بسمه تعالی

کتاب راهنمای کار با پایگاه داده

PostgreSQL

تهیه کننده:

نگار الماسیان

آرش معبودی

فهرست مطالب

۱- سرآغاز

۱-۱- PostgreSQL چیست؟

PostgreSQL یک سیستم مدیریتی پایگاه داده(Object-Relational (ORDBMS می‌باشد که بر اساس POSTGRES version 4.21 در بخش علوم کامپیوتری برکلی دانشگاه کالیفرنیا توسعه داده شده است.

پروژه POSTGRES توسط پرفسور Michael Stonebraker هدایت و گروه هایی چون آژانس پروژه های تحقیقات پیشرفته دفاع (DARPA)، اداره تحقیقات ارتش(ARO)، بنیاد علوم ملی (NFS) ، ESL و اعضای وابسطه دیگری، حامیان آن بودند .

PostgreSQL نسخه Open-Source ازاین کد اصلی برکلی می باشد و از زبان SQL92/SQL99 و دیگر ابزارهای امروزی پشتیبانی می کند.

اکنون POSTGRES بعنوان پیشگام بسیاری از مفاهیم Object-Relational، در بعضی از پایگاه داده های تجاری عرضه میگردد. در سیستم مدیریت پایگاه داده Relational (RDBMS) قدیمی، از مجموعه نام های وابسته، که همگی شامل صفاتی همگون بودند پشتیبانی می‌شد و در سیستمهای تجاری فعلی، انواعی شامل Floating Point Number ، Integer، Character String، Money وDate قابل پشتیبانی می باشند. این مسئله نیز بدیهی است که این مدل برای برنامه های Data Processing آینده کافی نیست.

PostgreSQL چند قابلیت مهم اضافی را بطریقی که کاربر توانایی توسعه سیستم را دارا باشد در کنارمفاهیم زیر عرضه می دارد:

Inheritance

Data Type

---

۱. http://jakarta.apache.org/ant/index.html

[1] http://s2k-ftp.CS.Berkeley.EDU:8000/postgres/postgres.html

Function

و نیز ابزارهای دیگری که شامل قابلیت ها و انعطاف بیشتری می باشند:

Constraints

Triggers

Rules

Transactional Integrity

این قابلیت ها PostgreSQL را در زمره پایگاه داده Object-Relational قرار داده است وقابل توجه است که مفاهیم فوق وجه تمایزی با پایگاه های داده یی که با عنوان Object-Oriented عرضه شده اند - و با پایگاه های داده وابسته قدیمی سازگاری کامل ندارند- محسوب می شوند. بنابراین هر چند که PostgreSQL بعضی از قابلیتهای مدل Object-Oriented را دارد اما در رده پایگاه های داده Relational شناخته می‌شود.

۱-۲- تاریخچه ای بر PostgreSQL

امروزه سیستم مدیریت پایگاه های داده Object-Relational که به عنوان PostgreSQL شناخته شده است (و بطور خلاصه Postgres95 خطاب می شود) از بسته نرم افزاری POSTGRES که در دانشگاه کالیفرنیا در برکلی تولید شد بر گرفته شده است. PostgreSQL با بیش از یک دهه توسعه، پیشرفته ترین پایگاه داده اپن سورس در سراسر دنیاست که ارائه دهنده کنترل همزمان نسخه های متنوع، پشتیبانی از همه ساختارهایSQL (شامل گزینشهای تودرتو(Subselects)، Transactions ، توابع و انواع داده ای که کاربر تعریف میکند) و تعداد بسیار زیادی از زبانهای قابل اتصال (مانند C، C++، Java، Perl، Tcl و Python) می باشد.

۱-۳- پروژه PostgreSQL در دانشگاه برکلی

اجرای پروژه POSTGRES DBMS در سال ۱۹۸۶ آغاز گردید و پس از آن POSTGRES چندین انتشار را پشت سر گذاشت و اولین "نمونه افزار" (demo ware) سیستم در سال ۱۹۸۷ قابل استفاده و در کنفرانس ACM-SIGMOD سال ۱۹۸۸ عرضه گردید. نسخه ۱ در ماه ژوئن سال ۱۹۸۹ در اختیار تعدادی چند از کاربران آزاد قرار داده شد. سیستم قوانین POSTGRES در واکنش

بـه یـک انتقـاد از اولـین سیسـتم قـوانین (A commentary on the POSTGRES rules system) مجدداً طراحی شد(یعنی اصلاحاتی روی Rule هـا،procedureهـا، CachingوViewهـا در سیسـتم پایگاه داده) و نسخه ۲ آن به همراه سیستم قوانین جدید در ژوئـن ۱۹۹۰ عرضـه گردیـد. در سـال ۱۹۹۱ نسخه ۳ با اضافه نمودن پشتیبانی از سیستم مدیریت ذخایرمتنوع، یک اجرا کننده در خواست بهینه شده و یک سیستم قوانین قابل ویرایش از نو طراحـی و وارد بـازار شـد. همچنـین ویرایشـهای بعدی تا نسخهPOSTGRES95، در بسیاری از قسمتها، بر روی قابلیتهای جابجایی و پایداری متمرکز شده بودند.

