phẩy" - CSV (comma separated value), nó là thuận tiện để tải các lưu ký vào các chương trình. Xem Phần 18.7.4 để có các chi tiết. logging_collector phải được kích hoạt để sinh ra đầu ra lưu ký định dạng CSV.

> **Lưu ý**: Trong hầu hết các hệ thống Unix, bạn sẽ cần chỉnh cấu hình daemon syslog hệ thống của bạn để chắc chắn lựa chọn syslog cho log_destination. PostgreSQL có thể lưu ký cho các tiện ích syslog LOCAL0 qua LOCAL7 (xem syslog_facility), nhưng cấu hình syslog mặc định trong hầu hết các nền tảng sẽ bỏ qua tất cả các thông điệp như vậy. Bạn sẽ cần thêm thứ gì đó như:
>
> local0.* /var/log/postgresql
>
> cho tệp cấu hình của daemon syslog để làm cho nó làm việc được.

logging_collector (boolean)

Tham số này nắm lấy các thông điệp lưu ký có định dạng CSV và dạng thô được gửi cho stderr và tái định tuyến chúng vào trong các tệp lưu ký. Tiếp cận này thường hữu dụng hơn so với việc lưu ký tới syslog, vì một số dạng thông điệp có thể không xuất hiện ở đầu ra của

syslog (một ví dụ phổ biến là các thông điệp hỏng của trình liên kết động). Tham số này chỉ có thể thiết lập khi máy chủ khởi động.

Lưu ý: Bộ thu thập lưu ký được thiết kế để không bao giờ đánh mất các thông điệp. Điều này có nghĩa là trong trường hợp tải cực kỳ cao, thì các tiến trình máy chủ có thể bị khóa vì việc cố gửi thêm các thông điệp lưu ký khi bộ thu thập bị chậm ở phía sau. Ngược lại, syslog thích bỏ các thông điệp hơn nếu nó không thể ghi chúng, điều này có nghĩa là ít tin cậy hơn trong các trường hợp đó nhưng nó sẽ không khóa phần còn lại của hệ thống.

log_directory (string)

Khi logging_collector được kích hoạt, tham số này xác định thư mục ở đó các tệp lưu ký sẽ được tạo ra. Nó có thể được chỉ định như một đường dẫn tuyệt đối, hoặc tương đối cho thư mục dữ liệu bó. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

log_filename (string)

Khi logging_collector được kích hoạt, tham số này thiết lập các tên tệp của các tệp lưu ký được tạo ra. Giá trị này được đối xử như một mẫu strftime, nên % thoát (%-escapes) có thể được sử dụng để chỉ định các tên tệp khác nhau theo thời gian. (Lưu ý rằng nếu có bất kỳ %-escapes nào phụ thuộc vùng thời gian, thì tính toán được thực hiện trong vùng được log_timezone chỉ định). Lưu ý rằng strftime của hệ thống không được sử dụng trực tiếp, nên các mở rộng đặc thù nền tảng (phi tiêu chuẩn) sẽ không làm việc được.

Nếu bạn chỉ định một tên tệp mà không có các thoát, thì bạn nên có kế hoạch sử dụng một tiện ích quay vòng lưu ký để tránh cuối cùng sẽ làm đầy toàn bộ đĩa. Trong các phiên bản trước 8.4, nếu không có % thoát nào từng hiện diện, thì PostgreSQL có thể nối thêm sự bắt đầu của thời gian tạo tệp lưu ký mới, nhưng điều này không còn là quan trọng nữa.

Nếu đầu ra có định dạng CSV được kích hoạt trong log_destination, thì .csv sẽ được nối vào tên tệp lưu ký có dấu thời gian để tạo tên tệp cho đầu ra có định dạng CSV. (Nếu log_filename kết thúc trong .log, thì phần mở rộng sẽ được thay thế). Trong trường hợp ví dụ ở trên, tên tệp CSV sẽ là server_log.1093827753.csv.

Tham số này chỉ có thể được thiết lập ở tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

log_rotation_age (integer)

Khi logging_collector được kích hoạt, tham số này xác định thời gian sống tối đa của một tệp lưu ký riêng rẽ. Sau khi nhiều phút này đã trôi qua, thì một tệp lưu ký mới sẽ được tạo ra. Hãy thiết lập về zero để vô hiệu hóa sự tạo ra các tệp lưu ký mới dựa vào thời gian. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

log_rotation_size (integer)

Khi logging_collector được kích hoạt, tham số này xác định kích cỡ tối đa của một tệp lưu ký riêng rẽ. Sau khi nhiều KB này đã được chuyển vào một tệp lưu ký, thì một tệp lưu ký mới sẽ được tạo ra. Hãy thiết lập về zero để vô hiệu hóa sự tạo ra các tệp lưu ký mới dựa vào

kích cỡ. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

log_truncate_on_rotation (boolean)

Khi logging_collector được kích hoạt, tham số này sẽ làm cho PostgreSQL cắt ngắn bớt (ghi đè), thay vì nối thêm, bất kỳ tệp lưu ký đang tồn tại nào với tên y hệt. Tuy nhiên, sự cắt bớt sẽ chỉ xảy ra khi một tệp mới đang được mở vì sự xoay vòng dựa vào thời gian, chứ không phải trong quá trình máy chủ khởi động hoặc xoay vòng dựa vào kích cỡ. Khi tắt, các tệp tồn tại trước đó sẽ được nối thêm vào trong tất cả các trường hợp. Ví dụ, việc sử dụng thiết lập này kết hợp với một log_filename như postgresql-%H.log có khả năng sinh ra các tệp lưu ký theo giờ dạng 24 giờ và sau đó ghi đè chúng theo chu kỳ. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

Ví dụ: Để giữ 7 ngày các lưu ký, một tệp lưu ký theo từng ngày có tên là server_log.Mon, server_log.Tue, ... , và tự động ghi đè lên lưu ký của tuần trước bằng lưu ký của tuần này, nên hãy thiết lập log_filename về server_log.%a, log_truncate_on_rotation về bật (on), và log_rotation_age về 1440.

Ví dụ: Để giữ 24 tệp lưu ký, một tệp lưu ký mỗi giờ, mà còn xoay vòng sớm hơn nếu kích cỡ tệp lưu ký đó lớn hơn 1GB, hãy thiết lập log_filename về server_log.%H%M, log_truncate_on_rotation về bật (on), log_rotation_age về 60, và log_rotation_size về 1.000000. Việc đưa %M vào log_filename sẽ cho phép bất kỳ sự xoay vòng nào hướng kích cỡ có thể xảy ra để lựa chọn một tên tệp khác với tên tệp ban đầu theo giờ.

syslog_facility (enum)

Khi việc lưu ký tới syslog được kích hoạt, tham số này xác định "tiện ích" syslog sẽ được sử dụng. Bạn có thể chọn từ LOCAL0, LOCAL1, LOCAL2, LOCAL3, LOCAL4, LOCAL5, LOCAL6, LOCAL7; mặc định là LOCAL0. Xem thêm tài liệu daemon syslog hệ thống của bạn.

syslog_ident (string)

Khi việc lưu ký tới syslog được kích hoạt, tham số này xác định tên chương trình được sử dụng để nhận diện các thông điệp PostgreSQL trong các lưu ký syslog. Mặc định là postgres. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ. Tham số này là không có sẵn trừ phi máy chủ được biên dịch với sự hỗ trợ của syslog.

silent_mode (boolean)

Chạy máy chủ một cách âm thầm. Nếu tham số này được thiết lập, máy chủ sẽ tự động chạy ở phần nền và tách ra khỏi thiết bị kiểm tra đầu cuối. Tham số này chỉ có thể được thiết lập khi máy chủ khởi tạo.

Cảnh báo

Khi tham số này được thiết lập, đầu ra tiêu chuẩn và lỗi tiêu chuẩn của máy chủ được tái định tuyến tới tệp postmaster.log bên trong thư mục dữ liệu. Không có điều khoản nào cho việc xoay vòng tệp này, nên nó sẽ tăng vô định, trừ phi đầu ra lưu ký máy chủ được tái định tuyến ở đâu đó bằng các thiết lập khác. Được khuyến cáo rằng log_destination được thiết lập về syslog hoặc logging_collector sẽ được kích hoạt khi sử dụng lựa chọn này. Thậm chí với các biện pháp đó, các lỗi được nêu sớm trong quá trình khởi động có thể xuất hiện trong postmaster.log thay vì đích lưu ký thông thường.

18.7.2. Lưu ký khi nào

client_min_messages (enum)

Kiểm soát các mức thông điệp nào được gửi cho máy trạm. Các giá trị hợp lệ là DEBUG5, DEBUG4, DEBUG3, DEBUG2, DEBUG1, LOG, NOTICE, WARNING, ERROR, FATAL, và PANIC. Mỗi mức bao gồm tất cả các mức đi sau nó. Mức cuối cùng, ít các thông điệp hơn sẽ được gửi đi. Mặc định là NOTICE. Lưu ý rằng LOG có một thứ hạng khác ở đây so với trong log_min_messages.

log_min_messages (enum)

Kiểm soát các mức thông điệp nào được viết vào lưu ký máy chủ. Các giá trị hợp lệ là DEBUG5, DEBUG4, DEBUG3, DEBUG2, DEBUG1, INFO, NOTICE, WARNING, ERROR, LOG, FATAL, và PANIC. Mỗi mức bao gồm tất cả các mức đi sau nó. Mức sau cùng, ít các thông điệp hơn sẽ được gửi tới lưu ký. Mặc định là WARNING. Lưu ý rằng LOG có một thứ hạng khác ở đây so với trong client_min_messages. Chỉ các siêu người sử dụng có thể thay đổi thiết lập này.

log_min_error_statement (enum)

Kiểm soát các lệnh SQL nào gây ra một điều kiện lỗi sẽ được ghi lại trong lưu ký máy chủ. Lệnh SQL hiện hành được đưa vào trong khoản đầu vào lưu ký cho bất kỳ thông điệp nào về độ nghiêm trọng được chỉ định hoặc cao hơn. Các giá trị hợp lệ là DEBUG5, DEBUG4, DEBUG3, DEBUG2, DEBUG1, INFO, NOTICE, WARNING, ERROR, LOG, FATAL, và PANIC. Mặc định là ERROR, điều có nghĩa là các lệnh gây ra các lỗi, các thông điệp lưu ký, các lỗi sống còn, hoặc các hoảng loạn sẽ được lưu ký lại. Để tắt có hiệu quả việc lưu ký các lệnh hỏng, hãy thiết lập tham số này về PANIC. Chỉ các siêu người sử dụng có thể thay đổi thiết lập này.

log_min_duration_statement (integer)

Căn nguyên thời gian của từng lệnh được hoàn tất sẽ được lưu ký nếu lệnh được chạy cho ít nhất số lượng mili giây được chỉ định. Việc thiết lập điều này về zero in ra tất cả thời gian các lệnh. Minusone (mặc định) vô hiệu hóa việc lưu ký thời gian của lệnh. Ví dụ, nếu bạn

thiết lập nó về 250 mili giây (ms) thì sau đó tất cả các lệnh SQL mà chạy 250 ms hoặc lâu hơn sẽ được lưu ký lại. Việc kích hoạt tham số này có thể là hữu dụng trong việc theo dõi các truy vấn không tối ưu trong các ứng dụng của bạn. Chỉ các siêu người sử dụng có thể thay đổi thiết lập này.

Đối với các máy trạm sử dụng giao thức truy vấn được mở rộng, thời gian của các bước Parse, Bind, và Execute sẽ được lưu ký lại một cách độc lập.

Lưu ý: Khi sử dụng lựa chọn này cùng với log_statement, văn bản các lệnh sẽ được lưu ký vì log_statement sẽ không được lặp lại trong thông điệp lưu ký thời gian. Nếu bạn đang không sử dụng syslog, thì được khuyến cáo rằng bạn lưu ký PID hoặc IP phiên làm việc bằng việc sử dụng log_line_prefix sao cho bạn có thể liên kết thông điệp lệnh tới thông điệp thời gian sau bằng việc sử dụng ID tiến trình hoặc ID phiên làm việc.

Bảng 18-1 giải thích các mức độ nghiêm trọng thông điệp được PostgreSQL sử dụng. Nếu việc lưu ký đầu ra được gửi tới syslog hoặc eventlog của Windows, thì các mức độ nghiêm trọng sẽ được dịch như được chỉ ra trong bảng.

Bảng 18-1. Các mức độ nghiêm trọng thông điệp

Độ nghiêm trọng	Sử dụng	Syslog	eventlog
DEBUG1..DEBUG5	Đưa ra thông tin chi tiết thành công hơn để các lập trình viên sử dụng.	DEBUG	INFORMATION
INFO	Đưa ra thông tin được người sử dụng ngầm yêu cầu, như, đầu ra từ VACUUM VERBOSE.	INFO	INFORMATION
NOTICE	Đưa ra thông tin có thể là hữu dụng cho người sử dụng, như, lưu ý về sự cắt bớt các mã định danh.	NOTICE	INFORMATION
WARNING	Đưa ra các cảnh báo các vấn đề có khả năng, như, COMMIT bên ngoài một khối giao dịch.	NOTICE	WARNING
ERROR	Nêu một lỗi làm cho lệnh hiện hành bị bỏ đi.	WARNING	ERROR
LOG	Nêu thông tin cần quan tâm đối với các quản trị viên, như, hoạt động của các điểm kiểm tra.	INFO	INFORMATION
FATAL	Nêu một lỗi làm cho phiên làm việc hiện hành phải bỏ đi.	ERR	ERROR
PANIC	Nêu một lỗi làm cho tất các các phiên làm việc của cơ sở dữ liệu phải bỏ đi.	CRIT	ERROR

18.7.3. Lưu ký cái gì

application_name (string)

application_name có thể là bất kỳ chuỗi nào ít hơn NAMEDATALEN ký tự (64 ký tự trong một xây dựng tiêu chuẩn). Thường được một ứng dụng thiết lập dựa vào kết nối tới máy chủ. Tên đó sẽ được hiển thị trong kiểu nhìn pg_stat_activity và được đưa vào trong các khoản đầu vào lưu ký CSV. Nó cũng có thể được đưa vào trong các khoản đầu vào lưu ký thông thường qua tham số log_line_prefix. Chỉ các ký tự ASCII có khả năng in ra được mới có thể được sử dụng trong giá trị application_name. Các ký tự khác sẽ được thay thế bằng các dấu hỏi (?).

debug_print_parse (boolean)
debug_print_rewritten (boolean)
debug_print_plan (boolean)

Các tham số này cho phép việc gỡ rối các đầu ra khác nhau sẽ được phát hành. Khi được thiết lập, chúng in cây phân tích kết quả, đầu ra trình ghi lại truy vấn, hoặc kế hoạch thực thi cho từng truy vấn được thực thi. Các thông điệp đó được phát ở mức thông điệp LOG, vì thế mặc định chúng sẽ xuất hiện trong lưu ký máy chủ nhưng sẽ không được gửi tới máy trạm. Bạn có thể thay đổi điều đó bằng việc tinh chỉnh client_min_messages và/hoặc log_min_messages. Các tham số đó mặc định là tắt (off).

debug_pretty_print (boolean)

Khi được thiết lập, debug_pretty_print khắc vết các thông điệp được debug_print_parse, debug_print_rewritten, hoặc debug_print_plan sản xuất. Các kết quả này có khả năng đọc được nhiều hơn nhưng đầu ra dài hơn nhiều so với định dạng "ép chặt" được sử dụng khi nó là tắt (off). Mặc định nó là bật (on).

log_checkpoints (boolean)

Nguyên do các điểm kiểm tra sẽ được lưu ký trong lưu ký máy chủ. Một vài số liệu thống kê về từng điểm kiểm tra sẽ được đưa vào trong các thông điệp lưu ký, bao gồm số lượng các bộ nhớ đệm được viết và thời gian bỏ ra để viết chúng. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ. Mặc định là tắt (off).

log_connections (boolean)

Nguyên do từng kết nối định thử tới máy chủ sẽ được lưu ký lại, cũng như hoàn tất thành công xác thực máy trạm. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ. Mặc định là tắt (off).

> **Lưu ý**: Một vài chương trình máy trạm, như psql, cố kết nối 2 lần trong khi xác định liệu một mật khẩu có được yêu cầu hay không, nên đúp bản các thông điệp "kết nối được nhận" không nhất thiết chỉ ra là có vấn đề.

log_disconnections (boolean)

Điều này đưa ra một dòng trong lưu ký máy chủ tương tự như log_connections nhưng ở kết thúc của phiên làm việc, và bao gồm thời gian của phiên đó. Mặc định điều này là tắt (off). Tham số này chỉ có thể được thiết lập ở tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

log_duration (boolean)

Nguyên do thời gian của mỗi lệnh được hoàn tất sẽ được lưu ký lại. Mặc định là tắt (off). Chỉ các siêu người sử dụng có thể thay đổi thiết lập này.

Đối với các máy trạm đang sử dụng giao thức truy vấn mở rộng, thời gian của các bước Parse, Bind, và Execute sẽ được lưu ký lại một cách độc lập.

> **Lưu ý**: Sự khác biệt giữa việc thiết lập lựa chọn này và việc thiết lập log_min_duration_statement về zero là việc vượt quá log_min_duration_statement sẽ ép buộc văn bản của truy vấn sẽ được lưu ký lại, nhưng lựa chọn này sẽ không. Vì thế, nếu log_duration là bật (on) và log_min_duration_statement có một giá trị dương, thì tất cả

thời gian sẽ được lưu ký nhưng văn bản truy vấn sẽ chỉ được đưa vào cho các lệnh vượt quá ngưỡng đó. Hành vi này có thể là hữu dụng cho việc thu thập số liệu thống kê trong các cài đặt tải cao.

log_error_verbosity (enum)

Kiểm soát lượng chi tiết được viết trong lưu ký máy chủ cho từng thông điệp được lưu ký lại. Các giá trị hợp lệ là TERSE, DEFAULT, và VERBOSE, từng giá trị bổ sung thêm nhiều hơn các trường tới các thông điệp được hiển thị. TERSE loại bỏ việc lưu ký thông tin lỗi của DETAIL, HINT, QUERY, và CONTEXT. Đầu ra của VERBOSE bao gồm mã lỗi SQLSTATE (xem thêm Phụ lục A) và tên tệp mã nguồn, tên hàm, và số lượng dòng đã sinh ra lỗi. Chỉ các siêu người sử dụng có thể thay đổi thiết lập này.

log_hostname (boolean)

Mặc định, các thông điệp lưu ký kết nối chỉ đưa ra địa chỉ IP của máy chủ host kết nối. Việc bật lên tham số này cũng gây ra việc lưu ký tên máy chủ host. Lưu ý là phụ thuộc vào thiết lập giải pháp tên máy chủ host của bạn, điều này có thể áp đặt một thiệt hại về hiệu năng đáng kể. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

log_line_prefix (string)

Đây là một chuỗi dạng printf mà là đầu ra ở đầu mỗi dòng lưu ký. Các ký tự % bắt đầu "các tuần tự thoát" sẽ được thay thế bằng thông tin tình trạng như được phác thảo bên dưới. Các thoát không được thừa nhận sẽ bị bỏ qua. Các ký tự khác sẽ được sao chép ngay tới dòng lưu ký. Một vài thoát sẽ chỉ được các tiến trình phiên làm việc thừa nhận, và được các tiến trình nền bỏ qua như là tiến trình chính của máy chủ. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ. Mặc định là một chuỗi rỗng.

Thoát	Hiệu ứng	Chỉ phiên
%a	Tên ứng dụng	có
%u	Tên người sử dụng	có
%d	Tên cơ sở dữ liệu	có
%r	Tên hoặc địa chỉ IP máy chủ host ở xa, và cổng ở xa	có
%h	tên hoặc địa chỉ IP máy chủ host ở xa	có
%p	ID tiến trình	không
%t	Dấu thời gian không với mili giây	không
%m	Dấu thời gian với mili giây	không
%i	Thẻ lệnh: dạng lệnh hiện hành của phiên làm việc	có
%e	mã lỗi của SQLSTATE	không
%c	ID phiên làm việc: xem bên dưới	không
%l	Số dòng lưu ký cho từng phiên làm việc hoặc tiến trình, bắt đầu từ 1	không
%s	Dấu thời gian bắt đầu tiến tình	không
%v	ID giao dịch ảo (backendID/localXID)	không

Thoát	Hiệu ứng	Chỉ phiên
%x	ID giao dịch (0 nếu không giao dịch nào được chỉ định)	không
%q	Không tạo ra đầu ra, nhưng nói cho các tiến trình không có phiên làm việc để dừng ở điểm này trong chuỗi; được các tiến trình phiên làm việc bỏ qua	không
%%	% hằng	không

Thoát % in một mã định danh phiên làm việc hầu như duy nhất, tạo nên 2 số hệ 16 có 4 byte (không có zero ở đầu) tách biệt nhau bằng một dấu chấm. Các số đó là thời gian bắt đầu tiến trình và ID tiến trình, nên %c cũng có thể được sử dụng như một không gian tiết kiệm cách để in các khoản đó. Ví dụ, để sinh ra mã định danh của phiên làm việc từ pg_stat_activity, hãy sử dụng truy vấn này:

```
SELECT to_hex(EXTRACT(EPOCH FROM backend_start)::integer) || '.' ||
       to_hex(procpid)
FROM pg_stat_activity;
```

> **Mẹo**: Nếu bạn thiết lập một giá trị không rỗng cho log_line_prefix, bạn thường nên làm cho ký tự cuối cùng của nó là một dấu trống, để đưa ra sự tách biệt nhìn thấy được đối với phần còn lại của dòng lưu ký. Ký tự ngắt dòng cũng có thể được sử dụng.

> **Mẹo**: Syslog tạo ra dấu thời gian và thông tin ID tiến trình của riêng nó, nên bạn có thể không muốn đưa vào các thoát đó nếu bạn đang lưu ký tới syslog.

log_lock_waits (boolean)

Kiểm soát liệu một thông điệp lưu ký có được tạo ra khi một phiên làm việc chờ đợi lâu hơn so với deadlock_timeout để có được một sự khóa. Điều này là hữu dụng trong việc xác định liệu các cuộc chờ khóa có đang gây ra hiệu năng tồi hay không. Mặc định là tắt (off).

log_statement (enum)

Kiểm soát các lệnh SQL nào sẽ được lưu ký. Các giá trị hợp lệ là không - none (off), ddl, mod, và all (tất cả các lệnh). ddl lưu ký tất cả các lệnh định nghĩa dữ liệu, như các lệnh CREATE, ALTER, và DROP.

mod lưu ký tất cả các lệnh ddl, cộng với các lệnh sửa đổi dữ liệu như INSERT, UPDATE, DELETE, TRUNCATE, và COPY FROM. PREPARE, EXECUTE, và các lệnh EXPLAIN ANALYZE cũng sẽ được ghi lưu ký nếu lệnh bên trong chúng là ở dạng phù hợp. Đối với các máy trạm đang sử dụng giao thức truy vấn mở rộng, việc lưu ký xảy ra khi một thông điệp thực thi Execute là nhận được, và các giá trị của các tham số Bind sẽ bao gồm (với bất kỳ dấu ngoặc đơn được nhúng nào được đúp bản).

Mặc định là none. Chỉ các siêu người sử dụng có thể thay đổi thiết lập này.

> **Lưu ý**: Các lệnh mà có các lỗi cú pháp đơn giản sẽ không được lưu ký thậm chí bằng thiết lập log_statement = all, vì thông điệp lưu ký chỉ được phát đi sau khi việc phân tích cơ bản đã được thực hiện xong để xác định dạng lệnh. Trong trường hợp của

giao thức truy vấn mở rộng, thiết lập này có lẽ không lưu ký các lệnh mà thất bại trước pha thực thi Execute (nghĩa là, trong quá trình phân tích cú pháp hoặc lên kế hoạch). Hãy thiết lập log_min_error_statement về ERROR (hoặc thấp hơn) để lưu ký các lệnh như vậy.

log_temp_files (integer)

Kiểm soát việc lưu ký các tên và kích cỡ tệp tạm thời. Các tệp tạm thời có thể được tạo ra cho các sắp xếp, hàm băm, và các kết quả truy vấn tạm thời. Một khoản đầu vào lưu ký được làm cho từng tệp tạm thời khi nó bị xóa. Một giá trị zero sẽ lưu ký tất cả các thông tin tệp tạm thời, trong khi các giá trị dương sẽ lưu ký chỉ các tệp mà kích cỡ của chúng lớn hơn hoặc bằng số KB được chỉ định. Thiết lập mặc định là -1, nó vô hiệu hóa việc lưu ký như vậy. Chỉ các siêu người sử dụng có thể thay đổi được thiết lập này.

log_timezone (string)

Thiết lập vùng thời gian được sử dụng cho các dấu thời gian được viết trong lưu ký. Không giống như vùng thời gian timezone, giá trị này là bó rộng rãi, sao cho tất cả các phiên làm việc sẽ đưa ra các dấu thời gian một cách nhất quán. Mặc định là không biết unknown, nó có nghĩa là sử dụng bất kỹ thứ gì mà môi trường hệ thống chỉ định như là vùng thời gian. Xem Phần 8.5.3 để có thêm thông tin. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

18.7.4. Sử dụng đầu ra lưu ký có định dạng CSV

Việc đưa csvlog vào danh sách log_destination đưa ra một cách thức thuận tiện để nhập khẩu các tệp lưu ký vào một bảng cơ sở dữ liệu. Lựa chọn này đưa ra các dòng lưu ký ở định dạng các giá trị phân cách nhau bằng dấu phẩy - CSV (comma - separated - value), với các cột đó: dấu thời gian theo mili giây, tên người sử dụng, tên cơ sở dữ liệu, ID tiến trình, máy chủ cho các máy trạm: số cổng, ID phiên làm việc, số dòng cho từng phiên làm việc, thẻ lệnh, thời gian khởi đầu phiên làm việc, ID giao dịch thực, ID giao dịch thường xuyên, độ nghiêm trọng của lỗi, mã SQLSTATE, thông điệp lỗi, chi tiết thông điệp lỗi, gợi ý, truy vấn nội bộ mà dẫn tới lỗi đó (nếu có), đếm ký tự vị trí lỗi ở đó, ngữ cảnh lỗi, truy vấn của người sử dụng dẫn tới lỗi đó (nếu có và được log_min_error_statement kích hoạt), đếm ký tự vị trí lỗi ở đó, vị trí của lỗi trong mã nguồn PostgreSQL (nếu log_error_verbosity được thiết lập về verbose), và tên ứng dụng. Đây là một định nghĩa bảng ví dụ cho việc lưu trữ đầu ra lưu ký có định dạng CSV:

```
CREATE TABLE postgres_log
(
        log_time timestamp(3) with time zone,
        user_name text,
        database_name text,
        process_id integer,
        connection_from text,
        session_id text,
        session_line_num bigint,
        command_tag text,
        session_start_time timestamp with time zone,
        virtual_transaction_id text,
```

```
        transaction_id bigint,
        error_severity text,
        sql_state_code text,
        message text,
        detail text,
        hint text,
        internal_query text,
        internal_query_pos integer,
        context text,
        query text,
        query_pos integer,
        location text,
        application_name text,
        PRIMARY KEY (session_id, session_line_num)
);
```

Để nhập khẩu một tệp lưu ký vào bảng này, hãy sử dụng lệnh COPY FROM:

```
COPY postgres_log FROM '/full/path/to/logfile.csv' WITH csv;
```

Có vài điều bạn cần làm để chỉ định việc nhập khẩu các tệp lưu ký CSV:

1. Thiết lập log_filename và log_rotation_age để đưa ra một sơ đồ đặt tên nhất quán, có thể đoán trước được cho các tệp lưu ký của bạn. Điều này cho phép bạn đoán trước được tên tệp nào sẽ là và biết được khi nào một tệp lưu ký riêng rẽ là hoàn chỉnh và vì thế sẵn sàng để được nhập khẩu.
2. Thiết lập log_rotation_size về 0 để vô hiệu hóa sự xoay vòng lưu ký dựa vào kích cỡ, khi nó làm cho tên tệp lưu ký khó để đoán trước được.
3. Thiết lập log_truncate_on_rotation về bật (on) sao cho dữ liệu lưu ký cũ sẽ không bị trộn vào với dữ liệu mới trong cùng y hệt tệp đó.
4. Định nghĩa bảng ở trên bao gồm 1 đặc tả khóa chính. Điều này là hữu dụng để bảo vệ chống lại việc nhập khẩu ngẫu nhiên thông tin y hệt 2 lần. Lệnh COPY thực hiện tất cả các dữ liệu mà nó nhập khẩu cùng một thời điểm, nên bất kỳ lỗi nào cũng sẽ gây ra cho toàn bộ sự nhập khẩu đó hỏng. Nếu bạn nhập khẩu một tệp lưu ký theo từng phần và sau đó nhập khẩu tệp đó một lần nữa khi nó hoàn chỉnh, thì sự vi phạm khóa chính sẽ làm cho sự nhập khẩu đó hỏng. Hãy chờ cho tới khi lưu ký đó là hoàn chỉnh và đóng nó lại trước khi nhập khẩu. Thủ tục này cũng sẽ bảo vệ chống lại việc nhập khẩu ngẫu nhiên một dòng theo từng phần mà còn chưa được viết xong hoàn chỉnh, điều cũng có thể làm cho lệnh COPY bị hỏng.

18.8. Số liệu thống kê thời gian chạy

18.8.1. Bộ thu thập số liệu thống kê truy vấn và chỉ số

Các tham số đó kiểm soát các tính năng thu thập số liệu thống kê rộng khắp máy chủ. Khi thu thập số liệu thống kê được kích hoạt, các dữ liệu được tạo ra có thể được truy cập qua họ các kiểu nhìn hệ thống pg_stat và pg_statio. Tham chiếu tới Chương 27 để có thêm thông tin.

track_activities (boolean)

Kích hoạt thu thập thông tin trong lệnh đang tồn tại hiện hành của từng phiên làm việc, cùng

với thời gian mà lệnh đó đã bắt đầu thực thi. Tham số này mặc định là bật (on). Lưu ý rằng thậm chí được kích hoạt, thông tin này là không nhìn thấy đối với tất cả những người sử dụng, mà chỉ thấy đối với các siêu người sử dụng và người sử dụng sở hữu phiên làm việc đang được báo cáo, nên nó sẽ không thể hiện một rủi ro an toàn. Chỉ các siêu người sử dụng có thể thay đổi thiết lập này.

track_activity_query_size (integer)

Chỉ định số byte được giữ lại để theo dõi lệnh đang thực thi hiện hành đối với từng phiên làm việc tích cực, đối với trường pg_stat_activity.current_query. Giá trị mặc định là 1024. Tham số này chỉ có thể được thiết lập khi máy chủ khởi động.

track_counts (boolean)

Kích hoạt thu thập số liệu thống kê trong hoạt động của cơ sở dữ liệu. Tham số này mặc định được bật (on), vì daemon autovacuum cần thông tin được thu thập. Chỉ các siêu người sử dụng có thể thay đổi được thiết lập này.

track_functions (enum)

Kích hoạt việc theo dõi các tính toán và thời gian gọi làm được sử dụng. Chỉ định pl để theo dõi chỉ các hàm ngôn ngữ thủ tục, tất cả cũng theo dõi các hàm ngôn ngữ SQL và C. Mặc định là none, nó vô hiệu hóa việc theo dõi số liệu thống kê của hàm. Chỉ các siêu người sử dụng có thể thay đổi được thiết lập này.

Lưu ý: các hàm ngôn ngữ SQL mà đủ đơn giản để là "thành phần nằm trong" truy vấn đang gọi sẽ không bị theo dõi, bất chấp thiết lập này.

update_process_title (boolean)

Kích hoạt việc cập nhật đầu đề tiến trình mỗi lần một lệnh SQL mới được máy chủ nhận được. Đầu đề tiến trình thường được lệnh ps nhìn thấy, hoặc trong Windows bằng việc sử dụng Process Explorer. Chỉ các siêu người sử dụng có thể thay đổi được thiết lập này.

stats_temp_directory (string)

Thiết lập thư mục để lưu trữ các dữ liệu thống kê tạm thời trong đó. Điều này có thể là một đường dẫn tương đối cho thư mục dữ liệu hoặc một đường dẫn tuyệt đối. Mặc định là pg_stat_tmp. Việc chỉ ra điều này ở một hệ thống tệp dựa vào RAM sẽ làm giảm các yêu cầu I/O vật lý và có thể dẫn tới hiệu năng được cải thiện. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

18.8.2. Giám sát số liệu thống kê

log_statement_stats (boolean)
log_parser_stats (boolean)
log_planner_stats (boolean)
log_executor_stats (boolean)

Đối với từng truy vấn, số liệu thống kê hiệu năng đầu ra của module tương ứng với lưu ký máy chủ. Đây là một công cụ định hình thô, tương tự như tiện ích hệ điều hành getrusage() của Unix. log_statement_stats nêu tổng số liệu thống kê lệnh, trong khi đó những thứ khác nêu số liệu thống kê theo từng module. log_statement_stats không thể được kích hoạt cùng

với bất kỳ lựa chọn theo từng module nào. Tất cả các lựa chọn đó mặc định bị vô hiệu hóa. Chỉ các siêu người sử dụng có thể thay đổi được thiết lập này.

18.9. Việc hút chân không tự động

Các thiết lập này kiểm soát hành vi của tính năng hút chân không tự động. Tham chiếu tới Phần 23.1.5 để có thêm thông tin.

autovacuum (boolean)

Kiểm soát liệu máy chủ có nên chạy daemon trình khởi động hút chân không tự động hay không. Điều này mặc định là bật (on); tuy nhiên, track_counts cũng phải được kích hoạt để sự hút chân không tự động làm việc được. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

Lưu ý rằng thậm chí khi tham số này bị vô hiệu hóa, hệ thống vẫn sẽ khởi tạo các tiến trình hút chân không tự động nếu cần thiết để ngăn chặn ID giao dịch bị bao quanh. Xem Phần 23.1.4 để có thêm thông tin.

log_autovacuum_min_duration (integer)

Nguyên do từng hành động do hút chân không tự động thực thi sẽ được lưu ký lại nếu nó chạy qua ít nhất số lượng mili giây được chỉ định. Việc thiết lập điều này về zero sẽ lưu ký tất cả các hành động hút chân không tự động. Âm một (-1) (mặc định) vô hiệu hóa việc lưu ký các hành động hút chân không tự động. Ví dụ, nếu bạn thiết lập điều này về 250 ms sau đó tất cả các vacuums và phân tích tự động mà chạy 250 ms hoặc lâu hơn sẽ được lưu ký lại. Việc kích hoạt tham số này có thể là hữu dụng trong việc theo dõi hoạt động hút chân không tự động. Thiết lập này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

autovacuum_max_workers (integer)

Chỉ định số lượng cực đại các tiến trình hút chân không tự động (khác với trình khởi tạo hút chân không tự động) mà có thể sẽ chạy trong bất kỳ thời điểm nào. Mặc định là 3. Tham số này chỉ có thể được thiết lập khi máy chủ khởi động.

autovacuum_naptime (integer)

Chỉ định sự trễ tối thiểu giữa các cuộc hút chân không tự động chạy trong bất kỳ cơ sở dữ liệu được đưa ra nào. Trong từng vòng thì daemon kiểm tra cơ sở dữ liệu và đưa ra các lệnh VACUUM và ANALYZE khi cần thiết cho các bảng trong cơ sở dữ liệu đó. Sự trễ được đo đếm theo giây, và mặc định là 1 phút. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

autovacuum_vacuum_threshold (integer)

Chỉ định số lượng tối thiểu các bộ dữ liệu được cập nhật hoặc bị xóa cần thiết để làm bật dậy một VACUUM trong bất kỳ một bảng nào. Mặc định là 50 bộ dữ liệu. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ. Thiết lập này có thể bị ghi đè đối với các bảng riêng rẽ bằng việc thay đổi các tham số lưu trữ.

autovacuum_analyze_threshold (integer)

Chỉ định số lượng tối thiểu các bộ dữ liệu được chèn vào, được cập nhật hoặc bị xóa cần thiết để kích hoạt một ANALYZE trong bất kỳ một bảng nào. Mặc định là 50 bộ dữ liệu. Tham số này chỉ có thể được thiết lập ở tệp postgresql.conf hay trong dòng lệnh của máy chủ. Thiết lập này có thể bị ghi đè đối với các bảng riêng rẽ bằng việc thay đổi các tham số kho lưu trữ.

autovacuum_vacuum_scale_factor (floating point)

Chỉ định một phần nhỏ kích cỡ bảng để thêm vào autovacuum_vacuum_threshold khi quyết định liệu có làm bật dậy một VACUUM hay không. Mặc định là 0.2 (20% kích cỡ bảng). Tham số này chỉ có thể được thiết lập ở tệp postgresql.conf hoặc trong dòng lệnh máy chủ. Thiết lập này có thể bị ghi đè đối với các bảng riêng rẽ bằng việc thay đổi các tham số lưu trữ.

autovacuum_analyze_scale_factor (floating point)

Chỉ định một phần nhỏ kích cỡ bảng để thêm vào autovacuum_analyze_threshold khi quyết định liệu có làm bật dậy một ANALYZE hay không. Mặc định là 0.1 (10% kích cỡ bảng). Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ. Thiết lập này có thể bị ghi đè đối với các bảng riêng rẽ bằng việc thay đổi các tham số lưu trữ.

autovacuum_freeze_max_age (integer)

Chỉ định tuổi cực đại (trong các giao dịch) mà trường của một bảng pg_class.relfrozenxid field có thể đạt tới trước khi một hoạt động VACUUM bị ép buộc để ngăn chặn IP giao dịch bao quanh bên trong bảng đó. Lưu ý là hệ thống đó sẽ khởi tạo các tiến trình hút chân không tự động để ngăn chặn sự bao quanh thậm chí khi sự hút chân không tự động nếu khác sẽ bị vô hiệu hóa.

Vacuum cũng cho phép loại bỏ các tệp cũ từ thư mục con pg_clog, điều giải thích vì sao mặc định là khá thấp 200 triệu giao dịch. Tham số này chỉ có thể được thiết lập khi máy chủ khởi động, nhưng thiết lập đó có thể được giảm nhẹ cho các bảng riêng rẽ bằng việc thay đổi các tham số lưu trữ. Để có thêm thông tin, xem Phần 23.1.4.

autovacuum_vacuum_cost_delay (integer)

Chỉ định giá trị trễ về chi phí sẽ được sử dụng trong các hoạt động VACUUM tự động. Nếu -1 được chỉ định, thì giá trị thông thường vacuum_cost sẽ được sử dụng. Giá trị mặc định là 20 mili giây. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ. Thiết lập này có thể bị ghi đè đối với các bảng riêng rẽ bằng việc thay đổi các tham số lưu trữ.

autovacuum_vacuum_cost_limit (integer)

Chỉ định giá trị giới hạn chi phí sẽ được sử dụng trong các hoạt động tự động VACUUM. Nếu -1 được chỉ định (nó là mặc định), thì giá trị thông thường vacuum_cost_limit sẽ được sử dụng. Lưu ý là giá trị đó được phân phối theo tỷ lệ giữa các trình công việc chạy hút chân không tự động, nếu có nhiều hơn 1, sao cho tổng các giới hạn của từng trình công việc không bao giờ vượt quá giới hạn trong biến này. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ. Thiết lập này có thể bị ghi đè đối với các

bảng riêng rẽ bằng việc thay đổi các tham số lưu trữ.

18.10. Mặc định kết nối máy trạm

18.10.1. Hành vi của lệnh

search_path (string)

Biến này chỉ định trật tự theo đó các sơ đồ được tìm thấy khi một đối tượng (bảng, dạng dữ liệu, hàm, ...) được một tên đơn giản tham chiếu tới khi không sơ đồ nào được chỉ định. Khi có các đối tượng có tên y hệt nhau trong các sơ đồ khác nhau, thì tên được tìm thấy trước tiên trong đường tìm kiếm sẽ được sử dụng. Một đối tượng không nằm trong bất kỳ sơ đồ nào trong đường tìm kiếm chỉ có thể được tham chiếu tới bằng việc chỉ định sơ đồ chứa nó với một cái tên đủ điều kiện (có dấu chấm).

Giá trị cho search_path phải là một danh sách các cái tên sơ đồ được phân cách nhau bằng dấu phẩy. Nếu một tên từ các khoản của danh sách là giá trị đặc biệt $user, thì sau đó sơ đồ có tên được SESSION_USER trả về sẽ được thay thế, nếu có một sơ đồ như vậy. (Nếu không, $user bị bỏ qua).

Sơ đồ catalog hệ thống, pg_catalog, luôn được tìm thấy, bất kể nó được nhắc tới trong đường đó hay không. Nếu nó được nhắc tới trong đường đó thì nó sẽ được tìm thấy theo trật tự được chỉ định. Nếu pg_catalog không có trong đường đó thì nó sẽ được tìm thấy trước khi việc tìm kiếm bất kỳ khoản nào của đường đó.

Tương tự, sơ đồ bảng tạm của phiên làm việc hiện hành, pg_temp_nnn, luôn được tìm thấy nếu nó tồn tại. Nó có thể được liệt kê rõ ràng trong đường đó bằng việc sử dụng tên hiệu pg_temp. Nếu nó không được liệt kê trong đường đó thì sau đó nó có thể được tìm thấy trước tiên (thậm chí trước cả pg_catalog). Tuy nhiên, sơ đồ tạm thời chỉ được tìm thấy cho quan hệ (bảng, kiểu nhìn, tuần tự, ...) và các tên dạng dữ liệu. Nó không bao giờ tìm thấy cho các tên hàm và toán tử.

Khi các đối tượng được tạo ra không có việc chỉ định một sơ đồ đích đặc biệt, thì chúng sẽ được đặt trong sơ đồ đầu tiên được liệt kê trong đường tìm kiếm. Một lỗi được nêu nếu đường tìm kiếm là rỗng.

Giá trị mặc định cho tham số này là '"$user", public' (trong đó phần thứ 2 sẽ bị bỏ qua nếu không có sơ đồ nào có tên public). Điều này hỗ trợ sử dụng một cơ sở dữ liệu được chia sẻ (nơi mà không người sử dụng nào có các sơ đồ riêng, và tất cả chia sẻ sử dụng chung public), các sơ đồ riêng theo từng người sử dụng, và các kết hợp của chúng. Các hiệu ứng khác có thể có được bằng việc chỉnh sửa thiết lập đường tìm kiếm mặc định, hoặc theo từng người sử dụng, hoặc toàn thể.

Giá trị hiệu lực hiện hành của đường tìm kiếm có thể được kiểm tra thông qua hàm SQL current_schemas (xem Phần 9.23). Đây không hoàn toàn là y hệt như việc kiểm tra giá trị của search_path, vì current_schemas chỉ ra cách các khoản xuất hiện trong search_path đã được giải quyết.

Để có nhiều thông tin hơn về điều hành sơ đồ, xem Phần 5.7.

default_tablespace (string)

Biến này chỉ định không gian bảng mặc định trong đó để tạo các đối tượng (các bảng và các chỉ số) khi một lệnh CREATE không chỉ định rõ ràng một không gian bảng.

Giá trị đó hoặc là tên của một không gian bảng, hoặc là một chuỗi rỗng để chỉ định việc sử dụng không gian bảng mặc định của cơ sở dữ liệu hiện hành. Nếu giá trị đó không khớp với bất kỳ tên của không gian bảng nào đang tồn tại, thì PostgreSQL sẽ tự động sử dụng không gian bảng mặc định của cơ sở dữ liệu hiện hành. Nếu một không gian bảng không mặc định được chỉ định, thì người sử dụng phải để CREATE có quyền ưu tiên cho nó, hoặc những cố gắng tạo ra sẽ hỏng.

Biến này không được sử dụng cho các bảng tạm; đối với chúng, temp_tablespaces được tư vấn thay.

Để có thêm thông tin về không gian bảng, xem Phần 21.6.

temp_tablespaces (string)

Biến này chỉ định các không gian bảng trong đó để tạo các đối tượng tạm thời (các bảng tạm và các chỉ số trong các bảng tạm) khi một lệnh CREATE không chỉ định rõ ràng một không gian bảng. Các tệp tạm thời vì các mục đích như sắp xếp các tập hợp dữ liệu lớn cũng sẽ được tạo ra trong các không gian bảng đó.

Giá trị đó là một danh sách các tên của các không giản bảng. Khi có nhiều hơn 1 tên trong danh sách đó, PostgreSQL chọn một số ngẫu nhiên của danh sách đó mỗi lần một đối tượng tạm thời được tạo ra; ngoại trừ là trong một giao dịch, các đối tượng tạm thời được tạo ra thành công sẽ được đặt trong các không gian bảng thành công từ danh sách đó. Nếu yếu tố được chọn của danh sách là một chuỗi rỗng, thì thay vào đó PostgreSQL sẽ tự động sử dụng không gian bảng mặc định của cơ sở dữ liệu hiện hành.

Khi temp_tablespaces được thiết lập một cách tương tác, thì việc chỉ định một không gian bảng chưa tồn tại là một lỗi, khi việc chỉ định một không gian bảng theo đó người sử dụng không có quyền ưu tiên của CREATE. Tuy nhiên, khi sử dụng một giá trị tập hợp trước đó, thì các không gian bảng chưa tồn tại sẽ bị bỏ qua, khi là các không gian bảng theo đó người sử dụng thiếu quyền ưu tiên của CREATE. Đặc biệt, qui tắc này áp dụng khi sử dụng một tập hợp giá trị trong postgresql.conf.

Giá trị mặc định là một chuỗi rỗng, nó tạo ra tất cả các đối tượng tạm thời đang được tạo ra trong không gian bảng mặc định của cơ sở dữ liệu hiện hành.

Xem thêm default_tablespace.

check_function_bodies (boolean)

Tham số này thường là bật (on). Khi được thiết lập là tắt (off), nó vô hiệu hóa sự hợp lệ của chuỗi thân hàm trong quá trình CREATE FUNCTION. Việc vô hiệu hóa sự kiểm tra hợp lệ thỉnh thoảng là hữu dụng để tránh các vấn đề như chuyển tiếp các tham chiếu khi phục hồi các xác định hàm từ sự hỏng hóc.

default_transaction_isolation (enum)

Từng giao dịch SQL có một mức cách li, nó có thể hoặc là "đọc không được thực hiện", "đọc được thực hiện", "đọc lặp đi lặp lại", hoặc "tuần tự hóa". Tham số này kiểm soát mức cách li mặc định của từng giao dịch mới. Mặc định là "đọc được thực hiện".

Tư vấn Chương 13 và SET TRANSACTION để có thêm thông tin.

default_transaction_read_only (boolean)

Một giao dịch SQL chỉ đọc không thể sửa các bảng không phải tạm thời. Tham số này kiểm soát tình trạng mặc định chỉ đọc của từng giao dịch mới. Mặc định là tắt (off) (đọc/ghi). Tư vấn SET TRANSACTION để có thêm thông tin.

session_replication_role (enum)

Kiểm soát bật các trigger có liên quan tới nhân bản và các qui tắc cho phiên làm việc hiện hành. Việc thiết lập biến này đòi hỏi quyền ưu tiên của siêu người sử dụng và các kết quả trong việc bỏ bất kỳ kế hoạch truy vấn nào được nhớ tạm trước đó. Các giá trị có thể là origin (mặc định), replica và local. Xem ALTER TABLE để có thêm thông tin.

statement_timeout (integer)

Bỏ bất kỳ lệnh nào vượt lượng được chỉ định theo mili giây, bắt đầu từ thời điểm lệnh đó tới máy chủ từ máy trạm. Nếu log_min_error_statement được thiết lập về ERROR hoặc thấp hơn, thì lệnh mà hết thời gian cũng sẽ được lưu ký lại. Một giá trị zero (mặc định) sẽ tắt điều này.

Việc thiết lập statement_timeout trong postgresql.conf không được khuyến cáo vì nó ảnh hưởng tới tất cả các phiên làm việc.

vacuum_freeze_table_age (integer)

VACUUM thực hiện một sự quét toàn bộ bảng nếu trường pg_class.relfrozenxid của bảng đã đạt độ tuổi được thiết lập này chỉ định. Mặc định là 150 triệu giao dịch. Dù những người sử dụng có thể thiết lập giá trị này ở bất kỳ đâu từ zero tới 1 tỷ, thì VACUUM vẫn sẽ âm thầm giới hạn giá trị hiệu lực về 95% của autovacuum_freeze_max_age, sao cho một VACUUM định kỳ bằng tay có một cơ hội để chạy trước khi sự hút chân không tự động chống bao quanh được khởi tạo cho bảng đó. Xem Phần 23.1.4 để có thêm thông tin.

vacuum_freeze_min_age (integer)

Chỉ định tuổi bị cắt hết (trong các giao dịch) mà VACUUM sẽ sử dụng để quyết định liệu có thay thế các ID giao dịch bằng FrozenXID khi quét một bảng hay không. Mặc định là 50 triệu giao dịch. Dù những người sử dụng có thể thiết lập giá trị này ở bất kỳ đâu đó từ zero tới 1 tỷ, thì VACUUM cũng sẽ âm thầm giới hạn giá trị hiệu lực này về nửa giá trị của autovacuum_freeze_max_age, sao cho không có một thời gian ngắn không hợp lý nào giữa các cuộc hút chân không tự động bị ép buộc. Xem Phần 23.1.4 để có thêm thông tin.

bytea_output (enum)

Thiết lập định dạng đầu ra cho các giá trị dạng bytea. Các giá trị hợp lệ là hex (mặc định) và escape (định dạng truyền thống của PostgreSQL). Xem Phần 8.4 để có thêm thông tin. Dạng bytea luôn chấp nhận cả 2 định dạng ở đầu vào, bất kể thiết lập này.

xmlbinary (enum)

Thiết lập cách các giá trị nhị phân sẽ bị ép vào trong XML. Điều này áp dụng cho ví dụ khi các giá trị bytea sẽ được các hàm xmlelement hoặc xmlforest chuyển đổi thành XML. Các giá trị có khả năng là base64 và hex, cả 2 được xác định trong tiêu chuẩn XML Schema. Mặc định là base64. Để có thêm thông tin về các hàm có liên quan tới XML, xem Phần 9.14.

Lựa chọn thực sự ở đây hầu hết là một vấn đề ý thích, chỉ bị ràng buộc bằng những ràng buộc có khả năng trong các ứng dụng máy trạm. Cả 2 phương pháp đều hỗ trợ tất cả giá trị có khả năng, dù việc mã hex là thứ gì đó lớn hơn so với việc mã base64.

xmloption (enum)

Thiết lập liệu DOCUMENT hoặc CONTENT có là ngầm hiểu khi chuyển đổi giữa XML và các giá trị chuỗi ký tự hay không. Xem Phần 8.13 để có mô tả về điều này. Các giá trị hợp lệ là DOCUMENT và CONTENT. Mặc định là CONTENT.

Theo tiêu chuẩn SQL, lệnh để thiết lập lựa chọn này là

SET XML OPTION { DOCUMENT | CONTENT };

Cú pháp này cũng sẵn sàng trong PostgreSQL

18.10.2. Định vị và định dạng

DateStyle (string)

Thiết lập định dạng hiển thị cho các giá trị ngày tháng và thời gian, cũng như các qui tắc cho việc diễn giải các giá trị đầu vào ngày tháng mù mờ. Vì các lý do lịch sử, biến này có 2 thành phần độc lập: đặc tả định dạng đầu ra (ISO, Postgres, SQL, hoặc German [Đức]) và đặc tả cho đầu vào/đầu ra đối với trật tự năm/tháng/ngày (DMY, MDY, hoặc YMD). Chúng có thể được thiết lập tách rời nhau hoặc cùng với nhau. Các từ khóa châu Âu (Euro) và thuộc về châu Âu (European) là các đồng nghĩa cho DMY; các từ khóa Mỹ (US), không châu Âu, và không thuộc về châu Âu là đồng nghĩa cho MDY. Xem Phần 8.5 để có thêm thông tin. Mặc định được xây dựng sẵn là ISO, MDY, nhưng initdb sẽ khởi tạo tệp cấu hình đó với một thiết lập tương ứng với hành vi bản địa lc_time được chọn.

IntervalStyle (enum)

Thiết lập định dạng hiển thị cho các giá trị khoảng thời gian. Giá trị sql_standard sẽ đưa ra các hằng khoảng thời gian tiêu chuẩn SQL khớp với đầu ra. Giá trị postgres (là mặc định) sẽ đưa ra đầu ra khớp với các phiên bản PostgreSQL trước 8.4 khi tham số DateStyle từng được thiết lập về ISO. Giá trị postgres_verbose sẽ đưa ra đầu ra khớp với các phiên bản PostgreSQL trước 8.4 khi tham số DateStyle từng được thiết lập cho đầu ra không phải ISO. Giá trị iso_8601 sẽ đưa ra đầu ra khớp với khoảng thời gian "định dạng với các bộ chỉ định" được xác định trong Phần 4.4.3.2 của ISO 8601.

Tham số IntervalStyle cũng ảnh hưởng tới sự diễn giải đầu vào khoảng thời gian mù mờ. Xem Phần 8.5.4 để có thêm thông tin.

timezone (string)

Thiết lập vùng thời gian cho việc hiển thị và diễn giải các dấu thời gian. Mặc định là không biết unknown, nó có nghĩa là sử dụng bất kỳ thứ gì mà môi trường hệ thống chỉ định như là

vùng thời gian. Xem Phần 8.5.3 để có thêm thông tin.

timezone_abbreviations (string)

Thiết lập bộ các từ viết tắt vùng thời gian sẽ được máy chủ chấp nhận đối với đầu vào ngày tháng thời gian. Mặc định là 'Default', nó là bộ sưu tập làm việc ở hầu hết trên thế giới; cũng có cả Úc và Ấn Độ ('Australia' and 'India'), và các bộ sưu tập khác có thể được xác định cho một cài đặt đặc thù. Xem Phụ lục B để có thêm thông tin.

extra_float_digits (integer)

Tham số này tinh chỉnh số các chữ số được hiển thị cho các giá trị dấu chấm động, bao gồm cả float4, float8, và các dạng dữ liệu địa lý. Giá trị tham số được thêm vào số các chữ số tiêu chuẩn (FLT_DIG hoặc DBL_DIG là phù hợp). Giá trị đó có thể được thiết lập cao tới 3, bao gồm các chữ số có nghĩa một phần; điều này đặc biệt hữu dụng cho việc đánh đống các dữ liệu động mà cần được phục hồi chính xác. Hoặc nó có thể được thiết lập về số âm để bỏ các ký tự số không mong muốn.

client_encoding (string)

Thiết lập mã (tập hợp các ký tự) phía máy trạm. Mặc định để sử dụng mã hóa cơ sở dữ liệu.

lc_messages (string)

Thiết lập ngôn ngữ theo đó các thông điệp sẽ được hiển thị. Các giá trị được chấp nhận là độc lập với hệ thống; xem Phần 22.1 để có thêm thông tin. Nếu biến này được thiết lập về chuỗi rỗng (nó là mặc định) thì giá trị đó được thừa hưởng từ môi trường thực thi của máy chủ theo một cách thức phụ thuộc hệ thống.

Trong một vài hệ thống, chủng loại bản địa này không tồn tại. Việc thiết lập biến này vẫn sẽ làm việc, nhưng sẽ không có kết quả. Hơn nữa, có một cơ hội rằng các thông điệp không được diễn giải cho sự tồn tại của ngôn ngữ mong muốn. Trong trường hợp đó bạn sẽ tiếp tục thấy các thông điệp tiếng Anh.

Chỉ các siêu người sử dụng có thể thay đổi thiết lập này, vì nó ảnh hưởng tới các thông điệp được gửi tới lưu ký máy chủ cũng như tới máy trạm, và một giá trị không phù hợp có thể làm mờ khả năng đọc các lưu ký máy chủ.

lc_monetary (string)

Thiết lập bản địa để sử dụng cho việc định dạng lượng tiền, ví dụ với họ các hàm to_char. Các giá trị chấp nhận được là phụ thuộc hệ thống; xem Phần 22.1 để có thêm thông tin. Nếu biến này được thiết lập về chuỗi rỗng (nó là mặc định) thì giá trị đó được thừa hưởng từ môi trường thực thi của máy chủ theo một cách thức phụ thuộc hệ thống.

lc_numeric (string)

Thiết lập bản địa để sử dụng cho việc định dạng các số, ví dụ với họ các hàm to_char. Các giá trị chấp nhận được là phụ thuộc hệ thống; xem Phần 22.1 để có thêm thông tin. Nếu biến này được thiết lập về chuỗi rỗng (nó là mặc định) thì giá trị đó được thừa hưởng từ môi trường thực thi của máy chủ theo một cách thức phụ thuộc hệ thống.

lc_time (string)

Thiết lập bản địa để sử dụng cho việc định dạng ngày tháng và thời gian, ví dụ với họ các hàm to_char. Các giá trị chấp nhận được là phụ thuộc hệ thống; xem Phần 22.1 để có thêm thông tin. Nếu biến này được thiết lập về chuỗi rỗng (nó là mặc định) thì giá trị đó được thừa hưởng từ môi trường thực thi của máy chủ theo một cách thức phụ thuộc hệ thống.

default_text_search_config (string)

Chọn cấu hình tìm kiếm văn bản được các biến thể đó của các hàm văn bản sử dụng mà không có một đối số rõ ràng chỉ định cấu hình đó. Xem Chương 12 để có thêm thông tin. Mặc định được xây dựng sẵn là pg_catalog.simple, nhưng initdb sẽ khởi tạo tệp cấu hình đó với một thiết lập tương ứng với bản địa lc_ctype locale được chọn, nếu một cấu hình khớp bản địa đó có thể được nhận diện.

18.10.3. Mặc định khác

dynamic_library_path (string)

Nếu một module khả tải động cần phải được mở và tên tệp được chỉ định trong lệnh CREATE FUNCTION hoặc LOAD không có một thành phần thư mục (như, tên không có một dấu chéo), thì hệ thống sẽ tìm đường dẫn này cho tệp được yêu cầu.

Giá trị cho dynamic_library_path phải là một danh sách các đường dẫn thư mục tuyệt đối được phân cách nhau bằng dấu 2 chấm (hoặc dấu chấm phẩy trong Windows). Nếu một yếu tố danh sách bắt đầu bằng chuỗi đặc biệt $libdir, thì thư mục thư viện gói được biên dịch trong PostgreSQL được thay thế cho $libdir; đây là nơi các module được phân phối được cài đặt của PostgreSQL tiêu chuẩn cung cấp. (Hãy sử dụng pg_config --pkglibdir để tìm ra tên của thư mục này). Ví dụ:

dynamic_library_path = '/usr/local/lib/postgresql:/home/my_project/lib:$libdir'

hoặc, trong môi trường Windows:

dynamic_library_path = 'C:\tools\postgresql;H:\my_project\lib;$libdir'

Giá trị mặc định cho tham số này là '$libdir'. Nếu giá trị đó được thiết lập về một chuỗi rỗng, thì tìm kiếm đường dẫn tự động sẽ được tắt (off).

Tham số này có thể được các siêu người sử dụng thay đổi khi chạy, nhưng một thiết lập được thực hiện theo cách đó sẽ chỉ duy trì cho tới kết thúc kết nối của máy trạm, nên phương pháp này sẽ được giữ lại cho các mục đích phát triển. Cách thức được khuyến cáo để thiết lập tham số này là trong tệp postgresql.conf.

gin_fuzzy_search_limit (integer)

Làm nhẹ giới hạn trên kích cỡ của tập hợp được các quét chỉ số GIN trả về. Xem Phần 53.4 để có thêm thông tin.

local_preload_libraries (string)

Biến này chỉ định một hoặc nhiều hơn các thư viện được chia sẻ sẽ được tải lên trước khi khởi động kết nối. Nếu hơn một thư viện sẽ được tải lên, thì các tên của nó được phân cách bằng dấu phẩy. Tất cả các tên thư viện được chuyển thành chữ thường trừ phi có các dấu ngoặc kép. Tham số này không thể bị thay đổi sau khi bắt đầu một phiên làm việc đặc biệt.

Vì điều này không phải là lựa chọn của chỉ một siêu người sử dụng, các thư viện mà có thể được tải lên bị hạn chế cho những ai đang nghe trong thư mục con plugins của thư mục thư viện tiêu chuẩn của cài đặt đó. (Đây là trách nhiệm của người quản trị cơ sở dữ liệu để đảm bảo rằng chỉ các thư viện "an toàn" sẽ được cài đặt ở đó). Các khoản đầu vào trong local_preload_libraries có thể chỉ định thư mục này một cách rõ ràng, ví dụ $libdir/plugins/mylib, hoặc chỉ chỉ định tên thử viện - mylib có thể có hiệu ứng y hệt như $libdir/plugins/mylib.

Không giống như shared_preload_libraries, không có ưu thế hiệu năng cho việc tải lên một thư viện khi bắt đầu phiên làm việc thay vì khi nó lần đầu được sử dụng. Thay vào đó, ý định của tính năng này là để cho phép gỡ rối hoặc đo đếm hiệu năng các thư viện sẽ được tải lên trong các phiên làm việc đặc thù mà không có một lệnh LOAD rõ ràng được đưa ra. Ví dụ, việc sửa lỗi có thể được kích hoạt cho tất cả các phiên làm việc dưới một cái tên người sử dụng bằng việc thiết lập tham số này với ALTER USER SET.

Nếu một thư viện được chỉ định không được tìm thấy, thì cố gắng kết nối sẽ hỏng.

Mỗi thư viện được PostgreSQL hỗ trợ có một "khối ma thuật" được kiểm tra để đảm bảo tính tương thích. Vì lý do này, các thư viện không phải PostgreSQL không thể được tải lên theo cách này.

18.11. Quản lý khóa

deadlock_timeout (integer)

Đây là lượng thời gian, theo mili giây, để chờ đợi trong một cái khóa trước khi kiểm tra để xem liệu có một điều kiện khóa chết hay không. Sự kiểm tra khóa chết là tương đối đắt giá, nên máy chủ không chạy nó mỗi lần chờ đợi một khóa. Chúng ta giả thiết tối ưu rằng các khóa chết không là phổ biến trong các ứng dụng sản xuất và chỉ chờ trong khóa một lúc trước khi kiểm tra một khóa chết. Việc tăng giá trị này làm giảm lượng thời gian bỏ ra trong các kiểm tra khóa chết không cần thiết, nhưng làm chậm việc báo cáo các lỗi khóa chết thực sự. Mặc định là 1 giây, nó có thể là về giá trị nhỏ nhất mà bạn có thể muốn trong thực tế. Trong một máy chủ có tải nặng thì bạn có thể muốn tăng nó lên. Lý tưởng thì các thiết lập đó nên vượt quá thời gian giao dịch điển hình của bạn, sao cho cải thiện được những sự kỳ dị mà một khóa sẽ được phát hành trước khi người chờ đợi quyết định kiểm tra khóa chết.

Khi log_lock_waits được thiết lập, tham số này cũng xác định độ dài thời gian để chờ trước khi một thông điệp lưu ký được phát ra về sự chờ khóa. Nếu bạn đang cố gắng điều tra sự trễ của việc khóa thì bạn có thể muốn thiết lập một deadlock_timeout ngắn hơn bình thường.

max_locks_per_transaction (integer)

Bảng khóa được chia sẻ theo dõi các khóa trong max_locks_per_transaction * (các đối tượng max_connections + max_prepared_transactions) (như, các bảng); vì thế, không lớn hơn nhiều đối tượng khác biệt này có thể bị khóa cùng một thời điểm. Tham số này kiểm soát số lượng trung bình các khóa đối tượng được phân bổ cho từng giao dịch; các giao dịch riêng rẽ có thể khóa nhiều đối tượng hơn miễn là các khóa của tất cả các giao tiếp vừa khớp trong

bảng khóa. Đây không phải là số hàng có thể bị khóa; giá trị đó là không có giới hạn. Mặc định, 64, đã chứng minh về lịch sử là đủ, nhưng bạn có thể cần tăng giá trị này nếu bạn có các máy trạm động chạm tới nhiều bảng khác trong một giao dịch duy nhất. Tham số này chỉ có thể được thiết lập khi máy chủ khởi động.

Việc tăng tham số này có thể làm cho PostgreSQL yêu cầu nhiều bộ nhớ được chia sẻ của System V hơn so với cấu hình mặc định hệ điều hành của bạn cho phép. Xem Phần 17.4.1 để có thêm thông tin về cách tinh chỉnh các tham số đó, nếu cần thiết.

Khi chạy một máy chủ nhàn rỗi, bạn phải thiết lập tham số này về giá trị y hệt hoặc cao hơn so với trên máy chủ chính. Nếu không, các truy vấn sẽ không được phép trong máy chủ nhàn rỗi.

18.12. Tính tương thích nền tảng và phiên bản

18.12.1. Các phiên bản trước đây của PostgreSQL

array_nulls (boolean)

Điều này kiểm soát liệu trình phân tích cú pháp đầu vào mảng có nhận thức được NULL không trong các dấu ngoặc kép như việc chỉ định một phần tử mảng null hay không. Mặc định, điều này là bật (on), cho phép các giá trị mảng chứa các giá trị null sẽ được nhập vào. Tuy nhiên, PostgreSQL các phiên bản trước 8.2 đã không hỗ trợ các giá trị null trong mảng, và vì thể có thể đối xử với NULL như việc chỉ định một thành phần mảng thông thường với giá trị chuỗi "NULL". Vì tính tương thích ngược với các ứng dụng đòi hỏi hành xử cũ, biến này có thể được tắt (off).

Lưu ý là có khả năng tạo ra giá trị mạng có các giá trị null thậm chí khi biến này tắt (off).

backslash_quote (enum)

Điều này kiểm soát liệu một dấu ngoặc có thể được thể hiện trong một hằng chuỗi \' hay không. Cách thức được ưa thích, theo tiêu chuẩn SQL để thể hiện một dấu ngoặc là bằng việc đúp bản nó ('') mà PostgreSQL về mặt lịch sử cũng chấp nhận \'. Tuy nhiên, sử dụng \' sẽ có các rủi ro an toàn vì trong một vài mã bộ ký tự máy trạm, có các ký tự nhiều byte mà trong đó byte cuối tương đương về số với \ ASCII. Nếu mã phía máy trạm không thoát đúng thì một cuộc tấn công tiêm SQL là có khả năng. Rủi ro này có thể ngăn chặn được bằng việc làm cho máy chủ từ chối các truy vấn trong đó một dấu ngoặc sẽ được thoát bằng một dấu chéo ngược. Các giá trị được phép của backslash_quote là bật (on) (luôn cho phép \'), tắt (off) (luôn từ chối), và safe_encoding (chỉ cho phép nếu việc mã hóa máy trạm không cho phép \ ASCII bên trong một ký tự nhiều byte). safe_encoding là thiết lập mặc định.

Lưu ý là trong một hằng chuỗi tuân thủ tiêu chuẩn, \ chỉ có nghĩa là \. Tham số này chỉ ảnh hưởng tới việc điều khiển các hằng không tuân thủ tiêu chuẩn, bao gồm cú pháp chuỗi thoát (E'...').

default_with_oids (boolean)

Điều này kiểm soát liệu CREATE TABLE và CREATE TABLE AS có bao gồm một cột OID trong các bảng mới được tạo hay không, nếu WITH OIDS hoặc WITHOUT OIDS đều không được chỉ

định. Nó cũng xác định liệu các OID có được đưa vào trong các bảng được SELECT INTO tạo ra hay không. Tham số đó mặc định là tắt (off); trong PostgreSQL 8.0 và trước đó, nó mặc định là bật (on).

Sử dụng các OID trong các bảng người sử dụng được cân nhắc không tán thành, nên hầu hết các cài đặt sẽ để biến này bị vô hiệu hóa. Các ứng dụng đòi hỏi các OID cho một bảng đặc biệt sẽ chỉ định WITH OIDS khi tạo bảng. Biến này có thể được kích hoạt vì tính tương thích với các ứng dụng cũ không tuân theo hành vi này.

escape_string_warning (boolean)

Khi bật, một cảnh báo được đưa ra nếu một dấu chéo ngược (\) xuất hiện trong một hằng chuỗi thông thường (cú pháp '...') và standard_conforming_strings là tắt. Mặc định là bật.

Các ứng dụng muốn sử dụng dấu chéo ngược để thoát sẽ được sửa đổi để sử dụng cú pháp chuỗi thoát (E'...'), vì hành vi mặc định của các chuỗi thông thường sẽ thay đổi trong phiên bản trong tương lai vì tính tương thích của PostgreSQL. Biến này có thể được kích hoạt để giúp dò tìm ra các ứng dụng sẽ ngắt.

lo_compat_privileges (boolean)

Trong các phiên bản PostgreSQL trước 9.0, các đối tượng lớn đã không có các quyền ưu tiên truy cập và từng, có hiệu lực, đọc được và ghi được đối với tất cả những người sử dụng. Việc thiết lập biến này về bật (on) sẽ vô hiệu hóa các kiểm tra quyền ưu tiên mới, vì tính tương thích với các phiên bản trước đó. Mặc định là tắt.

Việc thiết lập biến này không vô hiệu hóa tất cả các kiểm tra an toàn có liên quan tới các đối tượng lớn - chỉ các đối tượng theo đó hành vi mặc định đã thay đổi trong PostgreSQL 9.0. Ví dụ, lo_import() và lo_export() cần các quyền ưu tiên của siêu người sử dụng độc lập với thiết lập này.

sql_inheritance (boolean)

Thiết lập này kiểm soát liệu các tham chiếu bảng không được tô vẽ có được cân nhắc để đưa vào các bảng con kế thừa hay không. Mặc định là bật (on), nó có nghĩa là các bảng con được đưa vào (vì thế, một hậu tố * mặc định được giả định). Nếu được tắt, các bảng con sẽ không được đưa vào (vì thế, một tiền tố ONLY được giả định). Tiêu chuẩn SQL yêu cầu các bảng con được đưa vào, nên thiết lập tắt (off) là không đặc biệt tuân thủ, nhưng nó được đưa ra vì tính tương thích với các phiên bản PostgreSQL tới 7.1. Xem Phần 5.8 để có thêm thông tin.

Việc tắt sql_inheritance là không được tán thành, vì hành vi đó từng được thấy sẽ là dễ bị lỗi cũng như đối nghịch với tiêu chuẩn SQL. Các thảo luận về hành vi kế thừa ở đâu đó nữa trong sách chỉ dẫn này thường giả thiết nó là bật (on).

standard_conforming_strings (boolean)

Điều này kiểm soát liệu các hằng chuỗi thông thường ('...') có ứng xử với các dấu chéo ngược đúng hay không, như được chỉ định trong tiêu chuẩn SQL. Mặc định hiện nay là tắt (off), làm cho PostgreSQL có hành vi hành xử theo lịch sử của nó đối với các dấu chéo ngược như các ký tự thoát. Mặc định sẽ thay đổi thành bật (on) trong một phiên bản trong

tương lai để cải thiện tính tương thích với tiêu chuẩn SQL. Các ứng dụng có thể kiểm tra tham số này để xác định cách mà các hằng chuỗi sẽ được xử lý. Sự hiện diện của tham số này cũng có thể được coi như một chỉ số rằng cú pháp chuỗi thoát (E'...') được hỗ trợ. Cú pháp chuỗi thoát (Phần 4.1.2.2) sẽ được sử dụng nếu một ứng dụng muốn các dấu chéo ngược sẽ được đối xử như các ký tự thoát.

synchronize_seqscans (boolean)

Điều này cho phép các quét tuần tự các bảng lớn đồng bộ hóa với nhau, sao cho các quét đồng thời đọc cùng y hệt khối trong khoảng thời gian y hệt và vì thế chia sẻ tải công việc của I/O. Khi điều này được kích hoạt, một sự quét có thể bắt đầu ở giữa bảng và sau đó "bao quanh" kết thúc để bao trùm tất cả các hàng, sao cho để đồng bộ hóa được với hoạt động của các quét được tiến triển. Điều này làm các thay đổi không thể đoán trước được trong trật tự hàng được các truy vấn trả về mà không có mệnh đề ORDER BY. Việc thiết lập tham số này về tắt (off) đảm bảo hành vi các phiên bản trước 8.3 theo đó một sự quét tuần tự luôn bắt đầu từ đầu của bảng. Mặc định là bật (on).

18.12.2. Nền tảng và tính tương thích máy trạm

transform_null_equals (boolean)

Khi bật, các biểu thức mẫu biểu expr = NULL (hoặc NULL = expr) sẽ được đối xử như expr IS NULL, đó là, chúng trả về đúng nếu expr định giá về giá trị null, và sai nếu khác. Hành vi tuân thủ đặc thù SQL đúng của expr = NULL sẽ luôn trả về null (không biết). Vì thế mặc định tham số này là tắt (off).

Tuy nhiên, các mẫu biểu được lọc trong Microsoft Access sinh ra các truy vấn dường như sử dụng expr = NULL để kiểm thử đối với các giá trị null, nên nếu bạn sử dụng giao diện đó để truy cập cơ sở dữ liệu thì bạn có thể muốn bật lựa chọn này lên. Vì các biểu thức mẫu biểu expr = NULL luôn trả về giá trị null (bằng việc sử dụng diễn giải tiêu chuẩn SQL), chúng rất không hữu dụng và không xuất hiện thường xuyên trong các ứng dụng thông thường nên lựa chọn này ít gây hại trong thực tiễn. Nhưng nếu những người sử dụng mới thường xuyên lẫn lộn về ngữ nghĩa của các biểu thức có liên quan tới các giá trị null, thì lựa chọn này là tắt theo mặc định.

Lưu ý rằng lựa chọn này chỉ ảnh hưởng chính xác tới mẫu biểu form = NULL, chứ không tới các toán tử so sánh hay các biểu thức khác mà tính tương đương với một vài biểu thức có liên quan tới toán tử bằng (như IN). Vì thế lựa chọn này không phải là một sửa lỗi chung đối với việc lập trình tồi.

Hãy tham chiếu tới Phần 9.2 để có các thông tin liên quan.

18.13. Lựa chọn thiết lập trước

"Các tham số" sau là chỉ đọc, và được xác định khi PostgreSQL được biên dịch hoặc khi nó được cài đặt. Như vậy, chúng từng được loại trừ khỏi tệp mẫu postgresql.conf. Các lựa chọn đó nêu các khía cạnh khác nhau về hành xử của PostgreSQL mà có thể có quan tâm tới các ứng dụng nhất định, đặc biệt các giao diện mặt tiền (front-end) quản trị.

block_size (integer)

Nêu kích cỡ của một khối đĩa. Nó được giá trị BLCKSZ xác định khi xây dựng máy chủ. Giá trị mặc định là 8192 byte. Ý nghĩa của một số biến cấu hình (như shared_buffers) bị ảnh hưởng bởi block_size. Xem Phần 18.4 để có thông tin.

integer_datetimes (boolean)

Nêu bản địa theo đó việc sắp xếp các dữ liệu văn bản được thực hiện. Xem Phần 22.1 để có thêm thông tin. Giá trị này được xác định khi một cơ sở dữ liệu được tạo ra.

lc_collate (string)

Nêu bản địa trong đó việc sắp xếp các dữ liệu văn bản được thực hiện. Xem Phần 22.1 để có thêm thông tin.

Giá trị này được xác định khi một cơ sở dữ liệu được tạo ra.

lc_ctype (string)

Nêu bản địa xác định các phân loại ký tự. Xem Phần 22.1 để có thêm thông tin. Giá trị này được xác định khi một cơ sở dữ liệu được tạo ra. Điều này thường sẽ là y hệt như lc_collate, nhưng đối với các ứng dụng đặc biệt mà nó có thể được thiết lập khác.

max_function_args (integer)

Nêu số cực đại các đối số hàm. Nó được giá trị của FUNC_MAX_ARGS xác định khi xây dựng máy chủ. Giá trị mặc định là 100 đối số.

max_identifier_length (integer)

Nêu độ dài cực đại mã định danh. Nó được xác định như một độ dài nhỏ hơn giá trị của NAMEDATALEN khi xây dựng máy chủ. Giá trị mặc định của NAMEDATALEN là 64; vì thế giá trị mặc định max_identifier_length là 63 byte, nó có thể là ít hơn so với 63 ký tự khi sử dụng các mã nhiều byte.

max_index_keys (integer)

Nêu số lượng cực đại các khóa chỉ số. Nó được giá trị INDEX_MAX_KEYS xác định khi xây dựng máy chủ. Giá trị mặc định là 32 khóa.

segment_size (integer)

Nêu số lượng các khối (các trang) có thể được lưu trữ trong phân đoạn của một tệp. Nó được giá trị của RELSEG_SIZE xác định khi xây dựng máy chủ. Kích cỡ cực đại của một tệp phân đoạn theo byte là bằng segment_size nhân với block_size; Mặc định điều này là 1GB.

server_encoding (string)

Nêu việc mã hóa cơ sở dữ liệu (bộ ký tự). Nó được xác định khi cơ sở dữ liệu được tạo ra. Thông thường, các máy trạm chỉ cần được kết nối với giá trị client_encoding.

server_version (string)

Nêu số phiên bản của máy chủ. Nó được giá trị PG_VERSION xác định khi xây dựng máy chủ.

server_version_num (integer)

Nêu số phiên bản của máy chủ như một số nguyên. Nó được giá trị của PG_VERSION_NUM xác định khi xây dựng máy chủ.

wal_block_size (integer)

Nêu kích cỡ của khối đĩa WAL. Nó được giá trị của XLOG_BLCKSZ xác định khi xây dựng máy chủ. Giá trị mặc định là 8192 byte.

wal_segment_size (integer)

Nêu số lượng các khối (các trang) trong một tệp phân đoạn của WAL. Tổng kích cỡ của tệp phân đoạn WAL theo byte bằng với wal_segment_size nhân với wal_block_size; Mặc định điều này là 16MB. Xem Phần 29.4 để có thêm thông tin.

18.14. Lựa chọn tùy biến

Tính năng này đã được thiết kế để cho phép các tham số thường không được biết đối với PostgreSQL để được thêm vào bằng các module bổ sung thêm (add-on) (như các ngôn ngữ thủ tục). Điều này cho phép các module bổ sung thêm sẽ được thiết lập cấu hình theo cách thức tiêu chuẩn.

custom_variable_classes (string)

Biến này chỉ định một hoặc vài tên lớp sẽ được sử dụng cho các biến tùy ý, ở dạng một danh sách được phân cách nhau bằng dấu phẩy. Một biến tùy ý là một biến bình thường không được biết đối với PostgreSQL một cách thích đáng nhưng được một vài module bổ sung thêm sử dụng. Các biến như vậy phải có các tên gồm có một tên lớp, một dấu chấm, và một tên biến, custom_variable_classes chỉ định tất cả các tên lớp trong sử dụng trong một cài đặt đặc biệt. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

Khó khăn với việc thiết lập các biến tùy ý trong postgresql.conf là việc tệp đó phải được đọc trước khi các module bổ sung thêm đã được tải lên, và vì thế các biến tùy biến có thể thường bị từ chối như là không được biết. Khi custom_variable_classes được thiết lập, máy chủ sẽ chấp nhận các định nghĩa của các biến tùy đó bên trong từng lớp được chỉ định. Các biến đó sẽ được đối xử như là các chỗ trống và sẽ không có chức năng cho tới khi module mà xác định chúng được tải lên. Khi một module cho một lớp đặc thù được tải lên, thì nó sẽ bổ sung thêm các định nghĩa biến đúng phù hợp cho tên lớp của nó, biến đổi bất kỳ giá trị chỗ trống nào theo các định nghĩa đó, và đưa ra các cảnh báo cho bất kỳ chỗ trống nào lớp của nó còn chưa được thừa nhận mà vẫn còn.

Đây là một ví dụ về những gì postgresql.conf có thể có khi sử dụng các biến tùy ý:

```
custom_variable_classes = 'plpgsql,plperl'
plpgsql.variable_conflict = use_variable
plperl.use_strict = true
plruby.use_strict = true # generates error: unknown class name
```

18.15. Lựa chọn của lập trình viên

Các tham số sau có ý định để làm việc trong mã nguồn PostgreSQL, và trong một số trường hợp để hỗ trợ bằng việc phục hồi các cơ sở dữ liệu bị thiệt hại nghiêm trọng. Không nên có lý do để sử

dụng chúng trong một cơ sở dữ liệu sản xuất. Như vậy, chúng đã bị loại trừ khỏi tệp mẫu postgresql.conf. Lưu ý rằng nhiều trong số các tham số đó đòi hỏi các cờ biên dịch nguồn đặc biệt để làm việc được.

allow_system_table_mods (boolean)

Cho phép sửa đổi cấu trúc các bảng hệ thống. Điều này được initdb sử dụng. Tham số này chỉ có thể được thiết lập khi máy chủ khởi động.

debug_assertions (boolean)

Bật các kiểm tra xác nhận khác nhau. Đây là một trợ giúp khắc phục lỗi. Nếu bạn đang trải nghiệm các vấn đề hoặc các hỏng hóc kỳ lạ thì bạn có thể muốn bật điều này lên, như nó có thể mở ra các sai sót lập trình. Để sử dụng tham số này, macro USE_ASSERT_CHECKING phải được xác định khi PostgreSQL được xây dựng (được lựa chọn configure hoàn tất --enable-cassert). Lưu ý rằng debug_assertions mặc định là bật (on) nếu PostgreSQL từng được xây dựng với các xác nhận được kích hoạt.

ignore_system_indexes (boolean)

Bỏ qua các chỉ số hệ thống khi đọc các bảng hệ thống (nhưng vẫn cập nhật các chỉ số khi sửa đổi các bảng). Điều này là hữu dụng khi phục hồi từ các chỉ số hệ thống bị tổn hại. Tham số này không thể bị thay đổi sau khi phiên làm việc bắt đầu.

post_auth_delay (integer)

Nếu không là zero, thì một sự trễ nhiều giây này sẽ xảy ra khi một tiến trình mới của máy chủ được khởi tạo, sau khi nó tiến hành thủ tục xác thực. Điều này có ý định trao cho các lập trình viên một cơ hội để gắn vào tiến trình máy chủ bằng một trình gỡ lỗi. Tham số này không thể bị thay đổi sau khi phiên làm việc bắt đầu.

pre_auth_delay (integer)

Nếu không phải zero, một sự trễ nhiều giây này sẽ xảy ra chỉ sau một tiến trình máy chủ mới bị rẽ nhánh, trước khi nó tiến hành thủ tục xác thực. Điều này có ý định để trao cho các lập trình viên một cơ hội để gắn với tiến trình máy chủ bằng một trình gỡ lỗi để theo dõi hành vi sai trái trong xác thực. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

trace_notify (boolean)

Sinh ra một lượng lớn đầu ra gỡ lỗi cho các lệnh LISTEN và NOTIFY. client_min_messages hoặc log_min_messages phải là DEBUG1 hoặc thấp hơn để gửi đầu ra này cho các lưu ký máy trạm hoặc máy chủ. một cách tương ứng.

trace_recovery_messages (enum)

Cho phép việc lưu ký đầu ra việc gỡ lỗi có liên quan tới sự phục hồi mà nếu không có thể sẽ không được lưu ký. Tham số này cho phép người sử dụng ghi đè lên thiết lập bình thường của log_min_messages, nhưng chỉ cho các thông điệp đặc thù. Điều này có ý định để sử dụng trong việc gỡ lỗi Hot Standby. Các giá trị hợp lệ là DEBUG5, DEBUG4, DEBUG3,

DEBUG2, DEBUG1, và LOG. Mặc định, LOG, không ảnh hưởng tới các quyết định lưu ký hoàn toàn. Các giá trị khác phục hồi các thông điệp gỡ lỗi có liên quan tới sự phục hồi ưu tiên đó hoặc cao hơn sẽ được lưu ký dường như chúng đã có ưu tiên LOG; đối với các thiết lập chung của log_min_messages thì các kết quả này trong việc gửi vô điều kiện chúng tới lưu ký máy chủ. Tham số này chỉ có thể được thiết lập trong tệp postgresql.conf hoặc trong dòng lệnh máy chủ.

trace_sort (boolean)

Nếu bật, hãy phát thông tin về sử dụng tài nguyên trong quá trình các hoạt động sắp xếp. Tham số này chỉ sẵn sàng nếu macro TRACE_SORT đã được xác định khi PostgreSQL đã được biên dịch. (Tuy nhiên, TRACE_SORT hiện được xác định mặc định).

trace_locks (boolean)

Nếu bật, hãy phát thông tin về sử dụng khóa. Thông tin được đánh đống bao gồm dạng hoạt động khóa, dạng khóa và mã định danh duy nhất của đối tượng đang được khóa hoặc mở khóa. Cũng được đưa vào là các mặt nạ bit cho các dạng khóa được trao rồi trong đối tượng này cũng như cho các dạng khóa được chờ đợi trong đối tượng này. Đối với từng dạng khóa, một tính toán số các khóa được trao và các khóa chờ cũng được đánh đống như là tổng số. Một ví dụ về đầu ra tệp lưu ký được chỉ ra ở đây:

```
LOG:  LockAcquire: new: lock(0xb7acd844) id(24688,24696,0,0,0,1)
        grantMask(0) req(0,0,0,0,0,0,0)=0 grant(0,0,0,0,0,0,0)=0
        wait(0) type(AccessShareLock)
LOG:  GrantLock: lock(0xb7acd844) id(24688,24696,0,0,0,1)
        grantMask(2) req(1,0,0,0,0,0,0)=1 grant(1,0,0,0,0,0,0)=1
        wait(0) type(AccessShareLock)
LOG:  UnGrantLock: updated: lock(0xb7acd844) id(24688,24696,0,0,0,1)
        grantMask(0) req(0,0,0,0,0,0,0)=0 grant(0,0,0,0,0,0,0)=0
        wait(0) type(AccessShareLock)
LOG:  CleanUpLock: deleting: lock(0xb7acd844) id(24688,24696,0,0,0,1)
        grantMask(0) req(0,0,0,0,0,0,0)=0 grant(0,0,0,0,0,0,0)=0
        wait(0) type(INVALID)
```

Các chi tiết của cấu trúc đang được đánh đống có thể được thấy trong src/include/storage/lock.h.

Tham số này chỉ sẵn sàng nếu macro LOCK_DEBUG đã được xác định khi PostgreSQL từng được biên dịch.

trace_lwlocks (boolean)

Nếu bật, hãy phát thông tin về sử dụng khóa nhẹ. Các khóa nhẹ ban đầu có ý định cung cấp sự loại trừ truy cập đôi bên đối với các cấu trúc dữ liệu bộ nhớ được chia sẻ.

Tham số này chỉ sẵn sàng nếu macro LOCK_DEBUG đã được xác định khi PostgreSQL đã được biên dịch.

trace_userlocks (boolean)

Nếu bật, hãy phát thông tin về sử dụng khóa của người sử dụng. Đầu ra là y hệt như đối với trace_locks, chỉ cho các khóa cố vấn.

Tham số này chỉ sẵn sàng nếu macro LOCK_DEBUG đã được xác định khi PostgreSQL đã

được biên dịch.

trace_lock_oidmin (integer)

Nếu được thiết lập, không theo dõi các khóa đối với các bảng bên dưới OID này. (sử dụng để tránh đầu ra trong các bảng hệ thống).

Tham số chỉ sẵn sàng nếu macro LOCK_DEBUG đã được xác định khi PostgreSQL đã được biên dịch.

trace_lock_table (integer)

Theo dõi vô điều kiện các khóa trong bảng này (OID).

Tham số này chỉ sẵn sàng nếu macro LOCK_DEBUG đã được xác định khi PostgreSQL đã được biên dịch.

debug_deadlocks (boolean)

Nếu được thiết lập, đánh thành đống thông tin về tất cả các khóa hiện hành khi sự hết hạn thời gian (timeout) của một khóa chết xảy ra.

Tham số này chỉ sẵn sàng nếu macro LOCK_DEBUG đã được xác định khi PostgreSQL đã được biên dịch.

log_btree_build_stats (boolean)

Nếu được thiết lập, lưu ký số liệu thống kê sử dụng tài nguyên hệ thống (bộ nhớ và CPU) trong các hoạt động B-tree khác nhau.

Tham số này chỉ sẵn sàng nếu macro BTREE_BUILD_STATS đã được xác định khi PostgreSQL đã được biên dịch.

wal_debug (boolean)

Nếu bật, phát đầu ra việc gỡ lỗi có liên quan tới WAL. Tham số này chỉ sẵn sàng nếu macro WAL_DEBUG đã được xác định khi PostgreSQL đã được biên dịch.

zero_damaged_pages (boolean)

Sự dò tìm ra đầu đề một trang bị thiệt hại thường làm cho PostgreSQL nêu một lỗi, bỏ đi lệnh hiện hành. Việc thiết lập zero_damaged_pages về bật (on) làm cho hệ thống thay vào đó sẽ nêu một cảnh báo, zero đi tới trang bị thiệt hại, và tiếp tục xử lý. Hành vi này sẽ phá hủy dữ liệu, ấy là tất cả các hàng ở trang bị thiệt hại. Nhưng nó cho phép bạn vượt qua lỗi đó và truy xuất các hàng từ bất kỳ trang nào không bị thiệt hại mà có thể hiện diện trong bảng đó. Vì thế là hữu dụng cho việc phục hồi các dữ liệu nếu sự hỏng đã xảy ra vì một lỗi phần cứng hoặc phần mềm. Bạn thường không nên đặt điều này cho tới khi bạn đã từ bỏ hy vọng phục hồi dữ liệu từ các trang bị thiệt hại của một bảng. Thiết lập mặc định là tắt (off), và nó chỉ có thể được một siêu người sử dụng thay đổi.

18.16. Lựa chọn ngắn

Để thuận tiện cũng có sự chuyển lựa chọn dòng lệnh ký tự duy nhất sẵn sàng cho một vài tham số.

Chúng được mô tả trong Bảng 18-2. Một vài trong số các lựa chọn đó tồn tại vì các lý do lịch sử, và sự hiện diện của chúng như một lựa chọn ký tự duy nhất không nhất thiết chỉ ra một sự đồng thuận

để sử dụng lựa chọn đó một cách nặng ký.

Bảng 18-2. Khóa lựa chọn ngắn gọn

Lựa chọn ngắn gọn	Tương ứng
-A x	debug_assertions = x
-B x	shared_buffers = x
-d x	log_min_messages = DEBUGx
-e	datestyle = euro
-fb, -fh, -fi, -fm, -fn, -fs, -ft	enable_bitmapscan = off, enable_hashjoin = off, enable_indexscan = off, enable_mergejoin = off, enable_nestloop = off, enable_seqscan = off, enable_tidscan = off
-F	fsync = off
-h x	listen_addresses = x
-i	listen_addresses = ’*’
-k x	unix_socket_directory = x
-l	ssl = on
-N x	max_connections = x
-O	allow_system_table_mods = on
-p x	port = x
-P	ignore_system_indexes = on
-s	log_statement_stats = on
-S x	work_mem = x
-tpa, -tpl, -te	log_parser_stats = on, log_planner_stats = on, log_executor_stats = on
-W x	post_auth_delay = x

Chương 19. Xác thực máy trạm

Khi ứng dụng của một máy trạm kết nối với máy chủ cơ sở dữ liệu, nó chỉ định tên người sử dụng cơ sở dữ liệu PostgreSQL nào nó muốn kết nối như, phần lớn hệt như cách mà một người đăng nhập vào một máy tính Unix như một người sử dụng đặc biệt. Trong môi trường SQL thì tên người sử dụng cơ sở dữ liệu tích cực xác định các quyền ưu tiên truy cập tới các đối tượng của cơ sở dữ liệu - xem Chương 20 để có thêm thông tin. Vì thế, là cơ bản để hạn chế những người sử dụng cơ sở dữ liệu nào có thể được kết nối.

> **Lưu ý**: Như được giải thích trong Chương 20, PostgreSQL thực sự làm quản lý quyền ưu tiên theo "các vai trò" (role). Trong chương này, chúng tôi nhất quán sử dụng người sử dụng cơ sở dữ liệu để ngụ ý "vai trò với quyền ưu tiên LOGIN".

Xác thực là tiến trình theo đó máy chủ cơ sở dữ liệu thiết lập nhận diện máy trạm, và ở mức độ nào đó xác định liệu ứng dụng máy trạm đó (hoặc người sử dụng mà chạy ứng dụng máy trạm đó) có được phép kết nối tới tên người sử dụng cơ sở dữ liệu đó theo yêu cầu hay không.

PostgreSQL đưa ra một số phương pháp xác thực máy trạm khác nhau. Phương pháp được sử dụng để xác thực kết nối một máy trạm đặc biệt có thể được lựa chọn trên cơ sở địa chỉ máy đặt chỗ host của (máy trạm), cơ sở dữ liệu, và người sử dụng.

Các tên người sử dụng cơ sở dữ liệu PostgreSQL về logic là khác với các tên người sử dụng của hệ điều hành theo đó máy chủ đang chạy. Nếu tất cả những người sử dụng của một máy chủ nhất định nào đó cũng có các tài khoản trong máy chủ đó, thì có nghĩa để chỉ định các tên người sử dụng cơ sở dữ liệu mà khớp với các tên người sử dụng của hệ điều hành đó. Tuy nhiên, một máy chủ mà chấp nhận các kết nối ở xa có thể có nhiều người sử dụng cơ sở dữ liệu mà không có tài khoản cục bộ đối với hệ điều hành đó, và trong các trường hợp như vậy thì không cần kết nối giữa các tên người sử dụng cơ sở dữ liệu với các tên của hệ điều hành.

19.1. Tệp pg_hba.conf

Xác thực máy trạm được một tệp cấu hình kiểm soát, theo truyền thống nó được đặt tên là pg_hba.conf và được lưu trữ trong thư mục cơ sở dữ liệu của bó cơ sở dữ liệu. (HBA là cho xác thực dựa vào đặt chỗ host). Một tệp mặc định pg_hba.conf được cài đặt khi thư mục dữ liệu được initdb khởi tạo. Có khả năng đặt tệp cấu hình xác thực ở nơi khác, dù vậy; xem tham số cấu hình hba_file.

Định dạng chung của tệp pg_hba.conf là một tập hợp các bản ghi, mỗi tệp một dòng. Các dòng trống bị bỏ qua, như là bất kỳ văn bản nào sau ký tự chú giải #. Các bản ghi không thể được tiếp tục xuyên các dòng. Một bản ghi được tạo nên từ một số trường được phân cách nhau bằng các dấu trống và/hoặc các tab. Các trường có thể có ký tự trắng nếu giá trị trường đó nằm trong ngoặc kép. Việc đưa vào dấu ngoặc kép các từ khóa trong một cơ sở dữ liệu hoặc trường tên người sử dụng (như, tất cả hoặc nhân bản) làm cho từ đó mất ký tự đặc biệt của nó, và chỉ khớp với một cơ sở dữ liệu hoặc người sử dụng với tên đó.

Mỗi bản ghi chỉ định một dạng kết nối, một dải địa chỉ IP máy trạm (nếu phù hợp cho dạng kết nối đó), một tên cơ sở dữ liệu, một tên người sử dụng, và phương pháp xác thực sẽ được sử dụng cho

các kết nối khớp với các tham số đó. Bản ghi đầu tiên với một dạng kết nối trùng khớp, không "đi qua" (fall-through) hoặc "sao lưu" (backup): nếu một bản ghi được chọn và xác thực hỏng, thì các bản ghi tiếp sau sẽ không được xem xét. Nếu không bản ghi nào khớp, thì truy cập bị từ chối.

Một bản ghi có thể có một trong 7 định dạng sau:

```
local           database  user  auth-method                          [auth-options]
host            database  user  CIDR-address auth-method             [auth-options]
hostssl         database  user  CIDR-address auth-method             [auth-options]
hostnossl       database  user  CIDR-address auth-method             [auth-options]
host            database  user  IP-address  IP-mask auth-method      [auth-options]
hostssl         database  user  IP-address  IP-mask auth-method      [auth-options]
hostnossl       database  user  IP-address  IP-mask auth-method      [auth-options]
```

Ý nghĩa của các trường là như sau:

local

Bản ghi này khớp với các ý định kết nối có sử dụng các khe cắm (socket) miền Unix. Không có một bản ghi dạng này, thì các kết nối khe cắm miền Unix sẽ bị vô hiệu hóa.

host

Bản ghi này khớp với các cố gắng kết nối được thực hiện bằng việc sử dụng TCP/IP. Các bản ghi của host khớp với những cố gắng kết nối hoặc SSL hoặc không SSL.

> **Lưu ý**: Các kết nối TCP/IP ở xa sẽ không có khả năng trừ phi máy chủ được khởi động với một giá trị phù hợp cho tham số cấu hình listen_addresses, vì hành vi mặc định là nghe các kết nối TCP/IP chỉ trong địa chỉ lặp ngược (loopback) cục bộ localhost.

hostssl

Bản ghi này khớp với các cố gắng kết nối được làm bằng việc sử dụng TCP/IP, nhưng chỉ khi kết nối đó được làm với mã hóa SSL.

Để sử dụng lựa chọn này thì máy chủ phải được xây dựng với sự hỗ trợ của SSL. Hơn nữa, SSL phải được kích hoạt lúc khởi động máy chủ bằng việc thiết lập tham số cấu hình ssl (xem Phần 17.8 để có thêm thông tin).

hostnossl

Dạng bản ghi này có hành vi ngược lại với hostssl; nó chỉ khớp với các cố gắng kết nối được làm qua TCP/IP mà không sử dụng SSL.

database

Chỉ định (các) tên cơ sở dữ liệu nào mà bản ghi này khớp. Giá trị all chỉ định rằng nó khớp với tất cả các cơ sở dữ liệu. Giá trị sameuser chỉ định rằng bản ghi khớp nếu cơ sở dữ liệu được yêu cầu có tên y hệt như người sử dụng được yêu cầu. Giá trị samerole chỉ định rằng người sử dụng được yêu cầu phải là một thành viên của vai trò với tên y hệt như cơ sở dữ liệu được yêu cầu đó. (samegroup là lỗi thời nhưng vẫn còn được chấp nhận viết mẫu tự cho samerole). Giá trị replication chỉ định rằng bản ghi khớp nếu một kết nối nhân bản được yêu cầu (lưu ý là các kết nối nhân bản không chỉ định bất kỳ cơ sở dữ liệu đặc biệt nào). Nếu khác, đây là tên của một cơ sở dữ liệu PostgreSQL đặc biệt. Nhiều tên cơ sở dữ liệu có thể

được cung cấp bằng việc phân cách chúng với các dấu phẩy. Một tệp tách biệt chứa các tên cơ sở dữ liệu có thể được chỉ định bằng việc đặt trước tên tệp đó với dấu @.

user

Chỉ định (các) tên người sử dụng cơ sở dữ liệu mà bản ghi này khớp. Giá trị all chỉ định rằng nó khớp với tất cả các những người sử dụng. Nếu khác, điều này hoặc là tên của người sử dụng cơ sở dữ liệu đặc biệt, hoặc một tên nhóm có dấu + đứng ở trước. (Gợi nhớ rằng không có sự khác biệt thực sự nào giữa những người sử dụng và các nhóm trong PostgreSQL; Một dấu + thực sự có nghĩa là "khớp với bất kỳ vài trò nào mà là các thành viên trực tiếp hoặc gián tiếp của vai trò này", trong khi một tên không có dấu + chỉ khớp với vai trò đặc biệt đó. Nhiều tên người sử dụng có thể được cung cấp bằng việc phân cách chúng bằng các dấu phẩy. Một tệp riêng có chứa các tên người sử dụng có thể được chỉ định bằng việc đặt trước tên tệp với dấu @.

CIDR-address

Chỉ định dải địa chỉ IP máy tính trạm mà bản ghi này khớp. Trường này chứa một địa chỉ IP trong ký hiệu thập phân có dấu chấm tiêu chuẩn và độ dài mặt nạ một CIDR. (Các địa chỉ IP chỉ có thể được chỉ định bằng số, không như các tên miền hoặc máy chủ host). Độ dài mặt nạ chỉ ra số các bit có trật tự cao của địa chỉ IP máy trạm mà phải khớp. Các bit ở bên phải của thứ này phải là zero trong địa chỉ IP được đưa ra. Không có không gian trống bất kỳ nào ở giữa địa chỉ IP, dấu /, và độ dài mặt nạ CIDR.

Thay vì một CIDR-address, bạn có thể viết samehost để khớp bất kỳ địa chỉ nào trong số các địa chỉ IP của riêng máy chủ đó, hoặc samenet để khớp với bất kỳ địa chỉ nào trong bất kỳ mạng con nào mà máy chủ đó trực tiếp được kết nối tới.

Các ví dụ điển hình của một CIDR-address là 172.20.143.89/32 cho một host duy nhất, hoặc 172.20.143.0/24 cho một mạng nhỏ, hoặc 10.6.0.0/16 cho một mạng lớn hơn, 0.0.0.0/0 ("tất cả các số 0") đại diện cho tất cả các địa chỉ. Để chỉ định một host duy nhất, hãy sử dụng một mặt nạ CIDR 32 bit cho IPv4 hoặc 128 bit cho IPv6. Trong một địa chỉ mạng, không bỏ qua các số 0 đi sau.

Một địa chỉ IP được đưa ra theo định dạng IPv4 sẽ khớp với các kết nối IPv6 mà có địa chỉ tương ứng, ví dụ 127.0.0.1 sẽ khớp với địa chỉ ::ffff:127.0.0.1 của IPv6. Một khoản đầu vào được đưa ra trong dải định dạng IPv6 sẽ chỉ khớp với các kết nối IPv6, thậm chí nếu địa chỉ được đại diện đó nằm trong dải IPv4-IPv6. Lưu ý rằng các khoản đầu vào ở định dạng IPv6 sẽ bị khước từ nếu thư viện C của hệ thống không hỗ trợ cho các địa chỉ IPv6.

Trường này chỉ áp dụng cho các bản ghi host, hostssl, và hostnossl.

IP-address
IP-mask

Các trường này có thể được sử dụng như một lựa chọn thay thế cho ký hiệu địa chỉ CIDR. Thay vì việc chỉ định độ dài mặt nạ, mặt nạ thực sự được chỉ định trong một cột riêng rẽ. Ví dụ, 255.0.0.0 đại diện cho một độ dài mặt nạ CIDR của IPv4 là 8, và 255.255.255.255 đại diện cho một độ dài mặt nạ CIDR là 32.

Các trường đó chỉ áp dụng cho các bản ghi của host, hostssl, và hostnossl.

auth-method

Chỉ định phương pháp xác thực để sử dụng khi một kết nối khớp với bản ghi này. Các lựa chọn có khả năng sẽ được tóm tắt ở đây; các chi tiết có trong Phần 19.3.

trust

Cho phép kết nối không có điều kiện. Phương pháp này cho phép bất kỳ ai cũng có thể kết nối được tới máy chủ cơ sở dữ liệu PostgreSQL để đăng nhập như bất kỳ người sử dụng PostgreSQL nào mà họ muốn, không cần một mật khẩu hoặc bất kỳ sự xác thực nào khác. Xem Phần 19.3.1 để có thông tin chi tiết.

Reject

Khước từ kết nối không điều kiện. Điều này là hữu dụng cho "việc lọc ra" các host nhất định từ một nhóm, ví dụ một dòng reject có thể khóa một host nhất định khỏi việc kết nối, trong khi một dòng sau đó cho phép các host còn lại kết nối trong một mạng nhất định.

md5

Đòi hỏi máy trạm cung cấp một mật khẩu được mã hóa bằng MD5 để xác thực. Xem Phần 19.3.2 để có thêm thông tin chi tiết.

password

Đòi hỏi máy trạm cung cấp một mật khẩu không được mã hóa để xác thực. Vì mật khẩu được gửi ở dạng văn bản thô qua mạng, điều này sẽ không được sử dụng trong các mạng tin cậy. Xem Phần 19.3.2 để có thêm thông tin chi tiết.

gss

Sử dụng GSSAPI để xác thực người sử dụng. Điều này chỉ sẵn sàng cho các kết nối TCP/IP. Xem Phần 19.3.3 để có thêm các chi tiết.

sspi

Sử dụng SSPI để xác thực người sử dụng. Điều này chỉ sẵn sàng trong Windows. Xem Phần 19.3.4 để có các chi tiết.

krb5

Sử dụng Kerberos V5 để xác thực người sử dụng. Điều này chỉ sẵn sàng cho các kết nối TCP/IP. Xem Phần 19.3.5 để có thêm chi tiết.

ident

Có tên người sử dụng hệ điều hành của máy trạm (cho các kết nối TCP/IP bằng việc liên hệ với máy chủ đó trên máy trạm, còn cho các kết nối cục bộ bằng việc có nó từ hệ điều hành) và kiểm tra xem liệu nó có khớp với tên người sử dụng cơ sở dữ liệu được yêu cầu hay không. Xem Phần 19.3.6 để có thêm các chi tiết.

ldap

Xác thực bằng việc sử dụng một máy chủ LDAP. Xem Phần 19.3.7 để có thêm các chi tiết.

radius

Xác thực bằng việc sử dụng một máy chủ RADIUS. Xem Phần 19.3.8 để có thêm các chi tiết.

cert

Xác thực bằng việc sử dụng các chứng thực SSL máy trạm. Xem Phần 19.3.9 để có thêm chi tiết.

pam

Xác thực bằng việc sử dụng dịch vụ các Module Xác thực Cài cắm được - PAM (Pluggable Authentication Modules) được hệ điều hành cung cấp. Xem Phần 19.3.10 để có thêm thông tin.

auth-options

Sau trường auth-options, có thể có (các) trường dạng name=value mà chỉ định các lựa chọn cho phương pháp xác thực. Các chi tiết về các lựa chọn nào sẽ sẵn sàng cho các phương pháp xác thực nào xuất hiện bên dưới.

Các tệp được các cấu trúc @ đưa vào sẽ được đọc như các danh sách tên, nó có thể được tìm kiếm bằng các ký tự trắng hoặc dấu phẩy. Các ghi chú sẽ được nêu bằng dấu #, hệt như trong pg_hba.conf, và các cấu trúc @ lồng nhau sẽ được phép. Trừ phi tên tệp sau @ là một đường dẫn tuyệt đối, nó sẽ được lấy là tương đối cho thư mục có chứa tệp tham chiếu.

Vì các bản ghi pg_hba.conf được kiểm tra tuần tự cho từng cố gắng kết nối, nên trật tự các bản ghi là quan trọng. Thông thường, các bản ghi sớm hơn sẽ có các tham số trùng khớp kết nối chặt chẽ và các phương pháp xác thực yếu hơn, trong khi các bản ghi sau sẽ có các tham số trùng khớp lỏng lẻo hơn và các phương pháp xác thực mạnh hơn. Ví dụ, một người có thể muốn sử dụng xác thực trust cho các kết nối TCP/IP cục bộ nhưng đòi hỏi một mật khẩu cho các kết nối TCP/IP ở xa. Trong trường hợp này một bản ghi chỉ định xác thực trust cho các kết nối từ 127.0.0.1 có thể xuất hiện trước một bản ghi chỉ định xác thực mật khẩu cho một dải rộng lớn hơn các địa chỉ IP được phép của máy trạm.

Tệp pg_hba.conf được đọc khi khởi động và khi tiến trình chính của máy chủ nhận được một tín hiệu SIGHUP. Nếu bạn sửa tệp đó trong một hệ thống đang tích cực, thì bạn sẽ cần đánh tín hiệu cho postmaster (bằng sử dụng pg_ctl reload hoặc kill -HUP) để làm cho nó đọc lại tệp đó.

Mẹo: Để kết nối tới một cơ sở dữ liệu đặc biệt, một người sử dụng phải không chỉ vượt qua được các kiểm tra pg_hba.conf, mà còn phải có quyền ưu tiên CONNECT cho cơ sở dữ liệu đó. Nếu bạn muốn giới hạn những người sử dụng nào đó có thể kết nối tới các cơ sở dữ liệu đó, thì thường dễ dàng hơn hãy kiểm soát điều này bằng việc trao/thu hồi quyền ưu tiên CONNECT hơn là đặt ra các qui tắc trong các khoản đầu vào của pg_hba.conf.

Một vài ví dụ về các khoản đầu vào pg_hba.conf được chỉ ra trong Ví dụ 19-1. Xem phần tiếp sau để có các chi tiết về các phương pháp xác thực khác nhau.

Ví dụ 19-1. Ví dụ các khoản đầu vào pg_hba.conf

```
# Allow any user on the local system to connect to any database with
# any database user name using Unix-domain sockets (the default for local
# connections).
#
# TYPE          DATABASE        USER    CIDR-ADDRESS                IP-MASK METHOD
local           all             all                                 trust

# The same using local loopback TCP/IP connections.
#
# TYPE          DATABASE        USER    CIDR-ADDRESS                IP-MASK METHOD
host            all             all     127.0.0.1/32                trust

# The same as the previous line, but using a separate netmask column
#
# TYPE          DATABASE        USER    IP-ADDRESS                  IP-MASK METHOD
host            all             all     127.0.0.1 255.255.255.255   trust

# Allow any user from any host with IP address 192.168.93.x to connect
# to database "postgres" as the same user name that ident reports for
# the connection (typically the operating system user name).
#
# TYPE          DATABASE        USER    CIDR-ADDRESS                IP-MASK METHOD
host            postgres        all     192.168.93.0/24             ident

# Allow any user from host 192.168.12.10 to connect to database
# "postgres" if the user's password is correctly supplied.
#
# TYPE          DATABASE        USER    CIDR-ADDRESS                IP-MASK METHOD
host            postgres        all     192.168.12.10/32            md5

# In the absence of preceding "host" lines, these two lines will
# reject all connections from 192.168.54.1 (since that entry will be
# matched first), but allow Kerberos 5 connections from anywhere else
# on the Internet. The zero mask causes no bits of the host IP
# address to be considered, so it matches any host.
#
# TYPE          DATABASE        USER    CIDR-ADDRESS                IP-MASK METHOD
host            all             all     192.168.54.1/32             reject
host            all             all     0.0.0.0/0                   krb5

# Allow users from 192.168.x.x hosts to connect to any database, if
# they pass the ident check. If, for example, ident says the user is
# "bryanh" and he requests to connect as PostgreSQL user "guest1", the
# connection is allowed if there is an entry in pg_ident.conf for map
# "omicron" that says "bryanh" is allowed to connect as "guest1".

# TYPE          DATABASE        USER    CIDR-ADDRESS                IP-MASK METHOD
host            all             all     192.168.0.0/16              ident map=omicron

# If these are the only three lines for local connections, they will
# allow local users to connect only to their own databases (databases
# with the same name as their database user name) except for administrators
# and members of role "support", who can connect to all databases. The file
# $PGDATA/admins contains a list of names of administrators. Passwords
# are required in all cases.
#
# TYPE          DATABASE        USER    CIDR-ADDRESS                IP-MASK METHOD
local           sameuser        all                                 md5
local           all             @admins                             md5
local           all             +support                            md5
# The last two lines above can be combined into a single line:
```

```
local       all             @admins,+support                md5

# The database column can also use lists and file names:
local       db1,db2,@demod  all
```

19.2. Bản đồ tên người sử dụng

Khi sử dụng một hệ thống xác thực ngoài như Ident hoặc GSSAPI, tên của người sử dụng hệ điều hành đã khởi tạo kết nối có thể không hệt như tên người sử dụng cơ sở dữ liệu mà anh ta cần tới để kết nối. Trong trường hợp này, bản đồ một người sử dụng có thể được áp dụng để ánh xạ tên người sử dụng hệ điều hành tới một người sử dụng cơ sở dữ liệu. Để sử dụng việc ánh xạ tên người sử dụng, hãy chỉ định map=map-name trong trường các lựa chọn trong pg_hba.conf. Lựa chọn này được hỗ trợ cho tất cả các phương pháp xác thực mà nhận các tên người sử dụng ngoài. Vì các ánh xạ khác nhau có thể là cần thiết cho các kết nối khác nhau, nên tên của bản đồ sẽ được sử dụng được chỉ định trong tham số map-name trong pg_hba.conf để chỉ bản đồ nào sẽ sử dụng cho từng kết nối riêng rẽ.

Các bản đồ tên người sử dụng được xác định trong tệp bản đồ thực, mặc định nó được đặt tên là pg_ident.conf và được lưu trữ trong thư mục dữ liệu bó máy. (Tuy nhiên, có khả năng đặt tệp bản đồ đó ở nơi khác; xem tham số cấu hình ident_file). Tệp bản đồ thực có các dòng dạng chung là:

```
map-name system-username database-username
```

Các ghi chú và dấu trắng được điều khiển theo cách y hệt như trong pg_hba.conf. Còn map-name là một tên tùy ý sẽ được sử dụng để tham chiếu tới việc ánh xạ này trong pg_hba.conf. 2 trường khác chỉ định tên người sử dụng của một hệ điều hành và tên người sử dụng của một cơ sở dữ liệu trùng khớp. Tên bản đồ map-name y hệt có thể được sử dụng lặp đi lặp lại để chỉ định nhiều ánh xạ người sử dụng bên trong một bản đồ duy nhất.

Không có hạn chế về bao nhiêu người sử dụng cơ sở dữ liệu một người sử dụng hệ điều hành được đưa ra có thể tương ứng với, và ngược lại. Vì thế, các khoản đầu vào trong một bản đồ sẽ được nghĩ là có nghĩa như "người sử dụng hệ điều hành này được phép kết nối như người sử dụng cơ sở dữ liệu này", thay vì ngụ ý rằng chúng là tương đương nhau. Kết nối đó sẽ được phép nếu có bất kỳ khoản đầu vào nào của bản đồ có cặp đôi tên người sử dụng đó lấy được từ hệ thống xác thực ngoài với tên người sử dụng cơ sở dữ liệu mà người sử dụng đó đã yêu cầu kết nối.

Nếu trường system-username bắt đầu bằng dấu chéo (/), thì phần còn lại của trường đó được đối xử như một biểu thức thông thường. (Xem Phần 9.7.3.1 để có các chi tiết về cú pháp biểu thức thông thường của PostgreSQL). Biểu thức thông thường có thể bao gồm một sự nắm bắt duy nhất, hoặc biểu thức con mà có thể sau đó được tham chiếu tới trong trường database-username như là \1. Điều này cho phép ánh xạ nhiều tên người sử dụng trong một dòng duy nhất, điều đặc biệt là hữu dụng cho các thay thế cú pháp đơn giản. Ví dụ, các khoản đầu vào

```
mymap       /^(.*)@mydomain\.com$       \1
mymap       /^(.*)@otherdomain\.com$    guest
```

sẽ loại bỏ phần miền đối với những người sử dụng với các tên người sử dụng hệ thống mà có kết thúc bằng @mydomain.com, và cho phép bất kỳ người sử dụng nào mà tên hệ thống của người sử dụng đó có kết thúc bằng @otherdomain.com để đăng nhập vào như là khách guest.

> **Mẹo**: Nhớ trong đầu rằng mặc định, một biểu thức thông thường có thể khớp với chỉ một phần của một chuỗi. Thường là khôn ngoan để sử dụng ^ và $, như được chỉ ra trong ví dụ ở trên, để ép sự khớp sẽ cho tên người sử dụng toàn bộ hệ thống.

Tệp pg_ident.conf được đọc khi khởi động và khi tiến trình chính của máy chủ nhận được một tín hiệu SIGHUP. Nếu bạn sửa tệp đó trong một hệ thống đang hoạt động, thì bạn sẽ cần đánh tín hiệu cho postmaster (bằng việc sử dụng pg_ctl reload hoặc kill -HUP) để làm cho nó đọc lại tệp đó.

Tệp pg_ident.conf có thể được sử dụng cùng với tệp pg_hba.conf trong Ví dụ 19-1 được chỉ ra trong Ví dụ 19-2. Trong ví dụ này, bất kỳ ai đã đăng nhập vào một máy trong mạng 192.168 mà không có tên người sử dụng hệ điều hành là bryanh, ann, hoặc robert có lẽ không được trao quyền truy cập. Người sử dụng Unix robert chỉ có thể được phép truy cập khi anh ta cố kết nối như ann. Người sử dụng bryanht có thể được phép kết nối hoặc như bryanh hoặc như guest1.

Ví dụ 19-2. Tệp ví dụ pg_ident.conf

```
# MAPNAME       SYSTEM-USERNAME         PG-USERNAME
omicron         bryanh                  bryanh
omicron         ann                     ann
# bob has user name robert on these machines
omicron         robert                  bob
# bryanh can also connect as guest1
omicron         bryanh                  guest1
```

19.3. Phương pháp xác thực

Các phần con sau đây mô tả các phương pháp xác thực chi tiết hơn.

19.3.1. Xác thực tin cậy

Khi xác thực trust được chỉ định, PostgreSQL giả thiết rằng bất kỳ ai cũng có thể kết nối tới máy chủ được ủy quyền để truy cập cơ sở dữ liệu với bất kỳ tên người sử dụng cơ sở dữ liệu nào mà họ chỉ định (thậm chí các tên của siêu người sử dụng). Tất nhiên, những hạn chế được thực hiện trong các cột người sử dụng và cơ sở dữ liệu vẫn áp dụng được. Phương pháp này chỉ nên được sử dụng khi có một sự bảo vệ mức hệ điều hành phù hợp trong các kết nối tới máy chủ đó.

Xác thực trust bản thân nó là phù hợp và rất thuận tiện cho các kết nối cục bộ trong một máy tính trạm của một người sử dụng duy nhất. Thường bản thân nó là không phù hợp trong một máy tính nhiều người sử dụng. Tuy nhiên, bạn có thể có khả năng sử dụng trust thậm chí trong một máy tính nhiều người sử dụng, nếu bạn hạn chế sự truy cập tới tệp khe cắm (socket) miền Unix của máy chủ đó bằng việc sử dụng các quyền của hệ thống tệp. Để làm điều này, hãy thiết lập các tham số cấu hình unix_socket_permissions (và có khả năng unix_socket_group) như được mô tả trong Phần 18.3. Hoặc bạn có thể thiết lập tham số cấu hình unix_socket_directory để đặt tệp khe cắm vào một thư mục có hạn chế phù hợp.

Việc thiết lập các quyền của hệ thống tệp chỉ giúp cho các kết nối khe cắm Unix. Các kết nối TCP/IP cục bộ không bị các quyền của hệ thống tệp hạn chế. Vì thế, nếu bạn muốn sử dụng các quyền của hệ thống tệp cho an toàn cục bộ, hãy loại bỏ dòng host ... 127.0.0.1 ... khỏi pg_hba.conf, hoặc thay đổi nó thành một phương pháp xác thực không tin cậy (non-trust).

Xác thực trust chỉ phù hợp cho các kết nối TCP/IP nếu bạn tin cậy mọi người sử dụng trong mọi máy tính được phép kết nối tới máy chủ đó bằng các dòng pg_hba.conf chỉ định trust. Hiếm khi hợp lý để sử dụng trust cho bất kỳ kết nối TCP/IP nào khác với các kết nối từ localhost (127.0.0.1).

19.3.2. Xác thực mật khẩu

Các phương pháp xác thực dựa vào mật khẩu là md5 và password. Các phương pháp đó vận hành tương tự ngoại trừ cách mà mật khẩu được gửi đi qua kết nối đó, ấy là được băm MD5 và văn bản thô, một cách tương ứng.

Nếu bạn có quan tâm về các cuộc tấn công "đánh hơi" (sniffing) mật khẩu thì md5 được ưu tiên. Mật khẩu thô luôn nên được tránh nếu có thể. Tuy nhiên, md5 không thể được sử dụng với tính năng db_user_namespace. Nếu kết nối được mã hóa SSL bảo vệ thì password có thể được sử dụng an toàn (dù xác thực chứng thực SSL có thể là lựa chọn tốt hơn nếu một người phụ thuộc vào việc sử dụng SSL).

Các mật khẩu cơ sở dữ liệu PostgreSQL là khác với các mật khẩu người sử dụng hệ điều hành. Mật khẩu cho từng người sử dụng cơ sở dữ liệu trong catalog hệ thống pg_authid. Các mật khẩu có thể được quản lý bằng các lệnh SQL CREATE USER và ALTER USER, như, CREATE USER foo WITH PASSWORD 'secret'. Nếu không mật khẩu nào được thiết lập cho một người sử dụng, thì mật khẩu được lưu trữ là null và sự xác thực mật khẩu sẽ luôn hỏng đối với người sử dụng đó.

19.3.3. Xác thực GSSAPI

GSSAPI là một giao thức tiêu chuẩn công nghiệp cho xác thực an toàn được xác định trong RFC 2743. PostgreSQL hỗ trợ GSSAPI với xác thực Kerberos theo RFC 1964. GSSAPI đưa ra xác thực tự động (đăng nhập duy nhất) cho các hệ thống hỗ trợ nó. Bản thân sự xác thực đó là an toàn, nhưng các dữ liệu được gửi qua kết nối cơ sở dữ liệu sẽ được gửi đi mà không có mã hóa trừ phi SSL được sử dụng.

Khi GSSAPI sử dụng Kerberos, nó dùng nguyên tắc cơ bản có định dạng servicename/hostname@realm. Để có thông tin về nguyên tắc đó, và cách thiết lập các khóa theo yêu cầu, xem Phần 19.3.5.

Sự hỗ trợ GSSAPI phải được kích hoạt khi PostgreSQL được xây dựng; xem Chương 15 để có thêm thông tin.

Các lựa chọn cấu hình sau đây sẽ được hỗ trợ cho GSSAPI:

include_realm

Nếu thiết lập về 1, thì tên lãnh địa từ nguyên tắc người sử dụng được xác thực được đưa vào tên của người sử dụng hệ thống mà được truyền qua việc ánh xạ tên người sử dụng (Phần 19.2). Điều này là hữu dụng cho việc điều khiển những người sử dụng từ nhiều lãnh địa.

map

Cho phép việc ánh xạ giữa các tên người sử dụng hệ thống và cơ sở dữ liệu. Xem Phần 19.2 để có các chi tiết. Đối với nguyên tắc Kerberos username/hostbased@EXAMPLE.COM, thì tên người sử dụng được sử dụng cho việc ánh xạ là username/hostbased nếu include_realm bị vô hiệu hóa, và username/hostbased@EXAMPLE.COM nếu include_realm được kích hoạt.

krb_realm

Thiết lập lãnh địa để khớp với các tên nguyên tắc người sử dụng ngược lại. Nếu tham số này được thiết lập, chỉ những người sử dụng của lãnh địa đó sẽ được chấp nhận. Nếu nó không được thiết lập, thì những người sử dụng của bất kỳ lãnh địa nào cũng có thể kết nối, tuân theo bất kỳ việc ánh xạ tên người sử dụng nào được thực hiện.

19.3.4. Xác thực SSPI

SSPI là một công nghệ của Windows để xác thực an toàn với đăng nhập duy nhất. PostgreSQL sẽ sử dụng SSPI trong chế độ negotiate, nó sẽ sử dụng Kerberos khi có thể và tự động quay lại về NTLM trong các trường hợp khác. Xác thực SSPI chỉ làm việc khi cả máy chủ và máy trạm đang chạy Windows.

Khi sử dụng xác thực Kerberos, SSPI làm việc cách y hệt GSSAIP làm; xem Phần 19.3.3 để có các chi tiết.

Các lựa chọn cấu hình sau đây được hỗ trợ cho SSPI:

include_realm

Nếu được thiết lập về 1, tên lãnh địa từ nguyên tắc người sử dụng được xác thực được đưa vào trong tên người sử dụng hệ thống được truyền qua việc ánh xạ tên người sử dụng (Phần 19.2). Điều này là hữu dụng cho việc điều khiển những người sử dụng từ nhiều lãnh địa.

Map

Cho phép cho việc ánh xạ giữa các tên người sử dụng hệ thống và cơ sở dữ liệu. Xem Phần 19.2 để có các chi tiết.

krb_realm

Thiết lập lãnh địa để khớp với các tên nguyên tắc người sử dụng ngược lại. Nếu tham số này được thiết lập, thì chỉ những người của lãnh địa đó sẽ được chấp nhận. Nếu nó không được thiết lập, thì những người sử dụng của bất kỳ lãnh địa nào cũng có thể kết nối, tuân theo bất kỳ việc ánh xạ tên người sử dụng nào được thực hiện.

19.3.5. Xác thực Kerberos

Lưu ý: Xác thực Kerberos bẩm sinh từng bị phản đối và chỉ nên được sử dụng cho tính tương thích ngược. Các cài đặt mới và được nâng cấp sẽ được khuyến khích sử dụng phương pháp xác thực theo tiêu chuẩn công nghiệp GSSAPI (xem Phần 19.3.3).

Kerberos là một hệ thống xác thực an toàn tiêu chuẩn công nghiệp phù hợp cho tính toán phân tán qua một mạng công cộng. Một mô tả hệ thống Kerberos là nằm ngoài phạm vi của tài liệu này; nói chung nó có thể là khá phức tạp (vâng mạnh mẽ). Các câu hỏi - đáp (FAQ[1]) về Kerberos hoặc trang[2] về Kerberos của MIT có thể là những điểm khởi đầu tốt để khai phá. Vài tài nguyên về những phân phối Kerberos tồn tại. Kerberos đưa ra xác thực an toàn nhưng không mã hóa các truy vấn hoặc các dữ liệu được truyền qua mạng; đối với điều đó, hãy sử dụng SSL.

PostgreSQL hỗ trợ Kerberos phiên bản 5. Sự hỗ trợ Kerberos sẽ phải được kích hoạt khi

1 http://www.cmf.nrl.navy.mil/CCS/people/kenh/kerberos-faq.html
2 http://web.mit.edu/kerberos/www/

PostgreSQL được xây dựng; xem Chương 15 để có thêm thông tin.

PostgreSQL vận hành giống như một dịch vụ Kerberos thông thường. Tên của nguyên tắc dịch vụ là servicename/hostname@realm.

servicename có thể được thiết lập ở phía máy chủ bằng việc sử dụng tham số cấu hình krb_srvname, và ở phía máy trạm bằng việc sử dụng tham số kết nối krbsrvname. (Xem Phần 31.1). Mặc định cài đặt có thể được thay đổi từ mặc định postgres ở thời điểm xây dựng bằng việc sử dụng ./configure --with-krb-srvnam=whatever. Trong hầu hết các môi trường, tham số này không bao giờ cần phải thay đổi. Tuy nhiên, là cần thiết khi hỗ trợ nhiều cài đặt PostgreSQL trên cùng một host. Một số cài đặt Kerberos cũng có thể đòi hỏi một tên dịch vụ khác, như Active Directory của Microsoft đòi hỏi tên dịch vụ phải được viết hoa (POSTGRES).

hostname là tên host đầy đủ điều kiện của máy chủ. Lãnh địa nguyên tắc dịch vụ là lãnh địa được ưu tiên của máy chủ.

Các nguyên tắc máy trạm phải có tên người sử dụng cơ sở dữ liệu PostgreSQL của chúng như là thành phần đầu tiên của chúng, ví dụ pgusername@realm. Như một sự lựa chọn, bạn có thể sử dụng việc ánh xạ tên một người sử dụng để ánh xạ từ thành phần đầu tiên của tên nguyên tắc cho tên người sử dụng cơ sở dữ liệu đó. Mặc định, lãnh địa của máy trạm không được PostgreSQL kiểm tra. Nếu bạn có xác thực liên lãnh địa được kích hoạt và cần phải kiểm tra lãnh địa đó, hãy sử dụng tham số krb_realm, hoặc kích hoạt include_realm và sử dụng việc ánh xạ tên người sử dụng để kiểm tra lãnh địa đó.

Hãy chắc chắn rằng tệp keytab máy chủ của bạn là đọc được (và ưu tiên chỉ đọc được) đối với tài khoản máy chủ PostgreSQL. (Xem Phần 17.1). Vị trí của tệp khóa đó được tham số cấu hình krb_server_keyfile chỉ định. Mặc định là /usr/local/pgsql/etc/krb5.keytab (hoặc bất kỳ thư mục nào từng được chỉ định như sysconfdir tại thời điểm xây dựng).

Tệp keytab được phần mềm Kerberos sinh ra; xem tài liệu Kerberos để có các chi tiết.

Ví dụ sau đây là cho các triển khai Kerberos 5 tương thích MIT:

```
kadmin% ank -randkey postgres/server.my.domain.org
kadmin% ktadd -k krb5.keytab postgres/server.my.domain.org
```

Khi kết nối với cơ sở dữ liệu hãy chắc chắn bạn có một vé cho nguyên tắc khớp với tên người sử dụng cơ sở dữ liệu được yêu cầu. Ví dụ, đối với tên người sử dụng cơ sở dữ liệu fred, nguyên tắc fred@EXAMPLE.COM có thể kết nối. Để cho phép nguyên tắc fred/users.example.com@EXAMPLE.COM, hãy sử dụng một bản đồ tên người sử dụng, như được mô tả trong Phần 19.2.

Nếu bạn sử dụng mod_auth_kerb[3] và mod_perl trên máy chủ web Apache, thì bạn có thể sử dụng dạng xác thực AuthType KerberosV5SaveCredentials với một script mod_perl. Điều này trao sự truy cập cơ sở dữ liệu an toàn qua web, không có các mật khẩu bổ sung được yêu cầu.

Các lựa chọn cấu hình sau đây được hỗ trợ cho Kerberos:

3 http://modauthkerb.sf.net

map

Cho phép việc ánh xạ giữa các tên người sử dụng hệ thống và cơ sở dữ liệu. Xem Phần 19.2 để có các chi tiết.

include_realm

Nếu được thiết lập về 1, tên lãnh địa từ nguyên tắc người sử dụng được xác thực sẽ được đưa vào trong tên người sử dụng hệ thống được truyền qua việc xác thực tên người sử dụng (Phần 19.2). Điều này là hữu dụng để điều khiển những người sử dụng từ nhiều lãnh địa.

krb_realm

Thiết lập lãnh địa để khớp với các tên người sử dụng ngược lại. Nếu tham số này được thiết lập, thì chỉ những người sử dụng của lãnh địa đó sẽ được chấp nhận. Nếu nó không được thiết lập, thì những người sử dụng của bất kỳ lãnh địa nào cũng có thể kết nối, tuân theo bất kỳ việc ánh xạ tên người sử dụng nào được thực hiện.

krb_server_hostname

Thiết lập một phần tên host của nguyên tắc dịch vụ. Điều này, được kết hợp với krb_srvname, được sử dụng để sinh nguyên tắc dịch vụ hoàn chỉnh, đó là krb_srvname/krb_server_hostname@REALM. Nếu không được thiết lập, mặc định là tên host máy chủ đó.

19.3.6. Xác thực nhận diện

Phương pháp xác thực nhận diện (ident) làm việc bằng việc có được tên người sử dụng hệ điều hành máy trạm và sử dụng nó như là tên người sử dụng cơ sở dữ liệu được phép (với một ánh xạ tên người sử dụng tùy chọn). Sự xác định tên người sử dụng máy trạm là điểm an toàn sống còn, và nó làm việc hoàn toàn khác, phụ thuộc vào dạng kết nối, như được mô tả bên dưới.

Các lựa chọn cấu hình sau đây sẽ được hỗ trợ cho nhận diện:

map

Cho phép việc ánh xạ giữa các tên người sử dụng hệ thống và cơ sở dữ liệu. Xem Phần 19.2 để có các chi tiết.

19.3.6.1. Xác thực nhận diện qua TCP/IP

“Giao thức nhận diện” (Identification Protocol) được mô tả trong RFC 1413. Hầu hết mỗi hệ điều hành giống Unix đều xuất xưởng với một máy chủ nhận diện mặc định nghe TCP trên cổng 113. Chức năng cơ bản của một máy chủ nhận diện là để trả lời các câu hỏi như “Người sử dụng nào đã khởi tạo kết nối đi ra khỏi cổng X của bạn và kết nối tới cổng Y của tôi?”. Vì PostgreSQL biết cả X và Y khi một kết nối vật lý được thiết lập, nó có thể tra hỏi máy chủ nhận diện trong host của máy trạm kết nối và có thể, về lý thuyết, xác định người sử dụng hệ điều hành cho bất kỳ kết nối nào.

Nhược điểm của thủ tục này là nó phụ thuộc vào tính toàn vẹn của máy trạm: nếu máy trạm không được tin cậy hoặc bị tổn thương, thì một kẻ tấn công có thể chạy đúng là bất kỳ chương trình nào trên cổng 113 và trả về bất kỳ tên người sử dụng nào mà anh ta chọn. Phương pháp xác thực này vì thế chỉ phù hợp cho các mạng đóng nơi mà từng máy trạm dưới sự kiểm soát chặt chẽ và nơi mà các

quản trị hệ thống và cơ sở dữ liệu vận hành theo liên hệ gần gũi. Nói cách khác, bạn phải tin tưởng chiếc máy đang chạy máy chủ nhận diện. Hãy chú ý cảnh báo:

> Giao thức nhận diện không có ý định như một giao thức kiểm soát truy cập hoặc ủy quyền.
>
> —RFC 1413

Một vài máy chủ nhận diện có một lựa chọn phi tiêu chuẩn làm cho tên người sử dụng được trả về sẽ được mã hóa, bằng việc sử dụng một khóa chỉ người quản trị máy gốc ban đầu biết được. Lựa chọn này *phải không* được sử dụng khi sử dụng máy chủ nhận diện với PostgreSQL, vì PostgreSQL không có bất kỳ cách gì để giải mã chuỗi được trả về để xác định tên thực sự của người sử dụng.

19.3.6.2. Xác thực nhận diện qua các khe cắm (socket) cục bộ

Trong các hệ thống hỗ trợ các yêu cầu SO_PEERCRED đối với các socket miền Unix (hiện hành như Linux, FreeBSD, NetBSD, OpenBSD, BSD/OS, và Solaris), xác thực nhận diện cũng có thể được áp dụng cho các kết nối cục bộ. PostgreSQL sử dụng SO_PEERCRED để tìm ra tên hệ điều hành của tiến trình máy trạm được kết nối. Trong trường hợp này, không rủi ro an toàn nào được thêm vào bằng việc sử dụng xác thực nhận diện; quả thực đây là một lựa chọn được ưu tiên cho các kết nối cục bộ trong các hệ thống như vậy.

Trong các hệ thống không có các yêu cầu SO_PEERCRED, xác thực nhận diện chỉ sẵn sàng cho các kết nối TCP/IP. Như một sự khắc phục, có khả năng chỉ định địa chỉ localhost 127.0.0.1 và đưa các kết nối về địa chỉ này. Phương pháp này là đáng tin cậy ở mức độ mà bạn tin cậy máy chủ nhận diện cục bộ.

19.3.7. Xác thực LDAP

Phương pháp xác thực này vận hành tương tự như password ngoại trừ là nó sử dụng LDAP như là phương pháp kiểm tra hợp lệ mật khẩu. LDAP chỉ được sử dụng để kiểm tra hợp lệ các cặp tên/mật khẩu của người sử dụng. Vì thế người sử dụng phải tồn tại rồi trong cơ sở dữ liệu trước khi LDAP có thể được sử dụng để xác thực.

Xác thực LDAP có thể vận hành ở 2 chế độ. Trong chế độ đầu, máy chủ sẽ ràng buộc tới tên phân biệt được xây dựng như là tiền tố tên người sử dụng hậu tố (prefix username suffix). Thông thường, tham số prefix được sử dụng để chỉ định cn=, hoặc DOMAIN\ trong một môi trường Active Directory. Còn suffix được sử dụng để chỉ định phần còn lại của DN (tên miền) trong một môi trường không là Active Directory.

Trong chế độ thứ 2, máy chủ trước hết ràng buộc tới thư mục LDAP bằng một tên người sử dụng và mật khẩu cố định, được chỉ định bằng ldapbinddn và ldapbindpasswd, và thực hiện sự tìm kiếm người sử dụng đang cố gắng đăng nhập vào cơ sở dữ liệu đó. Nếu không người sử dụng và mật khẩu nào được thiết lập cấu hình, thì một ràng buộc vô danh sẽ được thử cho thư mục đó. Sự tìm kiếm sẽ được thực hiện đối với cây con ở ldapbasedn, và sẽ cố thực hiện một sự khớp nối chính xác thuộc tính được chỉ định trong ldapsearchattribute. Nếu không thuộc tính nào được chỉ định, thì thuộc tính uid sẽ được sử dụng. Một khi người sử dụng đã được tìm thấy trong tìm kiếm này, thì máy chủ sẽ bỏ kết nối và ràng buộc lại tới thư mục đó như người sử dụng này, bằng việc sử dụng mật khẩu được máy trạm chỉ định, để kiểm tra hợp lệ xem đăng nhập đó có đúng hay không. Phương pháp này cho

phép sự mềm dẻo đáng kể hơn ở nơi mà các đối tượng người sử dụng sẽ nằm trong thư mục đó, nhưng sẽ làm cho 2 kết nối tới máy chủ LDAP sẽ được tạo ra.

Các lựa chọn cấu hình sau đây sẽ được hỗ trợ cho LDAP:

ldapserver

Các tên hoặc địa chỉ IP của các máy chủ LDAP để kết nối tới. Nhiều máy chủ có thể được chỉ định, phân cách nhau bằng các dấu trống.

ldapport

Số cổng trên máy chủ LDAP để kết nối tới. Nếu không cổng nào được chỉ định, thì thiết lập cổng mặc định của thư viện LDAP sẽ được sử dụng.

ldaptls

Thiết lập về 1 sẽ tạo kết nối giữa PostgreSQL và máy chủ LDAP sử dụng mã hóa TLS. Lưu ý rằng điều này chỉ mã hóa giao thông tới máy chủ LDAP - kết nối tới máy trạm sẽ vẫn không được mã hóa, trừ phi SSL được sử dụng.

ldapprefix

Chuỗi đứng trước tên người sử dụng khi hình thành DN để ràng buộc, khi tiến hành xác thực ràng buộc đơn giản.

ldapsuffix

Chuỗi nối vào tên người sử dụng khi hình thành DN để ràng buộc, khi thực hiện xác thực ràng buộc đơn giản.

ldapbasedn

DN gốc để bắt đầu tìm kiếm người sử dụng bên trong, khi thực hiện xác thực tìm kiếm + ràng buộc.

ldapbinddn

DN của người sử dụng sẽ ràng buộc tới thư mục đó để thực hiện tìm kiếm khi tiến hành xác thực tìm kiếm + ràng buộc.

ldapbindpasswd

Mật khẩu cho người sử dụng để ràng buộc tới thư mục đó để thực hiện tìm kiếm khi tiến hành xác thực tìm kiếm + ràng buộc.

ldapsearchattribute

Ghi nhận khớp chống lại tên người sử dụng trong tìm kiếm đó khi thực hiện xác thực tìm kiếm + ràng buộc.

Lưu ý: Vì LDAP thường sử dụng các dấu phẩy và ký tự trống để phân cách các phần khác nhau của một DN, thường là cần thiết để sử dụng các giá trị tham số trong các dấu ngoặc kép khi thiết lập cấu hình các lựa chọn LDAP, ví dụ:

```
ldapserver=ldap.example.net ldapprefix="cn=" ldapsuffix=", dc=example, dc=net"
```

19.3.8. Xác thực RADIUS

Phương pháp xác thực này vận hành tương tự như password ngoại trừ là nó sử dụng RADIUS như là

phương pháp kiểm tra hợp lệ mật khẩu. RADIUS chỉ được sử dụng để kiểm tra hợp lệ các cặp tên/mật khẩu người sử dụng. Vì thế người sử dụng phải tồn tại rồi trong cơ sở dữ liệu trước khi RADIUS có thể được sử dụng để xác thực.

Khi sử dụng xác thực RADIUS, một thông điệp yêu cầu truy cập (Access Request) sẽ được gửi tới máy chủ RADIUS được thiết lập cấu hình. Yêu cầu này sẽ có dạng chỉ xác thực (Authenticate Only), và bao gồm các tham số cho user name, password (được mã hóa) và NAS Identifier. Yêu cầu đó sẽ được mã hóa bằng việc sử dụng một bí mật được chia sẻ với máy chủ. Máy chủ RADIUS sẽ trả lời cho máy chủ này hoặc bằng việc chấp nhận truy cập Access Accept hoặc bằng việc từ chối truy cập Access Reject. Không có sự hỗ trợ cho việc kiểm toán RADIUS.

Các lựa chọn cấu hình sau đây được hỗ trợ cho RADIUS:

radiusserver

Tên hoặc địa chỉ IP của máy chủ RADIUS để kết nối tới. Tham số này được yêu cầu.

radiussecret

Bí mật được chia sẻ được sử dụng khi việc nói một cách bí mật với máy chủ RADIUS. Điều này phải chính xác có giá trị y hệt trong các máy chủ PostgreSQL và RADIUS. Được khuyến cáo rằng đây sẽ là một chuỗi ít nhất 16 ký tự. Tham số này được yêu cầu.

Lưu ý: Vector mã hóa chỉ được sử dụng là mật mã mạnh nếu PostgreSQL được xây dựng với sự hỗ trợ cho OpenSSL. Trong các trường hợp khác, sự biến đổi sang máy chủ RADIUS chỉ nên được xem xét một cách mù mờ, không có an toàn, và các biện pháp an toàn ngoài được áp dụng nếu cần.

radiusport

Số cổng trong máy chủ RADIUS để kết nối tới. Nếu không cổng nào được chỉ định, thì cổng mặc định 1812 sẽ được sử dụng.

radiusidentifier

Chuỗi đó được sử dụng như là NAS Identifier trong các yêu cầu RADIUS. Tham số này có thể được sử dụng như một tham số nhận diện thứ 2, ví dụ, người sử dụng cơ sở dữ liệu nào đang cố gắn xác thực, người sử dụng nào có thể được sử dụng cho việc khớp nối chính sách trên máy chủ RADIUS. Nếu không có mã định danh nào được chỉ định, thì mặc định postgresql sẽ được sử dụng.

19.3.9. Xác thực bằng chứng thực

Phương pháp xác thực này sử dụng các chứng thực SSL máy trạm để thực hiện xác thực. Vì thế nó chỉ sẵn sàng cho các kết nối SSL. Khi sử dụng phương pháp xác thực này, máy chủ sẽ yêu cầu rằng máy trạm cung cấp một chứng chỉ hợp lệ. Không có sự nhắc nhở mật khẩu nào sẽ được gửi tới máy trạm cả. Thuộc tính cn (tên chung - Common Name) của chứng thực sẽ được so sánh với tên người sử dụng cơ sở dữ liệu được yêu cầu, và nếu chúng khớp thì sự đăng nhập sẽ được phép. Việc ánh xạ tên người sử dụng có thể được sử dụng để cho phép cn sẽ khác với tên người sử dụng cơ sở dữ liệu.

Các lựa chọn cấu hình sau đây được hỗ trợ cho xác thực chứng thực SSL:

map

Cho phép việc ánh xạ giữa các tên người sử dụng hệ thống và cơ sở dữ liệu. Xem Phần 19.2 để có các chi tiết.

19.3.10. Xác thực PAM

Phương pháp xác thực này vận hành tương tự như password ngoại trừ là nó sử dụng các module xác thực cài cắm được - PAM (Pluggable Authentication Modules) như là cơ chế xác thực. Tên dịch vụ PAM mặc định là postgresql. PAM chỉ được sử dụng để kiểm tra hợp lệ các cặp tên/mật khẩu của người sử dụng. Vì thế người sử dụng phải tồn tại rồi trong cơ sở dữ liệu trước khi PAM có thể được sử dụng để xác thực. Để có thêm thông tin về PAM, hãy đọc trang[4] Linux - PAM và trang[5] Solaris PAM.

Các lựa chọn cấu hình sau đây được hỗ trợ cho PAM:

pamservice

Tên dịch vụ PAM.

Lưu ý: nếu PAM được thiết lập để đọc /etc/shadow, thì xác thực sẽ hỏng vì máy chủ PostgreSQL được một người sử dụng không phải gốc root khởi tạo. Tuy nhiên, điều này không phải là vấn đề khi PAM được thiết lập cấu hình để sử dụng LDAP hoặc các phương pháp xác thực khác.

19.4. Vấn đề xác thực

Các hỏng hóc xác thực và các vấn đề có liên quan thường tự chúng nêu lên thông qua các thông điệp lỗi như sau:

FATAL: no pg_hba.conf entry for host "123.123.123.123", user "andym", database "testdb"

Đây là những gì bạn có khả năng nhất sẽ có nếu bạn thành công trong việc liên lạc với máy chủ, nhưng nó không muốn nói với bạn. Như thông điệp đó gợi ý, máy chủ đã khước từ yêu cầu kết nối vì nó thấy không có khoản trùng khớp nào trong tệp cấu hình pg_hba.conf của nó.

FATAL: password authentication failed for user "andym"

Các thông điệp giống thế này chỉ ra rằng bạn đã liên lạc được với máy chủ, và nó có thiện chí nói với bạn, nhưng không được cho tới khi bạn truyền phương pháp xác thực được chỉ định trong tệp pg_hba.conf. Hãy kiểm tra mật khẩu mà bạn đang cung cấp, hoặc kiểm tra Kerberos của bạn hoặc phần mềm nhận diện nếu kêu ca đó nhắc tới một trong những dạng xác thực đó.

FATAL: user "andym" does not exist

Tên người sử dụng cơ sở dữ liệu được chỉ ra đã không được tìm thấy.

FATAL: database "testdb" does not exist

4 http://www.kernel.org/pub/linux/libs/pam/
5 http://www.sun.com/software/solaris/pam/

Cơ sở dữ liệu mà bạn đang cố kết nối tới không tồn tại. Lưu ý rằng nếu bạn không chỉ định một tên cơ sở dữ liệu, thì nó mặc định về tên người sử dụng cơ sở dữ liệu đó, điều có thể có hoặc có thể không là điều đúng đắn.

> **Mẹo**: Lưu ký máy chủ có thể có nhiều thông tin về hỏng hóc xác thực hơn được máy trạm nêu. Nếu bạn lúng túng về lý do hỏng hóc, hãy kiểm tra lưu ký của máy chủ.

Chương 20. Vai trò và quyền ưu tiên của cơ sở dữ liệu

PostgreSQL quản lý các quyền truy cập cơ sở dữ liệu bằng việc sử dụng khái niệm các *vai trò*. Một vai trò có thể được nghĩ như hoặc là một người sử dụng cơ sở dữ liệu, hoặc một nhóm những người sử dụng cơ sở dữ liệu, phụ thuộc vào cách mà vai trò đó được thiết lập. Các vai trò có thể sở hữu các đối tượng cơ sở dữ liệu (ví dụ, các bảng) và có thể chỉ định các quyền ưu tiên trong các đối tượng đó để trao *cơ chế thành viên* trong một vai trò này cho một vai trò khác, vì thế cho phép vai trò thành viên đó sử dụng các quyền ưu tiên được chỉ định cho vai trò khác.

Khái niệm về các vai trò gộp các khái niệm về "những người sử dụng" và "các nhóm". Trong các phiên bản PostgreSQL trước 8.1, những người sử dụng và các nhóm là các dạng thực thể khác nhau, nhưng bây giờ chỉ có các vai trò. Bất kỳ vai trò nào cũng có thể hành động như một người sử dụng, một nhóm, hoặc cả 2.

Chương này mô tả cách tạo ra và quản lý các vai trò và giới thiệu hệ thống quyền ưu tiên. Nhiều thông tin hơn về các dạng khác nhau các đối tượng cơ sở dữ liệu và các hiệu ứng về các quyền ưu tiên có thể thấy trong Chương 5.

20.1. Vai trò của cơ sở dữ liệu

Các vai trò của cơ sở dữ liệu, về khái niệm, là hoàn toàn khác biệt với những người sử dụng hệ điều hành. Trong thực tế có thể là thuận tiện để duy trì một sự tương ứng, nhưng điều này không được yêu cầu. Các vai trò của cơ sở dữ liệu là tổng thể xuyên khắp một cài đặt bó cơ sở dữ liệu (và không phải theo từng cơ sở dữ liệu riêng rẽ). Để tạo một vai trò, hãy sử dụng lệnh SQl CREATE ROLE:

```
CREATE ROLE name;
```

name đi sau các qui tắc cho mã định danh SQL: hoặc để tự nhiên mà không có các ký tự đặc biệt, hoặc nằm trong các dấu ngoặc kép. (Trong thực tế, bạn thường sẽ muốn thêm các lựa chọn bổ sung, như LOGIN, vào lệnh đó. Nhiều chi tiết hơn ở bên dưới). Để loại bỏ một vai trò đang tồn tại, hãy sử dụng lệnh tương tự DROP ROLE:

```
DROP ROLE name;
```

Để thuận tiện, người sử dụng tạo ra (createuser) và người sử dụng loại bỏ (dropuser) các chương trình được cung cấp như là người bao gói xung quanh các lệnh SQL mà có thể được gọi từ dòng lệnh của trình biên dịch shell:

```
createuser name
dropuser name
```

Để xác định tập hợp các vai trò đang tồn tại, hãy kiểm tra catalog hệ thống pg_roles, ví dụ SELECT rolname FROM pg_roles;

Siêu lệnh \du của chương trình psql cũng là hữu dụng cho việc liệt kê các vai trò đang tồn tại.

Để mồi khởi động hệ thống cơ sở dữ liệu, một hệ thống được khởi động tươi mới luôn có một vai trò được xác định trước. Vai trò này luôn là một "siêu người sử dụng", và là mặc định (trừ phi bị sửa khi chạy initdb) nó sẽ có tên y hệt như người sử dụng hệ điều hành mà đã khởi tạo bó cơ sở dữ liệu đó. Thông thường, vai trò này sẽ được đặt tên là postgres. Để tạo nhiều vai trò hơn thì bạn trước hết phải kết nối như vai trò ban đầu này.

Mỗi kết nối tới máy chủ cơ sở dữ liệu được làm bằng việc sử dụng tên của một vài vai trò đặc biệt, và vai trò này xác định các quyền ưu tiên truy cập ban đầu cho các lệnh được đưa ra trong kết nối đó. Tên vai trò để sử dụng cho một kết nối cơ sở dữ liệu đặc biệt được máy trạm chỉ ra mà đang khởi tạo yêu cầu kết nối đó theo một cách thức đặc thù ứng dụng. Ví dụ, chương trình psql sử dụng lựa chọn dòng lệnh -U để chỉ ra vai trò để kết nối. Nhiều ứng dụng giả thiết tên của người sử dụng hệ điều hành hiện hành bằng mặc định (bao gồm cả createuser và psql). Vì thế thường là thuận tiện để duy trì một sự tương ứng đặt tên giữa những người sử dụng hệ điều hành và các vai trò.

Tập hợp các vai trò cơ sở dữ liệu mà một kết nối máy trạm đưa ra có thể kết nối như được thiết lập xác thực máy trạm xác định, như được giải thích trong Chương 19. (Vì thế, một máy trạm không bị hạn chế để kết nối như vai trò khớp với người sử dụng hệ điều hành của nó, hệt như tên đăng nhập của một người không khớp với tên thực tế của chị ta). Vì sự nhận diện vai trò xác định tập hợp các quyền ưu tiên sẵn sàng cho một máy trạm được kết nối, là quan trọng để cẩn thận thiết lập cấu hình các quyền ưu tiên khi thiết lập môi trường nhiều người sử dụng.

20.2. Thuộc tính của vai trò

Vai trò của một cơ sở dữ liệu có thể có một số thuộc tính xác định quyền ưu tiên của nó và tương tác với hệ thống xác thực máy trạm.

login privilege (quyền ưu tiên đăng nhập)

> Chỉ các vai trò có thuộc tính LOGIN có thể được sử dụng như tên vai trò ban đầu cho một kết nối cơ sở dữ liệu. Một vai trò với thuộc tính LOGIN có thể được xem là y hệt như một "người sử dụng cơ sở dữ liệu". Để tạo một vai trò với quyền ưu tiên đăng nhập, hãy sử dụng hoặc:
>
> ```
> CREATE ROLE name LOGIN;
> CREATE USER name;
> ```
>
> (CREATE USER là tương đương với CREATE ROLE ngoại trừ là CREATE USER giả thiết LOGIN theo mặc định, còn CREATE ROLE thì không).

superuser status (tình trạng của siêu người sử dụng)

> Siêu người sử dụng của một cơ sở dữ liệu vượt qua được tất cả các kiểm tra cho phép. Đây là một quyền ưu tiên nguy hiểm và nên không được sử dụng một cách bất cẩn; tốt nhất là làm hầu hết các công việc của bạn như một vai trò không phải là một siêu người sử dụng. Để tạo một siêu người sử dụng của một cơ sở dữ liệu mới, hãy sử dụng CREATE ROLE name SUPERUSER. Bạn phải làm điều này như một vai trò là một siêu người sử dụng rồi.

database creation (tạo cơ sở dữ liệu)

> Một vai trò phải rõ ràng được trao quyền tạo các cơ sở dữ liệu (ngoại trừ đối với các siêu người sử dụng, vì chúng bỏ qua các kiểm tra quyền). Để tạo ra một vai trò như vậy, hãy sử

dụng CREATE ROLE name CREATEROLE.

role creation (tạo vai trò)

Một vai trò phải rõ ràng được trao quyền để tạo nhiều vai trò hơn (ngoại trừ đối với những siêu người sử dụng, vì họ vượt qua được tất cả các kiểm tra). Để tạo ra một vai trò như vậy, hãy sử dụng CREATE ROLE name CREATEROLE. Một vai trò với quyền ưu tiên CREATEROLE cũng có thể sửa đổi và loại bỏ các vai trò khác, cũng như trao hoặc thu hồi cơ chế thành viên trong chúng. Tuy nhiên, để tạo, sửa, bỏ, hoặc thay đổi cơ chế thành viên của vai trò của một siêu người sử dụng, thì tình trạng của siêu người sử dụng được yêu cầu; CREATEROLE là không đủ cho điều đó.

password (mật khẩu)

Một mật khẩu chỉ quan trọng nếu phương pháp xác thực máy trạm đòi hỏi người sử dụng đó cung cấp một mật khẩu khi kết nối tới cơ sở dữ liệu đó. Các phương pháp xác thực password và md5 sử dụng các mật khẩu. Các mật khẩu cơ sở dữ liệu là khác với các mật khẩu của hệ điều hành. Hãy chỉ định một mật khẩu dựa vào sự tạo vai trò với CREATE ROLE name PASSWORD 'string'.

Các thuộc tính của một vai trò có thể được sửa đổi sau khi tạo bằng ALTER ROLE. Xem các trang tham chiếu đối với các lệnh CREATE ROLE và ALTER ROLE để có các chi tiết.

Mẹo: Là thực tiễn tốt để tạo ra một vai trò có các quyền ưu tiên CREATEDB và CREATEROLE, nhưng không phải là một siêu người sử dụng, và sau đó sử dụng vai trò này cho tất cả sự quản lý thường nhật các cơ sở dữ liệu và vai trò. Tiếp cận này tránh được các mối nguy hiểm của việc vận hành như một siêu người sử dụng cho các tác vụ không thực sự đòi hỏi nó.

Một vai trò cũng có thể có các mặc định đặc thù vai trò cho nhiều trong số các thiết lập cấu hình thời gian chạy được mô tả trong Chương 18. Ví dụ, nếu vì một số lý do nào đó bạn muốn vô hiệu hóa các quét chỉ số (gợi ý: không phải là ý tưởng tốt) bất kỳ khi nào bạn kết nối, bạn có thể sử dụng: ALTER ROLE myname SET enable_indexscan TO off;

Điều này sẽ lưu thiết lập (nhưng không thiết lập nó ngay lập tức). Trong các kết nối tiếp sau của vai trò này thì nó sẽ xuất hiện như thể SET enable_indexscan TO off từng được thực thi ngay trước khi phiên làm việc đó bắt đầu. Bạn còn có thể sửa đổi thiết lập này trong khi diễn ra phiên làm việc đó; nó sẽ chỉ là mặc định. Để loại bỏ một thiết lập mặc định đặc thù vai trò, hãy sử dụng ALTER ROLE rolename RESET varname. Lưu ý là các mặc định đặc thù vai trò được gắn vào các vai trò mà không có quyền ưu tiên LOGIN thì khá là vô dụng, vì chúng sẽ không bao giờ bị thu hồi cả.

20.3. Quyền ưu tiên

Khi một đối tượng được tạo ra, nó được chỉ định như là chủ sở hữu. Chủ sở hữu thường là vai trò được thực thi lệnh tạo ra. Đối với hầu hết các dạng các đối tượng, tình trạng ban đầu là chỉ chủ sở hữu (hoặc một siêu người sử dụng) có thể làm bất kỳ điều gì với đối tượng đó. Để cho phép các vai trò khác sử dụng nó, các quyền ưu tiên phải được trao. Có vài dạng quyền ưu tiên khác nhau: SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER, CREATE, CONNECT, TEMPORARY,

EXECUTE, và USAGE. Để có thêm thông tin về các dạng quyền ưu tiên khác nhau được PostgreSQL hỗ trợ, hãy xem trang tham chiếu GRANT.

Để chỉ định các quyền ưu tiên, lệnh GRANT được sử dụng. Vì thế, nếu joe là một vai trò đang tồn tại, và accounts là một bảng đang tồn tại, thì quyền ưu tiên để cập nhật bảng đó có thể được trao bằng:

```
GRANT UPDATE ON accounts TO joe;
```

Tên đặc biệt PUBLIC có thể được sử dụng để trao một quyền ưu tiên cho mỗi vai trò trong hệ thống. Việc viết ALL ở nơi của một quyền ưu tiên đặc biệt chỉ định rằng tất cả các quyền ưu tiên mà áp dụng cho đối tượng đó sẽ được trao.

Để thu hồi một quyền ưu tiên, hãy sử dụng lệnh được đặt tên thích hợp là REVOKE:

```
REVOKE ALL ON accounts FROM PUBLIC;
```

Các quyền ưu tiên đặc biệt của một chủ sở hữu đối tượng (như, quyền để sửa đổi hoặc phá hủy đối tượng đó) luôn là ngầm hiểu là chủ sở hữu đó, và không thể được trao hoặc thu hồi. Nhưng chủ sở hữu có thể chọn để thu hồi các quyền ưu tiên thông thường của riêng anh ta, ví dụ để làm cho một bảng chỉ đọc đối với bản thân anh ta cũng như những người khác.

Một đối tượng có thể được chỉ định cho một chủ sở hữu mới bằng một lệnh ALTER đối với dạng phù hợp cho đối tượng đó. Các siêu người sử dụng luôn có thể làm điều này; các vai trò thường ngày chỉ có thể làm điều này nếu họ vừa là chủ sở hữu hiện hành của đối tượng (hoặc một thành viên sở hữu vai trò đó) và vừa là thành viên của việc sở hữu vai trò mới đó.

20.4. Cơ chế thành viên vai trò

Thường thuận tiện để nhóm những người sử dụng cùng với nhau để dễ quản lý các quyền ưu tiên: bằng cách đó, các quyền ưu tiên có thể được trao cho, hoặc được thu hồi từ, một nhóm như một tổng thể. Trong PostgreSQL thì điều này được việc tạo ra một vai trò thực hiện mà đại diện cho nhóm đó, và sau đó trao cơ chế thành viên trong vai trò của nhóm đó cho các vai trò của người sử dụng cá nhân riêng rẽ.

Để thiết lập vai trò của một nhóm, trước hết hãy tạo vai trò đó:

```
CREATE ROLE name;
```

Thường thì một vai trò đang được sử dụng như một nhóm có thể không có thuộc tính LOGIN, dù bạn có thể thiết lập nó nếu bạn muốn.

Một khi vai trò của nhóm đó tồn tại, thì bạn có thể thêm và loại bỏ các thành viên bằng việc sử dụng các lệnh GRANT và REVOKE:

```
GRANT group_role TO role1, ... ;
REVOKE group_role FROM role1, ... ;
```

Bạn cũng có thể trao cơ chế thành viên cho các vai trò nhóm khác, (vì thực sự không có bất kỳ sự khác biệt nào giữa các vai trò nhóm và các vai trò không nhóm). Cơ sở dữ liệu sẽ không cho phép

bạn thiết lập các vòng lặp cơ chế thành viên quay vòng. Hơn nữa, không được phép trao cơ chế thành viên trong một vai trò cho PUBLIC.

Các thành viên vai trò của một nhóm có thể sử dụng các quyền ưu tiên của vai trò đó theo 2 cách. Trước hết, từng thành viên của một nhóm có thể rõ ràng thực hiện SET ROLE để tạm thời "trở thành" vai trò nhóm. Ở tình trạng này, phiên làm việc của cơ sở dữ liệu có sự truy cập tới các quyền ưu tiên của vai trò nhóm thay vì vai trò đăng nhập gốc ban đầu, và bất kỳ đối tượng cơ sở dữ liệu nào được tạo ra cũng sẽ được xem là được vai trò nhóm đó chứ không phải vai trò đăng nhập sở hữu. Thứ 2, các vai trò thành viên có thuộc tính INHERIT tự động có sử dụng các quyền ưu tiên các vai trò của những gì họ là các thành viên, bao gồm bất kỳ quyền ưu tiên nào được các vai trò đó kế thừa. Như một ví dụ, hãy giả thiết chúng ta đã thực hiện:

```
CREATE ROLE joe LOGIN INHERIT;
CREATE ROLE admin NOINHERIT;
CREATE ROLE wheel NOINHERIT;
GRANT admin TO joe;
GRANT wheel TO admin;
```

Ngay sau khi kết nối như là vai trò joe, một phiên làm việc của cơ sở dữ liệu sẽ có sử dụng các quyền ưu tiên được trao trực tiếp cho joe cộng với bất kỳ các quyền ưu tiên nào được trao cho admin, vì joe "kế thừa" các quyền ưu tiên của admin; Tuy nhiên, các quyền ưu tiên được trao cho wheel là không sẵn sàng, vì thậm chí dù joe là một thành viên không trực tiếp của wheel, thì cơ chế thành viên là qua admin mà nó có thuộc tính NOINHERIT. Sau đó:

```
SET ROLE admin;
```

phiên làm việc đó có thể có sử dụng chỉ các quyền ưu tiên mà được trao cho admin, và không phải là các quyền ưu tiên được trao cho joe. Sau đó:

```
SET ROLE wheel;
```

phiên làm việc đó có thể sử dụng chỉ các quyền ưu tiên nào được trao cho wheel, và không sử dụng các quyền ưu tiên được trao cho hoặc joe hoặc admin. Tình trạng quyền ưu tiên ban đầu có thể được phục hồi với bất kỳ điều gì sau đây:

```
SET ROLE joe;
SET ROLE NONE;
RESET ROLE;
```

> **Lưu ý**: Lệnh SET ROLE luôn cho phép việc lựa chọn bất kỳ vai trò nào mà vai trò đăng nhập gốc ban đầu trực tiếp hoặc gián tiếp là một thành viên của nó. Vì thế, trong ví dụ ở trên, không nhất thiết để trở thành admin trước khi trở thành wheel.
>
> **Lưu ý**: Theo tiêu chuẩn SQL, có một sự khác biệt rõ ràng giữa những người sử dụng và các vai trò, và những người sử dụng không tự động kế thừa các quyền ưu tiên trong khi các vai trò thì có. Hành vi này có thể có được trong PostgreSQL bằng việc trao các vai trò đang được sử dụng như là thuộc tính INHERIT của các vai trò SQL, trong khi việc trao các vai trò đang được sử dụng như là thuộc tính NOINHERIT của những người sử dụng SQL. Tuy nhiên, PostgreSQL mặc định trao tất cả các vai trò thuộc tính INHERIT, vì sự tương thích ngược với các phiên bản trước 8.1 trong đó những người sử dụng đã luôn có sử dụng các quyền được

trao cho các nhóm mà họ từng là thành viên.

Các thuộc tính vai trò LOGIN, SUPERUSER, CREATEDB, và CREATEROLE có thể được xem như là các quyền ưu tiên đặc biệt, nhưng chúng không bao giờ được kế thừa như các quyền ưu tiên thông thường trong các đối tượng cơ sở dữ liệu là. Bạn thực sự phải SET ROLE cho một vai trò đặc biệt có một trong các thuộc tính đó để chắc chắn về thuộc tính đó. Việc tiếp tục ví dụ ở trên, chúng tôi có thể chọn để trao CREATEDB và CREATEROLE cho vai trò admin. Sau đó một phiên làm việc kết nối như vai trò joe có thể không có các quyền ưu tiên ngay, mà chỉ sau khi thực hiện SET ROLE admin.

Để phá bỏ vai trò của một nhóm, hãy sử dụng DROP ROLE:

```
DROP ROLE name;
```

Bất kỳ cơ chế thành viên nào trong vai trò nhóm cũng tự động được thu hồi (nhưng các vai trò thành viên thì không vì nếu không sẽ bị ảnh hưởng). Tuy nhiên hãy lưu ý rằng bất kỳ đối tượng nào được vai trò nhóm sở hữu cũng trước hết phải được thu hồi.

20.5. Chức năng và an toàn Trigger

Các chức năng và các trigger cho phép những người sử dụng chèn mã vào máy chủ phụ trợ (backend) mà những người sử dụng khác có thể thực thi một cách không cố ý. Vì thế, cả 2 cơ chế đều cho phép những người sử dụng trao "con ngựa thành Tơ roa" (Trojan horse) cho những người khác khá dễ dàng. Sự bảo vệ thực sự duy nhất là kiểm soát chặt chẽ đối với những ai có thể định nghĩa các hàm đó.

Các hàm chạy bên trong tiến trình máy chủ phụ trợ với các quyền hệ điều hành đối với daemon máy chủ cơ sở dữ liệu. Nếu ngôn ngữ lập trình được sử dụng cho hàm đó cho phép bộ nhớ không được kiểm tra truy cập, thì có khả năng thay đổi các cấu trúc dữ liệu nội bộ máy chủ đó. Vì thế, trong số nhiều điều khác, các hàm như vậy có thể phá vỡ bất kỳ các kiểm soát truy cập hệ thống nào. Các ngôn ngữ hàm mà cho phép sự truy cập như vậy được xem là "không tin cậy", và PostgreSQL chỉ cho phép các siêu người sử dụng tạo các hàm được viết trong các ngôn ngữ đó.

Chương 21. Quản lý cơ sở dữ liệu

Mỗi cài đặt máy chủ chạy PostgreSQL đều quản lý một hoặc nhiều hơn các cơ sở dữ liệu. Các cơ sở dữ liệu vì thế có mức kế thừa cao nhất đối với việc tổ chức các đối tượng SQL ("các đối tượng cơ sở dữ liệu"). Chương này mô tả các đặc tính của các cơ sở dữ liệu, và cách để tạo, quản lý, và loại bỏ chúng.

21.1. Tổng quan

Một cơ sở dữ liệu là một bộ sưu tập được đặt tên các đối tượng SQL ("các đối tượng cơ sở dữ liệu"). Thông thường, mỗi đối tượng cơ sở dữ liệu (các bảng, các hàm, ...) thuộc về một và chỉ một cơ sở dữ liệu. (Tuy nhiên có một ít catalog hệ thống, ví dụ, pg_database, thuộc về tổng thể một bó và có khả năng truy cập được từ từng cơ sở dữ liệu bên trong bó đó). Chính xác hơn, một cơ sở dữ liệu là một bộ sưu tập các sơ đồ và các sơ đồ đó có các bảng, các hàm, ... Vì thế tôn ti trật tự đầy đủ là: máy chủ, cơ sở dữ liệu, sơ đồ, bảng (hoặc một số dạng đối tượng khác, như một hàm).

Khi kết nối với máy chủ cơ sở dữ liệu, một máy trạm phải chỉ định trong yêu cầu kết nối của nó tên của cơ sở dữ liệu mà nó muốn kết nối tới. Không có khả năng để truy cập nhiều hơn một cơ sở dữ liệu cho mỗi kết nối. Tuy nhiên, một ứng dụng không bị hạn chế về số lượng các kết nối mà nó mở tới các cơ sở dữ liệu cùng y hệt hoặc khác. Các cơ sở dữ liệu là cách biệt nhau một cách vật lý và sự kiểm soát truy cập được quản lý ở mức kết nối. Nếu một cài đặt máy chủ PostgreSQL sẽ quản lý các dự án hoặc những người sử dụng mà sẽ cách biệt nhau và hầu hết các phần là không nhận biết ra nhau, vì thế được khuyến cáo đặt chúng vào các cơ sở dữ liệu tách biệt nhau. Nếu các dự án hoặc những người sử dụng có liên quan tới nhau và sẽ có khả năng sử dụng các tài nguyên của nhau, thì chúng nên được đặt trong cùng một cơ sở dữ liệu nhưng có khả năng trong các sơ đồ tách biệt nhau. Các sơ đồ thuần túy là một cấu trúc logic và ai có thể truy cập những gì được hệ thống quyền ưu tiên quản lý. Nhiều thông tin hơn về việc quản lý các sơ đồ có trong Phần 5.7.

Các cơ sở dữ liệu được tạo ra bằng lệnh CREATE DATABASE (xem Phần 21.2) và bị loại bỏ bằng lệnh DROP DATABASE (xem Phần 21.5). Để xác định tập hợp các cơ sở dữ liệu đang tồn tại, hãy kiểm tra catalog hệ thống pg_database, ví dụ

```
SELECT datname FROM pg_database;
```

Siêu lệnh \l và lựa chọn dòng lệnh -l của chương trình psql cũng là hữu dụng cho việc liệt kê các cơ sở dữ liệu đang tồn tại.

> **Lưu ý**: Tiêu chuẩn SQL gọi các cơ sở dữ liệu là "các catalog", nhưng không có sự khác biệt nào trong thực tế.

21.2. Tạo cơ sở dữ liệu

Để tạo một cơ sở dữ liệu, máy chủ PostgreSQL phải được bật và đang chạy (xem Phần 17.3).

Các cơ sở dữ liệu được tạo ra bằng lệnh SQL CREATE DATABASE:

```
CREATE DATABASE name;
```

trong đó name tuân theo các qui tắc thông thường đối với các mã định danh SQL. Vai trò hiện hành tự động trở thành chủ sở hữu của cơ sở dữ liệu mới. Đây là quyền ưu tiên của chủ sở hữu một cơ sở

dữ liệu để loại bỏ nó sau này (mà cũng loại bỏ tất cả các đối tượng trong nó, thậm chí nếu chúng có một người chủ sở hữu khác).

Sự tạo ra các cơ sở dữ liệu là một hoạt động có giới hạn. Xem Phần 20.2 về cách để trao quyền.

Vì bạn cần được kết nối tới máy chủ cơ sở dữ liệu để thực thi lệnh CREATE DATABASE, nên câu hỏi vẫn là cách mà cơ sở dữ liệu đầu tiên ở bất kỳ nơi nào được đưa ra có thể được tạo ra như thế nào. Cơ sở dữ liệu đầu tiên luôn được lệnh initdb tạo ra khi vùng lưu trữ dữ liệu được khởi tạo. (Xem Phần 17.2). Cơ sở dữ liệu này được gọi là postgres. Vì thế để tạo cơ sở dữ liệu "thông thường" đầu tiên thì bạn có thể kết nối tới postgres.

Một cơ sở dữ liệu thứ 2, template1, cũng được tạo ra trong quá trình khởi tạo bó cơ sở dữ liệu. Bất kỳ khi nào một cơ sở dữ liệu mới được tạo ra trong bó đó, thì template1 sẽ được mô phỏng một cách cơ bản. Điều này có nghĩa là bất kỳ thay đổi nào mà bạn thực hiện trong template1 sẽ được nhân giống tới từng cơ sở dữ liệu mới được tạo ra. Nhiều chi tiết hơn có trong Phần 21.3.

Vì sự thuận tiện, có một chương trình bạn có thể thực thi từ trình biên dịch shell để tạo các cơ sở dữ liệu mới, createdb.

```
createdb dbname
```

createdb dbname không làm gì huyền bí cả. Nó kết nối tới cơ sở dữ liệu postgres và đưa ra lệnh CREATE DATABASE, chính xác như được mô tả ở trên. Trang tham chiếu của createdb có các chi tiết thu hồi. Lưu ý rằng createdb không có bất kỳ đối số nào sẽ tạo ra một cơ sở dữ liệu với tên người sử dụng hiện hành.

> **Lưu ý**: Chương 19 có thông tin về cách hạn chế ai có thể kết nối tới một cơ sở dữ liệu được đưa ra.

Đôi khi bạn muốn tạo một cơ sở dữ liệu cho ai đó khác, và nhờ anh ta trở thành chủ sở hữu của cơ sở dữ liệu mới đó, nên anh ta có thể tự thiết lập cấu hình và quản lý nó. Để có được điều đó, hãy sử dụng một trong các lệnh sau:

```
CREATE DATABASE dbname OWNER rolename;
```

từ môi trường SQL, hoặc:

```
createdb -O rolename dbname
```

từ trình biên dịch shell. Chỉ siêu người sử dụng là được phép tạo một cơ sở dữ liệu cho ai đó khác (đó là, cho một vai trò mà bạn không phải là một thành viên của nó).

21.3. Cơ sở dữ liệu mẫu template

CREATE DATABASE thực sự làm việc bằng cách sao chép một cơ sở dữ liệu đang tồn tại. Mặc định, nó sao chép cơ sở dữ liệu hệ thống tiêu chuẩn có tên là template1. Vì thế cơ sở dữ liệu đó là "mẫu template" từ đó các cơ sở dữ liệu mới sẽ được tạo ra. Nếu bạn thêm các đối tượng vào template1, thì các đối tượng đó sẽ được sao chép vào các cơ sở dữ liệu do người sử dụng tạo ra sau đó. Hành vi này cho phép những sửa đổi cục bộ trên thực địa đối với tập hợp tiêu chuẩn các đối tượng trong các cơ sở dữ liệu. Ví dụ, nếu bạn cài đặt ngôn ngữ thủ tục PL/Perl vào template1, thì nó sẽ tự động sẵn sàng trong các cơ sở dữ liệu của người sử dụng mà không có bất kỳ hành động thêm ra nào khác được tiến hành khi các cơ sở dữ liệu đó được tạo ra.

Có một cơ sở dữ liệu hệ thống tiêu chuẩn thứ 2 có tên là template0. Cơ sở dữ liệu này chứa các dữ liệu y hệt như các nội dung ban đầu của template1, đó là, chỉ các đối tượng tiêu chuẩn được phiên bản PostgreSQL của bạn xác định trước. template0 sẽ không bao giờ bị thay đổi sau khi bó cơ sở dữ liệu đã được khởi tạo. Bằng việc lệnh cho CREATE DATABASE để sao chép template0 thay cho template1, bạn có thể tạo một cơ sở dữ liệu người sử dụng "trinh nguyên" không chứa các bổ sung thêm cục bộ trên thực địa nào trong template1. Điều này là đặc biệt hữu dụng khi khôi phục lại sự hỏng hóc của pg_dump: script hỏng sẽ được phục hồi trong một cơ sở dữ liệu trinh nguyên để đảm bảo rằng người ta tái tạo lại các nội dung đúng của cơ sở dữ liệu bị hỏng, không có xung đột với các đối tượng mà có thể từng được bổ sung thêm vào template1 sau đó.

Lý do phổ biến khác cho việc sao chép template0 thay vì template1 là việc mã hóa mới và các thiết lập bản địa có thể được chỉ định kho sao chép template0, trong khi một bản sao template1 phải sử dụng các thiết lập y hệt mà nó có. Điều này là vì template1 có thể chứa các dữ liệu mã hóa đặc biệt hoặc đặc thù bản địa, trong khi template0 thì không.

Để tạo một cơ sở dữ liệu bằng việc sao chép template0, hãy sử dụng:

```
CREATE DATABASE dbname TEMPLATE template0;
```

từ môi trường SQL, hoặc:

```
createdb -T template0 dbname
```

từ trình biên dịch shell.

Có khả năng tạo các cơ sở dữ liệu mẫu template bổ sung, và quả thực người ta có thể sao chép bất kỳ cơ sở dữ liệu nào trong một bó bằng việc chỉ định tên của nó như là mẫu template cho CREATE DATABASE. Tuy nhiên, điều quan trọng phải hiểu là điều này còn không (chưa) có ý định như một tiện ích "COPY DATABASE " mục đích chung. Hạn chế cơ bản là không phiên làm việc nào khác có thể được kết nối tới cơ sở dữ liệu nguồn trong khi nó đang được sao chép. CREATE DATABASE sẽ hỏng nếu bất kỳ kết nối nào khác tồn tại khi nó khởi động; trong quá trình thực hiện sao chép, các kết nối mới với cơ sở dữ liệu nguồn sẽ bị cản trở.

Tồn tại 2 cờ hữu dụng trong pg_database cho từng cơ sở dữ liệu: các cột datistemplate và datallowconn. datistemplate có thể được thiết lập để chỉ rằng một cơ sở dữ liệu có ý định như một mẫu template cho CREATE DATABASE. Nếu cờ này được thiết lập, thì cơ sở dữ liệu có thể được bất kỳ người sử dụng nào với các quyền ưu tiên CREATEDB mô phỏng; Nếu nó không được thiết lập, thì chỉ những siêu người sử dụng và chủ sở hữu của cơ sở dữ liệu đó có thể sao chép nó. Nếu datallowconn là sai (false), thì không kết nối mới nào tới cơ sở dữ liệu đó sẽ được phép (nhưng các phiên làm việc đang tồn tại sẽ không bị chấm dứt đơn giản bằng việc thiết lập cờ sai đó). Cơ sở dữ liệu template0 thường được đánh dấu là datallowconn = false để ngăn chặn sự sửa đổi nó. Cả template0 và template1 đều sẽ luôn được đánh dấu với datistemplate = true.

> **Lưu ý**: template1 và template0 không có bất kỳ tình trạng đặc biệt nào ngoài thực tế là tên template1 là tên cơ sở dữ liệu nguồn mặc định cho CREATE DATABASE. Ví dụ, người ta có thể bỏ template1 và tái tạo lại nó từ template0 mà không có bất kỳ hiệu ứng tồi tệ nào. Quá trình hành động này có thể được khuyến cáo nếu người ta đã thêm một cách bất cẩn một đống rác trong template1. (Để xóa template1, phải có pg_database.datistemplate = false).

Cơ sở dữ liệu postgres cũng được tạo ra khi một bó cơ sở dữ liệu được khởi tạo. Cơ sở dữ liệu này có nghĩa như là một cơ sở dữ liệu mặc định cho những người sử dụng và các ứng dụng để kết nối tới. Là đơn giản để sao chép template1 và có thể bị loại bỏ và được tái tạo lại nếu cần.

21.4. Thiết lập cấu hình cơ sở dữ liệu

Nhớ lại từ Chương 18 rằng máy chủ PostgreSQL đưa ra một số lượng lớn các biến cấu hình thời gian chạy. Bạn có thể thiết lập các giá trị mặc định đặc thù cơ sở dữ liệu cho nhiều trong các thiết lập đó.

Ví dụ, nếu vì vài lý do bạn muốn vô hiệu hóa trình tối ưu hóa GEQO đối với một cơ sở dữ liệu được đưa ra nào đó, bạn thường phải hoặc là vô hiệu hóa nó đối với tất cả các cơ sở dữ liệu hoặc phải chắc chắn rằng mọi việc kết nối máy trạm là cẩn trọng để đưa ra SET geqo TO off. Để làm cho thiết lập này thành mặc định trong một cơ sở dữ liệu đặc biệt nào đó, bạn có thể chạy lệnh:

```
ALTER DATABASE mydb SET geqo TO off;
```

Điều này sẽ cứu được thiết lập đó (nhưng không thiết lập nó ngay lập tức). Trong các kết nối tiếp sau tới cơ sở dữ liệu này, nó sẽ hiện diện như thể SET geqo TO off; đã được thực thi ngay sau khi phiên làm việc được khởi tạo. Lưu ý rằng những người sử dụng vẫn có thể sửa thiết lập này trong quá trình các phiên làm việc của họ; nó sẽ chỉ là mặc định. Để hoãn bất kỳ thiết lập nào như vậy, hãy sử dụng ALTER DATABASE dbname RESET varname.

21.5. Hủy một cơ sở dữ liệu

Các cơ sở dữ liệu sẽ bị hủy bằng lệnh DROP DATABASE:

```
DROP DATABASE name;
```

Chỉ chủ sở hữu của cơ sở dữ liệu, hoặc một siêu người sử dụng, có thể bỏ một cơ sở dữ liệu. Việc bỏ một cơ sở dữ liệu sẽ hủy tất cả các đối tượng mà từng có bên trong cơ sở dữ liệu đó. Sự hủy một cơ sở dữ liệu không thể hoãn lại được.

Bạn không thể thực thi lệnh DROP DATABASE trong khi được kết nối tới cơ sở dữ liệu nạn nhân. Tuy nhiên, bạn có thể được kết nối tới bất kỳ cơ sở dữ liệu nào khác, bao gồm cả cơ sở dữ liệu template1. template1 chỉ có thể là lựa chọn cho việc bỏ cơ sở dữ liệu người sử dụng cuối cùng của một bó được đưa ra.

Vì sự thuận tiện, cũng có một chương trình shell để bỏ các cơ sở dữ liệu, dropdb:

```
dropdb dbname
```

(Không giống như createdb, không là hành động mặc định để bỏ cơ sở dữ liệu với tên người sử dụng hiện hành).

21.6. Không gian bảng (tablespace)

Các không gian bảng trong PostgreSQL cho phép những người quản trị cơ sở dữ liệu xác định các vị trí trong hệ thống tệp, nơi mà các tệp đại diện cho các đối tượng cơ sở dữ liệu được lưu trữ. Một khi được tạo ra thì không gian bảng có thể được tham chiếu tới bằng tên khi tạo các đối tượng cơ sở dữ liệu.

Bằng việc sử dụng các không gian bảng, một người quản trị có thể kiểm soát hình thức đĩa của một cài đặt PostgreSQL. Điều này là hữu dụng ít nhất theo 2 cách. Trước hết, nếu phân vùng hoặc dung lượng trong đó bó cơ sở dữ liệu từng được khởi tạo dùng hết không gian và không thể được thực thi, thì một không gian bảng có thể được tạo ra trong một phân vùng khác và được sử dụng cho tới khi hệ thống có thể được thiết lập lại cấu hình.

Thứ 2, không gian bảng cho phép một người quản trị viên sử dụng tri thức mẫu sử dụng của các đối tượng cơ sở dữ liệu để tối ưu hóa hiệu năng. Ví dụ, một chỉ số mà được sử dụng rất nhiều có thể được đặt trong một đĩa rất nhanh, có độ sẵn sàng cao, như một thiết bị tình trạng cứng - SSD (Solid State Device) đắt giá. Cùng lúc, một bảng có chứa các dữ liệu được lưu trữ mà hiếm khi được sử dụng hoặc không phải là thực thi sống còn có thể được lưu trữ trong một hệ thống đĩa chậm hơn, ít đắt giá hơn.

Để xác định một không gian đĩa, hãy sử dụng lệnh CREATE TABLESPACE, ví dụ:

```
CREATE TABLESPACE fastspace LOCATION '/mnt/sda1/postgresql/data';
```

Vị trí đó phải là một thư mục rỗng, đang tồn tại mà được người sử dụng hệ điều hành của PostgreSQL sở hữu. Tất cả các đối tượng được tạo ra sau đó trong không gian bảng đó sẽ được lưu trữ trong các tệp nằm liền kề thư mục này.

> **Lưu ý**: Thường không có nhiều điểm trong việc tạo ra nhiều hơn 1 không gian đĩa cho từng hệ thống tệp logic, vì bạn không thể kiểm soát được vị trí của các tệp riêng rẽ bên trong một hệ thống tệp logic. Tuy nhiên, PostgreSQL không ép bất kỳ giới hạn nào như vậy, và quả thực nó không trực tiếp nhận biết được các biên giới hệ thống tệp trên hệ thống của bạn. Nó chỉ lưu trữ các tệp trong các thư mục mà bạn nói cho nó để sử dụng.

Bản thân sự tạo không gian bảng phải được thực hiện như siêu người sử dụng một cơ sở dữ liệu, nhưng sau đó bạn có thể cho phép những người sử dụng thông thường của cơ sở dữ liệu sử dụng nó. Để làm thế, hãy trao cho họ quyền ưu tiên CREATE trong đó.

Các bảng, chỉ số, và toàn bộ các cơ sở dữ liệu có thể được chỉ định cho không gian bảng đặc biệt. Để làm thế, một người sử dụng với quyền ưu tiên CREATE trong một không gian bảng được đưa ra phải truyền tên không gian bảng như một tham số tới lệnh phù hợp. Ví dụ, thứ sau đây tạo một bảng trong không gian bảng space1:

```
CREATE TABLE foo(i int) TABLESPACE space1;
```

Như một sự lựa chọn, hãy sử dụng tham số không gian bảng mặc định:

```
SET default_tablespace = space1;
CREATE TABLE foo(i int);
```

Khi default_tablespace được thiết lập về bất kỳ thứ gì khác 1 chuỗi rỗng, nó cung cấp một mệnh đề ngầm TABLESPACE cho các lệnh CREATE TABLE và CREATE INDEX mà không có một mệnh đề rõ ràng.

Cũng có một tham số temp_tablespaces, nó xác định chỗ tạm thời của các bảng và chỉ số, cũng

như các tệp tạm thời sẽ được sử dụng cho các mục đích như việc sắp xếp các tập hợp dữ liệu lớn. Điều này có thể là một danh sách các tên không gian bảng, thay vì chỉ một tên, sao cho tải có liên quan tới các đối tượng tạm thời có thể tràn sang nhiều không gian bảng. Một số ngẫu nhiên của danh sách đó được lấy mỗi lần một đối tượng tạm thời được tạo ra.

Không gian bảng có liên quan tới một cơ sở dữ liệu được sử dụng để lưu trữ các catalog hệ thống của cơ sở dữ liệu đó. Hơn nữa, là mặc định cho không gian bảng được sử dụng cho các bảng, các chỉ số, và các tệp tạm thời được tạo ra trong cơ sở dữ liệu, nếu không có mệnh đề TABLESPACE nào được đưa ra và không có sự lựa chọn nào khác được default_tablespace hoặc temp_tablespaces (một cách đúng phù hợp) chỉ định. Nếu một cơ sở dữ liệu được tạo ra không có việc chỉ định một không gian bảng cho nó, thì nó sử dụng không gian bảng y hệt như là cơ sở dữ liệu mẫu template mà nó được sao chép từ đó.

2 không gian bảng được tự động tạo ra khi bó cơ sở dữ liệu được khởi tạo. Không gian bảng pg_global được sử dụng cho các catalog hệ thống được chia sẻ. Không gian bảng pg_default là không gian bảng mặc định của các cơ sở dữ liệu template1 và template0 (và vì thế, cũng sẽ là không gian bảng mặc định cho các cơ sở dữ liệu khác, trừ phi được một mệnh đề TABLESPACE trong CREATE DATABASE ghi đè lên).

Một khi được tạo ra, một không gian bảng có thể được sử dụng từ bất kỳ cơ sở dữ liệu nào, miễn là người sử dụng đang yêu cầu có được quyền ưu tiên đủ. Điều này có nghĩa là một không gian bảng không thể bị loại bỏ cho tới khi tất cả các đối tượng trong tất cả các cơ sở dữ liệu đang sử dụng không gian bảng đó đã bị loại bỏ.

Để loại bỏ một không gian bảng rỗng, hãy sử dụng lệnh DROP TABLESPACE.

Để xác định một tập hợp các không gian bảng đang tồn tại, hãy kiểm tra catalog hệ thống pg_tablespace, ví dụ

```
SELECT spcname FROM pg_tablespace;
```

Siêu lệnh \db của chương trình psql cũng là hữu dụng để liệt kê các không gian bảng đang tồn tại.

PostgreSQL sử dụng các liên kết biểu tượng để đơn giản hóa sự triển khai các không gian bảng. Điều này có nghĩa là các không gian bảng chỉ được sử dụng trong các hệ thống mà hỗ trợ các liên kết biểu tượng.

Thư mục $PGDATA/pg_tblspc chứa các liên kết biểu tượng chỉ tới từng trong số các không gian bảng không được xây dựng sẵn được xác định trong bó đó. Dù không được khuyến cáo, có khả năng để tinh chỉnh hình thức không gian bảng bằng việc xác định lại các liên kết đó. 2 cảnh báo: không làm thế trong khi máy chủ đang chạy; và sau khi bạn khởi động lại máy chủ, hãy cập nhật catalog pg_tablespace với các vị trí mới. (nếu bạn không làm, pg_dump sẽ tiếp tục đưa ra các vị trí không gian bảng cũ).

Chương 22. Bản địa hóa

Chương này mô tả các đặc điểm bản địa hóa sẵn sàng từ quan điểm của người quản trị.

PostgreSQL hỗ trợ 2 tiện ích bản địa hóa:

- Sử dụng các tính năng bản địa của hệ điều hành để đưa ra trật tự so sánh, việc định dạng số, các thông điệp được dịch, và các khía cạnh khác có đặc thù bản địa.
- Cung cấp một số tập hợp ký tự khác để hỗ trợ cho việc lưu trữ văn bản trong tất cả các dạng ngôn ngữ, và đưa ra bản dịch tập hợp các ký tự giữa máy trạm và máy chủ.

22.1. Hỗ trợ bản địa

Hỗ trợ bản địa tham chiếu tới một ứng dụng lưu ý tới các ưu tiên về văn hóa xếp theo vần abc, việc sắp xếp, việc định dạng các số, ... PostgreSQL sử dụng các tiện ích bản địa C và POSIX tiêu chuẩn ISO được hệ điều hành máy chủ cung cấp. Để có thêm thông tin, hãy tham chiếu tới tài liệu hệ thống của bạn.

22.1.1. Tổng quan

Hỗ trợ bản địa được tự động khởi tạo khi một bó cơ sở dữ liệu được tạo ra bằng việc sử dụng initdb. initdb sẽ khởi tạo bó cơ sở dữ liệu đó với thiết lập bản địa đối với môi trường thực thi của nó một cách mặc định, nêu nếu hệ thống của bạn được thiết lập rồi để sử dụng bản địa mà bạn muốn trong bó cơ sở dữ liệu của bạn thì bạn sẽ không cần phải làm gì thêm. Nếu bạn muốn sử dụng một bản địa khác (hoặc bạn không chắc bản địa nào hệ thống của bạn được thiết lập theo), thì bạn có thể ra lệnh cho initdb chính xác bản địa nào phải sử dụng bằng việc chỉ định lựa chọn –locale. Ví dụ:

```
initdb --locale=sv_SE
```

Ví dụ này cho các hệ thống Unix thiết lập bản địa cho Thụy Điển (sv) như được nói ở Thụy Điển (SE). Các khả năng khác có thể là en_US (Tiếng Anh Mỹ) và fr_CA (Tiếng Pháp Canada). Nếu nhiều hơn một tập hợp các ký tự có thể được sử dụng cho một bản địa thì các đặc tả đó có thể có dạng language_territory.codeset. Ví dụ, fr_BE.UTF-8 đại diện cho tiếng Pháp (fr) như được nói ở Bỉ (BE), với một bộ mã ký tự UTF-8.

Các bản địa nào là sẵn sàng trong hệ thống của bạn dưới các tên nào sẽ phụ thuộc vào những gì đã được nhà bán hàng hệ điều hành cung cấp và những gì đã được cài đặt. Trong hầu hết các hệ thống Unix, lệnh locale -a sẽ đưa ra một danh sách các bản địa sẵn có. Windows sử dụng các tên bản địa dài dòng hơn, như German_Germany hoặc Swedish_Sweden.1252, nhưng các nguyên tắc là y hệt.

Một cách ngẫu nhiên là hữu dụng để trộn các qui tắc từ vài bản địa, như, sử dụng các qui tắc so sánh tiếng Anh nhưng các thông điệp tiếng Tây Ban Nha. Để hỗ trợ điều đó, một tập hợp các chủng loại con bản địa tồn tại mà kiểm soát chỉ các khía cạnh nhất định các qui tắc bản địa hóa:

LC_COLLATE	Trật tự sắp xếp chuỗi
LC_CTYPE	Phân loại ký tự (một chữ là gì? Nó có chữ hoa tương ứng hay không?)
LC_MESSAGES	Ngôn ngữ các thông điệp
LC_MONETARY	Định dạng số lượng tiền

LC_NUMERIC	Định dạng các số
LC_TIME	Định dạng ngày tháng và thời gian

Các tên chủng loại dịch sang các tên của các lựa chọn initdb sẽ ghi đè lựa chọn bản địa đối với một chủng loại đặc biệt. Ví dụ, để thiết lập bản địa về Tiếng Pháp Canada, nhưng sử dụng các qui tắc của Mỹ cho việc định dạng tiền tệ, hãy sử dụng initdb --locale=fr_CA –lc-monetary=en_US.

Nếu bạn muốn hệ thống hành xử như thể nó đã không có sự hỗ trợ bản địa, hãy sử dụng bản địa đặc biệt C hoặc POSIX.

Một vài chủng loại bản địa phải có các giá trị của chúng được cố định khi cơ sở dữ liệu được tạo ra. Bạn có thể sử dụng các thiết lập khác nhau cho các cơ sở dữ liệu khác nhau, nhưng một khi một cơ sở dữ liệu được tạo ra, bạn không thể thay đổi chúng cho cơ sở dữ liệu đó hơn nữa. LC_COLLATE và LC_CTYPE là dạng các chủng loại đó. Chúng ảnh hưởng tới trật tự sắp xếp các chỉ số, nên chúng phải được giữ cố định, nếu không các chỉ số trong các cột văn bản có thể sẽ bị hỏng. Các giá trị mặc định cho các chủng loại đó sẽ được xác định khi initdb được chạy, và các giá trị đó sẽ được sử dụng khi các cơ sở dữ liệu mới sẽ được tạo ra, trừ phi được chỉ định khác trong lệnh CREATE DATABASE.

Các chủng loại bản địa khác có thể bị thay đổi bất kỳ khi nào bạn muốn bằng việc thiết lập các tham số cấu hình máy chủ mà có tên y hệt như các chủng loại bản địa (xem Phần 18.10.2 để có các chi tiết). Các giá trị được initdb chọn chỉ thực sự được ghi vào tệp cấu hình postgresql.conf để phục vụ như là các mặc định khi máy chủ được khởi tạo. Nếu bạn vô hiệu hóa các chỉ định đó từ postgresql.conf thì máy chủ sẽ kế thừa các thiết lập từ môi trường thực thi của nó.

Lưu ý là hành vi bản địa của máy chủ được các biến môi trường xác định được máy chủ, chứ không phải môi trường của bất kỳ máy trạm nào nhìn thấy. Vì thế, hãy thận trọng để thiết lập cấu hình các thiết lập bản địa đúng trước khi khởi động máy chủ. Hệ quả của điều này là nếu máy trạm và máy chủ được thiết lập các bản địa khác nhau, thì các thông điệp có thể xuất hiện theo các ngôn ngữ khác nhau, phụ thuộc vào việc chúng có xuất xứ ở đâu.

> **Lưu ý**: Khi chúng ta nói về việc kế thừa bản địa từ môi trường thực thi, thì điều này có nghĩa cái sau có trong hầu hết các hệ điều hành: Đối với một chủng loại bản địa được đưa ra, như so sánh, thì các biến môi trường sau đây được tư vấn theo trật tự này cho tới khi một thứ được thấy được thiết lập: LC_ALL, LC_COLLATE (hoặc biến tương ứng với chủng loại tương ứng đó), LANG. Nếu không có biến môi trường nào được thiết lập thì bản địa mặc định là C.
>
> Vài thư viện bản địa hóa thông điệp cũng xem biến môi trường LANGUAGE mà nó ghi đè tất cả các thiết lập bản địa khác cho mục đích thiết lập ngôn ngữ các thông điệp. Nếu có nghi ngờ, xin tham chiếu tới tài liệu hệ điều hành của bạn, đặc biệt tài liệu về gettext.

Để cho phép các thông điệp được dịch sang ngôn ngữ ưa thích của người sử dụng, NLS phải được chọn lúc được xây dựng (cấu hình --enable-nls). Tất cả hỗ trợ bản địa khác được xây dựng tự động.

22.1.2. Hành xử

Các thiết lập bản địa ảnh hưởng tới các đặc tính SQL sau:

- Trật tự sắp xếp trong các truy vấn có sử dụng ORDER BY hoặc các toán tử so sánh tiêu chuẩn đối với các dữ liệu văn bản
- Khả năng sử dụng các chỉ số với các mệnh đề LIKE
- Các hàm upper, lower, và initcap
- Họ các hàm to_char

Trở ngại của việc sử dụng các bản địa khác với C hoặc POSIX trong PostgreSQL là ảnh hưởng về hiệu năng của nó. Nó làm chậm việc điều khiển các ký tự và ngăn cản các chỉ số thông thường khỏi việc đang được LIKE sử dụng. Vì lý do này hãy sử dụng các bản địa chỉ nếu bạn thực sự cần chúng.

Như một sự khắc phục để cho phép PostgreSQL sử dụng các chỉ số với các mệnh đề LIKE theo bản địa không phải C, vài lớp toán tử tùy biến tồn tại. Chúng cho phép tạo một chỉ số thực hiện có mẹo một sự so sánh từng ký tự một, bỏ qua các qui tắc so sánh bản địa. Hãy tham chiếu tới Phần 11.9 để có thêm thông tin.

22.1.3. Vấn đề

Nếu hỗ trợ bản địa không làm việc phù hợp với giải thích ở trên, hãy kiểm tra hỗ trợ bản địa trong hệ điều hành của bạn xem có được thiết lập cấu hình đúng hay không. Để kiểm tra các bản địa nào được cài đặt trong hệ thống của bạn, bạn có thể sử dụng lệnh locale -a nếu hệ điều hành của bạn có.

Hãy kiểm tra rằng PostgreSQL thực sự đang sử dụng bản địa mà bạn nghĩ là đúng. Các thiết lập LC_COLLATE và LC_CTYPE được xác định khi một cơ sở dữ liệu được tạo ra, và không thể bị thay đổi ngoại trừ bằng việc tạo ra một cơ sở dữ liệu mới. Các thiết lập bản địa khác, bao gồm cả LC_MESSAGES và LC_MONETARY ngay từ đầu được môi trường máy chủ được khởi động trong đó xác định, nhưng có thể bị thay đổi khi chạy. Bạn có thể kiểm tra các thiết lập bản địa tích cực bằng việc sử dụng lệnh SHOW.

Thư mục src/test/locale trong phân phối nguồn có một bộ kiểm thử cho hỗ trợ bản địa của PostgreSQL.

Các ứng dụng máy trạm mà điều khiển các lỗi phía máy chủ bằng việc phân tích văn bản các thông điệp lỗi chắc chắn sẽ có các vấn đề khi các thông điệp máy chủ là trong các ngôn ngữ khác nhau. Các tác giả của các ứng dụng như vậy được khuyến cáo sử dụng sơ đồ mã lỗi.

Việc duy trì các catalog các bản dịch thông điệp đòi hỏi các nỗ lực liên tục của nhiều người tình nguyện mà muốn thấy PostgreSQL nói tốt ngôn ngữ ưa thích của họ. Nếu các thông điệp trong ngôn ngữ của bạn hiện còn chưa sẵn sàng hoặc còn chưa được dịch, thì sự trợ giúp của bạn có thể sẽ được đánh giá cao. Nếu bạn muốn giúp, hãy tham chiếu tới Chương 48 hoặc viết vào danh sách thư của các lập trình viên.

22.2. Hỗ trợ tập hợp các ký tự

Hỗ trợ tập hợp các ký tự trong PostgreSQL cho phép bạn lưu văn bản trong một loạt các tập hợp ký tự (còn được gọi là các bảng mã), bao gồm các tập hợp ký tự 1 byte như loạt ISO 8859 và các tập hợp ký tự nhiều byte như Mã Unix Mở rộng - EUC (Extended Unix Code), UTF-8, và mã nội bộ Mule. Tất cả các tập hợp ký tự được hỗ trợ có thể được các máy trạm sử dụng cởi mở, nhưng một ít

sẽ không được hỗ trợ để sử dụng trong máy chủ (đó là, như một bộ mã phía máy chủ). Tập hợp mặc định các ký tự được chọn trong khi khởi tạo bó cơ sở dữ liệu PostgreSQL của bạn bằng việc sử dụng initdb. Nó có thể bị ghi đè khi bạn tạo một cơ sở dữ liệu, nên bạn có thể có nhiều cơ sở dữ liệu mà từng cơ sở dữ liệu với tập hợp ký tự khác nhau.

Tuy nhiên, một hạn chế quan trọng là tập hợp ký tự của từng cơ sở dữ liệu phải tương thích với các thiết lập bản địa của cơ sở dữ liệu đó như LC_CTYPE (phân loại ký tự) và LC_COLLATE (trật tự sắp xếp chuỗi). Đối với bản địa C hoặc POSIX, bất kỳ tập hợp ký tự nào cũng được phép, nhưng đối với các bản địa khác thì chỉ có một tập hợp ký tự sẽ làm việc đúng. (Tuy nhiên, trong Windows, bảng mã UTF-8 có thể sử dụng được với bất kỳ bản địa nào).

22.2.1. Tập hợp ký tự được hỗ trợ

Bảng 22-1 chỉ ra các tập hợp ký tự sẵn sàng để sử dụng trong PostgreSQL.

Bảng 22-1. Các tập hợp ký tự PostgreSQL

Tên	Mô tả	Ngôn ngữ	Máy chủ?	Byte/Char	Tên hiệu-Aliases
BIG5	Big Five	Traditional Chinese	No	1-2	WIN950, Windows950
EUC_CN	Extended UNIX Code-CN	Simplified Chinese	Yes	1-3	
EUC_JP	Extended UNIX Code-JP	Japanese	Yes	1-3	
EUC_JIS_2004	Extended UNIX Code-JP, JIS X 0213	Japanese	Yes	1-3	
EUC_KR	Extended UNIX Code-KR	Korean	Yes	1-3	
EUC_TW	Extended UNIX Code-TW	Traditional Chinese, Taiwanese	Yes	1-3	
GB18030	National Standard	Chinese	No	1-2	
GBK	Extended National Standard	Simplified Chinese	No	1-2	WIN936, Windows936
ISO_8859_5	ISO 8859-5, ECMA 113	Latin/Cyrillic	Yes	1	
ISO_8859_6	ISO 8859-6, ECMA 114	Latin/Arabic	Yes	1	
ISO_8859_7	ISO 8859-7, ECMA 118	Latin/Greek	Yes	1	
ISO_8859_8	ISO 8859-8, ECMA 121	Latin/Hebrew	Yes	1	
JOHAB	JOHAB	Korean (Hangul)	No	1-3	
KOI8R	KOI8-R	Cyrillic (Russian)	Yes	1	KOI8
KOI8U	KOI8-U	Cyrillic (Ukrainian)	Yes	1	
LATIN1	ISO 8859-1, ECMA 94	Western European	Yes	1	ISO88591
LATIN2	ISO 8859-2, ECMA 94	Central European	Yes	1	ISO88592
LATIN3	ISO 8859-3, ECMA 94	South European	Yes	1	ISO88593
LATIN4	ISO 8859-4, ECMA 94	North European	Yes	1	ISO88594
LATIN5	ISO 8859-9, ECMA 128	Turkish	Yes	1	ISO88599
LATIN6	ISO 8859-10, ECMA 144	Nordic	Yes	1	ISO885910
LATIN7	ISO 8859-13	Baltic	Yes	1	ISO885913

Tên	Mô tả	Ngôn ngữ	Máy chủ?	Byte/Char	Tên hiệu-Aliases
LATIN8	ISO 8859-14	Celtic	Yes	1	ISO885914
LATIN9	ISO 8859-15	LATIN1 with Euro and accents	Yes	1	ISO885915
LATIN10	ISO 8859-16, ASRO SR 14111	Romanian	Yes	1	ISO885916
MULE_INTERNAL	Mule internal code	Multilingual Emacs	Yes	1-4	
SJIS	Shift JIS	Japanese	No	1-2	Mskanji, ShiftJIS, WIN932, Windows932
SHIFT_JIS_2004	Shift JIS, JIS X 0213	Japanese	No	1-2	
SQL_ASCII	unspecified (see text)	any	Yes	1	
UHC	Unified Hangul Code	Korean	No	1-2	WIN949, Windows949
UTF8	Unicode, 8-bit	all	Yes	1-4	
Unicode WIN866	Windows CP866	Cyrillic	Yes	1	ALT
WIN874	Windows CP874	Thai	Yes	1	
WIN1250	Windows CP1250	Central European	Yes	1	
WIN1251	Windows CP1251	Cyrillic	Yes	1	WIN
WIN1252	Windows CP1252	Western European	Yes	1	
WIN1253	Windows CP1253	Greek	Yes	1	
WIN1254	Windows CP1254	Turkish	Yes	1	
WIN1255	Windows CP1255	Hebrew	Yes	1	
WIN1256	Windows CP1256	Arabic	Yes	1	
WIN1257	Windows CP1257	Baltic	Yes	1	
WIN1258	Windows CP1258	Vietnamese	Yes	1	ABC, TCVN, TCVN5712, VSCII

Không phải tất cả các giao diện lập trình ứng dụng - API (Application Programming Interface) đều hỗ trợ tất cả các tập hợp ký tự được liệt kê ở trên. Ví dụ, trình điều khiển JDBC của PostgreSQL không hỗ trợ MULE_INTERNAL, LATIN6, LATIN8, và LATIN10.

Thiết lập SQL_ASCII hành xử khác nhau đáng kể đối với các thiết lập khác. Khi tập hợp ký tự của máy chủ là SQL_ASCII, thì máy chủ biên dịch các giá trị byte 0-127 theo tiêu chuẩn ASCII, trong khi các giá trị byte 128-255 được lấy như là các ký tự không biên dịch. Không biến đổi mã nào sẽ được thực hiện khi thiết lập là SQL_ASCII. Vì thế, thiết lập này không phải là một khai báo nhiều rằng một bảng mã đặc thù đang được sử dụng, như một khai báo bỏ qua về bảng mã. Trong hầu hết các trường hợp, nếu bạn đang làm việc với bất kỳ dữ liệu nào không phải ASCII, thì sẽ là không khôn ngoan để sử dụng thiết lập SQL_ASCII vì PostgreSQL sẽ không có khả năng giúp bạn bằng việc biến đổi hoặc kiểm tra hợp lệ các ký tự không phải là ASCII.

22.2.2. Thiết lập tập hợp ký tự

initdb xác định tập hợp (bộ mã) ký tự mặc định cho một bó PostgreSQL. Ví dụ,

```
initdb -E EUC_JP
```

thiết lập tập hợp ký tự mặc định cho EUC_JP (mã Unix mở rộng cho Nhật bản). Bạn có thể sử dụng --encoding thay cho -E nếu bạn ưa các chuỗi lựa chọn dài hơn. Nếu lựa chọn -E hoặc --encoding không được đưa ra, thì initdb sẽ cố xác định bộ mã phù hợp để sử dụng dựa vào bản địa được chỉ định hoặc mặc định.

Bạn có thể chỉ định một bảng mã không mặc định ở thời điểm tạo cơ sở dữ liệu, miễn là bảng mã đó tương thích với bản địa được chọn:

```
createdb -E EUC_KR -T template0 --lc-collate=ko_KR.euckr --lc-ctype=ko_KR.euckr korean
```

Điều này sẽ tạo một cơ sở dữ liệu có tên là korean mà sử dụng tập hợp ký tự EUC_KR, và bản địa ko_KR. Cách khác để thực hiện điều này là sử dụng lệnh SQL này:

```
CREATE DATABASE korean WITH ENCODING 'EUC_KR' LC_COLLATE='ko_KR.euckr'
LC_CTYPE='ko_KR.euckr'
```

Lưu ý là các lệnh ở trên chỉ định việc sao chép cơ sở dữ liệu template0. Khi sao chép bất kỳ cơ sở dữ liệu nào, các thiết lập bảng mã và bản địa không thể bị thay đổi từ những thứ đó của cơ sở dữ liệu nguồn, vì điều đó có thể gây ra hỏng dữ liệu. Để có thêm thông tin, xem Phần 21.3.

Bảng mã đối với một cơ sở dữ liệu được lưu trữ trong catalog hệ thống pg_database. Bạn có thể thấy nó bằng việc sử dụng lựa chọn psql -l hoặc lệnh \l.

```
$ psql -l
```

Danh sách các cơ sở dữ liệu

Name	Owner	Encoding	Collation	Ctype	Access Privileges
clocaledb	hlinnaka	SQL_ASCII	C	C	
englishdb	hlinnaka	UTF8	en_GB.UTF8	en_GB.UTF8	
japanese	hlinnaka	UTF8	ja_JP.UTF8	ja_JP.UTF8	
korean	hlinnaka	EUC_KR	ko_KR.euckr	ko_KR.euckr	
postgres	hlinnaka	UTF8	fi_FI.UTF8	fi_FI.UTF8	
template0	hlinnaka	UTF8	fi_FI.UTF8	fi_FI.UTF8	{=c/hlinnaka,hlinnaka=CTc/
template1	hlinnaka	UTF8	fi_FI.UTF8	fi_FI.UTF8	{=c/hlinnaka,hlinnaka=CTc/

(7 rows)

> **Quan trọng**: Trong hầu hết các hệ điều hành hiện đại, PostgreSQL có thể xác định tập hợp ký tự nào được thiết lập LC_CTYPE ngụ ý, và nó sẽ ép buộc điều đó chỉ bảng mã cơ sở dữ liệu trùng khớp được sử dụng. Trong các hệ thống cũ hơn thì đó là trách nhiệm của bạn để đảm bảo rằng bạn sử dụng bảng mã được kỳ vọng bằng bản địa mà bạn đã chọn. Một sai lầm trong vùng này có khả năng dẫn tới hành vi lạ lẫm các hoạt động phụ thuộc vào bản địa như việc sắp xếp.
>
> PostgreSQL sẽ cho phép những siêu người sử dụng tạo ra các cơ sở dữ liệu với bảng mã

SQL_ASCII thậm chí khi LC_CTYPE không phải là C hoặc POSIX. Như được lưu ý ở trên, SQL_ASCII không ép buộc rằng dữ liệu được lưu trữ trong cơ sở dữ liệu đó có bất kỳ bảng mã đặc biệt nào, và vì thế lựa chọn này đặt ra các rủi ro đối với hành xử sai phụ thuộc bản địa. Việc sử dụng sự kết hợp các thiết lập này bị phản đối và có thể một ngày nào đó bị cấm.

22.2.3. Chuyển đổi tập hợp ký tự tự động giữa máy chủ/máy trạm

PostgreSQL hỗ trợ chuyển đổi tập hợp ký tự tự động giữa máy chủ và máy trạm cho các kết hợp tập hợp ký tự nhất định. Thông tin chuyển đổi được lưu giữ trong catalog hệ thống pg_conversion.

PostgreSQL đi với một vài sự chuyển đổi được xác định trước, như được chỉ ra trong Bảng 22-2. Bạn có thể tạo một chuyển đổi mới bằng việc sử dụng lệnh SQL CREATE CONVERSION.

Bảng 22-2. Các chuyển đổi tập hợp ký tự máy trạm/máy chủ

Tập hợp ký tự máy chủ	Các tập hợp ký tự máy trạm có sẵn
BIG5	không được hỗ trợ như một bộ mã máy chủ
EUC_CN	EUC_CN, MULE_INTERNAL, UTF8
EUC_JP	EUC_JP, MULE_INTERNAL, SJIS, UTF8
EUC_KR	EUC_KR, MULE_INTERNAL, UTF8
EUC_TW	EUC_TW, BIG5, MULE_INTERNAL, UTF8
GB18030	không được hỗ trợ như một bộ mã máy chủ
GBK	không được hỗ trợ như một bộ mã máy chủ
ISO_8859_5	ISO_8859_5, KOI8R, MULE_INTERNAL, UTF8, WIN866, WIN1251
ISO_8859_6	ISO_8859_6, UTF8
ISO_8859_7	ISO_8859_7, UTF8
ISO_8859_8	ISO_8859_8, UTF8
JOHAB	JOHAB, UTF8
KOI8R	KOI8R, ISO_8859_5, MULE_INTERNAL, UTF8, WIN866, WIN1251
KOI8U	KOI8U, UTF8
LATIN1	LATIN1, MULE_INTERNAL, UTF8
LATIN2	LATIN2, MULE_INTERNAL, UTF8, WIN1250
LATIN3	LATIN3, MULE_INTERNAL, UTF8
LATIN4	LATIN4, MULE_INTERNAL, UTF8
LATIN5	LATIN5, UTF8
LATIN6	LATIN6, UTF8
LATIN7	LATIN7, UTF8
LATIN8	LATIN8, UTF8
LATIN9	LATIN9, UTF8
LATIN10	LATIN10, UTF8
MULE_INTERNAL	MULE_INTERNAL, BIG5, EUC_CN, EUC_JP, EUC_KR, EUC_TW, ISO_8859_5, KOI8R, LATIN1 to LATIN4, SJIS, WIN866, WIN1250, WIN1251
SJIS	không được hỗ trợ như một bộ mã máy chủ

Tập hợp ký tự máy chủ	Các tập hợp ký tự máy trạm có sẵn
SQL_ASCII	bất kỳ (không sự chuyển đổi nào sẽ được thực hiện)
UHC	không được hỗ trợ như một bộ mã máy chủ
UTF8	tất cả các bộ mã được hỗ trợ
WIN866	WIN866, ISO_8859_5, KOI8R, MULE_INTERNAL, UTF8, WIN1251
WIN874	WIN874, UTF8
WIN1250	WIN1250, LATIN2, MULE_INTERNAL, UTF8
WIN1251	WIN1251, ISO_8859_5, KOI8R, MULE_INTERNAL, UTF8, WIN866
WIN1252	WIN1252, UTF8
WIN1253	WIN1253, UTF8
WIN1254	WIN1254, UTF8
WIN1255	WIN1255, UTF8
WIN1256	WIN1256, UTF8
WIN1257	WIN1257, UTF8
WIN1258	WIN1258, UTF8

Để kích hoạt chuyển đổi tập hợp ký tự tự động, bạn phải nói cho PostgreSQL tập hợp (bảng mã) ký tự mà bạn muốn sử dụng trong máy trạm đó. Có vài cách để thực hiện điều này:

- Sử dụng lệnh \encoding trong psql. \encoding cho phép bạn thay đổi bảng mã máy trạm khi đang chạy. Ví dụ, để thay đổi bảng mã về SJIS, hãy gõ:

  ```
  \encoding SJIS
  ```

- libpq (Phần 31.9) có các hàm để kiểm soát bảng mã máy trạm.
- Sử dụng SET client_encoding TO. Việc thiết lập bảng mã máy trạm có thể được thực hiện bằng lệnh SQL này:

  ```
  SET CLIENT_ENCODING TO 'value';
  ```

  Bạn cũng có thể sử dụng cú pháp SQL tiêu chuẩn SET NAMES cho mục đích này:

  ```
  SET NAMES 'value';
  ```

  Để truy vấn bảng mã máy trạm hiện hành:

  ```
  SHOW client_encoding;
  ```

  Để trở về bảng mã mặc định:

  ```
  RESET client_encoding;
  ```

- Sử dụng PGCLIENTENCODING. Nếu biến môi trường PGCLIENTENCODING được xác định trong môi trường máy trạm, thì bảng mã máy trạm tự động được chọn khi một kết nối tới máy chủ được thực hiện. (Điều này có thể sau đó bị ghi đè bằng việc sử dụng bất kỳ phương pháp nào khác được nhắc tới ở trên).
- Sử dụng biến cấu hình client_encoding. Nếu biến client_encoding được thiết lập, bảng mã máy trạm đó được tự động lựa chọn khi một kết nối tới máy chủ được thực hiện. (Điều này

có thể sau đó bị ghi đè bằng việc sử dụng bất kỳ phương pháp nào khác được nêu ở trên).

Nếu sự chuyển đổi một ký tự đặc biệt là không thể - giả thiết bạn chọn EUC_JP cho máy chủ và LATIN1 cho máy trạm, và một vài ký tự tiếng Nhật được trả về mà không có một trình diễn ở LATIN1 - một lỗi sẽ được báo cáo.

Nếu tập hợp ký tự máy trạm được xác định như là SQL_ASCII, thì chuyển đổi bảng mã bị vô hiệu hóa, bất kể tập hợp ký tự máy chủ nào. Chỉ là cho máy chủ, việc sử dụng SQL_ASCII là không khôn ngoan trừ phi bạn đang làm việc với tất các các dữ liệu ASCII.

22.2.4. Đọc thêm

Đó là các nguồn tốt để bắt đầu học về các dạng khác nhau các hệ thống bảng mã.

http://www.i18ngurus.com/docs/984813247.html

Một bộ sưu tập mở rộng các tài liệu về các tập hợp ký tự, các bảng mã, và các trang mã.

Việc xử lý thông tin CJKV: Điện toán của Trung Quốc, Nhật, Hàn & Việt Nam

Có những giải thích chi tiết về EUC_JP, EUC_CN, EUC_KR, EUC_TW.

http://www.unicode.org/

Website của nhóm Unicode.

RFC 3629

UTF-8 (Định dạng biến đổi 8 bit UCS/Unicode) được định nghĩa ở đây.

Chương 23. Các tác vụ duy trì cơ sở dữ liệu thường ngày

PostgreSQL, giống như bất kỳ phần mềm cơ sở dữ liệu nào, đòi hỏi các tác vụ nhất định được thực hiện thường xuyên để đạt được hiệu năng tối ưu. Các tác vụ được thảo luận ở đây được yêu cầu, nhưng chúng lặp đi lặp lại về bản chất tự nhiên và có thể dễ dàng được tự động hóa bằng việc sử dụng các công cụ tiêu chuẩn như các cron script hoặc trình lập lịch tác vụ (Task Scheduler) của Windows. Chính người quản trị cơ sở dữ liệu có trách nhiệm thiết lập các script đúng phù hợp, và kiểm tra xem chúng có thực thi thành công hay không.

Một tác vụ duy trì rõ ràng là sự tạo ra các bản sao lưu các dữ liệu theo lịch thường xuyên. Không có một sao lưu gần đây, bạn sẽ không có cơ hội phục hồi sau một thảm họa (hỏng đĩa, cháy, loại bỏ một bảng sống còn một cách sai lầm, ...). Các cơ chế sao lưu và phục hồi sẵn sàng trong PostgreSQL sẽ được thảo luận dài trong Chương 24.

Chủng loại tác vụ duy trì chính khác là "việc hút chân không" (vacuuming) định kỳ cho cơ sở dữ liệu. Hoạt động này được thảo luận trong Phần 23.1. Có liên quan chặt chẽ tới điều này là việc cập nhật các số liệu thống kê sẽ được trình lên kế hoạch truy vấn sử dụng, như được thảo luận ở Phần 23.1.3.

Một tác vụ khác có thể cần sự chú ý định kỳ là quản lý tệp lưu ký. Điều này được thảo luận trong Phần 23.3.

check_postgres[1] là sẵn sàng cho việc giám sát sức khỏe cơ sở dữ liệu và báo cáo các điều kiện không bình thường. check_postgres tích hợp với Nagios và MRTG, nhưng cũng có thể được chạy riêng rẽ một mình.

PostgreSQL là duy trì thấp khi so với một vài hệ thống quản trị cơ sở dữ liệu khác. Mặc dù vậy, sự chú ý đúng phù hợp đối với các tác vụ đó sẽ đi được xa trong việc đảm bảo một trải nghiệm có hiệu suất và hài lòng với hệ thống.

23.1. Việc hút chân không thường ngày

Các cơ sở dữ liệu PostgreSQL đòi hỏi duy trì định kỳ được biết như là việc hút chân không. Đối với nhiều cài đặt, là đủ để cho phép việc hút chân không được thực hiện bằng daemon hút chân không tự động (autovacuum), nó được mô tả trong Phần 23.1.5. Bạn có thể cần tinh chỉnh các tham số của việc hút chân không tự động được mô tả ở đó để có được các kết quả tốt nhất cho tình huống của bạn. Một vài nhà quản trị cơ sở dữ liệu sẽ muốn bổ sung hoặc thay thế các hoạt động của daemon đó bằng các lệnh VACUUM được quản lý bằng tay, chúng thường được thực thi tùy theo lịch của cron hoặc các script của trình lên lịch các tác vụ (Task Scheduler). Để thiết lập cho việc hút chân không tự động được quản lý bằng tay một cách đúng phù hợp, là cơ bản phải hiểu các vấn đề được thảo luận trong một vài phần con tiếp sau. Các nhà quản trị mà dựa vào việc hút chân không tự động có thể vẫn muốn gạn lọc tư liệu này để giúp họ hiểu và tinh chỉnh việc hút chân không tự động.

23.1.1. Cơ bản về hút chân không tự động

Lệnh VACUUM của PostgreSQL phải xử lý từng bảng trên cơ sở thường xuyên vì vài lý do:

1 http://bucardo.org/wiki/Check_postgres

1. Để phục hồi hoặc sử dụng lại không gian đĩa bị các hàng được cập nhật hoặc bị xóa chiếm.
2. Để cập nhật các số liệu thống kê dữ liệu được trình lên kế hoạch truy vấn của PostgreSQL sử dụng.
3. Để bảo vệ chống lại sự mất các dữ liệu quá cũ vì sự *bao bọc mã giao dịch* (transaction ID).

Mỗi trong số các lý do đó chỉ ra việc thực hiện các hoạt động của VACUUM với tần suất và phạm vi khác nhau, như được giải thích trong các phần con sau đây.

Có 2 phương án của VACUUM: VACUUM tiêu chuẩn và VACUUM FULL. VACUUM FULL có thể khai báo nhiều không gian đĩa hơn nhưng chạy chậm hơn nhiều. Hơn nữa, dạng VACUUM tiêu chuẩn có thể chạy song song với các hoạt động cơ sở dữ liệu sản xuất. (Các lệnh như SELECT, INSERT, UPDATE, và DELETE sẽ tiếp tục hoạt động bình thường, dù bạn sẽ không có khả năng để sửa đổi định nghĩa một bảng với các lệnh như ALTER TABLE trong khi nó đang được vacuum). VACUUM FULL đòi hỏi khóa hoàn toàn bảng mà nó đang làm việc, và vì thế không thể được thực hiện song song khi cùng sử dụng bảng đó. Vì thế, thường các quản trị viên sẽ cố sử dụng VACUUM chuẩn và tránh VACUUM FULL.

VACUUM tạo ra một lượng giao thông I/O đáng kể, nó có thể làm cho hiệu năng tồi đi đối với các phiên làm việc đang hoạt động khác. Có các tham số cấu hình mà có thể được tinh chỉnh để làm giảm ảnh hưởng hiệu năng của việc hút chân không phụ trợ - xem Phần 18.4.3.

23.1.2. Phục hồi không gian đĩa

Trong PostgreSQL, một lệnh UPDATE hoặc DELETE một hàng không ngay lập tức loại bỏ phiên bản hàng cũ. Tiếp cận này là cần thiết để có được các lợi ích kiểm soát đồng thời nhiều phiên bản (MVCC, xem Chương 13): phiên bản hàng phải không bị xóa trong khi nó vẫn còn tiềm tàng nhìn thấy được đối với các giao dịch khác. Nhưng rốt cuộc, một phiên bản hàng lỗi thời hoặc bị xóa không còn được quan tâm đối với bất kỳ giao dịch nào. Không gian mà nó chiếm sau đó được khai báo để sử dụng lại đối với các hàng mới, để tránh sự gia tăng vô tận các yêu cầu không gian đĩa. Điều này được làm bằng việc chạy VACUUM.

Dạng VACUUM tiêu chuẩn loại bỏ các phiên bản hàng chết trong các bảng và các chỉ số và đánh dấu không gian đó sẵn sàng cho sử dụng lại trong tương lai. Tuy nhiên, nó sẽ không trả không gian đó về cho hệ điều hành, ngoại trừ trong trường hợp đặc biệt nơi mà một hoặc nhiều trang hơn ở cuối của một bảng trở nên hoàn toàn tự do và một khóa bảng hoàn toàn có thể dễ dàng có được. Ngược lại, VACUUM FULL ảnh hưởng tích cực tới các bảng bằng việc ghi một phiên bản mới hoàn toàn tệp bảng đó mà không có không gian chết. Điều này làm giảm tối thiểu kích cỡ của bảng, nhưng có thể mất thời gian lâu hơn. Nó cũng đòi hỏi thêm không gian đĩa cho bản sao mới của bảng, cho tới khi hoạt động đó hoàn tất.

Mục đích thông thường của việc hút chân không thường xuyên là để thực hiện VACUUM tiêu chuẩn đủ thường xuyên để tránh việc cần tới VACUUM FULL. Deamon hút chân không tự động (autovacuum) cố gắng làm việc theo cách này, và trong thực tế sẽ không bao giờ đưa ra VACUUM FULL. Theo tiếp cận này, ý tưởng là không giữ các bảng ở kích cỡ tối thiểu của chúng, mà duy trì sử dụng không gian đĩa ở tình trạng đều đặn: mỗi bảng chiếm không gian tương ứng với kích thước hơn với tối thiểu của nó tuy nhiên nhiều không gian được sử dụng hết giữa các cuộc hút chân không. Dù

VACUUM FULL có thể được sử dụng để thu một bảng về kích cỡ tối thiểu của nó và trả không gian đĩa về cho hệ điều hành, không có nhiều điểm trong việc này nếu bảng đó sẽ chỉ tăng lại một lần nữa trong tương lai. Vì thế, chạy VACUUM tiêu chuẩn thường xuyên vừa phải sẽ là một tiếp cận tốt hơn so với chạy VACUUM FULL không thường xuyên cho việc duy trì các bảng được cập nhật nhiều.

Một vài quản trị viên thích đặt lịch hút chân không hơn cho bản thân họ, ví dụ làm tất cả công việc vào buổi tối khi tải là thấp. Khó khăn với việc hút chân không theo một lịch cố định là vì nếu một bảng có một hỏng hóc không mong đợi trong hoạt động cập nhật, thì nó có thể nở nhanh tới điểm mà VACUUM FULL thực sự là cần thiết để cải tạo không gian. Sử dụng daemon hút chân không tự động làm giảm bớt vấn đề này, vì daemon đó đặt lịch cho việc hút chân không một cách năng động để phản hồi cho hoạt động cập nhật. Là không khôn ngoan để vô hiệu hóa daemon này hoàn toàn trừ phi bạn có một tải công việc có thể cực kỳ dễ đoán trước được. Một sự tổn thương có khả năng là thiết lập các tham số của daemon đó sao cho nó sẽ chỉ phản ứng với hoạt động cập nhật rất không bình thường, vì thế giữ được cho mọi điều không tuột khỏi tầm tay, trong khi VACUUM được lên lịch sẽ được kỳ vọng thực hiện đống công việc khi tải là điển hình.

Đối với những ai không sử dụng hút chân không tự động, một tiếp cận điển hình là hãy đặt lịch cho một VACUUM rộng khắp một cơ sở dữ liệu 1 lần mỗi ngày trong một giai đoạn sử dụng ít, được bổ sung bằng việc hút chân không thường xuyên hơn các bảng được cập nhật nhiều như cần thiết. (Một vài cài đặt với tỷ lệ cập nhật cực kỳ cao sẽ vacuum các bảng bận rộn nhất của chúng thường xuyên 1 lần trong vài phút). Nếu bạn có nhiều bảng trong một bó, đừng quên VACUUM từng bảng; chương trình vacuumdb có thể là hữu dụng.

> **Mẹo**: VACUUM đơn giản có thể không hài lòng khi một bảng chứa số lượng lớn các phiên bản hàng chết như là kết quả của hoạt động cập nhật hoặc xóa rất nhiều. Nếu bạn có một bảng như vậy và bạn cần cải tạo không gian đĩa quá mức mà nó chiếm, thì bạn sẽ cần sử dụng VACUUM FULL, hoặc như một sự lựa chọn sử dụng CLUSTER hoặc một trong các phương án ghi lại bảng của ALTER TABLE. Các lệnh đó ghi lại toàn bộ một bản sao mới của bảng đó và xây dựng các chỉ số mới cho nó. Tất cả các lựa chọn đó đòi hỏi sự khóa hoàn toàn. Lưu ý là chúng cũng sử dụng tạm thời không gian đĩa thêm khoảng chừng bằng kích cỡ của bảng đó, vì các bản sao cũ của bảng và các chỉ số không thể được đưa ra cho tới khi những thứ mới sẽ hoàn chỉnh.

> **Mẹo**: Nếu bạn có một bảng mà toàn bộ nội dung của nó bị xóa trong một khoảng thời gian, hãy xem xét làm điều đó bằng lệnh TRUNCATE hơn là bằng việc sử dụng DELETE theo sau VACUUM. Lệnh TRUNCATE loại bảo toàn bộ nội dung của bảng ngay lập tức, không đòi hỏi VACUUM hoặc VACUUM FULL tiếp sau để cải tạo không gian đĩa bây giờ không được sử dụng. Nhược điểm là ngữ nghĩa chặt chẽ của MVCC sẽ bị vi phạm.

23.1.3. Cập nhật số liệu thống kê của trình lên kế hoạch

Trình lên kế hoạch truy vấn của PostgreSQL dựa vào thông tin số liệu thống kê về các nội dung các bảng để lập các kế hoạch tốt cho các truy vấn. Các số liệu thống kê đó thường được lệnh ANALYZE thu thập, nó có thể tự nó bị triệu gọi hoặc như một bước tùy chọn trong VACUUM. Điều quan trọng phải có các số liệu thống kê chính xác một cách hợp lý, nếu không các lựa chọn tồi của các kế hoạch có thể làm hỏng hiệu năng của cơ sở dữ liệu.

Daemon hút chân không tự động, nếu được kích hoạt, sẽ tự động đưa ra lệnh ANALYZE bất kể khi nào nội dung của một bảng đã thay đổi đủ thích đáng. Tuy nhiên, các nhà quản trị có thể ưa thích dựa vào các hoạt động được lên lịch bằng tay của ANALYZE, đặc biệt nếu được biết rằng hoạt động cập nhật trong một bảng sẽ không ảnh hưởng tới các số liệu thống kê của các cột "có quan tâm". Deamon đó đặt lịch cho ANALYZE chặt chẽ như một hàm của số lượng các hàng được chèn vào hoặc được cập nhật; nó không biết liệu điều đó có dẫn tới những thay đổi thống kê có ý nghĩa hay không.

Như với việc hút chân không để phục hồi không gian, các bản cập nhật thường xuyên các số liệu thống kê là hữu dụng cho các bảng được cập nhật nhiều hơn là các bảng hiếm khi được cập nhật. Nhưng thậm chí đối với một bảng được cập nhật nhiều, vẫn có thể không cần cho các bản cập nhật các số liệu thống kê nếu sự phân phối các dữ liệu thống kê không làm thay đổi nhiều. Một qui tắc ngón tay cái đơn giản là hãy nghĩ xem tối đa và tối thiểu có bao nhiêu giá trị của các cột trong bảng thay đổi. Ví dụ, một cột timestamp có chứa thời gian cập nhật hàng sẽ có giá trị cực đại gia tăng liên tục khi các hàng được bổ sung thêm và được cập nhật; một cột như vậy có thể sẽ cần nhiều bản cập nhật các số liệu thống kê hơn, ví dụ, một cột có chứa các URL cho các trang được truy cập trên một website. Cột URL có thể nhận các thay đổi thật thường xuyên, nhưng sự phân phối thống kê các giá trị của nó có lẽ thay đổi tương đối chậm.

Có khả năng chạy ANALYZE trong các bảng đặc biệt và thậm chí chỉ các cột đặc biệt của một bảng, nên sự mềm dẻo tồn tại để cập nhật một vài số liệu thống kê thường xuyên hơn so với các số liệu khác nếu ứng dụng của bạn đòi hỏi nó. Tuy nhiên, trong thực tế, thường là tốt nhất để chỉ phân tích toàn bộ cơ sở dữ liệu, vì nó là một hoạt động nhanh. ANALYZE sử dụng một cách lấy mẫu ngẫu nhiên các số liệu thống kê các hàng của một bảng hơn là đọc từng hàng một.

> **Mẹo**: Dù việc thường xuyên tùy biến từng cột của lệnh ANALYZE có thể rất không hiệu quả, bạn có thể thấy nó đáng để thực hiện sự tinh chỉnh theo từng cột ở mức chi tiết đối với các số liệu thống kê được ANALYZE thu thập. Các cột được sử dụng nhiều trong các mệnh đề WHERE và có các phân phối dữ liệu không thường xuyên cao có thể đòi hỏi một biểu đồ dữ liệu mịn hơn so với các cột khác. Xem ALTER TABLE SET STATISTICS, hoặc thay đổi mặc định rộng khắp cơ sở dữ liệu bằng việc sử dụng tham số cấu hình default_statistics_target.
>
> Hơn nữa, mặc định có thông tin có giới hạn sẵn sàng về khả năng lựa chọn các hàm. Tuy nhiên, nếu bạn tạo một chỉ số biểu thức mà sử dụng một lời gọi hàm, thì các số liệu thống kê hữu dụng sẽ được thu thập về hàm đó, nó có thể cải thiện lớn các kế hoạch truy vấn mà sử dụng chỉ số biểu thức đó.

23.1.4. Ngăn chặn hỏng hóc quanh mã giao dịch (Transaction ID)

Ngữ nghĩa giao dịch MVCC của PostgreSQL phụ thuộc vào khả năng so sánh các số mã số giao dịch ID (XID): một phiên bản hàng với một sự chèn XID lớn hơn so với XID giao dịch hiện hành là "trong tương lai" và sẽ không nhìn thấy được đối với giao dịch hiện hành. Nhưng vì các mã ID giao dịch có kích cỡ có giới hạn (32 bit) nên một bó mà chạy thời gian dài (hơn 4 tỷ giao dịch) có thể chịu sự bao quanh mã ID giao dịch: bộ đếm XID bao quanh zero, và tất cả các giao dịch ngẫu nhiên từng có trong quá khứ dường như sẽ có trong tương lai - nó có nghĩa là đầu ra của chúng đã trở nên không nhìn thấy được. Ngắn gọn, thảm họa mất dữ liệu. (Thực sự thì các dữ liệu vẫn còn ở đó, nhưng đó là sự thuận tiện lạnh lẽo nếu bạn không thể có được nó). Để tránh điều này, cần thiết phải vacuum từng bảng trong từng cơ sở dữ liệu ít nhất 1 lần cho mỗi 2 tỷ giao dịch.

Lý do việc hút chân không định kỳ giải quyết được vấn đề là vì PostgreSQL giữ lại một XID đặc biệt như là FrozenXID. XID này không tuân theo các qui tắc so sánh XID thông thường và luôn được xem là cũ hơn so với mọi XID thông thường. Các XID thông thường được so sánh bằng việc sử dụng tính toán số học modulo-2^{31}. Điều này có nghĩa là đối với mỗi XID thông thường, có 2 tỷ các XID mà là "cũ hơn" và 2 tỷ là "mới hơn"; cách khác để nói là không gian XID thông thường là sống còn mà không có điểm kết thúc. Vì thế, một khi một phiên bản hàng đã được tạo ra với một XID đặc biệt thông thường, thì phiên bản hàng dường như sẽ là "trong quá khứ" cho 2 tỷ giao dịch tiếp theo, bất kể XID thông thường nào chúng ta đang nói tới. Nếu phiên bản hàng vẫn tồn tại sau hơn 2 tỷ giao dịch, thì nó bỗng nhiên sẽ dường như là ở trong tương lai. Để ngăn chặn điều này, các phiên bản hàng cũ phải được chỉ định lại XID cho FrozenXID đôi khi trước khi chúng đạt tới được mốc 2 tỷ giao dịch cũ đó. Một khi chúng được chỉ định XID đặc biệt này, thì chúng dường như sẽ là "trong quá khứ" đối với tất cả các giao dịch thông thường bất kể các vấn đề bao quanh, và vì thế các phiên bản hàng như vậy sẽ là hợp lệ cho tới khi bị xóa, bất kể điều đó dài lâu thế nào. Sự tái chỉ định này các XID cũ được VACUUM điều khiển.

vacuum_freeze_min_age kiểm soát một giá trị XID sẽ là cũ thế nào trước khi nó được thay thế bằng FrozenXID. Các giá trị rộng lớn hơn của thiết lập này giữ lại thông tin giao dịch lâu hơn, trong khi các giá trị nhỏ hơn làm gia tăng số lượng các giao dịch trôi qua trước khi bảng đó phải được hút chân không một lần nữa.

VACUUM thường bỏ qua các trang không có bất kỳ phiên bản hàng chết nào, nhưng các trang đó có thể vẫn có các phiên bản hàng với các giá trị XID cũ. Để đảm bảo tất cả các XID cũ đã được FrozenXID thay thế, một sự quét toàn bộ bảng là cần thiết. vacuum_freeze_table_age kiểm soát khi VACUUM làm điều đó; toàn bộ sự quét một bảng bị ép buộc nếu bảng đó còn chưa được quét toàn bộ đối với các giao dịch vacuum_freeze_table_age trừ đi vacuum_freeze_min_age. Việc thiết lập nó về 0 sẽ ép VACUUM về luôn quét tất cả các trang, bỏ qua có hiệu quả khả năng nhìn thấy được.

Thời gian tối đa mà một bảng có thể không được đi hút chân không là 2 tỷ giao dịch trừ đi giá trị vacuum_freeze_min_age ở thời điểm VACUUM đã quét lần cuối toàn bộ bảng đó. Nếu nó từng được đi hút chân không dài hơn so với điều đó, thì sự mất dữ liệu có thể xảy ra. Để đảm bảo rằng điều này không xảy ra, hút chân không tự động được triệu gọi trong bất kỳ bảng nào mà có thể chứa các XID cũ hơn so với tuổi được tham số cấu hình autovacuum_freeze_max_age chỉ định. (Điều này sẽ xảy

ra thậm chí nếu sự hút chân không tự động bị vô hiệu hóa).

Điều này ngụ ý rằng nếu một bảng nếu khác đi không được hút chân không, thì sự hút chân không tự động sẽ được triệu gọi trong bảng đó khoảng 1 lần mỗi autovacuum_freeze_max_age trừ đi vacuum_freeze_min_age các giao dịch. Đối với các bảng mà thường xuyên được hút chân không vì các mục đích cải tạo không gian, thì điều này ít quan trọng.

Tuy nhiên, đối với các bảng tĩnh (bao gồm các bảng mà nhận các bản chèn, nhưng không cập nhật hoặc xóa), sẽ không cần phải hút chân không để cải thiện không gian, nên có thể là hữu dụng để thử tối đa hóa khoảng thời gian giữa các lần tự động hút chân không bị ép trong từng bảng tĩnh lớn. Rõ ràng người ta có thể làm điều này hoặc bằng việc làm gia tăng autovacuum_freeze_max_age hoặc làm giảm vacuum_freeze_min_age.

Tối đa có hiệu quả cho vacuum_freeze_table_age là 0.95*autovacuum_freeze_max_age; một thiết lập cao hơn điều đó sẽ ép về tối đa. Một giá trị cao hơn autovacuum_freeze_max_age có lẽ không có ý nghĩa vì một sự tự động hút chân không chống lại sự bao quanh có thể được kích hoạt ở bất kỳ điểm nào, và bội số của 0.95 để lại vài chỗ thở để chạy một VACUUM bằng tay trước khi điều đó xảy ra. Như một qui tắc ngón tay cái, vacuum_freeze_table_age sẽ được thiết lập về một giá trị đâu đó dưới autovacuum_freeze_max_age, để lại đủ khe hở sao cho một VACUUM được lên lịch thường xuyên hoặc một sự tự động hút chân không được hoạt động xóa và cập nhật thông thường làm bật dậy sẽ được chạy trong cửa sổ đó. Việc thiết lập nó quá sát có thể dẫn tới các tự động hút chân không chống bao quanh, thậm chí dù bảng đó từng được hút chân không gần đây để cải tạo không gian, trong khi các giá trị thấp hơn dẫn tới các cuộc quét toàn bộ bảng thường xuyên hơn.

Nhược điểm duy nhất của việc làm gia tăng autovacuum_freeze_max_age (và vacuum_freeze_table_age cùng với nó) là việc thư mục con pg_clog của bó cơ sở dữ liệu sẽ chiếm nhiều không gian hơn, vì nó phải lưu trữ tình trạng thực hiện (commit) tất cả các giao dịch ngược về đường nằm ngang autovacuum_freeze_max_age. Tình trạng thực hiện sử dụng 2 bit cho từng giao dịch, nên nếu autovacuum_freeze_max_age được thiết lập về giá trị cực đại 2 tỷ được phép của nó, thì pg_clog có thể được kỳ vọng gia tăng tới khoảng nửa GB. Nếu điều này là tầm thường so với kích cỡ tổng cơ sở dữ liệu của bạn, thì việc thiết lập autovacuum_freeze_max_age về giá trị cực đại được phép của nó là được khuyến cáo. Nếu không, hãy thiết lập nó phụ thuộc vào những gì bạn mong muốn cho phép cho lưu trữ của pg_clog. (Mặc định, 200 triệu giao dịch, tương đương khoảng 50MB lưu trữ của pg_clog).

Một nhược điểm của việc làm giảm vacuum_freeze_min_age là nó có thể làm cho VACUUM thực hiện một công việc vô ích: thay đổi XID của hàng một bảng sang FrozenXID là một việc phí thời gian nếu hàng đó bị sửa đổi ngay sau đó (vì nó có một XID mới). Vì thế việc thiết lập nên đủ lớn sao cho các hàng không bị đông cứng cho tới khi chúng có lẽ không thay đổi được hơn nữa. Một nhược điểm khác của việc làm giảm thiết lập này là các chi tiết về chính xác giao dịch nào được chèn vào hoặc được sửa đổi mà một hàng sẽ bị mất sớm hơn. Thông tin này đôi khi là thuận tiện, đặc biệt khi cố thử phân tích những gì đã sai sau một hỏng hóc cơ sở dữ liệu. Vì 2 lý do đó, việc làm giảm thiết lập này không được khuyến cáo, ngoại trừ cho các bảng tĩnh hoàn toàn.

Để theo dõi tuổi của các XID cũ nhất trong một cơ sở dữ liệu, VACUUM lưu trữ các số liệu thống kê

về XID trong các bảng hệ thống pg_class và pg_database. Đặc biệt, cột relfrozenxid của hàng pg_class của một bảng có XID cắt giảm đông cứng sẽ được đảm bảo sẽ được FrozenXID thay thế bên trong bảng đó. Tương tự, cột datfrozenxid của hàng pg_database của một cơ sở dữ liệu là một ràng buộc thấp hơn trong các XID thông thường xuất hiện trong cơ sở dữ liệu đó - chỉ là tối thiểu của các giá trị relfrozenxid theo từng bảng bên trong cơ sở dữ liệu đó. Một cách thuận tiện để kiểm tra thông tin này là thực thi các truy vấn như:

```
SELECT relname, age(relfrozenxid) FROM pg_class WHERE relkind = 'r';
SELECT datname, age(datfrozenxid) FROM pg_database;
```

Cột age đo đếm số lượng các giao dịch từ XID bị cắt giảm tới XID của giao dịch hiện hành.

VACUUM thường chỉ quét các trang từng bị sửa đổi kể từ lần hút chân không mới nhất, nhưng relfrozenxid chỉ có thể được thực hiện trước khi toàn bộ bảng được quét. Toàn bộ bảng được quét khi relfrozenxid là lớn hơn các giao dịch cũ vacuum_freeze_table_age, khi lựa chọn FREEZE của VACUUM được sử dụng, hoặc khi tất cả các trang ngẫu nhiên đòi hỏi việc hút chân không để loại bỏ các phiên bản hàng chết. Khi VACUUM quét toàn bộ bảng, sau khi age(relfrozenxid) được hoàn tất của nó sẽ là nhiều hơn một chút so với thiết lập vacuum_freeze_min_age mà đã được sử dụng (hơn với số lượng các giao dịch được bắt đầu kể từ khi VACUUM đã khởi động). Nếu không VACUUM quét toàn bộ bảng nào được đưa ra trong bảng đó cho tới khi autovacuum_freeze_max_age đạt được, thì một sự tự động hút chân không sẽ sớm bị ép thực hiện đối với bảng đó.

Nếu vì một vài lý do mà sự tự động hút chân không không dọn sạch được các XID cũ khỏi một bảng, thì hệ thống sẽ bắt đầu phát các thông điệp cảnh báo giống thế này khi các XID cũ nhất của cơ sở dữ liệu đạt tới 10 triệu giao dịch từ điểm bao quanh:

```
WARNING: database "mydb" must be vacuumed within 177009986 transactions
```

> **Gợi ý**: để tránh một sự sập cơ sở dữ liệu, hãy thực thi một lệnh VACUUM rộng khắp một cơ sở dữ liệu trong "mydb".

(Một VACUUM bằng tay) sẽ sửa vấn đề đó, như gợi ý đưa ra; nhưng lưu ý rằng VACUUM phải được một siêu người sử dụng thực hiện, nếu không thì nó sẽ không xử lý các catalog hệ thống và vì thế không có khả năng thực hiện trước datfrozenxid của cơ sở dữ liệu). Nếu các cảnh báo đó bị bỏ qua, thì hệ thống sẽ tắt và từ chối khởi động lại bất kỳ giao dịch nào một khi có ít hơn 1 triệu giao dịch còn lại cho tới điểm bao quanh:

```
ERROR: database is not accepting commands to avoid wraparound data loss in database "mydb"
```

> **Gợi ý**: Hãy dừng postmaster và sử dụng một phụ trợ (backend) đứng một mình để VACUUM trong "mydb".

Ngưỡng an toàn 1 triệu giao dịch tồn tại để cho phép người quản trị phục hồi mà không mất dữ liệu, bằng việc thực thi bằng tay các lệnh VACUUM được yêu cầu. Tuy nhiên, vì hệ thống sẽ không thực thi các lệnh một khi nó đã đi vào chế độ tắt an toàn, nên cách duy nhất để làm điều này là phải dừng máy chủ và sử dụng một phụ trợ người sử dụng duy nhất để thực thi VACUUM. Chế độ tắt sẽ không

bị một phụ trợ người sử dụng duy nhất ép buộc. Xem trang tham chiếu postgres để có các chi tiết về việc sử dụng phụ trợ người sử dụng duy nhất.

23.1.5. Daemon tự động hút chân không (Autovacuum)

PostgreSQL có một tính năng lựa chọn mà được khuyến cáo cao độ gọi là hút chân không tự động, mục đích của nó là để tự động thực thi các lệnh VACUUM và ANALYZE. Khi được kích hoạt, hút chân không tự động kiểm tra các bảng đã có một số lượng lớn các bộ dữ liệu được chèn, được cập nhật hoặc bị xóa. Các kiểm tra đó sử dụng tiện ích thu thập số liệu thống kê; vì thế, hút chân không tự động không thể được sử dụng trừ phi track_counts được thiết lập về đúng true. Trong cấu hình mặc định, việc hút chân không tự động được kích hoạt và các tham số cấu hình có liên quan được thiết lập một cách đúng phù hợp.

"Daemon hút chân không tự động (autovacuum)" thực sự cấu thành từ nhiều tiến trình. Có một tiến trình daemon nhất quán, được gọi là *trình khởi tạo hút chân không tự động*, nó có trách nhiệm khởi tạo các tiến trình làm việc của hút chân không tự động cho tất cả các cơ sở dữ liệu. Trình khởi tạo đó sẽ phân phối công việc khắp mọi thời gian, cố gắng khởi động một trình công việc (worker) trong từng cơ sở dữ liệu mỗi autovacuum_naptime giây đồng hồ. (Vì thế, nếu cài đặt có N cơ sở dữ liệu, thì một trình công việc mới sẽ được khởi tạo mỗi autovacuum_naptime/N giây). Cực đại các tiến trình của trình công việc (worker) autovacuum_max_workers sẽ được phép chạy cùng một lúc. Nếu có nhiều hơn autovacuum_max_workers cơ sở dữ liệu sẽ được xử lý, thì cơ sở dữ liệu tiếp sau sẽ được xử lý ngay khi trình công việc đầu tiên hoàn tất. Mỗi tiến trình của trình công việc sẽ kiểm tra từng bảng bên trong cơ sở dữ liệu của nó và thực thi các lệnh VACUUM và/hoặc ANALYZE như cần thiết.

Nếu vài bảng lớn tất cả trở thành hợp lệ cho việc hút chân không trong một khoảng thời gian ngắn, thì tất cả các trình công việc hút chân không tự động có thể trở nên bị chiếm bằng việc hút chân không các bảng đó trong một giai đoạn dài. Điều này có thể gây ra trong các bảng và cơ sở dữ liệu khác không đang được hút chân không cho tới khi một trình công việc trở nên sẵn sàng. Không có giới hạn về có bao nhiêu trình công việc có thể trong một cơ sở dữ liệu duy nhất, nhưng các trình công việc cố tránh công việc trùng lặp đã được các trình công việc khác thực hiện rồi. Lưu ý rằng số lượng các trình công việc đang chạy không tính tới các giới hạn max_connections hoặc uperuser_reserved_connections.

Các bảng mà giá trị relfrozenxid của nó là lớn hơn autovacuum_freeze_max_age các giao dịch cũ sẽ luôn được hút chân không. (điều này cũng áp dụng cho các bảng mà tuổi cực đại đông cứng của chúng đã bị sửa đổi thông qua các tham số lưu trữ). Nếu không, nếu số lượng các bộ dữ liệu đã lỗi thời vì VACUUM cuối cùng vượt quá "ngưỡng hút chân không", bảng đó được hút chân không. Ngưỡng hút chân không được xác định như là:

ngưỡng hút chân không = ngưỡng cơ bản hút chân không + yếu tố phạm vi hút chân không * số lượng các bộ dữ liệu (vacuum threshold = vacuum base threshold + vacuum scale factor * number of tuples)

trong đó ngưỡng cơ bản hút chân không là autovacuum_vacuum_threshold, yếu tố phạm vi hút

chân không là autovacuum_vacuum_scale_factor, và số các bộ dữ liệu là pg_class.reltuples. Số lượng các bộ dữ liệu lỗi thời có được từ trình thu thập số liệu thống kê; đó là một tính toán tương đối chính xác được từng hoạt động của UPDATE và DELETE cập nhật. (Đó chỉ là tương đối chính xác vì một vài thông tin có thể bị mất khi tải lớn). Nếu giá trị relfrozenxid của bảng lớn hơn các giao dịch cũ vacuum_freeze_table_age, thì toàn bộ bảng sẽ được quét để đông cứng các bộ dữ liệu cũ và relfrozenxid trước đó, nếu không chỉ các trang đã được sửa đổi kể từ lần hút chân không mới nhất sẽ được quét.

Để phân tích, một điều kiện tương tự được sử dụng: ngưỡng, được xác định như là:

ngưỡng phân tích = ngưỡng cơ bản phân tích + yếu tố phạm vi phân tích * số bộ dữ liệu (analyze threshold = analyze base threshold + analyze scale factor * number of tuples)

được so sánh với tổng số các bộ dữ liệu được chèn, được cập nhật hoặc bị xóa kể từ lần ANALYZE mới nhất.