POSTGRES، درگیر اجرای تحقیقات و تولید برنامه هـایی از قبیـل: سیسـتم تحلیـل اطلاعـات مالی، کنترل بهره وری موتور جت، پایگاه داده برای دنبـال کـردن سـتاره هـا، پایگـاه داده اطلاعـات پزشکی و چندین سیستم اطلاعات جغرافیایی شد. همچنـین در بعضـی از دانشـگاهها POSTGRES تبدیل به یک ابزار آموزشی گردید. در نهایت فناوری اطلاعات Illustrate (که بعدهها بـه Informix1 پیوست و اکنون وابسته به IBM۲ است.) با در اختیار گرفتن کـدهای مرجـع، آن را تجـاری سـاخت. بدین سان POSTGRES در اواخر ۱۹۹۲، اولین مدیر پایگاه داده در پروژه محاسبات علمی Sequoia 20003 گردید.

در سال ۱۹۹۳ تعداد کـاربران آزاد دو برابرشـد و واضـح بـود کـه نگهـداری از کـدهای اولیـه و پشتیبانی، زمان بسیار زیادی را که باید به تحقیقات روی پایگاه داده صرف می‌شد به خـود اختصـاص داده است. بنابراین در تلاشی جهت کـاهش بـار ایـن پشـتیبانی هـا پـروژه Berkeley POSTGRES بصورت رسمی با نسخه ۴،۲ به کار خود پایان بخشید.

---

۱. http://jakarta.apache.org/ant/index.html

1 http://www.informix.com/
2 http://www.ibm.com/
3 http://meteora.ucsd.edu/s2k/s2k_home.html

Postgres95 -۴-۱

در سال ۱۹۹۴ Andrew Yu و Jolly Chen یک مترجم زبان SQL به POSTGRES افزودند و متعاقباً Postgres95 را از طریق وب انتشار دادند.

Postgres95 به طور کامل در ANSI C شکل گرفت و اندازه آن نیز ۲۵ درصد کاهش یافت. همچنین بسیاری از تغییرات درونی، به بهره وری و قابلیت بازیابی سیستم افزودند. بنا بر استاندارد وسکانسن، نسخه 1.0.x، POSTGRES95، ۳۰ تا ۵۰ درصد سریعتر از نسخه ۴،۲، POSTGRES انتشار یافت، همچنین در کنار رفع برخی ازایرادهای موجود، موارد زیر از جمله پیشرفت های مهم در POSTGRES95 برشمرده می شوند:

- زبان در خواست(query) PostQUEL، با SQL تعویض گردید (در Server تعبیه گردید.). البته در نسخه های قبلی از قابلیت (Subquery) پشتیبانی نمی‌شد و این قابلیت توسط توابعی که کاربر در SQL تعریف می کرد شبیه سازی می شد. Aggregate function ها دوباره پیاده سازی شدند و امکاناتی چون عبارات GROUP BY جهت استفاده در Queryها افزوده شد. همچنین اینترفیس Libpq جهت برنامه نویسی در C باقی ماند.

- به منظور مانیتورینگ برنامه ها، یک برنامه جدید به نام Psql برای ارتباط محاوره ای با queryهای SQL فراهم گردید.

- یک کتابخانه Front-end جدید به نام Libpgtcl، که از سرویس گیرنده های مبتنی بر TCL پشتیبانی می‌کند اضافه شد. برای نمونه می توان به پوسته pgtclsh اشاره نمود که دستورات Tcl جدیدی برای ارتباط برنامه Tcl با Postgres95 Back-end فراهم کرده است.

- رابط Large-object بازنگری گردید. وارونه سازی، تنها راه برای ذخیره مقادیر بزرگ گردید (وارونه سازی فایل سیستم حذف گردید.).

- instance-level rule system، حذف شد.

- یک خودآموز مختصر، به توصیف ابزارهای رایج SQL می پردازد، البته به همان خوبی که POSTGRES95 به همراه کدهای مرجع منتشر می کرد.

- از GNU make به جای BSD make استفاده شد. همچنین POSTGRES95 می توانست توسط کامپایلر GCC اصلاح نشده کامپایل شود.

PostgreSQL -۵-۱

در سال ۱۹۹۶ بود که مشخص شد نام Postgres95 برای ادامه کار مناسب نیست بنابراین ما نام PostgreSQL را برای انعکاس رابطه POSTGRES اولیه و نسخه های اخیر که قابلیتهای SQL را دارا بودند برگزیدیم تا با تمام نسخه ها سازگار باشد و در همین زمان شماره گذاری نسخه ها را نیز از 6.0، آغاز نمودیم.