Các bảng tạm không thể được hút chân không tự động truy cập. Vì thế các hoạt động hút chân không và phân tích đúng phù hợp sẽ được thực hiện thông qua phiên các lệnh SQL.

Các ngưỡng và các yếu tố phạm vi mặc định được lấy từ postgresql.conf, nhưng có khả năng ghi đè chúng trên cơ sở từng bảng một; xem Các tham số Lưu trữ để có thêm thông tin. Nếu một thiết lập từng bị thay đổi thông qua các tham số lưu trữ, thì giá trị đó được sử dụng; nếu không thì các thiết lập tổng thể sẽ được sử dụng. Xem Phần 18.9 để có thêm chi tiết về các thiết lập tổng thể.

Ngoài các giá trị và các yếu tố phạm vi ngưỡng cơ bản, có 6 tham số hút chân không tự động nữa có thể được thiết lập cho từng bảng thông qua các tham số lưu trữ. Tham số đầu tiên, autovacuum_enabled, có thể được thiết lập về sai (false) để chỉ thị daemon hút chân không tự động bỏ qua hoàn toàn bảng đặc biệt đó. Trong trường hợp này hút chân không tự động sẽ chỉ động chạm tới bảng đó nếu nó phải làm thế để ngăn chặn sự bao quanh ID giao dịch. Hai tham số khác, autovacuum_vacuum_cost_delay và autovacuum_vacuum_cost_limit, sẽ được sử dụng để thiết lập các giá trị đặc thù của bảng cho tính năng trễ hút chân không dựa vào chi phí (xem Phần 18.4.3). autovacuum_freeze_min_age, autovacuum_freeze_max_age và autovacuum_freeze_table_age sẽ được sử dụng để thiết lập các giá trị cho vacuum_freeze_min_age, autovacuum_freeze_max_age và vacuum_freeze_table_age một cách tương ứng.

Khi nhiều trình công việc đang chạy, giới hạn chi phí sẽ "được cân bằng" giữa tất cả các trình công việc đang chạy, sao cho tổng ảnh hưởng lên hệ thống là y như nhau, bất kể số lượng các trình công việc thực sự đang chạy.

23.2. Việc đánh chỉ số lại thường ngày

Trong một vài tình huống đáng xây dựng lại các chỉ số một cách định kỳ với lệnh REINDEX.

Các trang chỉ số B-tree đã trở thành rỗng hoàn toàn sẽ được cải tạo để sử dụng lại. Tuy nhiên, vẫn còn có khả năng sử dụng không gian không hiệu quả: nếu tất cả ngoại trừ một ít các khóa chỉ số trong một trang đã bị xóa, thì trang đó vẫn còn được phân bổ. Vì thế, một mẫu sử dụng trong đó hầu hết, nhưng không phải tất cả, các khóa trong từng dải rốt cuộc sẽ bị xóa sẽ thấy sử dụng không gian không tốt. Đối với các mẫu sử dụng như vậy, thì việc đánh chỉ số lại định kỳ được khuyến cáo.

Tiềm năng cho sự nở quá đáng trong các chỉ số không phải là B-tree không được nghiên cứu tốt. Là ý tưởng tốt để giám sát định kỳ kích cỡ vật lý của chỉ số đó khi sử dụng bất kỳ dạng chỉ số không phải B-tree nào.

Hơn nữa, đối với các chỉ số B-tree, một chỉ số được xây dựng tươi mới là khá nhanh hơn để truy cập hơn là một chỉ số mà đã từng được cập nhật nhiều lần vì theo logic các trang liền kề thường cũng là liền kề về vật lý trong một chỉ số được xây dựng mới. (Cân nhắc này không áp dụng cho các chỉ số không phải là B-tree). Có thể đáng đánh chỉ số lại định kỳ chỉ để cải thiện tốc độ truy cập.

23.3. Duy trì tệp lưu ký

Là ý tưởng tốt để lưu đầu ra lưu ký máy chủ cơ sở dữ liệu ở đâu đó, hơn là chỉ bỏ nó qua /dev/null. Đầu ra lưu ký là vô giá trị khi chuẩn đoán các vấn đề. Tuy nhiên, đầu ra lưu ký có xu hướng sẽ to khổng lồ (đặc biệt ở các mức dò lỗi cao hơn) nên bạn sẽ không muốn lưu nó không biết tới bao giờ. Bạn cần xoay vòng các tệp lưu ký sao cho các tệp lưu ký mới sẽ được bắt đầu và các tệp lưu ký cũ bị loại bỏ sau một khoảng thời gian hợp lý.

Nếu bạn đơn giản lệnh cho stderr của postgres vào trong một tệp, thì bạn sẽ có đầu ra lưu ký, nhưng cách duy nhất để cắt bớt tệp lưu ký là dừng và khởi động lại máy chủ. Điều này có thể chấp nhận được nếu bạn đang sử dụng PostgreSQL trong một môi trường phát triển, nhưng ít máy chủ sản xuất thấy hành vi này là chấp nhận được.

Một tiếp cận tốt hơn là gửi đầu ra stderr của máy chủ tới vài dạng chương trình xoay vòng. Có một tiện ích xoay vòng lưu ký được xây dựng sẵn, mà bạn có thể sử dụng bằng việc thiết lập tham số cấu hình logging_collector về đúng (true) trong postgresql.conf. Các tham số kiểm soát cho chương trình này được mô tả trong Phần 18.7.1. Bạn cũng có thể sử dụng tiếp cận này để nắm bắt các dữ liệu lưu ký ở định dạng CSV máy đọc được (các giá trị phân cách nhau bằng dấu phẩy).

Như một sự lựa chọn, bạn có thể thích sử dụng một chương trình xoay vòng lưu ký ngoài hơn nếu bạn có một chương trình mà bạn đang sử dụng rồi với các phần mềm máy chủ khác. Ví dụ, công cụ xoay vòng các lưu ký (rotatelogs) được đưa vào trong phân phối Apache có thể được sử dụng trong PostgreSQL. Để làm điều này, hãy đặt đường cho đầu ra stderr của máy chủ tới chương trình mong muốn đó. Nếu bạn khởi động máy chủ bằng pg_ctl, thì stderr được tái định tuyến rồi tới stdout, nên bạn chỉ cần một lệnh dẫn đường, ví dụ:

```
pg_ctl start | rotatelogs /var/log/pgsql_log 86400
```

Một tiếp cận mức sản xuất khác cho việc quản lý đầu ra lưu ký là gửi nó tới lưu ký hệ thống syslog và để syslog làm việc với sự xoay vòng tệp. Để làm việc này, hãy thiết lập tham số cấu hình log_destination về syslog (để lưu ký chỉ về syslog) trong postgresql.conf. Sau đó bạn có thể gửi một tín hiệu SIGHUP tới daemon syslog bất kỳ khi nào bạn muốn ép nó khởi động việc ghi một tệp lưu ký mới. Nếu bạn muốn tự động hóa xoay vòng lưu ký, thì chương trình xoay vòng lưu ký logrotate có thể được thiết lập cấu hình để làm việc với các tệp lưu ký từ syslog.

Tuy nhiên, trong nhiều hệ thống, syslog là rất không tin cậy, đặc biệt với các thông điệp lưu ký lớn; nó có thể cắt bớt hoặc bỏ các thông điệp chỉ khi bạn cần chúng nhất. Hơn nữa, trong Linux, syslog

sẽ đẩy từng thông điệp lên đĩa, làm hiệu năng tồi đi. (Bạn có thể sử dụng một "-" ở đầu của tên tệp trong tệp cấu hình syslog để vô hiệu hóa syncing).

Lưu ý rằng tất cả các giải pháp được mô tả ở trên quan tâm tới việc khởi động các tệp lưu ký mới trong các khoảng thời gian có thể thiết lập cấu hình được, nhưng chúng không điều khiển sự xóa các tệp lưu ký cũ, không còn hữu dụng nữa. Bạn có lẽ sẽ muốn thiết lập một bó công việc để xóa định kỳ các tệp lưu ký cũ. Một khả năng khác là hãy thiết lập cấu hình cho chương trình xoay vòng sao cho các tệp lưu ký cũ sẽ bị ghi đè một cách xoay vòng.

pgFouine[2] là một dự án bên ngoài mà thực hiện phân tích tệp lưu ký phức tạp. check_postgres[3] đưa ra cho Nagios các cảnh báo khi các thông điệp quan trọng xuất hiện trong các tệp lưu ký, cũng như dò tìm ra nhiều điều kiện quá xá khác.

2 http://pgfouine.projects.postgresql.org/
3 http://bucardo.org/wiki/Check_postgres

Chương 24. Sao lưu và phục hồi

Như với mọi điều có chứa các dữ liệu có giá trị, các cơ sở dữ liệu PostgreSQL sẽ được sao lưu thường xuyên. Trong khi thủ tục về cơ bản là đơn giản, thì điều quan trọng phải có một sự hiểu biết rõ ràng về các kỹ thuật và giả thiết nằm bên dưới.

Về cơ bản có 3 tiếp cận khác nhau cho việc sao lưu các dữ liệu PostgreSQL:

- Tạo SQL dump
- Sao lưu mức hệ thống tệp
- Lưu trữ liên tục

Mỗi tiếp cận có các điểm mạnh và yếu của riêng nó và được thảo luận trong các phần sau đây.

24.1. Tạo SQL dump

Ý tưởng đằng sau phương pháp dump này là để tạo ra một tệp văn bản với các lệnh SQL mà, khi đưa trở lại máy chủ, sẽ tái tạo cơ sở dữ liệu ở tình trạng y hệt như nó từng có vào thời điểm lúc dump. PostgreSQL đưa ra chương trình tiện ích pg_dump cho mục đích này. Sử dụng cơ bản lệnh này là:

```
pg_dump dbname > outfile
```

Như bạn thấy, pg_dump ghi kết quả của nó tới đầu ra tiêu chuẩn. Chúng ta sẽ thấy cách mà điều này có thể là hữu dụng.

pg_dump là một ứng dụng máy trạm thường xuyên của PostgreSQL (và là một ứng dụng đặc biệt thông minh). Điều này có nghĩa là bạn có thể thực hiện thủ tục sao lưu này từ bất kỳ host ở xa nào mà có sự truy cập tới cơ sở dữ liệu. Nhưng hãy nhớ rằng pg_dump không vận hành với các quyền cho phép đặc biệt. Đặc biệt, nó phải có sự truy cập đọc tới tất cả các bảng mà bạn muốn sao lưu, nên trong thực tế bạn hầu như luôn phải chạy nó như một siêu người sử dụng cơ sở dữ liệu.

Để chỉ định pg_dump máy chủ cơ sở dữ liệu nào sẽ liên hệ, hãy sử dụng các lựa chọn dòng lệnh -h host và -p port. Host mặc định là host cục bộ (local host) hoặc bất kỳ thứ gì biến môi trường PGHOST của bạn chỉ định. Tương tự, cổng mặc định được chỉ định bằng biến môi trường PGHOST hoặc, nếu không, bằng mặc định được biên dịch. (Thông thường, máy chủ sẽ bình thường có mặc định y hệt được biên dịch).

Giống như bất kỳ ứng dụng máy trạm PostgreSQL nào, mặc định pg_dump sẽ kết nối với tên người sử dụng cơ sở dữ liệu bằng với tên người sử dụng hệ điều hành hiện hành. Để ghi đè điều này, hoặc chỉ định lựa chọn -U hoặc thiết lập biến môi trường PGUSER. Hãy nhớ rằng các kết nối pg_dump tuân theo các cơ chế xác thực máy trạm thông thường (chúng được mô tả trong Chương 19).

Một ưu thế quan trọng của pg_dump so với các phương pháp khác được mô tả sau là đầu ra của pg_dump thường có thể được tải lại vào các phiên bản mới hơn của PostgreSQL, trong khi đó các sao lưu mức tệp và việc lưu trữ liên tục đều cực kỳ là đặc thù theo phiên bản máy chủ. pg_dump cũng chỉ là phương pháp sẽ làm việc khi biến đổi một cơ sở dữ liệu thành một kiến trúc máy khác,

như việc đi từ một máy chủ 32 bit sang 64 bit vậy.

Các dump được pg_dump tạo ra là ổn định nhất quán trong nội bộ, nghĩa là, dump đại diện cho một bản chụp lại cơ sở dữ liệu ở thời điểm pg_dump đã bắt đầu chạy. pg_dump không khóa các hoạt động khác trong cơ sở dữ liệu trong khi nó đang làm việc. (Các ngoại lệ là các hoạt động mà cần vận hành với một khóa hoàn toàn, như hầu hết các dạng ALTER TABLE).

> **Quan trọng**: Nếu sơ đồ cơ sở dữ liệu của bạn dựa vào các OID (ví dụ, các khóa ngoại) thì bạn phải lệnh cho pg_dump để cũng dump các OID. Để làm điều này, hãy sử dụng lựa chọn -o command-line.

24.1.1. Phục hồi dump

Các tệp văn bản được pg_dump tạo ra có ý định sẽ là đọc được đối với chương trình psql. Mẫu lệnh chung để phục hồi một dump là

```
psql dbname < infile
```

trong đó infile là đầu ra tệp của lệnh pg_dump. Tên cơ sở dữ liệu dbname sẽ không được lệnh này tạo ra, nên bạn phải tự tạo nó từ template0 trước khi thực thi psql (như, với createdb -T template0 dbname). psql hỗ trợ các lựa chọn tương tự với pg_dump cho việc chỉ định máy chủ cơ sở dữ liệu để kết nối tới và tên người sử dụng để sử dụng. Xem trang tham chiếu psql để có thêm thông tin.

Trước khi phục hồi một SQL dump, tất cả những người sử dụng sở hữu các đối tượng hoặc từng được trao các quyền cho phép các đối tượng trong cơ sở dữ liệu được dump phải tồn tại rồi. Nếu họ không tồn tại, thì sự phục hồi sẽ không tái tạo được các đối tượng bằng các quyền cho phép và/hoặc quyền sở hữu gốc ban đầu. (Đôi khi điều này là những gì bạn bạn muốn, nhưng thường là không).

Mặc định, script psql sẽ tiếp tục thực thi sau khi một lỗi SQl xảy ra. Bạn có thể muốn chạy psql với biến ON_ERROR_STOP được thiết lập để sửa hành vi đó và để psql thoát ra với một tình trạng thoát ra là 3 nếu một lỗi SQL xảy ra:

```
psql --set ON_ERROR_STOP=on dbname < infile
```

Dù là cách nào, thì bạn cũng sẽ chỉ có một cơ sở dữ liệu được phục hồi một phần. Như một sự lựa chọn, bạn có thể chỉ định rằng toàn bộ sự dump sẽ được phục hồi như một giao dịch duy nhất, nên sự phục hồi hoặc được hoàn thành đầy đủ hoặc quay ngược lại (roll back) đầy đủ. Chế độ này có thể được chỉ định bằng việc tryền các lựa chọn dòng lệnh -1 hoặc --single-transaction tới psql. Khi sử dụng chế độ này, hãy nhận thức rằng thậm chí một lỗi nhỏ cũng có thể làm cho một sự phục hồi quay lại dù sự phục hồi đó đã chạy nhiều giờ đồng hồ. Tuy nhiên, điều đó có thể vẫn được ưa chuộng cho việc làm sạch bằng tay 1 cơ sở dữ liệu phức tạp sau một dump được phục hồi một phần.

Khả năng của pg_dump và psql ghi vào hoặc đọc từ các đường dẫn (pipe) làm cho nó có khả năng dump một cơ sở dữ liệu trực tiếp từ máy chủ này tới máy chủ khác, ví dụ:

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

> **Quan trọng**: Các dump được pg_dump tạo ra là tương đối với template0. Điều này có nghĩa là bất kỳ ngôn ngữ, thủ tục, ... nào được thêm vào qua template1 cũng sẽ bị dump bằng

pg_dump. Kết quả là, khi phục hồi, nếu bạn sử dụng một template1 được tùy biến, thì bạn phải tạo cơ sở dữ liệu rỗng đó từ template0, như trong ví dụ ở trên.

Sau khi phục hồi một sao lưu, là khôn ngoan để chạy ANALYZE trên từng cơ sở dữ liệu sao cho trình tối ưu hóa truy vấn có các số liệu thống kê hữu dụng; xem Phần 23.1.3 và Phần 23.1.5 để có thêm thông tin. Để có thêm tư vấn về cách tải lượng lớn các dữ liệu vào PostgreSQL một cách có hiệu quả, hãy tham chiếu tới Phần 14.4.

24.1.2. Sử dụng pg_dumpall

pg_dump sẽ chỉ dump một cơ sở dữ liệu duy nhất tại một thời điểm, và nó không dump thông tin về các vai trò hoặc các không gian bảng (vì chúng là rộng khắp bó thay vì theo từng cơ sở dữ liệu). Để hỗ trợ việc dump thuận thiện đối với toàn bộ các nội dung của một bó cơ sở dữ liệu, chương trình pg_dumpall được cung cấp. pg_dumpall sao lưu từng cơ sở dữ liệu trong một bó được đưa ra, và cũng giữ lại các dữ liệu rộng khắp bó đó như các định nghĩa vai trò và không gian bảng. Sử dụng cơ bản của lệnh này là:

```
pg_dumpall > outfile
```

Kết quả của dump có thể được phục hồi bằng psql:

```
psql -f infile postgres
```

(Thực sự, bạn có thể chỉ định bất kỳ tên cơ sở dữ liệu nào để bắt đầu từ đó, nhưng nếu bạn đang tải vào một bó rỗng thì sau đó postgres thường sẽ được sử dụng). Luôn cần thiết phải có sự truy cập của siêu người sử dụng cơ sở dữ liệu khi phục hồi một dump bằng pg_dumpall, khi điều đó được yêu cầu để khôi phục lại thông tin vai trò và không gian bảng. Nếu bạn sử dụng không gian bảng, hãy chắc chắn là các đường dẫn không gian bảng trong dump là đúng phù hợp cho cài đặt mới.

pg_dumpall làm việc bằng cách phát các lệnh để tái tạo các vai trò, các không gian bảng, và các cơ sở dữ liệu rỗng, sau đó triệu gọi pg_dump cho từng cơ sở dữ liệu. Điều này có nghĩa là trong khi từng cơ sở dữ liệu sẽ là ổn định nhất quán nội bộ, thì các bản chụp hình các cơ sở dữ liệu khác nhau sẽ không thật chính xác đồng bộ.

24.1.3. Điều khiển các cơ sở dữ liệu lớn

Một vài hệ điều hành có các giới hạn tối đa về kích cỡ tệp gây ra các vấn đề khi tạo các tệp đầu ra pg_dump lớn. May thay, pg_dump có thể ghi tới đầu ra tiêu chuẩn, nên bạn có thể sử dụng các công cụ Unix tiêu chuẩn để khắc phục vấn đề tiềm tàng này. Có vài phương pháp có thể:

Sử dụng các dump được nén. Bạn có thể sử dụng chương trình nén ưa thích, ví dụ gzip:

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

Tải lại bằng:

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

Sử dụng split. Lệnh split cho phép bạn chia đầu ra thành các tệp nhỏ hơn chấp nhận được về kích cỡ đối với hệ thống tệp nằm bên dưới. Ví dụ, để tạo một đoạn 1 MB:

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

Tải lại với:

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

Sử dụng định dạng dump tùy thích của pg_dump. Nếu PostgreSQL từng được xây dựng trên một hệ thống với thư viện nén zlib được cài đặt, thì định dạng dump tùy biến sẽ nén dữ liệu khi nó ghi dữ liệu tới tệp đầu ra. Điều này sẽ tạo ra các kích cỡ tệp dump tương tự như việc sử dụng gzip, nhưng nó có ưu thế bổ sung là các bảng có thể được phục hồi một cách có lựa chọn. Lệnh sau đây sẽ dump một cơ sở dữ liệu bằng việc sử dụng định dạng dump tùy biến:

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

Một dump định dạng tùy biến không phải là một script đối với psql, mà thay vào đó phải được phục hồi bằng pg_restore, ví dụ:

```
pg_restore -d dbname filename
```

Xem các trang tham chiếu pg_dump và pg_restore để có các chi tiết.

Đối với các cơ sở dữ liệu rất lớn, bạn có thể cần kết hợp split với một trong 2 tiếp cận khác.

24.2. Sao lưu mức hệ thống tệp

Một chiến lược sao lưu có thể lựa chọn là sao chép trực tiếp các tệp mà PostgreSQL sử dụng để lưu trữ dữ liệu trong cơ sở dữ liệu; Phần 17.2 giải thích đâu là nơi các tệp đó được đặt. Bạn có thể sử dụng bất kỳ phương pháp nào bạn ưa thích để thực hiện các sao lưu hệ thống tệp; ví dụ:

```
tar -cf backup.tar /usr/local/pgsql/data
```

Tuy nhiên, có 2 hạn chế làm cho phương pháp này không thực tế, hoặc ít nhất là yếm thế so với phương pháp pg_dump:

1. Máy chủ cơ sở dữ liệu phải được tắt để có được một sao lưu sử dụng được. Các biện pháp nửa vời như việc vô hiệu hóa tất cả các kết nối sẽ không làm việc (một phần vì tar và các công cụ tương tự không chụp được một bản chụp nguyên tử tình trạng hệ thống tệp, mà cũng vì việc nhớ tạm nội bộ bên trong máy chủ). Thông tin về việc dừng máy chủ có thể thấy trong Phần 17.5. Không cần nói, bạn cũng cần tắt máy chủ trước khi phục hồi dữ liệu.
2. Nếu bạn có sự đào bới trong các chi tiết hình thức hệ thống tệp của cơ sở dữ liệu, thì bạn có thể có ý muốn thử sao lưu hoặc phục hồi chỉ các bảng hoặc cơ sở dữ liệu cá thể nhất định nào đó từ các tệp hoặc thư mục tương ứng. Điều này sẽ không làm việc được vì thông tin có trong các tệp đó là không sử dụng được mà không có các tệp lưu ký được thực hiện, pg_clog/*, chúng chứa tình trạng thực hiện của tất cả các giao dịch.

Tệp của một bảng chỉ sử dụng được với thông tin này. Tất nhiên cũng có khả năng phục hồi chỉ một bảng và các dữ liệu pg_clog có liên quan vì điều đó có thể trả về tất cả các bảng khác trong bó cơ sở dữ liệu là vô dụng. Vì thế các sao lưu hệ thống tệp chỉ làm việc đối với sự sao lưu và phục hồi hoàn chỉnh của toàn bộ một bó cơ sở dữ liệu. Một tiếp cận có thể chọn để sao lưu hệ thống tệp là thực hiện một "bản chụp ổn định nhất quán" thư mục dữ liệu, nếu hệ thống tệp hỗ trợ chức năng đó (và bạn đang có thiện chí tin tưởng rằng nó được triển khai một cách đúng đắn). Thủ tục điển hình là thực hiện một "bản chụp đông cứng" dung lượng có trong cơ sở dữ liệu đó, sau đó sao chép toàn bộ thư mục dữ liệu (không chỉ các phần, xem ở trên) từ bản chụp đó tới một thiết bị sao lưu, rồi phát hành bản chụp đông cứng đó. Điều này sẽ làm việc thậm chí trong khi máy chủ cơ sở dữ liệu đang

chạy. Tuy nhiên, một sao lưu được tạo ra theo cách này cứu các tệp cơ sở dữ liệu ở một tình trạng dường như là máy chủ cơ sở dữ liệu từng bị tắt không đúng; vì thế, khi bạn khởi động máy chủ cơ sở dữ liệu trong các dữ liệu được sao lưu, thì nó sẽ nghĩ cài đặt máy chủ trước đó bị hỏng và sẽ chơi lại lưu ký WAL. Đây không phải là một vấn đề; chỉ phải nhận thức về nó (và chắc chắn đưa các tệp WAL vào sao lưu của bạn).

Nếu cơ sở dữ liệu của bạn được trải ra nhiều hệ thống tệp, có thể không có bất kỳ cách gì để có được các bản chụp đông cứng chính xác cùng một lúc của tất cả các dung lượng. Ví dụ, nếu các tệp dữ liệu và lưu ký WAL của bạn là trong các đĩa khác nhau, hoặc nếu các không gian bảng là ở trong các hệ thống tệp khác nhau, thì có lẽ là không có khả năng để sử dụng sao lưu bản chụp vì các bản chụp phải là đồng thời. Hãy đọc tài liệu hệ thống tệp của bạn rất cẩn thận trước khi tin vào kỹ thuật bản chụp ổn định nhất quán trong các tình huống như vậy.

Nếu các bản chụp đồng thời là không có khả năng, thì một lựa chọn là tắt máy chủ cơ sở dữ liệu đủ lâu để thiết lập tất cả các bản chụp đông cứng. Một lựa chọn khác là thực hiện một sao lưu cơ bản của lưu trữ liên tục (Phần 24.3.2) vì các sao lưu như vậy là miễn nhiễm với những thay đổi của hệ thống tệp trong quá trình sao lưu. Điều này đòi hỏi việc kích hoạt lưu trữ liên tục chỉ trong quá trình sao lưu; phục hồi được thực hiện bằng việc sử dụng phục hồi lưu trữ liên tục (Phần 24.3.3).

Một lựa chọn khác là sử dụng rsync để thực hiện một sao lưu hệ thống tệp. Điều này được thực hiện bằng việc chạy lần đầu rsync trong khi máy chủ cơ sở dữ liệu đang chạy, sau đó tắt máy chủ cơ sở dữ liệu đủ lâu để thực hiện một sao lưu rsync thứ 2. rsync lần 2 sẽ nhanh hơn nhiều so với lần đầu, vì nó có khá ít dữ liệu phải truyền, và kết quả cuối cùng sẽ là ổn định nhất quán vì máy chủ đã được tắt. Phương pháp này cho phép một sao lưu hệ thống tệp được thực hiện với thời gian chết tối thiểu.

Lưu ý rằng một sao lưu hệ thống tệp điển hình sẽ lớn hơn so với một dump SQL. (pg_dump không cần phải dump các nội dung của các chỉ số, ví dụ, chỉ là các lệnh sẽ tái tạo lại chúng). Tuy nhiên, thực hiện một sao lưu hệ thống tệp có thể là nhanh hơn.

24.3. Lưu trữ liên tục và phục hồi theo điểm đúng lúc (PITR)

Ở mọi lúc, PostgreSQL duy trì một lưu ký trước khi ghi - WAL (Write Ahead Log) trong thư mục pg_xlog/ của thư mục dữ liệu bó. Lưu ký đó ghi lại từng sự thay đổi được thực hiện đối với các tệp dữ liệu của cơ sở dữ liệu đó. Lưu ký này tồn tại trước hết vì các mục đích an toàn với sập hỏng: nếu hệ thống hỏng, thì cơ sở dữ liệu có thể được phục hồi về trạng thái ổn định bằng việc "chơi lại" các khoản đầu vào lưu ký được thực hiện kể từ điểm kiểm tra mới nhất. Tuy nhiên, sự tồn tại của lưu ký đó làm cho nó có khả năng sử dụng một chiến lược thứ 3 cho việc sao lưu các cơ sở dữ liệu: chúng ta có thể kết hợp một sao lưu mức hệ thống tệp với sao lưu các tệp WAL. Nếu sự phục hồi là cần thiết, chúng ta sẽ phục hồi sao lưu hệ thống tệp và sau đó chơi lại từ các tệp WAL được sao lưu để mang hệ thống tới một tình trạng hiện hành. Tiếp cận này là phức tạp hơn để quản trị so với các tiếp cận trước đó, nhưng nó có một vài lợi ích đáng kể:

- Chúng ta không cần một sao lưu hệ thống tệp ổn định nhất quán tuyệt hảo như là điểm khởi đầu. Bất kỳ sự không ổn định nhất quán nội bộ nào trong sao lưu cũng sẽ được sửa cho đúng bằng sự chơi lại lưu ký (điều này không khác đáng kể với những gì xảy ra trong quá trình

phục hồi sự sập hỏng). Vì thế chúng ta không cần khả năng bản chụp hệ thống tệp, chỉ cần tar hoặc một công cụ lưu trữ tương tự.

- Vì chúng ta có thể kết hợp một tuần tự dài vô hạn các tệp WAL để chơi lại, nên sao lưu liên tục có thể đạt được đơn giản bằng việc liên tục lưu trữ các tệp WAL. Điều này là đặc biệt có giá trị cho các cơ sở dữ liệu lớn, nơi mà có thể không thuận tiện để thực hiện một sao lưu đầy đủ một cách thường xuyên.
- Không cần thiết phải chơi lại các khoản đầu vào WAL bằng mọi cách tới cùng. Chúng ta có thể dừng chơi lại ở bất kỳ điểm nào và có một bản chụp ổn định nhất quán của cơ sở dữ liệu như nó từng ở thời điểm đó. Vì thế, kỹ thuật này hỗ trợ cho sự phục hồi đúng thời điểm: có khả năng phục hồi cơ sở dữ liệu về tình trạng của nó ở bất kỳ thời điểm nào kể từ khi sao lưu cơ bản của bạn đã được thực hiện.
- Nếu chúng ta liên tục đưa các loạt tệp WAL tới máy tính khác từng được tải lên với cùng y hệt tệp sao lưu cơ bản, thì chúng ta có một hệ thống dự phòng ấm: ở bất kỳ thời điểm nào chúng ta cũng có thể mang máy thứ 2 tới và nó sẽ có một bản sao gần với hiện hành của cơ sở dữ liệu đó.

> **Lưu ý**: pg_dump và pg_dumpall không tạo ra các sao lưu mức hệ thống tệp và không thể được sử dụng như một phần của một giải pháp lưu trữ liên tục. Các dump như vậy là logic và không có đủ thông tin để được sự chơi lại của WAL sử dụng.

Như với kỹ thuật sao lưu hệ thống tệp thông thường, phương pháp này chỉ có thể hỗ trợ sự phục hồi toàn bộ một bó cơ sở dữ liệu, chứ không một phần của nó. Hơn nữa, nó đòi hỏi nhiều chỗ lưu trữ: sự sao lưu cơ bản có thể là đồ sộ, và một hệ thống bận rộn sẽ sinh ra nhiều MB giao thông WAL mà phải được lưu trữ. Hơn nữa, đây là kỹ thuật sao lưu được ưa thích trong nhiều tình huống nơi mà độ tin cậy cao là cần thiết.

Để phục hồi thành công bằng việc sử dụng lưu trữ liên tục (còn được gọi là "sao lưu trực tuyến" từ nhiều nhà bán hàng cơ sở dữ liệu), bạn cần một sự tuần tự liên tục các tệp WAL được lưu trữ mà nới rộng ngược lại ít nhất là ở thời điểm khởi động sao lưu của bạn. Vì thế để bắt đầu, bạn nên thiết lập và kiểm thử thủ tục của bạn cho việc lưu trữ các tệp WAL trước khi bạn thực hiện sao lưu cơ bản đầu tiên của bạn. Tương tự, chúng ta trước hết thảo luận các cơ chế lưu trữ các tệp WAL.

24.3.1. Thiết lập lưu trữ WAL

Theo nghĩa trừu tượng, một hệ thống PostgreSQL đang chạy tạo ra một sự tuần tự dài vô định các bản ghi WAL. Hệ thống, về mặt vật lý, chia sự tuần tự này thành các tệp phân đoạn WAL, chúng thường là các mẩu 16MB (dù kích thước phân đoạn đó có thể được sửa đổi khi xây dựng PostgreSQL). Các tệp phân đoạn đó được trao các tên số mà phản ánh vị trí của chúng trong tuần tự trừu tượng WAL. Khi không sử dụng lưu trữ WAL, hệ thống thường chỉ tạo ít tệp của một phân đoạn và sau đó "tái chế" chúng bằng việc đổi tên các tệp phân đoạn không còn cần thiết nữa thành các số phân đoạn cao hơn. Được giả thiết rằng các tệp phân đoạn mà các nội dung của chúng ở trước điểm kiểm tra lần mới nhất là không còn được quan tâm tới nữa và có thể được tái chế lại.

Khi lưu trữ dữ liệu WAL, chúng ta cần nắm lấy các nội dung của từng tệp phân đoạn một khi nó

được điền, và lưu các dữ liệu đó ở đâu đó trước khi tệp phân đoạn được tái tạo lại để sử dụng lại. Phụ thuộc vào ứng dụng và phần cứng có sẵn, có thể có nhiều cách khác nhau để "lưu các dữ liệu ở đâu đó": chúng ta có thể sao chép các tệp phân đoạn tới một thư mục được định dạng NFS trong một máy tính khác, ghi chúng vào một đầu băng từ (đảm bảo rằng bạn có cách để nhận diện tên gốc ban đầu của từng tệp), hoặc bó chúng vào cùng nhau và đốt chúng vào trong các đĩa CD, hoặc thứ gì đó hoàn toàn khác. Để đưa ra cho người quản trị cơ sở dữ liệu sự mềm dẻo, PostgreSQL cố gắng không thực hiện bất kỳ giả định nào và cách thức việc lưu trữ đó sẽ được thực hiện. Thay vào đó, PostgreSQL cho phép người quản trị chỉ định một lệnh shell sẽ được thực thi để sao chép một tệp phân đoạn hoàn chỉnh tới bất kỳ chỗ nào nó cần phải tới. Lệnh đó có thể đơn giản như cp, hoặc nó có thể triệu gọi một script shell phức tạp - cái đó là tùy vào bạn.

Để kích hoạt việc lưu trữ WAL, hãy thiết lập tham số cấu hình wal_level về archive (hoặc hot_standby), archive_mode về bật (on), và chỉ định lệnh shell đó để sử dụng trong tham số cấu hình archive_command. Trong thực tế các thiết lập đó sẽ luôn được đặt trong tệp postgresql.conf. Trong archive_command, %p được thay thế bằng tên đường dẫn của tệp đó để lưu trữ, trong khi %f được thay thế chỉ bằng tên tệp đó. (Tên đường dẫn là tương đối với thư mục làm việc hiện hành, nghĩa là, thư mục dữ liệu của bó đó). Hãy sử dụng %% nếu bạn cần nhúng một ký tự % thực sự vào trong lệnh đó. Lệnh hữu dụng đơn giản nhất là thứ gì đó giống như sau:

```
archive_command = 'test ! -f /mnt/server/archivedir/%f && cp %p /mnt/server/archivedir/%f'
archive_command = 'copy "%p" "C:\\server\\archivedir\\%f"' # Windows
```

nó sẽ sao chép các phân đoạn WAL có khả năng lưu trữ được vào thư mục /mnt/server/archivedir. (Đây là một ví dụ, không phải một khuyến cáo, và có thể không làm việc trong tất cả các nền tảng). Sau khi các tham số %p và %f đã được thay thế, lệnh thực sự được thực thi có thể giống như sau:

```
test ! -f /mnt/server/archivedir/00000001000000A900000065 && cp
pg_xlog/00000001000000A900000065
```

Một lệnh tương tự sẽ được sinh ra cho từng tệp mới sẽ được lưu trữ.

Lệnh lưu trữ sẽ được thực thi dưới quyền sở hữu của người sử dụng y hệt mà máy chủ PostgreSQL đang chạy. Vì một loạt các tệp WAL đang được lưu trữ có chứa mọi điều có hiệu lực trong cơ sở dữ liệu của bạn, nên bạn sẽ muốn chắc chắn rằng các dữ liệu được lưu trữ đó được bảo vệ khỏi các con mắt soi mói; ví dụ, lưu trữ trong một thư mục không có truy cập đọc của nhóm hoặc toàn thể.

Điều quan trọng là lệnh lưu trữ archive trả về tình trạng thoát ra zero nếu và chỉ nếu nó thành công. Dựa vào việc có được kết quả zero, PostgreSQL sẽ giả thiết rằng tệp đó từng được lưu trữ thành công, và sẽ loại bỏ hoặc tái tạo lại nó. Tuy nhiên, một tình trạng zero nói cho PostgreSQL rằng tệp đó từng không được lưu trữ; nó sẽ cố thử một lần nữa một cách định kỳ cho tới khi nó thành công.

Lệnh lưu trữ archive thường sẽ được thiết kế để khước từ ghi đè bất kỳ tệp lưu trữ tồn tại nào trước đó. Đây là một tính năng an toàn quan trọng để giữ lại tính toàn vẹn lưu trữ của bạn trong trường hợp lỗi của người quản trị (như việc gửi đầu ra của 2 máy chủ khác nhau tới cùng thư mục lưu trữ y hệt nhau).

Được khuyến cáo kiểm thử lệnh lưu trữ được đề xuất của bạn để đảm bảo rằng nó quả thực không

ghi đè một tệp đang tồn tại, và rằng nó trả về tình trạng không phải zero trong trường hợp này. Lệnh ví dụ ở trên cho Unix đảm bảo cho điều này bằng việc đưa vào một bước kiểm thử test tách biệt. Trong một vài nền tảng Unix, cp có các chuyển mạch như -i có thể được sử dụng để làm điều y hệt ít dài dòng hơn, nhưng bạn không nên dựa vào điều đó mà không có việc kiểm tra xem tình trạng thoát ra đúng được trả về ra sao. (Đặc biệt, GNU cp sẽ trả về tình trạng zero khi -i được sử dụng và tệp đích tồn tại rồi, điều đó là hành vi không mong muốn).

Trong quá trình thiết kế thiết lập lưu trữ của bạn, hãy cân nhắc điều gì sẽ xảy ra nếu lệnh lưu trữ hỏng lặp đi lặp lại vì một vài khía cạnh đòi hỏi sự can thiệp của người vận hành hoặc sự lưu trữ chạy hết không gian. Ví dụ, điều này có thể xảy ra nếu bạn ghi vào băng từ mà không có một trình tự động thay đổi (autochanger); khi băng từ đầy, không gì hơn có thể được lưu trữ cho tới khi băng đó được làm sạch. Bạn nên chắc chắn rằng bất kỳ điều kiện hoặc yêu cầu lỗi nào đối với một người vận hành cũng được báo cáo đúng phù hợp sao cho tình huống đó có thể được giải quyết một cách nhanh chóng hợp lý. Thư mục pg_xlog/ tiếp được điền đầy bằng các tệp phân đoạn WAL cho tới khi tình huống đó được giải quyết. (Nếu hệ thống tệp có chứa pg_xlog/ được điền đầy, thì PostgreSQL sẽ thực hiện một tắt máy PANIC. Không giao dịch được thực hiện nào sẽ bị mất, nhưng cơ sở dữ liệu vẫn sẽ không làm việc cho tới khi bạn giải phóng một vài không gian).

Tốc độ của lệnh lưu trữ là không quan trọng miễn là nó có thể giữ được tỷ lệ trung bình mà ở đó máy chủ của bạn sinh ra các dữ liệu WAL. Hoạt động bình thường tiếp tục thậm chí nếu tiến trình lưu trữ chậm một chút đằng sau. Nếu việc lưu trữ ở lại sau đáng kể, thì điều này sẽ làm gia tăng lượng dữ liệu có thể bị mất trong trường hợp một thảm họa. Nó cũng có nghĩa là thư mục pg_xlog/ sẽ có số lượng lớn các tệp phân đoạn còn chưa được lưu trữ, rốt cuộc chúng có thể vượt quá không gian đĩa có sẵn đó. Bạn được khuyến cáo theo dõi tiến trình lưu trữ để đảm bảo rằng nó đang làm việc như bạn muốn.

Trong khi viết lệnh lưu trữ của bạn, bạn nên giả thiết rằng các tên tệp sẽ được lưu trữ có thể tới 64 ký tự dài và có thể có bất kỳ sự kết hợp nào các ký tự ASCII, chữ số và dấu chấm. Không nhất thiết phải giữ lại đường dẫn tương đối gốc ban đầu (%p) nhưng là cần thiết để giữ lại tên tệp đó (%f).

Lưu ý rằng dù việc lưu trữ WAL sẽ cho phép bạn phục hồi bất kỳ sửa đổi nào được thực hiện cho dữ liệu đó trong cơ sở dữ liệu PostgreSQL của bạn, thì nó sẽ không phục hồi các thay đổi được thực hiện đối với các tệp cấu hình (đó là, postgresql.conf, pg_hba.conf và pg_ident.conf), vì chúng sẽ được sửa đổi bằng tay hơn là thông qua các hoạt động SQL. Bạn có thể muốn giữ các tệp cấu hình ở vị trí sẽ được các thủ tục sao lưu hệ thống tệp thông thường của bạn sao lưu. Xem phần 12.8 để biết cách tái định vị các tệp cấu hình.

Lệnh lưu trữ chỉ được triệu gọi trong các phân đoạn WAL hoàn chỉnh. Vì thế, nếu máy chủ của bạn sinh ra chỉ ít giao thức WAL (hoặc có các giai đoạn chậm chạp nơi mà nó làm thế), thì có thể có một sự trễ dài lâu giữa sự hoàn tất một giao dịch và việc ghi lại an toàn của nó trong kho lưu trữ. Để đặt ra một giới hạn về cách các dữ liệu chưa được lưu trữ có thể là, bạn có thể thiết lập archive_timeout để ép máy chủ chuyển sang một tệp phân đoạn WAL mới ít nhất một cách thường xuyên. Lưu ý là các tệp được lưu trữ sẽ được lưu trữ sớm vì một sự chuyển bị ép buộc vẫn có độ dài y hệt như các tệp hoàn toàn đầy đủ. Chính vì thế sẽ là không khôn ngoan để thiết lập một archive_timeout rất ngắn

- nó sẽ làm sưng phồng kho lưu trữ của bạn. Các thiết lập archive_timeout 1 phút hoặc khoảng đó thường là hợp lý.

Hơn nữa, bạn có thể ép chuyển đổi một phân đoạn bằng tay với pg_switch_xlog nếu bạn muốn đảm bảo rằng một giao dịch vừa được kết thúc sẽ được lưu trữ càng sớm càng tốt. Các hàm tiện ích khác có liên quan tới quản lý WAL được liệt kê trong Bảng 9-56.

Khi wal_level là minimal thì một vài lệnh SQL được tối ưu hóa để tránh việc ghi lưu ký WAL, như được mô tả trong Phần 14.4.7. Nếu việc lưu trữ hoặc nhân bản dòng đã được bật trong quá trình thực thi một trong các lệnh đó, thì WAL có thể không có đủ thông tin để phục hồi lưu trữ. (Phục hồi sự sập hỏng sẽ không có tác dụng). Vì lý do này, wal_level chỉ có thể bị thay đổi khi máy chủ khởi động. Tuy nhiên, archive_command có thể bị thay đổi với một tải tệp cấu hình. Nếu bạn muốn dừng tạm thời việc lưu trữ, một cách để làm là thiết lập archive_command về chuỗi rỗng (" "). Điều này sẽ làm cho các tệp WAL tích cóp lại trong pg_xlog/ cho tới khi một archive_command làm việc được thiết lập.

24.3.2. Thực hiện một sao lưu cơ bản

Thủ tục cho việc thực hiện một sao lưu cơ bản là khá đơn giản:

1. Đảm bảo rằng việc lưu trữ WAL được kích hoạt và làm việc.
2. Kết nối tới cơ sở dữ liệu như một siêu người sử dụng và đưa ra lệnh:
   ```
   SELECT pg_start_backup('label');
   ```
   trong đó label là bất kỳ chuỗi nào bạn muốn sử dụng để nhận diện độc nhất hoạt động sao lưu này. (Một thực tiễn tốt là sử dụng đường dẫn đầy đủ nơi mà bạn định đặt tệp dump sao lưu). pg_start_backup tạo một tệp nhãn sao lưu, gọi là backup_label, trong thư mục của bó với thông tin về sao lưu của bạn, bao gồm thời điểm khởi đầu và chuỗi nhãn.

   Không là vấn đề cơ sở dữ liệu nào trong bó đó bạn kết nối tới để đưa ra lệnh này. Bạn có thể bỏ qua kết quả được hàm đó trả về; nhưng nếu nó đưa ra một lỗi, hãy làm việc với nó trước khi xử lý tiếp.

   Mặc định, pg_start_backup có thể mất nhiều thời gian để hoàn tất. Điều này là vì nó thực hiện một điểm kiểm tra, và I/O được yêu cầu cho điểm kiểm tra đó sẽ được lan truyền qua một giai đoạn thời gian đáng kể, mặc định là một nửa khoảng thời gian điểm kiểm tra nội bộ (xem tham số cấu hình checkpoint_completion_target). Đây thường là những gì bạn muốn, vì nó làm giảm tối thiểu ảnh hưởng lên việc xử lý truy vấn. Nếu bạn muốn khởi động sao lưu càng sớm càng tốt, hãy sử dụng:
   ```
   SELECT pg_start_backup('label', true);
   ```
   Điều này ép điểm kiểm tra sẽ được thực hiện càng nhanh càng tốt.
3. Thực hiện sao lưu, bằng việc sử dụng bất kỳ công cụ sao lưu hệ thống tệp thuận tiện nào như tar hoặc cpio (không pg_dump hoặc pg_dumpall). Là không cần thiết hoặc không mong muốn để dừng hoạt động bình thường của cơ sở dữ liệu khi bạn thực hiện điều này.
4. Một lần nữa kết nối tới cơ sở dữ liệu như là một siêu người sử dụng, và đưa ra lệnh
   ```
   SELECT pg_stop_backup();
   ```

Điều này kết thúc chế độ sao lưu và thực hiện một sự chuyển tự động sang phân đoạn WAL tiếp theo. Lý do cho sự chuyển là để dàn xếp cho tệp phân đoạn WAL mới nhất được ghi trong khoảng thời gian sao lưu sẽ sẵn sàng để lưu trữ.

5. Một khi các tệp phân đoạn WAL hoạt động trong khi sao lưu được lưu trữ, bạn sẽ làm xong. Tệp đó được nhận diện bằng kết quả của pg_stop_backup là phân đoạn mới nhất được yêu cầu để tạo nên một tập hợp hoàn chỉnh các tệp sao lưu. Nếu archive_mode được kích hoạt, thì pg_stop_backup không trả về cho tới khi phân đoạn mới nhất được lưu trữ. Việc lưu trữ các tệp đó xảy ra một cách tự động vì bạn đã thiết lập cấu hình rồi cho archive_command. Trong hầu hết các trường hợp điều này xảy ra nhanh chóng, nhưng bạn được khuyến cáo phải giám sát hệ thống lưu trữ của bạn để đảm bảo không có các chậm trễ. Nếu tiến trình lưu trữ đó bị chậm vì những hỏng hóc của lệnh lưu trữ, thì nó sẽ vẫn cố thử lại cho tới khi lưu trữ đó thành công và sự sao lưu được hoàn tất. Nếu bạn muốn đặt một giới hạn thời gian lên sự thực thi của pg_stop_backup, hãy thiết lập một giá trị cho statement_timeout đúng phù hợp.

Một vài công cụ sao lưu hệ thống tệp phát ra các cảnh báo hoặc lỗi nếu các tệp mà chúng đang cố sao chép thay đổi trong khi xử lý sao chép đó. Khi tiến hành một sao lưu cơ bản của một cơ sở dữ liệu đang hoạt động, tình huống này là thông thường và không phải là một lỗi. Tuy nhiên, bạn cần đảm bảo rằng bạn có thể phân biệt được các kêu ca dạng này với các lỗi thực sự. Ví dụ, vài phiên bản của rsync trả về một mã thoát ra tách biệt cho "các tệp nguồn bị biến mất", và bạn có thể ghi một script trình điều khiển để chấp nhận mã thoát ra này như một trường hợp không phải là lỗi. Hơn nữa, một vài phiên bản của GNU tar trả về một mã lỗi phân biệt được với một lỗi chí mạng (fatal) nếu một tệp từng bị cắt bớt trong khi tar đang sao chép nó. May thay các phiên bản GNU tar 1.16 và sau này thoát với 1 nếu một tệp đã được thay đổi trong quá trình sao lưu, và 2 cho các lỗi khác.

Không nhất thiết phải lo lắng về lượng thời gian đã trôi qua giữa pg_start_backup và sự khởi đầu của sao lưu thực sự, cũng không giữa sự kết thúc sao lưu và pg_stop_backup; ít phút chậm trễ sẽ không làm hại điều gì. (Tuy nhiên, nếu bạn thường chạy máy chủ với full_page_writes bị vô hiệu hóa, thì bạn có thể nhận thấy một sự giảm hiệu năng giữa pg_start_backup và pg_stop_backup, vì full_page_write bị ép có hiệu lực trong chế độ sao lưu). Bạn phải đảm bảo rằng các bước đó được triển khai tuần tự, không có bất kỳ sự chồng lấn nào có thể, hoặc bạn sẽ vô hiệu hóa sự sao lưu.

Hãy chắc là dump sao lưu của bạn bao gồm tất cả các tệp trong thư mục bó cơ sở dữ liệu (nghĩa là, /usr/local/pgsql/data). Nếu bạn đang sử dụng các không gian bảng mà không nằm bên dưới thư mục này, thì hãy cẩn thận để cũng đưa chúng vào (và hãy chắc chắn rằng dump sao lưu của bạn lưu trữ các liên kết biểu tượng như các liên kết, nếu không thì sự phục hồi sẽ làm hỏng các không gian bảng của bạn).

Tuy nhiên, bạn có thể bỏ qua khỏi dump sao lưu các tệp bên trong thư mục con pg_xlog/ của bó đó. Sự tinh chỉnh một chút này đáng giá vì nó làm giảm rủi ro các sai sót khi phục hồi. Điều này dễ dàn xếp nếu pg_xlog/ là một liên kết biểu tượng chỉ tới vài nơi ngoài thư mục của bó đó, nó là một thiết lập phổ biến vì các lý do hiệu năng.

Để sử dụng sao lưu đó, bạn sẽ cần giữ cho tất cả các tệp phân đoạn WAL được tạo ra trong và sau sự sao lưu hệ thống tệp. Để giúp bạn làm điều này, hàm pg_stop_backup tạo một tệp lịch sử sao lưu mà

ngay lập tức được lưu trữ trong khu vực lưu trữ của WAL. Tệp này được đặt tên sau tệp phân đoạn WAL đầu tiên mà bạn cần để sao lưu hệ thống tệp. Ví dụ, nếu tệp khởi đầu WAL là 000000010000123400000055CD thì tệp lịch sử sao lưu sẽ được đặt tên là thứ gì đó giống như là 000000010000123400000055CD.007C9330.backup. (Phần thứ 2 của tên tệp ngụ ý một vị trí chính xác trong tệp WAL, và có thể thường bị bỏ qua). Một khi bạn đã lưu trữ an toàn sao lưu hệ thống tệp và các tệp phân đoạn WAL được sử dụng trong quá trình sao lưu (như được chỉ định trong tệp lịch sử sao lưu), thì tất cả các phân đoạn WAL được lưu trữ với các tên được đánh số ít hơn sẽ không còn cần thiết để phục hồi sự sao lưu hệ thống tệp và có thể được xóa. Tuy nhiên, bạn nên cân nhắc giữa vài bộ sao lưu sẽ là tuyệt đối chắc chắn rằng bạn có thể phục hồi các dữ liệu của bạn.

Tệp lịch sử sao lưu chỉ là một tệp văn bản nhỏ. Nó chứa chuỗi nhãn mà bạn đã trao cho pg_start_backup, cũng như các thời điểm bắt đầu và kết thúc và các phân đoạn WAL của sao lưu đó. Nếu bạn đã sử dụng nhãn đó để nhận diện tệp dump có liên quan, thì tệp lịch sử được lưu trữ là đủ để nói cho bạn tệp dump nào phải phục hồi.

Vì bạn phải giữ khoảng tất cả các tệp WAL được lưu trữ ngược về cho sao lưu cơ bản mới nhất của bạn, khoảng thời gian giữa các các sao lưu cơ bản thường nên được chọn dựa vào bao nhiêu kho bạn muốn tiêu dùng trong các tệp WAL được lưu trữ. Bạn cũng nên xem xét mất bao lâu bạn được chuẩn bị để tiêu dùng việc phục hồi, nếu sự phục hồi sẽ là cần thiết - hệ thống sẽ phải chơi lại tất cả các phân đoạn WAL, và điều đó có lẽ mất chút thời gian nếu nó là đã lâu kể từ lần sao lưu mới nhất.

Cũng đáng lưu ý rằng hàm pg_start_backup tạo một tệp có tên là backup_label trong thư mục bó cơ sở dữ liệu, mà nó bị pg_stop_backup loại bỏ. Tệp này tất nhiên sẽ được lưu trữ như một phần của tệp dump sao lưu của bạn. Tệp nhãn sao lưu bao gồm chuỗi nhãn mà bạn trao cho pg_start_backup, cũng như thời gian mà ở đó pg_start_backup từng được chạy, và tên của tệp WAL khởi tạo. Trong trường hợp có sự lộn xộn vì thế có khả năng để nhìn vào bên trong một tệp dump sao lưu và xác định chính xác phiên làm việc sao lưu nào tệp dump đó tới từ.

Cũng có khả năng tiến hành một dump sao lưu trong khi máy chủ bị dừng. Trong trường hợp này, bạn rõ ràng không thể sử dụng pg_start_backup hoặc pg_stop_backup, và bạn vì thế sẽ bị để lại cho các thiết bị của riêng bạn để theo dõi dump sao lưu nào là gì và các tệp WAL có liên quan đi ngược lại xa tới đâu. Thường là tốt hơn để đi theo thủ tục lưu trữ liên tục ở trên.

24.3.3. Phục hồi bằng việc sử dụng một sao lưu lưu trữ liên tục

OK, điều tồi tệ nhất đã xảy ra và bạn cần phải phục hồi từ sao lưu của bạn. Đây là thủ tục:

1. Dừng máy chủ, nếu nó đang chạy
2. Nếu bạn có không gian để làm điều đó, hãy sao chép toàn bộ thư mục dữ liệu bó và bất kỳ không gian bảng nào sang một vị trí tạm thời trong trường hợp bạn cần chúng sau này. Lưu ý là sự đề phòng này sẽ đòi hỏi bạn có đủ không gian rỗi rãi trong hệ thống của bạn để giữ 2 bản sao của cơ sở dữ liệu đang tồn tại của bạn. Nếu bạn không có đủ không gian đĩa, bạn nên ít nhất lưu các nội dung của thư mục con pg_xlog của bó, vì nó có thể chứa các lưu ký mà từng không được lưu trữ trước khi hệ thống bị sập.
3. Loại bỏ tất cả các tệp và thư mục con đang tồn tại dưới thư mục dữ liệu bó và dưới các thư

mục gốc root của bất kỳ không gian bảng nào bạn đang sử dụng.

4. Phục hồi các tệp cơ sở dữ liệu từ sao lưu hệ thống tệp của bạn. Hãy chắc chắn rằng chúng được phục hồi bằng quyền sở hữu đúng (người sử dụng hệ thống cơ sở dữ liệu, chứ không phải gốc root!) và với các quyền cho phép đúng. Nếu bạn đang sử dụng các không gian bảng, thì bạn nên kiểm tra các đường liên kết biểu tượng trong pg_tblspc/ để xem chúng đã được phục hồi lại đúng hay chưa.
5. Loại bỏ bất kỳ tệp nào hiện diện trong pg_xlog/; chúng tới từ sao lưu hệ thống tệp và vì thế có lẽ đã lỗi thời chứ không phải hiện hành. Nếu bạn từng chưa lưu trữ pg_xlog/ hoàn toàn, thì sau đó hãy tái tạo nó bằng các quyền cho phép đúng phù hợp, hãy thận trọng để đảm bảo rằng bạn tái thiết lập nó như một liên kết biểu tượng nếu bạn đã có nó được thiết lập theo cách đó trước đó.
6. Nếu bạn còn chưa lưu trữ các tệp phân đoạn WAL mà bạn đã lưu ở bước 2, hãy sao chép chúng vào pg_xlog/. (Là tốt nhất để sao chép chúng, chứ không chuyển chúng, nên bạn vẫn có các tệp không bị sửa đổi nếu một vấn đề xảy ra và bạn phải bắt đầu lại).
7. Hãy tạo một tệp lệnh phục hồi recovery.conf trong thư mục dữ liệu bó (xem Chương 26). Bạn cũng có thể muốn tạm thời sửa đổi pg_hba.conf để ngăn chặn những người sử dụng thông thường khỏi việc kết nối cho tới khi bạn chắc chắn sự phục hồi đó đã thành công.
8. Hãy khởi động máy chủ. Máy chủ sẽ đi vào chế độ phục hồi và xử lý để đọc qua các tệp WAL được lưu trữ mà nó cần. Sự phục hồi sẽ chấm dứt vì một lỗi bên ngoài, máy chủ có thể đơn giản được khởi động lại và nó sẽ tiếp tục phục hồi. Dựa vào sự kết thúc của tiến trình phục hồi, máy chủ sẽ đặt tên lại cho recovery.conf thành recovery.done (để ngăn chặn việc ngẫu nhiên vào lại chế độ phục hồi sau này) và sau đó bắt đầu các hoạt động bình thường của cơ sở dữ liệu.
9. Nghiên cứu các nội dung cơ sở dữ liệu để đảm bảo bạn đã phục hồi về tình trạng mong muốn. Nếu không, hãy trở về bước 1. Nếu tất cả là tốt, hãy cho phép những người sử dụng kết nối bằng việc phục hồi pg_hba.conf về bình thường.

Phần chủ chốt của tất cả điều này là để thiết lập một tệp cấu hình phục hồi mà mô tả cách bạn muốn phục hồi và cách sự phục hồi sẽ chạy xa tới đâu. Bạn có thể sử dụng recovery.conf.sample (thường nằm trong thư mục cài đặt share/) như là một nguyên mẫu (prototype). Một điều bạn tuyệt đối phải chỉ định trong recovery.conf là restore_command, nó nói cho PostgreSQL cách để truy xuất các phân đoạn tệp WAL được lưu trữ. Giống như archive_command, đây là một chuỗi lệnh shell. Nó có thể chứa %f, nó được thay thế bằng tên của tệp lưu ký mong muốn, và %p, nó được thay thế bằng tên đường dẫn để sao chép tệp lưu ký đó. (Tên đường dẫn là tương đối đối với thư mục làm việc hiện hành, nghĩa là, thư mục dữ liệu bó).

Hãy viết %% nếu bạn cần nhúng một ký tự % vào lệnh đó. Lệnh hữu dụng đơn giản nhất là dạng:

```
restore_command = 'cp /mnt/server/archivedir/%f %p'
```

nó sẽ sao chép các phân đoạn WAL được lưu trữ trước đó từ thư mục /mnt/server/archivedir. Tất nhiên, bạn có thể sử dụng thứ gì đó phức tạp hơn nhiều, có thể thậm chí một script shell yêu cầu

người vận hành kích hoạt một đầu băng từ đúng phù hợp.

Điều quan trọng là lệnh đó trả về tình trạng thoát ra không phải là zero khi hỏng. Lệnh đó sẽ được gọi khi yêu cầu các tệp không hiện diện trong lưu trữ đó; nó phải trả về không phải zero khi được hỏi như vậy. Đây không phải là một điều kiện lỗi. Không phải tất cả các tệp được yêu cầu sẽ là các tệp phân đoạn WAL; bạn cũng nên kỳ vọng yêu cầu cho các tệp với hậu tố là .backup hoặc .history. Cũng nhận thức rằng tên cơ bản của đường dẫn %p sẽ là khác với %f; đừng kỳ vọng chúng sẽ có khả năng trao đổi được cho nhau.

Các phân đoạn WAL mà không thể tìm thấy trong lưu trữ sẽ được thấy trong pg_xlog/; điều này cho phép sử dụng các phân đoạn không được lưu trữ. Tuy nhiên, các phân đoạn mà sẵn sàng từ lưu trữ đó sẽ được sử dụng ưu tiên cho các tệp trong pg_xlog/. Hệ thống sẽ không ghi đè các nội dung đang tồn tại của pg_xlog/ khi truy xuất các tệp được lưu trữ.

Thông thường, sự phục hồi sẽ tiến hành qua tất cả các phân đoạn WAL, vì thế việc phục hồi cơ sở dữ liệu về điểm hiện hành đúng lúc (hoặc càng sát có thể càng tốt cho các phân đoạn WAL sẵn sàng được đưa ra). Vì thế, sự phục hồi thông thường sẽ kết thúc với thông điệp "không tìm thấy tệp" (file not found), văn bản y hệt của thông điệp lỗi phụ thuộc vào sự lựa chọn của bạn đối với restore_command. Bạn cũng có thể thấy một thông điệp lỗi ở đầu của sự phục hồi đối với một tệp có tên dạng như 00000001.history. Điều này cũng là bình thường và không chỉ ra một vấn đề trong các giải pháp phục hồi đơn giản; xem Phần 24.3.4 để thảo luận.

Nếu bạn muốn phục hồi về một vài điểm trước đó đúng lúc (ví dụ, ngay trước khi DBA junior đã bỏ đi bảng giao dịch chính của bạn), hãy chỉ định điểm dừng được yêu cầu trong recovery.conf. Bạn có thể chỉ định điểm dừng, được biết tới như là "cái đích phục hồi", hoặc bằng ngày tháng/thời gian (date/time) hoặc bằng sự hoàn tất của một ID giao dịch đặc biệt. Lựa chọn chỉ viết ngày tháng/thời gian là rất có khả năng dùng được, vì không có các công cụ để giúp bạn nhận diện với bất kỳ độ chính xác nào cho ID giao dịch nào sẽ sử dụng.

> **Lưu ý**: Điểm dừng phải là sau thời điểm kết thúc của sao lưu cơ bản, nghĩa là, thời điểm kết thúc của pg_stop_backup. Bạn không thể sử dụng một sao lưu cơ bản để phục hồi về một thời điểm khi mà sao lưu đó từng đang được thực hiện. (Để phục hồi về một thời điểm như vậy, bạn phải đi ngược về sao lưu cơ bản trước đó của bạn và tiến lên trước từ chỗ đó).

Nếu sự phục hồi thấy các dữ liệu WAL bị hỏng, sự phục hồi sẽ dừng ở điểm đó và máy chủ sẽ không khởi động. Trong trường hợp như vậy tiến trình phục hồi có thể được chạy lại từ đầu, chỉ định một "cái đích phục hồi" trước thời điểm hỏng sao cho sự phục hồi có thể hoàn tất bình thường. Nếu sự phục hồi hỏng vì một lý do bên ngoài, ví dụ như sự hỏng hệ thống hoặc nếu lưu trữ WAL đã trở nên không thể truy cập được, thì sự phục hồi có thể đơn giản được khởi động lại và nó sẽ khởi động hầu như ở đúng nơi mà nó bị hỏng. Sự khởi động lại phục hồi làm việc rất giống với việc kiểm tra điểm trong hoạt động thông thường: máy chủ định kỳ ép tất cả tình trạng của nó vào đĩa, và sau đó cập nhật tệp pg_control để chỉ ra rằng các dữ liệu WAL được xử lý trước đó không cần phải được quét lại một lần nữa.

24.3.4. Dòng thời gian

Khả năng phục hồi cơ sở dữ liệu về điểm trước đó đúng lúc tạo ra một vài sự phức tạp giống như các câu chuyện khoa học viễn tưởng về du lịch thời gian và vũ trụ song song. Ví dụ, trong lịch sử ban đầu của cơ sở dữ liệu, giả thiết bạn đã bỏ một bảng sống còn lúc 5:15PM buổi chiều tối hôm thứ ba, nhưng đã không nhận thức được lỗi của bạn cho tới buổi trưa ngày thứ tư. Không bị bối rối, bạn đi ra khỏi sự sao lưu của bạn, phục hồi về điểm đúng lúc 5:14PM chiều tối ngày thứ ba, và lại tiếp tục cho chạy. Trong lịch sử này của vũ trụ các cơ sở dữ liệu, bạn chưa bao giờ bỏ bảng cả. Nhưng giả sử sau này bạn nhận ra rằng đây không phải là một ý tưởng lớn như vậy, và có thể muốn quay về khoảng nào đó của buổi sáng ngày thứ tư theo lịch sử gốc ban đầu. Bạn sẽ không có khả năng nếu, trong khi cơ sở dữ liệu của bạn từng đang chạy, nó đã ghi đè vài tệp phân đoạn WAL mà đã dẫn tới một loạt các bản ghi WAL được tạo ra sau khi bạn đã thực hiện một phục hồi đúng thời điểm từ những bản ghi đã được tạo ra theo lịch sử cơ sở dữ liệu gốc ban đầu.

Để làm việc với vấn đề này, PostgreSQL có một ký hiệu dòng thời gian. Bất kỳ khi nào sự phục hồi lưu trữ hoàn tất, một dòng thời gian sẽ được tạo ra để nhận diện một loạt các bản ghi WAL được tạo ra sau sự phục hồi đó. Số ID dòng thời gian là một phần của các tên tệp phân đoạn WAL nên một dòng thời gian mới không ghi đè dữ liệu WAL được các dòng thời gian trước đó tạo ra. Trong thực tế có khả năng lưu trữ nhiều dòng thời gian khác nhau. Trong khi điều đó có lẽ dường như giống một tính năng vô dụng, thì nó thường là một phao cứu sinh. Hãy cân nhắc tình huống nơi mà bạn sẽ hoàn toàn không chắc đúng thời điểm nào để phục hồi về, và vì thế phải làm vài phục hồi đúng thời điểm bằng cách thử vá lỗi cho tới khi bạn tìm được nơi tốt nhất để phân nhánh từ lịch sử cũ. Không có các dòng thời gian thì tiến trình này có thể sớm tạo ra một mớ lộn xộn không thể quản lý nổi. Với các dòng thời gian, bạn có thể phục hồi về bất kỳ tình trạng nào trước đó, bao gồm cả các tình trạng trong các nhánh dòng thời gian mà bạn đã bỏ qua trước đó.

Mỗi lần một dòng thời gian được tạo ra, thì PostgreSQL tạo ra một tệp "lịch sử dòng thời gian" mà chỉ ra dòng thời gian nào nó đã phân nhánh từ đó và khi nào. Các tệp lịch sử đó là cần thiết để cho phép hệ thống chọn đúng các tệp phân đoạn WAL khi phục hồi từ một lưu trữ có chứa nhiều dòng thời gian. Vì thế, chúng được lưu trữ trong vùng lưu trữ WAL giống hệt như các tệp phân đoạn WAL. Các tệp lịch sử chỉ là các tệp văn bản nhỏ, nên là rẻ và đúng phù hợp để giữ chúng khoảng vô thời hạn (không giống như các tệp phân đoạn mà là lớn). Bạn có thể, nếu bạn muốn, thêm các bình luận vào một tệp lịch sử để ghi lại các chú giải của riêng bạn về làm thế nào và vì sao dòng thời gian đặc biệt này đã được tạo ra. Các chú giải như vậy sẽ đặc biệt có giá trị khi bạn có một rừng các dòng thời gian khác nhau như là kết quả của sự thí điểm.

Hành vi mặc định của sự phục hồi là phục hồi theo dòng thời gian y hệt từng là hiện hành khi sao lưu cơ bản đã được thực hiện. Nếu bạn muốn phục hồi trong một vài dòng thời gian con (đó là, bạn muốn trở về một vài tình trạng mà bản thân nó từng được tạo ra sau một cố gắng phục hồi), thì bạn cần chỉ định ID dòng thời gian đích trong recovery.conf. Bạn không thể phục hồi trong các dòng thời gian mà đã phân nhánh sớm hơn bản sao lưu cơ bản.

24.3.5. Mẹo và ví dụ

Một vài mẹo cho việc thiết lập cấu hình việc lưu trữ liên tục được đưa ra ở đây.

24.3.5.1. Sao lưu nóng máy đứng một mình

Có khả năng sử dụng các tiện ích sao lưu của PostgreSQL để tạo ra các sao lưu nóng máy đứng một mình. Chúng là các sao lưu mà không thể được sử dụng cho phục hồi đúng thời điểm, vâng thường nhanh hơn nhiều để sao lưu và phục hồi so với các dump pg_dump. (Chúng cũng lớn hơn nhiều so với các dump pg_dump, nên trong một vài trường hợp ưu thế tốc độ có thể bị phủ định).

Để chuẩn bị cho các sao lưu nóng máy đứng một mình, hãy thiết lập wal_level về archive (hoặc hot_standby), archive_mode về bật (on), và thiết lập một archive_command mà thực hiện việc lưu trữ chỉ khi một tệp chuyển đổi tồn tại. Ví dụ:

```
archive_command = 'test ! -f /var/lib/pgsql/backup_in_progress || (test ! -f /var/lib/pgsql/
```

Lệnh này sẽ thực hiện việc lưu trữ khi tồn tại, và nếu không sẽ âm thầm trả về tình trạng thoát ra zero (cho phép PostgreSQL tái chế tệp WAL không được mong muốn).

Với sự chuẩn bị này, một sao lưu có thể được tiến hành bằng việc sử dụng một script như sau:

```
touch /var/lib/pgsql/backup_in_progress
psql -c "select pg_start_backup('hot_backup');"
tar -cf /var/lib/pgsql/backup.tar /var/lib/pgsql/data/
psql -c "select pg_stop_backup();"
rm /var/lib/pgsql/backup_in_progress
tar -rf /var/lib/pgsql/backup.tar /var/lib/pgsql/archive/
```

Tệp chuyển đổi /var/lib/pgsql/backup_in_progress được tạo ra trước tiên, cho phép việc lưu trữ các tệp WAL hoàn chỉnh xảy ra. Sau sự sao lưu thì tệp chuyển đổi sẽ bị loạt bỏ. Các tệp WAL được lưu trữ sau đó sẽ được thêm vào sự sao lưu sao cho cả sao lưu cơ bản và tất cả các tệp WAL đều là một phần của cùng một tệp tar. Xin nhớ thêm việc điều khiển lỗi vào các script sao lưu của bạn.

24.3.5.2. Lưu ký lưu trữ được nén

Nếu kích cỡ kho lưu trữ là một mối quan tâm, hãy sử dụng pg_compresslog, http://pglesslog.projects.postgresql.org, để loại bỏ full_page_writes không cần thiết và kéo theo không gian từ các tệp WAL. Bạn có thể sau đó sử dụng gzip để nén tiếp đầu ra của pg_compresslog:

```
archive_command = 'pg_compresslog %p - | gzip > /var/lib/pgsql/archive/%f'
```

Bạn sau đó sẽ cần sử dụng gunzip và pg_decompresslog trong quá trình phục hồi:

```
restore_command = 'gunzip < /mnt/server/archivedir/%f | pg_decompresslog - %p'
```

24.3.5.3. Script archive_command

Nhiều người chọn sử dụng các script để xác định archive_command của họ, sao cho khoản đầu vào archive_command của họ trông rất đơn giản:

```
archive_command = 'local_backup_script.sh "%p" "%f"'
```

Việc sử dụng một tệp script tách bạch được khuyến cáo bất kỳ khi nào bạn muốn sử dụng nhiều hơn

một lệnh duy nhất trong tiến trình lưu trữ. Điều này cho phép tất cả sự phức tạp sẽ được quản lý trong script đó, điều có thể được viết trong một ngôn ngữ scripting phổ biến như bash hoặc perl.

Các ví dụ về các yêu cầu có thể được giải quyết trong một script bao gồm:

- Việc sao chép dữ liệu tới kho dữ liệu an toàn ở nơi bên ngoài
- Tạo bó các tệp WAL sao cho chúng được truyền đi mỗi 3 tiếng đồng hồ, thay vì cùng 1 lúc
- Việc đối diện với các phần mềm sao lưu và phục hồi khác
- Việc đối diện với việc giám sát phần mềm để báo cáo các lỗi

  Mẹo: Khi sử dụng một script archive_command, được mong đợi kích hoạt logging_collector. Bất kỳ cấu hình thông điệp nào cũng sẽ được chuẩn đoán dễ dàng nếu chúng hỏng.

24.3.6. Khiếm khuyết

Trong tài liệu này, có vài hạn chế kỹ thuật lưu trữ liên tục. Chúng có lẽ sẽ được sửa trong các phiên bản trong tương lai:

- Các hoạt động trong các chỉ số băm sẽ không được lưu ký WAL hiện hành, nên sự chơi lại sẽ không cập nhật các chỉ số đó. Điều này sẽ có nghĩa là bất kỳ cuộc chèn mới nào cũng sẽ bị bỏ qua đối với chỉ số đó, các hàng được cập nhật hình như sẽ biến mất và các hàng bị xóa vẫn sẽ giữ lại các con trỏ. Nói cách khác, nếu bạn sửa đổi một bảng bằng một chỉ số băm trong nó rồi sau đó bạn sẽ có các kết quả truy vấn không đúng trong một máy chủ dự phòng. Khi sự phục hồi hoàn tất thì được khuyến cáo rằng bạn REINDEX bằng tay từng chỉ số như vậy sau khi hoàn tất một hoạt động phục hồi.
- Nếu một lệnh CREATE DATABASE được thực thi trong khi một sao lưu cơ bản đang được thực hiện, và sau đó cơ sở dữ liệu mẫu template mà CREATE DATABASE đó được sao chép được sửa đổi trong khi sao lưu cơ bản vẫn còn đang chạy, thì có khả năng là sự phục hồi sẽ làm cho các sửa đổi đó cũng sẽ được nhân giống vào cơ sở dữ liệu được tạo ra đó. Điều này tất nhiên là không được mong đợi. Để tránh rủi ro này, tốt nhất hãy không sửa đổi bất kỳ cơ sở dữ liệu mẫu template nào trong khi tiến hành một sao lưu cơ bản.
- Lệnh CREATE DATABASE sẽ được lưu ký WAL với đường dẫn tuyệt đối, và vì thế sẽ được chơi lại như sự tạo ra không gian bảng với cùng đường dẫn tuyệt đối y hệt. Điều này có thể là không mong muốn nếu lưu ký đó đang được chơi lại trong một máy khác. Nó có thể là nguy hiểm thậm chí nếu được chơi lại trong máy y hệt, nhưng bên trong một thư mục dữ liệu mới: sự chơi lại đó sẽ vẫn ghi đè các nội dung của không gian bảng gốc ban đầu. Để tránh nhược điểm dạng này, thực tế tốt nhất là thực hiện một sao lưu cơ bản mới sau khi tạo hoặc bỏ các không gian bảng.

Cũng nên được lưu ý rằng định dạng WAL mặc định là khá lớn vì nó bao gồm nhiều hình chụp các trang đĩa. Các hình chụp trang đĩa đó được thiết kế để hỗ trợ phục hồi hỏng hóc, vì chúng ta có thể cần phải sửa các trang đĩa được ghi một phần. Phụ thuộc vào phần cứng và phần mềm hệ thống của bạn, rủi ro của việc ghi một phần có thể là đủ nhỏ để bỏ qua, trong trường hợp đó bạn có thể làm giảm đáng kể tổng dung lượng các lưu ký được lưu trữ bằng việc tắt các hình chụp trang bằng việc

sử dụng tham số full_page_writes. (Đọc các ghi chú và cảnh báo trong Chương 29 trước khi bạn làm thế). Việc tắt các hình chụp trang không ngăn cản được sử dụng các lưu ký cho các hoạt động PITR. Một lĩnh vực cho sự phát triển trong tương lai là nén các dữ liệu WAL được lưu trữ bằng việc loại bỏ các bản sao trang không cần thiết thậm chí khi full_page_writes đang bật. Trong khi chờ đợi, các quản trị viên có thể mong muốn làm giảm số các hình chụp trang được đưa vào trong WAL bằng việc làm gia tăng các tham số khoảng cách điểm kiểm tra càng nhiều có thể càng tốt.

24.4. Chuyển đổi giữa các phiên bản

Phần này thảo luận cách chuyển đổi dữ liệu của cơ sở dữ liệu của bạn từ một phiên bản PostgreSQL sang một phiên bản mới hơn. Thủ tục cài đặt phần mềm mỗi lần là không tuân theo phần này; các chi tiết đó có trong Chương 15.

Các phiên bản chính của PostgreSQL được thể hiện bằng các nhóm 2 chữ số đầu tiên của số phiên bản, như, 8.4. Các phiên bản phụ của PostgreSQL được thể hiện bằng nhóm 3 chữ số phiên bản, như, 8.4.2 là phiên bản phụ thứ 2 của 8.4. Các phiên bản phụ không bao giờ thay đổi định dạng kho nội bộ và luôn tương thích với các phiên bản phụ sớm trước và muộn sau của cùng y hệt số phiên bản chính đó, như 8.4.2 thì tương thích với 8.4, 8.4.1 và 8.4.6. Để cập nhật giữa các phiên bản tương thích nhau, bạn đơn giản thay thế các tệp thực thi trong khi máy chủ được tắt và khởi động lại máy chủ đó. Thư mục dữ liệu vẫn sẽ không thay đổi - các bản nâng cấp phụ là các bản đơn giản.

Đối với các phiên bản chính của PostgreSQL, định dạng kho dữ liệu nội bộ tuân theo sự thay đổi, vì thế làm phức tạp hóa các bản nâng cấp. Phương pháp truyền thống cho việc loại bỏ dữ liệu đối với một phiên bản chính là dump và tải lại cơ sở dữ liệu đó. Cách khác, các khả năng được kiểm thử ít tốt hơn là sẵn sàng, như được thảo luận bên dưới.

Các phiên bản chính mới cũng thường đưa ra một vài khả năng không tương thích mà người sử dụng có thể nhìn thấy được, nên những thay đổi của việc lập trình ứng dụng có thể được yêu cầu. Những người sử dụng thận trọng sẽ muốn kiểm thử các ứng dụng máy trạm trong phiên bản mới trước khi chuyển qua hoàn toàn; vì thế thường là ý tưởng tốt để thiết lập các cài đặt đồng thời các phiên bản cũ và mới. Khi việc kiểm thử một bản nâng cấp chính của PostgreSQL, hãy cân nhắc các chủng loại sau đây đối với các thay đổi có khả năng:

Administration (Quản trị)

> Các khả năng sẵn sàng cho những quản trị viên để giám sát và kiểm soát máy chủ thường thay đổi và cải tiến theo từng phiên bản chính.

SQL

> Thường điều này bao gồm các khả năng lệnh SQL mới và không thay đổi trong hành vi, trừ phi được nhắc đặc biệt trong các ghi chú phiên bản.

Library API (Thư viện API)

> Thường các thư viện như libpq chỉ thêm chức năng mới, một lần nữa trừ phi được nhắc tới trong các chú giải phiên bản.

System Catalogs (Các catalog hệ thống)

Catalog hệ thống thay đổi thường chỉ ảnh hưởng tới các công cụ quản lý cơ sở dữ liệu.

Server C - Language API (Giao diện lập trình ứng dụng API ngôn ngữ C của máy chủ)

Điều này liên quan tới những thay đổi trong API chức năng phụ trợ (backend), nó được viết trong ngôn ngữ lập trình C. Những thay đổi như vậy ảnh hưởng tới mã mà tham chiếu tới các chức năng phụ trợ nằm sâu bên trong máy chủ.

24.4.1. Chuyển đổi dữ liệu qua pg_dump

Để dump dữ liệu từ một phiên bản chính của PostgreSQL và tải lại nó trong một phiên bản khác, bạn phải sử dụng pg_dump; các phương pháp sao lưu mức hệ thống tệp sẽ không làm việc. (Có các kiểm tra tại chỗ để ngăn chặn bạn khỏi việc sử dụng một thư mục dữ liệu với một phiên bản không tương thích của PostgreSQL, nên không có thiệt hại lớn nào có thể được thực hiện bằng việc cố khởi tạo phiên bản máy chủ sai trong một thư mục dữ liệu). Được khuyến cáo rằng bạn sử dụng các chương trình pg_dump và pg_dumpall từ phiên bản mới hơn của PostgreSQL, để tận dụng những cải tiến có thể được thực hiện trong các chương trình đó.

Các phiên bản hiện hành của các chương trình dump có thể đọc dữ liệu từ bất kỳ phiên bản máy chủ nào ngược về 7.0. Thời gian không làm việc ít nhất có thể đạt được bằng việc cài đặt máy chủ mới trong một thư mục khác và chạy cả các máy chủ cũ và mới song song, trong các cổng khác nhau. Sau đó bạn có thể sử dụng thứ gì đó như là:

```
pg_dumpall -p 5432 | psql -d postgres -p 6543
```

để truyền các dữ liệu của bạn. Hoặc bạn có thể sử dụng một tệp trung gian nếu bạn muốn. Sau đó bạn có thể tắt máy chủ cũ và khởi động máy chủ mới bằng việc sử dụng cổng mà máy chủ cũ đã từng chạy trên đó. Bạn nên chắc chắn rằng cơ sở dữ liệu cũ không được cập nhật sau khi bạn bắt đầu chạy pg_dumpall, nếu không thì bạn sẽ mất các bản cập nhật đó. Xem Chương 19 để có các thông tin về cách để cấm truy cập.

Nếu bạn không thể hoặc không muốn chạy 2 máy chủ song song, thì bạn có thể thực hiện bước sao lưu trước khi cài đặt phiên bản mới, tắt máy chủ cũ, chuyển phiên bản cũ ra khỏi máy, cài đặt phiên bản mới, khởi động máy chủ mới, và phục hồi lại các dữ liệu. Ví dụ:

```
pg_dumpall > backup
pg_ctl stop
mv /usr/local/pgsql /usr/local/pgsql.old
# Rename any tablespace directories as well
cd ~/postgresql-9.0.13
gmake install
initdb -D /usr/local/pgsql/data
postgres -D /usr/local/pgsql/data
psql -f backup postgres
```

Xem Chương 17 về các cách thức để khởi động và dừng máy chủ và các chi tiết khác. Các chỉ dẫn cài đặt sẽ khuyến cáo bạn các địa điểm chiến lược để thực hiện các bước đó.

Lưu ý: Khi bạn "chuyển cài đặt ra khỏi máy chủ" thì nó có thể không sử dụng tuyệt vời được nữa. Một vài chương trình thực thi có chứa các đường dẫn tuyệt đối tới các chương trình khác nhau được cài đặt và các tệp dữ liệu. Điều này thường không phải là một vấn đề

lớn, nhưng nếu bạn có kế hoạch sử dụng 2 cài đặt song song nhau trong một khoảng thời gian thì bạn nên chỉ định chúng tới các thư mục cài đặt khác nhau lúc xây dựng. (Vấn đề này được chỉnh trong PostgreSQL phiên bản 8.0 và sau này, miễn là bạn chuyển tất cả các thư mục con có chứa các tệp được cài đặt cùng với nhau; ví dụ, nếu /usr/local/postgres/bin/ đi tới /usr/local/postgres.old/bin/, thì /usr/local/postgres/share/ phải đi tới /usr/local/postgres.old/share/. Trong các phiên bản trước 8.0 thì việc chuyển cài đặt giống như thế này sẽ không làm việc).

24.4.2. Các phương pháp chuyển đổi dữ liệu khác

Chương trình pg_upgrade của contrib cho phép một cài đặt sẽ được chuyển đổi tại chỗ từ một phiên bản chính của PostgreSQL tới phiên bản tiếp sau. Nhớ trong đầu rằng phương pháp này không cung cấp bất kỳ phạm vi nào cho việc chạy các phiên bản cũ và mới đồng thời. Hơn nữa, pg_upgrade là ít được kiểm thử chinh chiến hơn so với pg_dump, nên việc có một sao lưu được cập nhật được khuyến cáo mạnh mẽ trong trường hợp thứ gì đó không đúng.

Có khả năng sử dụng các phương pháp nhân bản nhất định, như Slony, để tạo ra một máy chủ dự phòng với phiên bản được cập nhật của PostgreSQL. Sự dự phòng có thể là trên cùng y hệt máy tính đó hoặc trên một máy tính khác. Một khi nó đã đồng bộ với máy chủ chính (chạy phiên bản cũ hơn của PostgreSQL), thì bạn có thể chuyển các máy chủ chính và thực hiện sự dự phòng cho máy chủ chính và tắt cài đặt cơ sở dữ liệu cũ hơn đi. Một sự chuyển qua như vậy chỉ làm máy ngừng chạy vài giây cho việc nâng cấp.

Chương 25. Tính sẵn sàng cao, cân bằng tải, và nhân bản

Các máy chủ cơ sở dữ liệu có thể làm việc cùng nhau để cho phép một máy chủ thứ 2 tiếp nhận nhanh chóng nếu máy chủ đầu tiên bị hỏng (tính sẵn sàng cao), hoặc cho phép các máy tính chủ phục vụ các dữ liệu y hệt đó (cân bằng tải). Lý tưởng, các máy chủ cơ sở dữ liệu có thể làm việc cùng nhau một cách trong suốt. Các máy chủ web phục vụ các trang web tĩnh có thể được kết hợp hoàn toàn dễ dàng chỉ bằng việc cân bằng tải các yêu cầu web tới nhiều máy. Trong thực tế, các máy chủ cơ sở dữ liệu chỉ đọc cũng có thể được kết hợp khá dễ dàng. Không may, hầu hết các máy chủ cơ sở dữ liệu có một sự pha trộn các yêu cầu đọc/ghi, và các máy chủ đọc/ghi là khó hơn nhiều để kết hợp. Điều này là vì dù các dữ liệu chỉ đọc cần phải đặt trong từng máy chủ chỉ một lần, thì một sự ghi tới bất kỳ máy chủ nào sẽ phải được nhân giống tới tất cả các máy chủ sao cho các yêu cầu đọc trong tương lai đối với các máy chủ đó sẽ trả về các kết quả nhất quán.

Vấn đề đồng bộ hóa này là khó khăn cơ bản đối với các máy chủ làm việc cùng nhau. Vì không có giải pháp duy nhất nào loại bỏ được ảnh hưởng của vấn đề đồng bộ cho tất cả các trường hợp sử dụng, nên có nhiều giải pháp. Từng giải pháp giải quyết vấn đề này theo một cách khác nhau, và làm giảm tối thiểu ảnh hưởng của nó đối với một tải làm việc đặc biệt.

Một vài giải pháp làm việc với sự đồng bộ hóa bằng việc cho phép chỉ một máy chủ sửa đổi dữ liệu. Các máy chủ mà có thể sửa đổi dữ liệu được gọi là các máy chủ đọc/ghi, chủ (master) hoặc chính. Các máy chủ mà theo dõi các thay đổi trong máy chủ chính được gọi là các máy chủ dự phòng hoặc phụ (slave). Một máy chủ dự phòng không thể được kết nối cho tới khi nó được khuyến khích tới một máy chủ chính được gọi là một máy chủ dự phòng ấm, và một máy chủ mà có thể chấp nhận các kết nối và phục vụ các yêu cầu chỉ đọc được gọi là một máy chủ dự phòng nóng. Một vài giải pháp là đồng bộ, nghĩa là một giao dịch sửa đổi dữ liệu không được xem là được thực hiện cho tới khi tất cả các máy chủ đã thực hiện xong giao dịch đó. Điều này đảm bảo rằng một sự chuyển đổi dự phòng (failover) sẽ không làm mất bất kỳ dữ liệu nào và tất cả các máy chủ được cân bằng tải sẽ trả về các kết quả nhất quán bất kể máy chủ nào được yêu cầu.

Ngược lại, các giải pháp đồng bộ cho phép một vài sự trễ giữa thời điểm của một sự thực hiện xong và sự nhân giống của nó tới các máy chủ khác, mở ra khả năng một vài giao dịch có thể bị mất khi chuyển tới một máy chủ sao lưu, và các máy chủ được cân bằng tải đó có thể trả về các kết quả khá lâu. Giao tiếp không đồng bộ được sử dụng khi sự đồng bộ có thể quá chậm.

Các giải pháp cũng có thể được phân loại theo độ mịn của chúng. Một vài giải pháp có thể chỉ làm việc được với toàn bộ một máy chủ cơ sở dữ liệu, trong khi các giải pháp khác cho phép kiểm soát ở mức từng bảng hoặc từng cơ sở dữ liệu.

Hiệu năng phải được cân nhắc trong mọi sự lựa chọn. Thường có một sự cân nhắc lựa chọn giữa chức năng và hiệu năng. Ví dụ, một giải pháp đồng bộ đầy đủ đối với một mạng chậm có thể làm giảm hiệu năng xuống hơn một nửa, trong khi một giải pháp không đồng bộ có thể có một tác động tối thiểu tới hiệu năng.

Phần còn lại của phần này phác thảo các giải pháp khác nhau về chuyển đổi sự phòng, nhân bản, và

cân bằng tải. Một từ điển thuật ngữ[1] cũng có sẵn.

25.1. So sánh các giải pháp khác nhau

Shared Disk Failover - Chuyển đổi dự phòng đĩa được chia sẻ

Chuyển đổi dự phòng đĩa được chia sẻ tránh tổng chi phí đồng bộ hóa bằng việc chỉ có một bản sao cơ sở dữ liệu. Nó sử dụng một mảng đĩa duy nhất được nhiều máy chủ chia sẻ. Nếu máy chủ cơ sở dữ liệu chính bị hỏng, thì máy chủ dự phòng có khả năng kích hoạt và khởi động cơ sở dữ liệu như thể nó đã đang phục hồi từ một sự sập cơ sở dữ liệu. Điều này cho phép chuyển đổi dự phòng nhanh mà không mất dữ liệu. Chức năng phần cứng được chia sẻ là phổ biến trong các thiết bị lưu trữ mạng. Việc sử dụng một hệ thống tệp mạng cũng là có thể, dù sự chăm sóc phải được thực hiện sao cho hệ thống tệp đó có hành vi POSIX đầy đủ (xem Phần 17.2.1). Một hạn chế đáng kể của phương pháp này là nếu mảng đĩa được chia sẻ bị hỏng hoặc bị lỗi, thì các máy chủ chính và dự phòng đều sẽ không hoạt động được. Một vấn đề khác là máy chủ dự phòng sẽ không bao giờ truy cập kho lưu trữ được chia sẻ trong khi máy chủ chính đang chạy.

File System (Block-Device) Replication - Nhân bản (khối thiết bị) hệ thống tệp

Một phiên bản được sửa đổi chức năng phần cứng được chia sẻ là nhân bản hệ thống tệp, nơi mà tất cả các thay đổi tới một hệ thống tệp được soi gương tới một hệ thống tệp nằm trong một máy tính khác. Hạn chế duy nhất là việc soi gương phải được thực hiện theo một cách thức mà đảm bảo cho máy chủ dự phòng có một bản sao nhất quán hệ thống tệp - đặc biệt, việc ghi vào sự dự phòng phải được thực hiện theo trật tự y hệt như ghi vào máy chủ chính. DRBD là giải pháp nhân bản hệ thống tệp phổ biến cho Linux.

Warm and Hot Standby Using Point-In-Time Recovery (PITR) - Dự phòng ấm và nóng bằng việc sử dụng phục hồi đúng thời điểm (PITR)

Các máy chủ dự phòng ấm và nóng có thể được giữ hiện hành bằng việc đọc một dòng các bản ghi lưu ký ghi trước - WAL (Write - Ahead Log). Nếu máy chủ chính hỏng, thì dự phòng có hầu hết tất cả các dữ liệu của máy chủ chính, và có thể được máy chủ mới chính cơ sở dữ liệu thực hiện nhanh chóng. Đây là sự đồng bộ và chỉ có thể được thực hiện cho toàn bộ máy chủ cơ sở dữ liệu.

Một máy chủ dự phòng PITR có thể được triển khai bằng việc sử dụng việc xuất xưởng lưu ký dựa vào tệp (Phần 25.2) hoặc nhân bản dòng (xem Phần 25.2.5), hoặc một sự kết hợp của cả 2. Để có thông tin về dự phòng nóng, xem Phần 25.5.

Trigger-Based Master-Standby Replication - Nhân bản dự phòng máy chủ chính dựa vào Trigger

Một thiết lập nhân bản dự phòng máy chủ chính gửi đi tất cả các truy vấn sửa đổi dữ liệu tới máy chủ chính. Máy chủ chính gửi không đồng bộ các thay đổi dữ liệu tới máy chủ dự phòng. Sự dự phòng có thể trả lời các truy vấn chỉ đọc trong khi máy chủ chính đang chạy. Máy chủ dự phòng là lý tưởng cho các truy vấn kho dữ liệu (data warehouse).

1 http://www.postgres-r.org/documentation/terms

Slony-I là một ví dụ dạng nhân bản này, với độ mịn theo từng bảng, và hỗ trợ nhiều máy chủ dự phòng. Vì nó cập nhật máy chủ dự phòng không đồng bộ (theo các bó), nên có khả năng bị mất dữ liệu khi chuyển đổi dự phòng.

Statement-Based Replication Middleware - Nhân bản phần mềm trung gian dựa vào lệnh

Với nhân bản phần mềm trung gian (middleware) dựa vào lệnh, một chương trình giao cắt từng truy vấn SQL và gửi nó tới một hoặc tất cả các máy chủ. Từng máy chủ vận hành một cách độc lập. Các truy vấn đọc - ghi được gửi tới tất cả các máy chủ, trong khi các truy vấn chỉ đọc có thể được gửi chỉ tới một máy chủ, cho phép tải công việc đọc sẽ được phân phối.

Nếu các truy vấn được phát đơn giản không có sửa đổi gì, thì các hàm như random(), CURRENT_TIMESTAMP, và các tuần tự có thể có các giá trị khác nhau trong các máy chủ khác nhau. Điều này là vì từng máy chủ vận hành độc lập, và vì các truy vấn SQL được phát đi (chứ không phải các hàng không bị sửa đổi thực sự). Nếu điều này là không thể chấp nhận được, thì hoặc phần mềm trung gian hoặc ứng dụng đó phải truy vấn các giá trị như vậy từ một máy chủ duy nhất và sau đó sử dụng các giá trị đó trong các truy vấn ghi. Một lựa chọn khác là sử dụng lựa chọn nhân bản này với một thiết lập dự phòng máy chủ chính theo truyền thống, nghĩa là các truy vấn sửa đổi dữ liệu chỉ được gửi tới máy chủ chính và sẽ được nhân giống tới các máy chủ dự phòng thông qua nhân bản dự phòng máy chủ chính, không bằng nhân bản phần mềm trung gian. Sự chăm sóc cũng phải được thực hiện sao cho tất cả các giao dịch hoặc được thực hiện hoặc bị bỏ trong tất cả các máy chủ, có thể bằng việc sử dụng sự thực hiện bằng 2 pha (PREPARE TRANSACTION và COMMIT PREPARED. Pgpool-II và Sequoia là các ví dụ nhân bản dạng này).

Asynchronous Multimaster Replication - Nhân bản không đồng bộ nhiều máy chủ chính

Đối với các máy chủ không thường xuyên được kết nối, như các máy tính xách tay hoặc các máy chủ ở xa, việc giữ cho dữ liệu nhất quán giữa các máy chủ là một thách thức. Việc sử dụng nhân bản không đồng bộ nhiều máy chủ chính, từng máy chủ làm việc độc lập, và định kỳ giao tiếp với các máy chủ khác để nhận diện các giao dịch xung đột. Các xung đột có thể được những người sử dụng hoặc các qui tắc xử lý xung đột giải quyết. Bucardo là một ví dụ về dạng nhân bản này.

Synchronous Multimaster Replication - Nhân bản đồng bộ nhiều máy chủ chính

Trong nhân bản đồng bộ nhiều máy chủ chính, từng máy chủ có thể chấp nhận các yêu cầu ghi, và các dữ liệu được sửa đổi sẽ được truyền từ máy chủ gốc ban đầu tới từng máy chủ khác trước khi từng giao dịch được thực hiện. Hoạt động ghi nhiều có thể gây ra việc khóa quá đáng, dẫn tới hiệu năng tồi. Trong thực tế, hiệu năng ghi thường tệ hơn so với hiệu năng của một máy chủ duy nhất. Các yêu cầu đọc có thể được gửi tới bất kỳ máy chủ nào. Một vài triển khai sử dụng đĩa được chia sẻ để làm giảm tổng chi phí giao tiếp. Nhân bản đồng bộ nhiều máy chủ chính là tốt nhất đối với hầu hết các tải công việc đọc, dù ưu thế lớn của nó là bất kỳ máy chủ nào cũng có thể chấp nhận các yêu cầu ghi - không có nhu cầu phân vùng các tải công việc giữa các máy chủ chính và dự phòng, và vì các thay đổi dữ liệu sẽ

được gửi từ máy chủ này tới máy chủ khác, nên không có vấn đề với các chức năng phi tất định như random().

PostgreSQL không đưa ra dạng nhận bản này, dù sự thực hiện 2 pha của PostgreSQL (PREPARE TRANSACTION và COMMIT PREPARED) có thể được sử dụng để triển khai điều này trong mã ứng dụng hoặc phần mềm trung gian.

Commercial Solutions - Các giải pháp thương mại

Vì PostgreSQL là nguồn mở và dễ dàng được mở rộng, một số công ty đã lấy PostgreSQL và đã tạo ra các giải pháp nguồn đóng thương mại với các khả năng độc nhất và chuyển đổi dự phòng, nhân bản, và cân bằng tải.

Bảng 25-1 tóm tắt các khả năng của các giải pháp khác nhau được liệt kê ở trên.

Bảng 25-1. Ma trận tính năng về hiệu năng cao, cân bằng tải và nhân bản

Tính năng	Chuyển đổi dự phòng đĩa được chia sẻ	Nhân bản hệ thống tệp	Dự phòng nóng/ấm bằng sử dụng PITR	Nhân bản dự phòng máy chủ chính dựa vào Trigger	Nhân bản phần mềm trung gian dựa vào lệnh	Nhân bản không đồng bộ nhiều máy chủ chính	Nhân bản đồng bộ nhiều máy chủ chính
Triển khai phổ biến nhất	NAS	DRBD	PITR	Slony	pgpool-II	Bucardo	
Phương pháp giao tiếp	đĩa được chia sẻ	các khối đĩa	WAL	các hàng của bảng	SQL	các hàng của bảng	các hàng của bảng & các khóa hàng
Không yêu cầu phần cứng đặc biệt		x	x	x	x	x	x
Cho phép nhiều máy chủ chính					x	x	x
Không tổng chi phí máy chủ chính	x		x		x		
Không chờ đợi nhiều máy chủ	x		x	x		x	
Hỏng máy chủ chính sẽ không làm mất dữ liệu	x	x			x		x
Dự phòng chấp nhận các truy vấn chỉ đọc			Chỉ nóng	x	x	x	x
Độ mịn theo từng bảng				x		x	x
Không cần thiết giải quyết xung đột	x	x	x	x			x

Có một ít giải pháp không phù hợp trong các chủng loại ở trên:

Data Partitioning - Việc phân vùng dữ liệu

Việc phân vùng dữ liệu chia các bảng thành các tập hợp dữ liệu. Từng tập hợp có thể chỉ được một máy chủ sửa đổi. Ví dụ, dữ liệu có thể được các văn phòng Luân Đôn và Paris phân vùng, với một máy chủ trong từng văn phòng. Nếu việc kết hợp các truy vấn dữ liệu

của Luân Đôn và Paris là cần thiết, thì một ứng dụng có thể truy vấn cả 2 máy chủ, hoặc nhân bản máy chủ chính/dự phòng có thể được sử dụng để giữ cho một bản sao chỉ đọc các dữ liệu của văn phòng khác trong từng máy chủ.

Multiple-Server Parallel Query Execution - Thực thi truy vấn song song nhiều máy chủ

Nhiều trong số các giải pháp ở trên cho phép nhiều máy chủ điều khiển nhiều truy vấn, nhưng không giải pháp nào cho phép một truy vấn duy nhất sử dụng nhiều máy chủ để hoàn thành nhanh hơn. Giải pháp này cho phép nhiều máy chủ làm việc đồng thời trong một truy vấn duy nhất. Nó thường được hoàn tất bằng việc chia dữ liệu giữa các máy chủ và nhờ từng máy chủ thực thi phần truy vấn của mình và trả về các kết quả tới một máy chủ trung tâm nơi mà chúng sẽ được kết hợp và được trả về cho người sử dụng. Pgpool-II có khả năng này. Hơn nữa, điều này có thể được triển khai bằng việc sử dụng bộ công cụ PL/Proxy.

25.2. Xuất lưu ký các máy chủ dự phòng

Việc lưu trữ liên tục có thể được sử dụng để tạo ra một cấu hình bó máy có tính sẵn sàng cao - HA (High Availability) với một hoặc nhiều máy chủ dự phòng sẵn sàng tiếp nhận các hoạt động nếu máy chủ chính hỏng. Khả năng này được tham chiếu rộng rãi tới như là dự phòng ấm hoặc xuất xưởng lưu ký. Máy chủ chính và dự phòng làm việc cùng nhau để cung cấp khả năng này, dù các máy chủ đó chỉ được ghép lỏng lẻo với nhau. Máy chủ chính vận hành ở chế độ lưu trữ liên tục, trong khi từng máy chủ dự phòng vận hành ở chế độ phục hồi liên tục, đọc các tệp WAL từ máy chủ chính. Không có thay đổi nào đối với các bảng cơ sở dữ liệu được yêu cầu để cho phép khả năng này, nên nó đưa ra tổng chi phí quản trị thấp so với một số giải pháp nhân bản khác. Cấu hình này cũng có ảnh hưởng về hiệu năng khá thấp lên máy chủ chính.

Việc chuyển trực tiếp các bản ghi WAL từ máy chủ cơ sở dữ liệu này sang máy chủ cơ sở dữ liệu khác thường được mô tả như là việc xuất xưởng lưu ký (log shipping). PostgreSQL triển khai việc xuất xưởng lưu ký dựa vào tệp, có nghĩa là các bản ghi WAL sẽ truyền một tệp (phân đoạn WAL) ở một thời điểm. Các tệp WAL (16MB) có thể được xuất dễ dàng và rẻ qua bất kỳ khoảng cách nào, bất kể nó tới một hệ thống liền kề nào, hệ thống khác ở nơi y hệt, hoặc hệ thống khác ở nơi xa trên trái đất. Băng thông rộng được yêu cầu cho kỹ thuật này là khác nhau theo tốc độ giao dịch của máy chủ chính. Việc xuất lưu ký dựa vào bản ghi cũng có khả năng với việc nắn nhân bản dòng (xem Phần 25.2.5).

Lưu ý là việc xuất lưu ký là không đồng bộ, nghĩa là, các bản ghi WAL sẽ được xuất sau khi thực hiện giao dịch. Kết quả là, có một cửa sổ cho sự mất dữ liệu mà máy chủ chính chịu sự hỏng thê thảm; các giao dịch còn chưa được xuất sẽ bị mất. Kích cỡ cửa sổ mất dữ liệu trong việc xuất lưu ký dựa vào tệp có thể bị giới hạn bằng việc sử dụng tham số archive_timeout, nó có thể được thiết lập cỡ vài giây đồng hồ. Tuy nhiên một thiết lập thấp như vậy sẽ làm gia tăng đáng kể băng thông rộng được yêu cầu cho việc xuất tệp. Nếu bạn cần một cửa sổ ít hơn một phút hoặc khoảng đó, hãy cân nhắc sử dụng nhân bản dòng (xem Phần 25.2.5) .

Hiệu năng của sự phục hồi là đủ tốt sao cho sự dự phòng thường sẽ chỉ là các thời điểm ngoài sự sẵn sàng đầy đủ một khi nó đã được kích hoạt. Kết quả là, điều này được gọi là một cấu hình dự

phòng ấm, nó đưa ra tính sẵn sàng cao. Việc phục hồi một máy chủ từ một sao lưu cơ bản được lưu trữ và cuộn về phía trước sẽ lâu hơn đáng kể, vì thế kỹ thuật này chỉ đưa ra một giải pháp cho phục hồi thảm họa, chứ không cho tính sẵn sàng cao. Một máy chủ dự phòng cũng có thể được sử dụng cho các truy vấn chỉ đọc, trong trường hợp đó nó được gọi là một máy chủ Dự phòng Nóng. Xem Phần 25.5 để có thêm thông tin.

25.2.1. Lên kế hoạch

Thường là thông thái để tạo các máy chủ chính và dự phòng sao cho chúng càng tương tự như nhau có thể càng tốt, ít nhất từ viễn cảnh của máy chủ cơ sở dữ liệu. Đặc biệt, các tên đường dẫn có liên quan tới các không gian bảng sẽ được truyền qua các máy chủ không bị sửa đổi, nên cả máy chủ chính và dự phòng đều phải có các dường dẫn kích hoạt hệt như nhau cho các không gian bảng nếu tính năng đó được sử dụng. Hãy nhớ trong đầu rằng nếu CREATE TABLESPACE được thực thi trên máy chủ chính, thì bất kỳ điểm kích hoạt mới nào cần thiết cho nó phải được tạo ra trên máy chủ chính và tất cả các máy chủ dự phòng trước khi lệnh đó được thực thi. Phần cứng không cần phải là chính xác y hệt, nhưng kinh nghiệm chỉ ra rằng việc duy trì 2 hệ thống y hệt nhau là dễ dàng hơn so với việc duy trì 2 hệ thống không giống nhau qua vòng đời của ứng dụng và hệ thống. Trong bất kỳ trường hợp nào thì kiến trúc hệ thống phần cứng cũng phải là y hệt - xuất xưởng, ví dụ, từ một hệ thống 32 bit sang 64 bit sẽ không làm việc.

Nói chung, việc xuất lưu ký giữa các máy chủ chạy các mức phát hành PostgreSQL chính khác nhau là không thể. Đó là chính sách của Nhóm Phát triển PostgreSQL Toàn cầu (PostgreSQL Global Development Group) không thực hiện các thay đổi đối với các định dạng đĩa trong quá trình nâng cấp phiên bản phụ, nên có khả năng là việc chạy các mức phiên bản phụ khác nhau trên các máy chủ chính và dự phòng sẽ làm việc thành công. Tuy nhiên, không có hỗ trợ chính thức cho điều đó được đưa ra và bạn được khuyến cáo giữ cho các máy chủ chính và dự phòng ở mức phiên bản y hệt nhau càng nhiều càng tốt. Khi việc cập nhật cho một phiên bản phụ mới, chính sách an toàn nhất là cập nhật các máy chủ dự phòng trước - một phiên bản phụ mới có khả năng nhiều hơn để đọc các tệp CREATE TABLESPACE từ một phiên bản phụ hơn là ngược lại.

25.2.2. Hoạt động của máy chủ dự phòng

Ở chế độ dự phòng, máy chủ liên tục sử dụng WAL nhận được từ máy chủ chính. Máy chủ dự phòng có thể đọc WAL từ một lưu trữ WAL (xem restore_command) hoặc trực tiếp từ máy chủ chính qua một kết nối TCP (nhân bản dòng). Máy chủ dự phòng cũng sẽ cố phục hồi bất kỳ WAL nào được thấy trong thư mục pg_xlog của bó dự phòng. Điều đó thường xảy ra sau một sự khởi động máy chủ, khi sự dự phòng lặp lại một lần nữa WAL mà từng được nắn dòng từ máy chủ chính trước khi khởi động, nhưng bạn cũng có thể sao chép bằng tay các tệp tới pg_xlog bất kỳ lúc nào để chúng lặp lại.

Khi khởi động, sự dự phòng bắt đầu bằng việc phục hồi tất cả WAL có sẵn ở vị trí lưu trữ, gọi restore_command. Một khi nó tới được kết thúc của WAL sẵn có ở đó và restore_command hỏng, thì nó cố phục hồi bất kỳ WAL nào có sẵn trong thư mục pg_xlog. Nếu điều đó hỏng hoặc nhân bản dòng không được thiết lập cấu hình, hoặc nếu kết nối đó sau đó được kết nối, thì sự dự phòng đó quay về bước 1 và cố phục hồi tệp đó từ lưu trữ một lần nữa. Vòng lặp các cố gắng này từ lưu trữ, pg_xlog,

và qua nhân bản dòng sẽ tiếp tục cho tới khi máy chủ bị dừng lại hoặc chuyển đổi dự phòng được một tệp trigger làm bật dậy.

Chế độ dự phòng được thoát ra và máy chủ chuyển sang hoạt động bình thường, khi một tệp trigger được thấy (trigger_file). Trước khi chuyển đổi dự phòng, bất kỳ WAL nào có sẵn ngay lập tức trong lưu trữ hoặc trong pg_xlog cũng sẽ được phục hồi, nhưng không cố gắng nào được thực hiện để kết nối tới máy chủ chính đó.

25.2.3. Chuẩn bị các máy chủ chính và dự phòng

Thiết lập lưu trữ liên tục trên máy chủ chính tới một thư mục lưu trữ có khả năng truy cập được từ sự dự phòng đó, như được mô tả trong Phần 24.3. Vị trí lưu trữ sẽ truy cập được từ sự dự phòng thậm chí khi máy chủ chính bị tắt, nghĩa là, bản thân nó sẽ nằm ở trên máy chủ dự phòng hoặc máy chủ được tin cậy khác, chứ không trên máy chủ chính.

Nếu bạn muốn sử dụng nhân bản dòng, hãy thiết lập xác thực trên máy chủ chính để cho phép các kết nối nhân bản từ (các) máy chủ dự phòng; đó là, cung cấp một hoặc nhiều lối vào phù hợp trong pg_hba.conf với trường đó của cơ sở dữ liệu được thiết lập về replication. Hơn nữa, hãy đảm bảo max_wal_senders sẽ được thiết lập về một giá trị đủ lớn trong tệp cấu hình của máy chủ chính.

Hãy thực hiện sao lưu cơ bản như được mô tả ở Phần 24.3.2 để tải khởi động máy chủ dự phòng.

25.2.4. Thiết lập máy chủ dự phòng

Để thiết lập một máy chủ dự phòng, hãy phục hồi sao lưu cơ bản được thực hiện từ máy chủ chính (xem Phần 24.3.3). Hãy tạo một tệp lệnh phục hồi recovery.conf trong thư mục dữ liệu bó của sự dự phòng đó, và bật standby_mode. Hãy thiết lập restore_command về một lệnh đơn giản để sao chép các tệp từ lưu trữ WAL.

> **Lưu ý**: Không sử dụng pg_standby hoặc các công cụ đơn giản với chế độ dự phòng được xây dựng sẵn được mô tả ở đây. restore_command sẽ trả về ngay lập tức nếu tệp đó không tồn tại; máy chủ sẽ cố thử lại lệnh đó một lần nữa nếu cần. Xem Phần 25.4 về việc sử dụng các công cụ như pg_standby.

Nếu bạn muốn sử dụng nhân bản dòng, hãy điền vào primary_conninfo bằng một chuỗi kết nối libpq, bao gồm tên máy chủ host (hoặc địa chỉ IP) và bất kỳ các chi tiết nào cần thiết để kết nối tới máy chủ chính đó. Nếu máy chủ chính cần một mật khẩu để xác thực, thì mật khẩu đó cũng cần phải được chỉ định trong primary_conninfo. Nếu bạn đang thiết lập máy chủ dự phòng cho các mục đích về tính sẵn sàng cao, hãy thiết lập WAL cho việc lưu trữ, các kết nối và xác thực như máy chủ chính, vì máy chủ dự phòng sẽ làm việc như một máy chủ chính sau chuyển đổi dự phòng. Bạn cũng sẽ cần thiết lập trigger_file để làm cho nó có khả năng chuyển đổi dự phòng. Nếu bạn đang thiết lập một máy chủ dự phòng cho các mục đích báo cáo mà không có các kế hoạch chuyển đổi dự phòng cho nó, thì trigger_file sẽ được yêu cầu.

Nếu bạn đang sử dụng một lưu trữ WAL, thì kích cỡ của nó có thể được tối thiểu hóa bằng việc sử dụng tham số archive_cleanup_command để loại bỏ các tệp không còn được máy chủ dự phòng yêu cầu nữa. Tiện ích pg_archivecleanup được thiết kế đặc biệt để được sử dụng với archive_cleanup_command trong các cấu hình dự phòng duy nhất điển hình, xem Phần F.22. Tuy

nhiên, lưu ý là nếu bạn đang sử dụng lưu trữ đó cho các mục đích sao lưu, thì bạn cần giữ lại các tệp cần thiết để phục hồi từ ít nhất bản sao lưu cơ bản mới nhất, thậm chí nếu chúng không còn cần thiết đối với sự dự phòng nữa.

Một ví dụ đơn giản của recovery.conf là:

```
standby_mode = 'on'
primary_conninfo = 'host=192.168.1.50 port=5432 user=foo password=foopass'
restore_command = 'cp /path/to/archive/%f %p'
trigger_file = '/path/to/trigger_file'
archive_cleanup_command = 'pg_archivecleanup /path/to/archive %r'
```

Bạn có thể có bất kỳ số lượng máy chủ dự phòng nào, nhưng nếu bạn sử dụng nhân bản dòng, hãy chắc chắn là bạn thiết lập max_wal_senders đủ cao trong máy chủ chính để cho phép chúng được kết nối đồng thời.

25.2.5. Nhân bản dòng

Nhân bản dòng cho phép một máy chủ dự phòng được cập nhật nhiều hơn có thể với việc xuất lưu ký dựa vào tệp. Sự dự phòng kết nối tới máy chủ chính, nó nắn dòng các bản ghi WAL về sự dự phòng đó khi chúng được tạo ra, không chờ cho tệp WAL đó được điền.

Nhân bản dòng là không đồng bộ, nên vẫn có một độ trễ nhỏ giữa việc thực hiện một giao dịch trong máy chủ chính và cho các thay đổi trở thành nhìn thấy được trong sự dự phòng đó. Tuy nhiên sự trễ đó nhỏ hơn nhiều so với việc xuất lưu ký dựa vào tệp, thường dưới 1 giây với giả thiết sự dự phòng là đủ mạnh để giữ được tải đó. Với nhân bản dòng, archive_timeout sẽ không được yêu cầu giảm cửa sổ mất mát dữ liệu.

Nếu bạn sử dụng nhân bản dòng mà không có việc lưu trữ liên tục dựa vào tệp, thì bạn phải thiết lập wal_keep_segments trong máy chủ chính một giá trị đủ cao để đảm bảo rằng các phân đoạn WAL cũ không được tái chế quá sớm, trong khi sự dự phòng có thể vẫn còn cần chúng để bắt kịp. Nếu sự dự phòng ở lại sau quá nhiều, thì nó cần phải được khởi tạo lại từ một sao lưu cơ bản mới. Nếu bạn thiết lập một lưu trữ WAL mà truy cập được từ sự dự phòng đó, thì wal_keep_segments không được yêu cầu khi sự dự phòng đó luôn có thể sử dụng lưu trữ đó để bắt kịp.

Để sử dụng nhân bản dòng, hãy thiết lập một máy chủ dự phòng xuất xưởng lưu ký dựa vào tệp như được mô tả trong Phần 25.2. Bước mà biến một dự phòng xuất xưởng lưu ký dựa vào tệp thành dự phòng nhân bản dòng đang thiết lập cấu hình cho thiết lập primary_conninfo trong tệp recovery.conf để trỏ tới máy chủ chính. Hãy thiết lập các lựa chọn xác thực và listen_addresses (xem pg_hba.conf) trong máy chủ chính sao cho máy chủ dự phòng có thể kết nối tới cơ sở dữ liệu giả lập trong máy chủ chính (xem Phần 25.2.5.1).

Trong các hệ thống có hỗ trợ lựa chọn giữ lại socket (keepalive socket), việc thiết lập tcp_keepalives_idle, tcp_keepalives_interval và tcp_keepalives_count giúp máy chủ chính lưu ý nhắc nhở một kết nối bị đứt đoạn.

Hãy thiết lập số các kết nối đồng thời cực đại từ các máy chủ dự phòng (xem max_wal_senders để có các chi tiết).

Khi sự dự phòng được khởi động và primary_conninfo được thiết lập đúng, sự dự phòng sẽ kết nối tới máy chủ chính sau khi chơi lại tất cả các tệp WAL sẵn có trong lưu trữ đó. Nếu kết nối đó được thiết lập thành công, thì bạn sẽ thấy một tiến trình tiếp nhận WAL (walreiceiver process) trong dự phòng đó, và một tiến trình gửi WAL (walsender process) trong máy chủ chính.

25.2.5.1. Xác thực

Rất quan trọng là các quyền ưu tiên truy cập cho nhân bản được thiết lập sao cho chỉ những người sử dụng được tin cậy có thể đọc được dòng WAL, vì nó dễ trích các thông tin có quyền ưu tiên từ đó.

Các máy chủ dự phòng phải xác thực trước hết như là một tài khoản của siêu người sử dụng. Vì thế một vai trò với các quyền ưu tiên SUPERUSER và LOGIN cần phải được tạo ra trong máy chủ chính.

Xác thực máy trạm cho sự nhân bản được kiểm soát bằng một bản ghi pg_hba.conf chỉ định sự nhân bản trong trường database đó. Ví dụ, nếu sự dự phòng đang chạy trên IP máy chủ host 192.168.1.100 và tên của siêu người sử dụng cho sự nhân bản đó là foo, thì người quản trị có thể bổ sung thêm dòng sau đây vào tệp pg_hba.conf trên máy chủ chính:

```
# Allow the user "foo" from host 192.168.1.100 to connect to the primary
# as a replication standby if the user's password is correctly supplied.
#
# TYPE          DATABASE        USER    CIDR-ADDRESS            METHOD
host            replication     foo     192.168.1.100/32        md5
```

Tên máy chủ host và số cổng của máy chủ chính, tên người sử dụng kết nối, và mật khẩu được chỉ định trong tệp recovery.conf. Mật khẩu cũng có thể được thiết lập trong tệp ~/.pgpass để dự phòng (chỉ định trong trường database). Ví dụ, nếu máy chủ chính đang chạy trong địa chỉ host IP 192.168.1.50, cổng 5432, tên của siêu người sử dụng đối với nhân bản là foo, và mật khẩu là foopass, thì người quản trị có thể thêm dòng sau đây vào tệp recovery.conf để dự phòng:

```
# The standby connects to the primary that is running on host 192.168.1.50
# and port 5432 as the user "foo" whose password is "foopass".
primary_conninfo = 'host=192.168.1.50 port=5432 user=foo password=foopass'
```

25.2.5.2. Giám sát

Một chỉ số lành mạnh quan trọng của nhân bản dòng là số lượng các bản ghi WAL được sinh ra trong máy chủ chính, mà chưa được áp dụng trong sự dự phòng. Bạn có thể tính toán sự trễ bằng việc so sánh vị trí ghi WAL trong máy chủ chính với vị trí WAL mới nhất nhận được bởi sự dự phòng. Chúng có thể được truy vấn bằng việc sử dụng pg_current_xlog_location trong máy chủ chính và pg_last_xlog_receive_location trong sự dự phòng, một cách tương ứng (xem Bảng 9-56 và Bảng 9-57 để có các chi tiết). Ví trí nhận được mới nhất của WAL trong sự dự phòng đó cũng được hiển thị trong tình trạng tiến trình của tiến trình nhận WAL, được hiển thị bằng việc sử dụng lệnh ps (xem Phần 27.1 để có các chi tiết).

25.3. Chuyển đổi dự phòng

Nếu máy chủ chính hỏng và sau đó máy chủ dự phòng sẽ bắt đầu các thủ tục chuyển đổi dự phòng.

Nếu máy chủ dự phòng hỏng và sau đó không sự chuyển đổi dự phòng nào cần diễn ra. Nếu máy

chủ dự phòng có thể được khởi động lại, thậm chí khi nào đó sau đó, thì tiến trình phục hồi cũng có thể được khởi động lại ngay lập tức, tận dụng sự phục hồi có khả năng khởi động lại được. Nếu máy chủ dự phòng không thể được khởi động lại, thì sau đó một cài đặt máy chủ dự phòng mới sẽ được tạo ra.

Nếu máy chủ chính hỏng và máy chủ dự phòng trở thành máy chủ chính mới, và sau đó máy chủ chính cũ khởi động lại, thì bạn phải có một cơ chế cho việc thông báo cho máy chủ chính cũ rằng nó không còn là máy chủ chính nữa. Điều này đôi khi được biết tới như là đá một nút khác vào trong đầu - STONITH (Shoot The Other Node In The Heaed), nó là cần thiết để tránh các tình huống nơi mà cả 2 hệ thống đều nghĩ chúng là máy chủ chính, điều sẽ dẫn tới sự lúng túng và cuối cùng là mất dữ liệu.

Nhiều hệ thống chuyển đổi dự phòng sử dụng chỉ 2 hệ thống, chính và dự phòng, được kết nối bằng một vài dạng cơ chế nhịp tim để kiểm tra hợp lệ liên tục tính kết nối giữa 2 thứ đó và khả năng sống được của máy chủ chính. Cũng có khả năng sử dụng một hệ thống thứ 3 (được gọi là một máy chủ làm chứng) để ngăn chặn vài trường hợp chuyển đổi dự phòng không phù hợp, nhưng sự phức tạp bổ sung có thể không đáng trừ phi nó được thiết lập với sự chăm sóc đầy đủ và kiểm thử chặt chẽ.

PostgreSQL không cung cấp phần mềm hệ thống được yêu cầu để nhận diện một sự hỏng hóc trong máy chủ chính và thông báo cho máy chủ cơ sở dữ liệu dự phòng. Nhiều công cụ như vậy tồn tại và được tích hợp tốt với các tiện ích hệ điều hành được yêu cầu cho sự chuyển đổi dự phòng thành công, như sự chuyển đổi địa chỉ IP.

Một khi chuyển đổi dự phòng sang dự phòng xảy ra, chỉ có một máy chủ duy nhất hoạt động. Điều này được biết tới như là một tình trạng suy biến. Máy dự phòng trước đó bây giờ là chính, còn máy chính trước đó bị tắt và có thể vẫn bị tắt. Để chuyển về hoạt động bình thường, một máy chủ dự phòng phải được tái tạo lại, hoặc trong hệ thống chính trước đó khi nó tới, hoặc trong một máy thứ 3, có khả năng là hệ thống mới. Một khi hoàn toàn máy chủ chính và dự phòng có thể được xem là đã chuyển đổi vài trò cho nhau. Một vài người chọn sử dụng một máy chủ thứ 3 để cung cấp sao lưu cho máy chủ chính mới cho tới khi máy chủ dự phòng mới được tái tạo, dù rõ ràng điều này làm phức tạp hóa cấu hình hệ thống và các tiến trình hoạt động.

Vì thế, việc chuyển từ máy chủ chính sang dự phòng có thể nhanh nhưng đòi hỏi một số thời gian để chuẩn bị trước bó chuyển đổi dự phòng. Việc chuyển đổi thường xuyên từ máy chủ chính sang dự phòng là hữu dụng, vì nó cho phép gián đoạn thường xuyên trong từng hệ thống để duy trì. Điều này cũng phục vụ như một kiểm thử cơ chế chuyển đổi dự phòng để đảm bảo rằng nó sẽ thực sự làm việc khi bạn cần nó. Các thủ tục quản trị bằng văn bản được khuyến cáo.

Để kích hoạt sự chuyển đổi dự phòng của một máy chủ dự phòng xuất xưởng lưu ký, hãy tạo một tệp trigger với tên tệp và đường dẫn được `trigger_file` trong `recovery.conf` chỉ định. Nếu `trigger_file` không được đưa ra, sẽ không có cách nào để thoát khỏi sự phục hồi trong sự dự phòng và đưa nó thành một máy chủ chính. Điều đó có thể là hữu dụng cho việc báo cáo cho các máy chủ sẽ chỉ được sử dụng để rút các truy vấn chỉ đọc khỏi máy chủ chính, không vì các mục đích sẵn sàng cao.

25.4. Phương pháp lựa chọn thay thế cho việc xuất xưởng lưu ký

Một lựa chọn thay thế cho chế độ xây dựng trong dự phòng được mô tả trong các phần trước là để sử dụng một restore_command khảo sát vị trí lưu trữ. Điều này từng là lựa chọn duy nhất sẵn có trong các phiên bản 8.4 hoặc trước đó. Trong thiết lập này, hãy thiết lập standby_mode tắt (off), vì bản thân bạn đang triển khai việc khảo sát được yêu cầu cho hoạt động dự phòng. Xem contrib/pg_standby (Phần F.28) về triển khai tham chiếu của điều này.

Lưu ý rằng trong chế độ này, máy chủ sẽ sử dụng một tệp WAL mỗi lần, sao cho nếu bạn sử dụng máy chủ dự phòng cho các truy vấn (xem Dự phòng Nóng - Hot Standby), sẽ có một sự trễ giữa một hành động trong máy chủ chính và khi hành động đó trở nên nhìn thấy được trong dự phòng đó, tương ứng thời gian mà nó lấy để điền tệp WAL.

archive_timeout có thể được sử dụng để làm cho sự trễ đó ngắn hơn. Hơn nữa lưu ý rằng bạn không thể kết hợp nhân bản dòng với phương pháp này.

Các hoạt động xảy ra trong cả các máy chủ chính và dự phòng là các tác vụ phục hồi và lưu trữ liên tục bình thường. Điểm liên lạc duy nhất giữa 2 máy chủ cơ sở dữ liệu là lưu trữ các tệp WAL mà cả 2 cùng chia sẻ: máy chính ghi vào lưu trữ đó, máy dự phòng đọc từ lưu trữ đó. Sự thận trọng cần được thực hiện để đảm bảo rằng các lưu trữ WAL từ các máy chủ chính riêng rẽ không trộn vào nhau hoặc gây lúng túng. Lưu trữ đó cần không lớn nếu nó chỉ được yêu cầu cho hoạt động dự phòng.

Sự huyền ảo làm cho 2 máy chủ được ghép lỏng lẻo với nhau đó đơn giản là một restore_command được sử dụng trong dự phòng mà, khi được yêu cầu cho tệp WAL tiếp sau, sẽ chờ nó trở nên sẵn sàng từ máy chủ chính. restore_command được chỉ định trong tệp recovery.conf trong máy chủ dự phòng. Việc xử lý phục hồi bình thường có thể yêu cầu một tệp từ lưu trữ WAL, nói hỏng nếu tệp đó không sẵn sàng. Đối với việc xử lý dự phòng, là thông thường đối với tệp WAL tiếp sau là không sẵn sàng, sao cho sự dự phòng phải chờ nó xuất hiện. Đối với các tệp kết thúc trong .backup hoặc .history không có nhu cầu phải chờ, và một mã trả về không bằng zero phải được trả về. Một restore_command chờ có thể được viết như một script tùy ý mà lặp lại sau khi khảo sát sự tồn tại của tệp WAL tiếp sau. Cũng phải có một vài cách để làm bật dậy sự chuyển đổi dự phòng, điều sẽ làm gián đoạn restore_command, phá vỡ vòng lặp và trả về một lỗi không tìm thấy tệp (file not found) cho máy chủ dự phòng. Điều này kết thúc sự phục hồi và sự dự phòng sau đó sẽ tới như một máy chủ bình thường.

Mã giả lập (pseudocode) cho một lệnh restore_command phù hợp là:

```
triggered = false;
while (!NextWALFileReady() && !triggered)
{
        sleep(100000L); /* wait for ~0.1 sec */
        if (CheckForExternalTrigger())
                triggered = true;
}
if (!triggered)
                CopyWALFileForRecovery();
```

Một ví dụ làm việc của một restore_command chờ đợi được cung cấp như một module contrib có tên là pg_standby. Nó sẽ được sử dụng như một tham chiếu về cách triển khai đúng logic được mô tả ở trên. Nó cũng có thể được mở rộng như cần thiết để hỗ trợ cho các cấu hình và môi trường đặc thù.

Phương pháp cho việc làm bật dậy sự chuyển đổi dự phòng đó là một phần quan trọng của việc lên kế hoạch và thiết kế. Một lựa chọn tiềm tàng là lệnh restore_command. Nó được thực thi một lần cho từng tệp WAL, nhưng tiến trình chạy restore_command đó được tạo ra và chế cho từng tệp, nên sẽ không có tiến trình daemon hoặc máy chủ, và các tín hiệu hoặc một trình điều khiển tín hiệu không thể được sử dụng. Vì thế, restore_command là không phù hợp để làm bật dậy sự chuyển đổi dự phòng. Có khả năng sử dụng một tiện ích timeout đơn giản, đặc biệt nếu được sử dụng trong sự kết hợp với một thiết lập archive_timeout trên máy chủ chính. Tuy nhiên, điều này là thứ gì đó có xu hướng lỗi vì một vấn đề mạng hoặc máy chủ chính bận có thể là đủ để khởi tạo một sự chuyển đổi dự phòng. Một cơ chế thông báo như sự tạo ra rõ ràng một tệp trigger là lý tưởng, nếu điều này có thể dàn xếp có được.

25.4.1. Triển khai

Thủ tục ngắn cho việc thiết lập cấu hình một máy chủ dự phòng bằng việc sử dụng phương pháp lựa chọn này là như sau. Để có các chi tiết đầy đủ của từng bước, hãy tham chiếu tới các phần trước như được lưu ý.

1. Thiết lập các hệ thống chính và dự phòng càng gần lý tưởng càng tốt, bao gồm 2 bản sao y hệt nhau của PostgreSQL ở cùng một mức phiên bản.
2. Thiết lập việc lưu trữ liên tục từ máy chính về một thư mục lưu trữ WAL trong máy dự phòng. Hãy đảm bảo rằng archive_mode, archive_command và archive_timeout được thiết lập đúng phù hợp trên máy chủ chính (xem Phần 24.3.1).
3. Tiến hành một sao lưu cơ bản máy chủ chính (xem Phần 24.3.2), và tải dữ liệu này vào dự phòng.
4. Bắt đầu phục hồi trong máy chủ dự phòng từ lưu trữ cục bộ WAL, bằng việc sử dụng một recovery.conf để chỉ định một restore_command chờ như được mô tả trước đó (xem Phần 24.3.3). Sự phục hồi có lưu trữ WAL như là chỉ đọc, nên một khi một tệp WAL được sao chép tới hệ thống dự phòng thì nó có thể được sao chép vào băng từ cùng một lúc như nó đang được máy chủ cơ sở dữ liệu dự phòng đọc.

Vì vậy, việc chạy một máy chủ dự phòng cho tính sẵn sàng cao có thể được thực hiện cùng lúc khi các tệp được lưu trữ cho những mục đích phục hồi thảm họa dài hạn.

Vì các mục đích kiểm thử, có khả năng chạy cả các máy chủ chính và dự phòng trong cùng một hệ thống. Điều này không đưa ra bất kỳ sự cải thiện đáng giá nào trong sự cường tráng máy chủ, cũng không được mô tả như là có tính sẵn sàng cao - HA (High Availability).

25.4.2. Xuất xưởng lưu ký dựa vào bản ghi

Cũng có khả năng để triển khai việc xuất xưởng lưu ký dựa vào bản ghi bằng việc sử dụng phương pháp lựa chọn này, dù điều này đòi hỏi sự phát triển tùy biến, và các thay đổi sẽ chỉ trở nên nhìn thấy được đối với các truy vấn dự phòng nóng sau khi một tệp WAL đầy đủ được xuất.

Một chương trình bên ngoài có thể gọi hàm pg_xlogfile_name_offset() (xem Phần 9.24) để tìm ra tên tệp và phần bù trừ byte chính xác bên trong nó đối với kết thúc WAL hiện hành. Nó sau đó có thể truy cập tệp WAL trực tiếp và sao chép các dữ liệu từ sự kết thúc WAL được biết mới nhất qua sự kết thúc hiện hành đối với các máy chủ dự phòng. Với tiếp cận này, cửa sổ mất dữ liệu là vòng đời khảo sát của chương trình sao chép, nó có thể rất nhỏ, và không có băng thông rộng bị bỏ phí từ việc ép các tệp phân khúc được sử dụng một phần sẽ được lưu trữ. Lưu ý rằng các script restore_command của các máy chủ dự phòng chỉ có thể làm việc với toàn bộ các tệp WAL, nên các dữ liệu được sao chép dần sẽ không được làm cho sẵn sàng một cách thông thường tới các máy chủ dự phòng. Nó chỉ được sử dụng khi máy chính chết - sau đó một phần tệp WAL mới nhất được đưa vào chế độ dự phòng trước khi cho phép nó xuất hiện. Triển khai đúng tiến trình này đòi hỏi sự cộng tác của script restore_command với chương trình sao chép dữ liệu.

Việc bắt đầu với PostgreSQL phiên bản 9.0, bạn có thể sử dụng nhân bản dòng (xem Phần 25.2.5) để lưu trữ những lợi ích y hệt với nỗ lực ít hơn.

25.5. Dự phòng nóng

Dự phòng nóng là khái niệm được sử dụng để mô tả khả năng kết nối máy chủ và chạy các truy vấn chỉ đọc trong khi máy chủ đó ở trong chế độ phục hồi hoặc dự phòng lưu trữ. Điều này là hữu dụng cả cho các mục đích nhân bản và cho việc phục hồi một sao lưu về một tình trạng mong muốn với độ chính xác lớn. Khái niệm Dự phòng Nóng (Hot Standby) cũng tham chiếu tới khả năng máy chủ chuyển từ phục hồi sang hoạt động bình thường trong khi những người sử dụng tiếp tục chạy các truy vấn và/hoặc giữ các kết nối của họ là mở.

Việc chạy các truy vấn trong chế độ dự phòng nóng là tương tự với hoạt động truy vấn thông thường, dù có vài sự khác biệt về sử dụng và quản trị được giải thích ở bên dưới.

25.5.1. Tổng quan về người sử dụng

Khi tham số hot_standby được thiết lập về đúng (true) trên một máy chủ dự phòng, nó sẽ bắt đầu chấp nhận các kết nối một khi sự phục hồi đã mang hệ thống về tình trạng ổn định. Tất cả các kết nối như vậy là chỉ đọc một cách chặt chẽ; thậm chí không bảng tạm nào có thể được ghi.

Dữ liệu trong dự phòng mất một chút thời gian để tới được từ máy chủ chính nên sẽ có một sự trễ có thể đo đếm được giữa máy chủ chính và dự phòng. Việc chạy cùng một truy vấn gần như cùng đồng thời trên cả máy chính và dự phòng có thể vì thế trả về các kết quả khác nhau. Chúng tôi nói rằng các dữ liệu trong dự phòng cuối cùng là ổn định với máy chính. Một khi bản ghi được thực hiện cho một giao dịch được chơi lại trong dự phòng, thì những thay đổi được giao dịch đó thực hiện sẽ là nhìn thấy được đối với bất kỳ hình chụp mới nào được thực hiện trong máy dự phòng. Các hình chụp có thể được thực hiện ở đầu của từng truy vấn hoặc ở đầu của từng giao dịch, phụ thuộc vào mức độ cách li giao dịch hiện hành. Để có thêm chi tiết, xem Phần 13.2.

Các giao dịch được bắt đầu trong quá trình dự phòng nóng có thể đưa ra các lệnh sau:

- Truy cập truy vấn - SELECT, COPY TO
- Các lệnh con trỏ - DECLARE, FETCH, CLOSE

- Các tham số - SHOW, SET, RESET
- Các lệnh quản lý giao dịch
  - BEGIN, END, ABORT, START TRANSACTION
  - SAVEPOINT, RELEASE, ROLLBACK TO SAVEPOINT
  - EXCEPTION khóa và các giao dịch phụ nội bộ khác
- LOCK TABLE, dù chỉ khi rõ ràng trong một trong các chế độ: ACCESS SHARE, ROW SHARE hoặc ROW EXCLUSIVE.
- Các kế hoạch và tài nguyên - PREPARE, EXECUTE, DEALLOCATE, DISCARD
- Các trình cài cắm và các mở rộng - LOAD

Các giao dịch được bắt đầu trong quá trình dự phòng nóng sẽ không bao giờ được chỉ định cho một ID giao dịch và không thể ghi vào lưu ký ghi trước của hệ thống. Vì thế, các hành động sau đây sẽ sinh ra các thông điệp lỗi:

- Ngôn ngữ Điều khiển Dữ liệu - DML (Data Manipulation Language) - INSERT, UPDATE, DELETE, COPY FROM, TRUNCATE. Lưu ý là có các hành động không được phép làm cho một trigger thực thi trong quá trình phục hồi. Hạn chế này thậm chí được áp dụng cho các bảng tạm, vì các hàng của bảng không thể đọc hoặc ghi được mà không có việc chỉ định một ID giao dịch, điều hiện là không thể trong môi trường dự phòng nóng.
- Ngôn ngữ Định nghĩa Dữ liệu - DDL (Data Definition Language) - CREATE, DROP, ALTER, COMMENT. Hạn chế này áp dụng thậm chí cho các bảng tạm, vì việc triển khai các hoạt động đó có thể đòi hỏi việc cập nhận các bảng catalog hệ thống.
- SELECT ... FOR SHARE | UPDATE, vì sự khóa hàng không thể được thực hiện mà không có việc cập nhật các tệp dữ liệu nằm bên dưới.
- Các qui tắc trong các lệnh SELECT mà sinh ra các lệnh DML.
- LOCK mà rõ ràng yêu cầu một chế độ cao hơn so với ROW EXCLUSIVE MODE.
- LOCK ở dạng mặc định ngắn, vì nó đòi hỏi ACCESS EXCLUSIVE MODE.
- Các lệnh quản lý giao dịch mà rõ ràng thiết lập tình trạng không chỉ đọc:
  - BEGIN READ WRITE, START TRANSACTION READ WRITE.
  - SET TRANSACTION READ WRITE , SET SESSION CHARACTERISTICS AS TRANSACTION READ WRITE
  - SET transaction_read_only = off
- Các lệnh thực hiện 2 pha - PREPARE TRANSACTION , COMMIT PREPARED , ROLLBACK PREPARED vì thậm chí các giao dịch chỉ đọc cũng cần phải ghi WAL trong pha chuẩn bị (pha đầu của 2 pha thực hiện).
- Các bản cập nhật tuần tự - nextval() , setval()
- LISTEN , UNLISTEN , NOTIFY

Trong hoạt động thông thường, các giao dịch "chỉ đọc" sẽ được phép cập nhật các tuần tự và sử dụng và, sao cho các phiên làm việc Dự phòng Nóng vận hành dưới các hạn chế khá chặt chẽ so với các phiên làm việc chỉ đọc thông thường. Có khả năng là một vài hạn chế đó có thể được nới lỏng trong một phiên bản trong tương lai.

Trong quá trình dự phòng nóng, tham số transaction_read_only luôn đúng và có thể không bị thay đổi. Nhưng miễn là không cố gắng nào được thực hiện để sửa đổi cơ sở dữ liệu đó, các kết nối trong quá trình dự phòng nóng sẽ hành động rất giống với bất kỳ kết nối cơ sở dữ liệu nào khác. Nếu sự chuyển đổi dự phòng hoặc sự chuyển qua xảy ra, thì cơ sở dữ liệu sẽ chuyển sang chế độ xử lý thông thường. Các phiên làm việc sẽ vẫn được kết nối trong khi máy chủ thay đổi chế độ. Một khi dự phòng nóng kết thúc, sẽ có khả năng khởi tạo các giao dịch chỉ ghi (thậm chí từ một phiên làm việc được bắt đầu trong quá trình dự phòng nóng).

Những người sử dụng có khả năng nói liệu phiên làm việc của họ có là chỉ đọc hay không bằng việc đưa ra SHOW transaction_read_only. Hơn nữa, một tập hợp các hàm (Bảng 9-57) cho phép những người sử dụng truy cập thông tin về máy chủ dự phòng. Chúng cho phép bạn viết các chương trình sẽ nhận biết được về tình trạng hiện hành của cơ sở dữ liệu. Chúng có thể được sử dụng để giám sát tiến trình phục hồi, hoặc cho phép bạn viết các chương trình phức tạp phục hồi cơ sở dữ liệu về các tình trạng nhất định.

25.5.2. Điều khiển các xung đột truy vấn

Các máy chủ chính và dự phòng theo nhiều cách được kết nối lỏng lẻo với nhau. Các hành động trong máy chính sẽ có một ảnh hưởng lên máy dự phòng. Kết quả là, có tiềm năng cho các tương tác hoặc các xung đột tiêu cực giữa chúng. Xung đột dễ dàng nhất để hiểu là hiệu năng: nếu một tải dữ liệu lớn diễn ra trong máy chính thì điều này sẽ sinh ra một dòng tương tự các bản ghi WAL trong máy dự phòng, nên các truy vấn dự phòng có thể tranh giành các tài nguyên hệ thống, như I/O.

Cũng có các dạng xung đột bổ sung có thể xảy ra với Dự phòng Nóng. Các xung đột đó là các xung đột nặng theo nghĩa là các truy vấn có thể cần phải được hoãn và, trong một số trường hợp, các phiên không được kết nối để giải quyết chúng. Người sử dụng được cung cấp vài cách thức để điều khiển các xung đột đó. Các trường hợp xung đột gồm:

- Sự khóa hoàn toàn truy cập được thực hiện trong máy chủ chính, bao gồm cả các lệnh LOCK rõ ràng và các hoạt động DDL khác nhau, xung đột với các truy cập bảng trong các truy vấn dự phòng.
- Việc loại bỏ một không gian bảng trong các xung đột máy chính với các truy vấn máy dự phòng bằng việc sử dụng không gian bảng cho các tệp làm việc tạm thời.
- Việc loại bỏ một cơ sở dữ liệu trong các xung đột máy chính với các phiên làm việc được kết nối tới cơ sở dữ liệu đó trong máy dự phòng.
- Ứng dụng của một bản ghi được làm sạch bằng hút chân không từ các xung đột WAL với các giao dịch dự phòng mà các hình chụp của nó có thể vẫn "thấy" bất kỳ hàng nào bị loại bỏ.
- Ứng dụng của một bản ghi làm sạch bằng hút chân không từ các xung đột WAL với các truy

vấn truy cập trang đích trong dự phòng, bất kể có hay không dữ liệu bị loại bỏ được thấy.

Trên máy chủ chính, các trường hợp đó đơn giản gây ra sự chờ đợi: và người sử dụng có thể chọn hoãn hoặc có sự xung đột các hành động. Tuy nhiên, trong máy dự phòng không có sự lựa chọn: hành động WAL được lưu ký đã xảy ra rồi trong máy chủ chính nên máy chủ dự phòng phải không bị hỏng để sử dụng nó. Hơn nữa, việc cho phép ứng dụng WAL chờ vô hạn định có thể rất không mong đợi, vì tình trạng dự phòng sẽ ngày càng trở nên tụt xa đằng sau tình trạng của máy chủ chính. Vì thế, một cơ chế được đưa ra để ép hoãn các truy vấn dự phòng mà xung đột với các bản ghi WAL sẽ được sử dụng.

Một ví dụ của tình huống có vấn đề đó là một quản trị viên trong máy chủ chính chạy DROP TABLE trong một bảng hiện đang được truy vấn trong máy chủ dự phòng. Rõ ràng truy vấn dự phòng không thể tiếp tục nếu DROP TABLE được sử dụng trong máy dự phòng. Nếu tình huống này xảy ra trong máy chủ chính, thì DROP TABLE có thể chờ cho tới khi truy vấn khác đã kết thúc. Nhưng khi DROP TABLE được chạy trong máy chủ chính, thì máy chính này không có thông tin về các truy vấn nào đang chạy trong máy dự phòng, nên nó sẽ không chờ bất kỳ truy vấn dự phòng nào như vậy. Các bản ghi WAL thay đổi đi qua tới máy dự phòng trong khi truy vấn dự phòng vẫn đang chạy, gây ra một xung đột. Máy chủ dự phòng phải hoặc là làm trễ ứng dụng các bản ghi WAL (và cả mọi điều sau chúng nữa) hoặc nếu không thì hoãn truy vấn xung đột để DROP TABLE có thể được sử dụng.

Khi một truy vấn xung đột là ngắn, thường mong muốn để cho phép nó hoàn tất bằng việc làm trễ ứng dụng WAL một chút; nhưng một sự trễ dài trong ứng dụng WAL thường không được mong đợi. Do đó cơ chế hoãn có tham số, max_standby_archive_delay và max_standby_streaming_delay, chúng xác định sự trễ cực đại được phép trong ứng dụng WAL. Các truy vấn xung đột sẽ bị hoãn một khi nó đã xảy ra lâu hơn so với thiết lập trễ thích đáng để áp dụng bất kỳ dữ liệu WAL mới được nhận nào. Có 2 tham số sao cho các giá trị trễ khác nhau có thể được chỉ định cho trường hợp việc đọc dữ liệu WAL từ một lưu trữ (như, phục hồi ban đầu từ một sao lưu cơ bản hoặc "việc nắm bắt được" một máy chủ dự phòng mà nằm xa lại sau) so với việc đọc dữ liệu WAL qua nhân bản hàng.

Trong một máy chủ dự phòng mà tồn tại trước hết vì tính sẵn sàng cao, tốt nhất để thiết lập các tham số trễ khá ngắn, sao cho máy chủ không thể nằm xa lại sau máy chủ chính vì các sự trễ do các truy vấn dự phòng gây ra. Tuy nhiên, nếu máy chủ dự phòng có ý để thực thi các truy vấn chạy lâu thời gian, thì một giá trị trễ cao hoặc thậm chí vô định có thể được ưu tiên. Tuy nhiên hãy nhớ là một truy vấn chạy lâu có thể làm cho các phiên làm việc khác trong máy chủ dự phòng không thấy được những thay đổi gần đây trong máy chính, nếu nó làm trễ ứng dụng các bản ghi của WAL.

Lý do thường thấy nhất đối với xung đột giữa các truy vấn dự phòng và sự chơi lại WAL là "sự làm sạch sớm". Thông thường, PostgreSQL cho phép làm sạch các phiên bản hàng cũ khi không có các giao dịch cần phải xem chúng để đảm bảo khả năng nhìn thấy đúng các dữ liệu theo các qui tắc của MVCC. Tuy nhiên, qui tắc này cũng có thể được áp dụng cho các giao dịch thực thi trên máy chủ chính. Vì thế có khả năng sự làm sạch trong máy chủ chính sẽ loại bỏ các phiên bản hàng mà vẫn còn thấy được đối với một giao dịch trong máy dự phòng.

Những người sử dụng có kinh nghiệm nên lưu ý rằng cả sự làm sạch phiên bản hàng và việc làm đông cứng phiên bản hàng sẽ tiềm tàng xung đột với các truy vấn dự phòng. Việc chạy một VACUUM

FREEZE bằng tay có khả năng gây ra các xung đột trong các bảng thậm chí khi không có các hàng được cập nhật hay bị xóa.

Một khi sự trễ được max_standby_archive_delay hoặc max_standby_streaming_delay chỉ định đã bị vượt quá, các truy vấn xung đột sẽ bị hoãn. Điều này thường chỉ gây ra một sự hoãn lỗi, dù trong trường hợp chơi lại DROP DATABASE thì toàn bộ phiên làm việc xung đột sẽ được kết thúc.

Hơn nữa, nếu xung đột vượt qua một khóa do một giao dịch nhàn rỗi nắm giữ, thì phiên làm việc xung đột sẽ được kết thúc (hành vi này có thể thay đổi trong tương lai).

Các truy vấn được hoãn có thể sẽ được thử lại ngay lập tức (sau khi bắt đầu một giao dịch mới, tất nhiên). Vì sự hoãn truy vấn phụ thuộc vào bản chất tự nhiên các bản ghi WAL đang được chơi lại, một truy vấn từng bị hoãn có thể cũng thành công tốt nếu nó được thực thi lại một lần nữa.

Hãy nhớ rằng các tham số trễ sẽ được so sánh với thời gian trôi đi kể từ khi dữ liệu WAL đã được máy chủ dự phòng nhận được. Vì thế, giai đoạn dễ chịu được phép cho bất kỳ một truy vấn nào trên máy chủ dự phòng cũng không bao giờ lớn hơn tham số trễ, và có thể ít hơn đáng kể nếu máy chủ dự phòng đã nằm đằng sau rồi như là một kết quả của việc chờ đợi các truy vấn trước đó hoàn tất, hoặc như một kết quả của việc không có khả năng để theo kịp một tải cập nhật lớn.

Những người sử dụng nên rõ ràng rằng các bảng sẽ được cập nhật thường xuyên và nặng nề trong máy chủ chính sẽ nhanh chóng gây ra sự hoãn các truy vấn dài hơn đang chạy trong máy chủ dự phòng. Trong các trường hợp như vậy thì việc thiết lập một giá trị hữu hạn cho max_standby_archive_delay hoặc max_standby_streaming_delay có thể được cân nhắc tương tự như việc thiết lập statement_timeout.

Các khả năng sửa là tồn tại nếu số các vụ hoãn truy vấn dự phòng được cho là không chấp nhận được. Lựa chọn đầu là kết nối tới máy chủ chính và giữ một truy vấn hoạt động lâu tới mức cần thiết để chạy các truy vấn trong máy dự phòng. Điều này ngăn chặn VACUUM khỏi việc loại bỏ các hàng chết gần đây và vì thế làm sạch các xung đột không để xảy ra. Điều này có thể được thực hiện bằng việc sử dụng contrib/dblink và pg_sleep(), hoặc qua các cơ chế khác. Nếu bạn làm điều này, bạn nên lưu ý rằng điều này sẽ làm trễ sự làm sạch các hàng chết trong máy chủ chính, điều có thể gây ra sự sưng phồng bảng không mong đợi. Tuy nhiên, tình huống làm sạch sẽ không tệ hơn nếu các truy vấn dự phòng từng chạy trực tiếp trong máy chủ dự phòng, và bạn vẫn còn có lợi đối với việc bỏ tải thực thi trong máy chủ dự phòng. max_standby_archive_delay phải được giữ lớn hơn trong trường hợp này, vì các tệp WAL bị trễ có thể có rồi các khoản đầu vào mà xung đột với các truy vấn dự phòng được mong muốn.

Một lựa chọn khác là làm gia tăng vacuum_defer_cleanup_age trong máy chủ chính, sao cho các hàng chết sẽ không bị làm sạch nhanh như chúng thường xảy ra. Điều này sẽ cho phép có nhiều thời gian hơn cho các truy vấn để thực thi trước khi chúng bị hoãn trong máy dự phòng, không phải thiết lập một max_standby_streaming_delay cao. Tuy nhiên là khó để đảm bảo bất kỳ cửa sổ thời gian thực thi đặc biệt nào với tiếp cận này, vì vacuum_defer_cleanup_age được đo đếm trong các giao dịch được thực thi trong máy chủ chính.

25.5.3. Tổng quan về quản trị viên

Nếu hot_standby được bật (on) trong postgresql.conf và có một tệp recovery.conf tồn tại, thì máy chủ đó sẽ chạy trong chế độ Dự phòng Nóng (Hot Standby). Tuy nhiên, có thể mất chút thời gian cho các kết nối Dự phòng Nóng để được phép, vì máy chủ sẽ không chấp nhận các kết nối cho tới khi nó đã hoàn tất sự phục hồi đủ để cung cấp một tình trạng ổn định đối với các truy vấn nào có thể chạy được. Trong giai đoạn này, các máy trạm mà cố kết nối sẽ bị từ chối với một thông điệp lỗi. Để khẳng định máy chủ đã sẵn sàng, hoặc lặp lại cố gắng kết nối từ ứng dụng đó, hoặc tìm các thông điệp đó trong các lưu ký máy chủ:

```
LOG: entering standby mode
... then some time later ...
LOG: consistent recovery state reached
LOG: database system is ready to accept read only connections
```

Thông tin ổn định được ghi lại một lần cho từng điểm kiểm tra trong máy chủ chính. Không có khả năng để cho phép dự phòng nóng khi đang đọc WAL được viết trong một giai đoạn khi mà wal_level từng chưa được thiết lập về hot_standby trong máy chủ chính. Việc đạt được tình trạng ổn định cũng có thể bị trễ trong sự hiện diện của cả 2 điều kiện đó:

- Một giao dịch ghi có hơn 64 giao dịch con
- Các giao dịch ghi sống rất lâu

Nếu bạn đang chạy việc xuất lưu ký dựa vào tệp ("dự phòng ấm"), thì bạn có thể cần phải chờ cho tới khi tệp WAL tiếp sau tới, điều có thể dài như việc thiết lập archive_timeout trong máy chủ chính.

Thiết lập một vài tham số trong máy chủ dự phòng sẽ cần tái cấu hình nếu chúng đã bị thay đổi trong máy chủ chính. Đối với các tham số đó, giá trị trong máy chủ dự phòng phải bằng với hoặc lớn hơn giá trị trong máy chủ chính. Nếu các tham số đó không được thiết lập đủ cao thì sự dự phòng sẽ từ chối khởi động.

Các giá trị cao hơn có thể sau đó được cung cấp và máy chủ được khởi động lại để bắt đầu sự phục hồi một lần nữa. Các tham số đó là:

- max_connections
- max_prepared_transactions
- max_locks_per_transaction

Điều quan trọng là quản trị viên chọn các thiết lập đúng phù hợp cho max_standby_archive_delay và max_standby_streaming_delay. Những lựa chọn tốt nhất thay đổi và phụ thuộc vào các ưu tiên nghiệp vụ. Ví dụ nếu máy chủ đó trước hết có nhiệm vụ như là một máy chủ Hiệu năng Cao, thì bạn sẽ muốn các thiết lập trễ thấp, có lẽ thậm chí zero, dù điều đó là một thiết lập rất cực đoan. Nếu máy chủ dự phòng có nhiệm vụ như một máy chủ bổ sung thêm cho các truy vấn hỗ trợ quyết định thì có thể là chấp nhận được để thiết lập các giá trị trễ cực đại về nhiều giờ đồng hồ, hoặc thậm chí -1, điều có nghĩa là chờ vĩnh viễn các truy vấn để chúng hoàn tất.

Tình trạng giao dịch "gợi ý các bit" (hint bits) được ghi trong máy chủ chính sẽ không được ghi lưu ký WAL, nên dữ liệu trong máy chủ dự phòng có khả năng sẽ ghi lại các gợi lý đó một lần nữa trong

máy chủ dự phòng. Vì thế máy chủ sự phòng vẫn sẽ thực hiện ghi đĩa thậm chí dù tất cả những người sử dụng là chỉ đọc; không có thay đổi nào xảy ra đối với bản thân các giá trị dữ liệu đó. Những người sử dụng vẫn sẽ ghi các tệp tạm thời dạng lớn và tái sinh relcache trong các tệp thông tin, sao cho không phần nào của cơ sở dữ liệu thực sự là chỉ đọc trong chế độ dự phòng nóng. Cũng lưu ý là ghi vào các cơ sở dữ liệu ở xa bằng việc sử dụng module dblink, và các hoạt động khác bên ngoài cơ sở dữ liệu đó bằng việc sử dụng các hàm PL sẽ vẫn có khả năng, thậm chí dù giao dịch đó là chỉ đọc một cách cục bộ.

Các dạng lệnh quản trị sau đây không được chấp nhận trong chế độ phục hồi:

- Ngôn ngữ Định nghĩa Dữ liệu - DDL (Data Definition Language) - như CREATE INDEX
- Quyền ưu tiên và Quyền sở hữu - GRANT , REVOKE , REASSIGN
- Các lệnh duy trì - ANALYZE , VACUUM , CLUSTER , REINDEX

Một lần nữa, lưu ý rằng một vài trong số các lệnh đó thực sự được phép trong các giao dịch của chế độ “chỉ đọc” trong máy chủ chính.

Kết quả là, bạn không thể tạo các chỉ số bổ sung mà chỉ tồn tại trong máy chủ dự phòng, cũng không số liệu thống kê mà chỉ tồn tại trong máy chủ dự phòng. Nếu các lệnh quản trị đó là cần thiết, thì chúng sẽ được thực thi trong máy chủ chính, và rốt cuộc những thay đổi đó sẽ nhân giống tới máy chủ dự phòng.

pg_cancel_backend() sẽ làm việc ở các phần phụ trợ (backend) của người sử dụng, nhưng không trong tiến trình Khởi động (Startup), điều thực hiện sự phục hồi. pg_stat_activity không chỉ ra một đầu vào cho tiến trình Khởi động, cũng không thực hiện việc phục hồi các giao dịch chỉ ra như là đang chạy. Kết quả là, pg_prepared_xacts luôn là rỗng trong khi phục hồi. Nếu bạn muốn giải quyết các giao dịch được chuẩn bị còn bị nghi ngờ, hãy xem pg_prepared_xacts trong máy chủ chính và đưa ra các lệnh để giải quyết các giao dịch ở đó.

pg_locks sẽ chỉ ra các khóa được các phần phụ trợ (backend) nắm giữ, như bình thường. pg_locks cũng chỉ ra một giao dịch ảo được tiến trình Khởi động quản lý sở hữu tất cả AccessExclusiveLocks được nắm giữ bởi các giao dịch đang được sự phục hồi chơi lại. Lưu ý rằng tiến trình Khởi động không có các khóa để tiến hành các thay đổi của cơ sở dữ liệu, và vì thế các khóa khác với AccessExclusiveLocks không chỉ ra pg_locks trong tiến trình Khởi động; chúng chỉ được giả thiết có tồn tại.

Trình cài cắm check_pgsql của Nagios sẽ làm việc, vì thông tin đơn giản nó kiểm tra về sự tồn tại. Script giám sát check_postgres cũng sẽ làm việc, dù một vài giá trị được nêu có thể đưa ra các kết quả khác nhau hoặc gây bối rối. Ví dụ, thời điểm hút chân không mới nhất sẽ không được duy trì, vì không có sự hút chân không nào xảy ra trong máy dự phòng. Sự hút chân không chạy trong máy chính cũng vẫn gửi những thay đổi của chúng tới máy dự phòng.

Các lệnh kiểm soát tệp WAL sẽ không làm việc trong quá trình phục hồi, như pg_start_backup, pg_switch_xlog ...

Các module tải động được sẽ làm việc, bao gồm cả pg_stat_statements.

Các khóa cố vấn (advisory lock) làm việc bình thường trong sự phục hồi, bao gồm cả sự dò tìm ra khóa chết. Lưu ý là các khóa cố vấn không bao giờ được lưu ký WAL, nên không có khả năng đối với một khóa cố vấn hoặc trong máy chính hoặc trong máy dự phòng xung đột với sự chơi lại WAL. Cũng không có khả năng để có được một khóa cố vấn trong máy chủ chính và nhờ nó khởi tạo một khóa cố vấn tương tự trong máy dự phòng. Các khóa cố vấn chỉ có liên quan tới máy chủ trong đó chúng hiện diện.

Các hệ thống nhân bản dựa vào trigger như Slony, Londiste và Bucardo sẽ không chạy hoàn toàn trong máy dự phòng, dù chúng sẽ chạy tốt trong máy chủ chính miễn là những thay đổi sẽ không được gửi tới các máy chủ dự phòng sẽ được sử dụng. Sự chơi lại WAL không dựa vào trigger nên bạn không thể chơi lại từ máy dự phòng cho bất kỳ hệ thống nào mà đòi hỏi việc ghi cơ sở dữ liệu bổ sung hoặc dựa vào sử dụng các trigger.

Các OID mới không thể được chỉ định, dù một vài bộ sinh UUID có thể vẫn còn làm việc miễn là chúng không dựa vào việc ghi tình trạng mới tới cơ sở dữ liệu đó.

Hiện hành, sự tạo bảng tạm không được phép trong quá trình các giao dịch chỉ đọc, nên trong một số trường hợp các script đang tồn tại sẽ không chạy đúng. Hạn chế này có thể được nới lỏng trong một phiên bản sau này. Điều này vừa là một vấn đề tuân thủ Tiêu chuẩn SQL và vừa là một vấn đề kỹ thuật.

DROP TABLESPACE chỉ có thể thành công nếu không gian bảng là rỗng. Một vài người sử dụng dự phòng có thể đang tích cực sử dụng không gian bảng thông qua tham số temp_tablespaces của họ. Nếu có các tệp tạm thời trong không gian bảng đó, thì tất cả các truy vấn tích cực sẽ bị hoãn để đảm bảo rằng các tệp tạm thời sẽ bị loại bỏ, nên không gian bảng có thể bị loại bỏ và sự chơi lại WAL có thể tiếp tục.

Việc chạy DROP DATABASE hoặc ALTER DATABASE ... SET TABLESPACE trong máy chủ chính sẽ sinh ra một khoản đầu vào WAL mà sẽ làm cho tất cả những người sử dụng được kết nối tới cơ sở dữ liệu đó trong máy dự phòng sẽ bị ép mất kết nối. Hành động này xảy ra ngay lập tức, bất kể thiết lập max_standby_streaming_delay như thế nào. Lưu ý là ALTER DATABASE ... RENAME không bỏ kết nối những người sử dụng, điều trong hầu hết các trường hợp sẽ diễn ra không được để ý tới, dù có thể trong một số trường hợp gây ra một sự bối rối chương trình nếu nó phụ thuộc vào vài cách đối với tên cơ sở dữ liệu.

Ở chế độ bình thường (không phải phục hồi), nếu bạn đưa ra DROP USER hoặc DROP ROLE cho một vai trò với khả năng đăng nhập trong khi người sử dụng đó vẫn còn được kết nối thì không có gì xảy ra đối với người sử dụng được kết nối đó - họ vẫn giữ được kết nối. Tuy nhiên, người sử dụng đó không thể tái kết nối. Hành vi này cũng áp dụng trong sự phục hồi, nên một DROP USER trong máy chính không bỏ kết nối người sử dụng đó trong máy dự phòng.

Bộ thu thập số liệu thống kê đó hoạt động trong quá trình phục hồi. Tất cả các hành động quét, đọc, các khối, sử dụng chỉ số ... , sẽ được ghi lại bình thường trong máy dự phòng. Các hành động được chơi lại sẽ không đúp bản các hiệu ứng của chúng trong máy chính, nên việc chơi lại một sự chèn sẽ

không làm tăng thêm cột chèn (Insert) của pg_stat_user_tables. Tệp thống kê (stats) bị xóa khi khởi động phục hồi, nên stats từ máy chủ và máy dự phòng sẽ khác nhau; điều này được coi là một tính năng, không phải một lỗi.

Hút chân không tự động không hoạt động trong quá trình phục hồi. Nó sẽ khởi động bình thường ở cuối của sự phục hồi.

Trình viết (writer) phụ trợ hoạt động trong quá trình phục hồi và sẽ thực hiện các điểm khởi động lại (tương tự như các điểm kiểm tra trong máy chính) và bình thường khóa các hoạt động làm sạch. Điều này có thể bao gồm các bản cập nhật thông tin gợi ý bit được lưu trữ trong máy chủ dự phòng. Lệnh CHECKPOINT được chấp nhận trong quá trình phục hồi, dù nó thực hiện một điểm khởi động lại hơn là một điểm kiểm tra mới.

25.5.4. Tham chiếu tham số dự phòng nóng

Các tham số khác nhau từng được nhắc tới ở trên trong Phần 25.5.2 và Phần 25.5.3.

Trong máy chính, các tham số wal_level và vacuum_defer_cleanup_age có thể được sử dụng. max_standby_archive_delay và max_standby_streaming_delay không có hiệu ứng nếu được thiết lập trong máy chủ chính.

Trong máy chủ dự phòng, các tham số hot_standby, max_standby_archive_delay và max_standby_streaming_delay có thể được sử dụng. vacuum_defer_cleanup_age không có hiệu ứng miễn là máy chủ đó vẫn ở chế độ dự phòng, dù nó sẽ trở nên phù hợp nếu máy dự phòng sẽ trở thành máy chính.

25.5.5. Khiếm khuyết

Có vài hạn chế của Dự phòng Nóng. Chúng có thể và có lẽ sẽ được sửa trong các phiên bản trong tương lai:

- Các hoạt động trong các chỉ số băm hiện không được ghi lưu ký WAL, nên sự chơi lại sẽ không cập nhật các chỉ số đó.
- Tri thức đầy đủ về việc chạy các giao dịch được yêu cầu trước khi các hình chụp có thể được thực hiện. Các giao dịch sử dụng số lượng lớn các giao dịch con (hiện lớn hơn 64) sẽ trễ sự khởi động các kết nối chỉ đọc cho tới khi hoàn thành giao dịch ghi chạy lâu nhất. Nếu tình huống này xảy ra, các thông điệp giải thích sẽ được gửi tới lưu ký máy chủ đó.
- Các điểm khởi động hợp lệ cho các truy vấn dự phòng được sinh ra ở từng điểm kiểm tra trong máy chủ chính. Nếu máy dự phòng bị tắt trong khi máy chính ở một chế độ tắt, thì có thể không có khả năng vào lại Dự phòng Nóng cho tới khi máy chủ chính được khởi động lại, vì thế nó sinh ra các điểm khởi động tiếp theo trong các lưu ký WAL. Tình huống này không là vấn đề trong các tình huống phổ biến nhất, nơi mà nó có thể xảy ra. Thường thì, nếu máy chính bị tắt và không sẵn sàng hơn được nữa, thì có khả năng điều đó là vì một sự hỏng nghiêm trọng và đòi hỏi máy dự phòng đang được chuyển đổi phải hoạt động như máy chủ chính mới bằng mọi cách. Và trong các tình huống nơi mà máy chủ chính đang cố ý được tắt, việc phối hợp để chắc chắn máy dự phòng trở thành máy chính mới một cách trơn

tru cũng là một thủ tục tiêu chuẩn.

- Ở cuối của sự phục hồi, AccessExclusiveLocks được các giao dịch được chuẩn bị nắm giữ sẽ đòi hỏi 2 lần số thông thường các đầu vào bảng khóa. Nếu bạn có kế hoạch trong việc chạy hoặc một số lượng lớn các giao dịch được chuẩn bị đồng thời mà thường lấy AccessExclusiveLocks, hoặc bạn có kế hoạch trong việc có một giao dịch lớn mà lấy nhiều AccessExclusiveLocks, thì bạn được khuyến cáo chọn một giá trị lớn hơn max_locks_per_transaction, có lẽ nhiều gấp 2 lần giá trị của tham số trong máy chủ chính. Bạn không cần xem điều này như là tất cả nếu thiết lập max_prepared_transactions của bạn bằng 0.

Chương 26. Cấu hình sự phục hồi

Chương này mô tả các thiết lập sẵn sàng trong tệp recovery.conf. Chúng áp dụng chỉ trong khoảng thời gian phục hồi. Chúng phải được thiết lập lại cho bất kỳ sự phục hồi tiếp sau nào mà bạn muốn thực hiện. Chúng không thể bị thay đổi một khi sự phục hồi đã bắt đầu.

Các thiết lập trong recovery.conf được chỉ định ở định dạng name = 'value'. Một tham số được chỉ định trên một dòng. Các dấu thăng (#) chỉ định phần còn lại của dòng như một chú giải. Để nhúng một dấu nháy đơn vào một giá trị tham số, hãy viết 2 dấu nháy đơn ('').

Một tệp ví dụ, share/recovery.conf.sample, được đưa ra trong thư mục share/ của cài đặt.

26.1. Thiết lập phục hồi lưu trữ

restore_command (string)

Lệnh shell này thực thi để truy xuất một phân đoạn được lưu trữ của một loạt tệp WAL. Tham số này được yêu cầu cho sự phục hồi lưu trữ, nhưng là tùy ý cho nhân bản dòng. Bất kỳ %f nào trong chuỗi đó cũng được thay thế bằng tên tệp để truy xuất từ lưu trữ đó, và bất kỳ %p nào cũng được thay thế bằng tên đường dẫn đích sao chép trên máy chủ. (Tên đường dẫn đó là tương đối cho thư mục làm việc hiện hành, như, thư mục dữ liệu bó). Bất kỳ %r nào cũng được thay thế bằng tên tệp có chứa điểm khởi động lại hợp lệ mới nhất. Đó là tệp sớm nhất mà phải được giữ để cho phép một sự phục hồi sẽ được khởi động lại, nên thông tin này có thể được sử dụng để cắt ngắn bớt lưu trữ tới mức tối thiểu được yêu cầu để hỗ trợ cho việc khởi động lại từ sự phục hồi hiện hành. %r thường chỉ được các cấu hình dự phòng ấm sử dụng (xem Phần 25.2). Hãy viết %% để nhúng một ký tự %.

Là quan trọng để lệnh đó trả về một tình trạng thoát ra zero chỉ nếu nó thành công. Lệnh đó sẽ được yêu cầu cho các tên tệp mà không hiện diện trong lưu trữ; nó phải trả về khác zero khi được yêu cầu như vậy. Các ví dụ:

```
restore_command = 'cp /mnt/server/archivedir/%f "%p"'
restore_command = 'copy "C:\\server\\archivedir\\%f" "%p"' # Windows
```

archive_cleanup_command (string)

Tham số tùy ý này chỉ định một lệnh shell mà sẽ được thực thi ở từng điểm khởi động lại. Mục đích của archive_cleanup_command là để cung cấp một cơ chế cho việc làm sạch các tệp WAL cũ được lưu trữ mà không còn cần thiết nữa đối với máy chủ dự phòng. Bất kỳ %r cũng được thay thế bằng tên của tệp có chứa điểm khởi động lại hợp lệ mới nhất. Đó là tệp sớm nhất mà phải được giữ để cho phép một sự phục hồi sẽ khởi động lại được, và vì thế tất cả các tệp sớm hơn so với %r có thể sẽ được loại bỏ một cách an toàn. Thông tin này có thể được sử dụng để cắt ngắn bớt lưu trữ tới mức tối thiểu được yêu cầu để hỗ trợ cho sự khởi động lại từ sự phục hồi hiện hành. Tiện ích pg_archivecleanup được cung cấp trong contrib (xem Phần F.22) phục vụ như một đích thuận lợi cho archive_cleanup_command trong các cấu hình dự phòng đơn, ví dụ:

```
archive_cleanup_command = 'pg_archivecleanup /mnt/server/archivedir %r'
```

Tuy nhiên hãy lưu ý rằng nếu nhiều máy chủ dự phòng đang phục hồi từ cùng y hệt thư mục

lưu trữ, thì bạn sẽ cần phải đảm bảo rằng bạn không xóa các tệp WAL cho tới khi không còn cần thiết nữa đối với bất kỳ máy chủ nào. Có lẽ archive_cleanup_command thường được sử dụng trong một cấu hình dự phòng ấm (xem Phần 15.2). Hãy viết %% để nhúng một ký tự % vào lệnh đó. Nếu lệnh đó trả về một tình trạng thoát ra khác zero thì một thông điệp lưu ký WARNING sẽ được viết.

recovery_end_command (string)

Tham số này chỉ định một lệnh shell mà sẽ được thực thi chỉ một lần ở cuối của sự phục hồi. Tham số này là tùy chọn. Mục đích của recovery_end_command là để cung cấp một cơ chế cho việc làm sạch sau nhân bản hoặc phục hồi. Bất kỳ %r nào cũng được thay thế bằng tên của tệp có chứa điểm khởi động lại hợp lệ mới nhất, như trong archive_cleanup_command.

Nếu lệnh đó trả về một tình trạng thoát ra khác zero thì một thông điệp lưu ký WARNING sẽ được viết và cơ sở dữ liệu sẽ xử lý để khởi động lại. Một ngoại lệ là nếu lệnh đó từng bị một tín hiệu kết thúc, thì cơ sở dữ liệu đó sẽ không xử lý với sự khởi động lại.

26.2. Các thiết lập đích phục hồi

recovery_target_time (timestamp)

Tham số này chỉ định dấu thời gian cho sự phục hồi nào sẽ tiến hành. Hầu như một trong recovery_target_time và recovery_target_xid có thể được chỉ định. Mặc định là để phục hồi tới cuối của lưu ký WAL. Điểm dừng chính xác cũng bị ảnh hưởng vì recovery_target_inclusive.

recovery_target_xid (string)

Tham số này chỉ định ID giao dịch cho sự phục hồi nào sẽ tiến hành. Hãy nhớ là trong khi các ID giao dịch được chỉ định tuần tự ở đầu giao dịch, thì các giao dịch có thể hoàn tất theo một trật tự số khác nhau. Các giao dịch mà sẽ được phục hồi sẽ là các giao dịch được thực hiện trước (và thường bao gồm) là một giao dịch được chỉ định. Hầu như một trong recovery_target_xid và recovery_target_time có thể được chỉ định. Mặc định là để phục hồi tới cuối của lưu ký WAL. Điểm dừng chính xác cũng bị ảnh hưởng vì recovery_target_inclusive.

recovery_target_inclusive (boolean)

Chỉ định liệu chúng ta có dừng ngay sau đích phục hồi được chỉ định (true) hay không, hay ngay trước đích phục hồi (false). Áp dụng cho cả recovery_target_time và recovery_target_xid, bất kể cái nào được chỉ định cho sự phục hồi này. Điều này chỉ ra liệu các giao dịch có thời gian hoặc ID được thực hiện cho đích có chính xác hay không, một cách tương ứng, sẽ được đưa vào trong sự phục hồi đó. Mặc định là đúng - true.

recovery_target_timeline (string)

Chỉ định việc phục hồi trong một dòng thời gian cụ thể. Mặc định là để phục hồi theo cùng y hệt dòng thời gian từng là hiện hành khi sao lưu cơ bản đã được thực hiện. Bạn chỉ cần thiết lập tham số này trong các tình huống phục hồi lại phức tạp, nơi mà bạn cần trả về một tình trạng mà bản thân nó từng đạt được sau một sự phục hồi đúng thời điểm. Xem Phần 24.3.4

để thảo luận.

26.3. Thiết lập máy chủ dự phòng

standby_mode (boolean)

Chỉ định liệu có khởi động máy chủ PostgreSQL như một máy chủ dự phòng hay không. Nếu tham số này được bật (on), thì máy chủ sẽ không dừng phục hồi khi kết thúc WAL được lưu trữ đạt tới, nhưng sẽ cố gắng tiếp tục phục hồi bằng việc lấy các phân đoạn mới của WAL bằng việc sử dụng restore_command và/hoặc kết nối tới máy chủ chính như được thiết lập primary_conninfo chỉ định.

primary_conninfo (string)

Chỉ định một chuỗi kết nối sẽ được sử dụng cho máy chủ dự phòng để kết nối với máy chủ chính. Chuỗi này là ở định dạng được chấp nhận bởi hàm libpq PQconnectdb, được mô tả trong Phần 31.1. Nếu bất kỳ lựa chọn nào không được chỉ định trong chuỗi này, thì biến môi trường tương ứng (xem Phần 31.13) sẽ được kiểm tra. Nếu biến môi trường cũng không được thiết lập, thì các mặc định sẽ được sử dụng.

Chuỗi kết nối sẽ chỉ định tên máy chủ host (hoặc địa chỉ) của máy chủ chính, cũng như số cổng nếu nó không là y hệt như mặc định của máy chủ dự phòng. Cũng chỉ định một tên người sử dụng tương ứng với một vai trò mà có các quyền ưu tiên SUPERUSER và LOGIN trong máy chủ chính (xem Phần 25.2.5.1). Một mật khẩu cũng cần phải được cung cấp, nếu máy chủ chính đòi hỏi xác thực mật khẩu. Nó có thể được cung cấp trong chuỗi primary_conninfo, hoặc trong một tệp riêng rẽ ~/.pgpass trong máy chủ dự phòng (sử dụng replication như tên của cơ sở dữ liệu). Không chỉ định tên một cơ sở dữ liệu trong chuỗi primary_conninfo.

Thiết lập này không có tác dụng nếu standby_mode là tắt (off).

trigger_file (string)

Chỉ định một tệp trigger mà sự hiện diện của nó kết thúc sự phục hồi trong máy chủ dự phòng. Nếu không có tệp trigger nào được chỉ định, thì sự dự phòng không bao giờ thoát ra khỏi sự phục hồi. Thiết lập này không có tác dụng nếu standby_mode là tắt (off).

Chương 27. Giám sát hoạt động của cơ sở dữ liệu

Một người quản trị cơ sở dữ liệu thường tự hỏi, "Hệ thống có đang làm việc đúng hay không nhỉ?". Chương này thảo luận cách để tìm ra điều đó.

Vài công cụ là sẵn sàng cho việc giám sát hoạt động của cơ sở dữ liệu và phân tích hiệu năng. Hầu hết chương này dành cho việc mô tả bộ sưu tập số liệu thống kê của PostgreSQL, nhưng người ta sẽ không bỏ qua các chương trình giám sát thường xuyên của Unix như ps, top, iostat, và vmstat. Hơn nữa, một khi người ta đã xác định một truy vấn thực hiện không tốt, thì sự điều tra tiếp tục có thể cần thiết bằng việc sử dụng lệnh EXPLAIN của PostgreSQL. Phần 14.1 thảo luận về EXPLAIN và các phương pháp khác để hiểu được hành vi của một truy vấn đơn lẻ.

27.1. Công cụ Unix tiêu chuẩn

Trong hầu hết các nền tảng Unix, PostgreSQL sửa đổi đầu đề lệnh của nó như được ps nêu, sao cho các tiến trình của máy chủ riêng lẻ có thể được nhận diện rồi. Một hiển thị mẫu là

```
$ ps auxww | grep ^postgres
postgres         960  0.0  1.1  6104  1480  pts/1  SN  13:17  0:00  postgres -i
postgres         963  0.0  1.1  7084  1472  pts/1  SN  13:17  0:00  postgres: writer process
postgres         965  0.0  1.1  6152  1512  pts/1  SN  13:17  0:00  postgres: stats collector
postgres         998  0.0  2.3  6532  2992  pts/1  SN  13:18  0:00  postgres: tgl runbug 127.0.0.1
postgres        1003  0.0  2.4  6532  3128  pts/1  SN  13:19  0:00  postgres: tgl regression [
postgres        1016  0.1  2.4  6532  3080  pts/1  SN  13:19  0:00  postgres: tgl regression [
```

(Lời gọi đúng phù hợp ps biến đổi xuyên khắp các nền tảng khác nhau, như các chi tiết của những gì được đưa ra. Ví dụ này là từ một hệ thống Linux gần đây). Tiến trình đầu được liệt kê ở đây là tiến trình của máy chủ chính. Các đối số lệnh đó là y hệt như các đối số được sử dụng khi nó từng được khởi động. 2 tiến trình tiếp sau là các tiến trình của trình làm việc phần phụ trợ (backend) tự động được tiến trình chính khởi động. (Tiến trình "bộ thu thập số liệu thống kê" sẽ không hiện diện nếu bạn đã thiết lập hệ thống không khởi động bộ thu thập số liệu thống kê đó). Mỗi trong số các tiến trình còn lại là một tiến trình máy chủ điều khiển một kết nối máy trạm. Mỗi tiến trình như vậy thiết lập hiển thị dòng lệnh của nó ở dạng

```
postgres: user database host activity.
```

Người sử dụng, cơ sở dữ liệu, và các khoản của host (máy trạm) vẫn là y hệt cho sự tồn tại của kết nối máy trạm, nhưng chỉ số hoạt động thay đổi. Hoạt động đó có thể là nhàn rỗi idle (như, chờ một lệnh máy trạm), nhàn rỗi trong giao dịch - idle in transaction (chờ máy trạm trong một khối BEGIN), hoặc một tên dạng lệnh như SELECT. Hơn nữa, việc chờ đợi được nối thêm vào nếu tiến trình máy chủ hiện đang chờ trong một khóa mà một phiên làm việc khác nắm giữ. Trong ví dụ ở trên chúng ta có thể phỏng đoán rằng tiến trình 1003 đang chờ tiến trình 1016 hoàn tất giao dịch của nó và vì thế đưa ra vài khóa.

Nếu bạn đã tắt update_process_title thì chỉ số hoạt động sẽ không được cập nhật; đầu đề tiến trình chỉ được thiết lập khi một tiến trình được khởi động. Trong một số nền tảng điều này tiết kiệm được số lượng lớn tổng chi phí theo từng lệnh; trong các hệ thống khác thì điều này là không đáng kể.

Mẹo: Solaris đòi hỏi việc điều khiển đặc biệt. Bạn phải sử dụng /usr/ucb/ps, thay vì /bin/ps. Bạn cũng phải sử dụng 2 cờ w, chứ không phải 1. Hơn nữa, sự triệu gọi ban đầu của bạn lệnh postgres phải có một hiển thị tình trạng ps ngắn hơn so với hiển thị được từng tiến trình máy chủ cung cấp. Nếu bạn không làm tất cả 3 điều trên, thì đầu ra của ps đối với từng tiến trình máy chủ sẽ là dòng lệnh postgres gốc ban đầu.

27.2. Bộ thu thập số liệu thống kê

Bộ sưu tập số liệu thống kê của PostgreSQL là một hệ thống con hỗ trợ thu thập và báo cáo thông tin về hoạt động của máy chủ. Hiện hành, bộ sưu tập đó có thể tính các truy cập tới các bảng và chỉ số, theo cả các hàng riêng rẽ và khối đĩa. Nó cũng theo dõi tổng số các hàng trong từng bảng, và lần hút chân không mới nhất và phân tích thời gian cho từng bảng. Nó cũng có thể tính các cuộc gọi tới các hàm do người sử dụng định nghĩa và tổng thời gian bỏ ra trong từng cuộc gọi đó.

PostgreSQL cũng hỗ trợ báo cáo lệnh chính xác hiện đang được các tiến trình khác thực thi. Đây là một tiện ích độc lập với tiến trình của bộ sưu tập.

27.2.1. Cấu hình sưu tập số liệu thống kê

Vì sự sưu tập các số liệu thống kê bổ sung thêm tổng chi phí đối với sự thực thi truy vấn, nên hệ thống có thể được thiết lập cấu hình để thu thập hoặc không thu thập thông tin. Điều này được các tham số cấu hình kiểm soát, chúng thường được thiết lập trong postgresql.conf. (Xem Chương 18 để có các chi tiết về việc thiết lập các tham số cấu hình).

Tham số track_counts kiểm soát liệu các số liệu thống kê có được thu thập về các truy cập bảng và chỉ số hay không.

Tham số track_functions cho phép theo dõi sử dụng các hàm mà người sử dụng tự định nghĩa.

Tham số track_activities cho phép giám sát lệnh hiện hành đang được bất kỳ tiến trình nào của máy chủ thực thi.

Thường thì các tham số đó được thiết lập trong postgresql.conf sao cho chúng áp dụng cho tất cả các tiến trình máy chủ, nhưng có khả năng bật hoặc tắt chúng trong các phiên làm việc riêng lẻ bằng việc sử dụng lệnh SET. (Để ngăn chặn những người sử dụng thông thường khỏi việc dấu hoạt động của họ đối với quản trị viên, chỉ những siêu người sử dụng sẽ được phép thay đổi các tham số đó với lệnh SET).

Bộ sưu tập số liệu thống kê giao tiếp với các phần phụ trợ (backend) cần thông tin (bao gồm cả tự động hút chân không) thông qua các tệp tạm thời. Các tệp đó được lưu trữ trong thư mục con pg_stat_tmp. Khi postmaster tắt, một bản sao vĩnh viễn các dữ liệu thống kê được lưu trữ trong thư mục con tổng thể global. Để hiệu năng được gia tăng, tham số stats_temp_directory có thể được trỏ trong một hệ thống tệp dựa vào RAM, làm giảm đi các yêu cầu I/O vật lý.

27.2.2. Kiểu nhìn số liệu thống kê được thu thập

Vài kiểu nhìn được xác định trước, được liệt kê trong Bảng 27-1, là sẵn sàng để chỉ ra các kết quả thu thập số liệu thống kê.

Như một sự lựa chọn, người ta có thể xây dựng các kiểu nhìn tùy biến bằng việc sử dụng các hàm

thống kê nằm bên dưới. Khi sử dụng số liệu thống kê để giám sát hoạt động hiện hành, điều quan trọng phải nhận thức được rằng thông tin không cập nhật cùng đồng thời. Từng tiến trình máy chủ riêng lẻ truyền các đếm số liệu thống kê mới tới bộ sưu tập ngay trước khi trở thành nhàn rỗi; vì thế một truy vấn hoặc giao dịch vẫn còn trong tiến trình không ảnh hưởng tới các tổng số được hiện thị. Hơn nữa, bản thân bộ sưu tập phát ra một báo cáo mới hầu như một lần cho từng PGSTAT_STAT_INTERVAL mili giây (500 trừ phi được chỉnh sửa khi xây dựng máy chủ đó). Vì thế thông tin được hiển thị lạc hậu sau hoạt động thực tế. Tuy nhiên, thông tin truy vấn hiện thời được track_activities thu thập luôn là cập nhật.

Điểm quan trọng khác là khi một tiến trình máy chủ được yêu cầu hiển thị bất kỳ số liệu thống kê nào, thì nó trước hết lấy báo cáo gần nhất được tiến trình của bộ sưu tập phát đi và sau đó tiếp tục sử dụng hình chụp này cho tất cả các kiểu nhìn và các hàm số liệu thống kê cho tới kết thúc giao dịch hiện hành của nó. Tương tự, thông tin về các truy vấn hiện hành của tất cả các phiên làm việc được thu thập khi bất kỳ thông tin nào như vậy lần đầu được yêu cầu bên trong một giao dịch, và thông tin y hệt sẽ được hiển thị thông qua giao dịch đó. Đây là một tính năng, không phải là một lỗi, vì nó cho phép bạn thực hiện vài truy vấn về số liệu thống kê và đối sánh các kết quả mà không lo lắng về các số đang thay đổi bên dưới bạn. Nhưng nếu bạn muốn thấy các kết quả mới với từng truy vấn, hãy chắc chắn thực hiện các truy vấn đó bên ngoài bất kỳ khối giao dịch nào. Như một sự lựa chọn, bạn có thể gọi tới pg_stat_clear_snapshot (), nó sẽ bỏ qua hình chụp số liệu thống kê của giao dịch hiện hành (nếu có). Sử dụng tiếp theo thông tin số liệu thống kê sẽ làm cho một hình chụp mới sẽ được lấy về.

Bảng 27-1. Các kiểu nhìn số liệu thống kê tiêu chuẩn

Tên kiểu nhìn	Mô tả
pg_stat_activity	Một hàng cho từng tiến trình máy chủ, chỉ ra OID cơ sở dữ liệu, tên cơ sở dữ liệu, ID tiến trình, OID người sử dụng, tên người sử dụng, tên ứng dụng, địa chỉ và số cổng máy trạm, các thời điểm ở đó tiến trình máy chủ, giao dịch hiện hành, và truy vấn hiện hành đã bắt đầu thực thi, tình trạng chờ của tiến trình, và văn bản truy vấn hiện hành. Các cột mà báo cáo dữ liệu về truy vấn hiện hành sẽ là sẵn sàng trừ phi tham số track_activities đã bị tắt. Hơn nữa, các cột đó chỉ nhìn thấy nếu người sử dụng xem xét kiểu nhìn là một siêu người sử dụng hoặc y hệt như người sử dụng sở hữu tiến trình đang được báo cáo.
pg_stat_bgwriter	Chỉ một hàng, chỉ ra các số liệu thống kê rộng cả bó từ trình ghi nền: số các điểm kiểm tra được lên lịch, các điểm kiểm tra được yêu cầu, các bộ nhớ đệm được các điểm kiểm tra và các vụ quét ghi, và số lần trình ghi nền đã dừng một vụ quét làm sạch vì nó đã ghi quá nhiều bộ nhớ đệm. Còn bao gồm cả các số liệu thống kê về kho bộ nhớ đệm được chia sẻ, bao gồm các bộ nhớ đệm được các phần phụ trợ (backend) ghi (đó là, không phải do trình ghi nền) và tổng số các bộ nhớ đệm được phân bổ.
pg_stat_database	Một hàng cho từng cơ sở dữ liệu, chỉ ra OID cơ sở dữ liệu, tên cơ sở dữ liệu, số tiến trình máy chủ đang hoạt động được kết nối tới cơ sở dữ liệu đó, số giao dịch được thực hiện và được quay ngược lại trong cơ sở dữ liệu đó, tổng số các khối đĩa được đọc, tổng số lần làm việc với bộ nhớ đệm (như, các yêu cầu đọc khối tránh được bằng việc tìm khối đó sẵn rồi trong bộ nhớ tạm của bộ nhớ đệm), số các hàng được trả về, được lấy về, được chèn vào, được cập nhật và bị xóa.
pg_stat_all_tables	Đối với từng bảng trong cơ sở dữ liệu hiện hành (bao gồm các bảng TOAST), OID bảng, sơ đồ và tên bảng, số các vụ quét tuần tự được khởi tạo, số các hàng hoạt động được lấy về bằng các vụ quét tuần tự, số các vụ quét chỉ số được khởi tạo (qua tất cả các chỉ số thuộc về bảng đó), số các hàng hoạt động được lấy về bằng các vụ quét chỉ số, số các vụ chèn

Tên kiểu nhìn	Mô tả
	hàng, các cập nhật, và các vụ xóa, số các bản cập nhật hàng từng là HOT (như, không bản cập nhật chỉ số riêng rẽ nào), số các hàng hoạt động và chết, lần cuối bảng đó từng được hút chân không bằng tay, lần cuối nó từng được hút chân không bằng daemon tự động hút chân không, lần cuối nó từng được phân tích bằng tay, và lần cuối nó từng được phân tích bằng daemon tự động hút chân không.
pg_stat_sys_tables	Hệt như pg_stat_all_tables, ngoại trừ là chỉ các bảng hệ thống được chỉ ra.
pg_stat_user_tables	Hệt như pg_stat_all_tables, ngoại trừ là chỉ các bảng hệ thống được chỉ ra.
pg_stat_all_indexes	Đối với từng chỉ số trong cơ sở dữ liệu hiện hành, OID bảng và chỉ số, sơ đồ, tên bảng và chỉ số, số các vụ quét chỉ số được khởi tạo trong chỉ số đó, số các khoản đầu vào của chỉ số được trả về bằng các vụ quét chỉ số đó, và số các hàng hoạt động của bảng được lấy về bằng các vụ quét chỉ số đó thông qua việc sử dụng chỉ số đó.
pg_stat_sys_indexes	Hệt như pg_stat_all_indexes, ngoại trừ chỉ các chỉ số trong các bảng hệ thống được chỉ ra.
pg_stat_user_indexes	Hệt như pg_stat_all_indexes, ngoại trừ chỉ các chỉ số trong các bảng người sử dụng được chỉ ra.
pg_statio_all_tables	Đối với từng bảng trong cơ sở dữ liệu hiện hành (bao gồm các bảng TOAST), OID bảng, tên sơ đồ và bảng, số các khối đĩa được đọc từ bảng đó, số các truy cập bộ nhớ đệm, số các khối đĩa được đọc và truy cập bộ nhớ đệm trong tất cả các chỉ số của bảng đó, số các khối đĩa được đọc và các truy cập bộ nhớ đệm từ bảng TOAST bổ sung của bảng đó (nếu có), và số các khối đĩa được đọc và truy cập bộ nhớ đệm cho chỉ số bảng TOAST đó.
pg_statio_sys_tables	Hệt như pg_statio_all_tables, ngoại trừ chỉ các bảng hệ thống được chỉ ra.
pg_statio_user_tables	Hệt như pg_statio_all_tables, ngoại trừ chỉ các bảng người sử dụng được chỉ ra.
pg_statio_all_indexes	Đối với từng chỉ số trong cơ sở dữ liệu hiện hành, OID bảng và chỉ số, sơ đồ, tên bảng và chỉ số, số các khối đĩa được đọc và truy cập bộ nhớ đệm trong chỉ số đó.
pg_statio_sys_indexes	Hệt như pg_statio_all_indexes, trừ chỉ các chỉ số trong các bảng hệ thống được chỉ ra.
pg_statio_user_indexes	Hệt như pg_statio_all_indexes, trừ việc chỉ các chỉ số trong các bảng người sử dụng được chỉ ra.
pg_statio_all_sequences	Đối với từng đối tượng tuần tự trong cơ sở dữ liệu hiện hành, OID tuần tự, tên sơ đồ và sự tuần tự, số các khối đĩa được đọc và truy cập bộ nhớ đệm trong sự tuần tự đó.
pg_statio_sys_sequences	Hệt như pg_statio_all_sequences, ngoại trừ rằng chỉ các tuần tự hệ thống được chỉ ra. (Hiện hành, không sự tuần tự hệ thống nào được xác định, nên kiểu nhìn này luôn rỗng).
pg_statio_user_sequences	Hệt như pg_statio_all_sequences, ngoại trừ rằng chỉ các tuần tự người sử dụng được chỉ ra.
pg_stat_user_functions	Đối với tất cả các hàm được theo dõi, OID hàm, sơ đồ, tên, số các lời gọi, tổng số thời gian, và bản thân thời gian đó. Bản thân thời gian là lượng thời gian bỏ ra trong bản thân hàm đó, tổng thời gian bao gồm thời gian bỏ ra trong các hàm mà nó đã gọi. Các giá trị thời gian được tính bằng mili giây.

Các số liệu thống kê cho từng chỉ số đặc biệt là hữu dụng để xác định các chỉ số nào đang được sử dụng và chúng hiệu quả như thế nào.

Ban đầu trong PostgreSQL 8.1, các chỉ số có thể được sử dụng hoặc trực tiếp hoặc thông qua “các quét bitmap”. Trong một quét bitmap thì đầu ra của vài chỉ số có thể được kết hợp thông qua các qui tắc AND hoặc OR; vì thế là khó khăn để liên kết với những thứ lấy về từ các hàng theo đống riêng lẻ với các chỉ số đặc biệt khi một quét bitmap được sử dụng. Vì thế, một quét bitmap tăng dần dần (các) tính đếm pg_stat_all_indexes.idx_tup_read đối với (các) chỉ số mà nó sử dụng, và nó tăng dần

tính đếm pg_stat_all_tables.idx_tup_fetch đối với bảng đó, nhưng nó không ảnh hưởng tới pg_stat_all_indexes.idx_tup_fetch.

> **Lưu ý**: Trước phiên bản PostgreSQL 8.1, các tính đếm idx_tup_read và idx_tup_fetch về cơ bản từng luôn bằng nhau. Bây giờ chúng có thể khác nhau thậm chí không có việc xem xét các quét bitmap, vì idx_tup_read tính các khoản đầu vào chỉ số được truy xuất từ chỉ số đó trong khi idx_tup_fetch tính các hàng hoạt động được lấy về từ bảng đó; cái sau sẽ là ít hơn nếu bất kỳ hàng chết hoặc còn chưa được thực hiện nào được lấy về bằng việc sử dụng chỉ số đó.

Các kiểu nhìn pg_statio_ views trước hết là hữu dụng để xác định tính hiệu quả của bộ nhớ tạm của bộ nhớ đệm. Khi số lượng đọc đĩa nhỏ hơn nhiều so với số lượng truy cập bộ nhớ đệm, thì bộ nhớ tạm làm thỏa mãn được hầu hết các yêu cầu đọc mà không có việc triệu gọi tới một nhân. Tuy nhiên, các số liệu thống kê đó không đưa ra toàn bộ câu chuyện: vì cách thức theo đó PostgreSQL điều khiển I/O của đĩa, các dữ liệu mà không ở trong bộ nhớ tạm của bộ nhớ đệm của PostgreSQL có thể vẫn nằm trong bộ nhớ tạm I/O của nhân, và có thể vì thế vẫn được lấy về mà không có yêu cầu một sự đọc vật lý. Những người sử dụng có quan tâm trong việc có được nhiều thông tin chi tiết hơn về hành vi I/O của PostgreSQL được khuyến cáo sử dụng bộ thu thập số liệu thống kê của PostgreSQL kết hợp với các tiện ích của hệ điều hành mà cho phép hiểu sâu sắc trong việc điều khiển I/O của nhân.

Các cách thức khác nhìn vào số liệu thống kê có thể được thiết lập bằng việc viết các truy vấn mà sử dụng cùng các hàm truy cập số liệu thống kê nằm bên dưới như các kiểu nhìn tiêu chuẩn đó làm. Các hàm đó được liệt kê trong Bảng 27-2. Các hàm truy cập theo từng cơ sở dữ liệu lấy OID một cơ sở dữ liệu như là đối số để nhận diện cơ sở dữ liệu nào để báo cáo. Các hàm theo từng bảng và theo từng chỉ số lấy OID của một bảng hoặc một chỉ số. Các hàm cho các số liệu thống kê lời gọi hàm lấy một OID hàm. (Lưu ý rằng chỉ các bảng, các chỉ số, và các hàm trong cơ sở dữ liệu hiện hành có thể được xem với các hàm đó). Các hàm truy cập theo từng máy chủ lấy số tiến trình một máy chủ, trải từ 1 tới số các tiến trình máy chủ hoạt động hiện hành.

Bảng 27-2. Các hàm truy cập số liệu thống kê

Hàm	Dạng trả về	Mô tả
pg_stat_get_db_numbackends(oid)	integer	Số tiến trình máy chủ hoạt động cho cơ sở dữ liệu
pg_stat_get_db_xact_commit(oid)	bigint	Số giao dịch được thực hiện trong cơ sở dữ liệu
pg_stat_get_db_xact_rollback(oid)	bigint	Số giao dịch đã quay lại trong cơ sở dữ liệu
pg_stat_get_db_blocks_fetched(oid)	bigint	Số yêu cầu lấy về khối đĩa cho cơ sở dữ liệu
pg_stat_get_db_blocks_hit(oid)	bigint	Số yêu cầu lấy về khối đĩa thấy trong bộ nhớ tạm cho cơ sở dữ liệu
pg_stat_get_db_tuples_returned(iod)	bigint	Số bộ dữ liệu được trả về cho cơ sở dữ liệu
pg_stat_get_db_tuples_fetched(oid)	bigint	Số bộ dữ liệu lấy được về cho cơ sở dữ liệu
pg_stat_get_db_tuples_inserted(oid)	bigint	Số bộ dữ liệu được chèn vào cơ sở dữ liệu
pg_stat_get_db_tuples_updated(oid)	bigint	Số bộ dữ liệu được cập nhật trong cơ sở dữ liệu
pg_stat_get_db_tuples_deleted(oid)	bigint	Số bộ dữ liệu bị xóa trong cơ sở dữ liệu
pg_stat_get_numscans(oid)	bigint	Số quét tuần tự được thực hiện khi đối số là một

Hàm	Dạng trả về	Mô tả
		bảng, hoặc số các quét chỉ số được thực hiện khi đối số là một chỉ số
pg_stat_get_tuples_returned(oid)	bigint	Số hàng được các quét tuần tự đọc khi đối số là một bảng, hoặc số các khoản đầu vào chỉ số được trả về khi đối số là một chỉ số
pg_stat_get_tuples_fetched(oid)	bigint	Số hàng của bảng được các quét bitmap lấy về khi đối số là một bảng, hoặc các hàng của bảng được các quét chỉ số đơn giản lấy về bằng việc sử dụng chỉ số đó khi đối số là một chỉ số
pg_stat_get_tuples_inserted(oid)	bigint	Số hàng được chèn vào trong bảng
pg_stat_get_tuples_updated(oid)	bigint	Số hàng được cập nhật trong bảng (bao gồm cả các bản cập nhật nóng - HOT)
pg_stat_get_tuples_deleted(oid)	bigint	Số hàng bị xóa khỏi bảng
pg_stat_get_tuples_hot_updated(oid)	bigint	Số hàng được cập nhật nóng - HOT trong bảng
pg_stat_get_live_tuples(oid)	bigint	Số hàng hoạt động trong bảng
pg_stat_get_dead_tuples(oid)	bigint	Số hàng chết trong bảng
pg_stat_get_blocks_fetched(oid)	bigint	Số yêu cầu lấy về khối đĩa cho bảng hoặc chỉ số
pg_stat_get_blocks_hit(oid)	bigint	Số yêu cầu khối đĩa được thấy trong bộ nhớ tạm cho bảng hoặc chỉ số
pg_stat_get_last_vacuum_time(oid)	timestamptz	Thời gian lần hút chân không mới nhất được người sử dụng trong bảng này khởi tạo
pg_stat_get_last_autovacuum_time(oid)	timestamptz	Thời gian lần hút chân không mới nhất được daemon hút chân không tự động trong bảng này khởi tạo
pg_stat_get_last_analyze_time(oid)	timestamptz	Thời gian phân tích lần mới nhất được người sử dụng trong bảng này khởi tạo
pg_stat_get_last_autoanalyze_time(oid)	timestamptz	Thời gian phân tích lần mới nhất được daemon hút chân không tự động trong bảng này khởi tạo
pg_backend_pid()	integer	ID tiến trình của tiến trình máy chủ được gắn vào phiên làm việc hiện hành
pg_stat_get_activity(integer)	setof record	Trả về một bản ghi thông tin về phần phụ trợ với PID được chỉ định, hoặc một bản ghi cho từng phần phụ trợ tích cực trong hệ thống nếu NULL được chỉ định. Các trường được trả về là tập con của các trường trong kiểu nhìn pg_stat_activity.
pg_stat_get_function_calls(oid)	bigint	Số lần hàm đã được gọi
pg_stat_get_function_time(oid)	bigint	Tổng thời gian đồng hồ tường bỏ ra trong hàm, theo micro giây. Bao gồm thời gian bỏ ra trong các hàm được hàm này gọi
pg_stat_get_function_self_time(oid)	bigint	Thời gian bỏ ra chỉ trong hàm này. Thời gian bỏ ra trong các hàm được gọi được loại trừ.
pg_stat_get_backend_idset()	setof integer	Tập hợp các số tiến trình máy chủ tích cực hiện hành (từ 1 tới số các tiến trình máy chủ tích cực). Xem ví dụ sử dụng trong văn bản.
pg_stat_get_backend_pid(integer)	integer	ID tiến trình của tiến trình máy chủ được đưa ra

Hàm	Dạng trả về	Mô tả
pg_stat_get_backend_dbid(oid)	integer	ID cơ sở dữ liệu tiến trình máy chủ được đưa ra
pg_stat_get_backend_userid(oid)	integer	ID người sử dụng tiến trình máy chủ được đưa ra
pg_stat_get_backend_activity(integer)	text	Lệnh tích cực của tiến trình máy chủ được đưa ra, nhưng chỉ nếu người sử dụng hiện hành là một siêu người sử dụng hoặc người sử dụng y hệt như người của phiên đang được truy vấn (và track_activities là bật)
pg_stat_get_backend_waiting(integer)	boolean	Đúng (true) nếu tiến trình máy chủ được đưa ra đang chờ một sự khóa, nhưng chỉ nếu người sử dụng hiện hành là một siêu người sử dụng hoặc người sử dụng y hệt như người của phiên làm việc đang được truy vấn (và track_activities là bật)
pg_stat_get_backend_activity_start(integer)	timestamp with time zone	Thời gian ở đó tiến trình máy chủ được đưa ra hiện đang thực thi truy vấn đã được bắt đầu, nhưng chỉ nếu người sử dụng hiện hành là một siêu người sử dụng hoặc người sử dụng y hệt như người của phiên làm việc đang được truy vấn (và track_activities là bật)
pg_stat_get_backend_xact_start(integer)	timestamp with time zone	Thời gian ở đó giao dịch đang thực thi hiện hành của tiến trình máy chủ được đưa ra đã được bắt đầu, nhưng chỉ nếu người sử dụng hiện hành là một siêu người sử dụng hoặc người sử dụng y hệt như người của phiên bản đang được truy vấn (và track_activities là bật).
pg_stat_get_backend_start(integer)	timestamp with time zone	Thời gian ở đó tiến trình máy chủ được đưa ra đã được bắt đầu, hoặc null nếu người sử dụng hiện hành không phải là siêu người sử dụng, cũng không là người sử dụng y hệt của phiên làm việc đang được truy vấn.
pg_stat_get_backend_client_addr(integer)	integer	Địa chỉ IP máy trạm được kết nối tới tiến trình máy chủ được đưa ra; null nếu kết nối đi qua một socket miền Unix, cũng null nếu người sử dụng hiện hành không là một siêu người sử dụng hoặc không phải người sử dụng y hệt của phiên đang được truy vấn.
pg_stat_get_backend_client_port(integer)	integer	Số cổng TCP của máy trạm được kết nối tới tiến trình máy chủ được đưa ra; -1 nếu kết nối đó là qua một socket miền Unix, null nếu người sử dụng hiện hành không là một siêu người sử dụng hoặc không là người sử dụng y hệt như người của phiên làm việc đang được truy vấn.
pg_stat_get_bgwriter_timed_checkpoints()	bigint	Số thời gian trình viết nền đã bắt đầu các điểm kiểm tra theo thời gian (vì thời gian checkpoint_timeout đã hết)
pg_stat_get_bgwriter_requestedcheckpoints()	bigint	Số thời gian trình viết nền đã bắt đầu các điểm kiểm tra dựa vào các yêu cầu từ các phần phụ trợ vì checkpoint_segments đã vượt quá hoặc vì lệnh CHECKPOINT đã được đưa ra.
pg_stat_get_bgwriter_buf_written_checkpoints()	bigint	Số bộ nhớ đệm được trình viết nền viết trong quá trình kiểm tra các điểm

Hàm	Dạng trả về	Mô tả
pg_stat_get_bgwriter_buf_written_clean()	bigint	Số bộ nhớ đệm được trình viết nền viết cho việc làm sạch thường ngày các trang bẩn
pg_stat_get_bgwriter_maxwrittenclean()	bigint	Số thời gian trình viết nền đã dừng quét làm sạch của nó vì nó đã viết nhiều vào bộ nhớ đệm hơn được chỉ định ở tham số bgwriter_lru_maxpages
pg_stat_get_buf_written_backend()	bigint	Số bộ nhớ đệm được các phụ trợ ghi vì chúng cần phải phân bổ một bộ nhớ đệm mới
pg_stat_get_buf_alloc()	bigint	Tổng số các phân bổ bộ nhớ đệm
pg_stat_clear_snapshot()	void	Bỏ qua hình chụp số liệu thống kê hiện hành
pg_stat_reset()	void	Thiết lập lại tất cả các con đếm số liệu thống kê cho cơ sở dữ liệu hiện hành về zero (đòi hỏi các quyền ưu tiên của siêu người sử dụng)
pg_stat_reset_shared(text)	void	Thiết lập lại vài bộ đếm số liệu thống kê được chia sẻ cho bó cơ sở dữ liệu về zero (đòi hỏi các quyền ưu tiên của siêu người sử dụng). Việc gọi pg_stat_reset_shared('bgwriter') sẽ là zero cho tất cả các giá trị được trình viết pg_stat_bg bày ra.
pg_stat_reset_single_table_counters(oid)	void	Thiết lập lại số liệu thống kê cho một bảng hoặc chỉ số duy nhất trong cơ sở dữ liệu hiện hành về zero (đòi quyền ưu tiên của siêu người sử dụng)
pg_stat_reset_single_function_counters(oid)	void	Thiết lập lại số liệu thống kê cho một hàm duy nhất trong cơ sở dữ liệu hiện hành về zero (đòi hỏi các quyền ưu tiên của siêu người sử dụng)

Lưu ý: Lấy pg_stat_get_blocks_fetched trừ đi pg_stat_get_blocks_hit sẽ có số các cuộc gọi read() của nhân được đưa ra cho bảng, chỉ số hoặc cơ sở dữ liệu; số lần đọc vật lý thực sự thường thấp hơn vì việc ghi bộ nhớ tạm mức nhân. Các cột số liệu thống kê *_blks_read sử dụng hiệu số này, như số truy cập trừ đi được lấy về.

Tất cả các hàm để truy cập thông tin về các phần phụ trợ (backend) được đánh chỉ số bằng số ID phần phụ trợ, ngoại trừ pg_stat_get_activity được đánh chỉ số bằng PID. Hàm pg_stat_get_backend_idset đưa ra một cách thức thuận tiện để so sánh một hàng cho từng tiến trình máy chủ tích cực. Ví dụ, để chỉ ra các PID và các truy vấn của tất cả các tiến trình máy chủ:

```
SELECT pg_stat_get_backend_pid(s.backendid) AS procpid,
       pg_stat_get_backend_activity(s.backendid) AS current_query
  FROM (SELECT pg_stat_get_backend_idset() AS backendid) AS s;
```

27.3. Xem các khóa

Một công cụ hữu dụng khác cho việc giám sát hoạt động của cơ sở dữ liệu là bảng hệ thống pg_locks. Nó cho phép quản trị viên cơ sở dữ liệu xem thông tin về các khóa nổi bật trong trình quản lý khóa. Ví dụ, khả năng này có thể được sử dụng để:

- Xem tất cả các khóa hiện đang nổi lên, tất cả các khóa trong các mối quan hệ trong một cơ sở dữ liệu cụ thể, tất cả các khóa trong một mối quan hệ đặc biệt, hoặc tất cả các khóa được một phiên làm việc đặc biệt của PostgreSQL nắm giữ.

- Xác định mối quan hệ trong cơ sở dữ liệu hiện hành với hầu hết các khóa không được trao (chúng có thể là một nguồn xung đột giữa các máy trạm cơ sở dữ liệu).
- Xác định ảnh hưởng của xung đột khóa trong toàn bộ hiệu năng cơ sở dữ liệu, cũng như mức độ ở đó xung đột biến đổi với toàn bộ giao thông của cơ sở dữ liệu.

Các chi tiết về kiểu nhìn pg_locks có trong Phần 45.50. Để có thêm thông tin về việc khóa và việc quản lý sự đồng thời với PostgreSQL, hãy tham chiếu tới Chương 13.

27.4. Theo dõi động

PostgreSQL đưa ra các tiện ích để hỗ trợ việc theo dõi máy chủ cơ sở dữ liệu. Điều này cho phép một tiện ích bên ngoài sẽ được gọi ở các điểm đặc biệt trong mã và vì thế theo dõi sự thực thi.

Một số điểm thăm dò hoặc theo dõi được chèn vào rồi trong mã nguồn. Các thăm dò đó có ý định sẽ được các lập trình viên và các quản trị viên cơ sở dữ liệu sử dụng. Mặc định các thăm dò đó không được biên dịch trong PostgreSQL; người sử dụng cần nói rõ ràng cho script cấu hình để làm cho các thăm dò đó sẵn sàng.

Hiện hành, chỉ tiện ích DTrace[1] được hỗ trợ, nó là sẵn sàng trong OpenSolaris, Solaris 10, và Mac OS X Leopard. Được kỳ vọng là DTrace sẽ là sẵn sàng trong tương lai trong FreeBSD và có khả năng trong các hệ điều hành khác nữa. Dự án SystemTap[2] cho Linux cũng đưa ra một thứ tương đương với DTrace. Việc hỗ trợ các tiện ích theo dõi động khác về mặt lý thuyết là có khả năng bằng việc thay đổi các định nghĩa cho các macro trong src/include/utils/probes.h.

27.4.1. Biên dịch cho theo dõi động

Mặc định, các thăm dò là không sẵn sàng, nên bạn sẽ cần nói rõ cho script cấu hình để làm cho các thăm dò đó sẵn sàng trong PostgreSQL. Để đưa hỗ trợ DTrace vào, hãy chỉ định --enable-dtrace tới cấu hình. Xem Phần 15.5 để có thêm thông tin.

27.4.2. Thăm dò được xây dựng sẵn

Một số thăm dò tiêu chuẩn được cung cấp trong mã nguồn, như được nêu trong Bảng 27-3. Nhiều hơn có thể chắc chắn được thêm vào để cải thiện khả năng quan sát của PostgreSQL.

Bảng 27-3. Các thăm dò DTrace được xây dựng sẵn

Tên	Các tham số	Mô tả
transaction-start	(LocalTransactionId)	Thăm dò diễn ra khi bắt đầu một giao dịch mới. arg0 là ID giao dịch.
transaction-commit	(LocalTransactionId)	Thăm dò diễn ra khi giao dịch kết thúc thành công. arg0 là ID giao dịch.
transaction-abort	(LocalTransactionId)	Thăm dò diễn ra khi giao dịch kết thúc thành công. arg0 là ID giao dịch.
query-start	(const char *)	Thăm dò diễn ra khi xử lý truy vấn được bắt đầu. arg0 là chuỗi truy vấn.
query-done	(const char *)	Thăm dò diễn ra khi xử lý một truy vấn hoàn tất. arg0 là chuỗi truy vấn.
query-parse-start	(const char *)	Thăm dò diễn ra khi phân tích cú pháp một truy vấn bắt đầu. arg là chuỗi truy vấn.
query-parse-done	(const char *)	Thăm dò diễn ra khi phân tích cú pháp một truy vấn hoàn tất. arg0 là

1 http://opensolaris.org/os/community/dtrace/
2 http://sourceware.org/systemtap/

Tên	Các tham số	Mô tả
		chuỗi truy vấn.
query-rewrite-start	(const char *)	Thăm dò diễn ra khi việc viết lại truy vấn bắt đầu. arg0 là chuỗi truy vấn.
query-rewrite-done	(const char *)	Thăm dò diễn ra khi việc viết lại truy vấn kết thúc. arg0 là chuỗi truy vấn.
query-plan-start	()	Thăm dò diễn ra khi việc lên kế hoạch một truy vấn bắt đầu.
query-plan-done	()	Thăm dò diễn ra khi việc lên kế hoạch một truy vấn hoàn tất.
query-execute-start	()	Thăm dò diễn ra khi sự thực thi một truy vấn bắt đầu.
query-execute-done	()	Thăm dò diễn ra khi sự thực thi một truy vấn hoàn tất.
statement-status	(const char *)	Thăm dò diễn ra bất kỳ lúc nào tiến trình máy chủ cập nhật tình trạng pg_stat_activity.current_query của nó. arg0 là chuỗi tình trạng mới.
checkpoint-start	(int)	Thăm dò diễn ra khi một điểm kiểm tra bắt đầu. arg0 nắm giữ các cờ bitwise được sử dụng để phân biệt các dạng điểm kiểm tra khác nhau, như tắt máy, ngay lập tức hoặc ép buộc.
checkpoint-done	(int, int, int, int, int)	Thăm dò diễn ra khi một điểm kiểm tra hoàn tất. (Các thăm dò được liệt kê diễn ra tuần tự trong quá trình xử lý các điểm kiểm tra). arg0 là số bộ nhớ đệm được ghi. arg1 là tổng số bộ nhớ đệm. arg2, arg3 và arg4 có số (các) tệp xlog được thêm, bị loại bỏ và được tái tạo một cách tương ứng.
clog-checkpoint-start	(bool)	Thăm dò diễn ra khi phần CLOG của một điểm kiểm tra bắt đầu. arg0 là đúng cho điểm kiểm tra thông thường, là sai cho điểm kiểm tra tắt.
clog-checkpoint-done	(bool)	Thăm dò diễn ra khi phần CLOG của một điểm kiểm tra hoàn tất. arg0 có nghĩa y hệt như đối với clog-checkpoint-start.
subtrans-checkpoint-start	(bool)	Thăm dò diễn ra khi phần SUBTRANS của một điểm kiểm tra bắt đầu. arg0 là đúng cho điểm kiểm tra thông thường, là sai cho điểm kiểm tra tắt.
subtrans-checkpoint-done	(bool)	Thăm dò diễn ra khi phần SUBTRANS của một điểm kiểm tra hoàn tất. arg0 có nghĩa y hệt như đối với subtrans-checkpoint-start.
multixact-checkpoint-start	(bool)	Thăm dò diễn ra khi phần MultiXact của một điểm kiểm tra bắt đầu. arg0 là đúng cho điểm kiểm tra thông thường, là sai cho điểm kiểm tra tắt.
multixact-checkpoint-done	(bool)	Thăm dò diễn ra khi phần MultiXact của một điểm kiểm tra là hoàn tất. arg0 có ý nghĩa y hệt như đối với multixact-checkpoint-start.
buffer-checkpoint-start	(int)	Thăm dò diễn ra khi phần ghi bộ nhớ đệm của một điểm kiểm tra bắt đầu. arg0 giữ các cờ bitwise được sử dụng để phân biệt các dạng điểm kiểm tra khác nhau, như tắt, ngay lập tức hoặc ép buộc.
buffer-sync-start	(int, int)	Thăm dò diễn ra khi chúng ta bắt đầu ghi các bộ nhớ đệm bẩn trong khi kiểm tra các điểm (sau việc nhận diện các bộ nhớ đệm nào phải được ghi). arg0 là tổng số bộ nhớ đệm. arg1 là số hiện là bẩn và cần được ghi.
buffer-sync-written	(int)	Thăm dò diễn ra sau từng bộ nhớ đệm đang được ghi trong khi kiểm tra điểm. arg0 là số ID của bộ nhớ đệm đó.
buffer-sync-done	(int, int, int)	Thăm dò diễn ra khi tất cả các bộ nhớ đệm bẩn đã được ghi. arg0 là tổng số bộ nhớ đệm. arg1 là số bộ nhớ đệm thực sự được tiến trình điểm kiểm tra ghi. arg2 là số chúng ta kỳ vọng được ghi (arg1 của buffer-sync-start); bất kỳ sự khác biệt nào cũng phản ánh các tiến trình khác đang xả các bộ nhớ đệm trong quá trình kiểm tra điểm.
buffer-checkpoint-sync-start	()	Thăm dò diễn ra sau khi các bộ nhớ đệm bẩn đã được ghi vào nhân, và trước khi bắt đầu đưa ra các yêu cầu fsync.
buffer-checkpoint-	()	Thăm dò diễn ra khi việc đồng bộ các bộ nhớ đệm tới đĩa kết thúc.

Tên	Các tham số	Mô tả
done		
twophase-checkpoint-start	()	Thăm dò diễn ra khi phần 2 pha của một điểm kiểm tra bắt đầu.
twophase-checkpoint-done	()	Thăm dò diễn ra khi phần 2 pha của một điểm kiểm tra hoàn tất.
buffer-read-start	(ForkNumber, BlockNumber, Oid, Oid, Oid, bool, bool)	Thăm dò diễn ra khi một sự đọc bộ nhớ đệm bắt đầu. arg0 và arg1 có các số rẽ nhánh và khối của trang (nhưng arg1 sẽ là -1 nếu điều này là một yêu cầu mở rộng quan hệ). arg2, arg3 và arg4 có không gian bảng, cơ sở dữ liệu và các OID quan hệ nhận diện mối quan hệ đó. arg5 là đúng cho một bộ nhớ đệm cục bộ, là sai cho một bộ nhớ đệm được chia sẻ. arg6 là đúng cho một yêu cầu mở rộng quan hệ, là sai cho đọc bình thường.
buffer-read-done	(ForkNumber, BlockNumber, Oid, Oid, Oid, bool, bool, bool)	Thăm dò diễn ra khi sự đọc bộ nhớ đệm hoàn tất. arg0 và arg1 có các số trang của rẽ nhánh và khối (nếu điều này là một yêu cầu mở rộng quan hệ, thì arg1 bây giờ có số khối của khối mới được thêm vào). arg2, arg3, và arg4 có các OID quan hệ, cơ sở dữ liệu và không gian bảng nhận diện mối quan hệ đó. arg5 là đúng cho một bộ nhớ đệm cục bộ, là sai cho một bộ nhớ đệm được chia sẻ. arg6 là đúng cho một yêu cầu mở rộng quan hệ, là sai cho sự đọc thông thường. arg7 là đúng nếu bộ nhớ đệm đã được thấy trong kho, là sai nếu không phải thế.
buffer-flush-start	(ForkNumber, BlockNumber, Oid, Oid, Oid)	Thăm dò diễn ra trước khi đưa ra bất kỳ yêu cầu ghi nào cho một bộ nhớ đệm được chia sẻ. arg0 và arg1 có các số trang của rẽ nhánh và khối. arg2, arg3 và arg4 có các OID quan hệ, cơ sở dữ liệu và không gian bảng mà đang nhận diện mối quan hệ đó.
buffer-flush-done	(ForkNumber, BlockNumber, Oid, Oid, Oid)	Thăm dò diễn ra khi một yêu cầu ghi hoàn tất. (Lưu ý rằng điều này chỉ phán ánh thời gian để truyền dữ liệu tới nhân; thường là không thực sự được ghi lên đĩa). Các đối số là y hệt cho buffer-flush-start.
buffer-write-dirty-start	(ForkNumber, BlockNumber, Oid, Oid, Oid)	Thăm dò diễn ra khi một tiến trình máy chủ bắt đầu ghi một bộ nhớ đệm bẩn. (Nếu điều này xảy ra thường xuyên, thì nó ngụ ý rằng shared_buffers là quá nhỏ hoặc các tham số kiểm soát bgwriter cần chỉnh). arg0 và arg1 có các số trang rẽ nhánh và khối. arg2, arg3 và arg4 có các OID quan hệ, cơ sở dữ liệu và không gian bảng mà đang nhận diện mối quan hệ đó.
buffer-write-dirty-done	(ForkNumber, BlockNumber, Oid, Oid, Oid)	Thăm dò diễn ra khi sự ghi một dirty-buffer hoàn tất. Các đối số là y hệt như đối với buffer-write-dirty-start.
wal-buffer-write-dirty-start	()	Thăm dò diễn ra khi một tiến trình máy chủ bắt đầu ghi một bộ nhớ đệm WAL bẩn vì không còn không gian bộ nhớ đệm WAL nào nữa sẵn sàng. (Nếu điều này xảy ra thường xuyên, nó ngụ ý rằng wal_buffers quá nhỏ.
wal-buffer-write-dirty-done	()	Thăm dò diễn ra khi sự ghi bộ nhớ đệm WAL bẩn hoàn tất.
xlog-insert	(unsigned char, unsigned char)	Thăm dò diễn ra khi một bản ghi WAL được chền vào. arg0 là trình quản lý nguồn tài nguyên (rmid) cho bản ghi đó. arg1 có các cờ thông tin.
xlog-switch	()	Thăm dò diễn ra khi sự chuyển một phân đoạn WAL được yêu cầu.
smgr-md-read-start	(ForkNumber, BlockNumber, Oid, Oid, Oid)	Thăm dò diễn ra khi bắt đầu đọc một khối từ một quan hệ. arg0 và arg1 có các số trang rẽ nhánh và khối. arg2, arg3 và arg4 có các OID quan hệ, cơ sở dữ liệu và không gian bảng đang nhận diện quan hệ đó.
smgr-md-read-done	(ForkNumber, BlockNumber, Oid, Oid, Oid, int, int)	Thăm dò diễn ra khi sự đọc một khối hoàn tất. arg0 và arg1 có các số trang rẽ nhánh và khối. arg2, arg3 và arg4 có các OID quan hệ, cơ sở dữ liệu và không gian bảng đang nhận diện quan hệ đó. arg5 là số byte thực

Tên	Các tham số	Mô tả
		sự được đọc, trong khi arg6 là số được yêu cầu (nếu chúng khác nhau thì nó chỉ ra sự lo ngại).
smgr-md-write-start	(ForkNumber, BlockNumber, Oid, Oid, Oid)	Thăm dò diễn ra khi bắt đầu viết một khối tới một quan hệ. arg0 và arg1 có các số trang rẽ nhánh và khối. arg2, arg3 và arg4 có các OID quan hệ, cơ sở dữ liệu và không gian bảng đang nhận diện quan hệ đó.
smgr-md-write-done	(ForkNumber, BlockNumber, Oid, Oid, Oid, int, int)	Thăm dò diễn ra khi sự đọc khối hoàn tất. arg0 và arg1 có các số trang rẽ nhánh và khối. arg2, arg3 và arg4 có các OID quan hệ, cơ sở dữ liệu và không gian bảng đang nhận diện quan hệ đó. arg5 là số các byte thực sự được ghi, trong khi arg6 là số được yêu cầu (nếu chúng khác nhau thì điều đó chỉ ra sự lo ngại).
sort-start	(int, bool, int, int, bool)	Thăm dò diễn ra khi một hoạt động sắp xếp bắt đầu. arg0 chỉ sắp xếp đống hoặc các dữ liệu. arg1 là đúng cho sự ép tuân thủ giá trị duy nhất. arg2 là số các cột khóa. arg3 là số kilobyte bộ nhớ làm việc được phép. arg4 là đúng nếu truy cập ngẫu nhiên tới kết quả sắp xếp được yêu cầu.
sort-done	(bool, long)	Thăm dò diễn ra khi sự sắp xếp hoàn tất. arg0 là đúng cho sắp xếp bên ngoài, là sai cho sắp xếp bên trong. arg1 là số các khối đĩa được sử dụng cho một sắp xếp bên ngoài, hoặc kilobyte bộ nhớ được sử dụng cho một sắp xếp nội bộ.
lwlock-acquire	(LWLockId, LWLockMode)	Thăm dò diễn ra khi một LWLock từng có được. arg0 là ID của LWLock. arg1 là chế độ khóa được yêu cầu, hoặc độc quyền hoặc được chia sẻ.
lwlock-release	(LWLockId)	Thăm dò diễn ra khi một LWLock được đưa ra (lưu ý rằng bất kỳ bộ chờ nào được đưa ra cũng còn chưa được tỉnh dậy). arg0 là ID của LWLock.
lwlock-wait-start	(LWLockId, LWLockMode)	Thăm dò diễn ra khi một LWLock đã không ngay lập tức sẵn sàng và một tiến trình máy chủ đã bắt đầu chờ khóa đó trở nên sẵn sàng. arg0 là ID của LWLock. arg1 là chế độ khóa được yêu cầu, độc quyền hoặc được chia sẻ.
lwlock-wait-done	(LWLockId, LWLockMode)	Thăm dò diễn ra khi một tiến trình máy chủ đã được đưa ra từ sự chờ đợi của nó đối với một LWLock (nó không thực sự có khóa). arg0 là ID của LWLock. arg1 là chế độ được yêu cầu, độc quyền hoặc được chia sẻ.
lwlock-condacquire	(LWLockId, LWLockMode)	Thăm dò diễn ra khi một LWLock đã có được thành công khi bộ gọi được chỉ định không chờ đợi. arg0 là ID của LWLock. arg1 là chế độ khóa được yêu cầu, độc quyền hoặc được chia sẻ.
lwlock-condacquire-fail	(LWLockId, LWLockMode)	Thăm dò diễn ra khi một LWLock đã có được không thành công khi bộ gọi được chỉ định không chờ đợi. arg0 là ID của LWLock. arg1 là chế độ khóa được yêu cầu, độc quyền hoặc được chia sẻ.
lock-wait-start	(unsigned int, unsigned int, unsigned int, unsigned int, unsigned int, LOCKMODE)	Thăm dò diễn ra khi một yêu cầu cho một khóa nặng ký (khóa lmgr) đã bắt đầu chờ vì khóa đó không sẵn sàng. arg0 qua arg3 là các trường thẻ đang nhận diện đối tượng đang bị khóa. arg4 chỉ dạng đối tượng đang bị khóa. arg5 chỉ dạng khóa đang được yêu cầu.
lock-wait-done	(unsigned int, unsigned int, unsigned int, unsigned int, unsigned int, LOCKMODE)	Thăm dò diễn ra khi một yêu cầu cho một khóa nặng ký (khóa lmgr) đã kết thúc chờ đợi (như, đã có được khóa). Các đối số là y hệt như với lock-wait-start.
deadlock-found	()	Thăm dò diễn ra khi một khóa chết được bộ dò tìm khóa chết tìm ra.

Bảng 27-4. Các dạng được định nghĩa, được sử dụng trong các tham số thăm dò

Dạng	Định nghĩa
LocalTransactionId	unsigned int
LWLockId	int
LWLockMode	int
LOCKMODE	int
BlockNumber	unsigned int
Oid	unsigned int
ForkNumber	int
bool	char

27.4.3. Sử dụng các thăm dò

Ví dụ bên dưới chỉ ra một script DTrace cho việc phân tích các đo đếm giao dịch trong hệ thống, như một lựa chọn để chụp hình pg_stat_database trước và sau một kiểm thử hiệu năng:

```
#!/usr/sbin/dtrace -qs
postgresql$1:::transaction-start
{
        @start["Start"] = count();
        self->ts = timestamp;
}
postgresql$1:::transaction-abort
{
        @abort["Abort"] = count();
}
postgresql$1:::transaction-commit
/self->ts/
{
        @commit["Commit"] = count();
        @time["Total time (ns)"] = sum(timestamp - self->ts);
        self->ts=0;
}
```

Khi được thực thi, script ví dụ D đưa ra đầu ra như sau:

```
# ./txn_count.d ‘pgrep -n postgres‘ or ./txn_count.d <PID>
^C

Start                          71
Commit                         70
Total time (ns)        2312105013
```

> **Lưu ý**: SystemTap sử dụng một ký hiệu khác cho các script theo dõi hơn là DTrace làm, thậm chí dù các điểm theo dõi nằm bên dưới là tương thích. Một điểm đáng lưu ý là trong việc viết này, các script SystemTap phải tham chiếu tới các tên thăm dò bằng việc sử dụng đúp bản đường gạch dưới ở chỗ của các gạch nối. Điều này được kỳ vọng sẽ được sửa trong tương lai các phiên bản SystemTap.

Bạn nên nhớ rằng các script Dtrace cần phải được viết cẩn thận và được gỡ lỗi, nếu không thì thông tin theo dõi thu thập được có thể là vô nghĩa. Trong hầu hết các trường hợp nơi mà các vấn đề được

thấy thì chính dàn phối trang bị đó có lỗi, chứ không phải hệ thống nằm bên dưới. Khi thảo luận về thông tin tìm thấy bằng việc sử dụng theo dõi động, hãy chắc chắn mở ra script được sử dụng để cho phép điều đó cũng sẽ được kiểm tra và được thảo luận.

Nhiều script ví dụ hơn có thể thấy trong dự án[3] PgFoundry dtrace.

27.4.4. Xác định thăm dò mới

Các thăm dò mới có thể được xác định trong mã bất kỳ khi nào lập trình viên mong muốn, dù điều này sẽ đòi hỏi một sự tái biên dịch. Bên dưới là các bước cho việc chèn các thăm dò mới:

1. Quyết định về các tên và dữ liệu thăm dò sẽ được làm sẵn sàng thông qua các thăm dò
2. Bổ sung thêm các định nghĩa thăm dò tới src/backend/utils/probes.d
3. Đưa vào pg_trace.h nếu nó không hiện diện rồi trong (các) module có các điểm thăm dò, và chèn macro thăm dò TRACE_POSTGRESQL vào các vị trí mong muốn trong mã nguồn
4. Tái biên dịch và kiểm tra hợp lệ xem các thăm dò mới có là sẵn sàng

Ví dụ: Đây là ví dụ về cách mà bạn có thể thêm một thăm dò để theo dõi tất cả các giao dịch mới bằng ID giao dịch.

1. Quyết định rằng thăm dò đó sẽ được đặt tên là transaction-start và đòi hỏi một tham số dạng LocalTransactionId
2. Thêm định nghĩa thăm dò tới src/backend/utils/probes.d:

   ```
   probe transaction__start(LocalTransactionId);
   ```

   Lưu ý sử dụng đúp bản đường gạch dưới trong tên thăm dò đó. Trong một script DTrace có sử dụng thăm dò đó, đúp bản đường gạch chân cần phải được thay thế bằng một dấu gạch nối, nên transaction-start là tên cho tài liệu đối với những người sử dụng.
3. Lúc biên dịch transaction__start được chuyển đổi thành một macro có tên là TRACE_POSTGRESQL_TRANSACTION_START (lưu ý các đường gạch chân là đơn ở đây), điều là sẵn sàng bằng việc đưa vào pg_trace.h. Hãy thêm lời gọi macro vào vị trí phù hợp trong mã nguồn. Trong trường hợp này, nó trông giống như sau:

   ```
   TRACE_POSTGRESQL_TRANSACTION_START(vxid.localTransactionId);
   ```
4. Sau việc biên dịch lại và chạy tệp nhị phân mới, hãy kiểm tra xem thăm dò mới được thêm vào của bạn có là sẵn sàng bằng việc thực thi lệnh DTrace sau hay không. Bạn nên xem đầu ra tương tự:

   ```
   # dtrace -ln transaction-start
   ID    PROVIDER            MODULE          FUNCTION NAME
   18705 postgresql49878     postgres        StartTransactionCommand transaction-start
   18755 postgresql49877     postgres        StartTransactionCommand transaction-start
   18805 postgresql49876     postgres         StartTransactionCommand transaction-start
   18855 postgresql49875     postgres        StartTransactionCommand transaction-start
   18986 postgresql49873     postgres        StartTransactionCommand transaction-start
   ```

Có ít điều phải cẩn thận khi thêm các macro theo dõi tới mã lệnh C:

3 http://pgfoundry.org/projects/dtrace/

- Bạn nên cẩn thận rằng các dạng dữ liệu được chỉ định cho các tham số của một thăm dò khớp với các dạng dữ liệu của các biến được sử dụng trong macro đó. Nếu không, bạn sẽ có các lỗi biên dịch.
- Trong hầu hết các nền tảng, nếu PostgreSQL được xây dựng với --enable-dtrace, thì các đối số đối với một macro theo dõi sẽ được đánh giá bất kỳ khi nào kiểm soát truyền qua macro đó, thậm chí nếu không việc theo dõi nào đang được thực hiện. Điều này thường không đáng lo nếu bạn chỉ đang báo cáo các giá trị của ít biến cục bộ.

Nhưng phải biết về việc đặt ra các lời gọi hàm đắt giá trong các đối số đó. Nếu bạn cần làm điều đó, hãy cân nhắc việc bảo vệ macro đó bằng một kiểm tra để thấy liệu sự theo dõi đó có thực sự được nhúng vào hay không:

```
if      (TRACE_POSTGRESQL_TRANSACTION_START_ENABLED())
        TRACE_POSTGRESQL_TRANSACTION_START(some_function(...));
```

Mỗi macro theo dõi có một macro ENABLED tương ứng.

Chương 28. Giám sát sử dụng đĩa

Chương này thảo luận cách giám sát sử dụng đĩa của một hệ thống cơ sở dữ liệu PostgreSQL.

28.1. Xác định sử dụng đĩa

Từng bảng có một tệp đống đĩa ban đầu nơi mà hầu hết các dữ liệu được lưu trữ. Nếu bảng đó có bất kỳ cột nào với các giá trị rộng tiềm năng, thì cũng có thể sẽ là một tệp TOAST có liên quan tới bảng đó, nó được sử dụng để lưu trữ các giá trị quá rộng không vừa trong bảng chính (xem Phần 54.2). Sẽ có một chỉ số trong bảng TOAST, nếu hiện diện. Cũng có thể có các chỉ số có liên quan tới bảng cơ bản đó.

Từng bảng và chỉ số được lưu trữ trong một tệp đĩa riêng rẽ - có khả năng hơn một tệp, nếu tệp đó có thể vượt quá 1 GB. Các qui ước đặt tên cho các tệp đó được mô tả trong Phần 54-1. Bạn có thể giám sát không gian đĩa theo 3 cách: bằng việc sử dụng các hàm SQL được liệt kê trong Bảng 9-58, bằng việc sử dụng các công cụ trong contrib/oid2name, hoặc bằng việc sử dụng điều tra bằng tay các catalog hệ thống. Các hàm SQL là dễ dàng nhất để sử dụng và thường được khuyến cáo. contrib/oid2name được mô tả trong Phần F.19. Phần còn lại của phần này chỉ cách làm điều này bằng điều tra các catalog hệ thống.

Sử dụng psql trong một cơ sở dữ liệu vừa được hút chân không hoặc được phân tích, bạn có thể đưa ra các truy vấn để xem sử dụng đĩa của bất kỳ bảng nào:

```
SELECT pg_relation_filepath(oid), relpages FROM pg_class WHERE relname = 'customer';

pg_relation_filepath | relpages
----------------------+----------
base/16384/16806  |       60
(1 row)
```

Từng trang thường là 8KB. (Hãy nhớ, relpages chỉ được cập nhật bằng VACUUM, ANALYZE, và một ít các lệnh DDL như CREATE INDEX). Tên đường dẫn tệp là mối quan tâm nếu bạn muốn xem xét trực tiếp tệp trên đĩa của bảng đó.

Để chỉ ra không gian được các bảng TOAST sử dụng, hãy sử dụng một truy vấn giống như sau:

```
SELECT relname, relpages
FROM pg_class,
        (SELECT reltoastrelid
        FROM pg_class
        WHERE relname = 'customer') AS ss
WHERE oid = ss.reltoastrelid OR
        oid = (  SELECT reltoastidxid
                FROM pg_class
                WHERE oid = ss.reltoastrelid)
ORDER BY relname;

relname                  | relpages
-------------------------------+----------
pg_toast_16806          |        0
pg_toast_16806_index  |        1
```

Bạn cũng có thể dễ dàng hiển thị các kích cỡ chỉ số:

```
SELECT c2.relname, c2.relpages
FROM pg_class c, pg_class c2, pg_index i
WHERE c.relname = 'customer' AND
        c.oid = i.indrelid AND
        c2.oid = i.indexrelid
ORDER BY c2.relname;
relname | relpages
----------------------+----------
customer_id_indexdex | 26
```

Dễ thấy các bảng và chỉ số lớn nhất của bạn bằng việc sử dụng thông tin này:

```
SELECT relname, relpages
FROM pg_class
ORDER BY relpages DESC;

relname          | relpages
---------------------+--------------
bigtable         |      3290
customer         |      3144
```

28.2. Hỏng vì đĩa đầy

Nhiệm vụ giám sát đĩa quan trọng nhất đối với một người quản trị cơ sở dữ liệu là phải chắc chắn đĩa không bị đầy. Một đĩa dữ liệu được xả sẽ không gây ra sự hỏng dữ liệu, nhưng nó có thể ngăn cản hoạt động hữu dụng xảy ra. Nếu đĩa nắm giữ các tệp WAL tăng đầy, thì sự sợ hãi máy chủ cơ sở dữ liệu và sự tắt sau đó có thể xảy ra.

Nếu bạn không thể giải phóng không gian bổ sung trên đĩa bằng việc xóa các thứ khác, thì bạn có thể chuyển vài tệp cơ sở dữ liệu sang các hệ thống khác bằng việc sử dụng các không gian bảng. Xem Phần 21.6 để có thêm thông tin về điều đó.

> **Mẹo**: Vài hệ thống tệp thực thi tồi khi chúng hầu như đầy, nên đừng chờ cho tới khi đĩa đầy hoàn toàn mới hành động.

Nếu hệ thống của bạn hỗ trợ chỉ tiêu đĩa (quota) theo từng người sử dụng, thì cơ sở dữ liệu tự nhiên sẽ tuân theo bất kỳ chỉ tiêu nào được thay thế về người sử dụng đó khi máy chủ chạy. Việc vượt quá quota sẽ có các hiệu ứng tồi tệ y hệt như việc dùng hết hoàn toàn không gian đĩa.

Chương 29. Độ tin cậy và lưu ký ghi trước

Chương này giải thích cách lưu ký ghi trước (Write-Ahead Log) được sử dụng để có được hoạt động có hiệu quả, tin cậy.

29.1. Độ tin cậy

Độ tin cậy là một thuộc tính quan trọng của bất kỳ hệ thống cơ sở dữ liệu nghiêm túc nào, và PostgreSQL làm mọi điều có thể để đảm bảo hoạt động tin cậy được. Một khía cạnh của hoạt động tin cậy là tất cả dữ liệu được một giao dịch thực hiện ghi lại sẽ được lưu trữ trong một khu vực không dễ bay hơi mà là an toàn đối với việc mất điện, hỏng hệ điều hành, và hỏng phần cứng (ngoại trừ hỏng bản thân khu vực không dễ bay hơi đó, tất nhiên). Việc ghi thành công các dữ liệu đó vào kho vĩnh viễn của máy tính (ổ đĩa hoặc tương tự) thường đáp ứng được yêu cầu này. Trên thực tế, thậm chí nếu một máy tính bị hỏng hoàn toàn, nếu các ổ đĩa sống sót thì chúng có thể được chuyển sang một máy tính khác với phần cứng tương tự và tất cả các giao dịch được thực hiện sẽ vẫn được giữ nguyên.

Trong khi việc ép các dữ liệu tới các đĩa định kỳ có thể xem giống như một hoạt động đơn giản, thì nó không phải thế. Vì các ổ đĩa là chậm hơn nhiều so với bộ nhớ chính và CPU, vài lớp nhớ tạm tồn tại giữa bộ nhớ chính và các đĩa của máy tính. Trước hết, có bộ nhớ tạm của bộ nhớ đệm của hệ điều hành, nó nhớ tạm thường xuyên các khối đĩa được yêu cầu và kết hợp các ghi đĩa. May thay, tất cả các hệ điều hành trao cho các ứng dụng một cách thức để áp các cuộc ghi từ bộ nhớ tạm của bộ nhớ đệm vào đĩa, và PostgreSQL sử dụng các tính năng đó. (Xem tham số wal_sync_method để tinh chỉnh cách mà điều này được thực hiện).

Tiếp theo, có thể là một bộ nhớ tạm trong trình kiểm soát ổ đĩa; điều này thường là phổ biến trong các card kiểm soát RAID. Một vài bộ nhớ tạm đó là ghi qua (write-through) được, nghĩa là các cuộc ghi sẽ được chuyển tới ổ ngay khi chúng tới. Các bộ nhớ tạm khác là ghi ngược (write-back), nghĩa là dữ liệu được gửi tới ổ lúc nào đó sau này. Các bộ nhớ tạm như vậy có thể là một mối nguy về độ tin cậy vì bộ nhớ trong bộ nhớ tạm của trình kiểm soát đĩa là dễ bay hơi, và sẽ đánh mất nội dung của nó khi mất điện. Các card kiểm soát tốt hơn có các đơn vị sao lưu pin - BBU (Battery - Backup Unit), nghĩa là card có một bộ pin duy trì năng lượng cho bộ nhớ tạm trong trường hợp mất điện hệ thống. Sau khi điện được phục hồi thì dữ liệu sẽ được ghi vào ổ đĩa.

Và cuối cùng, hầu hết các ổ đĩa có các bộ nhớ tạm. Vài ổ là ghi qua trong khi vài ổ khác là ghi ngược, và các mối quan tâm y hệt về mất dữ liệu tồn tại đối với các bộ nhớ tạm ổ ghi ngược như đối với các bộ nhớ tạm trình kiểm tra đĩa. Các ổ IDE và SATA mức người tiêu dùng đặc biệt có khả năng có các bộ nhớ tạm ghi ngược mà sẽ không sống sót được nếu mất điện. Nhiều ổ tình trạng cứng - SSD (Solid - State Drive) cũng có các bộ nhớ tạm ghi ngược dễ bay hơi.

Các bộ nhớ tạm đó thường có thể được vô hiệu hóa; tuy nhiên, phương pháp đó cho việc thực hiện điều này biến hóa theo các hệ điều hành và dạng ổ:

- Trên Linux, các ổ IDE có thể được truy vấn bằng việc sử dụng hdparm -I; việc nhớ tạm ghi được kích hoạt nếu có một dấu * bên cạnh Write cache. hdparm -W có thể được sử dụng để tắt

việc nhớ tạm ghi. Các ổ SCSI có thể được truy vấn bằng việc sử dụng sdparm[1]. Hãy sử dụng sdparm --get=WCE để kiểm tra liệu bộ nhớ tạm ghi có được kích hoạt hay không và sdparm --clear=WCE để vô hiệu hóa nó.

- Trên BSD, các ổ IDE có thể được truy vấn bằng việc sử dụng atacontrol, và các ổ SCSI sử dụng sdparm.
- Trên Solaris, bộ nhớ tạm ghi đĩa được kiểm soát bằng format -e[2]. (Hệ thống tệp ZFS của Solaris là an toàn với bộ nhớ tạm ghi đĩa được kích hoạt vì nó đưa ra các lệnh làm đầy bộ nhớ tạm đĩa của riêng nó).
- Trên Windows, nếu wal_sync_method là open_datasync (mặc định), thì việc nhớ tạm ghi có thể bị vô hiệu hóa bằng việc bỏ chọn My Computer\Open\disk drive\Properties\Hardware\Properties\Policies\Enable write caching on the disk. Như một lựa chọn, hãy thiết lập wal_sync_method về fsync hoặc fsync_writethrough, điều ngăn chặn được việc nhớ tạm ghi.
- Trên Mac OS X, việc nhớ tạm ghi có thể được ngăn chặn bằng việc thiết lập wal_sync_method về fsync_writethrough.

Các ổ SATA gần đây (các ổ sau ATAPI-6 hoặc sau này) đưa ra một lệnh xả bộ nhớ tạm một ổ (FLUSH CACHE EXT), trong khi các ổ SCSI từ lâu đã hỗ trợ một lệnh tương tự SYNCHRONIZE CACHE. Các lệnh đó không trực tiếp truy cập được tới PostgreSQL, nhưng một vài hệ thống tệp (như, ZFS, ext4) có thể sử dụng chúng để xả dữ liệu tới các đĩa trong các ổ được kích hoạt ghi ngược. Không may, các hệ thống tệp như vậy hành xử dưới mức tối ưu khi được kết hợp với các trình kiểm soát đĩa của các đơn vị sao lưu pin BBU. Trong các thiết lập như vậy, lệnh đồng bộ hóa ép tất cả các dữ liệu từ bộ nhớ tạm của trình kiểm soát vào các đĩa, loại bỏ nhiều lợi ích của BBU. Bạn có thể chạy tiện ích src/tools/fsync trong cây nguồn của PostgreSQL để thấy liệu bạn có bị ảnh hưởng hay không. Nếu bạn bị ảnh hưởng, thì những lợi ích về hiệu năng của BBU có thể giành lại được bằng việc tắt các pin ghi trong hệ thống tệp hoặc thiết lập lại cấu hình cho trình kiểm soát đĩa, nếu điều đó là một lựa chọn. Nếu các rào cản ghi được tắt, hãy chắc chắn pin vẫn hoạt động; một pin có lỗi có thể tiềm tàng dẫn tới mất dữ liệu. Hy vọng những người thiết kế hệ thống tệp và trình kiểm soát đĩa cuối cùng sẽ giải quyết được hành vi dưới mức tối ưu này.

Khi hệ điều hành gửi một yêu cầu ghi tới phần cứng lưu trữ, có ít điều nó có thể làm để chắc chắn dữ liệu đã tới ở khu vực kho lưu trữ thực sự không dễ bay hơi. Thay vào đó, đây là trách niệm của người quản trị để chắc chắn rằng tất cả các thành phần kho lưu trữ đảm bảo được tính toàn vẹn dữ liệu. Tránh các trình kiểm soát đĩa mà có các bộ nhớ tạm ghi không được pin hỗ trợ. Ở mức ổ, hãy vô hiệu hóa việc nhớ tạm ghi ngược nếu ổ không thể đảm bảo dữ liệu sẽ được ghi trước khi tắt. Nếu bạn sử dụng các SSD, hãy hiểu rằng nhiều trong số chúng không trung thực với các lệnh xả bộ nhớ tạm theo mặc định. Bạn có thể kiểm tra hành vi về độ tin cậy của các hệ thống con I/O bằng việc sử dụng diskchecker.pl[3].

1 http://sg.danny.cz/sg/sdparm.html
2 http://www.sun.com/bigadmin/content/submitted/format_utility.jsp
3 http://brad.livejournal.com/2116715.html

Một rủi ro khác về mất dữ liệu được đặt ra từ bản thân các hoạt động ghi đĩa. Các đĩa được chia thành các phân khu (sector), thường là 512 byte mỗi phân khu. Mỗi hoạt động đọc hoặc ghi vật lý xử lý toàn bộ một phân khu. Khi một yêu cầu ghi tới ổ, nó có thể là cho vài bội số của 512 byte (PostgreSQL thường ghi 8192 byte, hoặc 16 phân khu, cùng một lúc), và tiến trình ghi có thể hỏng vì mất điện bất kỳ lúc nào, nghĩa là một vài phân khu 512 byte đã được ghi trong khi các phân khu khác thì chưa. Để bảo vệ chống lại những hỏng hóc như vậy, PostgreSQL định kỳ ghi đầy các ảnh các trang tới kho lưu trữ WAL vĩnh viễn trước khi sửa đổi trang thực tế trên đĩa. Bằng cách này, trong quá trình phục hồi hỏng hóc thì PostgreSQL có thể phục hồi các trang ghi một phần từ WAL. Nếu bạn có phần mềm hệ thống tệp mà ngăn chặn được các ghi trang một phần (như, ZFS), thì bạn có thể tắt ảnh trang này bằng việc tắt tham số full_page_writes. Các trình kiểm soát đĩa BBU không ngăn chặn được các ghi trang một phần trừ phi chúng đảm bảo rằng dữ liệu được ghi tới BBU như là các trang đầy đủ (8KB).

29.2. Lưu ký ghi trước (WAL)

Lưu ký ghi trước - WAL (Write-Ahead Logging) là một phương pháp tiêu chuẩn cho việc đảm bảo tính toàn vẹn dữ liệu. Một mô tả chi tiết có thể thấy trong hầu hết (nếu không nói là tất cả) các sách về xử lý giao dịch. Ngắn gọn, khái niệm trọng tâm của WAL là những thay đổi tới các tệp dữ liệu (nơi mà các bảng và chỉ số được đặt) phải được ghi chỉ sau khi những thay đổi đó đã được lưu ký, đó là, sau việc mô tả các bản ghi lưu ký các thay đổi đã được xả tới kho lưu trữ vĩnh viễn. Nếu chúng ta tuân theo thủ tục này, thì chúng ta không cần xả các trang dữ liệu vào đĩa trong mỗi lần thực hiện giao dịch, vì chúng ta biết rằng trong trường hợp bị hỏng thì chúng ta có khả năng phục hồi cơ sở dữ liệu bằng việc sử dụng lưu ký đó: bất kỳ thay đổi nào từng được áp dụng cho các trang dữ liệu có thể làm lại được từ các bản ghi lưu ký đó. (Đây là sự phục hồi lăn tiến (roll-forward), còn được biết tới như là làm lại REDO).

> **Mẹo**: Vì WAL phục hồi các nội dung tệp cơ sở dữ liệu sau một hỏng hóc, các hệ thống tệp được ghi nhật ký không nhất thiết là để lưu trữ tin cậy các tệp dữ liệu hoặc các tệp WAL. Trong thực tế, tổng chi phí của việc ghi nhật ký có thể làm giảm hiệu năng, đặc biệt nếu việc ghi nhật ký gây ra cho dữ liệu hệ thống tệp sẽ bị xả vào đĩa. May thay, việc xả dữ liệu trong quá trình ghi nhật ký có thể thường bị vô hiệu hóa bằng một lựa chọn kích hoạt hệ thống tệp, như data=writeback trên một hệ thống tệp Linux ext3. Các hệ thống tệp được ghi nhật ký cải thiện thúc đẩy tốc độ sau hỏng hóc.

Việc sử dụng WAL làm cho số lượng ghi đĩa giảm đáng kể, vì chỉ tệp lưu ký nào cần phải được xả tới đĩa để đảm bảo rằng một giao dịch được thực hiện, thay vì mọi tệp dữ liệu được giao dịch đó thay đổi. Tệp lưu ký được viết một cách tuần tự, và vì thế chi phí của việc đồng bộ lưu ký là ít hơn nhiều so với chi phí xả các trang dữ liệu. Điều này đặc biệt đúng cho các máy chủ điều khiển nhiều giao dịch nhỏ động chạm tới các phần khác nhau của kho dữ liệu. Hơn nữa, khi máy chủ đang xử lý nhiều giao dịch nhỏ đồng thời, một fsync của tệp lưu ký có thể đủ để thực hiện nhiều giao dịch.

WAL cũng làm cho có khả năng để hỗ trợ sao lưu trực tuyến và phục hồi đúng thời điểm, như được mô tả trong Phần 24.3. Bằng việc lưu trữ dữ liệu WAL chúng ta có thể hỗ trợ việc lật lại về bất kỳ sự việc nào theo thời gian được dữ liệu WAL sẵn có bao trùm: chúng ta đơn giản cài đặt một sao lưu

vật lý cơ sở dữ liệu trước đó, và chơi lại lưu ký WAL ngay lúc mong muốn. Hơn nữa, sao lưu vật lý không phải là một hình chụp tức thì tình trạng cơ sở dữ liệu đó - nếu nó được làm qua một vài giai đoạn thời gian, thì việc chơi lại lưu ký WAL cho giai đoạn đó sẽ cố định lại bất kỳ những sự không ổn định nội bộ nào.

29.3. Thực hiện không đồng bộ

Thực hiện không đồng bộ là một lựa chọn cho phép các giao dịch hoàn tất nhanh hơn, với chi phí mà hầu hết các giao dịch gần đây có thể bị mất nếu cơ sở dữ liệu bị hỏng. Trong nhiều ứng dụng điều này là một sự lựa chọn chấp nhận được.

Như được mô tả trong phần trước, thực hiện (commit) giao dịch thường là đồng bộ: máy chủ chờ các bản ghi WAL của giao dịch sẽ được xả tới kho vĩnh viễn trước khi trả về một chỉ số thành công cho máy trạm. Máy trạm vì thế được đảm bảo rằng giao dịch được báo cáo được thực hiện (commit) sẽ được giữ lại, thậm chí trong trường hợp máy chủ hỏng ngay lập tức sau đó. Tuy nhiên, đối với các giao dịch ngắn thì độ trễ này là một thành phần chính của tổng thời gian giao dịch. Việc lựa chọn chế độ thực hiện (commit) không đồng bộ có nghĩa là máy chủ đó trả về thành công ngay khi giao dịch được hoàn tất về mặt logic, trước khi các bản ghi WAL mà nó đã tạo ra thực sự đã trên đường của chúng tới đĩa. Điều này có thể cung cấp một sự bùng nổ đáng kể về thông lượng đối với các giao dịch nhỏ.

Thực hiện không đồng bộ đưa ra rủi ro mất dữ liệu. Có thời gian ngắn cửa sổ giữa báo cáo hoàn tất giao dịch đối với máy trạm và thời gian mà giao dịch đó thực sự được thực hiện (đó là, nó được đảm bảo không bị mất nếu máy chủ bị hỏng). Vì thế thực hiện không đồng bộ sẽ không được sử dụng nếu máy trạm sẽ lấy các hành động bên ngoài dựa vào giả thiết rằng giao dịch đó sẽ được ghi nhớ. Như một ví dụ, một ngân hàng có thể chắc chắn không sử dụng thực hiện không đồng bộ cho một giao dịch ghi lại việc phân phối tiền mặt của một máy ATM. Nhưng trong nhiều kịch bản, như việc lưu ký sự kiện, không cần một đảm bảo mạnh dạng này.

Rủi ro là có bằng việc sử dụng thực hiện không đồng bộ là mất dữ liệu, chứ không phải hỏng dữ liệu. Nếu cơ sở dữ liệu bị hỏng, thì nó sẽ phục hồi bằng việc chơi lại WAL cho tới bản ghi cuối cùng từng được xả. Cơ sở dữ liệu đó vì thế sẽ được phục hồi tới một tình trạng tự ổn định, nhưng bất kỳ giao dịch nào từng chưa được xả vào đĩa sẽ không được phản ánh trong tình trạng đó. Ảnh hưởng tổng thể vì thế là mất một ít các giao dịch cuối cùng. Vì các giao dịch đó được chơi lại theo trật tự thực hiện, không có sự bất ổn định nào có thể có - ví dụ, nếu giao dịch B đã thực hiện các thay đổi dựa vào các kết quả của một giao dịch A trước đó, thì nó không có khả năng đối với các kết quả của A sẽ bị mất trong khi các kết quả của B được giữ lại.

Người sử dụng có thể chọn chế độ thực hiện của từng giao dịch, sao cho có khả năng có cả các giao dịch thực hiện đồng bộ và không đồng bộ chạy đồng thời. Điều này cho phép lựa chọn cân nhắc giữa hiệu năng và độ chắc chắn về tính bền lâu của giao dịch. Chế độ thực hiện được kiểm soát bằng tham số do người sử dụng thiết lập synchronous_commit, nó có thể bị thay đổi theo bất kỳ cách thức nào mà một tham số cấu hình có thể được thiết lập. Chế độ được sử dụng cho bất kỳ một giao dịch nào phụ thuộc vào giá trị của synchronous_commit khi giao dịch thực hiện bắt đầu.

Các lệnh tiện ích nhất định, ví dụ DROP TABLE, bị ép để thực hiện đồng bộ bất kể việc thiết lập synchronous_commit. Điều này là để đảm bảo ổn định giữa hệ thống tệp của máy chủ và tình trạng logic của cơ sở dữ liệu. Các lệnh hỗ trợ thực hiện 2 pha (two-phase commit), như PREPARE TRANSACTION, cũng sẽ luôn là đồng bộ.

Nếu cơ sở dữ liệu hỏng trong quá trình cửa sổ rủi ro giữa một sự thực hiện không đồng bộ và việc ghi các bản ghi WAL của giao dịch đó, thì những thay đổi được thực hiện trong quá trình giao dịch đó sẽ bị mất. Khoảng thời gian cửa sổ rui ro bị giới hạn vì một tiến trình nền ("trình ghi WAL") xả các bản ghi WAN không được ghi tới đĩa mỗi wal_writer_delay mili giây. Khoảng thời gian tối đa thực tế của cửa sổ rủi ro là gấp 3 lần wal_writer_delay vì trình ghi WAL được thiết kế để có lợi cho việc ghi toàn bộ các trang một lúc trong các giai đoạn bận rộn.

Cảnh báo

Một sự tắt chế độ ngay lập tức là tương đương với một sự hỏng máy chủ, và vì thế sẽ gây ra mất mát bất kỳ thực hiện không đồng bộ còn chưa được xả nào.

Thực hiện không đồng bộ đưa ra hành xử khác với việc thiết lập fsync = off. fsync là một thiết lập rộng khắp máy chủ mà sẽ tùy biến hành xử của tất cả các giao dịch. Nó vô hiệu hóa tất cả logic bên trong PostgreSQL mà cố đồng bộ hóa các cuộc ghi tới các phần khác nhau của cơ sở dữ liệu, và vì thế một sự hỏng hệ thống (đó là, hỏng một phần cứng hoặc hệ điều hành, không phải một sự hỏng hóc của bản thân PostgreSQL) có thể gây ra sự hỏng tồi tệ tùy ý tình trạng của cơ sở dữ liệu. Trong nhiều kịch bản, thực hiện không đồng bộ đưa ra hầu hết sự cải thiện hiệu năng mà có thể có được bằng việc tắt fsync, nhưng không có rủi ro hỏng dữ liệu.

commit_delay cũng có vẻ rất giống với thực hiện không đồng bộ, như nó thực sự là một phương pháp thực hiện đồng bộ (trong thực tế, commit_delay bị bỏ qua trong quá trình một thực hiện không đồng bộ). commit_delay gây ra một sự trễ ngay trước khi một thực hiện đồng bộ cố xả WAL tới đĩa, với hy vọng rằng một sự xả duy nhất được thực thi bằng một giao dịch như vậy cũng có thể phục vụ cho việc thực hiện các giao dịch khác trong khoảng thời gian y hệt. Việc thiết lập commit_delay chỉ có thể giúp khi có nhiều giao dịch thực hiện đồng thời, và là khó để tinh chỉnh nó về một giá trị mà thực sự giúp thay vì gây thiệt hại cho thông lượng.

29.4. Cấu hình WAL

Có vài tham số cấu hình có liên quan tới WAL mà ảnh hưởng tới hiệu năng của cơ sở dữ liệu.

Phần này giải thích sử dụng của chúng. Xem Chương 18 để có thông tin chung về việc thiết lập các tham số cấu hình máy chủ.

Các điểm kiểm tra () là các điểm trong tuần tự các giao dịch mà ở đó được đảm bảo rằng các tệp dữ liệu chỉ số và đống đã được cập nhật với tất cả các thông tin được ghi trước điểm kiểm tra đó. Tại thời điểm của điểm kiểm tra, tất cả các trang dữ liệu bẩn được xả tới đĩa và một bản ghi điểm kiểm tra đặc biệt được ghi tới tệp lưu ký. (Các thay đổi trước đó đã được xả tới các tệp WAL). Trong trường hợp có sự hỏng, thì thủ tục phục hồi hỏng hóc xem xét bản ghi điểm kiểm tra mới nhất để

xác định điểm đó trong lưu ký (được biết tới như là bản ghi làm lại) từ đó nó sẽ bắt đầu hành động REDO. Bất kỳ thay đổi nào được làm đối với các tệp dữ liệu trước điểm đó sẽ được đảm bảo sẵn sàng rồi trên đĩa. Vì thế, sau một điểm kiểm tra, các phân đoạn lưu ký xử lý phân đoạn có chứa bản ghi làm lại sẽ không còn cần thiết và có thể được tái tạo hoặc loại bỏ. (khi việc lưu trữ WAL đang được tiến hành, các phân đoạn lưu ký phải được lưu trữ trước khi được tái tạo hoặc loại bỏ).

Yêu cầu của điểm kiểm tra cho việc xả tất cả các trang dữ liệu bẩn tới đĩa có thể gây ra một tải I/O đáng kể. Vì lý do này, hoạt động của điểm kiểm tra được tiết lưu sao cho I/O bắt đầu ở đầu điểm kiểm tra và kết thúc trước khi điểm kiểm tra tiếp sau bắt đầu; điều này giảm tối thiểu sự xuống cấp hiệu năng trong quá trình kiểm tra các điểm.

Tiến trình của trình ghi nền của máy chủ tự động tiến hành một điểm kiểm tra rất thường xuyên. Một điểm kiểm tra được tạo ra cho mỗi phân đoạn lưu ký checkpoint_segments, hoặc mỗi checkpoint_timeout giây, bất kể điều gì tới trước. Các thiết lập mặc định là 3 phân đoạn và 300 giây (5 phút), một cách tương ứng. Cũng có khả năng ép một điểm kiểm tra bằng việc sử dụng lệnh SQL CHECKPOINT.

Việc giảm checkpoint_segments và/hoặc checkpoint_timeout làm cho các điểm kiểm tra phải xảy ra thường xuyên hơn. Điều này cho phép phục hồi sau hỏng hóc nhanh hơn (vì ít công việc sẽ cần phải làm lại hơn). Tuy nhiên, người ta phải cân bằng điều này với chi phí gia tăng của việc xả các trang dữ liệu bẩn thường xuyên hơn. Nếu full_page_writes được thiết lập (như là mặc định), thì sẽ có yếu tố khác để cân nhắc. Để đảm bảo sự ổn định các trang dữ liệu, sửa đổi đầu tiên một trang dữ liệu sau từng điểm kiểm tra gây ra việc lưu ký toàn bộ nội dung trang. Trong trường hợp đó, một khoảng thời gian điểm kiểm tra nhỏ hơn làm gia tăng lượng đầu ra tới lưu ký WAL, một phần phủ định mục tiêu của việc sử dụng một khoảng thời gian nhỏ hơn, và trong bất kỳ trường hợp nào cũng làm cho I/O đĩa nhiều hơn.

Các điểm kiểm tra là khá đắt giá, trước hết vì chúng đòi hỏi việc ghi ra tất cả các bộ nhớ đệm bẩn, và thứ 2 là vì chúng gây ra giao thông WAL thêm ra tiếp sau như được thảo luận ở trên. Vì thế là khôn ngoan để thiết lập các tham số điểm kiểm tra đủ cao để các điểm kiểm tra không xảy ra quá thường xuyên. Như một kiểm tra vệ sinh đơn giản về các tham số điểm kiểm tra của bạn, bạn có thể thiết lập tham số checkpoint_warning. Nếu các điểm kiểm tra xảy ra sát nhau hơn checkpoint_warning giây đồng hồ, thì một thông điệp sẽ được đưa ra cho lưu ký máy chủ khuyến cáo tăng checkpoint_segments. Sự xuất hiện đặc biệt của một thông điệp như vậy không phải là lý do báo động, mà nếu nó xuất hiện thường xuyên thì các tham số kiểm soát điểm kiểm tra nên được tăng. Bó các hoạt động như các vụ truyền COPY lớn có thể là lý do cho một số cảnh báo như vậy xuất hiện nếu bạn không có checkpoint_segments đủ cao.

Để tránh việc làm ngập hệ thống I/O bằng một sự bùng nổ ghi trang, việc ghi các bộ nhớ đệm bẩn trong quá trình điểm kiểm tra sẽ được lan truyền qua một khoảng thời gian. Khoảng thời gian đó được kiểm soát bằng checkpoint_completion_target, nó được đưa ra như một phần nhỏ của khoảng thời gian điểm kiểm tra đó. Tốc độ I/O được điều chỉnh sao cho điểm kiểm tra đó hoàn tất khi phần được đưa ra đó của các phân đoạn WAL checkpoint_segments đã được tiêu dùng kể từ khi điểm kiểm tra bắt đầu, hoặc phần được đưa ra đó của checkpoint_timeout giây đồng hồ đã trôi qua, bất kỳ điều

gì tới sớm hơn. Với giá trị mặc định là 0.5, PostgreSQL có thể được kỳ vọng sẽ hoàn tất từng điểm kiểm tra trong khoảng nửa thời gian trước khi điểm kiểm tra tiếp theo bắt đầu. Trong một hệ thống mà điều đó rất sát với thông lượng I/O cực đại trong hoạt động bình thường, thì bạn có thể muốn tăng checkpoint_completion_target để giảm tải I/O từ các điểm kiểm tra. Nhược điểm của điều này là việc kéo dài thời gian các điểm kiểm tra ảnh hưởng tới thời gian phục hồi, vì nhiều phân đoạn WAL hơn sẽ cần phải được giữ để có thể sử dụng trong phục hồi. Dù checkpoint_completion_target có thể được thiết lập cao như 1.0, thì tốt nhất hãy giữ cho nó ít hơn điều đó (có thể hầu hết là 0.9) vì các điểm kiểm tra bao gồm một vài hoạt động khác ngoài việc ghi các bộ nhớ đệm bẩn. Thiết lập 1.0 hoàn toàn có khả năng làm cho các điểm kiểm tra không hoàn tất được đúng thời gian, điều có thể làm cho mất hiệu năng vì sự biến động không mong đợi về số các phân đoạn WAL cần thiết.

Sẽ luôn có ít nhất 1 tệp phân đoạn WAL, và thường sẽ không có nhiều tệp hơn so với cao hơn wal_keep_segments hoặc (2 + checkpoint_completion_target) * checkpoint_segments + 1. Từng tệp phân đoạn thường là 16MB (dù kích cỡ này có thể được tùy biến khi xây dựng máy chủ). Bạn có thể sử dụng điều này để ước tính các yêu cầu không gian cho WAL. Thường thì, khi các tệp phân đoạn lưu ký cũ không còn cần thiết nữa, thì chúng được tái chế (được đổi tên để trở thành các phân đoạn tiếp sau theo tuần tự được đánh số). Nếu, vì một đỉnh điểm thời gian ngắn tốc độ đầu ra lưu ký, sẽ có nhiều hơn so với 3 * checkpoint_segments + 1 các tệp phân đoạn, thì các tệp phân đoạn không cần thiết sẽ bị xóa thay vì được tái chế cho tới khi hệ thống quay lại theo giới hạn này.

Trong chế độ phục hồi hoặc dự phòng lưu trữ, máy chủ định kỳ thực hiện các điểm khởi động lại mà chúng sẽ tương tự như các điểm kiểm tra trong hoạt động thông thường: máy chủ ép tất cả tình trạng của nó tới đĩa, cập nhật tệp pg_control để chỉ ra rằng dữ liệu WAL được xử lý rồi không cần phải được quét một lần nữa, và sau đó tái chế bất kỳ tệp phân đoạn lưu ký cũ nào trong thư mục pg_xlog. Một điểm khởi động lại được làm bật dậy nếu ít nhất một bản ghi điểm kiểm tra từng được chơi lại kể từ điểm khởi động lại mới nhất và ít nhất một bản ghi điểm kiểm tra từng được chơi lại. Các điểm khởi động lại không thể được thực hiện thường xuyên hơn so với các điểm kiểm tra trong máy chủ chính vì các điểm khởi động lại chỉ có thể được thực hiện ở các bản ghi điểm kiểm tra.

Có 2 hàm WAL nội bộ được sử dụng phổ biến: LogInsert và LogFlush. LogInsert được sử dụng để đặt một bản ghi mới vào các bộ nhớ đệm WAL trong bộ nhớ được chia sẻ. Nếu không có chỗ cho bản ghi mới, thì LogInsert sẽ phải ghi (chuyển tới bộ nhớ tạm của nhân) một ít bộ nhớ đệm WAL điền được. Điều này là không mong muốn vì LogInsert được sử dụng trong sửa đổi mức thấp của từng cơ sở dữ liệu (ví dụ, chèn hàng) ở thời điểm khi một khóa độc quyền được giữ trong các trang dữ liệu bị ảnh hưởng, nên hoạt động đó cần phải là càng nhanh càng tốt. Điều tồi tệ là, việc ghi các bộ nhớ đệm WAL cũng có thể ép tạo ra một phân đoạn lưu ký mới, nó mất thậm chí nhiều thời gian hơn. Thông thường, các bộ nhớ đệm WAL sẽ được ghi và được xả bằng một yêu cầu của LogFlush, nó được làm, cho hầu hết các phần, trong thời gian thực hiện giao dịch để đảm bảo rằng các bản ghi giao dịch được xả tới kho lưu trữ vĩnh viễn. Trong các hệ thống với đầu ra lưu ký cao, các yêu cầu của LogFlush có thể không xảy ra đủ thường xuyên để ngăn chặn Loginsert khỏi việc phải thực hiện các cuộc ghi. Trong các hệ thống như vậy người ta sẽ tăng số các bộ nhớ đệm WAL bằng việc sửa tham số cấu hình wal_buffers. Số các bộ nhớ đệm WAL mặc định là 8. Việc tăng giá trị này sẽ làm

tăng tương ứng sử dụng bộ nhớ được chia sẻ. Khi full_page_writes được thiết lập và hệ thống rất bận, việc thiết lập giá trị này cao hơn sẽ giúp trả lời trơn tru thời gian trong giai đoạn đó ngay lập tức sau từng điểm kiểm tra.

Tham số commit_delay xác định bao nhiêu mili giây tiến trình máy chủ sẽ ngủ sau việc ghi một bản ghi thực hiện tới lưu ký với Loginsert nhưng trước khi thực hiện một LogFlush. Sự trễ này cho phép các tiến trình máy chủ khác thêm các bản ghi thực hiện của chúng tới lưu ký đó sao cho để có tất cả chúng được xả với một sự đồng bộ lưu ký duy nhất. Sự ngủ sẽ không xảy ra nếu fsync không được kích hoạt, hoặc nếu ít hơn so với commit_siblings thì các phiên khác hiện trong các giao dịch tích cực; điều này tránh việc ngủ khi có lẽ không rằng bất kỳ phiên làm việc nào khác sẽ thực hiện sớm. Lưu ý rằng trong hầu hết các nền tảng, giải pháp một yêu cầu ngủ là 10 mili giây, sao cho bất kỳ thiết lập commit_delay khác zero nào giữa 1 và 10.000 mili giây cũng có thể có hiệu ứng y hệt. Các giá trị tốt cho các tham số đó còn chưa rõ ràng; thí nghiệm được khuyến khích ở đây.

Tham số wal_sync_method xác định cách mà PostgreSQL sẽ yêu cầu nhân ép các bản cập nhật WAL ra đĩa. Tất cả các lựa chọn sẽ là y hệt về độ tin cậy, với ngoại lệ của fsync_writethrough, nó có thể đôi lúc ép một sự xả của bộ nhớ tạm của đĩa thậm chí khi các lựa chọn khác không làm thế. Tuy nhiên, điều đó hoàn toàn là đặc thù nền tảng cái nào sẽ là nhanh nhất; bạn có thể kiểm thử lựa chọn đẩy nhanh việc sử dụng tiện ích src/tools/fsync trong cây nguồn PostgreSQL. Lưu ý rằng tham số này không phù hợp nếu fsync đã bị tắt.

Việc kích hoạt tham số cấu hình wal_debug (miễn là PostgreSQL đã được biên dịch có sự hỗ trợ cho nó) sẽ làm cho từng cuộc gọi WAL của từng LogInsert và LogFlush được lưu ký tới lưu ký máy chủ. Lựa chọn này có thể được thay thế bằng một cơ chế chung trong tương lai.

29.5. Nội bộ WAL

WAL tự động được kích hoạt; không hành động nào được yêu cầu từ quản trị viên ngoại trừ việc đảm bảo rằng các yêu cầu không gian đĩa cho các lưu ký WAL được đáp ứng, và bất kỳ việc tinh chỉnh cần thiết nào cũng được thực hiện (xem Phần 29.4).

Các lưu ký WAL được lưu trữ trong thư mục pg_xlog dưới thư mục dữ liệu, như một tập hợp các tệp phân đoạn, kích cỡ mỗi tệp thường là 16MB (nhưng kích cỡ đó có thể bị thay đổi bằng việc sửa lựa chọn cấu hình --with-wal-segsize khi xây dựng máy chủ). Từng phân đoạn được chia thành các trang, thường là mỗi trang 8KB (kích cỡ này có thể bị thay đổi qua lựa chọn cấu hình --with-wal-blocksize). Đầu đề các bản ghi lưu ký được mô tả trong access/xlog.h; nội dung bản ghi là phụ thuộc vào dạng sự kiện đang được lưu ký. Các tệp phân đoạn được các số gia tăng đưa ra như các tên, bắt đầu từ 000000010000000000000000. Các số không phủ hết, nhưng nó sẽ mất rất, rất lâu để cạn hết kho số có sẵn đó.

Là ưu điểm nếu lưu ký nằm trong một đĩa khác với các tệp cơ sở dữ liệu chính. Điều này có thể đạt được bằng việc chuyển thư mục pg_xlog tới vị trí khác (trong khi máy chủ đã được tắt, tất nhiên) và việc tạo một liên kết biểu tượng từ vị trí gốc ban đầu trong thư mục dữ liệu chính tới vị trí mới đó.

Mục tiêu của WAL là để đảm bảo rằng lưu ký được ghi trước khi các bản ghi cơ sở dữ liệu được tùy biến, nhưng điều này có thể bị phá vì các ổ đĩa và còn chưa được lưu trữ nó lên đĩa đó. Mất điện

trong tình huống như vậy có thể dẫn tới hỏng dữ liệu không thể phục hồi lại được. Những quản trị viên nên cố gắng đảm bảo rằng các đĩa đang giữ các tệp lưu ký WAL của PostgreSQL không làm các báo cáo sai như vậy. (Xem Phần 29.1).

Sau một điểm kiểm tra được thực hiện và lưu ký được xả, vị trí của điểm kiểm tra được lưu trong tệp pg_control. Vì thế, ở đầu của sự phục hồi, máy chủ trước hết đọc pg_control và sau đó bản ghi điểm kiểm tra; sau đó nó thực hiện hoạt động REDO bằng việc quét tiến từ vị trí lưu ký được chỉ ra trong bản ghi điểm kiểm tra. Vì toàn bộ nội dung các trang dữ liệu được lưu trong lưu ký trong sửa đổi trang lần đầu sau một điểm kiểm tra (giả thiết full_page_writes không được kích hoạt), tất cả các trang bị thay đổi kể từ điểm kiểm tra đó sẽ được phục hồi về một tình trạng ổn định.

Để làm việc với trường hợp nơi mà pg_control bị hỏng, chúng ta nên hỗ trợ khả năng quét các phân đoạn lưu ký đang tồn tại theo trật tự ngược - mới nhất tới cũ nhất - để tìm điểm kiểm tra mới nhất. Điều này còn chưa được triển khai. pg_control là đủ nhỏ (nhỏ hơn một trang đĩa) rằng nó không tuân theo các vấn đề ghi một phần, và như đối với việc ghi này từng không có các báo cáo các hỏng hóc cơ sở dữ liệu chỉ vì không có khả năng để đọc bản thân pg_control. Vì thế trong khi về mặt lý thuyết đó là một điểm yếu, thì pg_control không bị coi là một vấn đề trong thực tiễn.

Chương 30. Kiểm thử hồi quy

Các kiểm thử hồi quy là một tập hợp tổng hợp các kiểm thử để triển khai SQL trong PostgreSQL. Chúng kiểm thử các hoạt động SQL tiêu chuẩn cũng như các khả năng mở rộng của PostgreSQL.

30.1. Chạy kiểm thử

Các kiểm thử hồi quy có thể được chạy đối với một máy chủ được cài đặt và chạy rồi, hoặc sử dụng một cài đặt tạm thời trong cây xây dựng. Hơn nữa, có một chế độ "song song" và "tuần tự" cho việc chạy các kiểm thử. Phương pháp tuần tự chạy từng script kiểm thử một mình, trong khi phương pháp song song khởi tạo nhiều tiến trình máy chủ để chạy các nhóm kiểm thử song song. Việc kiểm thử song song đưa ra sự tin cậy mà giao tiếp giữa các tiến trình và việc khóa đang làm việc đúng.

Để chạy các kiểm thử hồi quy song song sau khi xây dựng nhưng trước khi cài đặt, hãy gõ:

`gmake check` vào thư mục mức đỉnh. (Hoặc có thể thay đổi về `src/test/regress` và chạy lệnh ở đó).

Điều này trước hết sẽ xây dựng vài tập bổ sung thêm, như các hàm trigger do người sử dụng định nghĩa, và sau đó chạy script trình điều khiển kiểm thử. Ở cuối bạn sẽ thấy thứ gì đó giống như:

```
=======================
All 115 tests passed.
=======================
```

hoặc nếu không thì một lưu ý về các kiểm thử nào bị hỏng. Xem Phần 30.2 bên dưới trước khi giả thiết là một "sự hỏng" đại diện cho một vấn đề nghiêm trọng.

Vì phương pháp kiểm thử này chạy một máy chủ tạm thời, nên nó sẽ không làm việc khi bạn là người sử dụng gốc root (vì máy chủ sẽ không khởi động như là root). Nếu bạn đã xây dựng rồi như là root, thì bạn không phải khởi động tất cả mọi thứ đó. Thay vào đó, hãy thực hiện kiểm thử hồi quy thư mục có khả năng ghi được bằng người sử dụng khác, đăng nhập như là người sử dụng đó, và khởi động lại các kiểm thử đó. Ví dụ:

```
root# chmod -R a+w src/test/regress
root# su - joeuser
joeuser$ cd top-level build directory
joeuser$ gmake check
```

("Rủi ra an toàn" có khả năng duy nhất ở đây là việc những người sử dụng khác có thể có khả năng sửa các kết quả kiểm thử hồi quy sau lưng bạn. Hãy sử dụng ý nghĩa chung khi quản lý các quyền của người sử dụng).

Như một sự lựa chọn, hãy chạy các kiểm thử sau cài đặt.

Nếu bạn đã thiết lập cấu hình cho PostgreSQL để cài đặt ở một vị trí nơi mà một cài đặt PostgreSQL cũ hơn đã tồn tại rồi, và bạn thực hiện `gmake` trước việc cài đặt phiên bản mới đó, thì bạn có lẽ thấy rằng các kiểm thử hỏng vì các chương trình mới cố sử dụng các thư viện được chia sẻ được cài đặt rồi. (Thường thì các triệu chứng là kêu ca về các biểu tượng không được xác định). Nếu bạn muốn chạy các kiểm thử trước khi ghi đè cài đặt cũ, thì bạn sẽ cần xây dựng với cấu hình `--disable-rpath`. Không được khuyến cáo rằng bạn sử dụng lựa chọn này cho cài đặt cuối cùng.

Kiểm thử hồi quy song song bắt đầu hoàn toàn một ít tiến trình dưới ID người sử dụng của bạn.

Hiện hành, sự đồng thời tối đa là 20 script kiểm thử cùng song hành, có nghĩa là 40 tiến trình: có một tiến trình máy chủ và một tiến trình psql cho từng script kiểm thử. Vì thế nếu hệ thống của bạn ép một giới hạn theo từng người sử dụng về số lượng các tiến trình, hãy chắc chắn giới hạn này ít nhất là 50 hoặc khoảng đó, nếu không bạn có thể gặp các hỏng hóc dường như ngẫu nhiên trong kiểm thử song song. Nếu bạn không ở trong vị trí phải nâng giới hạn đó, thì bạn có thể cắt giảm mức độ song song bằng việc thiết lập tham số MAX_CONNECTIONS. Ví dụ:

```
gmake MAX_CONNECTIONS=10 check
```

chạy không nhiều hơn 10 kiểm thử đồng thời.

Để chạy các kiểm thử sau cài đặt (xem Chương 15), hãy khởi tạo một vùng dữ liệu và khởi động máy chủ, như được giải thích ở Chương 17, rồi gõ:

```
gmake installcheck
```

hoặc cho một kiểm thử song song:

```
gmake installcheck-parallel
```

Các kiểm thử sẽ mong chờ để liên lạc với máy chủ ở host cục bộ (local host) và số cổng mặc định, trừ phi nếu không sẽ được định hướng bằng các biến môi trường PGHOST và PGPORT.

Phân phối nguồn cũng gồm các kiểm thử hồi quy cho các ngôn ngữ thủ tục lựa chọn và cho một vài module contrib. Hiện hành, các kiểm thử đó có thể chỉ được sử dụng đối với một máy chủ được cài đặt rồi. Để chạy các kiểm thử cho tất cả các ngôn ngữ thủ tục mà từng được xây dựng và được cài đặt, hãy thay đổi tới thư mục src/pl của cây xây dựng và gõ:

```
gmake installcheck
```

Bạn cũng có thể làm điều này trong bất kỳ thư mục con nào của src/pl để chạy các kiểm thử chỉ cho một ngôn ngữ thủ tục. Để chạy các kiểm thử cho tất cả các module contrib mà có chúng, hãy thay đổi về thư mục contrib của cây xây dựng và gõ:

```
gmake installcheck
```

Các module contrib phải đã được xây dựng và được cài đặt trước tiên. Bạn cũng có thể làm điều này trong một thư mục con của contrib để chạy các kiểm thử chỉ cho một module.

Phân phối nguồn cũng gồm các kiểm thử hồi quy hành vi tĩnh của Dự phòng Nóng. Các kiểm thử đó đòi hỏi một máy chủ chính đang chạy và một máy chủ dự phòng đang chạy mà đang chấp nhận những thay đổi WAL mới từ máy chủ chính bằng việc sử dụng hoặc việc xuất xưởng lưu ký dựa vào tệp hoặc nhân bản dòng. Các máy chủ đó không được tự động tạo ra cho bạn, cũng không là thiết lập được ghi tài liệu ở đây. Xin hãy kiểm tra các phần khác nhau của tài liệu dành riêng rồi cho các lệnh được yêu cầu và các vấn đề có liên quan.

Trước hết hãy tạo một cơ sở dữ liệu gọi là “regresion” (hồi quy) trong máy chủ chính.

```
psql -h primary -c "CREATE DATABASE regression"
```

Sau đó, chạy một script chuẩn bị trong máy chủ chính trong cơ sở dữ liệu hồi quy:

src/test/regress/sql/hs_primary_setup.sql, và cho phép những thay đổi được nhân giống tới máy chủ dự phòng, ví dụ

```
psql -h primary -f src/test/regress/sql/hs_primary_setup.sql regression
```

Bây giờ hãy khẳng định rằng kết nối mặc định cho trình kiểm thử là máy chủ dự phòng dưới sự kiểm thử và sau đó chạy đích standbycheck từ thư mục hồi quy:

```
cd src/test/regress
gmake standbycheck
```

Một vài hành vi cực đoan cũng có thể được sinh ra trong máy chủ chính bằng việc sử dụng script: src/test/regress/sql/hs_primary_extremes.sql để cho pheps hành vi đó của máy chủ dự phòng được kiểm thử.

Việc kiểm thử tự động hóa được bổ sung thêm có thể sẵn sàng trong các phiên bản sau này.

30.2. Đánh giá kiểm thử

Vài cài đặt PostgreSQL có chức năng đầy đủ và được cài đặt đúng có thể "thiếu" một vài kiểm thử hồi quy đó vì các chế tác đặc thù nền tảng như đại diện các dấu chấm động và ngôn từ các thông điệp khác nhau. Các kiểm thử đó hiện được đánh giá bằng việc sử dụng một so sánh đơn giản diff đối với các đầu ra được sinh ra trong một hệ thống tham chiếu, nên các kết quả là nhạy cảm đối với những khác biệt của các hệ thống nhỏ. Khi một kiểm thử được nêu như là "thất bại", hãy luôn xem xét những khác biệt giữa các kết quả được kỳ vọng và thực tế; bạn có thể thấy rằng những khác biệt đó không đáng kể. Dù vậy, chúng ta sẽ cố duy trì các tệp tham chiếu chính xác khắp tất cả các nền tảng được hỗ trợ, sao cho nó có thể được kỳ vọng rằng tất cả các kiểm thử đều vượt qua được.

Các đầu ra thực tế của các kiểm thử hồi quy nằm trong các tệp trong thư mục src/test/regress/results. Script kiểm thử sử dụng diff để so sánh từng tệp đầu ra đối với các đầu ra tham chiếu được lưu trữ trong thư mục src/test/regress/expected. (hoặc bạn có thể tự chạy diff, nếu thích hơn).

Nếu vì một vài lý do một nền tảng cụ thể sinh ra một "sự thất bại" đối với một kiểm thử được đưa ra, nhưng sự điều tra đầu ra thuyết phục bạn rằng kết quả đó là hợp lệ, thì bạn có thể thêm một tệp so sánh mới để làm im báo cáo thất bại đó khi chạy kiểm thử trong tương lai. Xem Phần 30.3 để có thêm chi tiết.

30.2.1. Khác biệt thông điệp lỗi

Một vài kiểm thử hồi quy có liên quan tới các giá trị đầu vào hợp lệ có chủ ý. Các thông điệp lỗi có thể tới hoặc từ mã PostgreSQL hoặc từ các thủ tục hệ thống nền tảng máy chủ host. Trong trường hợp sau, các thông điệp có thể khác nhau giữa các nền tảng, nhưng sẽ phản ánh thông tin đơn giản. Những khác biệt đó trong các thông điệp sẽ gây ra một kiểm thử hồi quy "bị hỏng" mà có thể được kiểm tra hợp lệ bằng điều tra.

30.2.2. Khác biệt bản địa

Nếu bạn chạy các kiểm thử đối với một máy chủ từng được khởi tạo bằng một bản địa có trật tự đối chiếu được với C, thì có thể có sự khác biệt vì trật tự sắp xếp và những sai hỏng tiếp sau. Bộ kiểm thử hồi quy được thiết lập để điều khiển vấn đề này bằng việc cung cấp các tệp kết quả xen kẽ nhau mà cùng nhau được biết để điều khiển một số lượng lớn các bản địa.

Để chạy các kiểm thử theo một bản địa khác khi sử dụng phương pháp cài đặt tạm thời, hãy truyền các biến môi trường có liên quan tới bản địa đúng phù hợp trong dòng lệnh make, ví dụ:

```
gmake check LANG=de_DE.utf8
```

(Trình điều khiển kiểm thử hồi quy bỏ thiết lập LC_ALL, nên nó không làm việc để chọn bản địa bằng việc sử dụng biến đó). Để không sử dụng bản địa, hoặc bỏ thiết lập tất cả các biến môi trường có liên quan tới bản địa (hoặc thiết lập chúng về C) hoặc sử dụng lời gọi đặc biệt sau:

```
gmake check NO_LOCALE=1
```

Khi chạy các kiểm thử đối với một cài đặt đang tồn tại, thiết lập bản địa được cài đặt đang tồn tại xác định. Để thay đổi nó, hãy khởi tạo bó cơ sở dữ liệu bằng một bản địa khác bằng việc truyền các lựa chọn đúng phù hợp tới initdb.

Nói chung, tuy được khuyến cáo thử chạy các kiểm thử hồi quy trong thiết lập bản địa như mong muốn cho sử dụng sản xuất, vì điều này sẽ trải nghiệm các phần mã có liên quan tới việc giải mã và bản địa mà thực sự sẽ được sử dụng trong sản xuất. Phụ thuộc vào môi trường hệ điều hành, bạn có thể có sai lầm, nhưng sau đó bạn ít nhất sẽ biết các hành vi đặc thù bản địa nào để kỳ vọng khi chạy các ứng dụng thực tế.

30.2.3. Khác biệt ngày tháng và thời gian

Hầu hết các kết quả ngày tháng và thời gian là phụ thuộc vào môi trường vùng thời gian. Các tệp tham chiếu được sinh ra cho vùng thời gian PST8PDT (Berkeley, California), và hình như sẽ sai nếu các kiểm thử không chạy với thiết lập vùng thời gian đó. Trình điều khiển hồi quy thiết lập biến môi trường PGTZ về PST8PDT, điều thường đảm bảo cho các kết quả đúng phù hợp.

30.2.4. Khác biệt dấu chấm động

Một vài kiểm thử có liên quan tới việc tính toán số dấu chấm động 64 bit (double precision) từ các cột của bảng. Các khác biệt trong các kết quả có liên quan tới các hàm toán học của các cột double precision đã từng được quan sát tới. Các kiểm thử float8 và geometry là đặc biệt có thiên hướng đối với các khác biệt nhỏ xuyên khắp các nền tảng, hoặc thậm chí với thiết lập tối ưu của các trình biên dịch khác nhau. So sánh bằng mắt thường là cần thiết để xác định tầm quan trọng thực tế của các khác biệt đó, chúng thường là 10 chỗ về bên phải của dấu thập phân.

Một vài hệ thống hiển thị trừ zero như là -0, trong khi một số khác chỉ là 0.

Một vài hệ thống ra dấu hiệu các lỗi từ pow() và exp() trực tiếp từ cơ chế được mã PostgreSQL hiện hành kỳ vọng.

30.2.5. Khác biệt trật tự hàng

Bạn có thể thấy những khác biệt theo đó các hàng y hệt là đầu ra theo một trật tự khác với những gì xuất hiện trong tệp được kỳ vọng. Trong hầu hết các trường hợp thì điều này không, nói nghiêm túc, là một lỗi. Hầu hết các script kiểm thử hồi quy không rởm như để sử dụng một ORDER BY cho từng lệnh SELECT, và vì thế trật tự hàng kết quả của chúng không được xác định tốt theo đặc tả SQL. Trong thực tế, vì chúng ta đang xem xét các truy vấn y hệt đang được thực thi trong cùng y hệt các dữ liệu của y hệt phần mềm, chúng ta thường có trật tự kết quả y hệt trong tất cả các nền tảng, nên sự thiếu ORDER BY không là một vấn đề. Tuy nhiên, một vài truy vấn không thể hiện các khác biệt

trật tự xuyên nền tảng. Khi kiểm thử đối với một máy chủ được cài đặt rồi, các khác biệt trật tự cũng có thể có lý do vì các thiết lập bản địa khác C hoặc các thiết lập tham số khác mặc định, như với các giá trị tùy biến của work_mem hoặc các tham số chi phí của trình lên kế hoạch.

Vì thế, nếu bạn thấy một sự khác biệt về trật tự, thì điều đó không là thứ gì đó phải lo lắng, trừ phi truy vấn đó có một ORDER BY mà kết quả của bạn đang vi phạm. Tuy nhiên, hãy báo cáo nó bằng mọi cách, sao cho chúng ta có thể thêm ORDER BY vào truy vấn đặc biệt đó để loại bỏ "sự thất bại" giả tạo đó trong các phiên bản trong tương lai.

Bạn có thể tự hỏi vì sao chúng ta không lệnh cho tất cả các truy vấn kiểm thử hồi quy rõ ràng để loại bỏ vấn đề này ngay và mãi mãi. Lý do là điều đó có thể làm cho các kiểm thử hồi quy ít hữu dụng hơn, không hơn, vì chúng có xu hướng thực thi các dạng kế hoạch truy vấn mà có sinh ra các kết quả có trật tự đối với sự loại bỏ các dạng không có trật tự.

30.2.6. Độ sâu không đủ của kho

Nếu các kết quả kiểm thử errors trong một sự hỏng máy chủ ở lệnh select infinite_recurse(), thì có nghĩa là giới hạn của nền tảng đó về kích cỡ kho các tiến trình là nhỏ hơn so với tham số max_stack_depth chỉ ra. Điều này có thể được sửa bằng việc chạy máy chủ dưới một giới hạn kích cỡ kho cao hơn (4MB được khuyến cáo với giá trị mặc định của max_stack_depth). Nếu bạn không có khả năng để làm điều đó, thì một lựa chọn là hãy làm giảm giá trị của max_stack_depth.

30.2.7. Kiểm thử "ngẫu nhiên"

Script kiểm thử ngẫu nhiên random có ý định để tạo ra các kết quả ngẫu nhiên. Trong các trường hợp hiếm hoi, điều này làm cho kiểm thử hồi quy ngẫu nhiên bị hỏng. Việc gõ:

```
diff results/random.out expected/random.out
```

sẽ chỉ sinh ra một hoặc một ít dòng khác nhau. Bạn không cần lo lắng trừ phi kiểm thử ngẫu nhiên đó hỏng lặp đi lặp lại.

30.3. Các tệp so sánh khác nhau

Vì một vài kiểm thử vốn dĩ sinh ra các kết quả phụ thuộc vào môi trường, nên chúng tôi đã đưa ra các cách thức để chỉ định luôn phiên các tệp kết quả "được kỳ vọng". Mỗi kiểm thử hồi quy có thể có vài tệp so sánh chỉ ra các kết quả có thể trong các nền tảng khác nhau. Có 2 cơ chế độc lập cho việc xác định tệp so sánh nào được sử dụng cho từng kiểm thử.

Cơ chế đầu tiên cho phép các tệp so sánh sẽ được chọn cho các nền tảng đặc biệt. Có một tệp ánh xạ, src/test/regress/resultmap, nó xác định tệp so sánh nào để sử dụng cho từng nền tảng. Để loại bỏ "thất bại" giả của các kiểm thử đối với một nền tảng cụ thể, trước hết bạn hãy chọn hoặc tạo một tệp kết quả, và sau đó thêm một dòng vào tệp resultmap.

Mỗi dòng trong tệp ánh xạ đều ở dạng

```
testname:output:platformpattern=comparisonfilename
```

Tên kiểm thử chỉ là tên của module kiểm thử hồi quy đặc biệt đó. Giá trị đầu ra chỉ tệp đầu ra nào phải kiểm tra. Đối với các kiểm thử hồi quy tiêu chuẩn, điều này luôn là out. Giá trị đó tương ứng với phần mở rộng tệp của tệp đầu ra. Mẫu nền tảng là một mẫu theo dạng cộng cụ Unix expr (đó là,

một biểu thức thông thường với một dấu ẩn ^ ở đầu). Nó khớp với tên nền tảng như được config.guess in ra. Tên tệp so sánh là tên cơ bản của tệp so sánh kết quả thay thế.

Ví dụ: một vài hệ thống dịch các giá trị dấu chấm động nhỏ như là zero, thay vì việc nêu một lỗi ngầm bên dưới. Điều này là lý do có một ít khác biệt trong kiểm thử hồi quy float8. Vì thế, chúng tôi đưa ra một tệp so sánh khác, float8-small-is-zero.out, nó bao gồm các kết quả sẽ được kỳ vọng trong các hệ thống đó. Để làm im thông điệp "hỏng" giả trong các nền tảng BSD, resultmap gồm:

```
float8:out:i.86-.*-openbsd=float8-small-is-zero.out
```

nó sẽ làm bật dậy trong bất kỳ máy nào nơi mà đầu ra của config.guess khớp với i.86-.*-openbsd. Các dòng khác trong resultmap chọn tệp so sánh khác cho các nền tảng khác nơi mà nó là phù hợp.

Cơ chế lựa chọn thứ 2 cho các tệp so sánh khác là tự động hơn nhiều: nó đơn giản sử dụng "sự trùng khớp tốt nhất" giữa vài tệp so sánh được cung cấp. Script trình điều khiển kiểm thử hồi quy xem xét cả tệp so sánh tiêu chuẩn cho một kiểm thử, testname.out, và các tệp khác có tên là testname_digit.out (trong đó digit là bất kỳ số đơn nhất nào từ 0 tới 9). Nếu bất kỳ tệp nào như vậy trùng khớp chính xác, thì kiểm thử được xem là đạt; nếu không, kiểm thử mà sinh ra diff ngắn nhất sẽ được sử dụng để tạo ra báo cáo lỗi. (nếu resultmap bao gồm một khoản đầu vào cho kiểm thử đặc biệt đó, thì testname cơ bản là tên thay thế được đưa ra trong resultmap).

Ví dụ, đối với kiểm thử char, tệp so sánh char.out gồm các kết quả được kỳ vọng trong các bản địa C và POSIX, trong khi tệp char_1.out gồm các kết quả được sắp xếp như chúng xuất hiện trong nhiều bản địa khác.

Cơ chế trùng khớp tốt nhất từng được phát minh để vượt qua được các kết quả phụ thuộc vào bản địa, nhưng nó có thể được sử dụng trong bất kỳ tình huống nào nơi mà các kết quả kiểm thử không thể đoán trước được dễ dàng từ chỉ tên nền tảng đó. Một hạn chế của cơ chế này là trình điều khiển kiểm thử không thể nói phương án nào thực sự là "đúng" đối với môi trường hiện hành đó; nó sẽ chỉ lấy phương án mà dường như là làm việc tốt nhất. Vì thế là an toàn nhất để sử dụng cơ chế này chỉ cho các kết quả phương án mà bạn đang muốn xem xét hợp lệ ngang bằng trong mọi ngữ cảnh.

30.4. Kiểm tra bao phủ kiểm thử

Mã nguồn PostgreSQL có thể được biên dịch với sự dàn phối kiểm thử bao phủ, sao cho nó trở thành có khả năng để xem xét các phần nào của mã được bao trùm đối với các kiểm thử hồi quy hoặc bất kỳ bộ kiểm thử nào khác được chạy với mã đó. Điều này hiện được hỗ trợ khi biên dịch với GCC và đòi hỏi các chương trình gcov và lcov. Một dòng công việc điển hình có thể là như sau:

```
./configure --enable-coverage ... OTHER OPTIONS ...
gmake
gmake check # or other test suite
gmake coverage-html
```

Sau đó hãy chỉ ra trình duyệt HTML của bạn tới coverage/index.html. Các lệnh gmake cũng làm việc trong các thư mục con.

Để thiết lập lại các đo đếm thực thi giữa các lần chạy kiểm thử, hãy chạy:

```
gmake coverage-clean
```

IV. Giao diện máy trạm

Phần này mô tả các giao diện lập trình máy trạm được phân phối với PostgreSQL. Mỗi trong số các chương đó có thể được đọc độc lập riêng rẽ. Lưu ý rằng có nhiều giao diện lập trình khác cho các chương trình máy trạm được phân phối riêng rẽ và có tài liệu liệu riêng của chúng (Phụ lục G liệt kê vài tài liệu phổ biến nhất). Các độc giả của phần này nên làm quen với việc sử dụng các lệnh SQL để điều khiển và truy vấn cơ sở dữ liệu (xem Phần II) và tất nhiên với ngôn ngữ lập trình mà giao diện đó sử dụng.

Chương 31. libpq - Thư viện C

libpq là giao diện lập trình ứng dụng C cho PostgreSQL. libpq là một tập hợp các hàm thư viện cho phép các chương trình máy trạm truyền các truy vấn tới máy chủ phụ trợ (backend) PostgreSQL và để nhận các kết quả của các truy vấn đó.

libpq cũng là động cơ nằm bên dưới cho vài giao diện dứng dụng PostgreSQL khác, bao gồm các giao diện được viết cho C++, Perl, Python, Tcl và ECPG. Vì thế vài khía cạnh hành vi của libpq sẽ là quan trọng cho bạn nếu bạn sử dụng một trong các gói đó. Đặc biệt, Phần 31.13, Phần 31.14 và Phần 31.17 mô tả hành vi nhìn thấy được đối với người sử dụng của bất kỳ ứng dụng nào sử dụng libpq.

Một vài chương trình ngắn được đưa vào ở cuối của chương này (Phần 31.20) để chỉ ra cách viết các chương trình có sử dụng libpq. Cũng có vài ví dụ phức tạp về các ứng dụng libpq trong thư mục src/test/examples trong phân phối mã nguồn đó.

Các chương trình máy trạm có sử dụng libpq phải bao gồm tệp đầu đề libpq-fe.h và phải liên kết với thư viện libpq.

31.1. Các hàm kiểm tra kết nối cơ sở dữ liệu

Các hàm sau đây làm việc với việc tạo ra một kết nối tới một máy chủ phụ trợ của PostgreSQL. Một chương trình ứng dụng có thể có vài kết nối phụ trợ được mở tại một thời điểm. (Một lý do để làm thế là để truy cập nhiều hơn một cơ sở dữ liệu). Từng kết nối được đại diện bằng một đối tượng PGconn, nó có được từ hàm PQconnectdb, PQconnectdbParams, hoặc PQsetdbLogin. Lưu ý rằng các hàm đó sẽ luôn trả về một con trỏ đối tượng khác null, trừ phi có thể thậm chí có quá ít bộ nhớ để phân bổ cho đối tượng PGconn. Hàm PQstatus sẽ được gọi để kiểm tra liệu một kết nối có được làm thành công trước khi các truy vấn được gửi qua đối tượng kết nối đó hay không.

Cảnh báo

Trong Unix, việc rẽ nhánh một tiến trình với các kết nối mở libpq có thể dẫn tới các kết quả không thể đoán trước vì các tiến trình cha và con chia sẻ cùng y hệt các socket và các tài nguyên hệ điều hành. Vì lý do này, sự sử dụng như vậy không được khuyến cáo, dù việc thực hiện một exec từ tiến trình con để tải một sự thực thi mới là an toàn.

Lưu ý: Trong Windows, có một cách để cải thiện hiệu năng nếu một kết nối cơ sở dữ liệu được khởi tạo và tắt lặp đi lặp lại. Trong nội bộ, libpq gọi WSAStartup() và WSACleanup() khởi động và tắt kết nối, một cách tương ứng. WSAStartup() tăng dần sự đếm tham chiếu thư viện nội bộ Windows, điều được WSACleanup() làm tăng dần. Khi sự đếm tham chiếu chỉ là một, thì việc gọi WSACleanup() sẽ giải phóng tất cả các tài nguyên và tất cả các DLL còn chưa được tải lên. Đây là một hoạt động đắt giá. Để tránh điều này, một ứng dụng có thể gọi bằng tay

WSAStartup() sao cho các tài nguyên sẽ không được giải phóng khi kết nối cơ sở dữ liệu cuối cùng được đóng lại.

PQconnectdbParams

Tạo một kết nối mới tới máy chủ cơ sở dữ liệu.

```
PGconn *PQconnectdbParams(const char **keywords, const char **values, int expand_dbname);
```

Hàm này mở một kết nối cơ sở dữ liệu mới bằng việc sử dụng các tham số được lấy từ 2 mảng có kết thúc là NULL. Mảng đầu, các từ khóa, được xác định như một mảng chuỗi, từng chuỗi là một từ khóa. Mảng thứ 2, các giá trị values, đưa ra giá trị cho từng từ khóa. Không giống như PQsetdbLogin bên dưới, thiết lập tham số đó có thể được mở rộng không có việc thay đổi chữ ký hàm, nên sử dụng hàm này (hoặc các hàm tương tự không có khóa PQconnectStartParams và PQconnectPoll) được ưu tiên cho việc lập trình ứng dụng mới.

Khi expand_dbname là khác zero, thì giá trị từ khóa dbname được phép sẽ được thừa nhận như một chuỗi conninfo. Xem bên dưới để có các chi tiết.

Các mảng được truyền qua có thể là rỗng để sử dụng tất cả các tham số mặc định, hoặc có thể có một hoặc nhiều hơn các thiết lập tham số. Chúng sẽ trùng khớp theo độ dài. Việc xử lý sẽ dừng với yếu tố khác NULL của mảng từ khóa keywords.

Các từ khóa tham số hiện hành được thừa nhận là:

host

Tên của máy chủ host để kết nối tới. Nếu điều này bắt đầu bằng một dấu chéo, thì nó chỉ định một giao tiếp miền Unix hơn là giao tiếp TCP/IP; giá trị đó là tên thư mục trong đó tệp socket được lưu giữ. Hành vi mặc định khi host không được chỉ định là để kết nối tới một socket miền Unix trong /tmp (hoặc thư mục socket bất kỳ từng được chỉ định khi PostgreSQL từng được xây dựng). Trong các máy không có các socket miền Unix, mặc định là để kết nối tới localhost.

hostaddr

Địa chỉ IP bằng số của host để kết nối tới. Điều này sẽ là theo định dạng địa chỉ IPv4 tiêu chuẩn, như 172.28.40.9. Nếu máy của bạn hỗ trợ IPv6, thì bạn cũng có thể sử dụng các địa chỉ đó. Giao tiếp TCP/IP luôn được sử dụng khi một chuỗi khác rỗng được chỉ định cho tham số này.

Việc sử dụng hostaddr thay vì host cho phép ứng dụng tránh được sự tra cứu tên host, điều có thể là quan trọng trong các ứng dụng với các ràng buộc thời gian. Tuy nhiên, một tên host được yêu cầu cho xác thực Kerberos, GSSAPI, hoặc SSPI, cũng như cho sự kiểm tra hợp lệ chứng thực SSL đầy đủ. Các qui tắc sau đây sẽ được sử dụng: Nếu host được chỉ định mà không có hostaddr, thì sự tra cứu tên một host sẽ xảy ra. Nếu hostaddr được chỉ định mà không có host, thì giá trị cho hostaddr đưa ra địa chỉ máy chủ. Sự cố gắng kết nối sẽ hỏng trong bất kỳ trường hợp nào nơi mà một tên host được yêu cầu. Nếu cả hostaddr và host đều được chỉ định, thì giá trị cho hostaddr đưa

ra địa chỉ máy chủ. Giá trị cho host bị bỏ qua trừ phi là cần thiết cho các mục đích xác thực hoặc kiểm tra hợp lệ, trong trường hợp đó nó sẽ được sử dụng như là tên host. Lưu ý rằng xác thực có khả năng hỏng nếu host không là tên của máy ở hostaddr. Lưu ý rằng host hơn là hostaddr được sử dụng để nhận diện kết nối trong ~/.pgpass (xem Phần 31.14).

Không có hoặc một tên host hoặc địa chỉ host, thì libpq sẽ kết nối bằng việc sử dụng một socket miền Unix; hoặc trong các máy không có các socket miền Unix, nó sẽ cố kết nối tới localhost.

port

Số cổng để kết nối tới ở máy chủ host, hoặc mở rộng tên tệp socket cho các kết nối miền Unix.

dbname

Tên cơ sở dữ liệu. Mặc định sẽ là tên như tên người sử dụng.

user

Tên người sử dụng PostgreSQL để kết nối. Mặc định là y hệt như tên người sử dụng hệ điều hành đang chạy ứng dụng đó.

password

Mật khẩu sẽ được sử dụng nếu máy chủ yêu cầu xác thực mật khẩu.

connect_timeout

Chờ đợi tối đa cho kết nối, theo giây (viết như một chuỗi số nguyên thập phân). Zero hoặc không chỉ định có nghĩa là chờ vô định. Không được khuyến cáo để sử dụng một timeout ít hơn 2 giây.

options

Bổ sung thêm các lựa chọn dòng lệnh để gửi tới máy chủ khi đang chạy. Ví dụ, thiết lập điều này tới -c geqo=off sẽ thiết lập giá trị phiên làm việc của tham số geqo về tắt (off). Để có thảo luận chi tiết về các lựa chọn có sẵn, hãy xem Chương 18.

application_name

Chỉ định một giá trị cho tham số cấu hình application_name.

fallback_application_name

Chỉ định một giá trị dự phòng cho tham số cấu hình application_name. Giá trị này sẽ được sử dụng nếu không giá trị nào được đưa ra cho application_name qua một tham số kết nối hoặc biến môi trường PGAPPNAME. Việc chỉ định một tên dự phòng là hữu dụng trong các chương trình tiện ích chung mà muốn thiết lập một tên ứng dụng mặc định nhưng cho phép nó được người sử dụng ghi đè.

keepalives

Kiểm soát liệu keepalive (giữ cho sống) TCP phía máy trạm có được sử dụng hay không. Giá trị mặc định là 1, nghĩa là mở (on), nhưng bạn có thể thay đổi về 0, nghĩa là tắt (off), nếu keepalive không được mong đợi. Tham số này bị bỏ qua đối với các kết nối được làm qua một socket miền Unix.

keepalives_idle

Kiểm soát số giây không hoạt động sau đó TCP sẽ gửi một thông điệp keepalive tới máy chủ đó. Một giá trị zero sử dụng mặc định của hệ thống. Tham số này bị bỏ qua đối với các kết nối được làm qua một socket miền Unix, hoặc nếu keepalive bị vô hiệu hóa. Chỉ được hỗ trợ trong các hệ thống nơi mà lựa chọn socket TCP_KEEPIDLE or TCP_KEEPALIVE là sẵn sàng, và trong Windows; trong các hệ thống khác, nó không có tác dụng.

keepalives_interval

Kiểm soát số giây sau đó một thông điệp keepalive TCP không được máy chủ thừa nhận sẽ được truyền lại. Giá trị zero sử dụng mặc định của hệ thống. Tham số này bị bỏ qua đối với các kết nối được làm qua một socket miền Unix, hoặc nếu keepalive bị vô hiệu hóa. Nó chỉ được hỗ trợ trong các hệ thống nơi mà lựa chọn socket TCP_KEEPCNT là sẵn sàng; trong các hệ thống khác, nó không có tác dụng.

keepalives_count

Kiểm soát số keepalive TCP mà có thể bị mất trước khi kết nối máy trạm tới máy chủ được coi là chết. Giá trị zero sử dụng mặc định của hệ thống. Tham số này bị bỏ qua đối với các kết nối được làm qua một socket miền Unix, hoặc nếu keepalive bị vô hiệu hóa. Nó chỉ được hỗ trợ trong các hệ thống nơi mà lựa chọn socket TCP_KEEPCNT là sẵn sàng; trong các hệ thống khác, nó không có tác dụng.

tty

Bị bỏ qua (trước đó, điều này được chỉ định nơi để gửi đầu ra gỡ lỗi của máy chủ).

sslmode

Lựa chọn này xác định liệu có hay không với ưu tiên nào một kết nối an toàn SSL TCP/IP sẽ bị bỏ qua với máy chủ. Có 6 chế độ:

Bảng 31-1. Các lựa chọn sslmode

Lựa chọn	Mô tả
disable (vô hiệu hóa)	chỉ cố một kết nối khác SSL
allow (cho phép)	trước hết cố một kết nối khác SSL; nếu hỏng, cố một kết nối SSL
prefer (ưu tiên-mặc định)	trước hết cố một kết nối SSL; nếu hỏng, cố một kết nối khác SSL
require (yêu cầu)	chỉ cố một kết nối SSL. Nếu một tệp CA root là có, hãy kiểm tra hợp lệ chứng thực theo cách y hệt như nếu verify-ca đã được chỉ định.
verify-ca	chỉ cố một kết nối SSL, và kiểm tra hợp lệ rằng chứng thực máy chủ được một CA tin cậy phát hành
verify-full	chỉ cố một kết nối SSL, kiểm tra hợp lệ chứng thực máy chủ được CA tin cậy phát hành và tên host của máy chủ đó khớp với chứng thực đó.

Xem Phần 31.17 để có một mô tả chi tiết về cách mà các lựa chọn đó làm việc.

sslmode bị bỏ qua đối với giao tiếp socket miền Unix. Nếu PostgreSQL được biên dịch không có hỗ trợ SSL, có sử dụng các lựa chọn require, verify-ca, hoặc verify-full

thì sẽ gây ra một lỗi, trong khi các lựa chọn allow và prefer sẽ là chấp nhận được nhưng libpq sẽ thực sự không cố có một kết nối SSL.

requiressl

Lựa chọn này bị phản đối có lợi đối với thiết lập sslmode.

Nếu được thiết lập về 1, thì một kết nối SSL tới máy chủ được yêu cầu (điều này tương đương với sslmode require), libpq sẽ thương lượng dạng kết nối với máy chủ (tương đương với sslmode prefer). Lựa chọn này chỉ sẵn sàng nếu PostgreSQL được biên dịch với hỗ trợ SSL.

sslcert

Tham số này chỉ định tên tệp chứng thực SSL máy trạm, thay thế mặc định ~/.postgresql/postgresql.crt. Tham số này bị bỏ qua nếu một kết nối SSL không được tạo ra.

sslkey

Tham số này chỉ định vị trí cho khóa bí mật được sử dụng cho chứng thực máy trạm. Nó có thể hoặc chỉ định một tên tệp sẽ được sử dụng thay vì mặc định ~/.postgresql/postgresql.key, hoặc nó có thể chỉ định một khóa có được từ một "động cơ" bên ngoài (các động cơ là các module tải được OpenSSL). Một đặc tả động cơ bên ngoài sẽ bao gồm một tên động cơ phân cách nhau bằng dấu phẩy và một mã nhận diện khóa đặc thù của động cơ đó. Tham số này bị bỏ qua nếu một kết nối SSL không được tạo ra.

sslrootcert

Tham số này chỉ định tên tệp có (các) chứng thực của cơ quan chứng thực (CA) SSL. Nếu tệp đó tồn tại, chứng thực của máy chủ sẽ được kiểm tra hợp lệ để được một trong số các cơ quan đó ký. Mặc định là ~/.postgresql/root.crt.

sslcrl

Tham số này chỉ định tên tệp của danh sách thu hồi chứng thực SSL - CRL (Certificate Revocation List). Các chứng thực có trong tệp này, nếu tồn tại, sẽ bị từ chối trong khi cố xác thực chứng thực máy chủ. Mặc định là ~/.postgresql/root.crl.

krbsrvname

Tên dịch vụ Kerberos để sử dụng khi xác thực bằng Kerberos 5 hoặc GSSAPI. Điều này phải khớp với tên dịch vụ được chỉ định trong cấu hình máy chủ cho xác thực Kerberos thành công. (Xem Phần 19.3.5 và Phần 19.3.3).

gsslib

Thư viện GSS để sử dụng cho xác thực GSSAPI. Chỉ được sử dụng trong Windows. Hãy thiết lập về gssapi để ép libpq sử dụng thư viện GSSAPI thay vì mặc định SSPI.

service

Tên dịch vụ để sử dụng cho các tham số bổ sung thêm. Nó chỉ định một tên dịch vụ trong pg_service.conf mà giữ các tham số kết nối bổ sung. Điều này cho phép các ứng dụng chỉ định chỉ một tên dịch vụ sao cho các tham số kết nối có thể được duy trì tập

trung. Xem Phần 31.15.

Nếu bất kỳ tham số nào không được chỉ định, thì biến môi trường tương ứng (xem Phần 31.13) được kiểm tra. Nếu biến môi trường cũng không được thiết lập, thì các mặc định được xây dựng sẵn được chỉ định sẵn sẽ được sử dụng.

Nếu expand_dbname là khác zero và dbname có một dấu bằng (=), thì nó được lấy như một chuỗi conninfo chính xác theo cách y hệt như nếu nó từng được truyền tới PQconnectdb (xem bên dưới). Các từ khóa được xử lý trước đó sẽ bị ghi đè bằng các từ khóa trong chuỗi conninfo.

Nói chung các từ khóa sẽ được xử lý từ đầu các mảng đó theo trật tự chỉ số. Hiệu quả của điều này là khi các từ khóa được lặp lại, thì giá trị được xử lý cuối cùng được giữ lại. Vì thế, qua việc đặt cẩn thận từ khóa dbname, có khả năng xác định những gì có thể bị ghi đè bằng một chuỗi conninfo, và những gì không bị.

PQconnectdb

Tạo một kết nối mới tới máy chủ cơ sở dữ liệu.

```
PGconn *PQconnectdb(const char *conninfo);
```

Hàm này mở một kết nối cơ sở dữ liệu bằng việc sử dụng các tham số được lấy từ chuỗi conninfo.

Chuỗi được truyền có thể là rỗng để sử dụng tất cả các tham số mặc định, hoặc nó có thể chứa một hoặc nhiều thiết lập tham số được phân cách nhau bằng dấu trắng. Từng thiết lập tham số là ở dạng keyword = value. Các không gian xung quanh dấu bằng là tùy chọn. Để viết một giá trị rỗng, hoặc một giá trị có chứa các không gian, xung quanh nó với các dấu nháy đơn, như, keyword = 'a value'. Các dấu nháy đơn và các dấu chéo ngược trong giá trị đó phải được thoát bằng một dấu chéo ngược, như, \' and \\.

Các từ khóa tham số hiện được thừa nhận là y hệt như ở trên.

PQsetdbLogin

Hãy làm một kết nối mới tới máy chủ cơ sở dữ liệu.

```
PGconn *PQsetdbLogin(const char *pghost,
        const char *pgport,
        const char *pgoptions,
        const char *pgtty,
        const char *dbName,
        const char *login,
        const char *pwd);
```

Đây là tiền thân của PQconnectdb với một tập hợp cố định các tham số. Nó có chức năng y hệt ngoại trừ là các tham số bị mất sẽ luôn lấy các giá trị mặc định. Hãy viết NULL hoặc một chuỗi rỗng cho bất kỳ một trong số các tham số cố định nào mà sẽ là được mặc định.

Nếu dbName chứa một dấu bằng (=), thì nó được coi như một chuỗi conninfo chính xác theo cách y hệt dường như nó từng được truyền tới PQconnectdb, và các tham số còn lại sau đó được áp dụng như ở trên.

PQsetdb

Hãy làm một kết nối tới máy chủ cơ sở dữ liệu.

```
PGconn *PQsetdb(char *pghost,
        char *pgport,
        char *pgoptions,
        char *pgtty,
        char *dbName);
```

Đây là macro gọi PQsetdbLogin với các con trỏ NULL cho các tham số login và pwd. Nó được đưa ra vì tính tương thích ngược với các chương trình rất cũ.

PQconnectStartParams
PqconnectStart
PQconnectPoll

Tạo một kết nối tới máy chủ cơ sở dữ liệu theo một cách thức không khóa.

```
PGconn *PQconnectStartParams(const char **keywords, const char **values, int expand_dbname);
PGconn *PQconnectStart(const char *conninfo);
PostgresPollingStatusType PQconnectPoll(PGconn *conn);
```

3 hàm đó được sử dụng để mở một kết nối tới một máy chủ cơ sở dữ liệu như máy chủ mà dòng thực thi ứng dụng của bạn không bị khóa ở I/O từ xa khi làm thế. Mấu chốt của tiếp cận này là những chờ đợi I/O để hoàn tất có thể xảy ra theo vòng lặp chính của ứng dụng, thay vì xuống bên trong PQconnectdbParams hoặc PQconnectdb, và vì thế ứng dụng có thể quản lý hoạt động này song song với các hoạt động khác.

Với PQconnectStartParams, kết nối cơ sở dữ liệu được thực hiện bằng việc sử dụng các tham số được lấy từ các mảng keywords và values, và được expand_dbname kiểm soát, như được mô tả ở trên cho PQconnectdbParams.

Với PQconnectStart, kết nối cơ sở dữ liệu được làm bằng việc sử dụng các tham số được lấy từ chuỗi conninfo như được mô tả ở trên cho Pqconnectdb.

Cả PQconnectStartParams, PQconnectStart và PQconnectPoll đều sẽ không khóa, miễn là số các giới hạn được đáp ứng:

- Các tham số hostaddr và host được sử dụng đúng phù hợp để đảm bảo rằng tên các truy vấn tên và tên nghịch đảo không được tạo ra. Xem tài liệu về các tham số đó theo PQconnectdbParams ở trên để có các chi tiết.
- Nếu bạn gọi PQtrace, hãy đảm bảo là đối tượng dòng trong đó bạn theo dõi sẽ không khóa.
- Bạn đảm bảo là socket là trong tình trạng đúng phù hợp trước khi gọi PQconnectPoll, như được mô tả bên dưới.

Lưu ý: việc sử dụng PQconnectStartParams là tương tự như PQconnectStart như nêu ở dưới.

Để bắt đầu một yêu cầu kết nối không khóa, hãy gọi conn = PQconnectStart("connection_info_string"). Nếu conn là null, thì libpq từng không có khả năng để phân bổ một cấu trúc PGconn mới. Nếu không, một con trỏ PGconn hợp lệ được trả về (dù còn chưa đại diện cho một kết nối hợp lệ tới cơ sở dữ liệu). Khi trả về từ PQconnectStart, hãy gọi status = PQstatus(conn). Nếu status bằng CONNECTION_BAD, thì PQconnectStart sẽ hỏng.

Nếu PQconnectStart thành công, giai đoạn tiếp sau sẽ là khảo sát libpq sao cho nó có thể xử lý với sự tuần tự kết nối. Hãy sử dụng PQsocket(conn) để có được trình mô tả socket nằm bên dưới kết nối cơ sở dữ liệu đó. Vòng lặp vì thế: Nếu PQconnectPoll(conn) last vừa trả về PGRES_POLLING_READING, hãy chờ cho tới khi socket đó sẵn sàng để đọc (như được select(), poll() hoặc hàm hệ thống tương tự chỉ ra). Sau đó gọi PQconnectPoll(conn) một lần nữa. Ngược lại, nếu PQconnectPoll(conn) mới trả về PGRES_POLLING_WRITING, hãy chờ cho tới khi socket đó sẵn sàng để ghi, rồi hãy gọi PQconnectPoll(conn) một lần nữa. Nếu bạn còn phải gọi PQconnectPoll, như, chỉ sau lời gọi tới PQconnectStart, hãy hành xử dường như nó mới trả về PGRES_POLLING_WRITING. Hãy tiếp tục vòng lặp này cho tới khi PQconnectPoll(conn) trả về PGRES_POLLING_FAILED, chỉ ra thủ tục kết nối đã hỏng, hoặc PGRES_POLLING_OK, chỉ ra kết nối đã được tạo ra thành công.

Bất kỳ lúc nào trong khi kết nối, tình trạng kết nối có thể được kiểm tra bằng việc gọi PQstatus. Nếu điều này đưa ra CONNECTION_BAD, thì thủ tục kết nối hỏng, còn nếu nó đưa ra CONNECTION_OK, thì kết nối là sẵn sàng. Cả 2 tình trạng đều có thể dò tìm ra được như nhau từ giá trị trả về của PQconnectPoll, được mô tả ở trên. Các tình trạng khác cũng có thể xảy ra trong quá trình (và chỉ trong quá trình) một thủ tục kết nối không đồng bộ. Chúng chỉ ra tình trạng hiện hành của thủ tục kết nối và có thể là hữu dụng cung cấp phản hồi cho người sử dụng, ví dụ thế. Các tình trạng đó là:

CONNECTION_STARTED
: Chờ kết nối sẽ được thực hiện.

CONNECTION_MADE
: Kết nối OK; chờ gửi.

CONNECTION_AWAITING_RESPONSE
: Chờ trả lời từ máy chủ.

CONNECTION_AUTH_OK
: Nhận được xác thực; chờ phần phụ trợ khởi động để kết thúc.

CONNECTION_SSL_STARTUP
: Thương lượng mã hóa SSL.

CONNECTION_SETENV
: Thương lượng các thiết lập tham số hướng môi trường.

Lưu ý rằng, dù các hằng số đó sẽ vẫn là (để duy trì tính tương thích), một ứng dụng nên không bao giờ dựa vào điều đó xảy ra theo một trật tự đặc biệt, hoặc trong tất cả, hoặc trong tình trạng luôn là một trong các giá trị được ghi thành tài liệu đó. Một ứng dụng có thể làm thứ gì đó giống thế này:

```
switch(PQstatus(conn))
{
        case CONNECTION_STARTED:
                feedback = "Connecting...";
                break;
        case CONNECTION_MADE:
                feedback = "Connected to server...";
                break;
.
.
.
        default:
                feedback = "Connecting...";
}
```

Tham số kết nối connect_timeout bị bỏ qua khi sử dụng PQconnectPoll; đây là trách nhiệm của ứng dụng để quyết định liệu một lượng thời gian vượt mức đã qua hay chưa. Nếu không PQconnectStart được một vòng lặp của PQconnectPoll đi theo là tương đương với Pqconnectdb.

Lưu ý rằng nếu PQconnectStart trả về một con trỏ không null, thì bạn phải gọi PQfinish khi bạn hoàn thành với nó, để sắp xếp cấu trúc và bất kỳ khối bộ nhớ nào có liên quan. Điều này phải được làm xong thậm chí nếu cố gắng kết nối hỏng hoặc bị bỏ qua.

PQconndefaults

Trả về các lựa chọn kết nối mặc định.

```
PQconninfoOption *PQconndefaults(void);
typedef struct
{
        char    *keyword;       /* The keyword of the option */
        char    *envvar;        /* Fallback environment variable name */
        char    *compiled;      /* Fallback compiled in default value */
        char    *val;           /* Option's current value, or NULL */
        char    *label;         /* Label for field in connect dialog */
        char    *dispchar;      /* Indicates how to display this field
                                   in a connect dialog. Values are:
                                   "" Display entered value as is
                                   "*" Password field - hide value
                                   "D" Debug option - don't show by default */
        int dispsize; /* Field size in characters for dialog */
} PQconninfoOption;
```

Trả về mảng các lựa chọn của một kết nối. Điều này có thể được sử dụng để xác định tất cả các lựa chọn PQconnectdb có khả năng và các giá trị mặc định hiện hành của chúng. Giá trị trả về chỉ ra một mảng các cấu trúc PQconninfoOption, nó kết thúc với một khoản đầu vào có một con trỏ keyword null. Con trỏ null đó được trả về nếu bộ nhớ có thể không được phân bổ. Lưu ý rằng các giá trị mặc định hiện hành (các trường val) sẽ phụ thuộc vào các biến môi trường và ngữ cảnh khác. Các bộ gọi phải hành xử với các dữ liệu các lựa chọn kết nối như là chỉ đọc.

Sau việc xử lý mảng các lựa chọn, hãy giải phóng nó bằng việc truyền nó tới PQconninfoFree. Nếu điều này không được thực hiện, thì một lượng nhỏ bộ nhớ sẽ bị rò rỉ đối với từng lời gọi

tới Pqconndefaults.

PQconninfoParse

Trả về các lựa chọn kết nối được phân tích từ chuỗi kết nối được cung cấp.

```
PQconninfoOption *PQconninfoParse(const char *conninfo, char **errmsg);
```

Phân tích cú pháp một chuỗi kết nối và trả về các lựa chọn kết quả như một mảng; hoặc trả về NULL nếu có một vấn đề với chuỗi kết nối đó. Điều này có thể được sử dụng để xác định các lựa chọn PQconnectdb trong chuỗi kết nối được cung cấp. Giá trị trả về chỉ tới một mảng các cấu trúc PQconninfoOption, nó kết thúc với một khoản đầu vào có con trỏ keyword null.

Lưu ý rằng chỉ các lựa chọn được chỉ định rõ ràng trong chuỗi đó sẽ có các giá trị được thiết lập trong mảng kết quả; không mặc định nào được chèn vào.

Nếu errmsg là không NULL, thì *errmsg được thiết lập về NULL thành công, nếu không về một chuỗi lỗi malloc giải thích vấn đề đó. (Cũng có khả năng *errmsg sẽ được thiết lập về NULL thậm chí khi NULL được trả về; điều này chỉ ra một tình huống tràn bộ nhớ [out-of-memory]).

Sau khi xử lý mảng các lựa chọn, hãy giải phóng nó bằng việc truyền nó tới PQconninfoFree. Nếu điều này không được thực hiện, thì một số bộ nhớ sẽ bị rò rỉ đối với từng cuộc gọi tới PQconninfoParse. Ngược lại, nếu một lỗi xảy ra và errmsg không là NULL, thì hãy chắc chắn phải giải phóng chuỗi lỗi bằng việc sử dụng Pqfreemem.

PQfinish

Đóng kết nối tới máy chủ. Cũng giải phóng bộ nhớ được sử dụng bằng đối tượng PGconn.

```
void PQfinish(PGconn *conn);
```

Lưu ý rằng nếu cố gắng kết nối máy chủ bị hỏng (như được PQstatus chỉ ra), thì ứng dụng sẽ gọi PQfinish để giải phóng bộ nhớ được đối tượng PGconn sử dụng. Con trỏ PGconn phải không được sử dụng một lần nữa sau khi PQfinish đã bị hoãn.

PQreset

Thiết lập lại kênh kết nối tới máy chủ.

```
void PQreset(PGconn *conn);
```

Hàm này sẽ đóng kết nối tới máy chủ và cố tái thiết lập một kết nối mới tới chính máy chủ đó, bằng việc sử dụng tất cả các tham số y hệt được sử dụng trước đó. Điều này có thể là hữu dụng cho sự phát hiện lỗi nếu một kết nối làm việc bị mất.

PQresetStart
PQresetPoll

Thiết lập lại kênh giao tiếp tới máy chủ, theo một cách thức không khóa.

```
int PQresetStart(PGconn *conn);
PostgresPollingStatusType PQresetPoll(PGconn *conn);
```

Các hàm này sẽ đóng kết nối tới máy chủ và cố thiết lập lại một kết nối mới tới chính máy chủ đó, bằng việc sử dụng tất cả các tham số y hệt được sử dụng trước đó. Điều này có thể là hữu dụng cho phát hiện lỗi nếu một kết nối làm việc bị mất. Chúng khác với PQreset (ở trên)

vì chúng hành động theo một cách thức không khóa. Các hàm đó chịu các giới hạn y hệt như PQconnectStartParams, PQconnectStart và PQconnectPoll.

Để khởi tạo một thiết lập lại kết nối, hãy gọi PQresetStart. Nếu nó trả về 0, sự thiết lập lại đó hỏng. Nếu nó trả về 1, thì hãy khảo sát sự thiết lập lại đó bằng PQresetPoll theo cách chính xác y hệt như bạn có thể tạo kết nối đó bằng việc sử dụng PQconnectPoll.

31.2. Hàm tình trạng kết nối

Các hàm đó có thể được sử dụng để hỏi về tình trạng của một đối tượng kết nối cơ sở dữ liệu đang tồn tại.

Mẹo: Các lập trình viên ứng dụng libpq nên cẩn thận để duy trì bổ trợ được mô tả bên dưới để có được các nội dung của PGconn. Tham chiếu tới các trường nội bộ PGconn bằng việc sử dụng libpq-int.h không được khuyến cáo vì chúng tuân theo sự thay đổi trong tương lai.

Các hàm sau đây trả về các giá trị tham số được thiết lập khi kết nối. Các giá trị đó được cố định cho tệp đối tượng PGconn.

PQdb

Trả về tên cơ sở dữ liệu của kết nối.

```
char *PQdb(const PGconn *conn);
```

PQuser

Trả về tên người sử dụng của kết nối.

```
char *PQuser(const PGconn *conn);
```

PQpass

Trả về mật khẩu của kết nối.

```
char *PQpass(const PGconn *conn);
```

PQhost

Trả về tên máy chủ host của kết nối.

```
char *PQhost(const PGconn *conn);
```

PQport

Trả về cổng của kết nối

```
char *PQport(const PGconn *conn);
```

PQtty

Trả về gỡ lỗi TTY của kết nối. (Điều này là lỗi thời, vì máy chủ không còn chú ý tới thiết lập TTY, nhưng hàm đó vẫn còn cho tính tương thích ngược).

```
char *PQtty(const PGconn *conn);
```

PQoptions

Trả về các lựa chọn dòng lệnh được truyền theo yêu cầu kết nối.

```
char *PQoptions(const PGconn *conn);
```

Các hàm sau đây trả về dữ liệu về tình trạng mà có thể thay đổi khi các hoạt động được thực thi trên đối tượng Pgconn.

PQstatus

Trả về tình trạng kết nối

```
ConnStatusType PQstatus(const PGconn *conn);
```

Tình trạng đó có thể là một trong số các giá trị. Tuy nhiên, chỉ 2 trong số đó được coi là nằm ngoài một thủ tục kết nối không đồng bộ: CONNECTION_OK và CONNECTION_BAD. Một kết nối tốt tới cơ sở dữ liệu có tình trạng CONNECTION_OK. Một cố gắng kết nối bị hỏng được đánh tín hiệu bằng CONNECTION_BAD. Thông thường, một tình trạng OK sẽ vẫn giữ như vậy cho tới PQfinish, nhưng một sự hỏng các giao tiếp có thể làm cho tình trạng đó thay đổi thành CONNECTION_BAD sớm. Trong trường hợp đó ứng dụng có thể cố phục hồi bằng việc gọi Pqreset.

Xem đầu vào cho PQconnectStartParams, PQconnectStart và PQconnectPoll với lưu ý tới các mã tình trạng khác có thể được thấy.

PQtransactionStatus

Trả về tình trạng đang trong giao dịch hiện hành của máy chủ.

```
PGTransactionStatusType PQtransactionStatus(const PGconn *conn);
```

Tình trạng đó có thể là PQTRANS_IDLE (currently idle), PQTRANS_ACTIVE (một lệnh là đang trong tiến trình), PQTRANS_INTRANS (nhàn rỗi, trong một khối giao dịch hợp lệ), hoặc PQTRANS_INERROR (nhàn rỗi, trong một khối giao dịch bị hỏng). PQTRANS_UNKNOWN được báo cáo nếu kết nối là tồi. PQTRANS_ACTIVE được nêu chỉ khi một truy vấn từng được gửi tới máy chủ và còn chưa hoàn tất.

Cảnh báo

PQtransactionStatus sẽ đưa ra các kết quả không đúng khi sử dụng một máy chủ PostgreSQL 7.3 mà có tham số autocommit được đặt về tắt (off). Tính năng tự động thực hiện ở phía máy chủ đã không được tán thành và không tồn tại trong các phiên bản máy chủ sau này.

PQparameterStatus

Tra cứu một thiết lập tham số hiện hành của máy chủ.

```
const char *PQparameterStatus(const PGconn *conn, const char *paramName);
```

Các giá trị tham số nhất định được máy chủ báo cáo tự động khi khởi động kết nối hoặc bất kỳ khi nào các giá trị của chúng thay đổi. PQparameterStatus có thể được sử dụng để chất vấn các thiết lập đó. Nó trả về giá trị hiện hành của tham số nếu biết, hoặc NULL nếu không được biết.

Các tham số được nêu như phiên bản hiện hành bao gồm server_version, server_encoding, client_encoding, application_name, is_superuser, session_authorization, DateStyle, IntervalStyle, TimeZone, integer_datetimes, và standard_conforming_strings. (server_encoding, TimeZone, và integer_datetimes đã không được nêu trong các phiên bản trước 8.0;

standard_conforming_strings đã không được nêu trong các phiên bản trước 8.1; IntervalStyle đã không được nêu trong các phiên bản trước 8.4; application_name đã không được nêu trong các phiên bản trước 9.0). Lưu ý rằng server_version, server_encoding và integer_datetimes không thể thay đổi sau khi khởi động.

Các máy chủ với các giao thức trước phiên bản 3.0 không nêu các thiết lập tham số, nhưng libpq bao gồm logic để có được các giá trị cho server_version và client_encoding bất kỳ cách gì. Các ứng dụng được khuyến khích sử dụng PQparameterStatus thay vì mã hiện đại để xác định các giá trị đó. (Tuy nhiên hãy thận trọng rằng trong một kết nối trước 3.0, việc thay đổi client_encoding thông qua SET sau khởi động kết nối sẽ không được PQparameterStatus phản ánh). Đối với server_version, xem thêm PQserverVersion, nó trả về thông tin ở dạng số mà là dễ dàng hơn nhiều để so sánh.

Nếu không giá trị nào cho standard_conforming_strings được nêu, thì các ứng dụng có thể giả thiết nó là tắt (off), đó là, các dấu chéo ngược được xem như là các thoát trong các hằng chuỗi. Hơn nữa, sự hiện diện của tham số này có thể được lấy như một chỉ số mà cú pháp chuỗi thoát (E'...') được chấp nhận.

Dù con trỏ được trả về được khai báo const, trong thực tế nó trỏ tới kho có thể biến đổi có liên quan tới cấu trúc PGconn. Là không khôn ngoan để giả thiết con trỏ đó sẽ vẫn giữ là hợp lệ xuyên khắp các truy vấn.

PQprotocolVersion

Chất vấn giao thức mặt tiền/phần phụ trợ (frontend/backend) đang được sử dụng.

```
int PQprotocolVersion(const PGconn *conn);
```

Các ứng dụng có thể muốn sử dụng điều này để xác định liệu các tính năng nhất định nào đó được hỗ trợ hay không. Hiện hành, các giá trị có khả năng là 2 (giao thức 2.0), 3 (giao thức 3.0), hoặc zero (kết nối tồi). Điều này sẽ không thay đổi sau khi khởi tạo kết nối được hoàn tất, nhưng nó có thể về lý thuyết thay đổi trong quá trình thiết lập lại kết nối. Giao thức 3.0 thường được sử dụng khi giao tiếp với các máy chủ PostgreSQL 7.4 hoặc sau này; máy chủ trước 7.4 chỉ hỗ trợ giao thức 2.0. (Giao thức 1.0 là lỗi thời và không được libpq hỗ trợ).

PQserverVersion

Trả về một trình diễn số nguyên phiên bản phụ trợ (backend).

```
int PQserverVersion(const PGconn *conn);
```

Các ứng dụng có thể sử dụng điều này để xác định phiên bản máy chủ cơ sở dữ liệu mà chúng được kết nối tới. Số đó được hình thành bằng việc biến đổi các số chính, phụ, và sửa lại thành 2 số có 2 chữ số thập phân và nối chúng vào với nhau. Ví dụ, phiên bản 8.1.5 sẽ được trả về như là 80105, và phiên bản 8.2 sẽ được trả về như là 80200 (các số zero ở đầu sẽ không được chỉ ra). Zero được trả về nếu kết nối là tồi.

PQerrorMessage

Trả về thông điệp lỗi gần nhất được một hoạt động trong kết nối tạo ra.

```
char *PQerrorMessage(const PGconn *conn);
```

Gần như tất cả các hàm libpq sẽ thiết lập một thông điệp cho PQerrorMessage nếu chúng hỏng. Lưu ý rằng bằng việc qui ước libpq, một kết quả PQerrorMessage không rỗng có thể là nhiều dòng, và sẽ bao gồm một dòng mới phía sau. Trình gọi sẽ không giải phóng kết quả trực tiếp. Nó sẽ được giải phóng khi điều khiển PGconn có liên quan được truyền tới PQfinish. Chuỗi kết quả sẽ không được kỳ vọng giữ là y hệt khắp các hoạt động trong cấu trúc PGconn.

PQsocket

Có số trình mô tả tệp của socket kết nối tới máy chủ. Một trình mô tả hợp lệ sẽ là lớn hơn hoặc bằng 0; một kết quả -1 chỉ ra rằng không kết nối máy chủ nào hiện đang được mở.

```
int PQsocket(const PGconn *conn);
```

PQbackendPID

Trả về ID tiến trình (PID) của tiến trình máy chủ phụ trợ điều khiển kết nối này.

```
int PQbackendPID(const PGconn *conn);
```

PID phần phụ trợ là hữu dụng cho các mục tiêu gỡ lỗi và cho sự so sánh với các thông điệp NOTIFY (nó bao gồm PID của việc thông báo tiến trình phần phụ trợ). Lưu ý rằng PID thuộc về một tiến trình đang thực thi trên host máy chủ cơ sở dữ liệu, chứ không phải local host!

PQconnectionNeedsPassword

Trả về đúng (1) nếu phương pháp xác thực kết nối đòi hỏi một mật khẩu, nhưng không sẵn sàng cái nào. Trả về sai (0) nếu không.

```
int PQconnectionNeedsPassword(const PGconn *conn);
```

Hàm này có thể được áp dụng sau một cố gắng kết nối bị hỏng để quyết định liệu có nhắc người sử dụng về một mật khẩu hay không.

PQconnectionUsedPassword

Trả về đúng (1) nếu phương pháp xác thực kết nối sử dụng mật khẩu. Trả về sai (0) nếu không.

```
int PQconnectionUsedPassword(const PGconn *conn);
```

Hàm này có thể được áp dụng sau hoặc một cố gắng kết nối hỏng hoặc thành công để dò tìm xem liệu máy chủ có đòi hỏi một mật khẩu hay không.

PQgetssl

Trả về cấu trúc SSL được sử dụng trong kết nói, hoặc null nếu SSL không được sử dụng.

```
SSL *PQgetssl(const PGconn *conn);
```

Cấu trúc này có thể được sử dụng để kiểm tra các mức mã hóa, kiểm tra các chứng thực máy chủ, và hơn thế. Hãy tham chiếu tới tài liệu OpenSSL để có thông tin về cấu trúc này.

Bạn phải xác định USE_SSL để có được mẫu đúng cho hàm này. Làm thế cũng sẽ tự động đưa ssl.h từ OpenSSL.

31.3. Hàm thực thi lệnh

Một khi một kết nối tới một máy chủ cơ sở dữ liệu được thiết lập thành công, thì các hàm được mô tả ở đây được sử dụng để thực hiện các truy vấn và các lệnh SQL.

31.3.1. Các hàm chính

PQexec

Thực hiện một lệnh tới máy chủ và chờ kết quả.

```
PGresult *PQexec(PGconn *conn, const char *command);
```

Trả về một con trỏ PGresult hoặc có khả năng một con trỏ null. Một con trỏ không null sẽ thường được trả về ngoại trừ trong các điều kiện tràn bộ nhớ hoặc các lỗi nghiêm trọng như không có khả năng gửi lệnh tới máy chủ. Nếu một con trỏ null được trả về, thì nó sẽ được đối xử giống như một kết quả PGRES_FATAL_ERROR. Hãy sử dụng PQerrorMessage để có thêm thông tin về các lỗi như vậy.

Được phép để đưa vào nhiều lệnh SQL (phân cách nhau bằng dấu chấm phẩy) trong chuỗi lệnh. Nhiều truy vấn được gửi trong một lời gọi duy nhất PQexec sẽ được xử lý trong một giao dịch duy nhất, trừ phi có các lệnh rõ ràng BEGIN/COMMIT được đưa vào trong chuỗi truy vấn để chia nó thành nhiều giao dịch. Tuy nhiên hãy lưu ý rằng cấu trúc PGresult được trả về chỉ mô tả kết quả của lệnh mới nhất được thực thi từ chuỗi đó. Nếu một trong các lệnh bị hỏng, thì việc xử lý chuỗi sẽ dừng với nó và PGresult được trả về mô tả điều kiện lỗi đó.

PQexecParams

Thực hiện một lệnh tới máy chủ và chờ kết quả, với khả năng truyền các tham số một cách riêng rẽ từ văn bản lệnh SQL.

```
PGresult *PQexecParams(PGconn *conn,
                       const char *command,
                       int nParams,
                       const Oid *paramTypes,
                       const char * const *paramValues,
                       const int *paramLengths,
                       const int *paramFormats,
                       int resultFormat);
```

PQexecParams là giống như PQexec, nhưng đưa ra chức năng bổ sung: các giá trị tham số có thể được chỉ định riêng rẽ từ chuỗi lệnh đúng phù hợp, và các kết quả truy vấn có thể được yêu cầu hoặc ở định dạng văn bản hoặc nhị phân. PQexecParams chỉ được hỗ trợ trong các kết nối giao thức 3.0 hoặc sau này; nó sẽ hỏng khi sử dụng giao thức 2.0.

Các đối số hàm là:

conn

Đối tượng kết nối sẽ gửi lệnh qua.

command

Chuỗi lệnh SQL sẽ được thực thi. Nếu các tham số được sử dụng, thì chúng được tham chiếu tới trong chuỗi lệnh như là $1, $2, …

nParams

Số các tham số được áp dụng; đây là độ dài các mảng paramTypes[], paramValues[], paramLengths[], và paramFormats[]. (Các con trỏ mảng có thể là NULL khi nParams

là zero).

paramTypes[]

Chỉ định, bằng OID, các dạng dữ liệu sẽ được chỉ định cho các ký hiệu tham số. Nếu paramTypes là NULL, hoặc bất kỳ yếu tố đặc biệt nào trong mảng là zero, thì máy chủ suy luận ra một dạng dữ liệu cho biểu tượng tham số theo cách y hệt như nó có thể làm cho một chuỗi hằng không có dạng.

paramValues[]

Chỉ định các giá trị thực của các tham số. Một con trỏ null trong mảng này có nghĩa là tham số tương ứng là null; nếu không thì con trỏ đó chỉ tới một chuỗi văn bản có kết thúc bằng zero (đối với định dạng văn bản) hoặc dữ liệu nhị phân ở định dạng được máy chủ kỳ vọng (đối với định dạng nhị phân).

paramLengths[]

Chỉ định độ dài dữ liệu thực các tham số định dạng nhị phân. Nó bị bỏ qua đối với các tham số null và các tham số định dạng văn bản. Con trỏ mảng có thể là null khi không có các tham số nhị phân.

paramFormats[]

Chỉ định liệu các tham số có là văn bản hay không (đặt zero vào khoản đầu vào mảng cho tham số tương ứng) hoặc nhị phân (đặt 1 vào khoản đầu vào mảng cho tham số tương ứng). Nếu con trỏ mảng là null thì tất cả các tham số sẽ được giả thiết là các chuỗi văn bản.

Các giá trị được truyền ở định dạng nhị phân đòi hỏi tri thức đại diện nội bộ được phần phụ trợ (backend) kỳ vọng. Ví dụ, các số nguyên phải được truyền theo trật tự byte mạng. Các giá trị numeric truyền qua đòi hỏi tri thức về định dạng kho máy chủ, như được triển khai trong src/backend/utils/adt/numeric.c::numeric_send() và src/backend/utils/adt/numeric.c::numeric_recv().

resultFormat

Chỉ định zero để có các kết quả ở định dạng văn bản, hoặc 1 để có các kết quả ở định dạng nhị phân. (Hiện không có điều khoản để có các cột kết quả khác nhau ở các định dạng khác nhau, dù điều đó có khả năng trong giao thức nằm bên dưới).

Ưu thế trước hết của PQexecParams hơn với PQexec là các giá trị tham số có thể được chia tách khỏi chuỗi lệnh, vì thế tránh được nhu cầu của việc cho vào các dấu ngoặc và việc thoát nặng nề và hay bị lỗi.

Không giống như PQexec, PQexecParams cho phép hầu như 1 lệnh SQL trong chuỗi được đưa ra. (Có thể có các dấu chấm phẩy trong đó, nhưng không nhiều hơn một lệnh không rỗng). Đây là một hạn chế của giao thức nằm bên dưới, nhưng có vài tiện dụng như một sự bảo vệ thêm chống lại các cuộc tấn công tiêm SQL.

Mẹo: Việc chỉ định các dạng tham số thông qua các OID là nặng nề, đặc biệt nếu bạn ưa thích hơn không buộc bằng tay các giá trị OID đặc biệt trong chương trình của bạn. Tuy

nhiên, bạn có thể tránh làm thế thậm chí trong các trường hợp nơi mà bản thân máy chủ không thể xác định được dạng tham số, hoặc chọn một dạng khác so với dạng bạn muốn. Trong văn bản lệnh SQL, hãy gắn một cast rõ ràng tới ký hiệu tham số để chỉ ra dạng dữ liệu nào bạn sẽ gửi. Ví dụ:

```
SELECT * FROM mytable WHERE x = $1::bigint;
```

Điều này ép tham số $1 sẽ được đối xử như là bigint, trong khi đó mặc định nó có thể được chỉ định dạng y hệt như x. Việc ép quyết định dạng tham số, hoặc cách này hoặc bằng việc chỉ định một OID dạng số, được mạnh mẽ khuyến cáo khi gửi các giá trị tham số ở định dạng nhị phân, vì định dạng nhị phân có ít dư thừa hơn định dạng văn bản và vì thế có ít cơ hội máy chủ sẽ dò tìm ra lỗi không khớp dạng cho bạn.

PQprepare

Thực hiện một yêu cầu để tạo ra một lệnh được chuẩn bị với các tham số được đưa ra, và chờ kết thúc.

```
PGresult *PQprepare(PGconn *conn,
                    const char *stmtName,
                    const char *query,
                    int nParams,
                    const Oid *paramTypes);
```

PQprepare tạo một lệnh được chuẩn bị cho sự thực thi sau này với PQexecPrepared. Tính năng này cho phép các lệnh mà sẽ được sử dụng lặp đi lặp lại sẽ được phân tích cú pháp và được lên kế hoạch chỉ một lần, thay vì mỗi lần chúng được thực thi. PQprepare chỉ được hỗ trợ trong giao thức 3.0 và các kết nối sau này; nó sẽ hỏng khi sử dụng giao thức 2.0.

Hàm này tạo ra một lệnh được chuẩn bị có tên là stmtName từ chuỗi query, nó phải có một lệnh SQL duy nhất. stmtName có thể là "" để tạo một lệnh không có tên, trong trường hợp đó bất kỳ lệnh không có tên tồn tại từ trước đó nào cũng tự động được thay thế; nếu không nó là một lỗi nếu tên lệnh được xác định rồi trong phiên làm việc hiện hành. Nếu bất kỳ tham số nào được sử dụng, thì chúng được tham chiếu tới trong truy vấn như là $1, $2, ... nParams là số các tham số theo đó các dạng sẽ được chỉ định trước trong mảng paramTypes[]. (Con trỏ mảng có thể là NULL khi nParams là zero). paramTypes[] chỉ định, bằng OID, các dạng dữ liệu sẽ được chỉ định cho các ký hiệu tham số. Nếu paramTypes là NULL, hoặc bất kỳ yếu tố đặc biệt nào trong mảng là zero, thì máy chủ chỉ định một dạng dữ liệu cho ký hiệu tham số đó theo cách y hệt mà nó có thể làm cho một chuỗi hằng không có dạng. Hơn nữa, truy vấn đó có thể sử dụng các ký hiệu tham số với các số cao hơn so với nParams; các dạng dữ liệu cũng sẽ được suy luận ra cho các ký hiệu đó. (Xem PQdescribePrepared về một phương tiện để tìm ra các dạng dữ liệu nào đã được suy luận ra).

Như với PQexec, kết quả đó thường là một đối tượng PGresult mà các nội dung của nó chỉ ra thành công hoặc thất bại ở phía máy chủ. Một kết quả null chỉ ra tràn bộ nhớ hoặc không có khả năng gửi lệnh đó hoàn toàn. Hãy sử dụng PQerrorMessage để có thêm thông tin về các lỗi như vậy.

Các lệnh được chuẩn bị để sử dụng với PQexecPrepared cũng có thể được tạo ra bằng việc thực thi

các lệnh PREPARE SQL. Hơn nữa, dù không có hàm libpq cho việc xóa một lệnh được chuẩn bị, thì lệnh DEALLOCATE SQL có thể được sử dụng cho mục đích đó.

PQexecPrepared

Gửi yêu cầu thực thi lệnh được chuẩn bị với các tham số đưa ra, và chờ kết quả.

```
PGresult *PQexecPrepared(PGconn *conn,
                         const char *stmtName,
                         int nParams,
                         const char * const *paramValues,
                         const int *paramLengths,
                         const int *paramFormats,
                         int resultFormat);
```

PQexecPrepared là giống như PQexecParams, nhưng lệnh đó sẽ được thực thi được chỉ định bằng việc đặt tên cho một lệnh được chuẩn bị trước đó, thay vì việc trao một chuỗi truy vấn. Tính năng này cho phép các lệnh sẽ được sử dụng lặp đi lặp lại sẽ được phân tích cú pháp và được lên kế hoạch chỉ một lần, thay vì mỗi lần chúng được thực thi. Lệnh đó phải được chuẩn bị trước đó trong phiên làm việc hiện hành. PQexecPrepared chỉ được hỗ trợ trong giao thức 3.0 và các kết nối sau này; nó sẽ hỏng khi sử dụng giao thức 2.0.

Các tham số là y hệt như với PQexecParams, ngoại trừ là tên của một lệnh được chuẩn bị được đưa ra thay vì một chuỗi truy vấn, và tham số paramTypes[] là không hiện diện (không cần thiết vì các dạng tham số lệnh được chuẩn bị đã được xác định khi nó từng được tạo ra).

PQdescribePrepared

Đệ trình một yêu cầu để có thông tin về lệnh được chuẩn bị được chỉ định, và chờ sự kết thúc.

```
PGresult *PQdescribePrepared(PGconn *conn, const char *stmtName);
```

PQdescribePrepared cho phép một ứng dụng có được thông tin về một lệnh được chuẩn bị trước đó. PQdescribePrepared chỉ được hỗ trợ trong giao thức 3.0 và các kết nối sau này; nó sẽ hỏng khi sử dụng giao thức 2.0.

stmtName có thể là "" hoặc NULL để tham chiếu tới lệnh không có tên, nếu không nó phải là tên của một lệnh được chuẩn bị đang tồn tại. Nếu thành công, một PGresult với tình trạng PGRES_COMMAND_OK được trả về. Các hàm PQnparams và PQparamtype có thể được áp dụng cho PGresult này để có được thông tin về các tham số của lệnh được chuẩn bị, và các hàm PQnfields, PQfname, PQftype ... cung cấp thông tin về các cột kết quả (nếu có) của lệnh đó.

PQdescribePortal

Đệ trình một yêu cầu để có thông tin về cổng được chỉ định, và chờ hoàn tất.

```
PGresult *PQdescribePortal(PGconn *conn, const char *portalName);
```

PQdescribePortal cho phép một ứng dụng có được thông tin về một cổng được tạo ra trước đó. (libpq không đưa ra bất kỳ truy cập trực tiếp nào tới các cổng, nhưng bạn có thể sử dụng hàm này để điều tra các thuộc tính của một con trỏ được tạo ra bằng một lệnh SQL DECLARE CURSOR). PQdescribePortal chỉ được hỗ trợ trong giao thức 3.0 và các kết nối sau này; nó sẽ hỏng khi sử dụng giao thức 2.0.

portalName có thể là "" hoặc NULL để tham chiếu tới cổng không có tên, nếu không nó phải là tên của một cổng đang tồn tại. Nếu thành công, một PGresult với tình trạng PGRES_COMMAND_OK được trả về. Các hàm PQnfields, PQfname, PQftype ... có thể được áp dụng cho PGresult để có được thông tin về các cột kết quả (nếu có) của cổng đó.

Cấu trúc PGresult bọc kết quả được máy chủ trả về. Các tham số ứng dụng libpq sẽ cẩn thận duy trì sự trích lọc PGresult. Hãy sử dụng các hàm bổ trợ bên dưới để có được các nội dung của PGresult. Hãy tránh trực tiếp tham chiếu tới các trường của cấu trúc PGresult vì chúng tuân theo sự thay đổi trong tương lai.

PQresultStatus

Trả về tình trạng kết quả của lệnh.

ExecStatusType PQresultStatus(const PGresult *res);

PQresultStatus có thể trả về một trong các giá trị sau:

PGRES_EMPTY_QUERY

Chuỗi này gửi tới máy chủ đã rỗng.

PGRES_COMMAND_OK

Sự hoàn tất thành công của một lệnh trả về dữ liệu (như lệnh SELECT hoặc SHOW).

PGRES_COPY_OUT

Truyền dữ liệu sao chép ra - Copy Out (từ máy chủ) được khởi tạo.

PGRES_COPY_IN

Truyền dữ liệu sao chép vào - Copy In (tới máy chủ) được khởi tạo.

PGRES_BAD_RESPONSE

Trả lời của máy chủ đã không được hiểu.

PGRES_NONFATAL_ERROR

Một lỗi không chết người (một lưu ý hoặc cảnh báo) đã xảy ra.

PGRES_FATAL_ERROR

Một lỗi chết người đã xảy ra.

Nếu tình trạng kết quả là PGRES_TUPLES_OK, thì các hàm được mô tả bên dưới có thể được sử dụng để truy xuất các hàng được truy vấn đó trả về. Lưu ý rằng một lệnh SELECT mà xảy ra để truy xuất các hàng zero vẫn còn chỉ ra PGRES_TUPLES_OK. PGRES_COMMAND_OK là cho các lệnh có thể không bao giờ trả về các hàng (INSERT, UPDATE,...). Trả lời của PGRES_EMPTY_QUERY có thể chỉ ra một lỗi trong phần mềm máy trạm.

Kết quả của tình trạng PGRES_NONFATAL_ERROR sẽ không bao giờ được trả về trực tiếp bằng PQexec hoặc các hàm thực thi truy vấn khác; thay vào đó các kết quả dạng này được truyền tới bộ xử lý thông báo (xem Phần 31.11).

PQresStatus

Chuyển đổi dạng được đánh số được PQresultStatus trả về trong một hàng chuỗi mô tả mã tình trạng đó. Bộ gọi đó sẽ không giải phóng kết quả.

```
char *PQresStatus(ExecStatusType status);
```

PQresultErrorMessage

Trả về thông điệp lỗi có liên quan tới lệnh đó, hoặc một chuỗi rỗng nếu từng không có lỗi.

```
char *PQresultErrorMessage(const PGresult *res);
```

Nếu từng có một lỗi, thì chuỗi được trả về sẽ bao gồm một dòng mới đằng sau. Bộ gọi sẽ không giải phóng kết quả trực tiếp. Nó sẽ được giải phóng khi điều khiển PGresult có liên quan được truyền tới PQclear. Ngay lập tức sau một lời gọi PQexec hoặc PQgetResult, PQerrorMessage (trong kết nối đó) sẽ trả về chuỗi y hệt như PQresultErrorMessage (trong kết quả). Tuy nhiên, PGresult sẽ giữ lại thông điệp lỗi của nó cho tới khi bị hủy, trong khi thông điệp lỗi của kết nối sẽ thay đổi khi các hoạt động tiếp sau sẽ được thực hiện. Hãy sử dụng PQresultErrorMessage khi bạn muốn biết tình trạng có liên quan tới một PGresult cụ thể; hãy sử dụng PQerrorMessage khi bạn muốn biết tình trạng đó từ hoạt động mới nhất ở kết nối đó.

PQresultErrorField

Trả về một trường riêng rẽ của một báo cáo lỗi.

```
char *PQresultErrorField(const PGresult *res, int fieldcode);
```

fieldcode là một mã định danh trường lỗi; xem các ký hiệu được liệt kê bên dưới. NULL được trả về nếu PGresult không là một lỗi hoặc kết quả cảnh báo, hoặc không đưa vào trường được chỉ định. Các giá trị trường sẽ thường không bao gồm một dòng mới ở đuôi. Bộ gọi sẽ không giải phóng kết quả một cách trực tiếp. Nó sẽ được giải phóng khi điều khiển PGresult có liên quan được truyền tới PQclear.

Các mã trường sau đây là sẵn sàng:

PG_DIAG_SEVERITY

Tính nghiêm trọng; các nội dung trường này là ERROR, FATAL, hoặc PANIC (trong một thông điệp lỗi), hoặc WARNING, NOTICE, DEBUG, INFO, hoặc LOG (trong một thông điệp lưu ý), hoặc một bản dịch được bản địa hóa của một trong những thứ đó. Luôn hiện diện.

PG_DIAG_SQLSTATE

Mã SQLSTATE cho lỗi. Mã SQLSTATE nhận diện dạng lỗi đã xảy ra; nó có thể được các ứng dụng mặt tiền sử dụng để thực hiện các hoạt động đặc thù (như việc điều khiển lỗi) trong việc trả lời cho một lỗi cơ sở dữ liệu cụ thể. Xem một danh sách các mã SQLSTATE có thể ở Phụ lục A. Trường này không có khả năng bản địa hóa, và luôn hiện diện.

PG_DIAG_MESSAGE_PRIMARY

Thông điệp lỗi người đọc được trước (thường là một dòng). Luôn hiện diện.

PG_DIAG_MESSAGE_DETAIL

Chi tiết: thông điệp lỗi tùy chọn thứ 2 mang theo nhiều chi tiết hơn về vấn đề đó. Có thể chạy nhiều dòng.

PG_DIAG_MESSAGE_HINT

Mẹo: gợi ý tùy chọn phải làm gì đối với vấn đề đó. Điều này có ý định phân biệt với chi tiết theo đó nó đưa ra khuyến cáo (tiềm tàng không phù hợp) hơn là các sự việc khó khăn. Có thể chạy nhiều dòng.

PG_DIAG_STATEMENT_POSITION

Chuỗi có chứa một số nguyên các dấu thập phân chỉ ra một vị trí con trỏ lỗi như một chỉ số trong chuỗi lệnh gốc. Ký tự đầu có chỉ số là 1, và các vị trí được đo đếm trong các ký tự không là các byte.

PG_DIAG_INTERNAL_POSITION

Điều này được xác định y hệt như trường PG_DIAG_STATEMENT_POSITION, nhưng nó được sử dụng khi vị trí con trỏ tham chiếu tới một lệnh được sinh ra nội bộ hơn là một lệnh được đệ trình từ máy trạm. Trường PG_DIAG_INTERNAL_QUERY sẽ luôn xuất hiện khi trường này xuất hiện.

PG_DIAG_INTERNAL_QUERY

Văn bản của một lệnh được sinh ra nội bộ bị hỏng. Điều này có thể, ví dụ, một truy vấn SQL được một hàm PL/pgSQL đưa ra.

PG_DIAG_CONTEXT

Một chỉ số của ngữ cảnh theo đó lỗi xảy ra. Hiện thời điều này có sự theo dõi ngược kho các lời gọi các hàm ngôn ngữ thủ tục tích cực và các truy vấn được sinh ra nội bộ. Sự theo dõi là một khoản đầu vào theo từng dòng, khoản gần nhất trước tiên.

PG_DIAG_SOURCE_FILE

Tên tệp của vị trí mã nguồn nơi mà lỗi đã được nêu.

PG_DIAG_SOURCE_LINE

Số dòng vị trí mã nguồn nơi mà lỗi đã được nêu.

PG_DIAG_SOURCE_FUNCTION

Tên hàm mã nguồn nêu lỗi.

Máy trạm có trách nhiệm đối với việc định dạng thông tin được hiển thị để đáp ứng các nhu cầu của nó; đặc biệt nó sẽ ngắt các dòng dài như cần thiết. Các ký tự dòng mới xuất hiện trong các trường thông điệp lỗi sẽ được coi như các ngắt đoạn, không phải các ngắt dòng.

Các lỗi được sinh ra nội bộ bằng libpq sẽ có thông điệp về tính nghiêm trọng và trước tiên, nhưng thường không là các trường khác. Các lỗi được máy chủ giao thức trước 3.0 trả về sẽ bao gồm thông điệp về tính nghiêm trọng và trước tiên, và đôi khi một thông điệp chi tiết, nhưng không có các trường khác.

Lưu ý rằng các trường lỗi chỉ sẵn sàng từ các đối tượng PGresult, không từ các đối tượng PGconn; không có hàm PqerrorField.

PQclear

Giải phóng kho lưu trữ có liên quan tới một PGresult. Mỗi kết quả lệnh sẽ được giải phóng qua PQclear khi nó không còn cần thiết nữa.

```
void PQclear(PGresult *res);
```

Bạn có thể giữ một đối tượng PGresult lâu bao nhiêu tùy theo nhu cầu của bạn; nó không đi khỏi khi bạn đưa ra một lệnh mới, thậm chí không nếu bạn đóng một kết nối. Để loại bỏ nó, bạn phải gọi PQclear. Không làm điều này sẽ làm cho bộ nhớ rò rỉ trong ứng dụng của bạn.

31.3.2. Truy xuất thông tin kết quả truy vấn

Các hàm đó được sử dụng để trích xuất thông tin từ một đối tượng PGresult đại diện cho một kết quả truy vấn thành công (đó là truy vấn có tình trạng PGRES_TUPLES_OK). Chúng cũng có thể được sử dụng để trích xuất thông tin từ một hoạt động Mô tả - Describe thành công; một kết quả Describe có tất cả thông tin cột y hệt mà sự thực thi thực tế của truy vấn đó có thể cung cấp, nhưng nó có các hàng zero. Đối với các đối tượng với các giá trị tình trạng khác, các hàm đó sẽ hành động dường như kết quả đó có các hàng zero và các cột zero.

PQntuples

Trả về số hàng (bộ dữ liệu) trong kết quả truy vấn đó. Vì nó trả về một kết quả số nguyên, nên các tập hợp kết quả lớn có lẽ làm tràn giá trị trả về trong các hệ điều hành 32 bit.

```
int PQntuples(const PGresult *res);
```

PQnfields

Trả về số cột (trường) theo từng hàng kết quả truy vấn.

```
int PQnfields(const PGresult *res);
```

PQfname

Trả về tên cột có liên quan tới số cột được đưa ra. Số lượng cột bắt đầu ở 0. Bộ gọi sẽ không giải phóng kết quả trực tiếp. Nó sẽ được giải phóng khi điều khiển PGresult có liên quan được truyền tới PQclear.

```
char *PQfname(const PGresult *res,
              int column_number);
```

NULL được trả về nếu số cột vượt ra khỏi dải.

PQfnumber

Trả về số cột có liên quan tới tên cột được đưa ra.

```
int PQfnumber(const PGresult *res,
              const char *column_name);
```

-1 được trả về nếu tên được đưa ra không khớp với bất kỳ cột nào.

Tên được đưa ra được đối xử giống một mã định danh trong một lệnh SQL, đó là, nó bị đánh giá thấp trừ phi nằm trong các dấu nháy kép. Ví dụ, kết quả một truy vấn được đưa ra được sinh ra từ lệnh SQL:

```
SELECT 1 AS FOO, 2 AS "BAR";
```

chúng ta có thể có các kết quả:

```
PQfname(res, 0)             foo
PQfname(res, 1)             BAR
PQfnumber(res, "FOO")       0
PQfnumber(res, "foo")       0
```

```
PQfnumber(res, "BAR")           -1
PQfnumber(res, "\"BAR\"")       1
```

PQftable

Trả về OID của bảng từ đó cột được đưa ra đã được lấy về. Số cột bắt đầu từ 0.

```
Oid PQftable(const PGresult *res,
             int column_number);
```

InvalidOid được trả về nếu số cột vượt khỏi dãy, hoặc nếu cột được chỉ định không phải là tham chiếu đơn giản tới một cột của bảng, hoặc khi sử dụng giao thức trước 3.0. Bạn có thể truy vấn bảng hệ thống pg_class để xác định chính xác bảng nào được tham chiếu.

Dạng Oid và hằng số InvalidOid sẽ được xác định khi bạn đưa vào tệp đầu đề libpq. Chúng cả 2 sẽ là dạng số nguyên.

PQftablecol

Trả về số cột (trong bảng của nó) của cột tạo ra cột kết quả truy vấn được chỉ định. Số cột kết quả truy vấn bắt đầu từ 0, nhưng các cột của bảng có các số khác zero.

```
int PQftablecol(const PGresult *res,
                int column_number);
```

Zero được trả về nếu số cột vượt khỏi dãy, hoặc nếu cột được chỉ định không là một tham chiếu đơn giản tới một cột của bảng, hoặc khi sử dụng giao thức trước 3.0.

PQfformat

Trả về mã định dạng chỉ ra định dạng của cột được đưa ra. Số cột bắt đầu từ 0.

```
int PQfformat(const PGresult *res,
              int column_number);
```

Mã định dạng zero chỉ ra sự đại diện dữ liệu văn bản, trong khi mã định dạng 1 chỉ đại diện nhị phân. (Các mã khác được giữ lại cho định nghĩa trong tương lai).

PQftype

Trả về dạng dữ liệu có liên quan tới số cột được đưa ra. Số nguyên được trả về là số dạng OID nội bộ. Số cột bắt đầu từ 0.

```
Oid PQftype(const PGresult *res,
            int column_number);
```

Bạn có thể truy vấn bảng hệ thống pg_type để có được các tên và thuộc tính của các dạng dữ liệu khác nhau. Các OID của các dạng dữ liệu được xây dựng sẵn được xác định trong tệp src/include/catalog/pg_type.h trong cây nguồn.

PQfmod

Trả về trình sửa đổi dạng của cột có liên quan tới số cột được đưa ra. Số cột bắt đầu từ 0.

```
int PQfmod(const PGresult *res,
           int column_number);
```

Sự giải thích về các giá trị của trình sửa đổi là đặc thù dạng; chúng thường chỉ ra các giới hạn về độ chính xác hoặc kích cỡ. Giá trị -1 được sử dụng để chỉ "không thông tin nào sẵn sàng". Hầu hết các dạng dữ liệu không sử dụng các trình sửa đổi, trong trường hợp đó giá trị đó luôn là -1.

PQfsize

Trả về kích cỡ theo byte của cột có liên quan tới số cột được đưa ra. Số cột bắt đầu từ 0.

```
int PQfsize(const PGresult *res,
            int column_number);
```

PQfsize trả về không gian được phân bổ cho cột này trong một hàng cơ sở dữ liệu, nói cách khác kích cỡ đại diện dạng dữ liệu nội bộ của máy chủ (tương tự, điều này không thực sự là rất hữu dụng đối với các máy trạm). Một giá trị âm chỉ ra dạng dữ liệu có độ dài thay đổi.

PQbinaryTuples

Trả về 1 nếu PGresult chứa dữ liệu nhị phân và 0 nếu nó chứa các dữ liệu văn bản.

```
int PQbinaryTuples(const PGresult *res);
```

Hàm này bị phản đối (ngoại trừ sử dụng nó trong kết nối với COPY), vì không có khả năng cho một PGresult duy nhất chứa dữ liệu văn bản trong một số cột và các dữ liệu nhị phân trong các cột khác. PQfformat được ưu tiên. PQbinaryTuples trả về 1 chỉ nếu tất cả các cột của kết quả là nhị phân (định dạng 1).

PQgetvalue

Trả về một giá trị trường duy nhất của một cột của một PGresult. Số hàng và cột bắt đầu từ 0. Bộ gọi sẽ không giải phóng kết quả trực tiếp. Nó sẽ được giải phóng khi điều khiển PGresult có liên quan được truyền tới Pqclear.

```
char *PQgetvalue(const PGresult *res,
                 int row_number,
                 int column_number);
```

Đối với các dữ liệu ở định dạng văn bản, giá trị đó được PQgetvalue trả về là một đại diện chuỗi ký tự kết thúc là null của giá trị trường đó. Đối với dữ liệu ở định dạng nhị phân, giá trị đó là trong đại diện nhị phân được xác định bằng các hàm dạng dữ liệu typsend và typreceive. (Giá trị đó thực sự cũng đi theo sau bằng một byte zero trong trường hợp này, nhưng điều đó không phải lúc nào cũng hữu dụng, vì giá trị đó có khả năng chứa các null được nhúng).

Một chuỗi rỗng được trả về nếu giá trị trường đó là null. Xem PQgetisnull để phân biệt các giá trị null với các giá trị chuỗi rỗng.

Con trỏ đó được PQgetvalue trả về chỉ tới kho lưu trữ mà là một phần của cấu trúc Pgresult.

Người ta sẽ không sửa các dữ liệu nó trỏ tới, và người ta phải sao chép rõ ràng các dữ liệu đó vào các kho dữ liệu khác nếu nó được sử dụng qua vòng đời bản thân cấu trúc Pgresult.

PQgetisnull

Kiểm thử một trường đối với giá trị null. Số hàng và cột bắt đầu từ 0.

```
int PQgetisnull(const PGresult *res,
                int row_number,
                int column_number);
```

Hàm này trả về 1 nếu trường đó là null và 0 nếu nó có một giá trị khác null. (Lưu ý rằng PQgetvalue sẽ trả về một chuỗi rỗng, không phải một con trỏ null, đối với một trường null).

PQgetlength

Trả về độ dài thực của giá trị một trường theo byte. Số hàng và cột bắt đầu từ 0.

```
int PQgetlength(const PGresult *res,
                int row_number,
                int column_number);
```

Đây là độ dài dữ liệu thực đối với giá trị dữ liệu đặc biệt, đó là, kích cỡ của đối tượng được PQgetvalue trỏ tới. Đối với định dạng dữ liệu văn bản thì điều này là y hệt như strlen(). Đối với định dạng nhị phân thì điều này là thông tin cơ bản. Lưu ý rằng người ta sẽ không dựa vào PQfsize để có được độ dài dữ liệu thực.

PQnparams

Trả về số các tham số của một lệnh được chuẩn bị.

```
int PQnparams(const PGresult *res);
```

Hàm này chỉ hữu dụng khi điều tra kết quả của PQdescribePrepared. Đối với các dạng truy vấn khác nó sẽ trả về zero.

PQparamtype

Trả về dạng dữ liệu của tham số lệnh được chỉ định. Số các tham số bắt đầu từ 0.

```
Oid PQparamtype(const PGresult *res, int param_number);
```

Hàm này chỉ hữu dụng khi điều tra kết quả của PQdescribePrepared. Đối với các dạng truy vấn khác thì nó sẽ trả về zero.

PQprint

In ra tất cả các hàng và, một cách tùy chọn, các tên cột tới dòng đầu ra được chỉ định.

```
void PQprint(FILE *fout, /* output stream */
             const PGresult *res,
             const PQprintOpt *po);
typedef struct
{
        pqbool header; /* print output field headings and row count */
        pqbool align; /* fill align the fields */
        pqbool standard; /* old brain dead format */
        pqbool html3; /* output HTML tables */
        pqbool expanded; /* expand tables */
        pqbool pager; /* use pager for output if needed */
        char *fieldSep; /* field separator */
        char *tableOpt; /* attributes for HTML table element */
        char *caption; /* HTML table caption */
        char **fieldName; /* null-terminated array of replacement field names */
} PQprintOpt;
```

Hàm này trước đó từng được psql sử dụng để in các kết quả truy vấn, nhưng điều này không còn như vậy nữa. Lưu ý rằng nó giả thiết tất cả các dữ liệu là ở định dạng văn bản.

31.3.3. Truy xuất thông tin kết quả khác

Các hàm đó được sử dụng để trích xuất thông tin khác từ các đối tượng PGresult.

PQcmdStatus

Trả về thẻ (tag) tình trạng lệnh từ lệnh SQL mà đã sinh ra PGresult.

```
char *PQcmdStatus(PGresult *res);
```

Thường thì điều này chỉ là tên của lệnh, nhưng nó có thể bao gồm các dữ liệu bổ sung như số hàng được xử lý. Bộ gọi sẽ không giải phóng kết quả trực tiếp. Nó sẽ được giải phóng khi điều khiển PGresult có liên quan được truyền tới PQclear.

PQcmdTuples

Trả về số hàng chịu ảnh hưởng vì lệnh SQL.

```
char *PQcmdTuples(PGresult *res);
```

Hàm này trả về một chuỗi có số hàng chịu ảnh hưởng vì lệnh SQL mà đã tạo ra PGresult. Hàm này chỉ có thể được sử dụng sau sự thực thi một lệnh SELECT, CREATE TABLE AS, INSERT, UPDATE, DELETE, MOVE, FETCH, hoặc COPY, hoặc một EXECUTE của một truy vấn được chuẩn bị mà có chứa một lệnh INSERT, UPDATE, hoặc DELETE. Nếu lệnh đã tạo ra PGresult từng là thứ gì khác nữa, thì PQcmdTuples trả về một chuỗi rỗng. Bộ gọi sẽ không giải phóng và trả về giá trị một cách trực tiếp. Nó sẽ được giải phóng khi điều khiển PGresult có liên quan được truyền tới Pqclear.

PQoidValue

Trả về OID hàng được chèn vào, nếu lệnh SQL từng là INSERT mà đã chèn chính xác 1 hàng vào một bảng có các OID, hoặc một EXECUTE của một truy vấn được chuẩn bị có một lệnh INSERT phù hợp. Nếu không, hàm này sẽ trả về InvalidOid. Hàm này cũng sẽ trả về InvalidOid nếu bảng bị ảnh hưởng vì lệnh INSERT không có chứa các OID.

```
Oid PQoidValue(const PGresult *res);
```

PQoidStatus

Trả về một chuỗi với OID của hàng được chèn vào, nếu lệnh SQL từng là một INSERT mà đã chèn chính xác một hàng, hoặc một EXECUTE của một lệnh được chuẩn bị và có một INSERT phù hợp. (Chuỗi sẽ là 0 nếu INSERT đã không chèn chính xác một hàng, hoặc nếu bảng đích không có các OID). Nếu lệnh đó từng không phải là INSERT, thì trả về một chuỗi rỗng.

```
char *PQoidStatus(const PGresult *res);
```

Hàm này bị phản đối có lợi cho PQoidValue. Nó không phải là luồng an toàn.

31.3.4. Các chuỗi thoát để đưa vào trong các lệnh SQL

PQescapeLiteral

```
char *PQescapeLiteral(PGconn *conn, const char *str, size_t length);
```

PQescapeLiteral thoát khỏi một chuỗi để sử dụng bên trong một lệnh SQL. Điều này là hữu dụng khi chèn các giá trị dữ liệu như là các hằng chữ trong các lệnh SQL. Các ký tự nhất định (như các dấu nháy và các dấu chéo ngược) phải được thoát để ngăn chúng khỏi bị trình phân tích cú pháp SQL diễn giải đặc biệt. PQescapeLiteral thực hiện hành động này.

PQescapeLiteral trả về một phiên bản thoát ra của tham số str trong bộ nhớ được phân bổ với malloc(). Bộ nhớ này sẽ được giải phóng bằng việc sử dụng PQfreemem() khi kết quả không còn cần thiết. Một byte zero kết thúc không được yêu cầu, và sẽ không được tính trong length. (Nếu một byte zero kết thúc được thấy trước length thì các byte sẽ được xử lý, PQescapeLiteral sẽ dừng ở zero; hành vi đó vì thế khá giống strncpy). Chuỗi trả về có tất cả các ký tự đặc biệt được thay thế sao cho chúng có thể được xử lý đúng phù hợp bằng trình phân tích cú pháp hằng chuỗi của PostgreSQL. Một byte zero kết thúc cũng được thêm vào. Các dấu nháy đơn phải có xung quanh các hằng chuỗi PostgreSQL sẽ được đưa vào trong chuỗi kết quả.

Nếu có lỗi, PQescapeLiteral trả về NULL và một thông điệp phù hợp được lưu trữ trong đối tượng conn.

> **Mẹo**: Đặc biệt quan trọng phải thực hiện việc thoát ra đúng phù hợp khi điều khiển các chuỗi đã nhận được từ một nguồn không tin cậy. Nếu không sẽ có một rủi ro an toàn: bạn sẽ bị tổn thương vì các cuộc tấn công "tiêm SQL" ở những nơi các lệnh SQL không được mong muốn được đưa vào cơ sở dữ liệu của bạn.

Lưu ý là không cần thiết cũng không đúng phải thực hiện việc thoát ra khi một giá trị dữ liệu được truyền như một tham số riêng rẽ trong PQexecParams hoặc các thủ tục anh em của nó.

PQescapeIdentifier

```
char *PQescapeIdentifier(PGconn *conn, const char *str, size_t length);
```

PQescapeIdentifier thoát ra một chuỗi để sử dụng như một mã nhận diện SQL, như tên một bảng, cột hoặc hàm. Điều này là hữu dụng khi một mã nhận diện được người sử dụng cung cấp có thể có các ký tự đặc biệt mà nếu khác có thể không được trình phân tích cú pháp SQL diễn giải nhà là một phần của mã nhận diện đó, hoặc khi mã nhận diện đó có thể chứa các ký tự chữ hoa mà sẽ được lưu giữ lại.

PQescapeIdentifier trả về một phiên bản của tham số str được thoát ra như một mã nhận diện SQL trong bộ nhớ được phân bổ với malloc(). Bộ nhớ này phải được giải phóng bằng việc sử dụng PQfreemem() khi kết quả không còn cần thiết nữa. Một byte zero kết thúc là không được yêu cầu, và sẽ không được tính trong length. (nếu một byte zero kết thúc được thấy trước length thì các byte sẽ được xử lý, PQescapeIdentifier sẽ dừng ở zero; hành vi đó vì thế khá giống strncpy). Chuỗi trả về có tất cả các ký tự đặc biệt được thay thế sao cho nó sẽ được xử lý đúng phù hợp như một mã nhận diện SQL.

Byte zero kết thúc cũng được thêm vào. Chuỗi trả về cũng được bao quanh bằng các dấu nháy đúp.

Nếu có lỗi, PQescapeIdentifier trả về NULL và một thông điệp phù hợp được lưu trữ trong đối tượng conn.

> **Mẹo**: Như với các hằng chuỗi, để tránh các cuộc tấn công tiêm SQL, các mã nhận diện SQL phải được thoát ra khi chúng được chấp nhận từ một nguồn không tin cậy.

PQescapeStringConn

```
size_t PQescapeStringConn(PGconn *conn,
                          char *to, const char *from, size_t length,
                          int *error);
```

PQescapeStringConn thoát ra các hằng chuỗi, rất giống PQescapeLiteral. Không giống như PQescapeLiteral, bộ gọi có trách nhiệm cung cấp một bộ nhớ tạm kích cỡ phù hợp.

Hơn nữa, PQescapeStringConn không sinh ra các dấu nháy đơn mà phải bao quanh các hằng chuỗi PostgreSQL; chúng sẽ được cung cấp trong lệnh SQL mà kết quả được chèn vào. Tham số from chỉ tới ký tự đầu của chuỗi mà sẽ được thoát ra, và tham số length đưa ra số các byte trong chuỗi này. Một byte zero kết thúc không được yêu cầu, và sẽ không được tính tới trong length. (Nếu một byte zero kết thúc được thấy trước length thì các byte sẽ được xử

lý, PQescapeStringConn sẽ dừng ở zero; hành vi này vì thế khá giống strncpy). to sẽ chỉ tới một bộ nhớ tạm mà có khả năng giữ ít nhất một byte nữa so với 2 lần giá trị của length, nếu không thì hành vi đó sẽ không được xác định. Hành vi cũng như vậy không được xác định nếu các chuỗi to và from chồng đè lên nhau.

Nếu tham số error là khác NULL, thì *error được thiết lập về zero nếu thành không, khác zero nếu có lỗi.

Hiện hành các điều kiện lỗi chỉ có khả năng liên quan tới việc mã hóa nhiều byte không hợp lệ trong chuỗi nguồn. Chuỗi đầu ra vẫn còn được sinh ra trong lỗi, nhưng có thể được kỳ vọng rằng máy chủ sẽ từ chối nó như là được tạo ra sai. Nếu có lỗi, thì một thông điệp phù hợp được lưu trữ trong đối tượng conn, bất chấp có hay không error là NULL.

PQescapeStringConn trả về số các byte được ghi vào to, không bao gồm byte zero kết thúc.

PQescapeString

PQescapeString là một phiên bản cũ hơn, bị phản đối của PqescapeStringConn.

```
size_t PQescapeString (char *to, const char *from, size_t length);
```

Sự khác biệt duy nhất với PQescapeStringConn là PQescapeString không lấy các tham số PGconn hoặc error. Vì điều này, không thể tinh chỉnh hành vi của nó dựa vào các thuộc tính kết nối (như việc mã hóa ký tự) và vì thế nó có thể đưa ra các kết quả sai. Hơn nữa, nó không có cách gì để báo cáo các điều kiện lỗi.

PQescapeString có thể được sử dụng an toàn trong các chương trình máy trạm mà làm việc chỉ với một kết nối PostgreSQL tại một thời điểm (trong trường hợp này có thể tìm ra những gì nó cần phải biết "đằng sau sân khấu"). Trong các ngữ cảnh khác thì đây là sự may rủi về an toàn và nên phải tránh được có lợi đối với PqescapeStringConn.

PQescapeByteaConn

Thoát ra các dữ liệu nhị phân để sử dụng bên trong một lệnh SQL với dạng bytea. Như với PQescapeStringConn, điều này chỉ có thể được sử dụng khi việc chèn dữ liệu trực tiếp vào một chuỗi lệnh SQL.

```
unsigned char *PQescapeByteaConn(PGconn *conn,
                                 const unsigned char *from,
                                 size_t from_length,
                                 size_t *to_length);
```

Các giá trị byte nhất định phải được thoát ra khi được sử dụng như một phần của một hằng bytea trong một lệnh SQL.

Các byte thoát PQescapeByteaConn sử dụng hoặc việc mã hóa theo hex hoặc việc thoát chéo ngược. Xem Phần 8.4 để có thêm thông tin.

Tham số from chỉ tới byte đầu tiên của chuỗi sẽ được thoát ra, và tham số from_length trao số byte trong chuỗi nhị phân này. (Một byte zero kết thúc vừa không cần thiết vừa không được tính). Tham số to_length trỏ tới một biến sẽ giữ độ dài chuỗi kết quả được thoát ra. Độ dài chuỗi kết quả này bao gồm byte zero kết thúc của kết quả đó.

PQescapeByteaConn trả về một phiên bản được thoát ra của chuỗi nhị phân tham số from

trong bộ nhớ được phân bổ với malloc(). Bộ nhớ này sẽ được giải phóng bằng việc sử dụng PQfreemem() khi kết quả không còn cần thiết nữa. Chuỗi trả về có tất cả các ký tự đặc biệt được thay thế sao cho chúng có thể được trình phân tích cú pháp hằng chuỗi PostgreSQL và hàm đầu vào bytea xử lý.

Một byte zero kết thúc cũng được thêm vào. Các dấu nháy đơn phải có xung quanh các hằng chuỗi PostgreSQL không phải là một phần của chuỗi kết quả đó.

Nếu có lỗi, một con trỏ null được trả về, và một thông điệp lỗi phù hợp được lưu trữ trong đối tượng conn.

Hiện hành, lỗi có khả năng duy nhất là không đủ bộ nhớ cho chuỗi kết quả.

PQescapeBytea

PQescapeBytea là một phiên bản cũ hơn, không được tán thành của PqescapeByteaConn.

```
unsigned char *PQescapeBytea(const unsigned char *from,
                             size_t from_length,
                             size_t *to_length);
```

Sự khác biệt duy nhất với PQescapeByteaConn là PQescapeBytea không lấy một tham số PGconn. Vì điều này, PQescapeBytea chỉ có thể được sử dụng an toàn trong các chương trình máy trạm mà sử dụng một kết nối PostgreSQL duy nhất tại một thời điểm (trong trường hợp nó có thể tìm ra những gì nó cần phải biết "đằng sau sân khấu"). Nó có thể đưa ra các kết quả sai nếu được sử dụng trong các chương trình mà sử dụng nhiều kết nối cơ sở dữ liệu (hãy sử dụng PQescapeByteaConn trong các trường hợp như vậy).

PQunescapeBytea

Chuyển đổi một trình diễn chuỗi dữ liệu nhị phân thành dữ liệu nhị phân - ngược lại của PQescapeBytea. Điều này là cần thiết khi truy xuất dữ liệu bytea ở định dạng văn bản, nhưng không khi truy xuất nó ở định dạng nhị phân.

```
unsigned char *PQunescapeBytea(const unsigned char *from, size_t *to_length);
```

Tham số from trỏ tới một chuỗi như có thể được PQgetvalue trả về khi được áp dụng cho một cột bytea. PQunescapeBytea biến đổi trình diễn chuỗi này thành trình diễn nhị phân của nó. Nó trả về một con trỏ tới một bộ nhớ tạm được phân bổ với malloc(), hoặc NULL nếu có lỗi, và đặt kích cỡ của bộ nhớ tạm đó vào trong to_length. Kết quả phải được giải phóng bằng việc sử dụng PQfreemem khi nó không còn cần thiết nữa.

Sự chuyển đổi này không chính xác là ngược lại với PQescapeBytea, vì chuỗi đó không được kỳ vọng sẽ "được thoát" khi được nhận từ PQgetvalue. Đặc biệt điều này có nghĩa là không cần đối với chuỗi trích các cân nhắc, và vì thế không cần đối với một tham số Pgconn.

31.4. Xử lý lệnh không đồng bộ

Hàm PQexec là phù hợp cho việc đệ trình các lệnh trong các ứng dụng thông thường, đồng bộ. Tuy nhiên, nó có một vài sự thiếu hụt mà có thể là quan trọng đối với một vài người sử dụng:

- PQexec chờ lệnh được hoàn tất. Ứng dụng đó có thể có công việc khác phải làm (như việc duy trì một giao diện người sử dụng), trong trường hợp đó nó sẽ không muốn khóa với chờ đợi câu trả lời.

- Vì sự thực thi ứng dụng máy trạm bị treo trong khi nó chờ kết quả, khó cho ứng dụng để quyết định là nó có muốn cố hoãn lệnh đang diễn ra hay không. (Nó có thể được thực hiện từ một trình điều khiển tín hiệu, nhưng không thể khác).
- PQexec có thể chỉ trả về một cấu trúc PGresult. Nếu chuỗi lệnh được đệ trình có chứa nhiều lệnh SQL, thì tất cả ngoại trừ PGresult mới nhất sẽ bị PQexec bỏ qua.

Các ứng dụng mà không thích những hạn chế đó có thể thay vào đó sử dụng các hàm nằm bên dưới mà PQexec được xây dựng từ đó: PQsendQuery và PQgetResult. Cũng có PQsendQueryParams, PQsendPrepare, PQsendQueryPrepared, PQsendDescribePrepared, và PQsendDescribePortal, chúng có thể được sử dụng với PQgetResult để đúp bản chức năng của PQexecParams, PQprepare, PQexecPrepared, PQdescribePrepared, và PQdescribePortal, một cách tương ứng.

PQsendQuery

Đệ trình một lệnh tới máy chủ mà không chờ (các) kết quả. 1 được trả về nếu lệnh đó được gửi đi thành công và 0 nếu không (trong trường hợp đó, hãy sử dụng PQerrorMessage để có nhiều thông tin hơn về sự thất bại).

```
int PQsendQuery(PGconn *conn, const char *command);
```

Sau việc gọi thành công PQsendQuery, hãy gọi PQgetResult một hoặc vài lần để có được các kết quả. PQsendQuery không thể bị gọi một lần nữa (trong cùng kết nối) cho tới khi PQgetResult đã trả về một con trỏ null, chỉ ra rằng lệnh đó đã được thực hiện.

PQsendQueryParams

Đệ trình một lệnh và các tham số tách bạch nhau cho máy chủ mà không chờ (các) kết quả.

```
int PQsendQueryParams(PGconn *conn,
                      const char *command,
                      int nParams,
                      const Oid *paramTypes,
                      const char * const *paramValues,
                      const int *paramLengths,
                      const int *paramFormats,
                      int resultFormat);
```

Điều này tương đương với PQsendQuery ngoại trừ là các tham số của truy vấn có thể được chỉ định riêng rẽ từ chuỗi truy vấn đó. Các tham số hàm sẽ được điều khiển hệt như với PQexecParams. Giống như PQexecParams, nó sẽ không làm việc trong các kết nối giao thức 2.0, và nó cho phép một lệnh trong chuỗi truy vấn.

PQsendPrepare

Gửi yêu cầu để tạo lệnh được chuẩn bị với các tham số được đưa ra, không chờ kết thúc.

```
int PQsendPrepare(PGconn *conn,
                  const char *stmtName,
                  const char *query,
                  int nParams,
                  const Oid *paramTypes);
```

Đây là phiên bản không đồng bộ của PQprepare: nó trả về 1 nếu nó từng có khả năng gửi đi yêu cầu, và 0 nếu không. Sau một lời gọi thành công, hãy gọi PQgetResult để xác định liệu máy chủ có thành công tạo ra lệnh được chuẩn bị hay không. Các tham số hàm được điều

khiển hệt như với PQprepare. Giống như PQprepare, nó sẽ không làm việc trong các kết nối giao thức 2.0.

PQsendQueryPrepared

Gửi yêu cầu để thực thi một lệnh được chuẩn bị với các tham số được đưa ra, không chờ (các) kết quả.

```
int PQsendQueryPrepared(PGconn *conn,
                        const char *stmtName,
                        int nParams,
                        const char * const *paramValues,
                        const int *paramLengths,
                        const int *paramFormats,
                        int resultFormat);
```

Điều này tương tự như PQsendQueryParams, nhưng lệnh sẽ được thực thi được chỉ định bằng việc đặt tên cho một lệnh được chuẩn bị trước đó, thay vì trao cho một chuỗi truy vấn. Các tham số hàm sẽ được điều khiển hết như với PQexecPrepared. Giống như PQexecPrepared, nó sẽ không làm việc trong các kết nối giao thức 2.0.

PQsendDescribePrepared

Đệ trình yêu cầu để có thông tin về lệnh được chuẩn bị được chỉ định, không chờ hoàn tất.

```
int PQsendDescribePrepared(PGconn *conn, const char *stmtName);
```

Đây là phiên bản không đồng bộ của PQdescribePrepared: nó trả về 1 nếu nó từng có khả năng gửi đi yêu cầu, và 0 nếu không. Sau một lời gọi thành công, hãy gọi PQgetResult để có các kết quả. Các tham số hàm sẽ được điều khiển hệt như với PQdescribePrepared. Giống như PQdescribePrepared, nó sẽ không làm việc trong các kết nối giao thức 2.0.

PQsendDescribePortal

Đệ trình yêu cầu để có thông tin về cổng được chỉ định, không chờ kết thúc.

```
int PQsendDescribePortal(PGconn *conn, const char *portalName);
```

Đây là phiên bản không đồng bộ của PQdescribePortal: nó trả về 1 nếu nó từng có khả năng gửi đi yêu cầu, và 0 nếu không. Sau một lời gọi thành công, hãy gọi PQgetResult để có được các kết quả. Các tham số hàm được điều khiển hệt như với PQdescribePortal. Giống như PQdescribePortal, nó sẽ không làm việc trong các kết nối giao thức 2.0.

PQgetResult

Chờ kết quả tiếp sau từ một lời gọi trước đó của PQsendQuery, PQsendQueryParams, PQsendPrepare, hoặc PQsendQueryPrepared, và trả về nó. Một con trỏ null được trả về khi lệnh hoàn tất và sẽ không có thêm kết quả nữa.

```
PGresult *PQgetResult(PGconn *conn);
```

PQgetResult phải được gọi lặp đi lặp lại cho tới khi nó trả về một con trỏ null, chỉ ra rằng lệnh được thực hiện xong. (nếu được gọi khi không lệnh nào tích cực, thì PQgetResult sẽ chỉ trả về hoàn toàn một con trỏ null). Từng kết quả khác null từ PQgetResult sẽ được xử lý bằng việc sử dụng các hàm bổ trợ PGresult y hệt được mô tả trước đó. Đừng quên giải phóng từng đối tượng kết quả bằng PQclear khi làm xong với nó. Lưu ý rằng PQgetResult sẽ chỉ khóa nếu một lệnh là tích cực và dữ liệu trả lời cần thiết còn chưa được PQconsumeInput đọc.

Bằng việc sử dụng PQsendQuery và PQgetResult giải quyết một trong những vấn đề của PQexec: Nếu một chuỗi lệnh có chứa nhiều lệnh SQL, thì các kết quả của các lệnh đó có thể có được một cách riêng rẽ. (Điều này cho phép một mẫu đơn giản của việc xử lý chồng chéo, bằng cách: máy trạm có thể điều khiển các kết quả của một lệnh trong khi máy chủ vẫn còn đang làm việc trong các truy vấn sau đó trong cùng một chuỗi lệnh y hệt). Tuy nhiên, việc gọi PQgetResult sẽ vẫn gây ra cho máy trạm phải khóa cho tới khi máy chủ hoàn tất lệnh SQL tiếp theo. Điều này có thể tránh được bằng việc sử dụng đúng phù hợp thêm 2 hàm nữa:

PQconsumeInput

Nếu đầu vào là sẵn sàng từ máy chủ, hãy dùng nó.

```
int PQconsumeInput(PGconn *conn);
```

PQconsumeInput thường trả về 1 chỉ "không có lỗi", nhưng trả về 0 nếu có vài dạng vấn đề (trong trường hợp đó PQerrorMessage có thể được tư vấn). Lưu ý rằng kết quả không nói lên liệu có bất kỳ dữ liệu đầu vào nào thực sự được thu thập. Sau việc gọi PQconsumeInput, ứng dụng có thể kiểm tra PQisBusy và/hoặc PQnotifies để xem liệu tình trạng của chúng đã thay đổi hay chưa.

PQconsumeInput có thể được gọi thậm chí nếu ứng dụng đó không được chuẩn bị để làm việc với một kết quả hoặc thông báo. Hàm đó sẽ đọc các dữ liệu có sẵn và lưu nó vào một bộ nhớ tạm, vì thế gây ra một chỉ số chỉ đọc select() sẽ đi ra. Ứng dụng đó có thể vì thế sử dụng PQconsumeInput để làm sạch điều kiện select() ngay lập tức, và sau đó xem xét các kết quả khi nhàn rỗi.

PQisBusy

Trả về 1 nếu một lệnh đang bận, đó là, PQgetResult có thể khóa chờ đầu vào. Nếu trả về là 0 chỉ rằng PQgetResult có thể được gọi với đảm bảo không khóa.

```
int PQisBusy(PGconn *conn);
```

PQisBusy bản thân nó sẽ cố đọc dữ liệu từ máy chủ; vì thế PQconsumeInput phải được triệu gọi trước, hoặc tình trạng bận rộn sẽ không bao giờ kết thúc.

Một ứng dụng điển hình sử dụng các hàm đó sẽ có một vòng lặp chính sử dụng select() hoặc poll() để chờ tất cả các điều kiện nó phải trả lời tới. Một trong những điều kiện sẽ là đầu vào sẵn sàng từ máy chủ, mà về select() có nghĩa là các dữ liệu đọc được trong trình mô tả tệp được PQsocket nhận diện. Khi vòng lặp chính dò tìm ra đầu vào sẵn sàng, nó sẽ gọi PQconsumeInput để đọc đầu vào đó. Nó có thể sau đó gọi PQisBusy, tiếp tới là PQgetResult nếu PQisBusy trả về sai (0). Nó cũng có thể gọi PQnotifies để dò tìm các thông điệp NOTIFY (xem Phần 31.7).

Một máy trạm sử dụng PQsendQuery/PQgetResult cũng có thể cố hoãn một lệnh mà vẫn còn đang được máy chủ xử lý; xem Phần 31.5. Nhưng bất chấp giá trị trả về của PQcancel, ứng dụng đó phải tiếp tục với sự tuần tự đọc kết quả bình thường bằng việc sử dụng PQgetResult. Một sự hoãn thành công sẽ đơn giản làm cho lệnh kết thúc sớm hơn so với nó có thể có nếu khác đi.

Bằng việc sử dụng các hàm được mô tả ở trên, có khả năng để tránh khóa trong khi chờ đầu vào từ máy chủ cơ sở dữ liệu. Tuy nhiên, vẫn có khả năng là ứng dụng sẽ khóa việc chờ để gửi đầu ra tới máy chủ. Điều này khá là không phổ biến nhưng có thể xảy ra nếu các lệnh SQL hoặc các giá trị dữ

liệu rất dài sẽ được gửi. (Tuy nhiên, nhiều khả năng hơn nếu ứng dụng đó gửi các dữ liệu thông qua COPY IN). Để tránh khả năng này và đạt được hoạt động cơ sở dữ liệu không khóa hoàn toàn, các hàm sau đây có thể được sử dụng.

Pqsetnonblocking

Thiết lập tình trạng không khóa của kết nối.

```
int PQsetnonblocking(PGconn *conn, int arg);
```

Thiết lập tình trạng kết nối không khóa nếu arg là 1, hoặc có khóa nếu arg là 0. Trả về 0 nếu OK, -1 nếu có lỗi.

Trong tình trạng không khóa, các lời gọi tới PQsendQuery, PQputline, PQputnbytes, và PQendcopy sẽ không khóa nhưng thay vào đó trả về một lỗi nếu chúng cần phải được gọi một lần nữa.

Lưu ý rằng PQexec không tôn trọng chế độ không khóa; nếu nó bị hoãn, nó sẽ hành động theo cách khóa bằng mọi cách.

PQisnonblocking

Trả về tình trạng khóa của kết nối cơ sở dữ liệu.

```
int PQisnonblocking(const PGconn *conn);
```

Trả về 1 nếu kết nối được thiết lập về chế độ không khóa và 0 nếu có khóa.

PQflush

Các cố gắng để xả bất kỳ dữ liệu đầu ra được xếp hàng nào tới máy chủ. Trả về 0 nếu thành công (hoặc nếu hàng đợi gửi là rỗng), -1 nếu hỏng vì vài lý do, hoặc 1 nếu không có khả năng gửi tất cả các dữ liệu trong hàng đợi gửi (trường hợp này chỉ có thể xảy ra nếu kết nối không khóa).

```
int PQflush(PGconn *conn);
```

Sau khi gửi bất kỳ lệnh hoặc dữ liệu nào trong một kết nối không khóa, hãy gọi PQflush. Nếu nó trả về 1, hãy chờ socket sẽ được ghi rồi và hãy gọi nó một lần nữa; lặp lại cho tới khi nó trả về 0. Một khi PQflush trả về 0, hãy chờ socket sẽ được đọc rồi và sau đó đọc câu trả lời như được mô tả ở trên.

31.5. Hoãn các truy vấn đang diễn ra

Một ứng dụng máy trạm có thể yêu cầu hoãn một lệnh vẫn còn đang được máy chủ xử lý, bằng việc sử dụng các hàm được mô tả trong phần này.

PQgetCancel

Tạo một cấu trúc dữ liệu chứa thông tin cần thiết để hoãn một lệnh được phát hành qua một kết nối cơ sở dữ liệu đặc biệt.

```
PGcancel *PQgetCancel(PGconn *conn);
```

PQgetCancel tạo một đối tượng PGcancel đưa ra một đối tượng kết nối PGconn. Nó sẽ trả về NULL nếu conn được đưa ra là NULL hoặc một kết nối không hợp lệ. Đối tượng PGcancel là một cấu trúc mù mờ không có nghĩa sẽ được ứng dụng đó truy cập trực tiếp; nó chỉ có thể được truyền tới PQcancel hoặc PQfreeCancel.

PQfreeCancel

Giải phóng một cấu trúc dữ liệu được PQgetCancel tạo ra.

```
void PQfreeCancel(PGcancel *cancel);
```

PQfreeCancel giải phóng một đối tượng dữ liệu trước đó được PQgetCancel tạo ra.

PQcancel

Yêu cầu máy chủ bỏ xử lý lệnh hiện hành.

```
int PQcancel(PGcancel *cancel, char *errbuf, int errbufsize);
```

Trả về giá trị là 1 nếu yêu cầu hủy được phát đi thành công và 0 nếu không. Nếu không, thì errbuf được điền đầy bằng một thông điệp lỗi giải thích vì sao không, errbuf phải là một mảng char có kích cỡ errbufsize (kích cỡ được khuyến cáo là 256 byte).

Tuy nhiên, sự gửi đi không đảm bảo yêu cầu đó sẽ có bất kỳ hiệu ứng nào. Nếu sự hoãn có hiệu lực, thì lệnh hiện hành sẽ kết thúc sớm và trả về một kết quả lỗi. Nếu sự hoãn hỏng (ví dụ, vì máy chủ đã xử lý lệnh đó rồi), thì sẽ không có kết quả nào nhìn thấy được cả.

PQcancel có thể được triệu gọi an toàn từ một bộ điều khiển tín hiệu, nếu errbuf là một biến cục bộ trong bộ điều khiển tín hiệu đó. Đối tượng PGcancel là chỉ đọc cho tới khi PQcancel được quan tâm, nên nó có thể cũng được triệu gọi từ một luồng riêng rẽ với một đối tượng PGconn đang điều khiển.

PQrequestCancel

Yêu cầu rằng máy chủ bỏ xử lý lệnh hiện hành.

```
int PQrequestCancel(PGconn *conn);
```

PQrequestCancel là phương án bị phản đối của PQcancel. Nó vận hành trực tiếp trong đối tượng PGconn, và trong trường hợp hỏng sẽ lưu trữ thông điệp lỗi trong đối tượng PGconn (nơi nó có thể được PQerrorMessage truy xuất). Dù chức năng đó là y hệt, thì tiếp cận này tạo ra sự may rủi đối với các chương trình nhiều luồng và các bộ điều khiển tín hiệu, vì nó có khả năng ghi đè thông điệp lỗi của PGconn sẽ gây lộn xộn cho hoạt động hiện đang diễn ra trong kết nối đó.

31.6. Giao diện đường dẫn nhanh

PostgreSQL cung cấp một giao diện dường dẫn nhanh để gửi các lời gọi hàm đơn giản tới máy chủ.

Mẹo: Giao diện này là thứ gì đó lỗi thời, như người ta có thể đạt được hiệu năng tương tự và chức năng lớn hơn bằng việc thiết lập một lệnh được chuẩn bị để xác định lời gọi hàm. Sau đó, việc thực thi lệnh với sự truyền các tham số nhị phân và các kết quả thay thế cho một lời gọi hàm đường dẫn nhanh.

Hàm PQfn yêu cầu thực thi hàm của một máy chủ thông qua giao diện đường dẫn nhanh:

```
PGresult *PQfn(PGconn *conn,
               int fnid,
               int *result_buf,
               int *result_len,
               int result_is_int,
               const PQArgBlock *args,
               int nargs);
typedef struct
```

```
{
        int len;
        int isint;
        union
        {
                int *ptr;
                int integer;
        } u;
} PQArgBlock;
```

Đối số fnid là OID của hàm sẽ được thực thi. args và nargs xác định các tham số sẽ được truyền tới hàm đó; chúng phải khớp với danh sách đối số hàm được khai báo. Khi trường isint của cấu trúc một tham số là đúng, thì giá trị u.integer được gửi tới máy chủ như một số nguyên có độ dài được chỉ ra (điều này phải là 1, 2 hoặc 4 byte); việc hoán đổi byte đúng phù hợp xảy ra. Khi isint là sai, số các byte được chỉ ra ở *u.ptr sẽ được gửi mà không có việc xử lý; dữ liệu phải là ở định dạng được máy chủ kỳ vọng cho sự truyền nhị phân của dạng dữ liệu đối số của hàm đó. result_buf là bộ nhớ tạm trong đó đặt giá trị trả về. Bộ gọi phải được phân bổ không gian đủ để lưu trữ giá trị trả về. (Không có sự kiểm tra!) Độ dài kết quả thực tế sẽ được trả về theo số nguyên được result_len chỉ tới. Nếu một kết quả số nguyên 1, 2 hoặc 4 byte được kỳ vọng, hãy thiết lập result_is_int về 1, nếu không thì hãy thiết lập nó về 0. Việc thiết lập result_is_int về 1 làm cho libpq hoán đổi byte giá trị nếu cần thiết, sao cho nó được phân phối như một giá trị int đúng phù hợp cho máy trạm đó. Khi result_is_int là 0, chuỗi byte định dạng nhị phân được máy chủ gửi đi sẽ được trả về không bị sửa đổi.

PQfn luôn trả về một con trỏ hợp lệ PGresult. Tình trạng kết quả sẽ được kiểm tra trước khi kết quả đó được sử dụng. Bộ gọi có trách nhiệm cho việc giải phóng PGresult với PQclear khi nó không còn cần thiết nữa.

Lưu ý rằng không có khả năng để điều khiển các đối số null, các kết quả null, các kết quả được thiết lập giá trị null khi sử dụng giao diện này.

31.7. Thông báo không đồng bộ

PostgreSQL đưa ra thông báo không đồng bộ qua các lệnh LISTEN và NOTIFY. Một phiên làm việc của máy trạm đăng ký lợi ích của nó trong một kênh thông báo đặc biệt với lệnh LISTEN (và có thể dừng nghe với lệnh UNLISTEN). Tất cả các phiên làm việc nghe trong một kênh sẽ được thông báo không đồng bộ khi một lệnh NOTIFY với tên kênh đó được thực thi với bất kỳ phiên làm việc nào. Một chuỗi "trọng tải" có thể được truyền để giao tiếp với các dữ liệu bổ sung cho các người nghe.

Các ứng dụng libpq đệ trình các lệnh LISTEN, UNLISTEN, và NOTIFY như các lệnh SQL thông thường. Sự tới của các thông điệp NOTIFY có thể sau đó được việc gọi PQnotifies dò tìm ra.

Hàm PQnotifies trả về thông báo tiếp sau từ một danh sách các thông điệp thông báo không được điều khiển nhận được từ máy chủ đó. Nó trả về một con trỏ null nếu không có các thông báo treo. Một khi một thông báo được trả về từ PQnotifies, thì nó được coi là được điều khiển và sẽ bị loại bỏ khỏi danh sách các thông báo.

```
PGnotify *PQnotifies(PGconn *conn);
typedef struct pgNotify
{
```

```
        char *relname;          /* notification channel name */
        int be_pid;             /* process ID of notifying server process */
        char *extra;            /* notification payload string */
} PGnotify;
```

Sau việc xử lý một đối tượng PGnotify được PQnotifies trả về, hãy chắc chắn phải giải phóng nó bằng PQfreemem. Là đủ để giải phóng con trỏ PGnotify; các trường relname và extra không đại diện cho các phân bổ riêng rẽ. (Các tên các trường đó là theo lịch sử; đặc biệt, các tên kênh không cần có điều gì phải làm với các tên liên quan).

Ví dụ 31-2 đưa ra một chương trình mẫu minh họa sử dụng thông báo không đồng bộ.

PQnotifies không thực sự đọc dữ liệu từ máy chủ; nó chỉ trả về các thông điệp bị hàm libpq khác bỏ qua trước đó. Trong các phiên bản trước của libpq, cách duy nhất để đảm bảo nhận đúng lúc các thông điệp NOTIFY từng là luôn đệ trình các lệnh, thậm chí các lệnh rỗng, và sau đó kiểm tra PQnotifies sau từng PQexec. Trong khi điều này vẫn còn làm việc, thì nó bị phản đối như một sự phí sức mạnh xử lý.

Cách tốt hơn để kiểm tra các thông điệp NOTIFY khi bạn không có các lệnh hữu dụng để thực thi là hãy gọi PQconsumeInput, sau đó kiểm tra PQnotifies. Bạn có thể sử dụng select() để chờ các dữ liệu tới từ máy chủ, vì thế không sử dụng sức mạnh của CPU trừ phi có thứ gì đó phải làm. (Xem PQsocket để có được số bộ mô tả tệp để sử dụng với select()). Lưu ý rằng điều này sẽ làm việc OK dù bạn đệ trình các lệnh với PQsendQuery /PQgetResult hoặc đơn giản sử dụng PQexec. Tuy nhiên, bạn sẽ nhớ phải kiểm tra PQnotifies sau từng PQgetResult hoặc PQexec, để xem liệu bất kỳ thông báo nào tới trong khi xử lý lệnh đó hay không.

31.8. Các hàm có liên quan tới lệnh COPY

Lệnh COPY trong PostgreSQL có các lựa chọn để đọc từ hoặc ghi tới kết nối mạng được libpq sử dụng. Các hàm được mô tả trong phần này cho phép các ứng dụng tận dụng ưu thế khả năng này bằng việc cung cấp hoặc tiêu thụ các dữ liệu được sao chép.

Toàn bộ qui trình là ứng dụng trước hết đưa ra lệnh SQL COPY qua PQexec hoặc một trong các hàm tương đương. Câu trả lời cho điều này (nếu không có lỗi trong lệnh) sẽ là một đối tượng Pgresult mang một mã tình trạng của PGRES_COPY_OUT hoặc PGRES_COPY_IN (phụ thuộc vào đường lối sao chép được chỉ định). Ứng dụng đó sau đó sẽ sử dụng các hàm của phần này để nhận hoặc truyền các hàng dữ liệu. Khi truyền dữ liệu kết thúc, một đối tượng PGresult khác được trả về để chỉ sự thành công hoặc thất bại của sự truyền. Tình trạng của nó sẽ là PGRES_COMMAND_OK nếu thành công hoặc PGRES_FATAL_ERROR nếu có vấn đề xảy ra. Tại thời điểm này các lệnh SQL tiếp sau có thể được đưa ra thông qua PQexec. (Không có khả năng để thực thi các lệnh SQL khác bằng việc sử dụng kết nối y hệt khi hoạt động COPY đang diễn ra).

Nếu một lệnh COPY được đưa ra qua PQexec trong một chuỗi mà có thể chứa các lệnh bổ sung, thì ứng dụng đó phải tiếp tục lấy các kết quả qua PQgetResult sau khi hoàn tất tuần tự COPY.

Chỉ khi PQgetResult trả về NULL thì chắc chắn rằng chuỗi lệnh PQexec được thực hiện xong và là an toàn để đưa ra nhiều lệnh hơn nữa.

Các hàm của phần này sẽ chỉ được thực thi sau khi có được một tình trạng kết quả của PGRES_COPY_OUT hoặc PGRES_COPY_IN từ PQexec hoặc PQgetResult.

Một đối tượng PGresult mang một trong các giá trị tình trạng đó mang theo vài dữ liệu bổ sung về hoạt động COPY mà nó đang khởi tạo. Dữ liệu bổ sung này là sẵn sàng sử dụng các hàm mà cũng được sử dụng trong kết nối với các kết quả truy vấn:

PQnfields

Trả về số các cột (trường) sẽ được sao chép.

PQbinaryTuples

0 chỉ toàn bộ định dạng sao chép là văn bản (các hàng phân cách bởi các dòng mới, các cột phân cách bởi các ký tự phân cách, ...). 1 chỉ toàn bộ định dạng sao chép là nhị phân. Xem COPY để có thêm thông tin.

PQfformat

Trả về mã định dạng (0 cho văn bản, 1 cho nhị phân) có liên quan tới từng cột của hoạt động sao chép. Các mã định dạng theo từng cột sẽ luôn là zero khi toàn bộ định dạng sao chép là văn bản, nhưng định dạng nhị phân có thể hỗ trợ cả các cột văn bản và nhị phân. (Tuy nhiên, như đối với triển khai hiện hành của COPY, chỉ các cột nhị phân xuất hiện trong một sao chép nhị phân; vì thế các định dạng theo từng cột luôn khớp với tổng thể định dạng hiện hành).

Lưu ý: Các giá trị dữ liệu bổ sung chỉ sẵn sàng khi sử dụng giao thức 3.0. Khi sử dụng giao thức 2.0, tất cả các hàm đó sẽ trả về 0.

31.8.1. Các hàm gửi dữ liệu COPY

Các hàm đó được sử dụng để gửi các dữ liệu trong quá trình COPY FROM STDIN. Chúng sẽ hỏng nếu được gọi khi kết nối không trong tình trạng COPY_IN.

PQputCopyData

Gửi dữ liệu tới máy chủ trong tình trạng COPY_IN.

```
int PQputCopyData(PGconn *conn,
                  const char *buffer,
                  int nbytes);
```

Truyền dữ liệu COPY vào bộ nhớ tạm được chỉ định buffer, độ dài nbytes, tới máy chủ. Kết quả là 1 nếu dữ liệu đã được gửi đi, zero nếu nó không được gửi đi vì cố gắng có thể khóa (trường hợp này chỉ có khả năng nếu kết nối là trong chế độ không khóa), hoặc -1 nếu một lỗi xảy ra. (Hãy sử dụng PQerrorMessage để truy xuất các chi tiết nếu giá trị trả về là -1. Nếu giá trị là zero, hãy chờ ghi xong và cố lần nữa).

Ứng dụng đó có thể chia dòng dữ liệu COPY thành các tải bộ nhớ tạm của bất kỳ kích cỡ thuận tiện nào. Các đường biên tải bộ nhớ tạm không quan trọng về ngữ nghĩa khi gửi. Các nội dung của dòng dữ liệu phải khớp định dạng dữ liệu được lệnh COPY kỳ vọng; xem COPY để có các chi tiết.

PQputCopyEnd

Gửi dấu hiệu hết dữ liệu tới máy chủ trong tình trạng COPY_IN.

```
int PQputCopyEnd(PGconn *conn,
                 const char *errormsg);
```

Kết thúc hoạt động COPY_IN thành công nếu errormsg là NULL. Nếu errormsg khác NULL thì COPY bị ép hỏng, với chuỗi được trỏ tới bằng errormsg được sử dụng như thông điệp lỗi. (Tuy nhiên, người ta sẽ không giả thiết thông điệp lỗi chính xác này sẽ quay lại từ máy chủ, khi máy chủ có thể đã hỏng COPY rồi vì các lý do của riêng nó. Hơn nữa lưu ý rằng lựa chọn ép hỏng không làm việc khi sử dụng các kết nối giao thức trước 3.0).

Kết quả là 1 nếu dữ liệu kết thúc đã được gửi đi, zero nếu nó không được gửi đi vì cố gắng có thể khóa (trường hợp này chỉ có khả năng nếu kết nối là trong chế độ không khóa), hoặc -1 nếu một lỗi xảy ra. (Hãy sử dụng PQerrorMessage để truy xuất các chi tiết nếu giá trị trả về là -1. Nếu giá trị đó là zero, hãy chờ cho ghi xong và cố thử lại một lần nữa).

Sau việc gọi thành công PQputCopyEnd, hãy gọi PQgetResult để có được tình trạng kết quả cuối cùng của lệnh COPY. Người ta có thể chờ kết quả này sẽ sẵn sàng theo cách thông thường. Sau đó trả về hoạt động bình thường.

31.8.2. Các hàm truy xuất dữ liệu COPY

Các hàm đó được sử dụng để truy xuất dữ liệu trong quá trình COPY TO STDOUT. Chúng sẽ hỏng nếu được gọi khi kết nối không ở trong tình trạng COPY_OUT.

PQgetCopyData

Nhận dữ liệu từ máy chủ trong tình trạng COPY_OUT.

```
int PQgetCopyData(PGconn *conn,
                  char **buffer,
                  int async);
```

Các cố gắng có được hàng dữ liệu khác từ máy chủ trong quá trình một COPY. Dữ liệu luôn được trả về một hàng dữ liệu ở một thời điểm; nếu chỉ một phần của hàng là sẵn sàng, thì nó sẽ không được trả về. Sự trả về thành công một hàng dữ liệu có liên quan tới việc phân bổ một nhúm bộ nhớ để giữ các dữ liệu đó. Tham số buffer phải là khác NULL. *buffer được thiết lập để trỏ tới bộ nhớ được phân bổ, hoặc tới NULL trong các trường hợp nơi mà không bộ nhớ tạm nào được trả về. Một bộ nhớ tạm kết quả khác NULL sẽ được giải phóng bằng việc sử dụng PQfreemem khi không còn cần thiết nữa.

Khi một hàng được trả về thành công, giá trị trả về là số byte dữ liệu trong hàng đó (điều này sẽ luôn lớn hơn zero). Chuỗi được trả về luôn kết thúc với null, dù điều này có lẽ chỉ hữu dụng cho COPY văn bản. Một kết quả zero chỉ ra rằng COPY vẫn còn đang diễn ra, nhưng không hàng nào là sẵn sàng (điều này chỉ có khả năng khi async là đúng). Một kết quả -1 chỉ ra rằng COPY được thực hiện xong. Một kết quả là -2 chỉ ra rằng một lỗi đã xảy ra (hãy tư vấn PQerrorMessage để biết lý do).

Khi async là đúng (khác zero), thì PQgetCopyData sẽ không khóa mà chờ đầu vào; nó sẽ trả về zero nếu COPY vẫn còn diễn ra nhưng không hàng hoàn chỉnh nào là sẵn sàng cả. (Trong

trường hợp này hãy chờ đọc xong và sau đó gọi PQconsumeInput trước khi gọi PQgetCopyData một lần nữa). Khi async là sai (zero), PQgetCopyData sẽ khóa cho tới khi dữ liệu là sẵn sàng hoặc hoạt động hoàn tất.

Sau khi PQgetCopyData trả về -1, hãy gọi PQgetResult để có tình trạng kết quả cuối cùng của lệnh COPY. Người ta có thể chờ vì kết quả này sẽ sẵn sàng theo cách thông thường. Sau đó trả về hoạt động bình thường.

31.8.3. Các hàm lỗi thời cho COPY

Các hàm đó đại diện cho các phương pháp cũ hơn về điều khiển COPY. Dù chúng vẫn còn làm việc, chúng bị phản đối vì các phương pháp điều khiển lỗi kém, không thuận tiện dò tìm kết thúc của dữ liệu, và thiếu hỗ trợ cho các cuộc truyền nhị phân hoặc không khóa.

PQgetline

Đọc một dòng các ký tự kết thúc bằng một dòng mới (được máy chủ truyền) trong một chuỗi bộ nhớ tạm kích cỡ length.

```
int PQgetline(PGconn *conn,
              char *buffer,
              int length);
```

Hàm này sao chép tới length -1 các ký tự trong bộ nhớ tạm và biến đổi dòng mới ở cuối thành một byte zero. PQgetline trả về EOF ở cuối của đầu vào, 0 nếu toàn bộ dòng đã được đọc, và 1 nếu bộ nhớ tạm đầy nhưng kết thúc dòng mới còn chưa được đọc.

Lưu ý rằng ứng dụng phải kiểm tra để xem nếu một dòng mới gồm 2 ký tự \. , điều chỉ ra rằng máy chủ đã kết thúc việc gửi các kết quả của lệnh COPY. Nếu ứng dụng đó có thể nhận các dòng mà lớn hơn length -1 các ký tự dài, thì sự chăm sóc là cần thiết để chắc chắn nó nhận dòng các ký tự \. một cách chính xác (và không, ví dụ, nhầm kết thúc của một dòng dữ liệu dài đối với một dòng ngắt).

PQgetlineAsync

Đọc một hàng dữ liệu COPY (được máy chủ truyền) trong một bộ nhớ tạm không khóa.

```
int PQgetlineAsync(PGconn *conn,
                   char *buffer,
                   int bufsize);
```

Hàm này là tương tự như PQgetline, nhưng nó có thể được các ứng dụng sử dụng mà phải đọc dữ liệu COPY một cách đồng bộ, đó là, không có khóa. Đã đưa ra lệnh COPY và có một trả lời PGRES_COPY_OUT, ứng dụng đó sẽ gọi PQconsumeInput và PQgetlineAsync cho tới khi tín hiệu kết thúc dữ liệu được tìm ra.

Không giống như PQgetline, hàm này nhận trách nhiệm dò tìm ra kết thúc dữ liệu.

Trong từng lời gọi, PQgetlineAsync sẽ trả về dữ liệu nếu một hàng dữ liệu hoàn chỉnh là sẵn sàng ở bộ nhớ tạm đầu vào của libpq. Nếu không, không dữ liệu nào được trả về cho tới khi phần còn lại của hàng tới. Hàm đó trả về -1 nếu bộ ghi kết thúc sao chép dữ liệu đã được thừa nhận, hoặc 0 nếu không dữ liệu nào sẵn sàng, hoặc một số dương đưa ra số lượng các byte dữ liệu được trả về. Nếu -1 được trả về, thì bộ gọi phải gọi tiếp PQendcopy, và sau đó

trả về cho việc xử lý bình thường.

Dữ liệu được trả về sẽ không mở rộng vượt khỏi ranh giới một hàng dữ liệu. Nếu có thể thì toàn bộ hàng sẽ được trả về tại một thời điểm. Nhưng nếu bộ nhớ tạm được bộ gọi đưa ra là quá nhỏ để giữ một hàng được máy chủ gửi đi, thì một phần hàng dữ liệu sẽ được trả về. Với dữ liệu văn bản thì điều này có thể được dò tìm ra bằng việc kiểm thử liệu byte được trả về cuối cùng có là \n hay không. (Trong một COPY nhị phân, việc phân tích cú pháp thực sự định dạng dữ liệu COPY sẽ là cần thiết để tiến hành xem xét tương đương). Chuỗi được trả về sẽ không kết thúc với null. (Nếu bạn muốn thêm một kết thúc null, hãy chắc chắn phải truyền một bufsize nhỏ hơn so với chỗ thực sự sẵn có).

PQputline

Gửi một chuỗi kết thúc null tới máy chủ. Trả về 0 nếu OK và EOF nếu không có khả năng gửi chuỗi đó.

```
int PQputline(PGconn *conn,
              const char *string);
```

Dòng dữ liệu COPY được một loạt lời gọi tới PQputline gửi đi có định dạng y hệt như định dạng được PQgetlineAsync trả về, ngoại trừ là các ứng dụng không bị buộc phải gửi chính xác một hàng dữ liệu theo từng lời gọi PQputline; là OK để gửi một phần dòng hoặc nhiều dòng theo từng lời gọi.

> **Lưu ý**: Trước giao thức 3.0 của PostgreSQL, từng là cần thiết cho ứng dụng để rõ ràng gửi 2 ký tự \. như một dòng kết để chỉ tới máy chủ rằng nó đã kết thúc gửi dữ liệu COPY. Trong khi điều này vẫn còn làm việc, thì nó không được tán thành và ý nghĩa đặc biệt của \. có thể được kỳ vọng sẽ được loại bỏ trong một phiên bản trong tương lai. Là đủ để gọi PQendcopy sau khi đã gửi dữ liệu thực sự.

PQputnbytes

Gửi một chuỗi có kết thúc khác null tới máy chủ. Trả về 0 là OK và EOF nếu không có khả năng gửi chuỗi đó.

```
int PQputnbytes(PGconn *conn,
                const char *buffer,
                int nbytes);
```

Điều này chính xác như PQputline, ngoại trừ là bộ nhớ tạm dữ liệu cần không được kết thúc null vì số byte để gửi được chỉ định trực tiếp. Sử dụng thủ tục này khi gửi dữ liệu nhị phân.

PQendcopy

Đồng bộ với máy chủ.

```
int PQendcopy(PGconn *conn);
```

Hàm này chờ cho tới khi máy chủ đã kết thúc việc sao chép. Nó sẽ hoặc được đưa ra khi chuỗi cuối cùng đã được gửi tới máy chủ bằng việc sử dụng PQputline hoặc khi chuỗi cuối cùng nhận được từ máy chủ bằng việc sử dụng PGgetline. Nó phải được đưa ra hoặc máy chủ sẽ "không đồng bộ được" (out of sync) với máy trạm. Được trả về từ hàm này, máy chủ sẵn sàng nhận lệnh SQL tiếp sau. Giá trị trả về là 0 khi hoàn tất thành công, nếu không thì sẽ khác zero. (Sử dụng PQerrorMessage để truy xuất các chi tiết nếu giá trị trả về khác zero).

Khi sử dụng PQgetResult, ứng dụng sẽ trả lời cho một kết quả PGRES_COPY_OUT bằng việc thực thi PQgetline lặp đi lặp lại, theo sau là PQendcopy sau khi dòng ngắt được thấy. Nó sau đó sẽ trả về vòng lặp PQgetResult cho tới khi PQgetResult trả về một con trỏ null. Tương tự như một kết quả PGRES_COPY_IN được xử lý bằng một loạt các lời gọi PQputline theo sau là PQendcopy, sau đó trả về vòng lặp PQgetResult. Dàn xếp này sẽ đảm bảo rằng một lệnh COPY được nhúng trong một loạt các lệnh SQL sẽ được thực thi đúng.

Các ứng dụng cũ hơn có khả năng sẽ đệ trình một COPY qua PQexec và giả thiết rằng giao dịch được thực hiện xong sau PQendcopy. Điều này sẽ làm việc đúng chỉ nếu COPY là lệnh SQL duy nhất trong chuỗi lệnh đó.

31.9. Các hàm kiểm soát

Các hàm đó kiểm soát các chi tiết hỗn tạp về hành vi của libpq.

PQclientEncoding

Trả về việc mã hóa máy trạm.

```
int PQclientEncoding(const PGconn *conn);
```

Lưu ý rằng nó trả về ID mã hóa, không phải một chuỗi biểu tượng như EUC_JP. Để biến đổi một ID mã hóa thành một tên mã hóa, bạn có thể sử dụng:

```
char *pg_encoding_to_char(int encoding_id);
```

PQsetClientEncoding

Thiết lập mã hóa máy trạm.

```
int PQsetClientEncoding(PGconn *conn, const char *encoding);
```

conn là một kết nối tới máy chủ, và encoding là mã hóa bạn muốn sử dụng. Nếu hàm thiết lập thành công việc mã hóa, thì nó trả về 0, nếu không là -1. Mã hóa hiện hành cho kết nối này có thể được xác định bằng việc sử dụng PqclientEncoding.

PQsetErrorVerbosity

Xác định độ rườm rà các thông điệp được PQerrorMessage và PQresultErrorMessage trả về.

```
typedef enum
{
        PQERRORS_TERSE,
        PQERRORS_DEFAULT,
        PQERRORS_VERBOSE
} PGVerbosity;
```

```
PGVerbosity PQsetErrorVerbosity(PGconn *conn, PGVerbosity verbosity);
```

PQsetErrorVerbosity thiết lập chế độ rườm rà, trả về thiết lập trước đó của kết nối.

Trong chế độ TERSE, các thông điệp được trả về bao gồm độ nghiêm trọng, văn bản đầu, và chỉ vị trí; điều này sẽ thường khít trong một dòng duy nhất. Chế độ mặc định đưa ra các thông điệp bao gồm thứ ở trên cộng với bất kỳ các trường chi tiết, gợi ý, hoặc ngữ cảnh nào (chúng có thể lan qua nhiều dòng). Chế độ VERBOSE bao gồm tất cả các trường có sẵn. Việc thay đổi độ rườm rà không ảnh hưởng tới các thông điệp có sẵn từ các đối tượng PGresult đang tồn tại rồi, chỉ các đối tượng được tạo ra sau đó.

PQtrace

Kích hoạt việc theo dõi giao tiếp của máy trạm/máy chủ cho việc gỡ lỗi luồng tệp.

```
void PQtrace(PGconn *conn, FILE *stream);
```

Lưu ý: Trong Windows, nếu thư viện libpq và một ứng dụng được biên dịch với các cờ khác nhau, thì lời gọi hàm này sẽ làm hỏng ứng dụng vì trình diễn nội bộ của các con trỏ tệp FILE khác nhau. Đặc biệt, các cờ đa luồng/đơn luồng, phát hành/gỡ lỗi, và tĩnh/động sẽ là y hệt đối với thư viện đó và tất cả các ứng dụng sử dụng thư viện đó.

PQuntrace

Vô hiệu hóa việc theo dõi được PQtrace khởi tạo.

```
void PQuntrace(PGconn *conn);
```

31.10. Các hàm hỗn hợp

Luôn có vài hàm thật không vừa ở bất kỳ chỗ nào.

PQfreemem

Giải phóng bộ nhớ được libpq phân bổ.

```
void PQfreemem(void *ptr);
```

Giải phóng bộ nhớ được libpq phân bổ, đặc biệt là PQescapeByteaConn, PQescapeBytea, PQunescapeBytea, và PQnotifies. Đặc biệt quan trọng là hàm này, hơn là free(), sẽ được sử dụng trong Microsoft Windows. Điều này là vì phân bổ bộ nhớ trong một DLL và đưa nó vào ứng dụng chỉ làm việc nếu các cờ đa luồng/đơn luồng, phát hành/sửa lỗi, và tĩnh/động sẽ là y hệt cho DLL và ứng dụng đó. Trong các nền tảng khác Microsoft Windows, hàm này là y hệt như hàm thư viện tiêu chuẩn free().

PQconninfoFree

Giải phóng các cấu trúc dữ liệu được PQconndefaults hoặc PQconninfoParse phân bổ.

```
void PQconninfoFree(PQconninfoOption *connOptions);
```

Một PQfreemem đơn giản sẽ không làm vì điều này, vì mảng đó gồm các tham chiếu tới các chuỗi trực thuộc.

PQencryptPassword

Chuẩn bị mẫu được mã hóa của một mật khẩu PostgreSQL.

```
char * PQencryptPassword(const char *passwd, const char *user);
```

Hàm này có ý định sẽ được các ứng dụng máy trạm sử dụng mà muốn gửi các lệnh như ALTER USER joe PASSWORD 'pwd'. Là thực tế tốt không gửi mật khẩu rõ ràng bằng văn bản gốc trong một lệnh như vậy, vì nó có thể bị phơi lộ trong các lưu ký lệnh, các hiển thị hoạt động, và hơn thế. Thay vào đó, hãy sử dụng hàm này để biến đổi mật khẩu thành dạng được mã hóa từ trước khi nó được gửi đi. Các đối số là mật khẩu văn bản rõ ràng, và tên người sử dụng SQL cho việc đó. Giá trị trả về là một chuỗi được malloc phân bổ, hoặc NULL nếu tràn bộ nhớ. Bộ gọi có thể giả thiết chuỗi đó không chứa bất kỳ ký tự đặc biệt nào mà có thể đòi

hỏi việc thoát. Hãy sử dụng PQfreemem để giải phóng kết quả khi làm xong với nó.

PQmakeEmptyPGresult

Xây dựng một đối tượng PGresult rỗng với tình trạng được đưa ra.

```
PGresult *PQmakeEmptyPGresult(PGconn *conn, ExecStatusType status);
```

Đây là hàm nội bộ của libpq để phân bổ và khởi tạo một đối tượng PGresult rỗng. Hàm này trả về NULL nếu bộ nhớ không thể được phân bổ. Nó được xuất khẩu vì vài ứng dụng thấy nó hữu dụng để sinh ra bản thân các đối tượng kết quả (đặc biệt các đối tượng với tình trạng lỗi). Nếu conn khác null và status chỉ ra một lỗi, thì thông điệp lỗi hiện hành của kết nối được chỉ định được sao chép vào PGresult. Hơn nữa, nếu conn khác null, thì bất kỳ thủ tục sự kiện nào cũng được đăng ký trong kết nối sẽ được sao chép vào PGresult. (Chúng không có các lời gọi PGEVT_RESULTCREATE, mà xem PQfireResultCreateEvents). Lưu ý rằng PQclear cuối cùng sẽ được gọi trong đối tượng đó, chỉ như với một PGresult được bản thân libpq trả về.

PQfireResultCreateEvents

Đưa ra một sự kiện PGEVT_RESULTCREATE (xem Phần 31.12) đối với từng thủ tục sự kiện được đăng ký trong đối tượng PGresult. Trả về khác zero nếu thành công, zero nếu bất kỳ thủ tục sự kiện nào bị hỏng.

```
int PQfireResultCreateEvents(PGconn *conn, PGresult *res);
```

Đối số conn được truyền qua tới các thủ tục sự kiện nhưng không được sử dụng trực tiếp. Nó có thể là NULL nếu các thủ tục sự kiện đó sẽ không sử dụng nó.

Các thủ tục sự kiện đã nhận được rồi một sự kiện PGEVT_RESULTCREATE hoặc PGEVT_RESULTCOPY cho đối tượng này sẽ không được đưa ra một lần nữa.

Lý do chính mà hàm này là tách bạch với PQmakeEmptyPGResult là vì thường là phù hợp để tạo một PGresult và điền nó bằng dữ liệu trước khi triệu gọi các thủ tục sự kiện.

PQcopyResult

Tạo một bản sao một đối tượng PGresult. Bản sao đó không có kết nối tới nguồn gây ra bất kỳ cách gì và PQclear phải được gọi khi bản sao đó không còn cần thiết nữa. Nếu hàm đó hỏng, thì NULL được trả về.

```
PGresult *PQcopyResult(const PGresult *src, int flags);
```

Điều này không có ý định thực hiện một bản sao chính xác. Kết quả được trả về luôn đặt vào tình trạng PGRES_TUPLES_OK, và không sao chép bất kỳ thông điệp lỗi nào trong nguồn. (Tuy nhiên, nó sao chép chuỗi tình trạng lệnh). Đối số flags xác định những gì nữa được sao chép. Nó là một bitwise OR của vài cờ. PG_COPYRES_ATTRS chỉ định việc sao chép các thuộc tính kết quả nguồn (các định nghĩa cột). PG_COPYRES_TUPLES chỉ định việc sao chép các bộ dữ liệu kết quả nguồn. (Điều này cũng ngụ ý việc sao chép các thuộc tính). PG_COPYRES_NOTICEHOOKS chỉ định việc sao chép các móc nối thông báo kết quả nguồn. PG_COPYRES_EVENTS chỉ định việc sao chép các sự kiện kết quả nguồn. (Nhưng bất kỳ dữ liệu ví dụ nào có liên quan tới nguồn sẽ không được sao chép).

PQsetResultAttrs

Thiết lập các thuộc tính của một đối tượng Pgresult.

```
int PQsetResultAttrs(PGresult *res, int numAttributes, PGresAttDesc *attDescs);
```

attDescs được cung cấp sẽ được sao chép vào kết quả. Nếu con trỏ attDescs là NULL hoặc numAttributes nhỏ hơn 1, thì yêu cầu đó bị bỏ qua và hàm thành công. Nếu res chứa các thuộc tính rồi, thì hàm đó sẽ hỏng. Nếu hàm đó hỏng, thì giá trị trả về là zero. Nếu hàm đó thành công, thì giá trị trả về khác zero.

PQsetvalue

Thiết lập một giá trị trường bộ dữ liệu của một đối tượng PGresult.

```
int PQsetvalue(PGresult *res, int tup_num, int field_num, char *value, int len);
```

Hàm đó sẽ tự động tăng mảng các bộ dữ liệu nội bộ kết quả như cần thiết. Tuy nhiên, đối số tup_num phải ít hơn hoặc bằng PQntuples, nghĩa là hàm này chỉ có thể tăng mảng các bộ dữ liệu một bộ dữ liệu tại một thời điểm. Nhưng bất kỳ trường bộ dữ liệu đang tồn tại nào cũng có thể được sửa đổi theo bất kỳ trật tự nào. Nếu một giá trị ở field_num tồn tại rồi, thì nó sẽ bị ghi đè. Nếu len là -1 hoặc value là NULL, thì giá trị trường đó sẽ được thiết lập về một giá trị null SQL. Giá trị value được sao chép vào kho riêng kết quả, vì thế không còn cần thiết sau khi hàm trả về. Nếu hàm hỏng, giá trị trả về là zero. Nếu hàm thành công, giá trị trả về là khác zero.

PQresultAlloc

Phân bổ kho phụ trợ cho một đối tượng Pgresult.

```
void *PQresultAlloc(PGresult *res, size_t nBytes);
```

Bất kỳ bộ nhớ nào được phân bổ với hàm này cũng sẽ được giải phóng khi res được làm sạch. Nếu hàm đó hỏng, giá trị trả về là NULL. Kết quả được đảm bảo sẽ được sắp đặt đúng phù hợp cho bất kỳ dạng dữ liệu nào, hệt như cho malloc.

31.11. Xử lý thông báo

Các thông điệp thông báo và cảnh báo được máy chủ sinh ra sẽ không được các hàm thực thi truy vấn trả về, vì chúng không ngụ ý sự hỏng truy vấn. Thay vào đó chúng được truyền tới một hàm điều khiển thông báo, và sự thực thi tiếp tục bình thường sau khi trình điều khiển trả về. Hàm điều khiển thông báo mặc định in thông điệp đó trong stderr, nhưng ứng dụng đó có thể ghi đè hành vi này bằng việc cung ứng hàm điều khiển riêng của nó.

Vì các lý do lịch sử, có 2 mức điều khiển thông báo, được gọi là trình nhận thông báo và trình xử lý thông báo. Hành vi mặc định là đối với trình nhận thông báo để định dạng thông báo và truyền một chuỗi tới trình xử lý thông báo để in. Tuy nhiên, một ứng dụng mà chọn cung cấp trình nhận thông báo của riêng mình thường sẽ bỏ qua lớp trình xử lý thông báo và chỉ làm tất cả công việc đó trong trình nhận thông báo đó.

Hàm PQsetNoticeReceiver thiết lập hoặc kiểm tra trình nhận thông báo hiện hành cho một đối tượng kết nối. Tương tự, PQsetNoticeProcessor thiết lập hoặc kiểm tra trình xử lý thông báo hiện hành.

```
typedef void (*PQnoticeReceiver) (void *arg, const PGresult *res);
PQnoticeReceiver
PQsetNoticeReceiver(PGconn *conn,
                    PQnoticeReceiver proc,
                    void *arg);
```

```
typedef void (*PQnoticeProcessor) (void *arg, const char *message);
PQnoticeProcessor
PQsetNoticeProcessor(PGconn *conn,
                     PQnoticeProcessor proc,
                     void *arg);
```

Từng trong số các hàm trả về cho trình nhận thông báo hoặc con trỏ hàm trình xử lý trước đó, và thiết lập giá trị mới. Nếu bạn cung cấp một con trỏ hàm null, thì không hành động nào được thực hiện, mà con trỏ hiện hành được trả về.

Khi một thông điệp thông báo hoặc cảnh báo được nhận từ máy chủ, hoặc được sinh ra trong nội bộ bằng libpq, thì hàm của trình nhận thông báo sẽ được gọi. Nó truyền thông điệp đó ở dạng của một PGRES_NONFATAL_ERROR PGresult. (Điều này cho phép trình nhận thông báo trích xuất các trường riêng rẽ bằng việc sử dụng PQresultErrorField, hoặc thông điệp được định dạng hoàn chỉnh từ trước bằng việc sử dụng PQresultErrorMessage). Con trỏ trống y hệt được truyền tới PQsetNoticeReceiver cũng được truyền qua. (Con trỏ này có thể được sử dụng để truy cập tình trạng đặc thù ứng dụng nếu cần thiết).

Trình nhận thông báo mặc định đơn giản trích thông điệp (bằng việc sử dụng PQresultErrorMessage) và truyền nó tới trình xử lý thông báo.

Trình xử lý thông báo có trách nhiệm cho việc điều khiển một thông điệp thông báo hoặc cảnh báo được đưa ra ở dạng văn bản. Nó truyền văn bản chuỗi thông điệp đó (bao gồm cả dòng mới sau đuôi), cộng với một con trỏ rỗng mà là y hệt con trỏ được truyền tới PQsetNoticeProcessor. (Con trỏ này có thể được sử dụng để truy cập tình trạng đặc thù ứng dụng nếu cần thiết).

Trình xử lý thông báo mặc định là đơn giản:

```
static void
defaultNoticeProcessor(void *arg, const char *message)
{
        fprintf(stderr, "%s", message);
}
```

Một khi bạn đã thiết lập một trình nhận hoặc xử lý thông báo, thì bạn nên kỳ vọng rằng hàm đó có thể được gọi miễn là hoặc đối tượng PGconn hoặc các đối tượng PGresult được làm từ nó tồn tại. Khi tạo một PGresult, các con trỏ điều khiển thông báo hiện hành của PGconn sẽ được sao chép vào PGresult để có thể được các hàm như PQgetvalue sử dụng.

31.12. Hệ thống sự kiện

Hệ thống sự kiện của libpq được thiết kế để thông báo cho các trình điều khiển sự kiện được đăng ký về các sự kiện thú vị của libpq, như tạo hoặc hủy các đối tượng PGconn và PGresult. Một trường hợp điển hình chính là điều này cho phép các ứng dụng liên hệ với các dữ liệu của riêng chúng với một PGconn hoặc PGresult và đảm bảo rằng các dữ liệu đó được giải phóng vào thời điểm phù hợp.

Từng trình điều khiển sự kiện có liên quan tới 2 mẩu dữ liệu, được biết đối với libpq chỉ như các con trỏ mù mờ void *. Có một con trỏ truyền qua mà được ứng dụng cung cấp khi trình điều khiển

sự kiện được đăng ký với một PGconn. Con trỏ truyền qua đó chưa bao giờ thay đổi trong đời Pgconn và tất cả các PGresult được sinh ra từ nó; nên nếu được sử dụng, nó phải trỏ tới dữ liệu đã sống lâu rồi. Hơn nữa có một con trỏ dữ liệu cá biệt, nó khởi tạo ra NULL trong từng PGconn và PGresult. Con trỏ này có thể được điều khiển bằng việc sử dụng các hàm PQinstanceData, PQsetInstanceData, PQresultInstanceData và PQsetResultInstanceData. Lưu ý rằng không giống như con trỏ truyền qua, dữ liệu cá biệt của một PGconn không tự động được kế thừa từ các PGresult được tạo ra từ nó. libpq không biết các con trỏ dữ liệu cá biệt và truyền qua nào trỏ tới (nếu có) và sẽ không bao giờ có ý định giải phóng chúng - đó là trách nhiệm của trình điều khiển sự kiện.

31.12.1. Dạng sự kiện

PGEventId có đánh số đặt tên cho các dạng sự kiện được hệ thống sự kiện điều khiển. Tất cả các giá trị của nó có các tên bắt đầu bằng PGEVT. Đối với từng dạng sự kiện, có một cấu trúc thông tin sự kiện tương ứng mang các tham số được truyền tới các trình điều khiển sự kiện. Các dạng sự kiện là:

PGEVT_REGISTER

Sự kiện đăng ký xảy ra khi PQregisterEventProc được gọi. Là thời điểm lý tưởng để khởi tạo bất kỳ instanceData nào mà một thủ tục sự kiện có thể cần. Chỉ một sự kiện đăng ký sẽ được đưa ra cho từng trình điều khiển sự kiện cho từng kết nối. Nếu thủ tục sự kiện hỏng, thì đăng ký đó bị bỏ qua.

```
typedef struct
{
        PGconn *conn;
} PGEventRegister;
```

Khi một sự kiện PGEVT_REGISTER được nhận, con trỏ evtInfo sẽ được phát tới một PGEventRegister *. Cấu trúc này có một PGconn sẽ có trong tình trạng CONNECTION_OK; được đảm bảo nếu người ta gọi PQregisterEventProc ngay sau khi có được một PGconn tốt. Khi trả về một mã hỏng, tất cả sự làm sạch phải được thực hiện như là không sự kiện PGEVT_CONNDESTROY nào sẽ được gửi đi.

PGEVT_CONNRESET

Sự kiện thiết lập lại kết nối được đưa ra trong sự kết thúc PQreset hoặc PQresetPoll. Trong cả 2 trường hợp, sự kiện đó chỉ được đưa ra nếu sự thiết lập lại là thành công. Nếu thủ tục sự kiện đó thất bại, thì toàn bộ sự thiết lập lại đó sẽ thất bại. PGconn được đặt vào tình trạng CONNECTION_BAD và PQresetPoll sẽ trả về PGRES_POLLING_FAILED.

```
typedef struct
{
        PGconn *conn;
} PGEventConnReset;
```

Khi một sự kiện PGEVT_CONNRESET được nhận, con trỏ evtInfo sẽ được phát tới một PGEventConnReset *. Dù PGconn vừa được thiết lập lại, tất cả dữ liệu sự kiện vẫn không bị thay đổi. Sự kiện này sẽ được sử dụng để thiết lập lại/tải lại/truy vấn lại bất kỳ instanceData

có liên quan.

Lưu ý rằng thậm chí nếu thủ tục sự kiện đó thất bại trong xử lý PGEVT_CONNRESET, thì nó sẽ vẫn nhận được một sự kiện PGEVT_CONNDESTROY khi kết nối đó được đóng lại.

PGEVT_CONNDESTROY

Sự kiện phá hủy kết nối được đưa ra để trả lời cho PQfinish. Đây là trách nhiệm của thủ tục sự kiện để làm sạch đúng phù hợp các dữ liệu sự kiện của nó khi libpq không có khả năng quản lý bộ nhớ này. Không làm sạch được sẽ dẫn tới rò rỉ bộ nhớ.

```
typedef struct
{
        PGconn *conn;
} PGEventConnDestroy;
```

Khi một sự kiện PGEVT_CONNDESTROY được nhận, con trỏ evtInfo sẽ được phát tới một PGEventConnDestroy *. Sự kiện này được thực hiện trước khi PQfinish thực hiện bất kỳ sự làm sạch nào. Giá trị trả về của thủ tục sự kiện đó bị bỏ qua vì không có cách gì chỉ thị một thất bại từ PQfinish. Hơn nữa, một thất bại của thủ tục sự kiện sẽ không bỏ qua tiến trình làm sạch bộ nhớ không mong muốn.

PGEVT_RESULTCREATE

Sự kiện tạo kết quả được đưa ra để trả lời cho bất kỳ hàm thực thi truy vấn nào sinh ra một kết quả, bao gồm cả PQgetResult. Sự kiện này sẽ chỉ được đưa ra sau khi kết quả đã được tạo ra thành công.

```
typedef struct
{
        PGconn *conn;
        PGresult *result;
} PGEventResultCreate;
```

Khi sự kiện PGEVT_RESULTCREATE được nhận, con trỏ evtInfo sẽ được phát tới PGEventResultCreate *. conn là kết nối được sử dụng để sinh ra kết quả đó. Đây là nơi lý tưởng để khởi tạo bất kỳ instanceData nào cần liên quan tới kết quả đó. Nếu thủ tục sự kiện đó hỏng, thì kết quả sẽ bị làm sạch và sự hỏng đó sẽ được nhân rộng. Thủ tục sự kiện đó phải không cố gắng PQclear bản thân đối tượng kết quả đó. Khi trả về một mã hỏng, tất cả sự làm sạch phải được thực hiện như không sự kiện PGEVT_RESULTDESTROY nào sẽ được gửi đi.

PGEVT_RESULTCOPY

Sự kiện sao chép kết quả được đưa ra để trả lời cho PQcopyResult. Sự kiện này sẽ chỉ được đưa ra sau khi sao chép đó hoàn tất. Chỉ các thủ tục sự kiện nào mà đã điều khiển thành công sự kiện PGEVT_RESULTCREATE hoặc PGEVT_RESULTCOPY đối với kết quả nguồn sẽ nhận được các sự kiện PGEVT_RESULTCOPY.

```
typedef struct
{
        const PGresult *src;
        PGresult *dest;
} PGEventResultCopy;
```

Khi một sự kiện PGEVT_RESULTCOPY nhận được, con trỏ evtInfo sẽ được phát tới một PGEventResultCopy *. Kết quả src là những gì đã được sao chép trong khi kết quả dest là đích

sao chép. Sự kiện này có thể được sử dụng để cung cấp một sao chép sâu của instanceData, khi PQcopyResult không thể làm điều đó. Nếu thủ tục sự kiện đó hỏng, thì toàn bộ hoạt động sao chép đó sẽ hỏng và kết quả dest sẽ bị làm sạch. Khi trả về một mã hỏng, tất cả sự làm sạch phải được thực hiện như không sự kiện PGEVT_RESULTDESTROY nào sẽ được gửi cho kết quả đích.

PGEVT_RESULTDESTROY

Sự kiện hủy kết quả được đưa ra để trả lời cho một PQclear. Đây là trách nhiệm của thủ tục sự kiện đó để làm sạch đúng phù hợp dữ liệu sự kiện của nó khi libpq không có khả năng quản lý bộ nhớ này. Không làm sạch được sẽ dẫn tới rò rỉ bộ nhớ.

```
typedef struct
{
        PGresult *result;
} PGEventResultDestroy;
```

Khi một sự kiện PGEVT_RESULTDESTROY được nhận, con trỏ evtInfo sẽ được phát tới một PGEventResultDestroy *. Sự kiện này được đưa ra trước khi PQclear thực thi bất kỳ sự làm sạch nào khác. Giá trị trả về của thủ tục sự kiện được bỏ qua vì không có cách nào chỉ thị sự thất bại từ PQclear. Hơn nữa, một thất bại của thủ tục sự kiện sẽ không từ bỏ tiến trình làm sạch bộ nhớ không mong muốn.

31.12.2. Thủ tục gọi ngược lại sự kiện

PGEventProc

PGEventProc là một định nghĩa dạng typedef cho một con trỏ tới một thủ tục sự kiện, đó là, người sử dụng gọi ngược lại hàm và nhận các sự kiện từ libpq. Chữ ký của một thủ tục sự kiện phải là int eventproc(PGEventId evtId, void *evtInfo, void *passThrough).

Tham số evtId chỉ sự kiện PGEVT nào đã xảy ra. Con trỏ evtInfo phải được phát tới dạng cấu trúc phù hợp để có được thông tin xa hơn về sự kiện đó. Tham số passThrough là con trỏ được PQregisterEventProc cung cấp khi thủ tục sự kiện đó đã được đăng ký. Hàm đó sẽ trả về một giá trị khác zero nếu nó thành công và zero nếu hỏng. Một thủ tục sự kiện đặc biệt có thể được đăng ký chỉ một lần trong bất kỳ PGconn nào. Điều này là vì địa chỉ của thủ tục được sử dụng như một khóa tra cứu để nhận diện dữ liệu đặc biệt có liên quan.

Cảnh báo

Trong Windows, các hàm có thể có 2 địa chỉ khác nhau: một nhìn thấy từ bên ngoài một DLL và địa chỉ kia nhìn thấy từ bên trong DLL đó. Nên cẩn thận là chỉ một trong số các địa chỉ đó được sử dụng với các hàm thủ tục sự kiện của libpq, nếu khác sẽ gây ra sự lúng túng. Qui tắc đơn giản nhất để viết mã sẽ làm việc là hãy đảm bảo rằng các thủ tục sự kiện được khai báo static. Nếu địa chỉ thủ tục đó phải sẵn sàng bên ngoài tệp nguồn của riêng nó, hãy mở hàm tách bạch để trả về địa chỉ đó.

31.12.3. Hàm hỗ trợ sự kiện

PQregisterEventProc

Đăng ký một thủ tục gọi ngược lại sự kiện với libpq.

```
int PQregisterEventProc(PGconn *conn, PGEventProc proc,
const char *name, void *passThrough);
```

Một thủ tục sự kiện phải được đăng ký một lần trong từng PGconn mà bạn muốn nhận các sự kiện đó. Không có giới hạn, khác với bộ nhớ, về số lượng thủ tục sự kiện có thể được đăng ký với một kết nối. Hàm đó trả về giá trị khác zero nếu nó thành công và zero nếu nó hỏng.

Đối số proc sẽ bị hoãn khi một sự kiện libpq được đưa ra. Địa chỉ bộ nhớ của nó cũng được sử dụng để tra cứu instanceData. Đối số name được sử dụng để tham chiếu tới thủ tục sự kiện trong các thông điệp lỗi. Giá trị này không thể là NULL hoặc chuỗi độ dài zero. Chuỗi tên được sao chép vào trong PGconn, sao cho những gì được truyền cần không sống lâu mãi được. Con trỏ passThrough được truyền tới proc bất kỳ khi nào một sự kiện xảy ra. Đối số này có thể là NULL.

PQsetInstanceData

Thiết lập kết nối instanceData của conn cho thủ tục proc đối với data. Điều này trả về khác zero nếu thành công và zero nếu thất bại. (Thất bại chỉ có khả năng nếu proc còn chưa được đăng ký đúng trong conn).

```
int PQsetInstanceData(PGconn *conn, PGEventProc proc, void *data);
```

PQinstanceData

Trả về kết nối instanceData của conn có liên quan tới thủ tục proc, hoặc NULL nếu không có.

```
void *PQinstanceData(const PGconn *conn, PGEventProc proc);
```

PQresultSetInstanceData

Thiết lập instanceData của kết quả cho proc tới data. Điều này trả về khác zero nếu thành công và zero nếu thất bại. (Thất bại sẽ chỉ có khả năng nếu proc còn chưa được đăng ký đúng phù hợp trong kết quả đó).

```
int PQresultSetInstanceData(PGresult *res, PGEventProc proc, void *data);
```

PQresultInstanceData

Trả về instanceData của kết quả có liên quan tới proc, hoặc NULL nếu không có.

```
void *PQresultInstanceData(const PGresult *res, PGEventProc proc);
```

31.12.4. Ví dụ sự kiện

Đây là một ví dụ xương sống của việc quản lý dữ liệu riêng có liên quan tới các kết nối và các kết quả của libpq.

```
/* required header for libpq events (note: includes libpq-fe.h) */
#include <libpq-events.h>
/* The instanceData */
typedef struct
{
        int n;
        char *str;
} mydata;
```

```
/* PGEventProc */
static int myEventProc(PGEventId evtId, void *evtInfo, void *passThrough);
int
main(void)
{
	mydata *data;
	PGresult *res;
	PGconn *conn = PQconnectdb("dbname = postgres");
	if (PQstatus(conn) != CONNECTION_OK)
	{
		fprintf(stderr, "Connection to database failed: %s",
			PQerrorMessage(conn));
		PQfinish(conn);
		return 1;
	}
	/* called once on any connection that should receive events.
	* Sends a PGEVT_REGISTER to myEventProc.
	*/
	if (!PQregisterEventProc(conn, myEventProc, "mydata_proc", NULL))
	{
		fprintf(stderr, "Cannot register PGEventProc\n");
		PQfinish(conn);
		return 1;
	}
	/* conn instanceData is available */
	data = PQinstanceData(conn, myEventProc);
	/* Sends a PGEVT_RESULTCREATE to myEventProc */
	res = PQexec(conn, "SELECT 1 + 1");
	/* result instanceData is available */
	data = PQresultInstanceData(res, myEventProc);
	/* If PG_COPYRES_EVENTS is used, sends a PGEVT_RESULTCOPY to myEventProc */
	res_copy = PQcopyResult(res, PG_COPYRES_TUPLES | PG_COPYRES_EVENTS);
	/* result instanceData is available if PG_COPYRES_EVENTS was
	* used during the PQcopyResult call.
	*/
	data = PQresultInstanceData(res_copy, myEventProc);
	/* Both clears send a PGEVT_RESULTDESTROY to myEventProc */
	PQclear(res);
	PQclear(res_copy);
	/* Sends a PGEVT_CONNDESTROY to myEventProc */
	PQfinish(conn);
	return 0;
}
static int
myEventProc(PGEventId evtId, void *evtInfo, void *passThrough)
{
	switch (evtId)
	{
	case PGEVT_REGISTER:
	{
		PGEventRegister *e = (PGEventRegister *)evtInfo;
		mydata *data = get_mydata(e->conn);
		/* associate app specific data with connection */
		PQsetInstanceData(e->conn, myEventProc, data);
		break;
	}
	case PGEVT_CONNRESET:
	{
		PGEventConnReset *e = (PGEventConnReset *)evtInfo;
		mydata *data = PQinstanceData(e->conn, myEventProc);
		if (data)
			memset(data, 0, sizeof(mydata));
```

```
            break;
        }
        case PGEVT_CONNDESTROY:
        {
            PGEventConnDestroy *e = (PGEventConnDestroy *)evtInfo;
            mydata *data = PQinstanceData(e->conn, myEventProc);
            /* free instance data because the conn is being destroyed */
            if (data)
                free_mydata(data);
            break;
        }
        case PGEVT_RESULTCREATE:
        {
            PGEventResultCreate *e = (PGEventResultCreate *)evtInfo;
            mydata *conn_data = PQinstanceData(e->conn, myEventProc);
            mydata *res_data = dup_mydata(conn_data);
            /* associate app specific data with result (copy it from conn) */
            PQsetResultInstanceData(e->result, myEventProc, res_data);
            break;
        }
        case PGEVT_RESULTCOPY:
        {
            PGEventResultCopy *e = (PGEventResultCopy *)evtInfo;
            mydata *src_data = PQresultInstanceData(e->src, myEventProc);
            mydata *dest_data = dup_mydata(src_data);
            /* associate app specific data with result (copy it from a result) */
            PQsetResultInstanceData(e->dest, myEventProc, dest_data);
            break;
        }
        case PGEVT_RESULTDESTROY:
        {
            PGEventResultDestroy *e = (PGEventResultDestroy *)evtInfo;
            mydata *data = PQresultInstanceData(e->result, myEventProc);
            /* free instance data because the result is being destroyed */
            if (data)
                free_mydata(data);
            break;
        }
        /* unknown event id, just return TRUE. */
        default:
            break;
        }
        return TRUE; /* event processing succeeded */
}
```

31.13. Các biến môi trường

Các biến môi trường sau đây có thể được sử dụng để chọn các giá trị tham số kết nối mặc định, nó sẽ được PQconnectdb, PQsetdbLogin và PQsetdb sử dụng nếu không giá trị nào trực tiếp được việc gọi mã chỉ định. Là hữu dụng để tránh việc lập trình cứng thông tin kết nối cơ sở dữ liệu trong các ứng dụng máy trạm đơn giản, vì dụ thế.

- PGHOST hành xử y hệt như tham số kết nối host.
- PGHOSTADDR hành xử y hệt như tham số kết nối hostaddr. Điều này có thể được thiết lập hoặc thêm vào PGHOST để tránh tổng chi phí tra cứu DNS.
- PGPORT hành xử y hệt như tham số kết nối cổng.
- PGDATABASE hành xử y hệt như tham số kết nối dbname.

- PGUSER hành xử y hệt như tham số kết nối người sử dụng.
- PGPASSWORD hành xử y hệt như tham số kết nối mật khẩu. Sử dụng biến môi trường này không được khuyến cáo vì các lý do an toàn, như một số hệ điều hành cho phép những người sử dụng không phải là gốc root xem các biến môi trường tiến trình qua ps; thay vào đó cân nhắc việc sử dụng tệp ~/.pgpass (xem Phần 31.14).
- PGPASSFILE chỉ định tên của tệp mật khẩu để sử dụng cho việc tra cứu. Nếu không được thiết lập, nó mặc định là ~/.pgpass (xem Phần 31.14).
- PGSERVICE hành xử y hệt như tham số kết nối dịch vụ.
- PGSERVICEFILE chỉ định tên của tệp dịch vụ kết nối theo từng người sử dụng. Nếu không được thiết lập, nó mặc định về ~/.pg_service.conf (xem Phần 31.15).
- PGREALM thiết lập địa hạt Kerberos để sử dụng với PostgreSQL, nếu nó là khác với địa hạt cục bộ. Nếu PGREALM được thiết lập, các ứng dụng libpq sẽ cố xác thực với các máy chủ cho địa hạt này và sử dụng các tệp vé (ticket) riêng rẽ để tránh các xung đột với các tệp vé cục bộ. Các biến môi trường này chỉ được sử dụng nếu xác thực Kerberos được máy chủ chọn.
- PGOPTIONS hành xử y hệt như tham số kết nối các lựa chọn.
- PGAPPNAME hành xử y hệt như tham số kết nối application_name.
- PGSSLMODE hành xử y hệt như tham số kết nối sslmode.
- PGREQUIRESSL hành xử y hệt như tham số kết nối requiressl.
- PGSSLCERT hành xử y hệt như tham số kết nối sslcert.
- PGSSLKEY hành xử y hệt như tham số kết nối sslkey.
- PGSSLROOTCERT hành xử y hệt như tham số kết nối sslrootcert.
- PGSSLCRL hành xử y hệt như tham số kết nối sslcrl.
- PGKRBSRVNAME hành xử y hệt như tham số kết nối krbsrvname.
- PGGSSLIB hành xử ý hệt như tham số kết nối gsslib.
- PGCONNECT_TIMEOUT hành xử y hệt như tham số kết nối connect_timeout.

Các biến môi trường sau có thể được sử dụng để chỉ định hành vi mặc định cho từng phiên làm việc PostgreSQL. (Xem thêm các lệnh ALTER USER và ALTER DATABASE về các cách thức để thiết lập hành vi mặc định trên cơ sở từng người sử dụng hoặc từng cơ sở dữ liệu).

- PGDATESTYLE thiết lập kiểu mặc định trình bày ngày tháng/thời gian. (Tương đương với SET datestyle TO ...).
- PGTZ thiết lập vùng thời gian mặc định. (Tương đương với SET timezone TO ...).
- PGCLIENTENCODING thiết lập bộ mã ký tự mặc định cho máy chủ. (Tương đương với SET client_encoding TO ...).
- PGGEQO thiết lập chế độ mặc định cho trình tối ưu hóa truy vấn gốc. (Tương đương với SET

geqo TO ...).

Hãy tham chiếu tới lệnh SQL SET để có thông tin về các giá trị đúng cho các biến môi trường đó.

Các biến môi trường sau đây xác định hành vi nội bộ của libpq; chúng ghi đè các mặc định được biên dịch.

- PGSYSCONFDIR thiết lập thư mục chứa tệp pg_service.conf và trong một phiên bản trong tương lai có khả năng các tệp cấu hình rộng khắp hệ thống khác.
- PGLOCALEDIR thiết lập thư mục chứa các tệp locale cho sự quốc tế hóa các thông điệp.

31.14. Tệp mật khẩu

Tệp .pgpass trong một thư mục gốc (home) của người sử dụng hoặc tệp được PGPASSFILE tham chiếu có thể chứa các mật khẩu sẽ được sử dụng nếu kết nối đó đòi hỏi một mật khẩu (và không mật khẩu nào từng được chỉ định nếu khác). Trong Microsoft Windows tệp được đặt tên %APPDATA %\postgresql\pgpass.conf (trong đó %APPDATA% tham chiếu tới thư mục con Dữ liệu Ứng dụng (Application Data) trong hồ sơ người sử dụng).

Tệp này sẽ có các dòng định dạng sau đây:

```
hostname:port:database:username:password
```

Từng trong số 4 trường đầu có thể là một giá trị chữ, hoặc *, nó khớp với mọi điều. Trường mật khẩu từ dòng đầu khớp với các tham số kết nối hiện hành sẽ được sử dụng. (Vì thế, hãy đặt các khoản đầu vào đặc thù hơn trước khi bạn sử dụng các dấu *). Nếu một khoản đầu vào cần có dấu hai chấm : hoặc \, thì thoát ra khỏi ký tự này bằng dấu \. Một tên host của localhost khớp với cả TCP (tên host localhost) và các kết nối socket miền Unix (thư mục socket mặc định hoặc rỗng pghost) tới từ máy cục bộ. Trong một máy chủ dự phòng, một tên cơ sở dữ liệu replication khớp với các kết nối nhân bản dòng được thực hiện cho máy chủ chính.

Trong các hệ thống Unix, sự cho phép trong .pgpass phải không cho phép bất kỳ truy cập nào tới thế giới hoặc nhóm; lưu trữ điều này bằng lệnh chmod 0600 ~/.pgpass. Nếu các sự cho phép đó là ít chặt chẽ hơn so với điều này, thì tệp đó sẽ bị bỏ qua. Trong Microsoft Windows, được giả thiết là tệp đó được lưu trữ trong một thư mục có an toàn, sao cho không kiểm tra sự cho phép đặc biệt nào được thực hiện.

31.15. Tệp dịch vụ kết nối

Tệp dịch vụ kết nối cho phép các tham số kết nối libpq sẽ có liên quan tới một tên dịch vụ duy nhất. Tên dịch vụ đó có thể sau đó được một kết nối libpq chỉ định, và các thiết lập có liên quan sẽ được sử dụng. Điều này cho phép các tham số kết nối sẽ được sửa đổi mà không đòi hỏi một sự biên dịch lại ứng dụng libpq. Tên dịch vụ đó cũng có thể được chỉ định bằng việc sử dụng biến môi trường PGSERVICE.

Tệp dịch vụ kết nối đó có thể là một tệp dịch vụ theo từng người sử dụng trong ~/.pg_service.conf hoặc vị trí được biến môi trường PGSERVICEFILE chỉ định, hoặc nó có thể là một tệp rộng khắp hệ thống ở etc/pg_service.conf hoặc trong thư mục được biến môi trường PGSYSCONFDIR chỉ định. Nếu

các định nghĩa dịch vụ với tên y hệt tồn tại trong tệp hệ thống và người sử dụng, thì tệp người sử dụng đó được ưu tiên trước.

Tệp đó sử dụng một định dạng "tệp INI" nơi mà tên phần đó là tên dịch vụ và các tham số là các tham số kết nối; xem Phần 31.1 để có một danh sách. Ví dụ:

```
# comment
[mydb]
host=somehost
port=5433
user=admin
```

Một tệp ví dụ được đưa ra tại share/pg_service.conf.sample.

31.16. Tra cứu LDAP các tham số kết nối

Nếu libpq từng được biên dịch với sự hỗ trợ của LDAP (lựa chọn --with-ldap cho cấu hình) thì có khả năng truy xuất các lựa chọn kết nối giống host hoặc dbname qua LDAP từ một máy chủ trung tâm. Ưu thế là nếu các tham số kết nối cho một cơ sở dữ liệu thay đổi, thì thông tin kết nối đó không phải cập nhật trong tất cả các máy trạm.

Sự tra cứu tham số kết nối LDAP sử dụng tệp dịch vụ kết nối pg_service.conf (xem Phần 31.15). Một dòng trong một đoạn pg_service.conf bắt đầu bằng ldap:// sẽ được thừa nhận như một URL LDAP và một truy vấn LDAP sẽ được thực hiện. Kết quả phải là một danh sách các cặp keyword = value mà sẽ được sử dụng để thiết lập các lựa chọn kết nối. URL đó phải tuân theo RFC 1959 và ở dạng

```
ldap://[hostname[:port]]/search_base?
```

trong đó các mặc định hostname đối với localhost và port mặc định là 389.

Việc xử lý pg_service.conf sẽ kết thúc sau một tra cứu LDAP thành công, nhưng sẽ được tiếp tục nếu máy chủ LDAP không thể liên hệ được. Điều này sẽ cung cấp một sự dự phòng với các dòng URL LDAP xa hơn mà chỉ tới các máy chủ LDAP khác, các cặp keyword = value kinh điển, hoặc các lựa chọn kết nối mặc định. Nếu bạn muốn có một thông điệp trong trường hợp này, hãy thêm một dòng không đúng cú pháp vào sau URL LDAP.

Một khoản đầu vào ví dụ LDAP từng được tạo ra với tệp LDIF là

```
version:1
dn:cn=mydatabase,dc=mycompany,dc=com
changetype:add
objectclass:top
objectclass:groupOfUniqueNames
cn:mydatabase
uniqueMember:host=dbserver.mycompany.com
uniqueMember:port=5439
uniqueMember:dbname=mydb
uniqueMember:user=mydb_user
uniqueMember:sslmode=require
```

có thể được truy vấn với URL LDAP sau đây:

```
ldap://ldap.mycompany.com/dc=mycompany,dc=com?uniqueMember?one?(cn=mydatabase)
```

Bạn cũng có thể trộn các khoản đầu vào tệp dịch vụ thông thường với các tra cứu LDAP. Một ví dụ hoàn chỉnh cho một đoạn trong pg_service.conf có thể là:

chỉ host và cổng port sẽ được lưu trữ trong LDAP, chỉ định rõ ràng dbname và user

```
[customerdb]
dbname=customer
user=appuser
ldap://ldap.acme.com/cn=dbserver,cn=hosts?pgconnectinfo?base?(objectclass=*)
```

31.17. Hỗ trợ SSL

PostgreSQL có hỗ trợ bẩm sinh cho việc sử dụng các kết nối SSL để mã hóa các giao tiếp máy trạm/máy chủ để nâng cao an toàn. Xem Phần 17.8 để có chi tiết về chức năng SSL phía máy chủ.

libpq đọc tệp cấu hình OpenSSL rộng khắp hệ thống. Mặc định, tệp này được đặt tên là openssl.cnf và nằm trong thư mục được nêu bằng openssl version -d. Mặc định này có thể bị ghi đè bằng việc thiết lập biến môi trường OPENSSL_CONF cho tên của tệp cấu hình mong muốn.

31.17.1. Kiểm tra hợp lệ chứng thực

Mặc định, PostgreSQL sẽ không thực hiện bất kỳ sự kiểm tra hợp lệ nào chứng thực máy chủ. Điều này có nghĩa là có khả năng để đánh lừa nhận diện máy chủ (ví dụ bằng việc sửa đổi một bản ghi DNS hoặc bằng việc tiếp quản địa chỉ IP máy chủ) mà không cần biết máy trạm. Để tránh sự đánh lừa, kiểm tra hợp lệ chứng thực SSL phải được sử dụng.

Nếu tham số sslmode được thiết lập về verify-ca, thì libpq sẽ kiểm tra hợp lệ xem máy chủ có tin cậy hay không bằng việc kiểm tra chuỗi chứng thực cho tới tận cơ quan chứng thực (CA). Nếu sslmode được thiết lập về verify-full, thì libpq cũng sẽ kiểm tra hợp lệ xem tên máy chủ host có trùng với chứng thực đó hay không. Kết nối SSL sẽ hỏng nếu chứng thực máy chủ không thể kiểm tra hợp lệ được. verify-full được khuyến cáo trong hầu hết các môi trường nhạy cảm nhất về an toàn.

Ở chế độ kiểm tra hợp lệ đầy đủ verify-full, thuộc tính cn (Tên Chung - Common Name) của chứng thực khớp với tên host. Nếu thuộc tính cn bắt đầu bằng một dấu sao (*), thì nó sẽ được coi như một ký tự đại diện, và sẽ khớp với tất cả các ký tự ngoại trừ dấu chấm (.). Điều này có nghĩa là chứng thực đó sẽ không khớp với các miền phụ. Nếu kết nối đó được làm bằng việc sử dụng một địa chỉ IP thay vì một tên host, thì địa chỉ IP đó sẽ được khớp (mà không làm bất kỳ tra cứu DNS nào).

Để cho phép kiểm tra hợp lệ chứng thực máy chủ, (các) chứng thực của một hoặc nhiều CA được tin cậy hơn phải được đặt trong tệp ~/.postgresql/root.crt trong thư mục gốc home của người sử dụng. (Trên Microsoft Windows tệp đó có tên là %APPDATA%\postgresql\root.crt).

Các khoản đầu vào của Danh sách Thu hồi Chứng thực - CRL (Certificate Revocation List) cũng được kiểm tra nếu tệp ~/.postgresql/root.crl tồn tại (%APPDATA%\postgresql\root.crl trong Microsoft Windows).

Vị trí của tệp chứng thực gốc root và CRL có thể được thay đổi bằng việc thiết lập các tham số kết nối sslrootcert và sslcrl hoặc các biến môi trường PGSSLROOTCERT và PGSSLCRL.

> **Lưu ý**: Để tương thích ngược với các phiên bản trước đó của PostgreSQL, nếu một tệp CA gốc root tồn tại, thì hành vi của sslmode=require sẽ là y hệt như hành vi của verify-ca, nghĩa là

chứng thực máy chủ được kiểm tra hợp lệ đối với CA. Việc dựa vào hành vi này được khuyến khích, và các ứng dụng mà cần sự kiểm tra hợp lệ chứng thực sẽ luôn sử dụng verify-ca hoặc verify-full.

31.17.2. Chứng thực máy trạm

Nếu máy chủ yêu cầu một chứng thực máy trạm tin cậy, thì libpq sẽ gửi chứng thực được lưu trữ trong tệp ~/.postgresql/postgresql.crt vào thư mục gốc home của người sử dụng. Chứng thực đó phải được ký bởi một trong các cơ quan chứng thực (CA) được máy chủ đó tin cậy. Một tệp khóa cá nhân phải không cho phép bất kỳ sự truy cập nào tới thế giới hoặc nhóm; đạt được điều này bằng lệnh chmod 0600 ~/.postgresql/postgresql.key. Trong Microsoft Windows các tệp đó được đặt tên là %APPDATA%\postgresql\postgresql.crt và %APPDATA%\postgresql\postgresql.key, và không có kiểm tra các quyền đặc biệt vì thư mục đó được coi là an toàn. Vị trí của chứng thực và các tệp khóa có thể bị ghi đè bằng các tham số kết nối sslcert và sslkey hoặc các biến môi trường PGSSLCERT và PGSSLKEY.

Trong một số trường hợp, chứng thực máy trạm có thể được ký bằng một CA "ngay lập tức", thay vì một CA mà được máy chủ tin cậy trực tiếp. Để sử dụng một chứng thực như vậy, hãy nối chứng thực của cơ quan ký tới tệp postgresql.crt, sau đó chứng thực của cơ quan cha của nó, và cứ thế lên tới một cơ quan gốc "root" mà được máy chủ đó tin cậy. Chứng thực gốc root sẽ được đưa vào trong từng trường hợp nơi mà postgresql.crt có chứa nhiều hơn một chứng thực.

Lưu ý rằng root.crt liệt kê các CA mức đỉnh mà được coi là tin cậy cho việc ký chứng thực máy chủ. Theo nguyên tắc thì không cần thiết liệt kê CA mà đã ký chứng thực của máy trạm, dù trong hầu hết các trường hợp CA đó cũng có thể được tin cậy cho các chứng thực máy chủ.

31.17.3. Bảo vệ được cung cấp trong các chế độ khác nhau

Các giá trị khác nhau cho tham số sslmode đưa ra các mức bảo vệ khác nhau. SSL có thể cung cấp sự bảo vệ chống lại 3 dạng tấn công:

Bảng 31-2. Các cuộc tấn công SSL

Dạng	Mô tả
Nghe trộm - (Eavesdropping)	Nếu một bên thứ ba có thể kiểm tra giao thông mạng giữa máy trạm và máy chủ, thì nó có thể đọc cả thông tin kết nối (bao gồm tên và mật khẩu của người sử dụng) và dữ liệu mà được truyền qua. SSL sử dụng mã hóa để ngăn chặn điều này.
Chặn giữa đường - Man in the middle (MITM)	Nếu một bên thứ ba có thể sửa đổi dữ liệu trong khi truyền giữa máy chủ và máy trạm, thì nó có thể giả vờ là máy chủ đó và vì thế thấy và sửa đổi được dữ liệu thậm chí nếu nó được mã hóa. Bên thứ 3 có thể sau đó chuyển tiếp thông tin và dữ liệu kết nối tới máy chủ gốc ban đầu, làm cho không có khả năng để dò tìm ra cuộc tấn công này. Các vật trung gian phổ biến để làm điều này bao gồm việc cướp địa chỉ và đầu độc DNS, trong khi đó máy trạm được định tuyến tới một máy chủ khác được mong đợi. Cũng có vài phương pháp tấn công khác có thể hoàn tất điều này. SSL sử dụng sự kiểm tra hợp lệ chứng thực để ngăn chặn điều này, bằng việc xác thực máy chủ tới máy trạm.
Thủ vai người khác -	Nếu một bên thứ ba có thể giả vờ là một máy trạm có quyền, thì nó có thể đơn giản truy cập dữ liệu mà nó đáng ra không được truy cập tới. Thường thì điều này có thể xảy ra qua sự quản lý mật khẩu

Dạng	Mô tả
Impersonation	không an toàn. SSL sử dụng các chứng thực máy trạm để ngăn chặn điều này, bằng việc chắc chắn rằng chỉ những người nắm giữ các chứng thực hợp lệ có thể truy cập được máy chủ.

Đối với một kết nối được hiểu là an toàn, sử dụng SSL phải được thiết lập cấu hình trên cả máy trạm và máy chủ trước khi kết nối đó được thực hiện. Nếu chỉ được thiết lập cấu hình trên máy chủ, thì máy trạm có thể kết thúc việc gửi các thông tin nhạy cảm (như mật khẩu) trước khi nó biết rằng máy chủ đó yêu cầu an toàn cao. Trong libpq, các kết nối an toàn có thể được đảm bảo bằng việc thiết lập tham số sslmode về verify-full hoặc verify-ca, và cung cấp cho hệ thống bằng một chứng thực gốc root để kiểm tra hợp lệ. Điều này là tương tự với việc sử dụng một URL https cho việc duyệt web được mã hóa.

Một khi máy chủ đã được xác thực, thì máy trạm có thể truyền các dữ liệu nhạy cảm. Điều này có nghĩa là cho tới thời điểm này, máy trạm không cần biết liệu các chứng thực sẽ được sử dụng cho xác thực, làm cho nó an toàn để chỉ định điều đó chỉ trong cấu hình máy chủ.

Tất cả các lựa chọn SSL mang tổng chi phí ở dạng mã hóa và trao đổi khóa, nên có một sự lựa chọn mà phải được thực hiện giữa hiệu năng và an toàn. Bảng sau đây minh họa các rủi ro mà các giá trị sslmode khác nhau bảo vệ, và tuyên bố nào chúng làm về an toàn và tổng chi phí:

Bảng 31-3. Các mô tả chế độ SSL

sslmode	Chống nghe lén	Chống chặn giữa đường	Tuyên bố
disable	Không	Không	Tôi không quan tâm về an toàn, và tôi không muốn trả chi phí mã hóa
allow	Có thể	Không	Tôi không quan tâm về an toàn, nhưng tôi sẽ trả tiền chi phí mã hóa nếu máy chủ khăng khăng về nó.
prefer	Có thể	Không	Tôi không quan tâm về mã hóa, nhưng tôi muốn trả chi phí mã hóa nếu máy chủ hỗ trợ điều này.
require	Có	Không	Tôi muốn dữ liệu của tôi được mã hóa, và tôi chấp nhận chi phí. Tôi tin tưởng rằng mạng sẽ chắc chắn tôi luôn kết nối tới máy chủ mà tôi muốn.
verify-ca	Có	Phụ thuộc vào chính sách CA	Tôi muốn dữ liệu của tôi được mã hóa, và tôi chấp nhận chi phí. Tôi muốn chắc rằng tôi kết nối tới máy chủ mà tôi tin cậy.
verify-full	Có	Có	Tôi muốn dữ liệu của tôi được mã hóa, và tôi chấp nhận chi phí. Tôi muốn chắc chắn rằng tôi kết nối tới một máy chủ tôi tin cậy, và đó là điều tôi chỉ định

Sự khác biệt giữa verify-ca và verify-full phụ thuộc vào chính sách của CA gốc root. Nếu một CA công cộng được sử dụng, verify-ca cho phép các kết nối tới một máy chủ mà ai đó khác nữa có thể đã đăng ký với CA đó. Trong trường hợp này, verify-full sẽ luôn được sử dụng. Nếu một CA địa phương được sử dụng, hoặc thậm chí một chứng thực tự ký, thì việc sử dụng verify-ca thường cung cấp đủ sự bảo vệ.

Giá trị mặc định cho sslmode là prefer. Như được chỉ ra trong bảng, điều này không có ý nghĩa từ một quan điểm an toàn, và nó chỉ hứa hẹn thực thi tổng chi phí nếu có khả năng. Cũng được đưa ra như mặc định cho tính tương thích ngược, và được khuyến cáo trong các triển khai an toàn.

31.17.4. Sử dụng tệp SSL

Bảng 31-4. Sử dụng tệp SSL Libpq/máy trạm

Tệp	Nội dung	Hiệu ứng
~/.postgresql/postgresql.crt	chứng thực máy trạm	được máy chủ yêu cầu
~/.postgresql/postgresql.key	khóa cá nhân máy trạm	chứng minh chứng thực máy trạm được chủ sở hữu gửi đi; không chỉ ra chủ sở hữu chứng thực là đáng tin cậy
~/.postgresql/root.crt	các cơ quan chứng thực tin cậy	kiểm tra chứng thực máy chủ được một cơ quan chứng thực tin cậy ký
~/.postgresql/root.crl	các chứng thực bị các cơ quan chứng thực thu hồi	chứng thực máy chủ phải không nằm trong danh sách này

31.17.5. Khởi tạo thư viện SSL

Nếu ứng dụng của bạn khởi tạo các thư viện libssl và/hoặc libcrypto và libpq được xây dựng có hỗ trợ SSL, thì bạn nên gọi PQinitOpenSSL để nói cho libpq rằng các thư viện libssl và/hoặc libcrypto đã được ứng dụng của bạn khởi tạo, sao cho libpq cũng sẽ không khởi tạo các thư viện đó. Xem http://h71000.www7.hp.com/doc/83final/BA554_90007/ch04.html để có thêm chi tiết về SSL API.

PQinitOpenSSL

Cho phép các ứng dụng chọn các thư viện an toàn nào để khởi tạo.

```
void PQinitOpenSSL(int do_ssl, int do_crypto);
```

Khi do_ssl là khác zero, thì libpq sẽ khởi tạo thư viện OpenSSL trước khi lần đầu mở một kết nối cơ sở dữ liệu. Khi do_crypto là khác zero, thì thư viện libcrypto sẽ được khởi tạo. Mặc định (nếu PQinitOpenSSL không được gọi), thì cả 2 thư viện sẽ được khởi tạo. Khi sự hỗ trợ SSL không được biên dịch bên trong, thì hàm đó đang hiện diện nhưng không làm gì.

Nếu ứng dụng của bạn sử dụng và khởi tạo hoặc OpenSSL hoặc thư viện libcrypto nằm bên dưới nó, thì bạn phải gọi hàm này với các zero cho (các) tham số đúng phù hợp trước khi lần đầu mở một kết nối cơ sở dữ liệu. Hơn nữa phải chắc chắn rằng bạn đã làm xong sự khởi tạo đó trước khi mở một kết nối cơ sở dữ liệu.

PQinitSSL

Cho phép các ứng dụng chọn các thư viện an toàn nào để khởi tạo.

```
void PQinitSSL(int do_ssl);
```

Hàm này là tương đương với PQinitOpenSSL(do_ssl, do_ssl). Đủ cho các ứng dụng mà khởi tạo cả 2 hoặc không cả OpenSSL lẫn libcrypto.

PQinitSSL từng hiện diện kể từ PostgreSQL 8.0, trong khi PQinitOpenSSL từng được đưa thêm vào trong PostgreSQL 8.4, nên PQinitSSL có lẽ được ưu tiên cho các ứng dụng cần làm việc với các phiên bản cũ hơn của libpq.

31.18. Hành vi trong các chương trình có luồng

libpq được đưa vào lại và mặc định là có luồng an toàn. Bạn có thể cần sử dụng các lựa chọn dòng lệnh của trình biên dịch đặc biệt khi bạn biên dịch mã ứng dụng của bạn. Hãy tham chiếu tới tài liệu hệ thống của bạn để có thông tin về cách xây dựng các ứng dụng có luồng, hoặc xem xét src/Makefile.global về PTHREAD_CFLAGS và PTHREAD_LIBS. Hàm này cho phép truy vấn tình trạng luồng an toàn của libpq:

PQisthreadsafe

Trả về tình trạng an toàn luồng của thư viện libpq.

```
int PQisthreadsafe();
```

Trả về 1 nếu libpq là luồng an toàn và 0 nếu không phải.

Một hạn chế luồng là không 2 luồng nào cố gắng điều khiển cùng đối tượng PGconn tại cùng y hệt một thời điểm. Đặc biệt, bạn không thể đưa ra các lệnh đồng thời từ các luồng khác nhau qua cùng y hệt đối tượng kết nối. (Nếu bạn cần chạy các lệnh đồng thời, hãy sử dụng nhiều kết nối).

Các đối tượng PGresult thường là chỉ đọc sau khi tạo ra, và vì thế có thể được truyền tự do giữa các luồng. Tuy nhiên, nếu bạn sử dụng bất kỳ hàm PGresult đang sửa đổi nào được mô tả trong Phần 31.10 hoặc 31.12, thì cũng tùy bạn để tránh các hoạt động đồng thời trong cùng y hệt một PGresult.

Các hàm không được tán thành PQrequestCancel và PQoidStatus không là luồng an toàn và sẽ không được sử dụng trong các chương trình nhiều luồng. PQrequestCancel có thể được thay thế bằng PQcancel. PQoidStatus có thể được thay thế bằng PqoidValue.

Nếu bạn đang sử dụng Kerberos bên trong ứng dụng của bạn (bổ sung vào trong libpq), thì bạn sẽ cần tiến hành việc khóa xung quanh các lời gọi Kerberos vì các hàm Kerberos không là luồng an toàn. Xem hàm PQregisterThreadLock trong mã nguồn libpq để có cách tiến hành việc khóa hợp tác giữa libpq và ứng dụng của bạn.

Nếu bạn trải nghiệm các vấn đề với các ứng dụng có luồng, hãy chạy chương trình trong src/tools/thread để thấy liệu nền tảng của bạn có các hàm luồng không an toàn hay không. Chương trình này được chạy bằng configure, nhưng đối với các phân phối nhị phân thì thư viện của bạn có thể không khớp với thư viện được sử dụng để xây dựng các nhị phân.

31.19. Xây dựng các chương trình libpq

Để xây dựng (như biên dịch và kết nối) một chương trình có sử dụng libpq bạn cần làm tất cả những điều sau đây:

- Đưa vào tệp đầu đề libpq-fe.h:

  ```
  #include <libpq-fe.h>
  ```

  Nếu bạn thất bại làm thế thì bạn sẽ thường có các thông điệp lỗi từ trình biên dịch của bạn tương tự như:

  ```
  foo.c: In    function ‘main’:
  ```

```
foo.c:34:   'PGconn' undeclared (first use in this function)
foo.c:35:   'PGresult' undeclared (first use in this function)
foo.c:54:   'CONNECTION_BAD' undeclared (first use in this function)
foo.c:68:   'PGRES_COMMAND_OK' undeclared (first use in this function)
foo.c:95:   'PGRES_TUPLES_OK' undeclared (first use in this function)
```

- Trỏ trình biên dịch của bạn tới thư mục nơi mà các tệp đầu đề PostgreSQL đã được cài đặt bằng việc cung cấp lựa chọn -Idirectory cho trình biên dịch của bạn. (Trong một số trường hợp thì trình biên dịch đó sẽ xem xét thư mục theo yêu cầu một cách mặc định, nên bạn có thể bỏ qua lựa chọn này). Ví dụ, dòng lệnh biên dịch của bạn có thể là:

  ```
  cc -c -I/usr/local/pgsql/include testprog.c
  ```

  Nếu bạn đang sử dụng makefiles thì hãy thêm lựa chọn đó vào biến CPPFLAGS:

  ```
  CPPFLAGS += -I/usr/local/pgsql/include
  ```

  Nếu có bất kỳ cơ hội nào chương trình của bạn có thể được biên dịch bằng những người sử dụng khác thì bạn nên không viết mã cứng vị trí thư mục như thế. Thay vào đó, bạn có thể chạy tiện ích pg_config để tìm ra nơi mà các tệp đầu đề nằm trong hệ thống cục bộ đó:

  ```
  $ pg_config –includedir
  /usr/local/include
  ```

  Thất bại để chỉ định lựa chọn đúng để biên dịch sẽ gây ra một thông điệp lỗi như sau:

  ```
  testlibpq.c:8:22: libpq-fe.h: No such file or directory
  ```

- Khi liên kết với chương trình cuối cùng, hãy chỉ định lựa chọn -lpq sao cho thư viện libpq được kéo vào, cũng như lựa chọn -Ldirectory để trỏ trình biên dịch tới thư mục nơi thư viện libpq nằm. (Một lần nữa, trình biên dịch sẽ tìm kiếm vài thư mục một cách mặc định). Vì tính khả chuyển tối đa, hãy đặt lựa chọn -L trước lựa chọn -lpq. Ví dụ:

  ```
  cc -o testprog testprog1.o testprog2.o -L/usr/local/pgsql/lib -lpq
  ```

  Bạn có thể tìm ra thư mục thư viện cũng bằng việc sử dụng pg_config:

  ```
  $ pg_config –libdir
  /usr/local/pgsql/lib
  ```

  Các thông điệp lỗi mà chỉ tới các vấn đề trong vùng này có thể trông giống như sau:

  ```
  testlibpq.o: In function 'main':
  testlibpq.o(.text+0x60): undefined reference to 'PQsetdbLogin'
  testlibpq.o(.text+0x71): undefined reference to 'Pqstatus'
  testlibpq.o(.text+0xa4): undefined reference to 'PQerrorMessage'
  ```

  Điều này có nghĩa là bạn đã quên -lpq.

  ```
  /usr/bin/ld: cannot find -lpq
  ```

  Điều này có nghĩa là bạn đã quên lựa chọn -L hoặc đã không chỉ định đúng thư mục.

31.20. Chương trình ví dụ

Các ví dụ đó và khác có thể thấy được trong thư mục src/test/examples trong phân phối mã nguồn.

Ví dụ 31-1. Chương trình ví dụ 1 của libpq

```
/*
* testlibpq.c
*
*                   Test the C version of libpq, the PostgreSQL frontend library.
*/
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>

static void
exit_nicely(PGconn *conn)
{
        PQfinish(conn);
        exit(1);
}
int
main(int argc, char **argv)
{
        const char      *conninfo;
        PGconn          *conn;
        PGresult        *res;
        int             nFields;
        int             i,
                        j;
        /*
        * If the user supplies a parameter on the command line, use it as the
        * conninfo string; otherwise default to setting dbname=postgres and using
        * environment variables or defaults for all other connection parameters.
        */
        if (argc > 1)
                conninfo = argv[1];
        else
                conninfo = "dbname = postgres";
        /* Make a connection to the database */
        conn = PQconnectdb(conninfo);
        /* Check to see that the backend connection was successfully made */
        if (PQstatus(conn) != CONNECTION_OK)
        {
                fprintf(stderr, "Connection to database failed: %s",
                        PQerrorMessage(conn));
                exit_nicely(conn);
        }
        /*
        * Our test case here involves using a cursor, for which we must be inside
        * a transaction block. We could do the whole thing with a single
        * PQexec() of "select * from pg_database", but that's too trivial to make
        * a good example.
        */
        /* Start a transaction block */
        res = PQexec(conn, "BEGIN");
        if (PQresultStatus(res) != PGRES_COMMAND_OK)
        {
                fprintf(stderr, "BEGIN command failed: %s", PQerrorMessage(conn));
                PQclear(res);
                exit_nicely(conn);
        }
        /*
```

```
        * Should PQclear PGresult whenever it is no longer needed to avoid memory
        * leaks
        */
        PQclear(res);
        /*
        * Fetch rows from pg_database, the system catalog of databases
        */
        res = PQexec(conn, "DECLARE myportal CURSOR FOR select * from pg_database");
        if (PQresultStatus(res) != PGRES_COMMAND_OK)
        {
                fprintf(stderr, "DECLARE CURSOR failed: %s", PQerrorMessage(conn));
                PQclear(res);
                exit_nicely(conn);
        }
        PQclear(res);
        res = PQexec(conn, "FETCH ALL in myportal");
        if (PQresultStatus(res) != PGRES_TUPLES_OK)
        {
                fprintf(stderr, "FETCH ALL failed: %s", PQerrorMessage(conn));
                PQclear(res);
                exit_nicely(conn);
        }
        /* first, print out the attribute names */
        nFields = PQnfields(res);
        for (i = 0; i < nFields; i++)
                printf("%-15s", PQfname(res, i));
        printf("\n\n");
        /* next, print out the rows */
        for (i = 0; i < PQntuples(res); i++)
        {
                for (j = 0; j < nFields; j++)
                        printf("%-15s", PQgetvalue(res, i, j));
                printf("\n");
        }
        PQclear(res);
        /* close the portal ... we don't bother to check for errors ... */
        res = PQexec(conn, "CLOSE myportal");
        PQclear(res);
        /* end the transaction */
        res = PQexec(conn, "END");
        PQclear(res);
        /* close the connection to the database and cleanup */
        PQfinish(conn);
        return 0;
}
```

Ví dụ 31-2. Chương trình ví dụ 2 của libpq

```
/*
* testlibpq2.c
*       Test of the asynchronous notification interface
*
* Start this program, then from psql in another window do
*       NOTIFY TBL2;
* Repeat four times to get this program to exit.
*
* Or, if you want to get fancy, try this:
* populate a database with the following commands
* (provided in src/test/examples/testlibpq2.sql):
*
*       CREATE TABLE TBL1 (i int4);
*
```

```
*       CREATE TABLE TBL2 (i int4);
*
*       CREATE RULE r1 AS ON INSERT TO TBL1 DO
*           (INSERT INTO TBL2 VALUES (new.i); NOTIFY TBL2);
* and do this four times:
*
*       INSERT INTO TBL1 VALUES (10);
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/time.h>
#include <libpq-fe.h>

static void
exit_nicely(PGconn *conn)
{
        PQfinish(conn);
        exit(1);
}
int
main(int argc, char **argv)
{
        const char      *conninfo;
        PGconn          *conn;
        PGresult        *res;
        PGnotify        *notify;
        int             nnotifies;
        /*
        * If the user supplies a parameter on the command line, use it as the
        * conninfo string; otherwise default to setting dbname=postgres and using
        * environment variables or defaults for all other connection parameters.
        */
        if (argc > 1)
                conninfo = argv[1];
        else
                conninfo = "dbname = postgres";
        /* Make a connection to the database */
        conn = PQconnectdb(conninfo);
        /* Check to see that the backend connection was successfully made */
        if (PQstatus(conn) != CONNECTION_OK)
        {
                fprintf(stderr, "Connection to database failed: %s",
                        PQerrorMessage(conn));
                exit_nicely(conn);
        }
        /*
        * Issue LISTEN command to enable notifications from the rule's NOTIFY.
        */
        res = PQexec(conn, "LISTEN TBL2");
        if (PQresultStatus(res) != PGRES_COMMAND_OK)
        {
                fprintf(stderr, "LISTEN command failed: %s", PQerrorMessage(conn));
                Pqclear(res);
                exit_nicely(conn);
        }
        /*
        * should PQclear PGresult whenever it is no longer needed to avoid memory
        * leaks
        */
        PQclear(res);
```

```
        /* Quit after four notifies are received. */
        nnotifies = 0;
        while (nnotifies < 4)
        {
                /*
                * Sleep until something happens on the connection. We use select(2)
                * to wait for input, but you could also use poll() or similar
                * facilities.
                */
                int       sock;
                fd_set    input_mask;
                sock = PQsocket(conn);
                if (sock < 0)
                break;                                  /* shouldn't happen */
                FD_ZERO(&input_mask);
                FD_SET(sock, &input_mask);
                if (select(sock + 1, &input_mask, NULL, NULL, NULL) < 0)
                {
                        fprintf(stderr, "select() failed: %s\n", strerror(errno));
                        exit_nicely(conn);
                }
                /* Now check for input */
                PQconsumeInput(conn);
                while ((notify = PQnotifies(conn)) != NULL)
                {
                        fprintf(stderr,
                                "ASYNC NOTIFY of '%s' received from backend pid %d\n",
                                notify->relname, notify->be_pid);
                        PQfreemem(notify);
                        nnotifies++;
                }
        }
        fprintf(stderr, "Done.\n");
        /* close the connection to the database and cleanup */
        PQfinish(conn);
        return 0;
}
```

Ví dụ 31-3. Chương trình ví dụ 3 của libpq

```
/*
* testlibpq3.c
*                 Test out-of-line parameters and binary I/O.
*
* Before running this, populate a database with the following commands
* (provided in src/test/examples/testlibpq3.sql):
*
* CREATE TABLE test1 (i int4, t text, b bytea);
*
* INSERT INTO test1 values (1, 'joe"s place', '\\000\\001\\002\\003\\004');
* INSERT INTO test1 values (2, 'ho there', '\\004\\003\\002\\001\\000');
*
* The expected output is:
*
* tuple 0: got
* i = (4 bytes) 1
* t = (11 bytes) 'joe's place'
* b = (5 bytes) \000\001\002\003\004
*
* tuple 0: got
* i = (4 bytes) 2
* t = (8 bytes) 'ho there'
```

```
* b = (5 bytes) \004\003\002\001\000
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <libpq-fe.h>

/* for ntohl/htonl */
#include <netinet/in.h>
#include <arpa/inet.h>

static void
exit_nicely(PGconn *conn)
{
        PQfinish(conn);
        exit(1);
}
/*
* This function prints a query result that is a binary-format fetch from
* a table defined as in the comment above. We split it out because the
* main() function uses it twice.
*/
static void
show_binary_results(PGresult *res)
{
        int     i,
                j;
        int     i_fnum,
                t_fnum,
                b_fnum;
/* Use PQfnumber to avoid assumptions about field order in result */
i_fnum = PQfnumber(res, "i");
t_fnum = PQfnumber(res, "t");
b_fnum = PQfnumber(res, "b");
for (i = 0; i < PQntuples(res); i++)
{
        char *iptr;
        char *tptr;
        char *bptr;
        int blen;
        int ival;

        /* Get the field values (we ignore possibility they are null!) */
        iptr = PQgetvalue(res, i, i_fnum);
        tptr = PQgetvalue(res, i, t_fnum);
        bptr = PQgetvalue(res, i, b_fnum);
        /*
        * The binary representation of INT4 is in network byte order, which
        * we'd better coerce to the local byte order.
        */
        ival = ntohl(*((uint32_t *) iptr));
        /*
        * The binary representation of TEXT is, well, text, and since libpq
        * was nice enough to append a zero byte to it, it'll work just fine
        * as a C string.
        *
        * The binary representation of BYTEA is a bunch of bytes, which could
        * include embedded nulls so we have to pay attention to field length.
        */
        blen = PQgetlength(res, i, b_fnum);
        printf("tuple %d: got\n", i);
```

```
        printf(" i = (%d bytes) %d\n",
                PQgetlength(res, i, i_fnum), ival);
        printf(" t = (%d bytes) '%s'\n",
                PQgetlength(res, i, t_fnum), tptr);
        printf(" b = (%d bytes) ", blen);
        for (j = 0; j < blen; j++)
                printf("\\%03o", bptr[j]);
        printf("\n\n");
        }
}
int
main(int argc, char **argv)
{
        const char *conninfo;
        PGconn *conn;
        PGresult *res;
        const char *paramValues[1];
        int paramLengths[1];
        int paramFormats[1];
        uint32_t binaryIntVal;
        /*
        * If the user supplies a parameter on the command line, use it as the
        * conninfo string; otherwise default to setting dbname=postgres and using
        * environment variables or defaults for all other connection parameters.
        */
        if (argc > 1)
                conninfo = argv[1];
        else
                conninfo = "dbname = postgres";
        /* Make a connection to the database */
        conn = PQconnectdb(conninfo);
        /* Check to see that the backend connection was successfully made */
        if (PQstatus(conn) != CONNECTION_OK)
        {
                fprintf(stderr, "Connection to database failed: %s",
                        PQerrorMessage(conn));
                exit_nicely(conn);
        }
        /*
        * The point of this program is to illustrate use of PQexecParams() with
        * out-of-line parameters, as well as binary transmission of data.
        *
        * This first example transmits the parameters as text, but receives the
        * results in binary format. By using out-of-line parameters we can
        * avoid a lot of tedious mucking about with quoting and escaping, even
        * though the data is text. Notice how we don't have to do anything
        * special with the quote mark in the parameter value.
        */
        /* Here is our out-of-line parameter value */
        paramValues[0] = "joe's place";
        res = PQexecParams(conn,
                                "SELECT * FROM test1 WHERE t = $1",
                                1,              /* one param */
                                NULL,           /* let the backend deduce param type */
                                paramValues,
                                NULL,           /* don't need param lengths since text */
                                NULL,           /* default to all text params */
                                1);             /* ask for binary results */
        if (PQresultStatus(res) != PGRES_TUPLES_OK)
        {
                fprintf(stderr, "SELECT failed: %s", PQerrorMessage(conn));
                PQclear(res);
```

```
            exit_nicely(conn);
        }
        show_binary_results(res);
        Pqclear(res);
        /*
        * In this second example we transmit an integer parameter in binary
        * form, and again retrieve the results in binary form.
        *
        * Although we tell PQexecParams we are letting the backend deduce
        * parameter type, we really force the decision by casting the parameter
        * symbol in the query text. This is a good safety measure when sending
        * binary parameters.
        */
        /* Convert integer value "2" to network byte order */
        binaryIntVal = htonl((uint32_t) 2);
        /* Set up parameter arrays for PQexecParams */
        paramValues[0] = (char *) &binaryIntVal;
        paramLengths[0] = sizeof(binaryIntVal);
        paramFormats[0] = 1; /* binary */
        res = PQexecParams(conn,
                           "SELECT * FROM test1 WHERE i = $1::int4",
                           1,           /* one param */
                           NULL,        /* let the backend deduce param type */
                           paramValues,
                           paramLengths,
                           paramFormats,
                           1);          /* ask for binary results */
        if (PQresultStatus(res) != PGRES_TUPLES_OK)
        {
            fprintf(stderr, "SELECT failed: %s", PQerrorMessage(conn));
            PQclear(res);
            exit_nicely(conn);
        }
        show_binary_results(res);
        PQclear(res);
        /* close the connection to the database and cleanup */
        Pqfinish(conn);
        return 0;
}
```

Chương 32. Các đối tượng lớn

PostgreSQL có một tiện ích đối tượng lớn, nó cung cấp sự truy cập dạng dòng tới dữ liệu người sử dụng được lưu trữ theo một cấu trúc đối tượng lớn. Truy cập dòng là hữu dụng khi làm việc với các giá trị dữ liệu là quá lớn để điều khiển thuận tiện như một tổng thể.

Chương này mô tả sự triển khai và lập trình và các giao diện ngôn ngữ truy vấn đối với dữ liệu đối tượng lớn của PostgreSQL. Chúng ta sử dụng thư viện C libpq cho các ví dụ trong chương này, nhưng hầu hết các giao diện lập trình bẩm sinh đối với PostgreSQL hỗ trợ chức năng tương đương. Các giao diện khác có thể sử dụng giao diện đối tượng lớn trong nội bộ để đưa ra sự hỗ trợ chung cho các giá trị lớn. Điều này không được mô tả ở đây.

32.1. Giới thiệu

Tất cả các đối tượng lớn được đặt trong một bảng hệ thống duy nhất gọi là pg_largeobject. PostgreSQL cũng hỗ trợ một hệ thống kho lưu trữ gọi là "TOAST" mà tự động lưu trữ các giá trị lớn hơn một trang cơ sở dữ liệu duy nhất trong một vùng kho lưu trữ thứ 2 đối với từng bảng. Điều này làm cho tiện ích đối tượng lớn một phần lỗi thời. Một ưu thế nữa của tiện ích đối tượng lớn là nó cho phép các giá trị tới 2GB về kích cỡ, trong khi các trường được TOAST hóa có thể nhiều nhất 1GB. Hơn nữa, các đối tượng lớn có thể ngẫu nhiên bị sửa đổi bằng việc sử dụng một API đọc/ghi mà hiệu quả hơn so với việc thực thi các hoạt động như vậy bằng việc sử dụng TOAST.

32.2. Các tính năng triển khai

Triển khai đối tượng lớn chia các đối tượng lớn thành "các khối" và lưu trữ các khối đó trong các hàng trong cơ sở dữ liệu. Một chỉ số dạng B-tree đảm bảo tìm kiếm nhanh cho số lượng đúng các khối khi tiến hành truy cập ngẫu nhiên các cuộc đọc và ghi.

Đối với PostgreSQL 9.0, các đối tượng lớn có một chủ sở hữu và một tập hợp các quyền truy cập, nó có thể được quản lý bằng việc sử dụng GRANT và REVOKE. Vì tính tương thích với các phiên bản trước, xem lo_compat_privileges. Các quyền ưu tiên SELECT được yêu cầu để đọc một đối tượng lớn, và các quyền ưu tiên UPDATE được yêu cầu để ghi hoặc cắt bớt nó. Chỉ chủ sở hữu các đối tượng lớn (hoặc siêu người sử dụng cơ sở dữ liệu) mới có thể bỏ liên kết, bình luận, hoặc thay đổi chủ sở hữu của một đối tượng lớn.

32.3. Giao diện máy trạm

Phần này mô tả các tiện ích mà các thư viện giao diện máy trạm PostgreSQL cung cấp cho việc truy cập các đối tượng lớn. Tất cả sự điều khiển đối tượng lớn bằng việc sử dụng các hàm đó phải diễn ra bên trong một khối giao dịch SQL. Giao diện đối tượng lớn của PostgreSQL được mô hình hóa sau giao diện hệ thống tệp Unix, với những tương tự về open, read, write, lseek, ...

Các ứng dụng máy trạm mà sử dụng giao diện đối tượng lớn trong libpq sẽ bao gồm tệp đầu đề libpq/libpq-fs.h và liên kết với thư viện libpq.

32.3.1. Tạo một đối tượng lớn

Hàm

```
Oid lo_creat(PGconn *conn, int mode);
```

tạo ra một đối tượng lớn mới. Giá trị trả về là OID mà đã được chỉ định tới đối tượng lớn mới đó, hoặc InvalidOid (zero) nếu thất bại. mode là không được sử dụng và bị bỏ qua đối với PostgreSQL 8.1; tuy nhiên, vì tính tương thích ngược với các phiên bản trước đó, là tốt nhất để thiết lập nó về INV_READ, INV_WRITE, hoặc INV_READ | INV_WRITE. (Các hằng biểu tượng đó được xác định trong tệp đầu để libpq/libpq-fs.h).

Một ví dụ:

```
inv_oid = lo_creat(conn, INV_READ|INV_WRITE);
```

Hàm

```
Oid lo_create(PGconn *conn, Oid lobjId);
```

cũng tạo ra một đối tượng lớn mới. OID sẽ được chỉ định có thể được lobjId chỉ định; nếu thế, thất bại sẽ xảy ra nếu OID đó được sử dụng rồi cho một vài đối tượng lớn. Nếu lobjId là InvalidOid (zero) thì lo_create chỉ định một OID không được sử dụng (đây là hành vi y hệt như lo_creat). Giá trị trả về là OID từng được chỉ định cho đối tượng lớn mới, hoặc InvalidOid (zero) nếu thất bại.

lo_create là mới đối với PostgreSQL 8.1; nếu hàm này được chạy đối với một phiên bản máy chủ cũ hơn, thì nó sẽ hỏng và trả về InvalidOid.

Một ví dụ:

```
inv_oid = lo_create(conn, desired_oid);
```

32.3.2. Nhập khẩu một đối tượng lớn

Để nhập khẩu một tệp hệ điều hành như một đối tượng lớn, hãy gọi

```
Oid lo_import(PGconn *conn, const char *filename);
```

filename chỉ định tên hệ điều hành của tệp sẽ được nhập khẩu như một đối tượng lớn. Giá trị trả về là OID mà từng được chỉ định tới đối tượng lớn mới, hoặc InvalidOid (zero) nếu thất bại. Lưu ý rằng tệp đó được đọc bởi thư viện giao diện máy trạm, chứ không phải bằng máy chủ; vì thế nó phải tồn tại trong hệ thống tệp máy chủ và có khả năng đọc được bằng ứng dụng máy trạm.

Hàm

```
Oid lo_import_with_oid(PGconn *conn, const char *filename, Oid lobjId);
```

cũng nhập khẩu một đối tượng lớn mới. OID sẽ được chỉ định có thể được lobjId chỉ định; nếu thế, thất bại xảy ra nếu OID đó đang sử dụng rồi cho một vài đối tượng lớn. Nếu lobjId là InvalidOid (zero) thì lo_import_with_oid chỉ định một OID không sử dụng được (điều này là hành vi y hệt như lo_import). Giá trị trả về là OID mà từng được chỉ định tới đối tượng lớn mới đó, hoặc InvalidOid (zero) nếu thất bại.

lo_import_with_oid là mới đối với PostgreSQL 8.4 và sử dụng lo_create nội bộ mà là mới trong 8.1; nếu hàm này được chạy đối với 8.0 hoặc trước đó, thì sẽ hỏng và trả về InvalidOid.

32.3.3. Xuất khẩu một đối tượng lớn

Để xuất khẩu một đối tượng lớn vào một tệp hệ điều hành, hãy gọi

```
int lo_export(PGconn *conn, Oid lobjId, const char *filename);
```

Đối số lobjId chỉ định OID của đối tượng lớn để xuất khẩu và đối số filename chỉ định tên hệ điều hành của tệp đó. Lưu ý rằng tệp đó được thư viện giao diện máy trạm ghi, chứ không phải máy chủ ghi. Trả về 1 nếu thành công, -1 nếu thất bại.

32.3.4. Mở một đối tượng lớn đang tồn tại

Để mở một đối tượng lớn đang tồn tại để đọc hoặc ghi, hãy gọi

```
int lo_open(PGconn *conn, Oid lobjId, int mode);
```

Đối số lobjId chỉ định OID của đối tượng lớn để mở. Các bit mode kiểm soát liệu đối tượng đó có được mở để đọc (INV_READ), ghi (INV_WRITE), hoặc cả 2 hay không. (Các hàng biểu tượng được xác định trong tệp đầu đề libpq/libpq-fs.h). Một đối tượng lớn không thể mở trước khi nó được tạo ra. lo_open trả về một trình mô tả đối tượng lớn (không tiêu cực) để sử dụng sau này trong lo_read, lo_write, lo_lseek, lo_tell, và lo_close. Trình mô tả chỉ hợp lệ trong khoảng thời gian của giao dịch hiện hành. Nếu thất bại, -1 được trả về.

Máy chủ hiện không phân biệt được giữa các chế độ INV_WRITE và INV_READ | INV_WRITE: bạn được phép đọc từ trình mô tả trong cả 2 trường hợp. Tuy nhiên có một khác biệt đáng kể giữa các chế độ đó và một mình INV_READ: với INV_READ bạn không thể ghi lên trình mô tả, và đọc dữ liệu từ nó sẽ phản ánh các nội dung của đối tượng lớn ở thời điểm chụp hình giao dịch mà từng tích cực khi lo_open từng được thực thi, bất kể các cuộc ghi sau này bởi điều này hoặc các giao dịch khác. Việc đọc từ một trình mô tả được mở bằng INV_WRITE trả về dữ liệu mà phản ánh tất cả các cuộc ghi các giao dịch được thực hiện khác cũng như các cuộc ghi giao dịch hiện hành. Điều này là tương tự với hành vi của các chế độ giao dịch SERIALIZABLE so với READ COMMITTED cho các lệnh SQL SELECT thông thường.

Một ví dụ:

```
inv_fd = lo_open(conn, inv_oid, INV_READ|INV_WRITE);
```

32.3.5. Ghi dữ liệu tới một đối tượng lớn

Hàm

```
int lo_write(PGconn *conn, int fd, const char *buf, size_t len);
```

ghi các byte len từ buf tới trình mô tả đối tượng lớn fd. Đối số fd phải được một lo_open trước đó trả về. Số các byte thực sự được ghi sẽ được trả về. Trong trường hợp có lỗi, thì giá trị trả về là âm.

32.3.6. Đọc dữ liệu từ một đối tượng lớn

Hàm

```
int lo_read(PGconn *conn, int fd, char *buf, size_t len);
```

đọc len byte từ trình mô tả đối tượng lớn fd vào buf. Đối số fd phải được một lo_open trước đó trả về. Số các byte thực sự được đọc sẽ được trả về. Trong trường hợp có lỗi, thì giá trị trả về là âm.

32.3.7. Tìm kiếm trong một đối tượng lớn

Để thay đổi vị trí đọc hoặc ghi hiện hành có liên quan tới một trình mô tả đối tượng lớn, hãy gọi

```
int lo_lseek(PGconn *conn, int fd, int offset, int whence);
```

Hàm này chuyển con trỏ vị trí hiện hành cho trình mô tả đối tượng lớn được xác định bằng fd đối với vị trí mới được offset chỉ định. Các giá trị hợp lệ đối với whence là SEEK_SET (tìm từ đầu đối tượng), SEEK_CUR (tìm từ vị trí hiện hành), và SEEK_END (tìm từ cuối đối tượng). Giá trị trả về là con trỏ vị trí mới, hoặc -1 nếu có lỗi.

32.3.8. Có được vị trì tìm kiếm của một đối tượng lớn

Để có được vị trí đọc và ghi hiện hành của một trình mô tả đối tượng lớn, hãy gọi

```
int lo_tell(PGconn *conn, int fd);
```

Nếu có một lỗi thì giá trị trả về là âm.

32.3.9. Cắt bớt một đối tượng lớn

Để cắt bớt một đối tượng lớn về một độ dài cần thiết, hãy gọi

```
int lo_truncate(PGcon *conn, int fd, size_t len);
```

cắt bớt trình mô tả đối tượng lớn fd về độ dài len. Đối số fd phải được một lo_open trước đó trả về rồi. Nếu len là lớn hơn độ dài đối tượng lớn hiện hành, thì đối tượng lớn đó được mở rộng với các byte null ('\0').

Phần bù trừ tệp không bị thay đổi.

Nếu thành công thì lo_truncate trả về zero. Nếu có lỗi, giá trị trả về là âm.

lo_truncate là mới đối với PostgreSQL 8.3; nếu hàm này được chạy đối với một phiên bản máy chủ cũ hơn, thì nó sẽ thất bại và trả về một giá trị âm.

32.3.10. Đóng một trình mô tả đối tượng lớn

Một trình mô tả đối tượng lớn có thể được đóng bằng việc gọi

```
int lo_close(PGconn *conn, int fd);
```

trong đó fd là một trình mô tả đối tượng lớn được lo_open trả về. Nếu thành công, lo_close trả về zero. Nếu có lỗi, giá trị trả về là âm.

Bất kỳ trình mô tả đối tượng lớn nào mà vẫn mở ở cuối của một giao dịch sẽ được đóng lại tự động.

32.3.11. Loại bỏ một đối tượng lớn

Để loại bỏ một đối tượng lớn khỏi cơ sở dữ liệu, hãy gọi

```
int lo_unlink(PGconn *conn, Oid lobjId);
```

Đối số lobjId chỉ định OID của đối tượng lớn để loại bỏ. Trả về 1 nếu thành công, -1 nếu thất bại.

32.4. Hàm phía máy chủ

Có các hàm phía máy chủ có khả năng gọi từ SQL mà tương ứng với từng trong số các hàm phía máy trạm được mô tả ở trên; quả thực, đối với hầu hết các phần của các hàm phía máy trạm là các giao diện đơn giản đối với các hàm tương ứng phía máy chủ. Các hàm đó thực sự là hữu dụng để gọi qua các lệnh SQL là lo_creat, lo_create, lo_unlink, lo_import, và lo_export. Đây là các ví dụ về sử

dụng chúng:

```
CREATE TABLE image (
        name    text,
        raster  oid
);
SELECT lo_creat(-1);                    -- returns OID of new, empty large object
SELECT lo_create(43213);                -- attempts to create large object with OID 43213
SELECT lo_unlink(173454);               -- deletes large object with OID 173454
INSERT INTO image (name, raster)
        VALUES ('beautiful image', lo_import('/etc/motd'));
INSERT INTO image (name, raster)        -- same as above, but specify OID to use
        VALUES ('beautiful image', lo_import('/etc/motd', 68583));
SELECT lo_export(image.raster, '/tmp/motd') FROM image
        WHERE name = 'beautiful image';
```

Các hàm phía máy chủ lo_import và lo_export hành xử khác nhau đáng kể từ các tương ứng của chúng ở phía máy trạm. 2 hàm đó đọc và ghi các tệp vào hệ thống tệp của máy chủ, sử dụng các quyền của người sử dụng sở hữu cơ sở dữ liệu đó. Vì thế, sử dụng của chúng được hạn chế cho các siêu người sử dụng. Ngược lại, các hàm xuất và nhập khẩu phía máy trạm đọc và ghi các tệp vào hệ thống tệp máy trạm, bằng việc sử dụng các quyền của chương trình máy trạm. Các hàm phía máy trạm không yêu cầu quyền ưu tiên của siêu người sử dụng.

32.5. Chương trình ví dụ

Ví dụ 32-1 là một chương trình mẫu chỉ ra cách mà giao diện đối tượng lớn trong libpq có thể được sử dụng. Các phần của chương trình đó được bình luận nhưng để lại trong nguồn vì lợi ích của độc giả. Chương trình này cũng có thể được thấy trong src/test/examples/testlo.c trong phân phối nguồn.

Ví dụ 32-1. Các đối tượng lớn với Chương trình Ví dụ libpq

```
/*--------------------------------------------------------------
*
* testlo.c--
*       test using large objects with libpq
*
* Copyright (c) 1994, Regents of the University of California
*
*--------------------------------------------------------------
*/
#include <stdio.h>
#include "libpq-fe.h"
#include "libpq/libpq-fs.h"

#define BUFSIZE 1024

/*
* importFile
*       import file "in_filename" into database as large object "lobjOid"
*
*/
        Oid
        importFile(PGconn *conn, char *filename)
        {
        Oid     lobjId;
        int     lobj_fd;
        char    buf[BUFSIZE];
        int     nbytes,
```

```
                tmp;
        int     fd;
        /*
        * open the file to be read in
        */
        fd = open(filename, O_RDONLY, 0666);
        if (fd < 0)
        {                                               /* error */
                fprintf(stderr, "cannot open unix file %s\n", filename);
        }
        /*
        * create the large object
        */
        lobjId = lo_creat(conn, INV_READ | INV_WRITE);
        if (lobjId == 0)
                fprintf(stderr, "cannot create large object\n");
        lobj_fd = lo_open(conn, lobjId, INV_WRITE);
        /*
        * read in from the Unix file and write to the inversion file
        */
        while ((nbytes = read(fd, buf, BUFSIZE)) > 0)
        {
                tmp = lo_write(conn, lobj_fd, buf, nbytes);
                if (tmp < nbytes)
                        fprintf(stderr, "error while reading large object\n");
        }
        (void) close(fd);
        (void) lo_close(conn, lobj_fd);
        return lobjId;
}
void
pickout(PGconn *conn, Oid lobjId, int start, int len)
{
        int     lobj_fd;
        char    *buf;
        int     nbytes;
        int     nread;

        lobj_fd = lo_open(conn, lobjId, INV_READ);
        if (lobj_fd < 0)
        {
                fprintf(stderr, "cannot open large object %d\n",
                        lobjId);
        }
        lo_lseek(conn, lobj_fd, start, SEEK_SET);
        buf = malloc(len + 1);
        nread = 0;
        while (len - nread > 0)
        {
                nbytes = lo_read(conn, lobj_fd, buf, len - nread);
                buf[nbytes] = ' ';
                fprintf(stderr, ">>> %s", buf);
                nread += nbytes;
        }
        free(buf);
        fprintf(stderr, "\n");
        lo_close(conn, lobj_fd);
}
void
overwrite(PGconn *conn, Oid lobjId, int start, int len)
{
        int     lobj_fd;
```

```
        char    *buf;
        int     nbytes;
        int     nwritten;
        int     i;

        lobj_fd = lo_open(conn, lobjId, INV_WRITE);
        if (lobj_fd < 0)
        {
                fprintf(stderr, "cannot open large object %d\n",
                        lobjId);
        }
        lo_lseek(conn, lobj_fd, start, SEEK_SET);
        buf = malloc(len + 1);
        for (i = 0; i < len; i++)
                buf[i] = 'X';
        buf[i] = ' ';
        nwritten = 0;
        while (len - nwritten > 0)
        {
                nbytes = lo_write(conn, lobj_fd, buf + nwritten, len - nwritten);
                nwritten += nbytes;
        }
        free(buf);
        fprintf(stderr, "\n");
        lo_close(conn, lobj_fd);
}
/*
* exportFile
* export large object "lobjOid" to file "out_filename"
*
*/
void
exportFile(PGconn *conn, Oid lobjId, char *filename)
{
        int     lobj_fd;
        char    buf[BUFSIZE];
        int     nbytes,
                tmp;
        int     fd;
        /*
        * open the large object
        */
        lobj_fd = lo_open(conn, lobjId, INV_READ);
        if (lobj_fd < 0)
        {
                fprintf(stderr, "cannot open large object %d\n",
                        lobjId);
        }
        /*
        * open the file to be written to
        */
        fd = open(filename, O_CREAT | O_WRONLY, 0666);
        if (fd < 0)
        { /* error */
                fprintf(stderr, "cannot open unix file %s\n",
                        filename);
        }
        /*
        * read in from the inversion file and write to the Unix file
        */
        while ((nbytes = lo_read(conn, lobj_fd, buf, BUFSIZE)) > 0)
        {
```

```
                tmp = write(fd, buf, nbytes);
                if (tmp < nbytes)
                {
                        fprintf(stderr, "error while writing %s\n",
                                filename);
                }
        }
        (void) lo_close(conn, lobj_fd);
        (void) close(fd);
        return;
}
void
exit_nicely(PGconn *conn)
{
        PQfinish(conn);
        exit(1);
}
int
main(int argc, char **argv)
{
        char            *in_filename,
                        *out_filename;
        char            *database;
        Oid              lobjOid;
        PGconn          *conn;
        PGresult        *res;

        if (argc != 4)
        {
                fprintf(stderr, "Usage: %s database_name in_filename out_filename\n",
                        argv[0]);
                exit(1);
        }
        database = argv[1];
        in_filename = argv[2];
        out_filename = argv[3];
        /*
        * set up the connection
        */
        conn = PQsetdb(NULL, NULL, NULL, NULL, database);
        /* check to see that the backend connection was successfully made */
        if (PQstatus(conn) == CONNECTION_BAD)
        {
                fprintf(stderr, "Connection to database '%s' failed.\n", database);
                fprintf(stderr, "%s", PQerrorMessage(conn));
                exit_nicely(conn);
        }
        res = PQexec(conn, "begin");
        PQclear(res);
        printf("importing file %s\n", in_filename);
/*      lobjOid = importFile(conn, in_filename); */
        lobjOid = lo_import(conn, in_filename);
/*
        printf("as large object %d.\n", lobjOid);
        printf("picking out bytes 1000-2000 of the large object\n");
        pickout(conn, lobjOid, 1000, 1000);
        printf("overwriting bytes 1000-2000 of the large object with X's\n");
        overwrite(conn, lobjOid, 1000, 1000);
*/
        printf("exporting large object to file %s\n", out_filename);
/*              exportFile(conn, lobjOid, out_filename); */
        lo_export(conn, lobjOid, out_filename);
```

```
        res = PQexec(conn, "end");
        PQclear(res);
        PQfinish(conn);
        exit(0);
}
```

Chương 33. ECPG - Nhúng SQL vào C

Chương này mô tả gói SQL nhúng cho PostgreSQL. Nó từng được Linus Tolke (<linus@epact.se>) và Michael Meskes (<meskes@postgresql.org>) viết. Ban đầu nó từng được viết để làm việc với C. Nó cũng làm việc được với C++, nhưng nó còn chưa thừa nhận tất cả các cấu trúc C++. Tài liệu này hoàn toàn chưa hoàn thiện. Nhưng vì giao diện này được tiêu chuẩn hóa, thông tin bổ sung có thể tìm thấy trong nhiều nguồn về SQL.

33.1. Khái niệm

Chương trình SQL nhúng gồm mã được viết trong một ngôn ngữ lập trình thông thường, trong trường hợp này là C, được trộn với các lệnh SQL trong các phần được đánh dấu đặc biệt. Để xây dựng chương trình đó, mã nguồn trước hết được truyền qua trình tiền xử lý SQL nhúng, làm biến đổi nó thành chương trình C thông thường, và sau đó nó có thể được một trình biên dịch C xử lý.

SQL nhúng có các ưu thế so với các phương pháp khác điều khiển các lệnh từ mã C. Trước hết, nó chăm sóc việc truyền nặng nhọc thông tin tới và từ các biến trong chương trình C của bạn. Thứ 2, mã SQL trong chương trình được kiểm tra ở thời điểm xây dựng cho việc chỉnh cho đúng cú pháp. Thứ 3, SQL nhúng trong C được chỉ định theo tiêu chuẩn SQL và được nhiều hệ thống cơ sở dữ liệu SQL khác hỗ trợ. Triển khai PostgreSQL được thiết kế để khớp với tiêu chuẩn này càng nhiều có thể càng tốt, và thường có khả năng để chuyển các chương trình SQL được viết cho các cơ sở dữ liệu SQL khác sang PostgreSQL khá dễ dàng.

Như đã nêu, các chương trình được viết cho giao diện SQL nhúng là các chương trình C thông thường với mã đặc biệt được chèn vào để thực hiện các hành động có liên quan tới cơ sở dữ liệu. Mã đặc biệt này luôn có dạng:

```
EXEC SQL ...;
```

Các lệnh đó về cú pháp diễn ra theo một lệnh C. Phụ thuộc vào lệnh đặc biệt, chúng có thể xuất hiện ở mức toàn cầu hoặc bên trong một hàm. Các lệnh SQL nhúng tuân theo các qui tắc phân biệt chữ hoa chữ thường của mã SQL thông thường, và không theo các qui tắc của C.

Các phần sau đây giải thích tất cả các lệnh SQL nhúng.

33.2. Kết nối tới máy chủ cơ sở dữ liệu

Người ta kết nối tới một cơ sở dữ liệu bằng việc sử dụng lệnh sau đây:

```
EXEC SQL CONNECT TO target [AS connection-name] [USER user-name];
```

target có thể được chỉ định theo các cách thức sau:

- dbname[@hostname][:port]
- tcp:postgresql://hostname[:port][/dbname][?options]
- unix:postgresql://hostname[:port][/dbname][?options]
- một hằng chuỗi SQL có một trong các dạng ở trên
- một tham chiếu tới một biến ký tự chứa một trong các dạng ở trên (xem các ví dụ)
- mặc định - DEFAULT

Nếu bạn chỉ định đích kết nối theo nghĩa đen (đó là, không qua một tham chiếu biến) và bạn không trích dẫn giá trị đó, thì các qui tắc phân biệt chữ hoa chữ thường của SQL thông thường sẽ được áp dụng. Trong trường hợp đó bạn cũng có thể nháy đúp các tham số riêng rẽ một cách tách bạch như cần thiết. Trong thực tiễn, có khả năng ít xu hướng lỗi để sử dụng một hằng chuỗi (được nháy đơn) hoặc một tham chiếu biến. Đích kết nối DEFAULT khởi tạo một kết nối tới cơ sở dữ liệu mặc định dưới cái tên người sử dụng mặc định. Không tên người sử dụng hoặc tên kết nối tách bạch nào có thể được chỉ định trong trường hợp đó.

Cũng có các cách thức khác để chỉ định tên người sử dụng:

- username
- username/password
- username IDENTIFIED BY password
- username USING password

Như ở trên, các tham số username và password có thể là một mã định danh SQL, một hằng chuỗi SQL, hoặc một tham chiếu tới một biến ký tự.

connection-name được sử dụng để điều khiển nhiều kết nối trong một chương trình. Nó có thể bị bỏ qua nếu một chương trình chỉ sử dụng một kết nối. Kết nối được mở gần đây nhất trở thành kết nối hiện hành, điều được sử dụng mặc định khi một lệnh SQL sẽ được thực thi (xem sau ở chương này).

Đây là một vài ví dụ về các lệnh kết nối CONNECT:

```
EXEC SQL CONNECT TO mydb@sql.mydomain.com;
EXEC SQL CONNECT TO unix:postgresql://sql.mydomain.com/mydb AS myconnection USER john;
EXEC SQL BEGIN DECLARE SECTION;
const char *target = "mydb@sql.mydomain.com";
const char *user = "john";
EXEC SQL END DECLARE SECTION;
...
EXEC SQL CONNECT TO :target USER :user;
```

Dạng cuối cùng sử dụng phương án được tham chiếu tới ở trên như là tham chiếu biến ký tự. Bạn sẽ thấy trong các phần sau cách mà các biến C có thể được sử dụng trong các lệnh SQL khi bạn đặt tiền tố cho chúng với một dấu hai chấm (:).

Được khuyến cáo rằng định dạng đích kết nối không được chỉ định trong tiêu chuẩn SQL. Nên nếu bạn muốn phát triển các ứng dụng khả chuyển, thì bạn có lẽ phải sử dụng thứ gì đó dựa vào ví dụ cuối cùng ở trên để bọc chuỗi đích kết nối ở đâu đó.

33.3. Đóng một kết nối

Để đóng một kết nối, hãy sử dụng lệnh sau:

```
EXEC SQL DISCONNECT [connection];
```

connection có thể được chỉ định theo các cách thức sau:

- connection-name
- DEFAULT
- CURRENT
- ALL

Nếu không tên kết nối nào được chỉ định, thì kết nối hiện hành được đóng.

Là kiểu tốt lành rằng một ứng dụng luôn bỏ kết nối rõ ràng từ bất kỳ kết nối nào mà nó đã mở.

33.4. Chạy các lệnh SQL

Bất kỳ lệnh SQL nào cũng có thể được chạy từ bên trong một ứng dụng SQL nhúng. Bên dưới là một vài ví dụ về cách làm điều đó.

Tạo một bảng:

```
EXEC SQL CREATE TABLE foo (number integer, ascii char(16));
EXEC SQL CREATE UNIQUE INDEX num1 ON foo(number);
EXEC SQL COMMIT;
```

Chèn các hàng:

```
EXEC SQL INSERT INTO foo (number, ascii) VALUES (9999, 'doodad');
EXEC SQL COMMIT;
```

Xóa các hàng:

```
EXEC SQL DELETE FROM foo WHERE number = 9999;
EXEC SQL COMMIT;
```

Chọn hàng duy nhất:

```
EXEC SQL SELECT foo INTO :FooBar FROM table1 WHERE ascii = 'doodad';
```

Chọn các con trỏ đang sử dụng:

```
EXEC SQL DECLARE foo_bar CURSOR FOR
        SELECT number, ascii FROM foo
        ORDER BY ascii;
EXEC SQL OPEN foo_bar;
EXEC SQL FETCH foo_bar INTO :FooBar, DooDad;
...
EXEC SQL CLOSE foo_bar;
EXEC SQL COMMIT;
```

Cập nhật:

```
EXEC SQL UPDATE foo
        SET ascii = 'foobar'
        WHERE number = 9999;
EXEC SQL COMMIT;
```

Các thẻ token của dạng: `something` là các biến host, đó là, chúng tham chiếu tới các biến trong chương trình C. Chúng được giải thích trong Phần 33.6.

Ở chế độ mặc định, các lệnh chỉ được thực hiện khi `EXEC SQL COMMIT` được đưa ra. Giao diện SQL nhúng cũng hỗ trợ thực hiện tự động các giao dịch (tương tự như hành vi của `libpq`) thông qua lựa chọn dòng lệnh `-t` đối với `ecpg` (xem bên dưới) hoặc qua lệnh `EXEC SQL SET AUTOCOMMIT TO ON`. Ở chế độ mặc định, từng lệnh được thực hiện tự động trừ phi nó là ở trong một khối giao dịch rõ ràng. Chế độ này có thể được tắt rõ ràng bằng việc sử dụng `EXEC SQL SET AUTOCOMMIT TO OFF`.

33.5. Chọn kết nối

Các lệnh SQL được chỉ ra trong phần trước được thực thi trong kết nối hiện hành, đó là, các kết nối được mở gần đây nhất. Nếu một ứng dụng cần quản lý nhiều kết nối, thì có 2 cách để làm điều này.

Lựa chọn đầu là chọn rõ ràng một kết nối cho từng lệnh SQL, ví dụ:

```
EXEC SQL AT connection-name SELECT ...;
```

Lựa chọn này là đặc biệt phù hợp nếu ứng dụng đó cần sử dụng vài kết nối theo trật tự pha trộn.

Nếu ứng dụng của bạn sử dụng nhiều luồng thực thi, chúng không thể chia sẻ một kết nối đồng thời.

Bạn phải hoặc rõ ràng kiểm soát truy cập tới kết nối đó (sử dụng mutexes) hoặc sử dụng một kết nối cho từng luồng. Nếu từng luồng sử dụng kết nối của riêng nó, thì bạn sẽ cần sử dụng mệnh đề AT để chỉ định kết nối nào luồng đó sẽ sử dụng.

Lựa chọn thứ 2 là thực thi một lệnh để chuyển kết nối hiện hành. Lệnh đó là:

```
EXEC SQL SET CONNECTION connection-name;
```

Lựa chọn này đặc biệt là thuận tiện nếu nhiều lệnh sẽ được thực thi trong cùng y hệt một kết nối. Nó không phải là nhận biết luồng.

33.6. Sử dụng các biến host

Trong Phần 33.4 bạn đã thấy cách có thể thực thi các lệnh SQL từ một chương trình SQL nhúng. Một vài lệnh đó đã chỉ sử dụng các giá trị cố định và đã không đưa ra cách thức để chèn các giá trị được người sử dụng cung cấp vào các lệnh hoặc nhờ chương trình đó xử lý các giá trị được truy vấn đó trả về. Các dạng lệnh đó thực sự không hữu dụng trong các ứng dụng thực tế. Phần này giải thích chi tiết cách bạn có thể truyền dữ liệu giữa chương trình C của bạn và các lệnh SQL nhúng bằng việc sử dụng một cơ chế đơn giản được gọi là các biến host. Trong một chương trình SQL nhúng thì chúng ta cân nhắc các lệnh SQL sẽ là khách (guest) trong mã chương trình C mà là ngôn ngữ chủ. Vì thế các biến của chương trình C được gọi là các biến chủ (host).

33.6.1. Tổng quan

Việc truyền dữ liệu giữa chương trình C và các lệnh SQL đặc biệt là đơn giản trong SQL nhúng. Thay vì nhờ chương trình dán dữ liệu vào lệnh, điều kéo theo nhiều sự phức tạp, như việc trích đúng giá trị, bạn có thể đơn giản viết tên của một biến C vào lệnh SQL, có tiền tố là một dấu hai chấm, ví dụ:

```
EXEC SQL INSERT INTO sometable VALUES (:v1, 'foo', :v2);
```

Các lệnh này tham chiếu tới 2 biến C có tên là v1 và v2 và cũng sử dụng một hằng chuỗi SQL thông thường, để minh họa rằng bạn không bị giới hạn phải sử dụng một dạng dữ liệu này hay khác.

Kiểu chèn này các biến C vào các lệnh SQL làm việc ở bất kỳ đâu một biểu thức giá trị được kỳ vọng trong một lệnh SQL.

33.6.2. Khai báo các phần

Để truyền dữ liệu từ chương trình tới cơ sở dữ liệu, ví dụ như các tham số trong một truy vấn, hoặc truyền dữ liệu từ cơ sở dữ liệu ngược về chương trình đó, các biến C có ý định chứa dữ liệu này cần

phải được khai báo trong các phần được đánh dấu đặc biệt, sao cho trình tiền xử lý SQL nhúng nhận thức được về chúng.

Phần này bắt đầu với:

```
EXEC SQL BEGIN DECLARE SECTION;
```

và kết thúc với:

```
EXEC SQL END DECLARE SECTION;
```

Giữa các dòng đó, phải có các khai báo biến C thông thường, như:

```
int x = 4;
char foo[16], bar[16];
```

Như bạn thấy, bạn có thể tùy ý chỉ định một giá trị ban đầu cho biến đó. Phạm vi của biến được xác định bằng vị trí của phần khai báo của nó trong chương trình. Bạn cũng có thể khai báo các biến với cú pháp sau, điều ngầm tạo ra một phần khai báo:

```
EXEC SQL int i = 4;
```

Bạn có thể có bao nhiêu phần khai báo trong chương trình là tùy ý bạn.

Các khai báo cũng được dội tới tệp đầu ra như các biến C thông thường, nên không cần thiết phải khai báo chúng một lần nữa. Các biến có ý định sẽ được sử dụng trong các lệnh SQL có thể được khai báo thông thường bên ngoài các phần đặc biệt đó.

Định nghĩa một cấu trúc hoặc liên minh phải được liệt kê bên trong phần DECLARE. Nếu không thì trình tiền xử lý không thể điều khiển các dạng đó vì nó không biết định nghĩa đó.

33.6.3. Các dạng khác nhau của các biến chủ host

Như biến chủ host bạn cũng có thể sử dụng các mảng, các typedefs, các cấu trúc và các con trỏ. Hơn nữa có các dạng đặc biệt các biến chủ host mà tồn tại chỉ trong ECPG.

Một ít ví dụ về các biến chủ host:

Array - Mảng

Một trong những sử dụng phổ biến nhất một khai báo mảng có lẽ là sự phân bổ một mảng ký tự như trong:

```
EXEC SQL BEGIN DECLARE SECTION;
        char str[50];
EXEC SQL END DECLARE SECTION;
```

Lưu ý rằng bạn phải chăm sóc độ dài cho chính bạn. Nếu bạn sử dụng biến chủ host này như là biến đích của một truy vấn mà trả về một chuỗi với hơn 49 ký tự, thì một sự tràn bộ nhớ tạm sẽ xảy ra.

Typedefs

Sử dụng từ khóa typedef để ánh xạ các dạng mới tới các dạng đang tồn tại rồi.

```
EXEC SQL BEGIN DECLARE SECTION;
        typedef char mychartype[40];
        typedef long serial_t;
EXEC SQL END DECLARE SECTION;
```

Lưu ý rằng bạn cũng có thể sử dụng:

```
EXEC SQL TYPE serial_t IS long;
```

Khai báo này không cần là một phần của một phần khai báo.

Pointers - Các con trỏ

Bạn có thể khai báo các con trỏ tới các dạng phổ biến nhất. Tuy nhiên hãy lưu ý rằng bạn không thể sử dụng các con trỏ như các biến đích của các truy vấn mà không có phân bổ tự động. Xem Phần 33.9 để có thêm thông tin về phân bổ tự động.

```
EXEC SQL BEGIN DECLARE SECTION;
        int *intp;
        char **charp;
EXEC SQL END DECLARE SECTION;
```

Các dạng đặc biệt của các biến

ECPG có vài dạng đặc biệt giúp bạn tương tác dễ dàng với các dữ liệu từ máy chủ SQL.

Ví dụ nó đã triển khai hỗ trợ cho các dạng varchar, numeric, date, timestamp, và interval. Phần 33.8 có các hàm cơ bản để làm việc với các dạng đó, như những hàm bạn không cần phải gửi một truy vấn tới máy chủ SQL chỉ để thêm một khoảng thời gian tới một dấu thời gian, ví dụ thế.

Dạng đặc biệt VARCHAR được biến đổi thành một cấu trúc struct có tên đối với mọi biến. Một khai báo như:

```
VARCHAR var[180];
```

sẽ được biến đổi thành:

```
struct varchar_var { int len; char arr[180]; } var;
```

Cấu trúc này là phù hợp cho việc giao diện với các dữ liệu SQL dạng varchar.

33.6.4. SELECT INTO và FETCH INTO

Bây giờ bạn nên có khả năng truyền dữ liệu được chương trình của bạn sinh ra vào một lệnh SQL. Nhưng làm thế nào truy xuất được các dữ liệu của một truy vấn? Vì mục đích đó, SQL nhúng cung cấp các phương án đặc biệt các lệnh thường dùng SELECT và FETCH. Các lệnh đó có một mệnh đề đặc biệt INTO mà chỉ định các biến chủ host nào các giá trị được truy xuất sẽ được lưu trữ trong đó.

Đây là một ví dụ:

```
/*
* assume this table:
* CREATE TABLE test1 (a int, b varchar(50));
*/
EXEC SQL BEGIN DECLARE SECTION;
int v1;
VARCHAR v2;
EXEC SQL END DECLARE SECTION;
...
EXEC SQL SELECT a, b INTO :v1, :v2 FROM test;
```

Nên mệnh đề INTO xuất hiện giữa danh sách chọn và mệnh đề FROM. Số yếu tố trong danh sách chọn và danh sách sau INTO (cũng được gọi là danh sách đích) phải bằng nhau.

Đây là một ví dụ có sử dụng lệnh FETCH:

```
EXEC SQL BEGIN DECLARE SECTION;
int v1;
VARCHAR v2;
EXEC SQL END DECLARE SECTION;
...
EXEC SQL DECLARE foo CURSOR FOR SELECT a, b FROM test;
...
do {
        ...
        EXEC SQL FETCH NEXT FROM foo INTO :v1, :v2;
        ...
} while (...);
```

Đây là mệnh đề INTO xuất hiện sau tất cả các mệnh đề thông thường.

33.6.5. Các chỉ số

Các ví dụ ở trên không điều khiển các giá trị null. Trong thực tế, các ví dụ truy xuất được sẽ làm nảy sinh một lỗi nếu chúng lấy về một giá trị null từ cơ sở dữ liệu. Để có khả năng truyền các giá trị null tới cơ sở dữ liệu hoặc truy xuất các giá trị null từ cơ sở dữ liệu, bạn cần nối thêm một đặc tả biến chủ host thứ 2 vào từng biến chủ host có chứa các dữ liệu. Biến chủ host thứ 2 này được gọi là chỉ số và gồm một cờ nói liệu các dữ liệu có là null hay không, trong trường hợp đó giá trị của biến chủ host thực tế bị bỏ qua. Đây là một ví dụ điều khiển sự truy xuất các giá trị null đúng đắn:

```
EXEC SQL BEGIN DECLARE SECTION;
VARCHAR val;
int val_ind;
EXEC SQL END DECLARE SECTION:
EXEC SQL SELECT b INTO :val :val_ind FROM test1;
```

Biến chỉ số val_ind sẽ là zero nếu giá trị đó từng khác null, và nó sẽ là âm nếu giá trị đó từng là null.

Chỉ số đó có chức năng khác: nếu giá trị chỉ số là dương, có nghĩa là giá trị đó khác null, nhưng nó đã bị cắt bớt khi nó được lưu trữ trong biến chủ host.

33.7. SQL động

Trong nhiều trường hợp, các lệnh SQL đặc biệt mà một ứng dụng phải thực thi được biết vào lúc ứng dụng đó được viết. Tuy nhiên, trong một số trường hợp, các lệnh SQL được sáng tác ở lúc chạy hoặc được một nguồn bên ngoài cung cấp. Trong các trường hợp đó bạn không thể nhúng các lệnh SQL trực tiếp vào mã nguồn C, nhưng có một tiện ích mà cho phép bạn gọi các lệnh SQL tùy ý mà bạn đưa ra trong một biến chuỗi.

Cách đơn giản nhất để thực thi lệnh SQL tùy thích là hãy sử dụng lệnh EXECUTE IMMEDIATE. Ví dụ:

```
EXEC SQL BEGIN DECLARE SECTION;
const char *stmt = "CREATE TABLE test1 (...);";
EXEC SQL END DECLARE SECTION;

EXEC SQL EXECUTE IMMEDIATE :stmt;
```

Bạn không thể thực thi các lệnh mà truy xuất dữ liệu (như, SELECT) theo cách này.

Một cách mạnh mẽ hơn để thực thi các lệnh SQL tùy ý là chuẩn bị cho chúng một lần và thực thi lệnh được chuẩn bị đó thường xuyên bao nhiêu tùy ý bạn. Cũng có khả năng để chuẩn bị một phiên bản được tổng quát hóa của một lệnh và sau đó thực thi các phiên bản đặc thù của nó bằng việc thay

thế các tham số. Khi chuẩn bị lệnh đó, hãy viết các dấu hỏi ở những nơi bạn muốn thay thế các tham số sau này. Ví dụ:

```
EXEC SQL BEGIN DECLARE SECTION;
const char *stmt = "INSERT INTO test1 VALUES(?, ?);";
EXEC SQL END DECLARE SECTION;

EXEC SQL PREPARE mystmt FROM :stmt;
...
EXEC SQL EXECUTE mystmt USING 42, 'foobar';
```

Nếu lệnh bạn đang thực thi trả về các giá trị, thì thêm một mệnh đề INTO:

```
EXEC SQL BEGIN DECLARE SECTION;
const char *stmt = "SELECT a, b, c FROM test1 WHERE a > ?";
int v1, v2;
VARCHAR v3;
EXEC SQL END DECLARE SECTION;

EXEC SQL PREPARE mystmt FROM :stmt;
...
EXEC SQL EXECUTE mystmt INTO v1, v2, v3 USING 37;
```

Một lệnh EXECUTE có thể có một mệnh đề INTO, một mệnh đề USING, cả hai, hoặc không cả hai.

Khi bạn không cần lệnh được chuẩn bị trước nữa, bạn sẽ bỏ phân bổ nó:

```
EXEC SQL DEALLOCATE PREPARE name;
```

33.8. Thư viện pgtypes

Thư viện pgtypes ánh xạ các dạng cơ sở dữ liệu PostgreSQL tới những tương đương của C mà có thể được sử dụng trong các chương trình C. Nó cũng đưa ra các hàm thực hiện các tính toán cơ bản với các dạng đó bên trong C, như, không có sự trợ giúp của máy chủ PostgreSQL. Xem ví dụ sau:

```
EXEC SQL BEGIN DECLARE SECTION;
        date date1;
        timestamp ts1, tsout;
        interval iv1;
        char *out;
EXEC SQL END DECLARE SECTION;

PGTYPESdate_today(&date1);
EXEC SQL SELECT started, duration INTO :ts1, :iv1 FROM datetbl WHERE d=:date1;
PGTYPEStimestamp_add_interval(&ts1, &iv1, &tsout);
out = PGTYPEStimestamp_to_asc(&tsout);
printf("Started + duration: %s\n", out);
free(out);
```

33.8.1. Dạng số

Dạng số đưa ra để thực hiện các tính toán với độ chính xác tùy ý. Xem Phần 8.1 để có dạng tương đương trong máy chủ PostgreSQL. Vì độ chính xác tùy ý đó nên biến này cần phải có khả năng mở rộng và thu hẹp một cách động. Điều đó giải thích vì sao bạn chỉ có thể tạo các biến số trong một đống, với công cụ của các hàm PGTYPESnumeric_new và PGTYPESnumeric_free. Dạng thập phân, điều

tương tự nhưng có hạn chế về độ chính xác, có thể được tạo ra trong kho cũng như trong đống.

Các hàm sau đây có thể được sử dụng để làm việc với dạng số:

PGTYPESnumeric_new

Đòi hỏi một con trỏ tới một biến số mới được phân bổ.

```
numeric *PGTYPESnumeric_new(void);
```

PGTYPESnumeric_free

Giải phóng một dạng số, thả tất cả bộ nhớ của nó.

```
void PGTYPESnumeric_free(numeric *var);
```

PGTYPESnumeric_from_asc

Phân tích cú pháp một dạng số từ ký hiệu chuỗi của nó.

```
numeric *PGTYPESnumeric_from_asc(char *str, char **endptr);
```

Các định dạng hợp lệ là như ví dụ: -2, .794, +3.44, 592.49E07 hoặc -32.84e-4. Nếu giá trị đó có thể được phân tích cú pháp thành công, thì một con trỏ hợp lệ được trả về, nếu không thì con trỏ NULL. Hiện hành ECPG luôn phân tích cú pháp chuỗi hoàn chỉnh và vì thế nó hiện không hỗ trợ để lưu trữ địa chỉ ký tự không hợp lệ đầu tiên trong *endptr. Bạn có thể thiết lập an toàn endptr về NULL.

PGTYPESnumeric_to_asc

Trả về một con trỏ cho một chuỗi được phân bổ bởi malloc mà có chứa đại diện chuỗi dạng số num.

```
char *PGTYPESnumeric_to_asc(numeric *num, int dscale);
```

Giá trị số sẽ được in với các số thập phân dscale, với việc làm tròn số áp dụng nếu cần.

PGTYPESnumeric_add

Thêm 2 biến số vào một biến số thứ 3.

```
int PGTYPESnumeric_add(numeric *var1, numeric *var2, numeric *result);
```

Hàm này thêm các biến var1 và var2 vào biến kết quả result. Hàm trả về 0 nếu thành công và -1 nếu có lỗi.

PGTYPESnumeric_sub

Trừ 2 biến số và trả về kết quả trong một biến số thứ 3.

```
int PGTYPESnumeric_sub(numeric *var1, numeric *var2, numeric *result);
```

Hàm này trừ biến var2 từ biến var1. Kết quả của hành động đó được lưu trữ trong biến result. Hàm trả về 0 nếu thành công và -1 nếu có lỗi.

PGTYPESnumeric_mul

Nhân 2 biến số và trả về kết quả trong biến số thứ 3.

```
int PGTYPESnumeric_mul(numeric *var1, numeric *var2, numeric *result);
```

Hàm này nhân các biến var1 và var2. Kết quả của hành động được lưu trữ trong biến result. Hàm trả về 0 nếu thành công và -1 nếu có lỗi.

PGTYPESnumeric_div

Chia 2 biến số cho nhau và trả về kết quả trong một biến số thứ 3.

```
int PGTYPESnumeric_div(numeric *var1, numeric *var2, numeric *result);
```

Hàm này chia các biến var1 cho var2. Kết quả của hành động được lưu trữ trong biến result. Hàm trả về 0 nếu thành công và -1 nếu có lỗi.

PGTYPESnumeric_cmp

So sánh 2 biến số với nhau.

```
int PGTYPESnumeric_cmp(numeric *var1, numeric *var2)
```

Hàm này so sánh 2 biến số với nhau. Trong trường hợp có lỗi, INT_MAX được trả về. Nếu thành công, hàm trả về 1 trong 3 kết quả có khả năng:

- 1, nếu var1 lớn hơn var2
- -1, nếu var1 nhỏ hơn var2
- 0, nếu var1 bằng var2

PGTYPESnumeric_from_int

Biến đổi một biến int thành một biến số.

```
int PGTYPESnumeric_from_int(signed int int_val, numeric *var);
```

Hàm này chấp nhận một biến int dạng được ký và lưu trữ nó trong biến số var.

Nếu thành công, 0 được trả về và -1 nếu thất bại.

PGTYPESnumeric_from_long

Biến đổi một biến long int thành một biến số.

```
int PGTYPESnumeric_from_long(signed long int long_val, numeric *var);
```

Hàm này chấp nhận một biến long int dạng được ký và lưu trữ nó trong biến số var. Nếu thành công, 0 được trả về và -1 nếu được trả về nếu thất bại.

PGTYPESnumeric_copy

Sao chép một biến số vào một biến số khác.

```
int PGTYPESnumeric_copy(numeric *src, numeric *dst);
```

Hàm này sao chép giá trị của biến mà src trỏ tới vào biến mà dst trỏ tới. Nó trả về 0 nếu thành công và -1 nếu có lỗi.

PGTYPESnumeric_from_double

Biến đổi một biến dạng đúp double thành một biến dạng số.

```
int PGTYPESnumeric_from_double(double d, numeric *dst);
```

Hàm này chấp nhận một biến dạng đúp double và lưu trữ kết quả trong biến mà dst trỏ tới. Nó trả về 0 nếu thành công và -1 nếu có lỗi.

PGTYPESnumeric_to_double

Biến đổi một biến dạng số thành dạng đúp double.

```
int PGTYPESnumeric_to_double(numeric *nv, double *dp)
```

Hàm này biến đổi giá trị số từ biến mà nv trỏ tới thành biến đúp double mà dp trỏ tới. Nó trả về 0 nếu thành công và -1 nếu có lỗi, bao gồm cả sự tràn.

Trong sự tràn, biến tổng thể errno sẽ được thiết lập về PGTYPES_NUM_OVERFLOW bổ sung.

PGTYPESnumeric_to_int

Biến đổi một biến dạng số thành int.

```
int PGTYPESnumeric_to_int(numeric *nv, int *ip);
```

Hàm này biến đổi giá trị số từ biến mà nv trỏ tới thành biến số nguyên integer mà ip trỏ tới. Nó trả về 0 nếu thành công và -1 nếu có lỗi, bao gồm cả sự tràn.

Nếu tràn, biến tổng thể errno sẽ được thiết lập về PGTYPES_NUM_OVERFLOW bổ sung thêm.

PGTYPESnumeric_to_long

Biến đổi một biến dạng số thành dạng long.

```
int PGTYPESnumeric_to_long(numeric *nv, long *lp);
```

Hàm này biến đổi giá trị số từ biến mà nv trỏ tới thành biến long integer mà lp trỏ tới. Nó trả về 0 nếu thành công và -1 nếu có lỗi, bao gồm cả sự tràn.

Nếu tràn, thì biến tổng thể errno sẽ được thiết lập về PGTYPES_NUM_OVERFLOW bổ sung thêm.

PGTYPESnumeric_to_decimal

Biến đổi một biến dạng số thành thập phân.

```
int PGTYPESnumeric_to_decimal(numeric *src, decimal *dst);
```

Hàm này biến đổi giá trị số từ biến mà src trỏ tới thành biến thập phân mà dst trỏ tới. Nó trả về 0 nếu thành công và -1 nếu có lỗi, bao gồm cả tràn bộ nhớ.

Khi tràn, biến tổng thể errno sẽ được thiết lập bổ sung về PGTYPES_NUM_OVERFLOW.

PGTYPESnumeric_from_decimal

Biến đổi một biến dạng thập phân thành số.

```
int PGTYPESnumeric_from_decimal(decimal *src, numeric *dst);
```

Hàm biến đổi giá trị thập phân từ biến mà src trỏ tới trong biến số mà dst trỏ tới. Nó trả về 0 nếu thành công và -1 nếu có lỗi. Vì dạng thập phân được triển khai như một phiên bản có giới hạn của dạng số, sự tràn không thể xảy ra với biến đổi này.

33.8.2. Dạng ngày tháng

Dạng ngày tháng trong C cho phép các chương trình của bạn làm việc với dữ liệu dạng ngày tháng của SQL. Xem Phần 8.5 cho dạng tương đương trong máy chủ PostgreSQL.

Các hàm sau có thể được sử dụng để làm việc với dạng ngày tháng:

PGTYPESdate_from_timestamp

Trích một phần ngày tháng từ một dấu thời gian.

```
date PGTYPESdate_from_timestamp(timestamp dt);
```

Hàm nhận một dấu thời gian như là đối số duy nhất của nó và trả về phần ngày tháng được trích ra từ dấu thời gian này.

PGTYPESdate_from_asc

Phân tích cú pháp ngày tháng từ trình bày văn bản của nó.

```
date PGTYPESdate_from_asc(char *str, char **endptr);
```

Hàm nhận một chuỗi ký tự char* của C là str và một con trỏ tới một chuỗi ký tự char* của C endptr. Lúc này ECPG luôn phân tích cú pháp chuỗi hoàn chỉnh và vì thế nó hiện không hỗ trợ để lưu trữ địa chỉ của ký tự không hợp lệ đầu tiên trong *endptr. Bạn có thể an toàn thiết lập endptr về NULL.

Lưu ý rằng hàm này luôn giả thiết các ngày tháng được định dạng theo MDY và hiện không có biến để thay đổi điều đó bên trong ECPG.

Bảng 33-1 chỉ các định dạng đầu vào được phép.

Bảng 33-1. Các định dạng đầu vào hợp lệ cho PGTYPESdate_from_asc

Đầu vào	Kết quả
January 8, 1999	January 8, 1999
1999-01-08	January 8, 1999
1/8/1999	January 8, 1999
1/18/1999	January 18, 1999
01/02/03	February 1, 2003
1999-Jan-08	January 8, 1999
Jan-08-1999	January 8, 1999
08-Jan-1999	January 8, 1999
99-Jan-08	January 8, 1999
08-Jan-99	January 8, 1999
08-Jan-06	January 8, 2006
Jan-08-99	January 8, 1999
19990108	ISO 8601; January 8, 1999
990108	ISO 8601; January 8, 1999
1999.008	year and day of year
J2451187	Julian day
January 8, 99 BC	year 99 before the Common Era

PGTYPESdate_to_asc

Trả về đại diện văn bản của một biến ngày tháng.

```
char *PGTYPESdate_to_asc(date dDate);
```

Hàm này nhận ngày tháng dDate như là tham số duy nhất của nó. Nó sẽ có đầu ra ngày tháng ở dạng 1999-01-18, nghĩa là, theo định dạng YYYY-MM-DD.

PGTYPESdate_julmdy

Trích các giá trị đối với ngày, tháng và năm từ một biến dạng ngày tháng.

```
void PGTYPESdate_julmdy(date d, int *mdy);
```

Hàm này nhận ngày tháng d và một con trỏ tới một mảng 3 giá trị số nguyên mdy. Tên biến đó chỉ trật tự tuần tự: mdy[0] sẽ được thiết lập để có chứa số tháng, mdy[1] sẽ được thiết lập về giá trị của ngày và mdy[2] sẽ chứa năm.

PGTYPESdate_mdyjul

Tạo một giá trị ngày tháng từ một mảng 3 số nguyên mà chỉ định ngày, tháng và năm của ngày tháng đó.

```
void PGTYPESdate_mdyjul(int *mdy, date *jdate);
```

Hàm này nhận mảng 3 số nguyên (mdy) như là đối số đầu tiên của nó và như là đối số thứ 2 của nó một con trỏ tới một biến dạng ngày tháng sẽ giữ kết quả của hoạt động này.

PGTYPESdate_dayofweek

Trả về một số đại diện cho ngày của tuần đối với một giá trị ngày tháng.

```
int PGTYPESdate_dayofweek(date d);
```

Hàm này nhận biến ngày tháng d như là đối số duy nhất của nó và trả về một số nguyên mà chỉ ngày của tuần đối với ngày tháng đó.

- 0 - chủ nhật
- 1 - thứ hai
- 2 - thứ ba
- 3 - thứ tư
- 4 - thứ năm
- 5 - thứ sáu
- 6 - thứ bảy

PGTYPESdate_today

Có ngày hiện hành

```
void PGTYPESdate_today(date *d);
```

Hàm nhận một con trỏ tới một biến ngày tháng (d) mà nó thiết lập về ngày tháng hiện hành.

PGTYPESdate_fmt_asc

Biến đổi một biến dạng ngày tháng về đại diện văn bản của nó bằng việc sử dụng một mặt nạ định dạng.

```
int PGTYPESdate_fmt_asc(date dDate, char *fmtstring, char *outbuf);
```

Hàm này nhận ngày tháng để biến đổi (dDate), mặt nạ định dạng (fmtstring) và chuỗi mà sẽ nắm giữ đại diện văn bản của ngày tháng đó (outbuf).

Nếu thành công, 0 được trả về và một giá trị âm nếu có lỗi xảy ra.

Các hằng sau là các chỉ định trường bạn có thể sử dụng:

- dd - Số ngày của tháng.
- mm - Số tháng của năm.
- yy - Số năm như một số có 2 chữ số.
- yyyy - Số năm như một số có 4 chữ số.
- ddd - Tên của ngày (viết tắt).

- mmm - Tên của tháng (viết tắt).

Tất cả các ký tự khác sẽ được sao chép tỷ lệ 1:1 tới chuỗi đầu ra.

Bảng 33-2 chỉ ra một ít định dạng có khả năng. Điều này sẽ trao cho bạn một ý tưởng làm thế nào để sử dụng hàm này.

Tất cả các dòng đầu ra được dựa vào ngày tháng y hệt: ngày 23 tháng 11 năm 1959.

Bảng 33-2. Các định dạng đầu vào hợp lệ cho PGTYPESdate_fmt_asc

Định dạng	Kết quả
mmddyy	112359
ddmmyy	231159
yymmdd	591123
yy/mm/dd	59/11/23
yy mm dd	59 11 23
yy.mm.dd	59.11.23
.mm.yyyy.dd.	.11.1959.23.
mmm. dd, yyyy	Nov. 23, 1959
mmm dd yyyy	Nov 23 1959
yyyy dd mm	1959 23 11
ddd, mmm. dd, yyyy	Mon, Nov. 23, 1959
(ddd) mmm. dd, yyyy	(Mon) Nov. 23, 1959

PGTYPESdate_defmt_asc

Sử dụng một mặt nạ định dạng để biến đổi một chuỗi char* C thành giá trị dạng ngày tháng.

```
int PGTYPESdate_defmt_asc(date *d, char *fmt, char *str);
```

Hàm này nhận một con trỏ tới giá trị ngày tháng mà sẽ giữ kết quả của hoạt động đó (d), mặt nạ định dạng để sử dụng cho việc phân tích cú pháp ngày tháng đó (fmt) và chuỗi char* C có đại diện văn bản của ngày tháng đó (str). Đại diện văn bản đó được kỳ vọng khớp với mặt nạ định dạng.

Tuy nhiên bạn không cần phải có một ánh xạ 1:1 của chuỗi đó đối với mặt nạ định dạng đó. Hàm chỉ phân tích trật tự tuần tự và tìm các hằng yy hoặc yyyy mà chỉ vị trí của năm, mm để chỉ vị trí của tháng và dd để chỉ vị trí của ngày.

Bảng 33-3 chỉ một ít các định dạng có khả năng. Điều này sẽ trao cho bạn ý tưởng làm thế nào để sử dụng hàm này.

Bảng 33-3. Các định dạng đầu vào hợp lệ cho rdefmtdate

Định dạng	Chuỗi	Kết quả
ddmmyy	21-2-54	1954-02-21
ddmmyy	2-12-54	1954-12-02
ddmmyy	20111954	1954-11-20

Định dạng	Chuỗi	Kết quả
ddmmyy	130464	1964-04-13
mmm.dd.yyyy	MAR-12-1967	1967-03-12
yy/mm/dd	1954, February 3rd	1954-02-03
mmm.dd.yyyy	041269	1969-04-12
yy/mm/dd	Trong năm 2525, vào tháng 7, loài người sẽ sống trong ngày thứ 28	2525-07-28
dd-mm-yy	Tôi nói vào ngày 28/07/2525	2525-07-28
mmm.dd.yyyy	9/14/58	1958-09-14
yy/mm/dd	47/03/29	1947-03-29
mmm.dd.yyyy	oct 28 1975	1975-10-28
mmddyy	Nov 14th, 1985	1985-11-14

33.8.3. Dạng dấu thời gian

Dạng dấu thời gian trong C cho phép các chương trình của bạn làm việc được với dữ liệu dạng dấu thời gian của SQL. Xem Phần 8.5 để có dạng tương đương trong máy chủ PostgreSQL.

Các hàm sau đây có thể được sử dụng để làm việc với dạng dấu thời gian:

PGTYPEStimestamp_from_asc

Phân tích một dấu thời gian từ đại diện văn bản của nó thành một biến dấu thời gian.

```
timestamp PGTYPEStimestamp_from_asc(char *str, char **endptr);
```

Hàm này nhận chuỗi để phân tích () và một con trỏ tới một chuỗi char* C (endptr). Hiện tại ECPG luôn phân tích chuỗi hoàn chỉnh và vì thế nó hiện không hỗ trợ để lưu trữ địa chỉ của ký tự không hợp lệ đầu tiên trong *endptr. Bạn có thể thiết lập an toàn endptr về NULL.

Hàm trả về dấu thời gian được phân tích nếu thành công. PGTYPESInvalidTimestamp được trả về nếu có lỗi và errno được thiết lập về PGTYPES_TS_BAD_TIMESTAMP. Để có được các lưu ý về giá trị này, xem PGTYPESInvalidTimestamp.

Nói chung, chuỗi đầu vào có thể chứa bất kỳ sự kết hợp nào của một đặc tả dữ liệu được phép, một ký tự không gian trống và một đặc tả thời gian được phép. Lưu ý rằng các vùng thời gian sẽ được ECPG hỗ trợ. Nó có thể phân tích chúng nhưng không áp dụng bất kỳ tính toán nào khi máy chủ PostgreSQL thực hiện, ví dụ thế. Các trình chỉ định vùng thời gian bị bỏ đi một cách âm thầm.

Bảng 33-4 có một ít ví dụ cho các chuỗi đầu vào.

Bảng 33-4. Các định dạng đầu vào hợp lệ cho PGTYPEStimestamp_from_asc

Đầu vào	Kết quả
1999-01-08 04:05:06	1999-01-08 04:05:06
January 8 04:05:06 1999 PST	1999-01-08 04:05:06

Đầu vào	Kết quả
1999-Jan-08 04:05:06.789-8	1999-01-08 04:05:06.789 (trình chỉ thị vùng thời gian bị bỏ qua)
J2451187 04:05-08:00	1999-01-08 04:05:00 (trình chỉ thị vùng thời gian bị bỏ qua)

PGTYPEStimestamp_to_asc

Biến đổi một ngày tháng về một chuỗi char* C.

```
char *PGTYPEStimestamp_to_asc(timestamp tstamp);
```

Hàm này nhận dấu thời gian tstamp như là đối số duy nhất của nó và trả về một chuỗi có chứa đại diện văn bản của dấu thời gian đó.

PGTYPEStimestamp_current

Truy xuất dấu thời gian hiện hành.

```
void PGTYPEStimestamp_current(timestamp *ts);
```

Hàm đó truy xuất dấu thời gian hiện hành và lưu nó vào biến dấu thời gian mà ts trỏ tới.

PGTYPEStimestamp_fmt_asc

Biến đổi một biến dấu thời gian thành một char* C bằng việc sử dụng một mặt nạ định dạng.

```
int PGTYPEStimestamp_fmt_asc(timestamp *ts, char *output, int str_len, char *fmtstr);
```

Hàm này nhận một con trỏ tới dấu thời gian để biến đổi như đối số đầu của nó (ts), một con trỏ tới bộ nhớ tạm đầu ra (output), độ dài cực đại mà đã được phân bổ cho bộ nhớ tạm đầu ra (str_len) và mặt nạ định dạng để sử dụng cho sự biến đổi đó (fmtstr).

Nếu thành công, hàm trả về 0 và một giá trị âm nếu có lỗi.

Bạn có thể sử dụng các trình chỉ định định dạng sau đây cho mặt nạ định dạng. Các trình chỉ định định dạng là y hệt các trình được sử dụng trong hàm strftime trong libc. Bất kỳ trình chỉ định không có định dạng nào cũng sẽ được sao chép vào bộ nhớ tạm đầu ra.

- %A - được thay thế bằng đại diện quốc gia về tên đầy đủ các ngày trong tuần.
- %a - được thay thế bằng đại diện quốc gia tên vắn tắt ngày trong tuần.
- %B - được thay thế bằng đại diện quốc gia tên tháng đầy đủ.
- %b - được thay thế bằng đại diện quốc gia tên vắn tắt của tháng.
- %C - được thay thế bằng (năm/100) số thập phân; chữ số duy nhất được thay thế bằng zero.
- %c - được thay thế bằng đại diện quốc gia thời gian và ngày tháng.
- %D - là tương đương với %m/%d/%y.
- %d - được thay thế bằng ngày của tháng như một số thập phân (01-31).
- %E* %O* - các mở rộng bản địa của POSIX. Các tuần tự %Ec %EC %Ex %EX %Ey %EY %Od %Oe %OH %OI %Om %OM %OS %Ou %OU %OV %Ow %OW %Oy được hỗ trợ để đưa ra các đại diện lựa chọn thay thế.

  %OB được triển khai bổ sung để đại diện cho các tên tháng lựa chọn (được sử dụng

một mình, không có ngày được nhắc tới).

- %e - được thay thế bằng ngày của tháng như một số thập phân (1-31); chữ số duy nhất sẽ được đặt trước bằng một dấu trắng.
- %F - là tương đương với %Y-%m-%d.
- %G - được thay thế bằng một năm như một số thập phân với thế kỷ. Năm nay là một năm có chứa phần tuần lớn hơn (Thứ hai như là ngày đầu tuần).
- %g - được thay thế bằng năm y hệt như trong %G, nhưng như một số thập phân không có thế kỷ (00-99).
- %H - được thay thế bằng giờ (đồng hồ 24 giờ) như một số thập phân (00-23).
- %h - là y hệt như %b.
- %I - được thay thế bằng giờ (đồng hồ 12 giờ) như một số thập phân (01-12).
- %j - được thay thế bằng ngày của năm như một số thập phân (001-366).
- %k - được thay thế bằng giờ (đồng hồ 24 giờ) như một số thập phân (0-23); chữ số duy nhất được thay thế bằng một dấu trắng.
- %l - được thay thế bằng giờ (đồng hồ 12 giờ) như một số thập phân (1-12); chữ số duy nhất được đặt trước bằng một dấu trắng.
- %M - được thay thế bằng phút như một số thập phân (00-59).
- %m - được thay thế bằng tháng như một số thập phân (01-12).
- %n - được thay thế bằng một dòng mới.
- %O* - là y hệt như %E*
- %p - được thay thế bằng đại diện quốc gia của hoặc “ante meridiem” - a.m. (trước giữa trưa) hoặc “post meridiem” - p.m. (sau giữa trưa) một cách tương ứng.
- %R - là tương đương với %H:%M.
- %r - là tương đương với %I:%M:%S %p.
- %S - được thay thế bằng giây như một số thập phân (00-60).
- %s - được thay thế bằng số giây kể từ Kỷ nguyên (Epoch), UTC.
- %T - là tương đương với %H:%M:%S
- %t - được thay thế bằng một thẻ tab.
- %U - được thay thế bằng số tuần của năm (chủ nhật như là ngày đầu của tuần) như một số thập phân (00-53).
- %u - được thay thế bằng ngày trong tuần (thứ hai như là ngày đầu của tuần) như một số thập phân (1-7).
- %V - được thay thế bằng số tuần của năm (thứ hai như ngày đầu tuần) như một số thập phân (01-53). Nếu tuần có chứa ngày 01 tháng 01 thì có 4 hoặc nhiều ngày hơn

trong năm mới, sau đó là tuần 1; nếu không thì đó là tuần cuối của năm trước, và tuần tiếp sau là tuần 1.

- %v - là tương đương với %e-%b-%Y.
- %W - được thay thế bằng số tuần của năm (thứ hai như là ngày đầu tuần) như một số thập phân (00-52).
- %w - được thay thế bằng ngày trong tuần (chủ nhật như là ngày đầu tuần) như một số thập phân (0-6).
- %X - được thay thế bằng đại diện quốc gia về thời gian.
- %x - được thay thế bằng đại diện quốc gia về ngày tháng.
- %Y - được thay thế bằng năm với thế kỷ như một số thập phân.
- %y - được thay thế bằng năm mà không có thế kỷ như một số thập phân (00-99).
- %Z - được thay thế bằng tên vùng thời gian.
- %z - được thay thế bằng bù trừ vùng thời gian từ UTC; một dấu cộng đứng trước có nghĩa là phía đông của UTC, một dấu trừ là phía tây của UTC, các giờ và phút đi sau với 2 chữ số và không có dấu ngắt giữa chúng (dạng phổ biến cho các đầu đề ngày tháng theo RFC 822).
- %+ - được thay thế bằng đại diện quốc gia về ngày tháng và thời gian.
- %-* - Mở rộng libc của GNU. Không làm bất kỳ việc chèn thêm nào khi thực hiện các đầu ra số.
- $_* - Mở rộng libc của GNU. Rõ ràng chỉ định không gian cho việc chèn thêm.
- %0* - Mở rộng libc của GNU. Rõ ràng chỉ định zero cho việc chèn thêm.
- %% - được thay thế bằng %.

PGTYPEStimestamp_sub

Trừ một dấu thời gian từ một dấu thời gian khác và lưu kết quả vào một biến dạng khoảng.

```
int PGTYPEStimestamp_sub(timestamp *ts1, timestamp *ts2, interval *iv);
```

Hàm này sẽ trừ biến dấu thời gian mà ts2 trỏ tới từ biến dấu thời gian mà ts1 trỏ tới và sẽ lưu trữ kết quả trong biến khoảng thời gian mà iv trỏ tới.

Nếu thành công, hàm trả về 0 và một giá trị âm nếu có lỗi xảy ra.

PGTYPEStimestamp_defmt_asc

Phân tích một giá trị dấu thời gian từ đại diện văn bản của nó bằng việc sử dụng một mặt nạ định dạng.

```
int PGTYPEStimestamp_defmt_asc(char *str, char *fmt, timestamp *d);
```

Hàm này nhận đại diện văn bản của một dấu thời gian trong biến str cũng như một mặt nạ định dạng để sử dụng trong biến fmt. Kết quả sẽ được lưu trữ trong biến mà d trỏ tới.

Nếu mặt nạ định dạng fmt là NULL, thì hàm sẽ rơi ngược về mặt nạ định dạng mặc định mà

là %Y-%m-%d %H:%M:%S.

Đây là hàm ngược đối với PGTYPEStimestamp_fmt_asc. Xem tài liệu để tìm ra các khoản đầu vào mặt nạ định dạng có khả năng.

PGTYPEStimestamp_add_interval

Thêm một biến khoảng thời gian vào một biến dấu thời gian.

```
int PGTYPEStimestamp_add_interval(timestamp *tin, interval *span, timestamp *tout);
```

Hàm này nhận một con trỏ tới một biến dấu thời gian tin và một con trỏ tới một biến khoảng thời gian span. Nó thêm khoảng thời gian vào dấu thời gian và lưu dấu thời gian kết quả trong biến mà tout trỏ tới.

Nếu thành công, hàm trả về 0 và một giá trị âm nếu có lỗi xảy ra.

PGTYPEStimestamp_sub_interval

Trừ một biến khoảng thời gian từ một biến dấu thời gian.

```
int PGTYPEStimestamp_sub_interval(timestamp *tin, interval *span, timestamp *tout);
```

Hàm này trừ biến khoảng thời gian mà span trỏ tới từ biến dấu thời gian mà tin trỏ tới và lưu kết quả trong biến mà tout trỏ tới.

Nếu thành công, hàm trả về 0 và một giá trị âm nếu có lỗi xảy ra.

33.8.4. Dạng khoảng thời gian

Dạng khoảng thời gian trong C cho phép các chương trình của bạn làm việc với dữ liệu dạng khoảng thời gian của SQL. Xem Phần 8.5 để có dạng tương đương trong máy chủ PostgreSQL.

Các hàm sau có thể được sử dụng để làm việc với dạng khoảng thời gian:

PGTYPESinterval_new

Trả về một con trỏ tới biến khoảng thời gian mới được phân bổ.

```
interval *PGTYPESinterval_new(void);
```

PGTYPESinterval_free

Giải phóng bộ nhớ của một biến khoảng thời gian trước đó được phân bổ.

```
void PGTYPESinterval_new(interval *intvl);
```

PGTYPESinterval_from_asc

Phân tích cú pháp một khoảng thời gian từ đại diện văn bản của nó.

```
interval *PGTYPESinterval_from_asc(char *str, char **endptr);
```

Hàm này phân tích cú pháp chuỗi đầu vào str và trả về một con trỏ tới một biến khoảng thời gian được phân bổ. Hiện tại ECPG luôn phân tích cú pháp chuỗi hoàn chỉnh và vì thế nó hiện không hỗ trợ để lưu trữ địa chỉ của ký tự không hợp lệ đầu tiên trong *endptr. Bạn có thể an toàn thiết lập endptr về NULL.

PGTYPESinterval_to_asc

Biến đổi một biến dạng khoảng thời gian thành đại diện văn bản của nó.

```
char *PGTYPESinterval_to_asc(interval *span);
```

Hàm này biến đổi biến thời gian mà span trỏ tới trong một char* C. Đầu ra trông giống ví dụ này: @ 1 day 12 hours 59 mins 10 secs.

PGTYPESinterval_copy

Sao chép một biến dạng khoảng thời gian.

```
int PGTYPESinterval_copy(interval *intvlsrc, interval *intvldest);
```

Hàm này sao chép biến khoảng thời gian mà intvlsrc trỏ tới thành biến mà intvldest trỏ tới. Lưu ý rằng bạn cần phân bổ bộ nhớ cho biến đích trước đó.

33.8.5. Dạng thập phân

Dạng thập phân là tương tự như dạng số. Tuy nhiên nó bị giới hạn về độ chính xác tối đa với 30 chữ số có nghĩa. Ngược lại dạng số có thể được tạo ra chỉ trong đống, dạng thập phân có thể được tạo ra hoặc trong kho hoặc trong đống (bằng các hàm PGTYPESdecimal_new và PGTYPESdecimal_free). Có nhiều hàm khác làm việc với dạng thập phân trong chế độ tương thích của Informix được mô tả trong Phần 33.10.

Các hàm sau đây có thể được sử dụng để làm việc với dạng thập phân và không chỉ bị bó trong thư viện libcompat.

PGTYPESdecimal_new

Yêu cầu một con trỏ tới một biến thập phân mới được phân bổ.

```
decimal *PGTYPESdecimal_new(void);
```

PGTYPESdecimal_free

Giải phóng một dạng thập phân, giải phóng tất cả bộ nhớ của nó.

```
void PGTYPESdecimal_free(decimal *var);
```

33.8.6. Các giá trị errno của pgtypeslib

PGTYPES_NUM_BAD_NUMERIC

Một đối số sẽ có một biến số (hoặc trỏ tới một biến số) nhưng trong thực tế đại diện trong bộ nhớ của nó từng là không hợp lệ.

PGTYPES_NUM_OVERFLOW

Một sự tràn đã xảy ra. Vì dạng số có thể làm việc với hầu hết độ chính xác tùy ý, việc biến đổi một biến số thành các dạng khác có thể gây ra sự tràn.

PGTYPES_NUM_UNDERFLOW

Một sự chảy ngầm đã xảy ra. Vì dạng số có thể làm việc với hầu hết độ chính xác tùy ý, việc biến đổi một biến số thành các dạng khác có thể gây ra sự chảy ngầm.

PGTYPES_NUM_DIVIDE_ZERO

Một sự cố chia cho zero.

PGTYPES_DATE_BAD_DATE
PGTYPES_DATE_ERR_EARGS
PGTYPES_DATE_ERR_ENOSHORTDATE
PGTYPES_INTVL_BAD_INTERVAL

PGTYPES_DATE_ERR_ENOTDMY
PGTYPES_DATE_BAD_DAY
PGTYPES_DATE_BAD_MONTH
PGTYPES_TS_BAD_TIMESTAMP

33.8.7. Các hằng đặc biệt của pgtypeslib

PGTYPESInvalidTimestamp

Một giá trị dạng dấu thời gian đại diện cho một dấu thời gian không hợp lệ. Điều này được trả về bằng hàm PGTYPEStimestamp_from_asc nếu phân tích cú pháp lỗi. Lưu ý rằng vì đại diện nội bộ của dạng dữ liệu timestamp, PGTYPESInvalidTimestamp cũng là một dấu thời gian hợp lệ cùng một lúc. Nó được thiết lập về 1899-12-31 23:59:59. Để dò tìm ra các lỗi, hãy chắc chắn ứng dụng của bạn không chỉ kiểm thử cho PGTYPESInvalidTimestamp mà còn cho errno != 0 sau từng lời gọi tới PGTYPEStimestamp_from_asc.

33.9. Sử dụng các khu vực của trình mô tả

Một khu vực trình mô tả SQL là một phương pháp phức tạp hơn cho việc xử lý kết quả của một lệnh SELECT, FETCH hoặc DESCRIBE. Một khu vực trình mô tả SQL nhóm dữ liệu một hàng dữ liệu cùng với các khoản siêu dữ liệu vào một cấu trúc cơ sở dữ liệu. Siêu dữ liệu đó đặc biệt hữu dụng khi thực thi các lệnh SQL động, nơi mà bản chất tự nhiên của các cột kết quả có thể không được biết trước về thời gian. PostgreSQL đưa ra 2 cách thức để sử dụng khu vực trình mô tả (Descriptor Areas): Khu vực Trình mô tả SQL được đặt tên và các SQLDA cấu trúc C.

33.9.1. Khu vực trình mô tả SQL được đặt tên

Một khu vực của trình mô tả SQL được đặt tên bao gồm một đầu đề chứa thông tin có liên quan tới toàn bộ trình mô tả đó, và một hoặc nhiều khoản đối với các khu vực của trình mô tả mà về cơ bản từng thứ đó mô tả một cột trong hàng kết quả.

Trước khi bạn có thể sử dụng một khu vực trình mô tả SQL, bạn cần phân bổ một khu vực:

```
EXEC SQL ALLOCATE DESCRIPTOR identifier;
```

Mã nhận diện phục vụ như là “tên biến” của khu vực trình mô tả. Khi bạn không cần trình mô tả hơn nữa, bạn nên bỏ phân bổ nó:

```
EXEC SQL DEALLOCATE DESCRIPTOR identifier;
```

Để sử dụng một khu vực trình mô tả, hãy chỉ định nó như là đích lưu trữ trong một mệnh đề INTO, thay vì việc liệt kê các biến chủ host;

```
EXEC SQL FETCH NEXT FROM mycursor INTO SQL DESCRIPTOR mydesc;
```

Nếu tập hợp kết quả là rỗng, thì Khu vực Trình mô tả sẽ vẫn có siêu dữ liệu từ truy vấn đó, nghĩa là các tên trường.

Đối với các truy vấn còn chưa được chuẩn bị được thực thi, lệnh DESCRIBE có thể được sử dụng để có siêu dữ liệu của tập các kết quả đó:

```
EXEC SQL BEGIN DECLARE SECTION;
char *sql_stmt = "SELECT * FROM table1";
EXEC SQL END DECLARE SECTION;
```

```
EXEC SQL PREPARE stmt1 FROM :sql_stmt;
EXEC SQL DESCRIBE stmt1 INTO SQL DESCRIPTOR mydesc;
```

Trước PostgreSQL 9.0, từ khóa SQL từng là tùy chọn, nên việc sử dụng các Khu vực Trình mô tả SQL được đặt tên được sản xuất DESCRIPTOR và SQL DESCRIPTOR. Bây giờ nó là bắt buộc, làm mờ đi từ khóa SQL sản xuất ra các Khu vực Trình mô tả SQLDA, xem Phần 33.9.2.

Trong các lệnh DESCRIBE và FETCH, các từ khóa INTO và USING có thể được sử dụng một cách tương tự: chúng tạo ra tập kết quả và siêu dữ liệu trong một Khu vực Trình mô tả.

Làm thế nào bây giờ để bạn có được dữ liệu ra khỏi khu vực trình mô tả? Bạn có thể nghĩ về khu vực trình mô tả như một cấu trúc với các trường được đặt tên. Để truy xuất giá trị của một trường từ đầu đề và lưu trữ nó vào một biến chủ host, hãy sử dụng lệnh sau đây:

```
EXEC SQL GET DESCRIPTOR name :hostvar = field;
```

Hiện hành, chỉ có một trường đầu đề được xác định: COUNT, nó nói cách mà nhiều khu vực trình mô tả khoản mục tồn tại (đó là, có bao nhiêu cột có trong kết quả đó). Biến chủ host cần phải là một dạng số nguyên. Để có một trường từ khu vực trình mô tả khoản mục, hãy sử dụng lệnh sau đây:

```
EXEC SQL GET DESCRIPTOR name VALUE num :hostvar = field;
```

num có thể là một hằng số nguyên hoặc một biến chủ host có một số nguyên. Các trường có khả năng gồm:

CARDINALITY (integer)

số các hàng trong tập kết quả

DATA

khoản mục thực sự của dữ liệu (vì thế, dạng dữ liệu của trường này phụ thuộc vào truy vấn)

DATETIME_INTERVAL_CODE (integer)

?

DATETIME_INTERVAL_PRECISION (integer)

không được triển khai

INDICATOR (integer)

chỉ số (chỉ một giá trị null hoặc một sự cắt bớt giá trị)

KEY_MEMBER (integer)

không được triển khai

LENGTH (integer)

độ dài các dữ liệu trong các ký tự

NAME (string)

tên cột

NULLABLE (integer)

không được triển khai

OCTET_LENGTH (integer)

độ dài của đại diện ký tự các dữ liệu theo byte

PRECISION (integer)

độ chính xác (cho dạng numeric)

RETURNED_LENGTH (integer)

độ dài dữ liệu theo các ký tự

RETURNED_OCTET_LENGTH (integer)

độ dài đại diện ký tự của các dữ liệu theo byte

SCALE (integer)

phạm vi (cho dạng numeric)

TYPE (integer)

mã theo số của dạng dữ liệu của cột

Trong các lệnh EXECUTE, DECLARE và OPEN, hiệu ứng của các từ khóa INTO và USING là khác nhau. Một Khu vực Trình mô tả cũng có thể được xây dựng bằng tay để cung cấp các tham số đầu vào cho một truy vấn hoặc một con trỏ và USING SQL DESCRIPTOR name là cách để truyền các tham số đầu vào vào trong một truy vấn được tham số hóa. Lệnh đó sẽ xây dựng Khu vực Trình mô tả SQL được đặt tên như bên dưới:

```
EXEC SQL SET DESCRIPTOR name VALUE num field = :hostvar;
```

PostgreSQL hỗ trợ việc truy xuất nhiều hơn một bản ghi trong một lệnh FETCH và việc lưu trữ dữ liệu trong các biến chủ host trong trường hợp này giả thiết rằng biến đó là một mảng. Như:

```
EXEC SQL BEGIN DECLARE SECTION;
int id[5];
EXEC SQL END DECLARE SECTION;

EXEC SQL FETCH 5 FROM mycursor INTO SQL DESCRIPTOR mydesc;

EXEC SQL GET DESCRIPTOR mydesc VALUE 1 :id = DATA;
```

33.9.2. Khu vực Trình mô tả SQLDA

Khu vực Trình mô tả SQLDA là một cấu trúc ngôn ngữ C cũng có thể được sử dụng để có tập các kết quả và siêu dữ liệu của một truy vấn. Một cấu trúc lưu trữ một bản ghi từ tập các kết quả.

```
EXEC SQL include sqlda.h;
sqlda_t         *mysqlda;

EXEC SQL FETCH 3 FROM mycursor INTO DESCRIPTOR mysqlda;
```

Lưu ý rằng từ khóa SQL bị bỏ qua. Các đoạn về các trường hợp điển hình của các từ khóa INTO và USING trong Phần 33.9.1 cũng áp dụng ở đây với một bổ sung. Trong một lệnh DESCRIBE thì từ khóa của DESCRIPTOR có thể hoàn toàn bị bỏ qua nếu từ khóa INTO được sử dụng:

```
EXEC SQL DESCRIBE prepared_statement INTO mysqlda;
```

Cấu trúc của SQLDA là:

```
#define NAMEDATALEN 64
struct sqlname
{
        short   length;
        char    data[NAMEDATALEN];
};
struct sqlvar_struct
{
        short   sqltype;
        short   sqllen;
        char    *sqldata;
        short   *sqlind;
        struct  sqlname sqlname;
};
struct sqlda_struct
{
        char    sqldaid[8];
        long    sqldabc;
        short   sqln;
        short   sqld;
        struct  sqlda_struct *desc_next;
        struct  sqlvar_struct sqlvar[1];
};
typedef struct sqlvar_struct sqlvar_t;
typedef struct sqlda_struct sqlda_t;
```

Dữ liệu được phân bổ cho một cấu trúc SQLDA là khác nhau khi nó phụ thuộc vào số lượng các trường trong một tập hợp kết quả và cũng phụ thuộc vào độ dài của các giá trị dữ liệu chuỗi trong một bản ghi. Các trường riêng rẽ của cấu trúc SQLDA gồm:

sqldaid

nó chứa chuỗi hằng “SQLDA”

sqldabc

nó chứa kích cỡ không gian được phân bổ theo byte

sqln

nó chứa số các tham số đầu vào cho một trường hợp truy vấn được tham số hóa mà nó được truyền vào các lệnh OPEN, DECLARE hoặc EXECUTE bằng việc sử dụng từ khóa USING. Trong trường hợp nó được sử dụng như là đầu ra của các lệnh SELECT, EXECUTE hoặc FETCH, thì giá trị của nó là y hệt như lệnh sqld.

sqld

Nó chứa số các trường trong một tập hợp kết quả.

desc_next

Nếu truy vấn đó trả về nhiều hơn một bản ghi, thì các cấu trúc nhiều liên kết SQLDA được trả về, và desc_next giữ một con trỏ tới khoản tiếp sau trong danh sách đó.

sqlvar

Đây là mảng các trường trong tập hợp kết quả. Các trường gồm:

sqltype

Nó gồm dạng mã định danh trường. Về các giá trị, xem enum ECPGttype trong

ecpgtype.h.

sqllen

Nó gồm độ dài nhị phân của trường. Như, 4 byte cho ECPGt_int.

sqldata

(char *)sqldata trỏ tới các dữ liệu đó.

sqlind

(char *)sqlind trỏ tới chỉ số NULL cho dữ liệu. 0 có nghĩa là NOT NULL, -1 có là NULL.

sqlname

cấu trúc sqlname sqlname gồm tên của trường trong một cấu trúc:

```
struct sqlname
{
        short    length;
        char     data[NAMEDATALEN];
};
```

length

sqlname.length gồm độ dài của tên trường.

data

sqlname.data gồm tên trường thực tế.

33.10. Chế độ tương thích với Informix

ecpg có thể được chạy trong một chế độ được gọi là tương thích với Informix. Nếu chế độ này được bật, nó cố hành xử như thể nó từng được biên dịch trước đó cho Informix E/SQL. Thường được nói điều này sẽ cho phép bạn sử dụng dấu đô la ($) thay vì EXEC SQL nguyên thủy để giới thiệu các lệnh SQL nhúng:

```
$int j = 3;
$CONNECT TO :dbname;
$CREATE TABLE test(i INT PRIMARY KEY, j INT);
$INSERT INTO test(i, j) VALUES (7, :j);
$COMMIT;
```

Có 2 chế độ tương thích: INFORMIX, INFORMIX_SE

Khi các chương trình liên kết sử dụng chế độ tương thích này, hãy nhớ liên kết đối với libcompat mà được xuất xưởng với ECPG.

Ngoài cú pháp được giải thích trước đó, chế độ tương thích với Informix chuyển vài hàm cho đầu vào, đầu ra và truyền dữ liệu cũng như các lệnh SQL nhúng được biết từ E/SQL sang ECPG.

Chế độ tương thích Informix được kết nối chặt chẽ với thư viện pgtypeslib của ECPG, pgtypeslib ánh xạ các dạng dữ liệu SQL tới các dạng dữ liệu trong chương trình chủ host C và hầu hết các chức năng bổ sung thêm của chế độ tương thích Informix cho phép bạn vận hành trong các dạng chương trình chủ host C đó. Tuy nhiên, lưu ý rằng sự mở rộng của tính tương thích có giới hạn. Nó không cố sao chép hành vi của Informix; nó cho phép bạn làm nhiều hơn hoặc ít hơn các hoạt động y hệt

và trao cho bạn các chức năng mà có cùng tên và cùng hành vi cơ bản nhưng nó không là sự thay thế nếu bạn đang sử dụng Informix lúc này. Hơn nữa, vài dạng dữ liệu là khác nhau. Ví dụ, các dạng ngày tháng thời gian và khoảng thời gian của PostgreSQL không biết về các dải như ví dụ YEAR TO MINUTE nên bạn cũng sẽ không thấy hỗ trợ trong ECPG cho điều đó.

33.10.1. Các dạng bổ sung thêm

"Chuỗi" đặc thù Informix dạng giả lập cho việc lưu trữ dữ liệu chuỗi ký tự được tỉa bớt bây giờ được hỗ trợ trong chế độ Informix mà không có việc sử dụng typedef. Trong thực tế, ở chế độ Informix, ECPG từ chối xử lý các tệp nguồn có chứa typedef sometype string;

```
EXEC SQL BEGIN DECLARE SECTION;
string userid;    /* this variable will contain trimmed data */
EXEC SQL END DECLARE SECTION;

EXEC SQL FETCH MYCUR INTO :userid;
```

33.10.2. Các lệnh SQL nhúng thêm/bỏ bớt

CLOSE DATABASE

Lệnh này đóng kết nối hiện hành. Trong thực tế, đây là một đồng nghĩa đối với DISCONNECT CURRENT của ECPG:

```
$CLOSE DATABASE;      /* close the current connection */
EXEC SQL CLOSE DATABASE;
```

FREE cursor_name

Vì những khác biệt cách mà ECPG làm việc khi so sánh với ESQL/C của Informix (như, các bước nào thuần túy là các biến đổi ngữ pháp và các bước nào dựa vào thư viện thời gian chạy nằm bên dưới) không có lệnh FREE cursor_name trong ECPG. Điều này là vì trong ECPG, DECLARE CURSOR không dịch thành một lời gọi hàm trong thư viện thời gian chạy mà sử dụng cho tên con trỏ. Điều này có nghĩa là không có việc thủ thư thời gian chạy của các con trỏ SQL trong thư viện thời gian chạy ECPG, chỉ trong máy chủ PostgreSQL.

FREE statement_name

FREE statement_name là một đồng nghĩa cho DEALLOCATE PREPARE statement_name.

33.10.3. Khu vực trình mô tả SQLDA tương thích Informix

Chế độ tương thích Informix hỗ trợ một cấu trúc khác so với cấu trúc được mô tả trong Phần 33.9.2. Xem bên dưới:

```
struct sqlvar_compat
{
        short sqltype;
        int sqllen;
        char *sqldata;
        short *sqlind;
        char *sqlname;
        char *sqlformat;
        short sqlitype;
        short sqlilen;
        char *sqlidata;
        int sqlxid;
```

```
        char *sqltypename;
        short sqltypelen;
        short sqlownerlen;
        short sqlsourcetype;
        char *sqlownername;
        int sqlsourceid;
        char *sqlilongdata;
        int sqlflags;
        void *sqlreserved;
};
struct sqlda_compat
{
        short sqld;
        struct sqlvar_compat *sqlvar;
        char desc_name[19];
        short desc_occ;
        struct sqlda_compat *desc_next;
        void *reserved;
};
typedef struct sqlvar_compat sqlvar_t;
typedef struct sqlda_compat sqlda_t;
```

Các thuộc tính toàn thể gồm:

sqld

Số các trường trong trình mô tả SQLDA.

sqlvar

Con trỏ tới các thuộc tính theo từng trường.

desc_name

Không được sử dụng, được điền bằng các byte zero.

desc_occ

Kích cỡ của cấu trúc được phân bổ.

desc_next

Con trỏ tới cấu trúc SQLDA tiếp theo nếu tập kết quả gồm nhiều hơn một bản ghi.

reserved

Con trỏ không được sử dụng, chứa NULL. Được giữ lại cho tính tương thích Informix.

Các thuộc tính theo từng trường là ở bên dưới, chúng được lưu trữ trong mảng sqlvar:

sqltype

Dạng của trường. Các hằng có trong sqltypes.h

sqllen

Độ dài dữ liệu của trường.

sqldata

Con trỏ tới dữ liệu của trường. Con trỏ đó có dạng char *, dữ liệu đó được nó trỏ ở định dạng nhị phân. Ví dụ:

```
int intval;
switch (sqldata->sqlvar[i].sqltype)
{
        case SQLINTEGER:
                intval = *(int *)sqldata->sqlvar[i].sqldata;
                break;
```

```
            ...
        }
```

sqlind

Con trỏ tới chỉ số NULL. Nếu được DESCRIBE hoặc FETCH trả về thì nó luôn là một con trỏ hợp lệ. Nếu được sử dụng như đầu vào cho EXECUTE ... USING sqlda; thì giá trị con trỏ NULL có nghĩa là giá trị cho trường này là khác NULL. Nếu không thì một con trỏ hợp lệ và sqlitype phải được thiết lập đúng phù hợp. Ví dụ:

```
if (*(int2 *)sqldata->sqlvar[i].sqlind != 0)
   printf("value is NULL\n");
```

sqlname

Tên của trường. Chuỗi kết thúc với 0.

sqlformat

Được giữ lại trong Informix, giá trị của PQfformat() cho trường đó.

sqlitype

Dạng dữ liệu của chỉ số NULL. Nó luôn là SQLSMINT khi trả về dữ liệu từ máy chủ.

Khi SQLSMINT được sử dụng cho một truy vấn có tham số, thì dữ liệu được đối xử theo dạng của tập hợp.

sqlilen

Độ dài dữ liệu con chỉ số NULL.

sqlxid

Dạng trường được mở rộng, kết quả của PQftype().

sqltypename
sqltypelen
sqlownerlen
sqlsourcetype
sqlownername
sqlsourceid
sqlflags
sqlreserved

Không được sử dụng.

sqlilongdata

Nó ngang bằng với sqldata nếu sqllen lớn hơn 32KB.

Ví dụ:

```
EXEC SQL INCLUDE sqlda.h;
        sqlda_t *sqlda; /* This doesn't need to be under embedded DECLARE SECTION */
        EXEC SQL BEGIN DECLARE SECTION;
        char *prep_stmt = "select * from table1";
        int i;
        EXEC SQL END DECLARE SECTION;
        ...
        EXEC SQL PREPARE mystmt FROM :prep_stmt;
        EXEC SQL DESCRIBE mystmt INTO sqlda;
        printf("# of fields: %d\n", sqlda->sqld);
        for (i = 0; i < sqlda->sqld; i++)
                printf("field %d: \"%s\"\n", sqlda->sqlvar[i]->sqlname);
        EXEC SQL DECLARE mycursor CURSOR FOR mystmt;
```

```
EXEC SQL OPEN mycursor;
EXEC SQL WHENEVER NOT FOUND GOTO out;
while (1)
{
        EXEC SQL FETCH mycursor USING sqlda;
}
EXEC SQL CLOSE mycursor;
free(sqlda); /* The main structure is all to be free(),
                * sqlda and sqlda->sqlvar is in one allocated area */
```

Xem đầu đề sqlda.h và kiểm thử hồi quy src/interfaces/ecpg/test/compat_informix/sqlda.pgc để có thêm thông tin.

33.10.4. Các hàm bổ sung

decadd

Thêm 2 giá trị dạng thập phân.

```
int decadd(decimal *arg1, decimal *arg2, decimal *sum);
```

Hàm này nhận một con trỏ tới toán hạng đầu tiên của dạng thập phân (arg1), một con trỏ tới toán hạng thứ 2 của dạng thập phân (arg2) và một con trỏ tới một giá trị dạng thập phân mà sẽ chứa tổng (sum). Nếu thành công, hàm trả về 0. ECPG_INFORMIX_NUM_OVERFLOW được trả về trong trường hợp tràn và ECPG_INFORMIX_NUM_UNDERFLOW trong trường hợp chảy ngầm. -1 được trả về cho các hỏng khác và errno được thiết lập cho số errno tương ứng của pgtypeslib.

deccmp

So sánh 2 biến dạng thập phân.

```
int deccmp(decimal *arg1, decimal *arg2);
```

Hàm này nhận một con trỏ tới giá trị thập phân đầu tiên (arg1), một con trỏ tới giá trị thập phân thứ 2 (arg2) và trả về một giá trị số nguyên chỉ ra cái nào là giá trị lớn hơn.

- 1, nếu giá trị mà arg1 trỏ tới là lớn hơn so với giá trị mà arg2 trỏ tới
- -1, nếu giá trị mà arg1 trỏ tới là nhỏ hơn so với giá trị mà arg2 trỏ tới
- 0, nếu giá trị mà arg1 trỏ tới và giá trị mà arg2 trỏ tới bằng nhau

deccopy

Sao chép một giá trị thập phân.

```
void deccopy(decimal *src, decimal *target);
```

Hàm này nhận một con trỏ tới giá trị thập phân mà sẽ được sao chép như là đối số đầu tiên (src) và một con trỏ tới cấu trúc đích của dạng thập phân (target) như là đối số thứ 2.

deccvasc

Biến đổi một giá trị từ đại diện ASCII của nó thành một dạng thập phân.

```
int deccvasc(char *cp, int len, decimal *np);
```

Hàm này nhận một con trỏ tới chuỗi chứa đại diện chuỗi của số sẽ được biến đổi (cp) cũng như độ dài của nó len. np là một con trỏ tới giá trị thập phân mà lưu kết quả của hoạt động.

Các định dạng hợp lệ là cho ví dụ: -2 , .794 , +3.44 , 592.49E07 or -32.84e-4.

Hàm này trả về 0 nếu thành công. Nếu tràn hoặc chảy ngầm xảy ra, thì ECPG_INFORMIX_NUM_OVERFLOW hoặc ECPG_INFORMIX_NUM_UNDERFLOW được trả về. Nếu đại diện ASCII có thể không được phân tích cú pháp, thì ECPG_INFORMIX_BAD_NUMERIC được trả về hoặc ECPG_INFORMIX_BAD_EXPONENT nếu vấn đề này đã xảy ra trong quá trình phân tích cú pháp lũy thừa đó.

deccvdbl

Biến đổi một giá trị dạng đúp double thành một giá trị dạng thập phân.

```
int deccvdbl(double dbl, decimal *np);
```

Hàm này nhận biến dạng double mà sẽ được biến đổi như đối số đầu tiên của nó (dbl). Như là đối số thứ 2 (np), hàm nhận một con trỏ tới biến thập phân mà sẽ giữ kết quả của hoạt động đó.

Hàm trả về 0 nếu thành công và một giá trị âm nếu biến đổi thất bại.

deccvint

Biến đổi một giá trị dạng int thành một giá trị dạng thập phân.

```
int deccvint(int in, decimal *np);
```

Hàm này nhận biến dạng int mà sẽ được biến đổi như là đối số đầu tiên của nó (in).

Như là đối số thứ 2 (np), hàm đó nhận một con trỏ tới biến thập phân mà sẽ giữ kết quả của hoạt động đó.

Hàm trả về 0 nếu thành công và một giá trị âm nếu biến đổi thất bại.

deccvlong

Biến đổi một giá trị dạng long thành một giá trị dạng thập phân.

```
int deccvlong(long lng, decimal *np);
```

Hàm này nhận biến dạng long mà sẽ được biến đổi như là đối số đầu tiên của nó (lng). Như là đối số thứ 2 (np), hàm nhận một con trỏ tới biến thập phân mà sẽ giữ kết quả của hoạt động đó.

Hàm trả về 0 nếu thành công và một giá trị âm nếu biến đổi thất bại.

decdiv

Chia 2 biến dạng thập phân.

```
int decdiv(decimal *n1, decimal *n2, decimal *result);
```

Hàm nhận các con trỏ tới các biến mà là các toán hạng đầu (n1) và thứ 2 (n2) và tính n1 /n2. result là một con trỏ tới biến mà sẽ giữ kết quả của hoạt động đó.

Nếu thành công, 0 được trả về và một giá trị âm nếu việc chia thất bại. Nếu tràn hoặc chảy ngầm xảy ra, thì hàm trả về ECPG_INFORMIX_NUM_OVERFLOW hoặc ECPG_INFORMIX_NUM_UNDERFLOW một cách tương ứng. Nếu một cố gắng chia cho zero được quan sát thấy, thì hàm trả về ECPG_INFORMIX_DIVIDE_ZERO.

decmul

Nhân 2 giá trị thập phân.

```
int decmul(decimal *n1, decimal *n2, decimal *result);
```

Hàm này nhận các con trỏ tới các biến mà là các toán hạng thứ nhất (n1) và thứ 2 (n2) và tính n1 * n2. result là một biến sẽ giữ kết quả của hành động trên.

Nếu thành công, 0 được trả về và một giá trị âm nếu phép nhân thất bại. Nếu tràn hoặc chảy ngầm xảy ra, thì hàm sẽ trả về ECPG_INFORMIX_NUM_OVERFLOW hoặc ECPG_INFORMIX_NUM_UNDERFLOW một cách tương ứng.

decsub

Trừ đi một giá trị thập phân từ một giá trị khác.

```
int decsub(decimal *n1, decimal *n2, decimal *result);
```

Hàm này nhận các con trỏ tới các biến mà sẽ là các toán hạng đầu tiên (n1) và thứ 2 (n2) và tính toán n1 -n2. result là một con trỏ tới biến mà sẽ giữ kết quả của hoạt động đó.

Nếu thành công, 0 được trả về và một giá trị âm nếu phép trừ thất bại. Nếu tràn hoặc chảy ngầm xảy ra, thì hàm trả về ECPG_INFORMIX_NUM_OVERFLOW hoặc ECPG_INFORMIX_NUM_UNDERFLOW một cách tương ứng.

dectoasc

Biến đổi một biến dạng thập phân thành đại diện ASCII của nó trong một chuỗi char* C.

```
int dectoasc(decimal *np, char *cp, int len, int right)
```

Hàm này nhận một con trỏ tới một biến dạng thập phân (np) mà nó biến đổi thành đại diện văn bản của nó. cp làmột bộ nhớ đệm sẽ giữ kết quả của hoạt động đó. Tham số right chỉ định, có bao nhiêu chữ số bên phải dấu chấm thập phân sẽ được đưa vào ở đầu ra. Kết quả sẽ được làm tròn về số các chữ số thập phân này. Việc thiết lập right về -1 chỉ rằng tất cả các chữ số thập phân có sẵn sẽ được đưa vào trong đầu ra. Nếu độ dài của bộ nhớ đệm đầu ra được len chỉ ra là không đủ để giữ đại diện văn bản bao gồm cả ký tự NULL theo sau, thì chỉ một ký tự * duy nhất được lưu trữ trong kết quả và -1 được trả về. Hàm trả về hoặc -1 nếu bộ nhớ đệm cp quá nhỏ hoặc ECPG_INFORMIX_OUT_OF_MEMORY nếu bộ nhớ từng bị cạn kiệt.

dectodbl

Biến đổi một biến dạng thập phân thành một dạng đúp double.

```
int dectodbl(decimal *np, double *dblp);
```

Hàm này nhận một con trỏ tới giá trị thập phân để biến đổi (np) và một con trỏ tới biến đúp mà sẽ giữ kết quả của hoạt động đó (dblp).

Nếu thành công, 0 được trả về và một giá trị âm nếu biến đổi thất bại.

dectoint

Biến đổi một biến dạng thập phân về dạng số nguyên interger.

```
int dectoint(decimal *np, int *ip);
```

Hàm này nhận một con trỏ tới giá trị thập phân để biến đổi (np) và một con trỏ tới biến số nguyên mà sẽ giữ kết quả của hành động này (ip).

Nếu thành công, 0 được trả về và một giá trị âm nếu biến đổi thất bại. Nếu một sự tràn xảy ra, thì ECPG_INFORMIX_NUM_OVERFLOW được trả về.

Lưu ý rằng triển khai ECPG khác với triển khai Informix. Informix hạn chế một số nguyên

về dãy từ -32767 đến 32767, trong khi các giới hạn trong triển khai ECPG phụ thuộc vào kiến trúc (-INT_MAX .. INT_MAX).

dectolong

Biến đổi một biến dạng dạng thập phân sang dạng long integer.

```
int dectolong(decimal *np, long *lngp);
```

Hàm này nhận một con trỏ tới giá trị thập phân để biến đổi (np) và một con trỏ tới biến long mà sẽ giữ kết quả của hoạt động này (lngp).

Nếu thành công, 0 được trả về và một giá trị âm nếu biến đổi thất bại. Nếu một sự tràn xảy ra, thì ECPG_INFORMIX_NUM_OVERFLOW được trả về.

Lưu ý rằng triển khai ECPG khác với triển khai Informix. Informix hạn chế một số nguyên dài về dãy từ -2.147.483.647 tới 2.147.483.647, trong khi các giới hạn trong triển khai ECPG phụ thuộc vào kiến trúc (-LONG_MAX .. LONG_MAX).

rdatestr

Biến đổi một ngày tháng sang một chuỗi char* C.

```
int rdatestr(date d, char *str);
```

Hàm này nhận 2 đối số, đối số đầu là ngày tháng để biến đổi (d và đối số thứ 2 là một con trỏ tới chuỗi đích. Định dạng đầu ra luôn là yyy-mm-dd, nên bạn cần phải cấp ít nhất 11 byte (bao gồm cả phần kết NUL) cho chuỗi đó.

Hàm này trả về 0 nếu thành công và một giá trị âm nếu có lỗi.

Lưu ý rằng triển khai của ECPG khác với triển khai Informix. Trong Informix thì định dạng có thể bị ảnh hưởng vì việc thiết lập các biến môi trường. Tuy nhiên, trong ECPG bạn không thể thay đổi định dạng đầu ra.

rstrdate

Phân tích cú pháp đại diện văn bản của một ngày tháng.

```
int rstrdate(char *str, date *d);
```

Hàm này nhận đại diện văn bản của ngày tháng để biến đổi (str) và một con trỏ tới một biến dạng ngày tháng (d). Hàm này không cho phép bạn chỉ định một mặt nạ định dạng. Nó sử dụng mặt nạ định dạng mặc định của Informix là mm/dd/yyyy. Trong nội bộ, hàm này được triển khai bằng công cụ rdefmtdate. Vì thế rstrdate là không nhanh hơn và nếu bạn có sự lựa chọn thì bạn nên chọn rdefmtdate, nó cho phép bạn chỉ định mặt nạ định dạng rõ ràng.

Hàm này trả về các giá trị y hệt như rdefmtdate.

rtoday

Lấy ngày tháng hiện hành

```
void rtoday(date *d);
```

Hàm này nhận một con trỏ tới biến ngày tháng (d) mà nó thiết lập cho ngày tháng hiện hành.

Trong nội bộ, hàm này sử dụng hàm PGTYPESdate_today.

rjulmdy

Trích các giá trị đối với ngày, tháng và năm từ một biến dạng ngày tháng.

```
int rjulmdy(date d, short mdy[3]);
```

Hàm này nhận ngày tháng d và một con trỏ tới một mảng 3 giá trị số nguyên ngắn mdy. Tháng, mdy[1] sẽ được thiết lập về giá trị của ngày và mdy[2] sẽ chứa năm.

Hàm này luôn trả về 0 vào lúc này.

Trong nội bộ hàm sử dụng hàm PGTYPESdate_julmdy.

rdefmtdate

Sử dụng một mặt nạ định dạng để biến đổi một chuỗi ký tự thành giá trị dạng ngày tháng.

```
int rdefmtdate(date *d, char *fmt, char *str);
```

Hàm này nhận một con trỏ tới giá trị ngày tháng mà sẽ giữ kết quả của hành động (d), mặt nạ định dạng để sử dụng cho việc phân tích cú pháp ngày tháng (fmt) và chuỗi char* C chứa đại diện văn bản của ngày tháng đó (str). Đại diện văn bản đó được kỳ vọng khớp với mặt nạ định dạng.

Tuy nhiên bạn không cần phải có một việc ánh xạ 1:1 của chuỗi đó đối với mặt nạ định dạng. Hàm đó chỉ phân tích trật tự tuần tự và tìm kiếm các hằng yy hoặc yyyy mà chỉ vị trí của năm, mm để chỉ vị trí của tháng và dd để chỉ vị trí của ngày. Hàm trả về các giá trị sau:

- 0 - Hàm được kết thúc thành công.
- ECPG_INFORMIX_ENOSHORTDATE - Ngày tháng không chứa các phân cách giữa ngày, tháng và năm. Trong trường hợp này chuỗi đầu vào phải chính xác là 6 hoặc 8 byte dài nhưng không phải.
- ECPG_INFORMIX_ENOTDMY - Chuỗi định dạng đã không chỉ chính xác trật tự tuần tự của năm, tháng và ngày.
- ECPG_INFORMIX_BAD_DAY - Chuỗi đầu vào không chứa một ngày hợp lệ.
- ECPG_INFORMIX_BAD_MONTH - Chuỗi đầu vào không chứa một tháng hợp lệ.
- ECPG_INFORMIX_BAD_YEAR - Chuỗi đầu vào không chứa một năm hợp lệ.

Trong nội bộ thì hàm này được triển khai để sử dụng hàm PGTYPESdate_defmt_asc. Xem tham chiếu ở đó cho một bảng ví dụ đầu vào.

rfmtdate

Biến đổi một biến dạng ngày tháng thành đại diện văn bản của nó bằng việc sử dụng mặt nạ định dạng.

```
int rfmtdate(date d, char *fmt, char *str);
```

Hàm này nhận ngày tháng để biến đổi (d), mặt nạ định dạng (fmt) và chuỗi mà sẽ giữ đại diện văn bản của ngày tháng đó (str).

Nếu thành công, 0 được trả về và một giá trị âm nếu một lỗi xảy ra.

Trong nội bộ thì hàm này sử dụng hàm PGTYPESdate_fmt_asc, xem tham chiếu ở đó cho các ví dụ.

rmdyjul

Tạo một giá trị ngày tháng từ một mảng 3 số nguyên ngắn mà chỉ định ngày, tháng và năm của ngày tháng đó.

```
int rmdyjul(short mdy[3], date *d);
```

Hàm này nhận mảng 3 số nguyên ngắn (mdy) và một con trỏ tới một biến dạng ngày tháng mà sẽ giữ kết quả của hoạt động đó.

Hiện hàm này luôn trả về 0.

Trong nội bộ thì hàm này được triển khai để sử dụng hàm PGTYPESdate_mdyjul.

rdayofweek

Hàm này nhận biến ngày tháng d như là đối số duy nhất của nó và trả về một số nguyên mà chỉ ngày của tuần cho ngày tháng này.

- 0 - Chủ nhật
- 1 - Thứ hai
- 2 - Thứ ba
- 3 - Thứ tư
- 4 - Thứ năm
- 5 - Thứ sáu
- 6 - Thứ bày

Trong nội bộ thì hàm này được triển khai để sử dụng hàm PGTYPESdate_dayofweek.

dtcurrent

Truy xuất dấu thời gian hiện hành .

```
void dtcurrent(timestamp *ts);
```

Hàm này truy xuất dấu thời gian hiện hành và lưu nó trong biến dấu thời gian mà ts trỏ tới.

dtcvasc

Phân tích cú pháp một dấu thời gian từ đại diện văn bản của nó thành một biến dấu thời gian

```
int dtcvasc(char *str, timestamp *ts);
```

Hàm này nhận chuỗi để phân tích cú pháp (str) và một con trỏ tới biến dấu thời gian mà sẽ giữ kết quả của hoạt động này (ts).

Hàm này trả về 0 nếu thành công và một giá trị âm nếu có lỗi.

Trong nội bộ thì hàm này sử dụng hàm PGTYPEStimestamp_from_asc. Xem tham chiếu ở đó cho một bảng với các đầu vào ví dụ.

dtcvfmtasc

Phân tích cú pháp một dấu thời gian từ đại diện văn bản của nó bằng việc sử dụng một mặt nạ định dạng trong một biến thời gian.

```
dtcvfmtasc(char *inbuf, char *fmtstr, timestamp *dtvalue)
```

Hàm này nhận chuỗi để phân tích cú pháp (inbuf), mặt nạ định dạng để sử dụng (fmtstr) và một con trỏ tới biến dấu thời gian mà sẽ giữ kết quả của hành động đó (dtvalue).

Hàm này được triển khai bằng công cụ của hàm PGTYPEStimestamp_defmt_asc. Xem tài liệu ở đó cho một danh sách các trình chỉ định định dạng mà có thể được sử dụng.

Hàm đó trả về 0 nếu thành công và một giá trị âm nếu có lỗi.

dtsub

Trừ một dấu thời gian từ một dấu thời gian khác và trả về một biến dạng khoảng thời gian.

```
int dtsub(timestamp *ts1, timestamp *ts2, interval *iv);
```

Hàm này sẽ trừ biến dấu thời gian mà ts2 trỏ tới từ biến dấu thời gian mà ts1 trỏ tới và sẽ lưu trữ kết quả trong biến khoảng thời gian mà iv trỏ tới.

Nếu thành công, hàm trả về 0 và một giá trị âm nếu lỗi xảy ra.

dttoasc

Biến đổi một biến dấu thời gian thành một chuỗi char* C.

```
int dttoasc(timestamp *ts, char *output);
```

Hàm này nhận một con trỏ tới biến dấu thời gian để biến đổi (ts) và chuỗi mà sẽ giữ kết quả của hành động output). Nó biến đổi ts thành đại diện văn bản của nó theo tiêu chuẩn SQL, điều sẽ là YYYY-MM-DD HH:MM:SS.

Nếu thành công, hàm trả về 0 và một giá trị âm nếu có lỗi xảy ra.

dttofmtasc

Biến đổi một biến dấu thời gian thành một char* C bằng việc sử dụng một mặt nạ định dạng.

```
int dttofmtasc(timestamp *ts, char *output, int str_len, char *fmtstr);
```

Hàm này nhận một con trỏ tới dấu thời gian để biến đổi như là đối số đầu tiên của nó (ts), một con trỏ tới bộ nhớ đệm đầu ra (output), độ dài tối đa mà từng được phân bổ cho bộ nhớ đệm đầu ra (str_len) và mặt nạ định dạng để sử dụng cho sự biến đổi đó (fmtstr).

Nếu thành công, hàm trả về 0 và một giá trị âm nếu có lỗi xảy ra.

Trong nội bộ, hàm này sử dụng hàm PGTYPEStimestamp_fmt_asc. Xem tham chiếu ở đó để có thông tin về các trình chỉ định mặt nạ định dạng nào có thể được sử dụng.

intoasc

Biến đổi một biến khoảng thời gian thành một chuỗi char* C.

```
int intoasc(interval *i, char *str);
```

Hàm này nhận một con trỏ tới biến khoảng thời gian để biến đổi (i) và chuỗi mà sẽ giữ kết quả của hành động này (str). Nó biến đổi i thành đại diện văn bản của nó theo tiêu chuẩn SQL, điều sẽ là YYYY-MM-DD HH:MM:SS.

Nếu thành công, hàm trả về 0 và một giá trị âm nếu có lỗi xảy ra.

rfmtlong

Biến đổi một giá trị số nguyên dài thành đại diện văn bản của nó bằng việc sử dụng một mặt nạ định dạng.

```
int rfmtlong(long lng_val, char *fmt, char *outbuf);
```

Hàm nhận giá trị dài lng_val, mặt nạ định dạng fmt và một con trỏ tới bộ nhớ đệm đầu ra outbuf. Nó biến đổi giá trị dài theo mặt nạ định dạng thành đại diện văn bản của nó.

Mặt nạ định dạng có thể được cấu tạo từ các ký tự chỉ định định dạng sau :

- * (dấu sao) - nếu vị trí này có thể nếu khác sẽ là trống, hãy điền nó với một dấu sao.
- & (dấu và) - nếu vị trí này có thể nếu khác sẽ là trống, hãy điền nó với một zero.
- # - biến các số zero dẫn đầu thành các dấu trắng.
- < - căn chỉnh sang trái số trong chuỗi
- , (dấu phẩy) - nhóm các số 4 chữ số hoặc hơn thành các nhóm 3 chữ số phân cách nhau bằng một dấu phẩy.
- . (dấu chấm) - ký tự này phân cách phần toàn là số của số đó khỏi phần phân số.
- - (dấu trừ) - dấu trừ xuất hiện nếu số đó là một giá trị âm.
- + (dấu cộng) - dấu cộng xuất hiện nếu số đó là một giá trị dương.
- (- điều này thay thế dấu trừ ở trước của số âm. Dấu trừ sẽ không xuất hiện.
-) - ký tự này thay thế dấu trừ và được in đằng sau giá trị âm.
- $ - ký hiệu tiền tệ.

rupshift

Biến đổi một chuỗi thành chữ hoa.

```
void rupshift(char *str);
```

Hàm này nhận một con trỏ tới chuỗi đó và biến đổi từng ký tự chữ thường thành chữ hoa.

byleng

Trả về số các ký tự trong một chuỗi mà không tính các dấu trống đằng sau.

```
int byleng(char *str, int len);
```

Hàm này kỳ vọng một chuỗi có độ dài cố định như là đối số đầu tiên của nó (str) và độ dài của nó như là đối số thứ 2 của nó (len). Nó trả về số các ký tự có nghĩa, đó là độ dài của chuỗi không có các dấu trắng đằng đuôi.

ldchar

Sao chép một chuỗi độ dài cố định vào trong một chuỗi kết thúc là null.

```
void ldchar(char *src, int len, char *dest);
```

Hàm này nhận chuỗi có độ dài cố định để sao chép (src), độ dài của nó (len) và một con trỏ tới bộ nhớ đích (dest). Lưu ý rằng bạn cần dự phòng ít nhất len+1 byte cho chuỗi mà dest trỏ tới. Hàm đó sao chép hầu hết len byte tới vị trí mới (ít hơn nếu chuỗi nguồn có các dấu trắng ở đuôi) và thêm vào kết thúc null.

rgetmsg

```
int rgetmsg(int msgnum, char *s, int maxsize);
```

Hàm này tồn tại nhưng không được triển khai vào lúc này!

rtypalign

```
int rtypalign(int offset, int type);
```

Hàm này tồn tại nhưng không được triển khai vào lúc này!

rtypmsize

```
int rtypmsize(int type, int len);
```

Hàm này tồn tại nhưng không được triển khai vào lúc này!

rtypwidth

```
int rtypwidth(int sqltype, int sqllen);
```

Hàm này tồn tại nhưng không được triển khai vào lúc này!

rsetnull

Thiết lập một biến về NULL.

```
int rsetnull(int t, char *ptr);
```

Hàm này nhận một số nguyên chỉ dạng của biến và một con trỏ tới bản thân biến đó mà được phát tới một char* p của C.

Các dạng sau đây tồn tại:

- CCHARTYPE - Cho một biến dạng char hoặc char*
- CSHORTTYPE - Cho một biến dạng short int
- CINTTYPE - Cho một biến dạng int
- CBOOLTYPE - Cho một biến dạng boolean
- CFLOATTYPE - Cho một biến dạng float
- CLONGTYPE - Cho một biến dạng long
- CDOUBLETYPE - Cho một biến dạng double
- CDECIMALTYPE - Cho một biến dạng decimal
- CDATETYPE - Cho một biến dạng date
- CDTIMETYPE - Cho một biến dạng timestamp

Đây là một ví dụ về một lời gọi hàm này:

```
$char c[] = "abc ";
$short s = 17;
$int i = -74874;

rsetnull(CCHARTYPE, (char *) c);
rsetnull(CSHORTTYPE, (char *) &s);
rsetnull(CINTTYPE, (char *) &i);
```

risnull

Kiểm thử nếu một biến là NULL.

```
int risnull(int t, char *ptr);
```

Hàm này nhận dạng biến để kiểm thử (t) cũng như một con trỏ tới biến này (ptr).

Lưu ý rằng cái sau cần phải được phát tới một char*. Xem hàm rsetnull để có một danh sách các dạng biến có khả năng.

Đây là một ví dụ về cách sử dụng hàm này:

```
$char c[] = "abc ";
```

```
$short s = 17;
$int i = -74874;

risnull(CCHARTYPE, (char *) c);
risnull(CSHORTTYPE, (char *) &s);
risnull(CINTTYPE, (char *) &i);
```

33.10.5. Các hằng bổ sung

Lưu ý rằng tất cả các hằng ở đây mô tả các lỗi và tất cả chúng được xác định để đại diện cho các giá trị âm.

Trong các mô tả các hằng khác nhau bạn cũng có thể thấy giá trị rằng các hằng đó đại diện trong triển khai hiện hành. Tuy nhiên, bạn không nên dựa vào số này. Dù vậy bạn có thể dựa vào thực tế là tất cả chúng được xác định để đại diện cho các giá trị âm.

ECPG_INFORMIX_NUM_OVERFLOW

Các hàm trả về giá trị này nếu một sự tràn xảy ra trong một tính toán. Trong nội bộ nó được xác định về -1200 (định nghĩa của Informix).

ECPG_INFORMIX_NUM_UNDERFLOW

Các hàm trả về giá trị này nếu một sự chảy ngầm xảy ra trong một tính toán. Trong nội bộ nó được xác định về -1200 (định nghĩa của Informix).

ECPG_INFORMIX_DIVIDE_ZERO

Các hàm trả về giá trị này nếu một cố gắng để chia cho zero được quan sát thấy. Trong nội bộ nó được xác định về -1200 (định nghĩa của Informix).

ECPG_INFORMIX_BAD_YEAR

Các hàm trả về giá trị này nếu một giá trị tồi cho một năm được thấy trong khi phân tích cú pháp một ngày tháng. Trong nội bộ nó được xác định về -1204 (định nghĩa của Informix).

ECPG_INFORMIX_BAD_MONTH

Các hàm trả về giá trị này nếu một giá trị tồi cho một tháng từng được thấy trong khi phân tích cú pháp ngày tháng. Trong nội bộ nó được xác định về -1205 (định nghĩa của Informix).

ECPG_INFORMIX_BAD_DAY

Các hàm trả về giá trị này nếu một giá trị tồi cho một ngày được thấy trong khi phân tích cú pháp ngày tháng. Trong nội bộ nó được xác định về -1206 (định nghĩa của Informix).

ECPG_INFORMIX_ENOSHORTDATE

Các hàm trả về giá trị này nếu một thủ tục phân tích cú pháp cần một đại diện ngày tháng ngắn gọn nhưng không có chuỗi ngày tháng ở độ dài đúng. Trong nội bộ nó được xác định về -1209 (định nghĩa của Informix).

ECPG_INFORMIX_DATE_CONVERT

Các hàm trả về giá trị này nếu một lỗi xảy ra trong quá trình định dạng ngày tháng. Trong nội bộ nó được xác định về -1210 (định nghĩa của Informix).

ECPG_INFORMIX_OUT_OF_MEMORY

Các hàm trả về giá trị này nếu bộ nhớ từng cạn kiệt trong quá trình hoạt động của chúng. Trong nội bộ nó được xác định về -1211 (định nghĩa của Informix).

ECPG_INFORMIX_ENOTDMY

Các hàm trả về giá trị này nếu một thủ tục phân tích cú pháp từng được hỗ trợ để có được một mặt nạ định dạng (giống như mmddyy) nhưng không phải tất cả các trường từng được liệt kê đúng. Trong nội bộ nó được xác định về -1212 (định nghĩa của Informix).

ECPG_INFORMIX_BAD_NUMERIC

Các hàm trả về giá trị này hoặc nếu một thủ tục phân tích cú pháp không thể phân tích đại diện văn bản cho một giá trị số vì nó chứa các lỗi hoặc nếu một thủ tục không thể hoàn thành một tính toán có liên quan tới các biến số vì ít nhất một trong các biến số là không hợp lệ. Trong nội bộ nó được xác định về -1213 (định nghĩa của Informix).

ECPG_INFORMIX_BAD_EXPONENT

Các hàm trả về giá trị này nếu trong nội bộ nó được xác định về -1216 (định nghĩa của Informix).

ECPG_INFORMIX_BAD_DATE

Các hàm trả về giá trị này nếu trong nội bộ nó được xác định về -1218 (định nghĩa của Informix).

ECPG_INFORMIX_EXTRA_CHARS

Các hàm trả về giá trị này nếu trong nội bộ nó được xác định về -1264 (định nghĩa của Informix).

33.11. Điều khiển lỗi

Phần này mô tả cách mà bạn có thể điều khiển các điều kiện ngoại lệ và các cảnh báo trong một chương trình SQL nhúng. Có vài tiện ích không độc quyền cho điều này.

33.11.1. Thiết lập gọi ngược lại

Một phương pháp đơn giản để bắt các lỗi và cảnh báo là thiết lập một hành động đặc thù sẽ được thực thi bất kỳ khi nào một điều kiện đặc biệt xảy ra. Nói chung:

```
EXEC SQL WHENEVER condition action;
```

condition có thể là một trong những thứ sau đây:

SQLERROR

Hành động được chỉ định được gọi bất kỳ khi nào một cảnh báo xảy ra trong khi thực thị một lệnh SQL.

SQLWARNING

Hành động được chỉ định được gọi bất kỳ khi nào một cảnh báo xảy ra trong khi thực thi một lệnh SQL.

NOT FOUND

Hành động được chỉ định được gọi bất kỳ khi nào một lệnh SQL truy xuất hoặc ảnh hưởng tới các hàng zero. (Điều kiện này không phải là một lỗi, nhưng bạn có thể có quan tâm trong việc điều khiển nó một cách đặc biệt).

action có thể là một trong những điều sau đây:

CONTINUE

Điều này có ý nghĩa hiệu lực rằng điều kiện bị bỏ qua. Đây là mặc định.

GOTO label
GO TO label

Nhảy tới nhãn được chỉ định (sử dụng một lệnh goto của C).

SQLPRINT

In một thông điệp tới một lỗi tiêu chuẩn. Điều này là hữu dụng cho các chương trình đơn giản hoặc trong khi làm mẫu. Các chi tiết của thông điệp không thể được thiết lập cấu hình.

STOP

Gọi exit(1), nó sẽ kết thúc chương trình đó.

DO BREAK

Thực thi lệnh C break. Điều này chỉ được sử dụng trong các vòng lặp hoặc các lệnh switch.

CALL name (args)
DO name (args)

Gọi các hàm C được chỉ định với các đối số được chỉ định.

Tiêu chuẩn SQL chỉ cung cấp cho các hành động CONTINUE và GOTO (và GO TO).

Đây là một ví dụ mà bạn có thể muốn sử dụng trong một chương trình đơn giản. Nó in một thông điệp đơn giản khi một cảnh báo xảy ra và bỏ qua chương trình khi một lỗi xảy ra:

```
EXEC SQL WHENEVER SQLWARNING SQLPRINT;
EXEC SQL WHENEVER SQLERROR STOP;
```

Lệnh EXEC SQL WHENEVER là một chỉ thị của trình tiền xử lý SQL, không phải là một lệnh C. Các hành động lỗi hoặc cảnh báo mà nó thiết lập áp dụng cho các lệnh SQL nhúng mà xuất hiện dưới điểm nơi mà trình điều khiển được thiết lập, trừ phi một hành động khác đã được thiết lập cho điều kiện y hệt giữa EXEC SQL WHENEVER đầu tiên và lệnh SQL gây ra điều kiện đó, bất kể dòng kiểm soát trong chương C thế nào. Vì thế không cái nào trong 2 trích đoạn chương trình C sau đây sẽ có được hiệu ứng mong muốn:

```
/*
* WRONG
*/
int main(int argc, char *argv[])
{
        ...
        if (verbose) {
        EXEC SQL WHENEVER SQLWARNING SQLPRINT;
        }
        ...
        EXEC SQL SELECT ...;
        ...
}
/*
* WRONG
*/
int main(int argc, char *argv[])
```

```
{
        ...
        set_error_handler();
        ...
        EXEC SQL SELECT ...;
        ...
}
static void set_error_handler(void)
{
        EXEC SQL WHENEVER SQLERROR STOP;
}
```

33.11.2. sqlca

Đối với việc điều khiển lỗi mạnh hơn, giao diện SQL nhúng cung cấp một biến tổng thể với tên sqlca mà có cấu trúc sau:

```
struct
{
        char sqlcaid[8];
        long sqlabc;
        long sqlcode;
        struct
        {
                int sqlerrml;
                char sqlerrmc[SQLERRMC_LEN];
        } sqlerrm;
        char sqlerrp[8];
        long sqlerrd[6];
        char sqlwarn[8];
        char sqlstate[5];
} sqlca;
```

(Trong một chương trình đa luồng, mỗi luồng tự động có bản sao sqlca của riêng nó. Điều này làm việc tương tự như với việc điều khiển biến toàn thể C tiêu chuẩn errno).

sqlca bao trùm cả các cảnh báo và các lỗi. Nếu nhiều cảnh báo hoặc lỗi xảy ra trong quá trình thực thi một lệnh, thì sqlca sẽ chỉ có thông tin về cái sau.

Nếu không lỗi nào xảy ra trong lệnh SQL, thì sqlca.sqlcode sẽ là 0 và sqlca.sqlstate sẽ là "00000". Nếu một cảnh báo hoặc lỗi xảy ra, thì sqlca.sqlcode sẽ là âm và sqlca.sqlstate sẽ khác với "00000". Một sqlca.sqlcode dương chỉ ra một điều kiện vô hại, như truy vấn cuối cùng đã trả về các hàng zero, sqlcode và sqlstate là 2 sơ đồ mã lỗi khác nhau; các chi tiết có bên dưới.

Nếu lệnh SQL cuối cùng đã thành công, thì sqlca.sqlerrd[1] chứa OID của hàng được xử lý, nếu áp dụng được, và sqlca.sqlerrd[2] chứa số các hàng được xử lý hoặc được trả về, nếu áp dụng được cho lệnh đó.

Trong trường hợp có lỗi hoặc cảnh báo, sqlca.sqlerrm.sqlerrmc sẽ chứa một chuỗi mà mô tả lỗi đó. Trường sqlca.sqlerrm.sqlerrml chứa độ dài thông điệp lỗi được lưu trữ trong sqlca.sqlerrm.sqlerrmc (kết quả của strlen(), không thực sự thú vị đối với một lập trình viên C). Lưu ý là một số thông điệp là quá dài để vừa trong mảng kích cỡ cố định sqlerrmc; chúng sẽ bị cắt ngắn bớt.

Trong trường hợp một cảnh báo, sqlca.sqlwarn[2] được thiết lập về W. (Trong tất cả các trường hợp khác, nó được thiết lập về thứ gì đó khác với W). Nếu sqlca.sqlwarn[1] được thiết lập về W, thì sau

đó một giá trị đã được cắt bớt khi nó từng được lưu trữ trong một biến chủ host. sqlca.sqlwarn[0] được thiết lập về W nếu bất kỳ yếu tố nào khác được thiết lập để chỉ ra một cảnh báo.

Các trường sqlcaid, sqlcabc, sqlerrp và các yếu tố còn lại của sqlerrd và sqlwarn hiện không chứa thông tin hữu dụng nào.

Cấu trúc sqlca không được xác định trong tiêu chuẩn SQL, nhưng được triển khai trong vài hệ thống cơ sở dữ liệu SQL khác. Các định nghĩa là tương tự về cốt lõi, nhưng nếu bạn muốn viết các ứng dụng khả chuyển, thì bạn nên điều tra nghiên cứu các triển khai khác một cách thận trọng.

33.11.3. SQLSTATE vs SQLCODE

Các trường sqlca.sqlstate và sqlca.sqlcode là 2 sơ đồ khác nhau cung cấp các mã lỗi. Cả 2 đều xuất xứ từ tiêu chuẩn SQL, nhưng SQLCODE từng được đánh dấu không được tán thành trong phiên bản SQL-92 của tiêu chuẩn đó và đã bị bỏ trong các phiên bản sau đó. Vì thế, các ứng dụng mới được khuyến cáo mạnh mẽ sử dụng SQLSTATE.

SQLSTATE là một mảng 5 ký tự. 5 ký tự đó gồm các chữ số hoặc các ký tự hoa mà đại diện cho các mã của các điều kiện cảnh báo và lỗi khác nhau. SQLSTATE có một sơ đồ có thứ bậc: 2 ký tự đầu tiên chỉ lớp điều kiện chung, 3 ký tự cuối chỉ một lớp con của điều kiện chung đó. Một tình trạng thành công được chỉ định bằng mã 00000. Các mã SQLSTATE là cho hầu hết các phần được xác định trong tiêu chuẩn SQL. Máy chủ PostgreSQL bẩm sinh hỗ trợ các mã lỗi SQLSTATE; vì thế một mức độ cao tính ổn định có thể đạt được bằng việc sử dụng sơ đồ mã lỗi này khắp tất cả các ứng dụng. Để có thêm thông tin, xem Phụ lục A.

SQLCODE, sơ đồ mã lỗi bị phản đối, là một số nguyên đơn giản. Một giá trị 0 chỉ thành công, một giá trị dương chỉ thành công với thông tin điều kiện, một giá trị âm chỉ một lỗi. Tiêu chuẩn SQL chỉ xác định giá trị dương +100, nó chỉ rằng lệnh cuối cùng đã trả về hoặc đã ảnh hưởng tới các hàng zero, và không giá trị âm đặc biệt nào. Vì thế, sơ đồ này chỉ có thể đạt được tính khả chuyển kém và không có một chỉ định mã có tôn ti trật tự.

Về lịch sử, trình xử lý SQL nhúng cho PostgreSQL đã chỉ định vài giá trị SQLCODE đặc thù cho sử dụng của nó, điều sẽ được liệt kê bên dưới bằng giá trị số của chúng và tên biểu tượng của chúng. Hãy nhớ rằng chúng không khả chuyển đổi với các triển khai SQL khác. Để đơn giản hóa việc chuyển các ứng dụng tới sơ đồ SQLSTATE, SQLSTATE tương ứng cũng được liệt kê. Tuy nhiên, không có việc ánh xạ một - một hoặc một - nhiều giữa 2 sơ đồ đó (quả thực nó là nhiều - nhiều), vì thế bạn nên tư vấn danh sách SQLSTATE tổng thế trong Phục lục A theo từng trường hợp.

Có các giá trị SQLCODE được chỉ định:

-12 (ECPG_OUT_OF_MEMORY)

Chỉ rằng bộ nhớ ảo của bạn cạn kiệt. (SQLSTATE YE001)

-200 (ECPG_UNSUPPORTED)

Chỉ bộ tiền xử lý đã sinh ra thứ gì đó mà thư viện không biết.

Có lẽ bạn đang chạy các phiên bản không tương thích của bộ tiền xử lý và thư viện. (SQLSTATE YE002)

-201 (ECPG_TOO_MANY_ARGUMENTS)

Điều này có nghĩa là lệnh được chỉ định ít hơn các biến chủ host mà lệnh đã kỳ vọng. (SQLSTATE 07001 hoặc 07002)

-202 (ECPG_TOO_FEW_ARGUMENTS)

Điều này có nghĩa là lệnh được chỉ định ít hơn các biến chủ host so với lệnh được kỳ vọng.

(SQLSTATE 07001 hoặc 07002)

-203 (ECPG_TOO_MANY_MATCHES)

Điều này có nghĩa một truy vấn đã trả về nhiều hàng nhưng lệnh đã chỉ được chuẩn bị để lưu trữ một hàng kết quả (ví dụ, vì các biến được chỉ định không phải là các mảng). (SQLSTATE 21000).

-204 (ECPG_INT_FORMAT)

Biến chủ host là dạng int và các dữ liệu trong cơ sở dữ liệu là dạng khác nhau và chứa một giá trị không thể được biên dịch như một int. Thư viện sử dụng strtol() cho biến đổi này. (SQLSTATE 42804)

-205 (ECPG_UINT_FORMAT)

Biến chủ host là dạng unsigned int và các dữ liệu trong cơ sở dữ liệu là dạng khác và chứa một giá trị mà không thể được biên dịch như một unsigned int. Thư viện sử dụng strtoul() cho biến đổi này. (SQLSTATE 42804)

-206 (ECPG_FLOAT_FORMAT)

Biến chủ host là dạng float và các dữ liệu trong cơ sở dữ liệu là dạng khác và chứa một giá trị không thể được biên dịch như một float. Thư viện sử dụng strtod() cho biến đổi này. (SQLSTATE 42804)

-211 (ECPG_CONVERT_BOOL)

Điều này có nghĩa là biến chủ host là dạng và các dữ liệu trong cơ sở dữ liệu hoặc không 't' cũng không 'f'. (SQLSTATE 42804)

-212 (ECPG_EMPTY)

Lệnh được gửi tới máy chủ PostgreSQL là rỗng. (Điều này không thể xảy ra bình thường trong một chương trình SQL nhúng, vì thế nó có thể chỉ tới một lỗi trong nội bộ). (SQLSTATE YE002)

-213 (ECPG_MISSING_INDICATOR)

Một giá trị null đã được trả về và không biến chỉ số null nào đã được cung ứng. (SQLSTATE 22002)

-214 (ECPG_NO_ARRAY)

Một biến thông thường đã được sử dụng tại chỗ mà đòi hỏi một mảng. (SQLSTATE 42804)

-215 (ECPG_DATA_NOT_ARRAY)

Cơ sở dữ liệu đã trả về một biến thông thường tại chỗ mà đòi hỏi giá trị mảng. (SQLSTATE

42804)

-220 (ECPG_NO_CONN)

Chương trình đã cố gắng truy cập một kết nối mà không tồn tại. (SQLSTATE 08003)

-221 (ECPG_NOT_CONN)

Chương trình cố truy cập một kết nối mà không tồn tại mà không mở. (Đây là một lỗi trong nội bộ). (SQLSTATE YE 002)

-230 (ECPG_INVALID_STMT)

Lệnh mà bạn đang cố sử dụng còn chưa được chuẩn bị. (SQLSTATE 26000)

-240 (ECPG_UNKNOWN_DESCRIPTOR)

Chỉ số trình mô tả được chỉ định đã không được tìm thấy. Lệnh bạn đang cố sử dụng còn chưa được chuẩn bị. (SQLSTATE 33000)

-241 (ECPG_INVALID_DESCRIPTOR_INDEX)

Chỉ số trình mô tả được chỉ định nằm ngoài dãy. (SQLSTATE 07009)

-242 (ECPG_UNKNOWN_DESCRIPTOR_ITEM)

Một khoản của trình mô tả không hợp lệ đã được yêu cầu. (Đây là một lỗi nội bộ). (SQLSTATE YE002)

-243 (ECPG_VAR_NOT_NUMERIC)

Trong quá trình thực thi một lệnh động, cơ sở dữ liệu đã trả về một giá trị số và biến chủ host từng không phải là số. (SQLSTATE 07006)

-244 (ECPG_VAR_NOT_CHAR)

Trong quá trình thực thi một lệnh động, cơ sở dữ liệu đã trả về một giá trị không phải số và biến chủ host đã là số. (SQLSTATE 07006)

-400 (ECPG_PGSQL)

Vài lỗi do máy chủ PostgreSQL sinh ra. Thông điệp đó chứa thông điệp lỗi từ máy chủ PostgreSQL.

-401 (ECPG_TRANS)

Máy chủ PostgreSQL đã đánh tín hiệu rằng chúng ta không thể khởi động, thực hiện, hoặc quay lại giao dịch đó. (SQLSTATE 09007)

-402 (ECPG_CONNECT)

Kết nối cố gắng tới cơ sở dữ liệu đã không thành công. (SQLSTATE 08001)

100 (ECPG_NOT_FOUND)

Đây là một điều kiện vô hại chỉ ra rằng lệnh cuối đã truy xuất hoặc đã xử lý các hàng zero, hoặc là bạn ở cuối của con trỏ. (SQLSTATE 02000)

33.12. Các hướng dẫn của bộ tiền xử lý

33.12.1. Bao gồm các tệp

Để đưa một tệp bên ngoài vào trong chương trình SQL nhúng của bạn, hãy sử dụng:

```
EXEC SQL INCLUDE filename;
```

Bộ tiền xử lý SQL nhúng sẽ tìm một tệp tên là filename.h, tiền xử lý nó, và đưa nó vào kết quả đầu ra C. Vì thế, các lệnh SQL nhúng trong tệp được đưa vào sẽ được điều khiển đúng.

Lưu ý rằng đây không phải là y hệt như:

```
#include <filename.h>
```

vì tệp này có thể không tuân theo việc tiền xử lý lệnh SQL. Về bản chất tự nhiên, bạn có thể tiếp tục sử dụng hướng dẫn #include của C để đưa vào các tệp đầu đề khác.

> **Lưu ý**: Tên tệp được đưa vào là phân biệt chữ hoa chữ thường, thậm chí dù phần còn lại của lệnh EXEC SQL INCLUDE tuân theo các qui tắc thông thường về phân biệt chữ hoa chữ thường của SQL.

33.12.2. Các hướng dẫn #define và #undef

Tương tự với hướng dẫn #define mà được biết từ C, SQL nhúng có một khái niệm tương tự:

```
EXEC SQL DEFINE name;
EXEC SQL DEFINE name value;
```

Nên bạn có thể xác định một cái tên:

```
EXEC SQL DEFINE HAVE_FEATURE;
```

Và bạn cũng có thể xác định các hằng số:

```
EXEC SQL DEFINE MYNUMBER 12;
EXEC SQL DEFINE MYSTRING 'abc';
```

Hãy sử dụng undef để loại bỏ một định nghĩa trước đó:

```
EXEC SQL UNDEF MYNUMBER;
```

Tất nhiên bạn có thể tiếp tục sử dụng các phiên bản #define và #undef của C trong chương trình SQL nhúng của bạn. Sự khác biệt là ở những nơi các giá trị được xác định của bạn được đánh giá. Nếu bạn sử dụng EXEC SQL DEFINE thì bộ tiền xử lý ecpg đánh giá các định nghĩa và thay thế các giá trị. Ví dụ, nếu bạn viết:

```
EXEC SQL DEFINE MYNUMBER 12;
...
EXEC SQL UPDATE Tbl SET col = MYNUMBER;
```

thì ecpg sẽ thực hiện rồi sự thay thế và trình biên dịch C của bạn sẽ không bao giờ thấy bất kỳ tên hay mã định danh MYNUMBER nào. Lưu ý rằng bạn không thể sử dụng #define cho một hằng số mà bạn đang định sử dụng trong một truy vấn SQL nhúng vì trong trường hợp này trình tiền xử lý SQL nhúng không có khả năng thấy khai báo này.

33.12.3. Các hướng dẫn ifdef, ifndef, else, elif, và endif

Bạn có thể sử dụng các hướng dẫn sau để biên dịch các phần mã theo điều kiện:

EXEC SQL ifdef name;

Kiểm tra một tên và xử lý các dòng tiếp sau nếu name đã được tạo ra với EXEC SQL define name;

EXEC SQL ifndef name;

Kiểm tra một tên và xử lý các dòng tiếp sau nếu name còn chưa được tạo ra với EXEC SQL define name.

EXEC SQL else;

Khởi động việc xử lý một phần lựa chọn cho một phần được hoặc EXEC SQL ifdef name hoặc EXEC SQL ifndef name giới thiệu.

EXEC SQL elif name;

Kiểm tra name và khởi tạo một phần lựa chọn nếu name từng được tạo ra với EXEC SQL define name.

EXEC SQL endif;

Kết thúc một phần lựa chọn.

Ví dụ:

```
EXEC SQL ifndef TZVAR;
EXEC SQL SET TIMEZONE TO 'GMT';
EXEC SQL elif TZNAME;
EXEC SQL SET TIMEZONE TO TZNAME;
EXEC SQL else;
EXEC SQL SET TIMEZONE TO TZVAR;
EXEC SQL endif;
```

33.13. Xử lý các chương trình SQL nhúng

Bây giờ bạn có một ý tưởng làm thế nào để tạo các chương trình C SQL nhúng, bạn có lẽ muốn biết làm thế nào để biên dịch chúng. Trước khi biên dịch bạn hãy chạy tệp qua trình tiền xử lý C SQL nhúng, nó biến đổi các lệnh SQL bạn đã sử dụng thành các lời gọi hàm đặc biệt. Sau khi biên dịch, bạn phải liên kết với một thư viện đặc biệt mà chứa các hàm cần thiết. Các hàm đó lấy thông tin từ các đối số, thực hiện lệnh SQL bằng việc sử dụng giao diện libpq, và đặt kết quả trong các đối số được chỉ định cho đầu ra.

Chương trình của bộ tiền xử lý được gọi là ecpg và được đưa vào trong một cài đặt PostgreSQL thông thường. Các chương trình SQL nhúng thường được đặt tên với một phần mở rộng .pgc. Nếu bạn có một tệp chương trình được gọi là prog1.pgc, thì bạn có thể xử lý nó đơn giản bằng việc gọi:

```
ecpg prog1.pgc
```

Điều này sẽ tạo ra một tệp gọi là prog1.c. Nếu các tệp đầu vào của bạn không tuân theo mẫu đặt tên được gợi ý, thì bạn có thể chỉ định tệp đầu ra một cách rõ ràng bằng việc sử dụng lựa chọn -o.

Tệp được tiền xử lý có thể được biên dịch bình thường, ví dụ:

```
cc -c prog1.c
```

Các tệp nguồn C được sinh ra bao gồm các tệp đầu đề từ cài đặt PostgreSQL, nên nếu bạn đã cài đặt PostgreSQL vào một vị trí mà mặc định không tìm thấy, thì bạn phải thêm một lựa chọn như -I/usr/local/pgsql/include vào dòng lệnh biên dịch.

Để liên kết một chương trình SQL nhúng, bạn cần đưa vào thư viện libecpg, giống thế này:

```
cc -o myprog prog1.o prog2.o ... -lecpg
```

Một lần nữa, bạn có thể phải thêm một lựa chọn như -L/usr/local/pgsql/lib vào dòng lệnh.

Nếu bạn định xây dựng tiến trình của một dự án lớn hơn bằng việc sử dụng make, thì có thể là thuận tiện để đưa qui tắc ngầm hiểu sau vào các tệp makefile:

```
ECPG = ecpg
%.c: %.pgc
        $(ECPG) $<
```

Cú pháp hoàn chỉnh của lệnh ecpg được chi tiết hóa trong ecpg.

Thư viện ecpg là luồng an toàn một cách mặc định. Tuy nhiên, bạn có thể cần sử dụng vài lựa chọn dòng lệnh theo luồng để biên dịch mã máy trạm của bạn.

33.14. Các hàm thư viện

Thư viện libecpg trước hết chứa các hàm "ẩn" được sử dụng để triển khai chức năng được các lệnh SQL nhúng thể hiện. Nhưng có vài hàm có thể hữu dụng được gọi trực tiếp. Lưu ý rằng điều này làm cho mã của bạn không khả chuyển được.

- ECPGdebug(int on, FILE *stream) bật việc lưu ký gỡ lỗi nếu được gọi với đối số đầu tiên khác zero. Việc lưu ký gỡ lỗi được thực hiện trong stream. Lưu ký chứa tất cả các lệnh SQL với tất cả các biến đầu vào được chèn vào, và các kết quả từ máy chủ PostgreSQL. Điều này có thể là rất hữu dụng khi tìm kiếm các lỗi trong các lệnh SQL.

  > **Lưu ý**: Trong Windows, nếu các thư viện ecpg và một ứng dụng được biên dịch với các cờ khác nhau, thì lời gọi hàm này sẽ làm hỏng ứng dụng vì sự đại diện bên trong của các con trỏ FILE khác nhau. Đặc biệt, các cờ đa luồng/đơn luồng, phát hành/gỡ lỗi, và tĩnh/động (multithreaded/single-threaded, release/debug, and static/dynamic flags) sẽ là y hệt đối với thư viện đó và tất cả các ứng dụng sử dụng thư viện đó.

- ECPGget_PGconn(const char *connection_name) trả về sự điều khiển kết nối cơ sở dữ liệu thư viện được nhận diện bằng tên được đưa ra. Nếu connection_name được thiết lập về NULL, thì sự điều khiển kết nối hiện hành được trả về. Nếu không điều khiển kết nối nào có thể được nhận diện, thì hàm đó trả về NULL. Điều kiện kết nối được trả về đó có thể được sử dụng để gọi bất kỳ hàm nào khác với libpq, nếu cần.

  > **Lưu ý**: Là ý tưởng tồi để điều khiển các kết nối cơ sở dữ liệu được làm từ ecpg trực tiếp với các thủ tục libpq.

- ECPGtransactionStatus(const char *connection_name) trả về tình trạng giao dịch hiện hành của kết nối được đưa ra được connection_name nhận diện. Xem Phần 31.2 và PQtransactionStatus() của libpq để có các chi tiết về các mã tình trạng được trả về.

- ECPGstatus(int lineno, const char* connection_name) trả về đúng nếu bạn được kết nối tới một cơ sở dữ liệu và sai nếu không, connection_name có thể là NULL nếu một kết nối duy nhất đang được sử dụng.

33.15. Nội bộ bên trong

Phần này giải thích cách mà ECPG làm việc trong nội bộ. Thông tin này có thể ngẫu nhiên là hữu dụng để giúp những người sử dụng hiểu cách sử dụng ECPG.

4 dòng đầu tiên được ecpg viết cho đầu ra là các dòng cố định. 2 là các bình luận và 2 là các dòng thêm vào cần thiết để giao diện với thư viện đó. Rồi thì trình tiền xử lý đọc qua tệp và ghi đầu ra. Thông thường nó chỉ nói lại mọi điều tới đầu ra.

Khi nó thấy một lệnh EXEC SQL, nó can thiệp vào và thay đổi nó. Lệnh đó bắt đầu với EXEC SQL và kết thúc bằng ; . Mọi điều ở giữa được đối xử như một lệnh SQL và được phân tích cú pháp cho sự thay thế các biến.

Sự thay các biến xảy ra khi một biểu tượng khởi động với một dấu hai chấm (:). Biến đó với tên được tra cứu giữa các biến đã được khai báo trước đó bên trong một phần EXEC SQL DECLARE.

Nó lấy một số đối số khác nhau. Điều này có thể dễ dàng thêm tới 50 đối số hoặc khoảng đó, và chúng ta hy vọng điều này sẽ không phải là một vấn đề trong bất kỳ nền tảng nào.

Các đối số đó là:

Số dòng

Đây là số dòng của dòng gốc: được sử dụng chỉ trong thông điệp lỗi.

Chuỗi

Đây là lệnh SQl sẽ được đưa ra. Nó được biến đầu vào sửa đổi, như, các biến mà trong khi không được biết lúc biên dịch nhưng được đưa vào trong lệnh. Trong đó các biến sẽ đi tới chuỗi chứa chăng?

Biến đầu vào

Mỗi biến đầu vào làm cho 10 đối số sẽ được tạo ra. (Xem bên dưới).

ECPGt_EOIT

Một enum nói rằng không có thêm các biến đầu vào nữa.

Biến đầu ra

Mỗi biến đầu ra làm cho 10 đối số sẽ được tạo ra. (Xem bên dưới). Các biến đó được hàm này điền đầy.

ECPGt_EORT

Một enum nói rằng không có các biến hơn nữa.

Đối với từng biến là một phần của lệnh SQL, hàm có 10 đối số:

1. Dạng như một biểu tượng đặc biệt.
2. Một con trỏ tới giá trị hoặc một con trỏ tới con trỏ đó.
3. Kích cỡ của biến nếu nó là char hoặc varchar.
4. Số các yếu tố trong mảng (đối với mảng lấy về).
5. Phần bù trừ đối với yếu tố tiếp theo trong mảng (đối với mảng lấy về).
6. Dạng biến chỉ số như một biểu tượng đặc biệt.

7. Một con trỏ tới biến chỉ số.
8. 0
9. Số các yếu tố trong mảng các chỉ số (đối với mảng lấy về).
10. Phần bù cho yếu tố tiếp sau trong mảng chỉ số (đối với mảng lấy về).

Lưu ý rằng không phải tất cả các lệnh SQL đều được đối xử theo cách này. Ví dụ, một lệnh con trỏ mở giống như:

```
EXEC SQL OPEN cursor;
```

không được sao chép tới đầu ra. Thay vào đó, lệnh DECLARE của con trỏ được sử dụng ở vị trí của lệnh OPEN vì nó quả thực mở ra con trỏ.

Đây là một ví dụ hoàn chỉnh mô tả đầu ra của trình tiền xử lý của một tệp foo.pgc (các chi tiết có thể thay đổi với từng phiên bản đặc biệt của trình tiền xử lý):

```
EXEC SQL BEGIN DECLARE SECTION;
int index;
int result;
EXEC SQL END DECLARE SECTION;
...
EXEC SQL SELECT res INTO :result FROM mytable WHERE index = :index;
```

được dịch thành:

```
/* Processed by ecpg (2.6.0) */
/* These two include files are added by the preprocessor */
#include <ecpgtype.h>;
#include <ecpglib.h>;
/* exec sql begin declare section */
#line 1 "foo.pgc"
int index;
int result;
/* exec sql end declare section */
...
ECPGdo(__LINE__, NULL, "SELECT res FROM mytable WHERE index = ? ",
        ECPGt_int,&(index),1L,1L,sizeof(int),
        ECPGt_NO_INDICATOR, NULL , 0L, 0L, 0L, ECPGt_EOIT,
        ECPGt_int,&(result),1L,1L,sizeof(int),
        ECPGt_NO_INDICATOR, NULL , 0L, 0L, 0L, ECPGt_EORT);
#line 147 "foo.pgc"
```

(Thụt lề ở đây được thêm vào vì khả năng đọc được và không phải thứ gì đó trình tiền xử lý làm).

Chương 34. Sơ đồ thông tin

Sơ đồ thông tin gồm một tập hợp các kiểu nhìn chứa thông tin về các đối tượng được xác định trong cơ sở dữ liệu hiện hành. Sơ đồ thông tin được xác định trong tiêu chuẩn SQL và có thể vì thế được kỳ vọng sẽ là khả chuyển và giữ được ổn định - không giống như các catalog hệ thống là đặc thù cho PostgreSQL và được mô hình hóa sau sự quan tâm triển khai. Tuy nhiên, các kiểu nhìn sơ đồ thông tin không chứa thông tin về các tính năng đặc thù PostgreSQL; để có chúng thì bạn cần truy vấn các catalog hệ thống hoặc các kiểu nhìn đặc thù khác của PostgreSQL.

34.1. Sơ đồ

Bản thân sơ đồ thông tin là một sơ đồ được đặt tên là information_schema. Sơ đồ này tự động tồn tại trong tất cả các cơ sở dữ liệu. Chủ sở hữu của sơ đồ này là người sử dụng cơ sở dữ liệu gốc ban đầu trong bó, và người sử dụng đó tự nhiên có tất cả các quyền ưu tiên trong sơ đồ này, bao gồm cả khả năng loại bỏ nó (mà việc lưu các không gian được điều đó lưu trữ là rất nhỏ).

Mặc định, sơ đồ thông tin không nằm trong đường tìm kiếm sơ đồ đó, nên bạn cần phải truy cập tất cả các đối tượng trong nó qua các tên có đủ điều kiện. Vì các tên của vài đối tượng trong sơ đồ thông tin là các tên chung mà có thể xảy ra trong các ứng dụng của người sử dụng, bạn nên cẩn thận nếu bạn muốn đặt sơ đồ thông tin vào đường dẫn đó.

34.2. Các dạng dữ liệu

Các cột các kiểu nhìn sơ đồ thông tin sử dụng các dạng dữ liệu đặc biệt được xác định trong sơ đồ thông tin. Chúng được xác định như là các miền đơn giản hơn là các dạng thường được xây dựng sẵn. Bạn không nên sử dụng các dạng đó cho công việc bên ngoài sơ đồ thông tin, mà các ứng dụng của bạn phải được chuẩn bị cho chúng nếu chúng chọn từ sơ đồ thông tin đó.

Các dạng đó là:

cardinal_number
: Một số nguyên không âm

character_data
: Một chuỗi ký tự (không có độ dài tối đa đặc thù).

sql_identifier
: Một chuỗi ký tự. Dạng này được sử dụng cho mã định danh SQL, dạng character_data được sử dụng cho bất kỳ dạng dữ liệu văn bản nào khác.

time_stamp
: Một miền đối với dạng timestamp with time zone (dấu thời gian với vùng thời gian)

yes_or_no
: Một miền chuỗi ký tự chứa hoặc YES hoặc NO. Điều này được sử dụng để đại diện cho dữ liệu Boolean (đúng/sai – true/false) trong sơ đồ thông tin. (Sơ đồ thông tin từng được tạo ra trước khi dạng boolean đã được đưa vào tiêu chuẩn SQL, nên qui ước này là cần thiết để giữ cho sơ đồ thông tin tương thích ngược được).

Mỗi cột trong sơ đồ thông tin có một trong số 5 dạng đó.

34.3. information_schema_catalog_name

information_schema_catalog_name là một bảng mà luôn chứa một hàng và một cột có tên của cơ sở dữ liệu hiện hành (catalog hiện hành, trong thuật ngữ SQL).

Bảng 34-1. Các cột information_schema_catalog_name

Tên	Dạng dữ liệu	Mô tả
catalog_name	sql_identifier	Tên cơ sở dữ liệu có chứa sơ đồ thông tin này

34.4. adminstrable_role_authorizations

Kiểu nhìn administrable_role_authorizations nhận diện tất cả các vai trò mà người sử dụng hiện hành có lựa chọn quản trị (admin) cho chúng.

Bảng 34-2. Các cột administrable_role_authorizations

Tên	Dạng dữ liệu	Mô tả
grantee	sql_identifier	Tên của vai trò theo đó cơ chế thành viên của vai trò này đã được trao (có thể là người sử dụng hiện hành, hoặc một vai trò khác trong trường hợp các cơ chế thành viên vai trò lồng nhau.
role_name	sql_identifier	Tên một vai trò
is_grantable	yes_or_no	Luôn là YES

34.5. applicable_roles

Kiểu nhìn applicable_roles nhận diện tất cả các vai trò mà các quyền ưu tiên của chúng người sử dụng hiện hành có thể sử dụng. Điều này có nghĩa là có vài chuỗi vai trò trao từ người sử dụng hiện hành cho vai trò theo yêu cầu. Bản thân người sử dụng hiện hành cũng là một vai trò áp dụng được. Tập hợp các vai trò áp dụng được thường được sử dụng cho việc kiểm tra các quyền.

Bảng 34-3. Các cột applicable_roles

Tên	Dạng dữ liệu	Mô tả
grantee	sql_identifier	Tên của vai trò theo đó cơ chế thành viên của vai trò này đã được trao (có thể là người sử dụng hiện hành, hoặc một vài trò khác trong trường hợp các cơ chế thành viên vai trò lồng nhau)
role_name	sql_identifier	Tên một vai trò
is_grantable	yes_or_no	YES nếu người được cấp có lựa chọn quản trị (admin) trong vai trò, NO nếu không

34.6. attributes

Kiểu nhìn attributes chứa thông tin về các thuộc tính của các dạng dữ liệu tổng hợp được xác định trong cơ sở dữ liệu. (Lưu ý rằng kiểu nhìn đó không đưa ra thông tin về các cột của bảng, điều đôi khi được gọi là các thuộc tính theo các ngữ cảnh của PostgreSQL).

Bảng 34-4. Các cột attributes

Tên	Dạng dữ liệu	Mô tả
udt_catalog	sql_identifier	Tên cơ sở dữ liệu chứa dạng dữ liệu (luôn là cơ sở dữ liệu hiện hành)
udt_schema	sql_identifier	Tên sơ đồ chứa dạng dữ liệu
udt_name	sql_identifier	Tên dạng dữ liêu
attribute_name	sql_identifier	Tên thuộc tính
ordinal_position	cardinal_number	Vị trí theo thứ tự của thuộc tính trong dạng dữ liệu (tính bắt đầu từ 1)
attribute_default	character_data	Biểu thức mặc định của thuộc tính
is_nullable	yes_or_no	YES nếu thuộc tính có khả năng null, NO nếu được biết là không null.
data_type	character_data	Dạng dữ liệu thuộc tính, nếu là dạng được xây dựng sẵn, hoặc ARRAY nếu là vài mảng (trong trường hợp đó, xem kiểu nhìn element_types), nếu không thì USER-DEFINED (trong trường hợp đó, dạng được xác định trong attribute_udt_name và có liên quan tới các cột).
character_maximum_length	cardinal_number	Nếu data_type xác định một ký tự hoặc dạng chuỗi bit, độ dài tối đa được thay đổi; null cho tất cả các dạng dữ liệu khác hoặc nếu không độ dài tối đa nào từng được khai báo.
character_octet_length	cardinal_number	Nếu data_type xác định một dạng ký tự, thì độ dài tối đa có thể theo octets (byte) của một dữ liệu; null cho tất cả các dạng dữ liệu khác. Độ dài octet tối đa phụ thuộc vào độ dài tối đa ký tự được khai báo (xem ở trên) và việc mã hóa máy chủ.
numeric_precision	cardinal_number	Nếu data_type nhận diện một dạng số, thì cột này chứa (được khai báo hoặc ngầm hiểu) độ chính xác dạng cho thuộc tính này. Độ chính xác này chỉ số các chữ số có nghĩa. Nó có thể được biểu diễn theo thập phân (cơ sở 10) hoặc nhị phân (cơ sở 2), như được chỉ định trong cột numeric_precision_radix. Đối với tất cả các dạng dữ liệu khác, cột này là null.
numeric_precision_radix	cardinal_number	Nếu data_type nhận diện một dạng số, thì cột này chỉ ra trong cơ sở nào các giá trị trong các cột numeric_precision và numeric_scale được thể hiện. Giá trị hoặc là 2 hoặc là 10. Đối với tất cả các dạng dữ liệu khác, cột này là null.
numeric_scale	cardinal_number	Nếu data_type nhận diện một dạng số chính xác, cột này chứa phạm vi (được khai báo hoặc ngầm hiểu) của dạng cho thuộc tính này. Phạm vi đó chỉ ra số các chữ số có nghĩa ở bên phải của dấu thập phân. Nó có thể được trình bày theo thập phân (cơ sở 10) hoặc nhị phân (cơ sở 2), như được chỉ định trong cột numeric_precision_radix. Đối với tất cả các dạng dữ liệu khác, cột này là null.
datetime_precision	cardinal_number	Nếu data_type nhận diện một dạng ngày tháng, thời gian, dấu thời gian hoặc khoảng thời gian, thì cột này chứa (được khai báo hoặc ngầm hiểu) độ chính xác phần giây của dạng cho thuộc tính này, đó là, số các chữ số thập phân được duy trì sau dấu thập phân trong giá trị các giây. Đối với tất cả các dạng dữ liệu khác, cột này là null.
interval_type	character_data	Còn chưa được triển khai
interval_precision	character_data	Còn chưa được triển khai
attribute_udt_catalog	sql_identifier	Tên của cơ sở dữ liệu mà dạng dữ liệu thuộc tính được xác định trong đó (luôn là cơ sở dữ liệu hiện hành)
attribute_udt_schema	sql_identifier	Tên sơ đồ mà dạng dữ liệu thuộc tính được xác định trong đó
attribute_udt_name	sql_identifier	Tên dạng dữ liệu thuộc tính

Tên	Dạng dữ liệu	Mô tả
scope_catalog	sql_identifier	Áp dụng cho một tính năng không sẵn có trong PostgreSQL
scope_schema	sql_identifier	Áp dụng cho một tính năng không sẵn có trong PostgreSQL
scope_name	sql_identifier	Áp dụng cho một tính năng không sẵn có trong PostgreSQL
maximum_cardinality	cardinal_number	Luôn null, vì các mảng luôn có bản số tối đa không giới hạn trong PostgreSQL
dtd_identifier	sql_identifier	Một mã định dạng của trình mô tả dạng dữ liệu của cột, độc nhất trong các trình mô tả dạng dữ liệu liên quan tới bảng đó. Điều này chủ yếu hữu dụng cho việc liên kết với các trường hợp khác của các mã định dạng như vậy. (Định dạng đặc thù của mã định danh không được xác định và không được đảm bảo để giữ lại y hệt trong các phiên bản sau)
is_derived_refere	nce_attryiebsutoer_no	Áp dụng cho một tính năng còn chưa có sẵn trong PostgreSQL

Xem thêm Phần 34.12, một kiểu nhìn có cấu trúc tương tự, để có thêm thông tin về một vài cột đó.

34.7. check_constraint_routine_usage

Kiểu nhìn check_constraint_routine_usage nhận diện các thủ tục (các hàm và các thủ tục) mà sẽ được một ràng buộc kiểm tra sử dụng. Chỉ những thủ tục nào được chỉ ra mới được sở hữu bằng một vai trò được phép hiện hành.

Bảng 34-5. Các cột check_constraint_routine_usage

Tên	Dạng dữ liệu	Mô tả
constraint_catalog	sql_identifier	Tên của cơ sở dữ liệu chứa ràng buộc (luôn là cơ sở dữ liệu hiện hành)
constraint_schema	sql_identifier	Tên của sơ đồ chứa ràng buộc
constraint_name	sql_identifier	Tên của ràng buộc
specific_catalog	sql_identifier	Tên của cơ sở dữ liệu chứa hàm (luôn là cơ sở dữ liệu hiện hành)
specific_schema	sql_identifier	Tên của sơ đồ chứa hàm
specific_name	sql_identifier	“Tên đặc biệt” của hàm. Xem Phần 34.33 để có thêm thông tin

34.8. check_constrains

Kiểu nhìn check_constraints chứa tất cả các ràng buộc kiểm tra, hoặc được xác định trong một bảng hoặc trong một miền, được sở hữu bằng một vai trò được kích hoạt hiện hành. (Chủ sở hữu của bảng hoặc miền đó là chủ của ràng buộc).

Bảng 34-6. Các cột check_constrains

Tên	Dạng dữ liệu	Mô tả
constraint_catalog	sql_identifier	Tên của cơ sở dữ liệu chứa ràng buộc (luôn là cơ sở dữ liệu hiện hành)
constraint_schema	sql_identifier	Tên của sơ đồ chứa ràng buộc
constraint_name	sql_identifier	Tên của ràng buộc
check_clause	character_data	Trình bày kiểm tra ràng buộc kiểm tra

34.9. column_domain_usage

Kiểu nhìn column_domain_usage nhận diện tất cả các cột (của một bảng hoặc một kiểu nhìn) mà sử dụng vài miền được xác định trong cơ sở dữ liệu hiện hành và được một vai trò được kích hoạt hiện hành làm chủ.

Bảng 34.7. Các cột column_domain_usage

Tên	Dạng dữ liệu	Mô tả
domain_catalog	sql_identifier	Tên của cơ sở dữ liệu chứa miền (luôn là cơ sở dữ liệu hiện hành)
domain_schema	sql_identifier	Tên của sơ đồ chứa miền
domain_name	sql_identifier	Tên của miền
table_catalog	sql_identifier	Tên của cơ sở dữ liệu chứa bảng (luôn là cơ sở dữ liệu hiện hành)
table_schema	sql_identifier	Tên của sơ đồ chứa bảng
table_name	sql_identifier	Tên của bảng
column_name	sql_identifier	Tên của cột

34.10. column_privileges

Kiểu nhìn column_privileges nhận diện tất cả các quyền ưu tiên được trao trong các cột cho một vai trò hiện hành được kích hoạt hoặc bằng một vai trò hiện được kích hoạt. Có 1 hàng cho từng kết hợp cột, người trao, và người được trao.

Nếu một quyền ưu tiên từng được trao trong toàn bộ một bảng, thì nó sẽ chỉ ra trong kiểu nhìn đó như một sự trao cho từng cột, nhưng chỉ đối với các dạng quyền ưu tiên nơi mà mức độ chi tiết các cột là có khả năng: SELECT, INSERT, UPDATE, REFERENCES.

Bảng 34-8. Các cột column_privileges

Tên	Dạng dữ liệu	Mô tả
grantor	sql_identifier	Tên của vai trò mà đã trao quyền ưu tiên
grantee	sql_identifier	Tên của vai trò mà quyền ưu tiên đã được trao cho
table_catalog	sql_identifier	Tên của cơ sở dữ liệu mà chứa bảng có cột đó (luôn là cơ sở dữ liệu hiện hành)
table_schema	sql_identifier	Tên sơ đồ chứa bảng có cột đó
table_name	sql_identifier	Tên bảng chứa cột đó
column_name	sql_identifier	Tên cột đó
privilege_type	character_data	Dạng quyền ưu tiên: SELECT, INSERT, UPDATE, hoặc REFERENCES
is_grantable	yes_or_no	YES nếu quyền ưu tiên được trao, NO nếu không

34.11. column_udt_usage

Kiểu nhìn column_udt_usage nhận diện tất cả các cột sử dụng các dạng dữ liệu được vai trò được kích hoạt hiện hành làm chủ. Lưu ý rằng trong PostgreSQL, các dạng dữ liệu được xây dựng sẵn

hành xử giống như các dạng do người sử dụng định nghĩa, nên chúng cũng sẽ được đưa vào ở đây. Xem thêm Phần 34.12 để có các chi tiết.

Bảng 34-9. Các cột column_udt_usage

Tên	Dạng dữ liệu	Mô tả
udt_catalog	sql_identifier	Tên cơ sở dữ liệu mà dạng dữ liệu cột (dạng nằm bên dưới miền đó, nếu áp dụng được) được xác định trong đó (luôn là cơ sở dữ liệu hiện hành)
udt_schema	sql_identifier	Tên của sơ đồ mà dạng dữ liệu cột (dạng nằm bên dưới miền đó, nếu áp dụng được) được xác định trong đó
udt_name	sql_identifier	Tên dạng dữ liệu cột (dạng miền nằm bên dưới, nếu áp dụng được)
table_catalog	sql_identifier	Tên cơ sở dữ liệu chứa bảng đó (luôn là bảng hiện hành)
table_schema	sql_identifier	Tên sơ đồ chứa bảng đó
table_name	sql_identifier	Tên bảng đó
column_name	sql_identifier	Tên cột đó

34.12. columns

Kiểu nhìn columns chứa thông tin về tất cả các cột của bảng (hoặc các cột kiểu nhìn) trong cơ sở dữ liệu đó. Các cột hệ thống (oid, …) sẽ được đưa vào. Chỉ các cột đó sẽ được chỉ ra mà người sử dụng hiện hành có sự truy cập tới (bằng cách là người chủ sở hữu hoặc có vài quyền ưu tiên).

Bảng 34-10. Các cột columns

Tên	Dạng dữ liệu	Mô tả
table_schema	sql_identifier	Tên sơ đồ chứa bảng đó
table_name	sql_identifier	Tên bảng đó
column_name	sql_identifier	Tên cột đó
ordinal_position	cardinal_number	Vị trí thứ tự cột trong bảng (tính bắt đầu từ 1)
column_default	character_data	Biểu hiện mặc định của cột
is_nullable	yes_or_no	YES nếu cột có khả năng null, NO nếu được biết không có khả năng null. Một ràng buộc khác null là cách một cột có thể được biết không có khả năng null, nhưng có thể là khác.
data_type	character_data	Dạng dữ liệu của cột, nếu là một dạng được xây dựng sẵn, hoặc ARRAY nếu nó là vài mảng (trong trường hợp đó, xem kiểu nhìn element_types), nếu không sẽ là USER-DEFINED (trong trường hợp đó, dạng được nhận diện trong udt_name và các cột liên quan). Nếu cột đó dựa vào một miền, thì cột này tham chiếu tới dạng nằm bên dưới miền đó (và miền đó được nhận diện trong tên miền và các cột có liên quan).
character_maximum_length	cardinal_number	Nếu data_type nhận diện một ký tự hoặc dạng chuỗi bit, độ dài tối đa được khai báo; null cho tất cả các dạng dữ liệu hoặc nếu không độ dài tối đa nào từng được khai báo.
character_octet_length	cardinal_number	Nếu data_type nhận diện một dạng ký tự, thì độ dài tối đa có thể theo octets (byte) của một dữ liệu; null cho tất cả các dạng dữ liệu khác. Độ dài octet tối đa phụ thuộc vào độ dài tối đa ký tự được khai báo (xem ở trên) và việc mã hóa máy chủ.

Tên	Dạng dữ liệu	Mô tả
numeric_precision	cardinal_number	Nếu data_type nhận diện một dạng số, cột này chứa (được khai báo hoặc ngầm hiểu) độ chính xác của dạng đó cho cột này. Độ chính xác chỉ ra số các chữ số có nghĩa. Nó có thể được thể hiện theo khoản thập phân (cơ sở 10) hoặc nhị phân (cơ sở 2), như được chỉ định trong cột numeric_precision_radix. Đối với tất cả các dạng dữ liệu khác, cột này là null.
numeric_precision_radix	cardinal_number	Nếu data_type nhận diện một dạng số, cột này chỉ ra theo cơ sở nào các giá trị trong các cột numeric_precision và numeric_scale được thể hiện. Giá trị đó hoặc là 2 hoặc là 10. Đối với tất cả các dạng dữ liệu khác, cột này là null.
numeric_scale	cardinal_number	Nếu data_type nhận diện một dạng số chính xác, cột này chứa (được khai báo hoặc ngầm hiểu) phạm vi của dạng cho cột này. Phạm vi đó chỉ ra số các chữ số có nghĩa về bên phải của dấu thập phân. Nó có thể được thể hiện theo dạng thập phân (cơ sở 10) hoặc nhị phân (cơ sở 2), như được chỉ định trong cột numeric_precision_radix. Đối với tất cả các dạng dữ liệu khác, cột này là null.
datetime_precision	cardinal_number	Nếu data_type nhận diện một dạng ngày tháng, thời gian, dấu thời gian hoặc khoảng thời gian, thì cột này chứa (được khai báo hoặc ngầm hiểu) độ chính xác phần giây của dạng đó cho cột này, đó là, số các chữ số thập phân được duy trì sau dấu thập phân theo giá trị giây. Đối với tất cả các dạng dữ liệu khác, cột này là null.
interval_type	character_data	Còn chưa được triển khai
interval_precision	character_data	Còn chưa được triển khai
character_set_catalog	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
character_set_schema	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
character_set_name	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
collation_catalog	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
collation_schema	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
collation_name	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
domain_catalog	sql_identifier	Nếu cột có dạng miền, thì tên của cơ sở dữ liệu mà miền đó được xác định trong đó (luôn là cơ sở dữ liệu hiện hành), nếu khác là null.
domain_schema	sql_identifier	Nếu cột có dạng miền, thì tên của sơ đồ mà miền đó được xác định trong đó, nếu khác là null.
domain_name	sql_identifier	Nếu cột có dạng miền, thì tên của miền đó, nếu khác là null.
udt_catalog	sql_identifier	Tên của cơ sở dữ liệu mà dạng dữ liệu cột (dạng nằm bên dưới miền đó, nếu áp dụng được) được xác định trong đó (luôn là cơ sở dữ liệu hiện hành)
udt_schema	sql_identifier	Tên của sơ đồ mà dạng dữ liệu cột đó (dạng miền nằm bên dưới, nếu áp dụng được) được xác định trong đó
udt_name	sql_identifier	Tên dạng dữ liệu cột (dạng miền nằm bên dưới, nếu áp dụng được)
scope_catalog	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
scope_schema	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
scope_name	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
maximum_cardinality	cardinal_number	Luôn là null, vì các mảng luôn có bản số tối đa không giới hạn trong PostgreSQL

Tên	Dạng dữ liệu	Mô tả
dtd_identifier	sql_identifier	Một mã định danh của trình mô tả dạng dữ liệu của cột, độc nhất trong số các trình mô tả dạng dữ liệu có liên quan tới bảng đó. Điều này chủ yếu hữu dụng cho việc liên kết với các trường hợp khác của mã nhận diện như vậy. (Định dạng đặc thù của mã định danh không được xác định và đảm bảo vẫn là y hệt trong các phiên bản sau).
is_self_referencing	yes_or_no	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
is_identity	yes_or_no	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
identity_generation	character_data	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
identity_start	character_data	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
identity_increment	character_data	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
identity_maximum	character_data	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
identity_minimum	character_data	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
identity_cycle	yes_or_no	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
is_generated	character_data	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
generation_expression	character_data	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
is_updatable	yes_or_no	YES nếu cột đó không cập nhật được. NO nếu không (Các cột trong các bảng cơ sở luôn có khả năng cập nhật được, các cột trong các kiểu nhìn thì không nhất thiết)

Vì các dạng dữ liệu có thể được xác định theo một loạt các cách thức trong SQL, và PostgreSQL có các cách thức bổ sung để xác định các dạng dữ liệu, sự đại diện của chúng trong sơ đồ thông tin có thể hơi khó khăn. Cột data_type được hỗ trợ để nhận diện dạng cột được xây dựng bên dưới. Trong PostgreSQL, điều này có nghĩa là dạng đó được xác định trong sơ đồ catalog hệ thống pg_catalog.

Cột này có thể là hữu dụng nếu ứng dụng có thể điều khiển các dạng nổi tiếng được xây dựng sẵn một cách đặc biệt (ví dụ, định dạng các dạng số khác hoặc sử dụng dữ liệu trong các cột chính xác). Các cột udt_name, udt_schema, và udt_catalog luôn nhận diện dạng dữ liệu cột nằm bên dưới, thậm chí nếu cột đó được dựa vào một miền. (Vì PostgreSQL đối xử với các dạng được xây dựng sẵn như là các dạng được người sử dụng định nghĩa, các dạng được xây dựng sẵn cũng xuất hiện ở đây. Đây là một mở rộng của tiêu chuẩn SQL). Các cột đó sẽ được sử dụng nếu một ứng dụng muốn xử lý dữ liệu khác tuân theo dạng đó, vì trong trường hợp đó nó không phải là vấn đề nếu cột đó thực sự được dựa vào một miền. Nếu cột đó được dựa vào một miền, thì sự nhận diện của miền đó được lưu trữ trong các cột domain_name, domain_schema, và domain_catalog. Nếu bạn muốn ghép đôi các cột với các dạng dữ liệu có liên quan của chúng và đối xử với các miền như các dạng riêng biệt, thì bạn có thể viết coalesce(domain_name, udt_name), …

34.13. constraint_column_usage

Kiểu nhìn constraint_column_usage nhận diện tất cả các cột trong cơ sở dữ liệu hiện hành mà được vài ràng buộc sử dụng. Chỉ các cột đó được chỉ ra sẽ được đặt trong một bảng do một vai trò hiện được kích hoạt sở hữu. Đối với một ràng buộc kiểm tra, kiểu nhìn này nhận diện các cột được sử dụng trong biểu hiện kiểm tra đó. Đối với một ràng buộc khóa ngoại, kiểu nhìn này nhận diện các

cột mà khóa ngoại tham chiếu. Đối với một ràng buộc khóa duy nhất hoặc chính, thì kiểu nhìn này nhận diện các cột có ràng buộc.

Bảng 34-11. Các cột constraint_column_usage

Tên	Dạng dữ liệu	Mô tả
table_catalog	sql_identifier	Tên cơ sở dữ liệu chứa bảng mà có cột được vài ràng buộc sử dụng (luôn là cơ sở dữ liệu hiện hành)
table_schema	sql_identifier	Tên sơ đồ chứa bảng có cột được vài ràng buộc sử dụng
table_name	sql_identifier	Tên bảng chứa cột được vài ràng buộc sử dụng
column_name	sql_identifier	Tên cột được vài ràng buộc sử dụng
constraint_catalog	sql_identifier	Tên cơ sở dữ liệu chứa ràng buộc (luôn là cơ sở dữ liệu hiện hành)
constraint_schema	sql_identifier	Tên sở đồ chứa ràng buộc đó
constraint_name	sql_identifier	Tên ràng buộc đó

34.14. constraint_table_usage

Kiểu nhìn constraint_table_usage nhận diện tất cả các bảng trong cơ sở dữ liệu hiện hành được vài ràng buộc sử dụng và được một vai trò hiện được kích hoạt làm chủ. (Điều này là khác với kiểu nhìn table_constraints, nó nhận diện tất cả các ràng buộc bảng cùng với bảng mà chúng được xác định trong đó). Đối với một ràng buộc khóa ngoại, kiểu nhìn này nhận diện bảng mà khóa ngoại đó tham chiếu. Đối với một ràng buộc duy nhất hoặc khóa chính, kiểu nhìn này đơn giản nhận diện bảng mà ràng buộc đó thuộc về. Các ràng buộc kiểm tra và các ràng buộc khác null sẽ không được đưa vào trong kiểu nhìn này.

Bảng 34-12. Các cột constraint_table_usage

Tên	Dạng dữ liệu	Mô tả
table_catalog	sql_identifier	Tên cơ sở dữ liệu chứa bảng được vài ràng buộc sử dụng (luôn là cơ sở dữ liệu hiện hành)
table_schema	sql_identifier	Tên sơ đồ chứa bảng được vài ràng buộc sử dụng
table_name	sql_identifier	Tên bảng được vài ràng buộc sử dụng
constraint_catalog	sql_identifier	Tên cơ sở dữ liệu chứa ràng buộc đó (luôn là cơ sở dữ liệu hiện hành)
constraint_schema	sql_identifier	Tên sơ đồ chứa ràng buộc đó
constraint_name	sql_identifier	Tên ràng buộc đó

34.15. data_type_privileges

Kiểu nhìn data_type_privileges nhận diện tất cả các trình mô tả dạng dữ liệu mà người sử dụng hiện hành có sự truy cập tới, bằng cách là chủ sở hữu của đối tượng được mô tả hoặc có vài quyền ưu tiên cho nó. Một trình mô tả dạng dữ liệu được sinh ra bất kỳ khi nào một dạng dữ liệu được sử dụng trong định nghĩa cột của một bảng, một miền, hoặc một hàm (như tham số hoặc dạng trả về) và lưu trữ vài thông tin về cách dạng dữ liệu đó được sử dụng trong trường hợp đó (ví dụ, độ dài tối đa được khai báo, nếu áp dụng được). Từng trình mô tả dạng dữ liệu được chỉ định một mã định

danh tùy ý là duy nhất trong các mã định danh của trình mô tả dạng dữ liệu được chỉ định cho một đối tượng (bảng, miền, hàm). Kiểu nhìn này có thể là không hữu dụng cho các ứng dụng, nhưng nó được sử dụng để xác định vài kiểu nhìn khác trong sơ đồ thông tin.

Bảng 34-13. Các cột data_type_privileges

Tên	Dạng dữ liệu	Mô tả
object_catalog	sql_identifier	Tên cơ sở dữ liệu chứa đối tượng được mô tả (luôn là cơ sở dữ liệu hiện hành)
object_schema	sql_identifier	Tên sơ đồ chứa đối tượng được mô tả
object_name	sql_identifier	Tên đối tượng được mô tả
object_type	character_data	Dạng đối tượng được mô tả: một của TABLE (trình mô tả dạng dữ liệu gắn với một cột của bảng đó), DOMAIN (các trình mô tả dạng dữ liệu gắn liền với miền đó), ROUTINE (trình mô tả dạng dữ liệu gắn với một tham số hoặc trả về dạng dữ liệu của hàm đó).
dtd_identifier	sql_identifier	Mã định danh của trình mô tả dạng dữ liệu, nó là duy nhất trong các trình mô tả dạng dữ liệu đối với đối tượng y hệt đó.

34.16. domain_constraints

Kiểu nhìn domain_constraints chứa tất cả các ràng buộc thuộc về các miền được xác định trong cơ sở dữ liệu hiện hành.

Bảng 34-14. Các cột domain_constraints

Tên	Dạng dữ liệu	Mô tả
constraint_catalog	sql_identifier	Tên cơ sở dữ liệu chứa ràng buộc đó (luôn là cơ sở dữ liệu hiện hành)
constraint_schema	sql_identifier	Tên sơ đồ chứa ràng buộc đó
constraint_name	sql_identifier	Tên ràng buộc đó
domain_catalog	sql_identifier	Tên cơ sở dữ liệu chứa miền đó (luôn là cơ sở dữ liệu hiện hành)
domain_schema	sql_identifier	Tên sơ đồ chứa miền đó
domain_name	sql_identifier	Tên miền đó
is_deferrable	yes_or_no	YES nếu ràng buộc đó là có thể hoãn lại, NO nếu không
initially_deferred	yes_or_no	YES nếu ràng buộc đó có thể hoãn lại và bắt đầu được hoãn lại, NO nếu không

34.17. domain_udt_usage

Kiểu nhìn domain_udt_usage nhận diện tất cả các miền dựa vào các dạng dữ liệu được một vai trò hiện đang được kích hoạt sở hữu. Lưu ý là trong PostgreSQL, các dạng dữ liệu được xây dựng sẵn hành xử giống như các dạng do người sử dụng định nghĩa, vì thế chúng cũng được đưa vào ở đây.

Bảng 34-15. Các cột domain_udt_usage

Tên	Dạng dữ liệu	Mô tả
udt_catalog	sql_identifier	Tên cơ sở dữ liệu mà dạng dữ liệu miền được xác định trong đó (luôn là cơ sở dữ liệu hiện hành)
udt_schema	sql_identifier	Tên sơ đồ mà dạng dữ liệu miền được xác định trong đó

Tên	Dạng dữ liệu	Mô tả
udt_name	sql_identifier	Tên dạng dữ liệu miền
domain_catalog	sql_identifier	Tên cơ sở dữ liệu chứa miền đó (luôn là cơ sở dữ liệu hiện hành)
domain_schema	sql_identifier	Tên sơ đồ chứa miền đó
domain_name	sql_identifier	Tên miền đó

34.18. domains

Kiểu nhìn domains chứa tất cả các miền được xác định trong cơ sở dữ liệu hiện hành.

Bảng 34-16. Các cột domains

Tên	Dạng dữ liệu	Mô tả
domain_catalog	sql_identifier	Tên cơ sở dữ liệu chứa miền đó (luôn là cơ sở dữ liệu hiện hành)
domain_schema	sql_identifier	Tên sơ đồ chứa miền đó
domain_name	sql_identifier	Tên miền đó
data_type	character_data	Dạng dữ liệu của miền, nếu đó là dạng được xây dựng sẵn, hoặc ARRAY nếu nó là vài mảng (trong trường hợp đó, xem kiểu nhìn element_types), nếu không là USER-DEFINED (trong trường hợp đó, dạng được nhận diện trong udt_name và các cột liên quan).
character_maximum_length	cardinal_number	Nếu miền đó có một dạng ký tự hoặc chuỗi bit, thì độ dài tối đa được khai báo; null cho tất cả các dạng dữ liệu khác hoặc nếu không độ dài tối đa nào từng được khai báo.
character_octet_length	cardinal_number	Nếu miền đó có một dạng ký tự, độ dài tối đa có thể trong octets (byte) của một dữ liệu; null cho tất cả các dạng dữ liệu khác. Độ dài octet cực đại phụ thuộc vào độ dài tối đa ký tự được khai báo (xem ở trên) và mã hóa máy chủ.
character_set_catalog	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
character_set_schema	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
character_set_name	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
collation_catalog	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
collation_schema	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
collation_name	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
numeric_precision	cardinal_number	Nếu miền đó có một dạng số, cột này chứa (được khai báo hoặc ngầm hiểu) độ chính xác của dạng cho miền này. Độ chính xác chỉ số các chữ số có nghĩa. Nó có thể được thể hiện theo khoản thập phân (cơ sở 10) hoặc nhị phân (cơ sở 2), như được chỉ định trong cột numeric_precision_radix. Đối với tất cả các dạng khác, cột này là null.
numeric_precision_radix	cardinal_number	Nếu miền đó có một dạng số, cột này chỉ trong cơ sở nào các giá trị trong các cột numeric_precision và numeric_scale được thể hiện. Giá trị hoặc bằng 2 hoặc 10. Đối với tất cả các dạng khác, cột này là null.
numeric_scale	cardinal_number	Nếu miền đó có một dạng số chính xác, cột này chứa (được khai báo hoặc ngầm hiểu) phạm vi của dạng cho miền này. Phạm vi chỉ số các chữ số có nghĩa ở bên phải của dấu thập phân. Nó có thể được thể hiện theo khoản thập phân (cơ sở 10) hoặc nhị phân (cơ sở 2), như được chỉ định trong cột numeric_precision_radix. Đối với tất cả các dạng dữ liệu khác, cột này là null.

Tên	Dạng dữ liệu	Mô tả
datetime_precision	cardinal_number	Nếu data_type nhận diện một dạng ngày tháng, thời gian, dấu thời gian, hoặc khoảng thời gian, cột này chứa (được khai báo hoặc ngầm hiểu) độ chính xác phần giây của dạng cho miền này, đó là, số các chữ số thập phân được duy trì sau dấu thập phân trong giá trị giây. Đối với tất cả các dạng khác, cột này là null.
interval_type	character_data	Còn chưa được triển khai
interval_precision	character_data	Còn chưa được triển khai
domain_default	character_data	Thể hiện mặc định của miền đó
udt_catalog	sql_identifier	Tên cơ sở dữ liệu mà dạng dữ liệu miền được xác định trong đó (luôn là cơ sở dữ liệu hiện hành)
udt_schema	sql_identifier	Tên sơ đồ mà dạng dữ liệu miền đó được xác định trong đó
udt_name	sql_identifier	Tên dạng dữ liệu miền đó
scope_catalog	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
scope_schema	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
scope_name	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
maximum_cardinality	cardinal_number	Luôn null, vì các mảng luôn có bản số tối đa có giới hạn trong PostgreSQL
dtd_identifier	sql_identifier	Một mã định danh của trình mô tả dạng dữ liệu của miền đó, duy nhất trong các trình mô tả dạng dữ liệu gắn với miền đó (nó là tầm thường, vì một miền chỉ chứa một trình mô tả dạng dữ liệu). Điều này chủ yếu hữu dụng cho việc liên kết với các trường hợp khác như các mã định danh như vậy. (Định dạng đặc thù của mã định danh không được xác định và không được đảm bảo vẫn giữ y hệt trong các phiên bản sau).

34.19. element_types

Kiểu nhìn element_types chứa các trình mô tả dạng dữ liệu của các phần tử mảng. Khi một cột của bảng, miền tham số hàm, hoặc hàm trả về các giá trị được xác định sẽ là một dạng mảng, thì kiểu nhìn sơ đồ thông tin tương ứng chỉ chứa ARRAY trong cột data_type. Để có thông tin về dạng phần tử mảng, bạn có thể ghép kiểu nhìn tương ứng với kiểu nhìn này. Ví dụ, để chỉ ra các cột của một bảng với các dạng dữ liệu và các dạng phần tử mảng, nếu áp dụng được, bạn có thể làm:

```
SELECT c.column_name, c.data_type, e.data_type AS element_type
FROM information_schema.columns c LEFT JOIN information_schema.element_types e
        ON ((c.table_catalog, c.table_schema, c.table_name, 'TABLE', c.dtd_identifier)
        = (e.object_catalog, e.object_schema, e.object_name, e.object_type, e.collection_type_
WHERE c.table_schema = '...' AND c.table_name = '...'
ORDER BY c.ordinal_position;
```

Kiểu nhìn này chỉ bao gồm các đối tượng mà người sử dụng hiện hành có sự truy cập tới, bằng cách là một chủ sở hữu hoặc có vài quyền ưu tiên.

Bảng 34-17. Các cột element_types

Tên	Dạng dữ liệu	Mô tả
object_catalog	sql_identifier	Tên cơ sở dữ liệu chứa đối tượng sử dụng mảng đang được mô tả (luôn là cơ sở dữ liệu hiện hành)
object_schema	sql_identifier	Tên sơ đồ chứa đối tượng sử dụng mảng đang được mô tả
object_name	sql_identifier	Tên đối tượng sử dụng mảng đang được mô tả
object_type	character_data	Dạng đối tường sử dụng mảng đang được mô tả: một trong TABLE (mảng được một cột của bảng đó sử dụng), DOMAIN (mảng được miền đó sử dụng), ROUTINE (mảng được một tham số hoặc dạng dữ liệu trả về của hàm đó sử dụng).
collection_type_identifier	sql_identifier	Mã định danh trình mô tả dạng dữ liệu của mảng đang được mô tả. Sử dụng điều này để gắn với các cột dtd_identifier của các kiểu nhìn sơ đồ thông tin khác.
data_type	character_data	Dạng dữ liệu các phần tử mảng, nếu nó là một dạng được xây dựng sẵn, nếu không USER-DEFINED (trong trường hợp đó, dạng được nhận diện trong udt_name và các cột có liên quan).
character_maximum_length	cardinal_number	Luôn null, vì thông tin này không áp dụng được cho các dạng dữ liệu phần tử mảng trong PostgreSQL
character_octet_length	cardinal_number	Luôn null, vì thông tin này không áp dụng được cho các dạng dữ liệu phần tử mảng trong PostgreSQL
character_set_catalog	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
character_set_schema	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
character_set_name	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
collation_catalog	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
collation_schema	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
collation_name	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
numeric_precision	cardinal_number	Luôn null, vì thông tin này không được áp dụng cho các dạng dữ liệu phần tử mảng trong PostgreSQL
numeric_precision_radix	cardinal_number	Luôn null, vì thông tin này không được áp dụng cho các dạng dữ liệu phần tử mảng trong PostgreSQL
numeric_scale	cardinal_number	Luôn null, vì thông tin này không được áp dụng cho các dạng dữ liệu phần tử mảng trong PostgreSQL
datetime_precision	cardinal_number	Luôn null, vì thông tin này không được áp dụng cho các dạng dữ liệu phần tử mảng trong PostgreSQL
interval_type	character_data	Luôn null, vì thông tin này không được áp dụng cho các dạng dữ liệu phần tử mảng trong PostgreSQL
interval_precision	character_data	Luôn null, vì thông tin này không được áp dụng cho các dạng dữ liệu phần tử mảng trong PostgreSQL
domain_default	character_data	Còn chưa được triển khai
udt_catalog	sql_identifier	Tên cơ sở dữ liệu mà dạng dữ liệu của các phần tử được xác định trong đó (luôn là cơ sở dữ liệu hiện hành)
udt_schema	sql_identifier	Tên sơ đồ mà dạng dữ liệu các phần tử được xác định trong đó
udt_name	sql_identifier	Tên dạng dữ liệu các phần tử
scope_catalog	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL

Tên	Dạng dữ liệu	Mô tả
scope_schema	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
scope_name	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
maximum_cardinality	cardinal_number	Luôn null, vì các mảng luôn có bản số tối đa không giới hạn trong PostgreSQL
dtd_identifier	sql_identifier	Một mã định danh của trình mô tả dạng dữ liệu của phần tử. Điều này là không hữu dụng hiện hành.

34.20. enabled_roles

Kiểu nhìn enabled_roles nhận diện "các vai trò được kích hoạt" hiện hành. Các vai trò được kích hoạt được xác định đệ quy như người sử dụng hiện hành cùng với tất cả các vai trò mà đã được trao cho các vai trò được kích hoạt với sự kế thừa tự động. Nói cách khác, chúng là tất cả các vai trò mà người sử dụng hiện hành có trực tiếp hoặc gián tiếp, tự động kế thừa cơ chế thành viên trong đó.

Để kiểm tra quyền, tập hợp "các vai trò áp dụng được" được áp dụng, nó có thể rộng lớn hơn so với các vai trò được kích hoạt. Vì thế thường là, là tốt hơn để sử dụng kiểu nhìn applicable_roles thay vì kiểu này; xem thêm ở đó.

Bảng 34-18. Các cột enabled_roles

Tên	Dạng dữ liệu	Mô tả
role_name	sql_identifier	Tên vai trò

34.21. foreign_data_wrapper_options

Kiểu nhìn foreign_data_wrapper_options chứa tất cả các lựa chọn được xác định cho các trình bọc dữ liệu ngoài trong cơ sở dữ liệu hiện hành. Chỉ các trình bọc dữ liệu ngoài nào được chỉ ra mà người sử dụng hiện hành có sự truy cập tới (bằng cách là chủ sở hữu hoặc có vài quyền ưu tiên).

Bảng 34-19. Các cột foreign_data_wrapper_options

Tên	Dạng dữ liệu	Mô tả
foreign_data_wrapper_catas	lqolg_identifier	Tên cơ sở dữ liệu mà trình bọc dữ liệu ngoài được xác định trong đó (luôn là cơ sở dữ liệu hiện hành)
foreign_data_wrapper_nam	esql_identifier	Tên trình bọc dữ liệu ngoài
option_name	sql_identifier	Tên một lựa chọn
option_value	character_data	Giá trị lựa chọn đó

34.22. foreign_data_wrappers

Kiểu nhìn foreign_data_wrappers chứa tất cả các trình bọc dữ liệu ngoài được xác định trong cơ sở dữ liệu hiện hành. Chỉ những trình bọc dữ liệu ngoài nào được chỉ ra mà người sử dụng hiện hành có sự truy cập tới (bằng cách là chủ sở hữu hoặc có vài quyền ưu tiên).

Bảng 34-20. Các cột foreign_data_wrappers

Tên	Dạng dữ liệu	Mô tả
foreign_data_wrapper_catalog	sql_identifier	Tên cơ sở dữ liệu chứa trình bọc dữ liệu ngoài (luôn là cơ sở dữ liệu hiện hành)
foreign_data_wrapper_name	sql_identifier	Tên trình bọc dữ liệu ngoài
authorization_identifier	sql_identifier	Tên chủ sở hữu máy chủ ngoài
library_name	character_data	Tên tệp thư viện mà đang triển khai trình bọc dữ liệu ngoài này
foreign_data_wrapper_language	character_data	Ngôn ngữ được sử dụng để triển khai trình bọc dữ liệu ngoài này

34.23. foreign_server_options

Kiểu nhìn foreign_server_options chứa tất cả các lựa chọn được xác định cho các máy chủ ngoài trong cơ sở dữ liệu hiện hành. chỉ những máy chủ ngoài nào được chỉ ra mà người sử dụng hiện hành có sự truy cập tới (bằng cách là chủ sở hữu hoặc có vài quyền ưu tiên).

Bảng 34-21. Các cột foreign_server_options

Tên	Dạng dữ liệu	Mô tả
foreign_server_catalog	sql_identifier	Tên cơ sở dữ liệu mà máy chủ ngoài được xác định trong đó (luôn là cơ sở dữ liệu hiện hành)
foreign_server_name	sql_identifier	Tên máy chủ ngoài
option_name	sql_identifier	Tên một lựa chọn
option_value	character_data	Giá trị lựa chọn đó

34.24. foreign_server

Kiểu nhìn foreign_servers chứa tất cả các máy chủ ngoài được xác định trong cơ sở dữ liệu hiện hành. Chỉ những máy chủ ngoài nào được chỉ ra mà người sử dụng hiện hành có sự truy cập tới (bằng cách là chủ sở hữu hoặc có vài quyền ưu tiên).

Bảng 34-22. Các cột foreign_server

Tên	Dạng dữ liệu	Mô tả
foreign_server_catalog	sql_identifier	Tên cơ sở dữ liệu mà máy chủ ngoài được xác định trong đó (luôn là cơ sở dữ liệu hiện hành)
foreign_server_name	sql_identifier	Tên máy chủ ngoài
foreign_data_wrapper_catalog	sql_identifier	Tên cơ sở dữ liệu mà chứa trình bọc dữ liệu ngoài được máy chủ ngoài sử dụng (luôn là cơ sở dữ liệu hiện hành)
foreign_data_wrapper_name	sql_identifier	Tên trình bọc dữ liệu ngoài được máy chủ ngoài sử dụng
foreign_server_type	character_data	Thông tin dạng máy chủ ngoài, nếu được chỉ định khi tạo
foreign_server_version	character_data	Thông tin phiên bản máy chủ ngoài, nếu được chỉ định khi tạo
authorization_identifier	sql_identifier	Tên chủ sở hữu máy chủ ngoài

34.25. key_column_usage

Kiểu nhìn key_column_usage nhận diện tất cả các cột trong cơ sở dữ liệu hiện hành được giới hạn bởi vài khóa duy nhất, chính, hoặc ràng buộc khóa ngoài. Các ràng buộc kiểm tra không được đưa vào trong kiểu nhìn này. Chỉ những cột nào được chỉ ra mà người sử dụng hiện hành có sự truy cập tới, bằng cách là chủ sở hữu hoặc có vài quyền ưu tiên.

Bảng 34-23. Các cột key_column_usage

Tên	Dạng dữ liệu	Mô tả
constraint_catalog	sql_identifier	Tên cơ sở dữ liệu chứa ràng buộc đó (luôn là cơ sở dữ liệu hiện hành)
constraint_schema	sql_identifier	Tên sơ đồ chứa ràng buộc đó
constraint_name	sql_identifier	Tên ràng buộc đó
table_catalog	sql_identifier	Tên cơ sở dữ liệu chứa bảng có cột bị ràng buộc này giới hạn (luôn là cơ sở dữ liệu hiện hành)
table_schema	sql_identifier	Tên sơ đồ chứa bảng có cột bị ràng buộc này hạn chế
table_name	sql_identifier	Tên bảng chứa cột bị ràng buộc này hạn chế
column_name	sql_identifier	Tên cột bị ràng buộc này hạn chế
ordinal_position	cardinal_number	Vị trí theo trật tự cột trong khóa ràng buộc (tính bắt đầu từ 1)
position_in_unique_constraint	cardinal_number	Đối với ràng buộc khóa ngoài, vị trí trật tự của cột được tham chiếu trong ràng buộc độc nhất của nó (tính bắt đầu từ 1); nếu khác là null

34.26. parameters

Kiểu nhìn parameters chứa thông tin về các tham số (các đối số) của tất cả các hàm trong cơ sở dữ liệu hiện hành. Chỉ các hàm nào được chỉ ra mà người sử dụng hiện hành có sự truy cập tới (bằng cách là người chủ sở hữu hoặc có vài quyền ưu tiên).

Bảng 34-24. Các cột parameters

Tên	Dạng dữ liệu	Mô tả
specific_catalog	sql_identifier	Tên cơ sở dữ liệu chứa hàm đó (luôn là cơ sở dữ liệu hiện hành)
specific_schema	sql_identifier	Tên sơ đồ chứa hàm đó
specific_name	sql_identifier	"Tên đặc biệt" hàm đó. Xem Phần 34.33 để có thêm thông tin.
ordinal_position	cardinal_number	Vị trí thứ tự của tham số trong danh sách đối số hàm đó (tính từ 1)
parameter_mode	character_data	IN cho tham số đầu vào, OUT cho tham số đầu ra, và INOUT cho tham số đầu vào/đầu ra.
is_result	yes_or_no	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
as_locator	yes_or_no	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
parameter_name	sql_identifier	Tên tham số, hoặc null nếu tham số đó không có tên
data_type	character_data	Dạng dữ liệu của tham số, nếu nó là dạng được xây dựng sẵn, hoặc ARRAY nếu đó là vài mảng (trong trường hợp đó, xem kiểu nhìn element_types), còn không thì USER-DEFINED (trong trường hợp đó, dạng được xác định trong udt_name và các cột có liên quan).
character_maximum_l	cardinal_number	Luôn null, vì thông tin này không áp dụng được cho các dạng dữ liệu tham

Tên	Dạng dữ liệu	Mô tả
engths		số trong PostgreSQL
character_octet_length	cardinal_number	Luôn null, vì thông tin này không được áp dụng cho các dạng dữ liệu tham số trong PostgreSQL
character_set_catalog	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
character_set_schema	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
character_set_name	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
collation_catalog	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
collation_schema	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
collation_name	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
numeric_precision	cardinal_number	Luôn null, vì thông tin này không áp dụng được cho các dạng dữ liệu tham số trong PostgreSQL
numeric_precision_radix	cardinal_number	Luôn null, vì thông tin này không áp dụng được cho các dạng dữ liệu tham số trong PostgreSQL
numeric_scale	cardinal_number	Luôn null, vì thông tin này không áp dụng được cho các dạng dữ liệu tham số trong PostgreSQL
datetime_precision	cardinal_number	Luôn null, vì thông tin này không áp dụng được cho các dạng dữ liệu tham số trong PostgreSQL
interval_type	character_data	Luôn null, vì thông tin này không áp dụng được cho các dạng dữ liệu tham số trong PostgreSQL
interval_precision	character_data	Luôn null, vì thông tin này không áp dụng được cho các dạng dữ liệu tham số trong PostgreSQL
udt_catalog	sql_identifier	Tên cơ sở dữ liệu mà dạng dữ liệu của tham số đó được xác định trong đó (luôn là cơ sở dữ liệu hiện hành)
udt_schema	sql_identifier	Tên sơ đồ mà dạng dữ liệu của tham số được xác định trong đó
udt_name	sql_identifier	Tên dạng dữ liệu của tham số đó
scope_catalog	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
scope_schema	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
scope_name	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
maximum_cardinality	cardinal_number	Luôn null, vì các mảng luôn có bản số tối đa có giới hạn trong PostgreSQL
dtd_identifier	sql_identifier	Một mã định danh của trình mô tả dạng dữ liệu của tham số đó, độc nhất trong số các trình mô tả dạng dữ liệu gắn với hàm đó. Điều này chủ yếu là hữu dụng cho việc gắn với các trường hợp khác của các mã định danh như vậy. (Định dạng đặc thù của mã định danh đó không được xác định và không được đảm bảo vẫn là y hệt trong các phiên bản trong tương lai).

34.27. referential_constraints

Kiểu nhìn referential_constraints chứa tất cả các ràng buộc tham chiếu (khóa ngoài) trong cơ sở dữ liệu hiện hành. Chỉ các ràng buộc nào mà được chỉ ra theo đó người sử dụng hiện hành có truy cập ghi tới bảng tham chiếu (bằng cách là chủ sở hữu hoặc có vài quyền ưu tiên khác với SELECT).

Bảng 34-25. Các cột referential_constraints

Tên	Dạng dữ liệu	Mô tả
constraint_catalog	sql_identifier	Tên cơ sở dữ liệu chứa ràng buộc đó (luôn là cơ sở dữ liệu hiện hành)
constraint_schema	sql_identifier	Tên sơ đồ chứa ràng buộc đó
constraint_name	sql_identifier	Tên ràng buộc đó
unique_constraint_catalog	sql_identifier	Tên cơ sở dữ liệu chứa ràng buộc duy nhất hoặc khóa chính mà ràng buộc khóa ngoài tham chiếu (luôn là cơ sở dữ liệu hiện hành)
unique_constraint_schema	sql_identifier	Tên sơ đồ chứa ràng buộc duy nhất hoặc khóa chính mà ràng buộc khóa ngoài tham chiếu
unique_constraint_name	sql_identifier	Tên ràng buộc duy nhất hoặc khóa chính mà ràng buộc khóa ngoài tham chiếu
match_option	character_data	Khớp lựa chọn ràng buộc khóa ngoài: FULL, PARTIAL, hoặc NONE.
update_rule	character_data	Cập nhật qui tắc ràng buộc khóa ngoài: CASCADE, SET NULL, SET DEFAULT, RESTRICT, hoặc NO ACTION.
delete_rule	character_data	Xóa qui tắc ràng buộc khóa ngoài: CASCADE, SET NULL, SET DEFAULT, RESTRICT, hoặc NO ACTION.

34.28. role_column_grants

Kiểu nhìn role_column_grants nhận diện tất cả các quyền ưu tiên được trao trong các cột nơi mà người trao hoặc người nhận là một vai trò hiện hành được kích hoạt. Thông tin nhiều hơn có thể thấy được theo column_privileges. Khác biệt duy nhất giữa kiểu nhìn này và column_privileges là kiểu nhìn này bỏ qua các cột mà đã được làm có khả năng truy cập được tới người sử dụng hiện hành bằng cách trao công khai.

Bảng 34-26. Các cột role_column_grants

Tên	Dạng dữ liệu	Mô tả
grantor	sql_identifier	Tên vai trò đã trao quyền ưu tiên
grantee	sql_identifier	Tên vai trò quyền ưu tiên đã được trao cho
table_catalog	sql_identifier	Tên cơ sở dữ liệu chứa bảng có cột đó (luôn là cơ sở dữ liệu hiện hành)
table_schema	sql_identifier	Tên sơ đồ chứa bảng có cột đó
table_name	sql_identifier	Tên bảng chứa cột đó
column_name	sql_identifier	Tên cột đó
privilege_type	character_data	Dạng quyền ưu tiên: SELECT, INSERT, UPDATE, hoặc REFERENCES
is_grantable	yes_or_no	YES nếu quyền ưu tiên có khả năng được trao, NO nếu không

34.29. role_routine_grants

Kiểu nhìn role_routine_grants nhận diện tất cả các quyền ưu tiên được trao trong các hàm nơi mà người trao hoặc người được trao là một vai trò hiện được kích hoạt. Thông tin xa hơn có thể được thấy theo routine_privileges. Khác biệt có hiệu lực duy nhất giữa kiểu nhìn này và routine_privileges là kiểu nhìn này bỏ qua các hàm đã có khả năng truy cập được tới người sử dụng hiện hành bằng

cách trao công khai.

Bảng 34-27. Các cột role_routine_grants

Tên	Dạng dữ liệu	Mô tả
grantor	sql_identifier	Tên vai trò đã trao quyền ưu tiên đó
grantee	sql_identifier	Tên vai trò đã được trao quyền ưu tiên đó
specific_catalog	sql_identifier	Tên cơ sở dữ liệu chứa hàm đó (luôn là cơ sở dữ liệu hiện hành)
specific_schema	sql_identifier	Tên sơ đồ chứa hàm đó
specific_name	sql_identifier	"Tên đặc biệt" của hàm đó. Xem Phần 34.33 để có thêm thông tin.
routine_catalog	sql_identifier	Tên cơ sở dữ liệu chứa hàm đó (luôn là cơ sở dữ liệu hiện hành)
routine_schema	sql_identifier	Tên sơ đồ chứa hàm đó
routine_name	sql_identifier	Tên hàm đó (có thể bị đúp bản trong trường hợp quá tải)
privilege_type	character_data	Luôn THỰC THI - EXECUTE (dạng quyền ưu tiên duy nhất cho các hàm)
is_grantable	yes_or_no	YES nếu quyền ưu tiên có khả năng trao được, NO nếu không

34.30. role_table_grants

Kiểu nhìn role_table_grants nhận diện tất cả các quyền ưu tiên được trao trong các bảng hoặc kiểu nhìn nơi mà người trao và người nhận là một vai trò hiện hành được kích hoạt. Thông tin nhiều hơn có thể thấy theo table_privileges. Điều khác biệt duy nhất giữa kiểu nhìn này và table_privileges là kiểu nhìn này bỏ qua các bảng đã truy cập tới người sử dụng hiện hành bằng cách trao công khai.

Bảng 34-28. Các cột role_table_grants

Tên	Dạng dữ liệu	Mô tả
privilege_type	character_data	Dạng quyền ưu tiên: SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, hoặc TRIGGER
is_grantable	yes_or_no	YES nếu quyền ưu tiên có khả năng trao được, NO nếu không
with_hierarchy	yes_or_no	Áp dụng cho một tính năng còn chưa có trong PostgreSQL

34.31. role_usage_grants

Kiểu nhìn role_usage_grants nhận diện các quyền ưu tiên USAGE được trao trong các dạng đối tượng khác nhau nơi mà người trao hoặc người nhận là một vai trò hiện hành được kích hoạt. Thông tin xa hơn có thể thấy theo usage_privileges. Khác biệt duy nhất giữa kiểu nhìn này và usage_privileges là kiểu nhìn này bỏ qua các đối tượng từng truy cập được tới người sử dụng hiện hành bằng cách có trao cho công khai.

Bảng 34-29. Các cột role_usage_grants

Tên	Dạng dữ liệu	Mô tả
grantor	sql_identifier	Tên vai trò đã trao quyền ưu tiên đó
grantee	sql_identifier	Tên vai trò đã được trao quyền ưu tiên đó
object_catalog	sql_identifier	Tên cơ sở dữ liệu chứa đối tượng đó (luôn là cơ sở dữ liệu hiện hành)

Tên	Dạng dữ liệu	Mô tả
object_schema	sql_identifier	Tên sơ đồ chứa đối tượng đó, nếu được áp dụng, nếu không là một chuỗi rỗng
object_name	sql_identifier	Tên đối tượng đó
object_type	character_data	DOMAIN hoặc FOREIGN DATA WRAPPER hoặc FOREIGN SERVER
privilege_type	character_data	Luôn là USAGE
is_grantable	yes_or_no	YES nếu quyền ưu tiên có khả năng được trao, NO nếu không

34.32. routine_privileges

Kiểu nhìn routine_privileges nhận diện tất cả các quyền ưu tiên được trao trong các hàm cho một vai trò hiện hành được kích hoạt hoặc bằng một vai trò hiện hành được kích hoạt. Có một hàng cho từng sự kết hợp của hàm, người trao và người nhận.

Bảng 34-30. Các cột routine_privileges

Tên	Dạng dữ liệu	Mô tả
grantor	sql_identifier	Tên vai trò đã trao quyền ưu tiên đó
grantee	sql_identifier	Tên vai trò đã được trao quyền ưu tiên đó
specific_catalog	sql_identifier	Tên cơ sở dữ liệu chứa hàm đó (luôn là cơ sở dữ liệu hiện hành)
specific_schema	sql_identifier	Tên sơ đồ chứa hàm đó
specific_name	sql_identifier	"Tên đặc biệt" của hàm đó. Xem Phần 34.33 để có thêm thông tin.
routine_catalog	sql_identifier	Tên cơ sở dữ liệu chứa hàm đó (luôn là cơ sở dữ liệu hiện hành)
routine_schema	sql_identifier	Tên sơ đồ chứa hàm đó
routine_name	sql_identifier	Tên hàm đó (có thể được đúp bản trong trường hợp quá tải)
privilege_type	character_data	Luôn EXECUTE (dạng quyền ưu tiên duy nhất cho các hàm)
is_grantable	yes_or_no	YES nếu quyền ưu tiên có khả năng được trao, NO nếu không

34.33. routines

Kiểu nhìn routines chứa tất cả các hàm trong cơ sở dữ liệu hiện hành. Chỉ các hàm nào được chỉ ra mà người sử dụng hiện hành có sự truy cập tới (bằng cách là chủ sở hữu hoặc có vài quyền ưu tiên).

Bảng 34-31. Các cột routines

Tên	Dạng dữ liệu	Mô tả
specific_catalog	sql_identifier	Tên cơ sở dữ liệu chứa hàm (luôn là cơ sở dữ liệu hiện hành)
specific_schema	sql_identifier	Tên sơ đồ chứa hàm đó
specific_name	sql_identifier	"Tên đặc thù" của hàm đó. Đây là tên nhận diện độc nhất hàm đó trong sơ đồ, thậm chí nếu tên thực tế của hàm bị quá tải. Định dạng tên đặc thù không được xác định, nó sẽ chỉ được sử dụng để so sánh nó với các trường hợp khác các tên thủ tục đặc thù.
routine_catalog	sql_identifier	Tên cơ sở dữ liệu chứa hàm (luôn là cơ sở dữ liệu hiện hành)
routine_schema	sql_identifier	Tên sơ đồ chứa hàm đó

Tên	Dạng dữ liệu	Mô tả
routine_name	sql_identifier	Tên hàm đó (có thể bị đúp bản trong trường hợp quá tải).
routine_type	character_data	Luôn là FUNCTION(Trong tương lai có thể là các dạng thủ tục khác).
module_catalog	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
module_schema	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
module_name	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
udt_catalog	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
udt_schema	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
udt_name	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
data_type	character_data	Trả về dạng dữ liệu của hàm đó, nếu là một dạng được xây dựng sẵn, hoặc ARRAY nếu nó là vài mảng (trong trường hợp đó, xem kiểu nhìn element_types), nếu không thì là USER-DEFINED (trong trường hợp đó, dạng được nhận diện trong type_udt và các cột có liên quan).
character_maximum_length	cardinal_number	Luôn null, vì thông tin này không được áp dụng để trả về các dạng dữ liệu trong PostgreSQL
character_octet_length	cardinal_number	Luôn null, vì thông tin này không được áp dụng để trả về các dạng dữ liệu trong PostgreSQL
character_set_catalog	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
character_set_schema	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
character_set_name	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
collation_catalog	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
collation_schema	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
collation_name	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
numeric_precision	cardinal_number	Luôn null, vì thông tin này không được áp dụng để trả về các dạng dữ liệu trong PostgreSQL
numeric_precision_radix	cardinal_number	Luôn null, vì thông tin này không được áp dụng để trả về các dạng dữ liệu trong PostgreSQL
numeric_scale	cardinal_number	Luôn null, vì thông tin này không được áp dụng để trả về các dạng dữ liệu trong PostgreSQL
datetime_precision	cardinal_number	Luôn null, vì thông tin này không được áp dụng để trả về các dạng dữ liệu trong PostgreSQL
interval_type	character_data	Luôn null, vì thông tin này không được áp dụng để trả về các dạng dữ liệu trong PostgreSQL
interval_precision	character_data	Luôn null, vì thông tin này không được áp dụng để trả về các dạng dữ liệu trong PostgreSQL
type_udt_catalog	sql_identifier	Tên cơ sở dữ liệu mà dạng dữ liệu trả về của hàm được xác định trong đó (luôn là cơ sở dữ liệu hiện hành)
type_udt_schema	sql_identifier	Tên sơ đồ mà dạng dữ liệu trả về của hàm được xác định ở đó
type_udt_name	sql_identifier	Tên dạng dữ liệu trả về của hàm đó
scope_catalog	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
scope_schema	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL

Tên	Dạng dữ liệu	Mô tả
scope_name	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
maximum_cardinality	cardinal_number	Luôn null, vì các mảng luôn có số bản cực đại không giới hạn trong PostgreSQL
dtd_identifier	sql_identifier	Một mã định danh của trình mô tả dạng dữ liệu của dạng dữ liệu trả về của hàm này, là duy nhất trong các trình mô tả dạng dữ liệu gắn với hàm đó. Điều này chủ yếu hữu dụng cho việc gắn với các trường hợp khác của các mã nhận diện như vậy. (Định dạng đặc thù của mã nhận diện không được xác định và không được đảm bảo để vẫn giữ y hệt trong các phiên bản sau)
routine_body	character_data	Nếu hàm đó là một hàm SQL, thì là SQL, nếu không thì là EXTERNAL.
routine_definition	character_data	Văn bản nguồn của hàm (null nếu hàm không được một vai trò hiện hành kích hoạt sở hữu). (Theo tiêu chuẩn SQL, cột này chỉ áp dụng được nếu routine_body là SQL, nhưng trong PostgreSQL nó sẽ chứa bất kỳ văn bản nguồn nào từng được chỉ định khi hàm đó đã được tạo ra).
external_name	character_data	Nếu hàm này là một hàm C, thì là tên bên ngoài (biểu tượng liên kết) của hàm đó; nếu không là null. (Điều này khắc phục để trở thành giá trị y hệt được chỉ ra trong routine_definition).
external_language	character_data	Ngôn ngữ mà hàm được viết trong đó
parameter_style	character_data	Luôn là GENERAL (Tiêu chuẩn SQL xác định các kiểu tham số khác, chúng không sẵn sàng trong PostgreSQL).
is_deterministic	yes_or_no	Nếu hàm được khai báo bất biến (được gọi là xác định trong tiêu chuẩn SQL), thì YES, nếu khác là NO. (Bạn không thể truy vấn các mức biến động khác có sẵn trong PostgreSQL qua sơ đồ thông tin).
sql_data_access	character_data	Luôn là MODIFIES, nghĩa là hàm có khả năng sửa đổi dữ liệu SQL. Thông tin này không hữu dụng cho PostgreSQL.
is_null_call	yes_or_no	Nếu hàm tự động trả về null nếu bất kỳ đối số nào của nó là null, thì là YES, nếu không là NO.
sql_path	character_data	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
schema_level_routine	yes_or_no	Luôn là YES (ngược lại có thể là một phương pháp của dạng do người sử dụng định nghĩa, nó là một tính năng còn chưa có trong PostgreSQL).
max_dynamic_result_sets	cardinal_number	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
is_user_defined_cast	yes_or_no	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
is_implicitly_invocable	yes_or_no	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
security_type	character_data	Nếu hàm đó chạy với các quyền ưu tiên của người sử dụng hiện hành, thì INVOKER, nếu hàm chạy với các quyền ưu tiên của người sử dụng đã định nghĩa nó, thì là DEFINER.
to_sql_specific_catalog	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
to_sql_specific_schema	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
to_sql_specific_name	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
as_locator	yes_or_no	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
created	time_stamp	Áp dụng cho một tính năng còn chưa có trong PostgreSQL

Tên	Dạng dữ liệu	Mô tả
last_altered	time_stamp	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
new_savepoint_level	yes_or_no	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
is_udt_dependent	yes_or_no	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
result_cast_from_data_type	character_data	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
result_cast_as_locator	yes_or_no	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
result_cast_char_max_length	cardinal_number	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
result_cast_char_octet_length	character_data	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
result_cast_char_set_catalog	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
result_cast_char_set_schema	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
result_cast_char_set_name	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
result_cast_collation_catalog	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
result_cast_collation_schema	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
result_cast_collation_name	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
result_cast_numeric_precision	cardinal_number	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
result_cast_numeric_precisionrdix	cardinal_number	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
result_cast_numeric_scale	cardinal_number	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
result_cast_datetime_precision	character_data	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
result_cast_interval_type	character_data	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
result_cast_interval_precision	character_data	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
result_cast_type_udt_catalog	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
result_cast_type_udt_schema	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
result_cast_type_udt_name	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
result_cast_scope_catalog	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
result_cast_scope_schema	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
result_cast_scope_name	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
result_cast_maximum_cardinality	cardinal_number	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
result_cast_dtd_identifier	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL

34.34. schemata

Kiểu nhìn schemata chứa tất cả các sơ đồ trong cơ sở dữ liệu hiện hành được một vai trò hiện hành được kích hoạt sở hữu.

Bảng 34-32. Các cột schemata

Tên	Dạng dữ liệu	Mô tả
catalog_name	sql_identifier	Tên cơ sở dữ liệu mà sơ đồ được chứa trong đó (luôn là cơ sở dữ liệu hiện hành)
schema_name	sql_identifier	Tên sơ đồ đó
schema_owner	sql_identifier	Tên chủ sở hữu sơ đồ đó

Tên	Dạng dữ liệu	Mô tả
default_character_set_catalog	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
default_character_set_schseqmla_identifier	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
default_character_set_name	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
sql_path	character_data	Áp dụng cho một tính năng còn chưa có trong PostgreSQL

34.35. sequences

Kiểu nhìn sequences chứa tất cả các tuần tự được xác định trong cơ sở dữ liệu hiện hành. Chỉ các tuần tự nào được chỉ ra mà người sử dụng hiện hành có sự truy cập tới (bằng cách là chủ sở hữu hoặc có một vài quyền ưu tiên).

Bảng 34-33. Các cột sequences

Tên	Dạng dữ liệu	Mô tả
sequence_catalog	sql_identifier	Tên cơ sở dữ liệu chứa tuần tự đó (luôn là cơ sở dữ liệu hiện hành)
sequence_schema	sql_identifier	Tên sơ đồ chứa tuần tự đó
sequence_name	sql_identifier	Tên tuần tự đó
data_type	character_data	Dạng dữ liệu của tuần tự đó. Trong PostgreSQL, điều này hiện hành luôn là bigint.
numeric_precision	cardinal_number	Cột này chứa (được khai báo hoặc ngầm hiểu) độ chính xác của dạng dữ liệu tuần tự (xem ở trên). Độ chính xác chỉ ra số các chữ số có nghĩa. Nó có thể được thể hiện ở các khoản thập phân (cơ sở 10) hoặc nhị phân (cơ sở 2), như được chỉ định trong cột numeric_precision_radix.
numeric_precision_radix	cardinal_number	Cột này chỉ trong cơ sở nào các giá trị trong các cột numeric_precision và numeric_scale được thể hiện. Giá trị hoặc là 2 hoặc là 10.
numeric_scale	cardinal_number	Cột này chứa (được khai báo hoặc ngầm hiểu) phạm vi dạng dữ liệu tuần tự đó (xem ở trên). Phạm vi đó chỉ ra số các chữ số có nghĩa ở bên phải của dấu thập phân. Nó có thể được thể hiện trong các khoản thập phân (cơ sở 10) hoặc nhị phân (cơ sở 2), như được chỉ định trong cột numeric_precision_radix.
maximum_value	cardinal_number	Còn chưa được triển khai
minimum_value	cardinal_number	Còn chưa được triển khai
increment	cardinal_number	Còn chưa được triển khai
cycle_option	yes_or_no	Còn chưa được triển khai

34.36. sql_features

Bảng sql_features chứa thông tin về các tính năng chính thức nào được xác định theo tiêu chuẩn SQL được PostgreSQL hỗ trợ. Đây là thông tin y hệt mà được thể hiện trong Phụ lục D.

Bạn cũng có thể thấy vài thông tin nền tảng bổ sung.

Bảng 34-34. Các cột sql_features

Tên	Dạng dữ liệu	Mô tả
feature_id	character_data	Chuỗi của mã định dạng của tính năng đó
feature_name	character_data	Tên mô tả của tính năng đó
sub_feature_id	character_data	Chuỗi của mã định danh của tính năng con, hoặc là chuỗi độ dài zero nếu không là một tính năng con
sub_feature_name	character_data	Tên mô tả của tính năng con, hoặc một chuỗi độ dài zero nếu không là tính năng con
is_supported	yes_or_no	YES nếu tính năng đó được phiên bản hiện hành của PostgreSQL hỗ trợ đầy đủ, NO nếu không
is_verified_by	character_data	Luôn null, vì nhóm phát triển PostgreSQL không thực hiện việc kiểm thử chính thức các bình luận tuân thủ tính năng character_data. Có khả năng một bình luận về tình trạng được hỗ trợ của tính năng đó.

34.37. sql_implementation_info

Bảng sql_implementation_info chứa thông tin về các khía cạnh khác nhau được để lại triển khai được tiêu chuẩn SQL xác định. Thông tin này ban đầu có ý định để sử dụng theo ngữ cảnh giao diện ODBC; những người sử dụng các giao diện khác có thể sẽ thấy thông tin này được sử dụng ít. Vì lý do này, các khoản thông tin triển khai riêng rẽ sẽ không được mô tả ở đây; bạn sẽ thấy chúng trong mô tả của giao diện ODBC.

Bảng 34-35. Các cột sql_implementation_info

Tên	Dạng dữ liệu	Mô tả
implementation_info_id	character_data	Chuỗi mã định danh của khoản thông tin triển khai
implementation_info_name	character_data	Tên mô tả của khoản thông tin triển khai
integer_value	cardinal_number	Giá trị của khoản thông tin triển khai, hoặc null nếu giá trị đó có trong cột character_value
character_value	character_data	Giá trị của khoản thông tin triển khai, hoặc null nếu giá trị đó nằm trong cột integer_value
comments	character_data	Có thể một bình luận gắn với khoản thông tin triển khai đó

34.38. sql_languages

Bảng sql_languages chứa một hàng cho từng ràng buộc ngôn ngữ SQL được PostgreSQL hỗ trợ. PostgreSQL hỗ trợ SQL trực tiếp và SQL nhúng trong C; đó là tất cả điều bạn sẽ học được từ bảng này.

Bảng 34-36. Các cột sql_languages

Tên	Dạng dữ liệu	Mô tả
sql_language_source	character_data	Tên nguồn định nghĩa ngôn ngữ; luôn là ISO 9075, đó là, tiêu chuẩn SQL
sql_language_year	character_data	Năm mà tiêu chuẩn được tham chiếu trong sql_language_source

Tên	Dạng dữ liệu	Mô tả
		từng được phê chuẩn; hiện thời là 2003
sql_language_conformance	character_data	Mức độ tuân thủ tiêu chuẩn đối với ràng buộc ngôn ngữ. Đối với ISO 9075:2003 thì điều này luôn là CORE.
sql_language_integrity	character_data	Luôn là null (Giá trị này phù hợp với một phiên bản trước đó của tiêu chuẩn SQL).
sql_language_implementation	chracter_data	Luôn null
sql_language_binding_style	character_data	Dạng ràng buộc ngôn ngữ, hoặc DIRECT hoặc EMBEDDED
sql_language_programming_language_data	character_data	Ngôn ngữ lập trình, nếu dạng ràng buộc là EMBEDDED, nếu khác là null. PostgreSQL chỉ hỗ trợ ngôn ngữ C.

34.39. sql_packages

Bảng sql_packages chứa thông tin về các gói tính năng nào được xác định trong tiêu chuẩn SQL được PostgreSQL hỗ trợ. Hãy tham chiếu tới Phụ lục D để có thông tin cơ bản về các gói tính năng.

Bảng 34-37. Các cột sql_packages

Tên	Dạng dữ liệu	Mô tả
feature_id	character_data	Chuỗi mã định danh của gói đó
feature_name	character_data	Tên mô tả của gói đó
is_supported	yes_or_no	YES nếu gói đó được phiên bản hiện hành của PostgreSQL hỗ trợ đầy đủ, NO nếu không
is_verified_by	character_data	Luôn là null, vì nhóm phát triển PostgreSQL không thực hiện việc kiểm thử chính thức các bình luận tuân thủ tính năng character_data. Có khả năng một bình luận về tình trạng được hỗ trợ của gói đó.

34.40. sql_parts

Bảng sql_parts chứa thông tin về phần nào trong vài phần tiêu chuẩn SQL được PostgreSQL hỗ trợ.

Bảng 34-38. Các cột sql_parts

Tên	Dạng dữ liệu	Mô tả
feature_id	character_data	Chuỗi mã định danh chứa số của phần đó
feature_name	character_data	Tên mô tả của phần đó
is_supported	yes_or_no	YES nếu phần đó được phiên bản hiện hành của PostgreSQL hỗ trợ đầy đủ, NO nếu không
is_verified_by	character_data	Luôn null, vì nhóm phát triển PostgreSQL không thực hiện kiểm thử chính thức tính năng tuân thủ bình luận character_data. Có khả năng một bình luận về tình trạng được hỗ trợ của phần đó.

34.41. sql_sizing

Bảng sql_sizing chứa thông tin về các giới hạn kích cỡ khác nhau và các giá trị cực đại trong PostgreSQL. Thông tin này ban đầu có ý định sử dụng trong ngữ cảnh của giao diện ODBD; những

người sử dụng các giao diện khác có thể sẽ thấy thông tin này ít được sử dụng. Vì lý do này, các khoản kích cỡ riêng rẽ không được mô tả ở đây; bạn sẽ tìm chúng trong mô tả của giao diện ODBC.

Bảng 34-39. Các cột sql_sizing

Tên	Dạng dữ liệu	Mô tả
sizing_id	cardinal_number	Mã định danh khoản kích cỡ đó
sizing_name	character_data	Tên mô tả khoản kích cỡ đó
supported_value	cardinal_number	Giá trị khoản kích cỡ, hoặc 0 nếu kích cỡ là không giới hạn hoặc không thể xác định được, hoặc null nếu các tính năng đối với chúng khoản kích cỡ có khả năng áp dụng được không được hỗ trợ
comments	character_data	Có khả năng một bình luận gắn với khoản kích cỡ đó

34.42. sql_sizing_profiles

Bảng sql_sizing_profiles chứa thông tin về các giá trị sql_sizing được yêu cầu bằng các hồ sơ tiêu chuẩn SQL khác nhau. PostgreSQL không theo dõi bất kỳ hồ sơ SQL nào, nên bảng này là rỗng.

Bảng 34-40. Các cột sql_sizing_profiles

Tên	Dạng dữ liệu	Mô tả
sizing_id	cardinal_number	Mã định danh khoản kích cỡ đó
sizing_name	character_data	Tên mô tả khoản kích cỡ đó
profile_id	character_data	Chuỗi mã định danh một hồ sơ
required_value	cardinal_number	Giá trị được hồ sơ SQL yêu cầu cho khoản kích cỡ, hoặc 0 nếu hồ sơ đó không đặt ra giới hạn về khoản kích cỡ, hoặc null nếu hồ sơ đó không yêu cầu bất kỳ tính năng nào theo đó khoản kích cỡ có khả năng áp dụng được.
comments	character_data	Có khả năng một bình luận gắn với khoản kích cỡ bên trong hồ sơ đó

34.43. table_constraints

Kiểu nhìn table_constraints chứa tất cả các ràng buộc thuộc về các bảng mà người sử dụng hiện hành sở hữu hoặc vài quyền ưu tiên không với SELECT.

Bảng 34-41. Các cột table_constraints

Tên	Dạng dữ liệu	Mô tả
constraint_catal	og sql_identifier	Tên cơ sở dữ liệu chứa ràng buộc đó (luôn là cơ sở dữ liệu hiện hành)
constraint_sche	ma sql_identifier	Tên sơ đồ chứa ràng buộc đó
constraint_name	sql_identifier	Tên ràng buộc đó
table_catalog	sql_identifier	Tên cơ sở dữ liệu chứa bảng đó (luôn là cơ sở dữ liệu hiện hành)
table_schema	sql_identifier	Tên sơ đồ chứa bảng đó
table_name	sql_identifier	Tên bảng đó
constraint_type	character_data	Dạng ràng buộc: CHECK, FOREIGN KEY, PRIMARY KEY, hoặc UNIQUE
is_deferrable	yes_or_no	YES nếu ràng buộc được hoãn lại, NO nếu không
initially_deferred	yes_or_no	YES nếu ràng buộc hoãn lại được và ban đầu được hoãn lại, NO nếu không

34.44. table_privileges

Kiểu nhìn table_privileges nhận diện tất cả các quyền ưu tiên được trao trong các bảng và kiểu nhìn cho một vai trò hiện hành được kích hoạt hoặc bằng một vài trò hiện hành được kích hoạt. Có một hàng cho từng kết hợp của bảng, người trao và người nhận.

Bảng 34-42. Các cột table_privileges

Tên	Dạng dữ liệu	Mô tả
grantor	sql_identifier	Tên vai trò đã trao quyền ưu tiên đó
grantee	sql_identifier	Tên vai trò đã được trao quyền ưu tiên đó
table_catalog	sql_identifier	Tên cơ sở dữ liệu chứa bảng đó (luôn là cơ sở dữ liệu hiện hành)
table_schema	sql_identifier	Tên sơ đồ chứa bảng đó
table_name	sql_identifier	Tên bảng privilege_type character_data. Dạng quyền ưu tiên: SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, hoặc TRIGGER
is_grantable	yes_or_no	YES nếu quyền ưu tiên có khả năng được trao, NO nếu không
with_hierarchy	yes_or_no	Áp dụng cho một tính năng còn chưa có trong PostgreSQL

34.45. tables

Kiểu nhìn tables chứa tất cả các bảng và kiểu nhìn được xác định trong cơ sở dữ liệu hiện hành. Chỉ các bảng và kiểu nhìn nào được chỉ ra mà người sử dụng hiện hành có sự truy cập (bằng cách là chủ sở hữu hoặc có vài quyền ưu tiên).

Bảng 34-43. Các cột tables

Tên	Dạng dữ liệu	Mô tả
table_catalog	sql_identifier	Tên cơ sở dữ liệu chứa bảng đó (luôn là cơ sở dữ liệu hiện hành)
table_schema	sql_identifier	Tên sơ đồ chứa bảng đó
table_name	sql_identifier	Tên bảng đó
table_type	character_data	Dạng bảng: BASE TABLE cho một bảng cơ sở nhất quán (dạng bảng thông thường), VIEW cho một kiểu nhìn, hoặc LOCAL TEMPORARY cho một bảng tạm
self_referencing_column_name	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
reference_generation	character_data	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
user_defined_type_catalog	sql_identifier	Nếu bảng đó là một bảng có dạng, thì tên đó của cơ sở dữ liệu chứa dạng bảng nằm bên dưới đó (luôn là cơ sở dữ liệu hiện hành, nếu không thì là null.
user_defined_type_schema	sql_identifier	Nếu bảng đó là một bảng có dạng, thì tên đó của sơ đồ chứa dạng dữ liệu nằm bên dưới, nếu không thì là null.
user_defined_type_name	sql_identifier	Nếu bảng là một bảng có dạng, thì tên đó của dạng dữ liệu nằm bên dưới, còn nếu không thì là null.
is_insertable_into	yes_or_no	YES nếu bảng đó có khả năng được chèn vào, NO nếu không (các bảng cơ bản luôn có khả năng chèn được vào, các kiểu nhìn thì không nhất thiết).

Tên	Dạng dữ liệu	Mô tả
is_typed	yes_or_no	YES nếu bảng đó là một bảng có dạng, NO nếu không
commit_action	character_data	Nếu bảng đó là một bảng tạm thời, thì PRESERVE, nếu không sẽ null. (Tiêu chuẩn SQL xác định các hành động thực hiện khác cho các bảng tạm, chúng không được PostgreSQL hỗ trợ).

34.46. triggered_update_columns

Đối với các trigger trong cơ sở dữ liệu hiện hành mà chỉ định một danh sách cột (như UPDATE OF column1, column2), thì kiểu nhìn triggered_update_columns nhận diện các cột đó. Các trigger mà không chỉ định một danh sách cột sẽ không được đưa vào trong kiểu nhìn đó. Chỉ những cột nào được chỉ ra mà người sử dụng hiện hành hoặc có vài quyền ưu tiên không SELECT.

Bảng 34-44. Các cột triggered_update_columns

Tên	Dạng dữ liệu	Mô tả
trigger_catalog	sql_identifier	Tên cơ sở dữ liệu chứa trigger đó (luôn là cơ sở dữ liệu hiện hành)
trigger_schema	sql_identifier	Tên sơ đồ chứa trigger đó
trigger_name	sql_identifier	Tên trigger đó
event_object_catalog	sql_identifier	Tên cơ sở dữ liệu chứa bảng mà trigger đó được xác định trong đó (luôn là cơ sở dữ liệu hiện hành)
event_object_schema	sql_identifier	Tên sơ đồ chứa bảng mà trigger đó được xác định trong đó
event_object_table	sql_identifier	Tên bảng mà trigger đó được xác định trong đó
event_object_column	sql_identifier	Tên cột mà trigger đó được xác định trong đó

34.47. triggers

Kiểu nhìn triggers chứa tất cả các trigger được xác định trong cơ sở dữ liệu hiện hành trong các bảng mà người sử dụng hiện hành sở hữu hoặc có vài quyền ưu tiên không SELECT.

Bảng 34-45. Các cột triggers

Tên	Dạng dữ liệu	Mô tả
trigger_catalog	sql_identifier	Tên cơ sở dữ liệu chứa trigger đó (luôn là cơ sở dữ liệu hiện hành)
trigger_schema	sql_identifier	Tên sơ đồ chứa trigger đó
trigger_name	sql_identifier	Tên trigger đó
event_manipulation	character_data	Sự kiện mà tạo ra trigger đó (INSERT, UPDATE, hoặc DELETE)
event_object_catalog	sql_identifier	Tên cơ sở dữ liệu chứa bảng mà trigger được xác định trong đó (luôn là cơ sở dữ liệu hiện hành)
event_object_schema	sql_identifier	Tên sơ đồ chứa bảng mà trigger đó được xác định trong đó
event_object_table	sql_identifier	Tên bảng mà trigger đó được xác định trong đó
action_order	cardinal_number	Còn chưa được triển khai
action_condition	character_data	Điều kiện WHEN của trigger, null nếu không có (cũng null nếu bảng đó không được một vai trò hiện hành được kích hoạt sở hữu)

Tên	Dạng dữ liệu	Mô tả
action_statement	character_data	Lệnh mà được trigger đó thực thi (hiện hành luôn là EXECUTE PROCEDURE function(...))
action_orientation	character_data	Nhận diện liệu trigger đó có chạy một lần hay không cho từng hàng hoặc một lần cho từng lệnh (ROW hoặc STATEMENT)
condition_timing	character_data	Thời gian ở đó trigger chạy (BEFORE hoặc AFTER)
condition_reference_old_tabe	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
condition_reference_new_table	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
condition_reference_old_row	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
condition_reference_new_row	sql_identifier	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
created	time_stamp	Áp dụng cho một tính năng còn chưa có trong PostgreSQL

Các trigger trong PostgreSQL có 2 khả năng không tương thích với tiêu chuẩn SQL mà ảnh hưởng tới sự đại diện trong sơ đồ thông tin. Thứ nhất, các tên trigger là cục bộ địa phương đối với bảng trong PostgreSQL, thay vì đang là các đối tượng sơ đồ độc lập. Vì thế có thể bị đúp bản các tên trigger được xác định trong một sơ đồ, miễn là chúng thuộc về các bảng khác. (trigger_catalog và trigger_schema thực sự là các giá trị gắn với bảng mà trigger đó được xác định trong đó). Thứ 2, các trigger có thể được xác định để chạy trong nhiều trường hợp trong PostgreSQL (như, ON INSERT OR UPDATE), trong đó tiêu chuẩn SQL chỉ cho phép một. Nếu một trigger được xác định để chạy trong nhiều trường hợp, nó được đại diện như nhiều hàng trong sơ đồ thông tin, một cho từng dạng trường hợp. Như một sự tuần tự của 2 vấn đề đó, khóa chính của kiểu nhìn triggers thực sự là (trigger_catalog, trigger_schema, trigger_name, event_object_table, event_manipulation) thay vì (trigger_catalog, trigger_schema, trigger_name), điều là những gì tiêu chuẩn SQL chỉ định. Dù vậy, nếu bạn xác định các trigger của bạn theo một cách thức tuân thủ với tiêu chuẩn SQL (các tên trigger độc nhất trong sơ đồ và chỉ một dạng sự kiện cho từng trigger), thì điều này sẽ không ảnh hưởng tới bạn.

34.48. usage_privileges

Kiểu nhìn usage_privileges nhận diện các quyền ưu tiên USAGE được trao trong các dạng đối tượng khác nhau cho một vai trò hiện hành được kích hoạt hoặc bằng một vai trò hiện hành được kích hoạt. Trong PostgreSQL, điều này hiện áp dụng cho các miền, các trình bọc dữ liệu ngoài, và các máy chủ ngoài. Có một hàng cho từng sự kết hợp của đối tượng, người trao và người nhận.

Vì các miền không có các quyền ưu tiên thực sự trong PostgreSQL, kiểu nhìn này chỉ ra các quyền ưu tiên ngầm có thể không trao USAGE được chủ sở hữu trao cho PUBLIC cho tất cả các miền. Tuy nhiên, các dạng đối tượng khác, chỉ ra các quyền ưu tiên thực tế.

Bảng 34-46. Các cột usage_privileges

Tên	Dạng dữ liệu	Mô tả
grantor	sql_identifier	Tên vai trò đã trao quyền ưu tiên đó
grantee	sql_identifier	Tên vai trò đã được trao quyền ưu tiên đó

Tên	Dạng dữ liệu	Mô tả
object_catalog	sql_identifier	Tên cơ sở dữ liệu chứa đối tượng đó (luôn là cơ sở dữ liệu hiện hành)
object_schema	sql_identifier	Tên sơ đồ chứa đối tượng đó, nếu áp dụng được, nếu không thì một chuỗi rỗng
object_name	sql_identifier	Tên đối tượng đó
object_type	character_data	DOMAIN hoặc FOREIGN DATA WRAPPER hoặc FOREIGN SERVER
privilege_type	character_data	Luôn là USAGE
is_grantable	yes_or_no	YES nếu quyền ưu tiên có khả năng được trao, NO nếu không

34.49. user_mapping_options

Kiểu nhìn user_mapping_options chứa tất cả các lựa chọn được xác định cho các ánh xạ người sử dụng trong cơ sở dữ liệu hiện hành. Chỉ những ánh xạ người sử dụng nào được chỉ ra ở những nơi mà người sử dụng hiện hành có sự truy cập tới máy chủ ngoài tương ứng (bằng cách là chủ sở hữu hoặc có vài quyền ưu tiên).

Bảng 34-47. Các cột user_mapping_options

Tên	Dạng dữ liệu	Mô tả
authorization_identifier	sql_identifier	Tên người sử dụng đang được ánh xạ, hoặc PUBLIC nếu ánh xạ là công khai
foreign_server_catalog	sql_identifier	Tên cơ sở dữ liệu mà máy chủ ngoài được sử dụng bằng việc ánh xạ này được xác định (luôn là cơ sở dữ liêu hiện hành)
foreign_server_name	sql_identifier	Tên của máy chủ ngoài được việc ánh xạ này sử dụng
option_name	sql_identifier	Tên của một lựa chọn
option_value	character_data	Giá trị của lựa chọn. Cột này sẽ chỉ ra null trừ phi người sử dụng hiện hành là người sử dụng đang được ánh xạ, hoặc việc ánh xạ là cho PUBLIC và người sử dụng hiện hành là chủ sở hữu máy chủ, hoặc người sử dụng hiện hành là một siêu người sử dụng. Ý định là để bảo vệ thông tin mật khẩu được lưu như là lựa chọn ánh xạ của người sử dụng.

34.50. user_mappings

Kiểu nhìn này chứa tất cả các ánh xạ người sử dụng được xác định trong cơ sở dữ liệu hiện hành. Chỉ những ánh xạ người sử dụng được chỉ ra ở những nơi người sử dụng hiện hành có sự truy cập tới máy chủ ngoài tương ứng (bằng cách là chủ sở hữu hoặc có vài quyền ưu tiên).

Bảng 34-48. Các cột user_mappings

Tên	Dạng dữ liệu	Mô tả
authorization_identifier	sql_identifier	Tên người sử dụng đang được ánh xạ, hoặc PUBLIC nếu ánh xạ là công khai
foreign_server_catalog	sql_identifier	Tên cơ sở dữ liệu mà máy chủ ngoài được sử dụng bằng việc ánh xạ này được xác định trong đó (luôn là cơ sở dữ liệu hiện hành)
foreign_server_name	sql_identifier	Tên máy chủ ngoài được việc ánh xạ này sử dụng

34.51. view_column_usage

Kiểu nhìn view_column_usage nhận diện tất cả các cột được sử dụng trong biểu thức truy vấn của một kiểu nhìn (lệnh SELECT xác định kiểu nhìn đó). Một cột duy nhất được đưa vào nếu bảng chứa cột đó được một vai trò hiện hành được kích hoạt sở hữu.

Lưu ý: Cột của các bảng hệ thống sẽ không được đưa vào. Điều này sẽ được sửa lúc nào đó.

Bảng 34-49. Các cột view_column_usage

Tên	Dạng dữ liệu	Mô tả
view_catalog	sql_identifier	Tên cơ sở dữ liệu chứa kiểu nhìn đó (luôn là cơ sở dữ liệu hiện hành)
view_schema	sql_identifier	Tên sơ đồ chứa kiểu nhìn đó
view_name	sql_identifier	Tên kiểu nhìn đó
table_catalog	sql_identifier	Tên cơ sở dữ liệu chứa bảng có cột được kiểu nhìn đó sử dụng (luôn là cơ sở dữ liệu hiện hành)
table_schema	sql_identifier	Tên của sơ đồ chứa bảng có cột được kiểu nhìn đó sử dụng
table_name	sql_identifier	Tên bảng chứa cột được kiểu nhìn đó sử dụng
column_name	sql_identifier	Tên cột được kiểu nhìn đó sử dụng

34.52. view_routine_usage

Kiểu nhìn view_routine_usage nhận diện tất cả các thủ tục (các hàm và thủ tục) được sử dụng trong biểu thức truy vấn của một kiểu nhìn (lệnh SELECT xác định kiểu nhìn đó). Một thủ tục duy nhất được đưa vào nếu thủ tục đó được một vai trò hiện hành được kích hoạt sở hữu.

Bảng 34-50. Các cột view_routine_usage

Tên	Dạng dữ liệu	Mô tả
table_catalog	sql_identifier	Tên cơ sở dữ liệu chứa kiểu nhìn đó (luôn là cơ sở dữ liệu hiện hành)
table_schema	sql_identifier	Tên sơ đồ chứa kiểu nhìn đó
table_name	sql_identifier	Tên kiểu nhìn đó
specific_catalog	sql_identifier	Tên cơ sở dữ liệu chứa hàm đó (luôn là cơ sở dữ liệu hiện hành)
specific_schema	sql_identifier	Tên sơ đồ chứa hàm đó
specific_name	sql_identifier	“Tên đặc biệt” của hàm đó. Xem Phần 34.33 để có thêm thông tin.

34.53. view_table_usage

Kiểu nhìn view_table_usage nhận diện tất cả các bảng được sử dụng trong biểu thức truy vấn của một kiểu nhìn (lệnh SELECT xác định kiểu nhìn đó). Một bảng chỉ được đưa vào nếu bảng đó được một vai trò hiện hành được kích hoạt sở hữu.

Lưu ý: Các bảng hệ thống không được đưa vào. Điều này sẽ được sửa lúc nào đó.

Bảng 34-51. Các cột view_table_usage

Tên	Dạng dữ liệu	Mô tả
view_catalog	sql_identifier	Tên cơ sở dữ liệu chứa kiểu nhìn đó (luôn là cơ sở dữ liệu hiện hành)
view_schema	sql_identifier	Tên sơ đồ chứa kiểu nhìn đó
view_name	sql_identifier	Tên kiểu nhìn đó
table_catalog	sql_identifier	Tên cơ sở dữ liệu chứa bảng mà được kiểu nhìn đó sử dụng (luôn là cơ sở dữ liệu hiện hành)
table_schema	sql_identifier	Tên sơ đồ chứa bảng được kiểu nhìn đó sử dụng
table_name	sql_identifier	Tên bảng được kiểu nhìn đó sử dụng

34.54. views

Kiểu nhìn views chứa tất cả các kiểu nhìn được xác định trong cơ sở dữ liệu hiện hành. Chỉ các kiểu nhìn nào được chỉ ra mà người sử dụng hiện hành có truy cập tới (bằng cách là chủ sở hữu hoặc có vài quyền ưu tiên).

Bảng 34-52. Các cột views

Tên	Dạng dữ liệu	Mô tả
table_catalog	sql_identifier	Tên cơ sở dữ liệu chứa kiểu nhìn đó (luôn là cơ sở dữ liệu hiện hành)
table_schema	sql_identifier	Tên sơ đồ chứa kiểu nhìn đó
table_name	sql_identifier	Tên kiểu nhìn đó
view_definition	character_data	Biểu thức truy vấn xác định kiểu nhìn (null nếu kiểu nhìn đó không được một vai trò hiện hành được kích hoạt sở hữu)
check_option	character_data	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
is_updatable	yes_or_no	YES nếu kiểu nhìn đó có khả năng cập nhật được (cho phép UPDATE và DELETE), NO nếu không
is_insertable_int	yes_or_no	YES nếu kiểu nhìn đó có khả năng chèn được vào (cho phép INSERT), NO nếu không
is_trigger_updatable	yes_or_no	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
is_trigger_deletable	yes_or_no	Áp dụng cho một tính năng còn chưa có trong PostgreSQL
is_trigger_insertable_into	yes_or_no	Áp dụng cho một tính năng còn chưa có trong PostgreSQL