تأکید اصلی برای توسعه Postgres۹۵ بر روی تشخیص و درک مشکلات موجود در کدهای Back-end بود. با PostgreSQL در حالی که همچنان به همه جوانب توجه می شد، تأکید و اهمیت برروی توسعه و تکمیل ابزارها و قابلیت ها افزایش یافت.

پیشرفتهای چشمگیر PostgreSQL شامل موارد زیر می باشند:

- قفل Table-level با کنترلرهای چند مفسر همزمان (multiversion concurrency control) جایگزین شدند، این تغییر به خواننده ها اجازه می داد تا به خواندن اطلاعات مشترک که نویسنده ها در حال نگارش آن بودند به صورت همزمان ادامه دهند و قادر باشند از pg_dump هنگامی که پایگاه داده برای پاسخ به Query ها آزاد است، پشتیبان بلادرنگ تهیه نمایند.

- ابزارهای بسیار مهمی از قبیل subselects، defaults،constrains و triggers تعبیه شدند.

- ابزارهای اضافی سازگار با زبان SQL92 که به سیستم اضافه شدند شامل Primary key، Quoted identifire، تصحیح کردن تایپ جملات لفظی، Typa casting، ورودی های باینری و اعداد صحیح هگزادسیمال بودند.

- Built-in typeها اصلاح شدند و شامل طیف وسیعی از انواع date/timeهای جدید و انواع جغرافیایی جدید گردیدند.

-هنگامی که نسخه 6.0، انتشار یافت سرعت کلیه کدهای Backe-end بصورت تقریبی ۲۰ تا ۴۰ درصد افزایش و زمان آماده به کار شدن آنها تا ۸۰ درصد کاهش یافت.

۱-۶- اصطلاحات و نمادها

بطور کلی شخصی که نصب و راهبری سرویس دهنده ها را به عهده دارد مدیر سیستم (Administrator) خطاب می شود و کاربر کسی است که توانایی استفاده از PostgreSQL را داشته و خواستاردسترسی به قسمتهای مختلف PostgreSQL باشد. البته توجه داشته باشید این اصطلاحات نباید به صورت خیلی محدود تفسیر شود و این مستندات و مراجع نیزحدود کاملاً ثابتی را برای رویه های مدیریت بیان نمی کند.

ما از /usr/local/pgsql به عنوان شاخه ریشه نصب و از /usr/local/pgsql/data به عنوان شاخه محتوی فایلهای پایگاه یاد می شود، این آدرس ها در سایت شما ممکن است متنوع باشند؛ که در این رابطه جزئیات بیشتر در Administrator's Guide قابل دسترس می باشند.

درنگارش یک دستور براکتها ([,]) نمایان عبارات اختیاری یا Keywordها هستند و هر آنچه که در آکولادها ({,}) قرار می گیرد و توسط خطوط عمودی " | " از هم جدا می شوند، نمایان این هستند که شما باید یکی از آن عبارات را انتخاب نماید.

مثالها دستوراتی را نشان خواهند داد که از حسابها و برنامه های مختلفی اجرا شده اند، ممکن است دستوراتی که از یک پوسته UNIX اجرا می شوند با علامت دلار"$" شروع شوند و دستوراتی که از حساب یک کاربر ویژه مانند root یا postgres باید اجرا شوند بگونه ای خاص علامت گذاری و به همراه توضیحات ارائه کند. همچنین ممکن است دستورات SQL با علامت "=>" شروع شوند و یا بسته به شرایط هیچگونه علامتی در آغاز نداشته باشند.

توجه: در همه قسمتهای مستندات و مراجع نکاتی که در کنار دستورات قید شده، به صورت متناوب آورده نشده است، لذا مشکلات را با Mailing List ،<pgsql_docs@postgresql.org > گروه مستندات درمیان بگذارید.

۲- راهنمایی برای راهبران PostgreSQL

۲-۱ رویه نصب

۲-۲-۱ نسخه کوتاه

```
./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/postmaster -D /usr/local/pgsql/data >logfile 2>&1 &
/usr/local/pgsql/bin/createdb test
/usr/local/pgsql/bin/psql test
```

نسخه بلند آن در دنباله آورده شده است.

۲-۲-۲ نیازمندیها

به طور کلی یک مودم سازگار با بستر یونیکس می‌تواند PostgreSQL را اجرا کند. لیست بسترهایی که در هنگام انتشار بر روی آنها تستهایی انجام شده است در بخشهای بعدی (۱،۷) آورده شده‌اند. در زیر فهرست سند توزیع، سوالات رایجی در رابطه با بسترهای ویژه آورده شده‌است.

بسته‌های نرم‌افزاری زیر برای ساختن PostgreSQL لازم است:

- GNU make مورد نیاز است: دیگر برنامه‌های make کار نخواهند کرد. GNU make اغلب در زیر نام gmake نصب می‌شود
- شما نیازبه یک کامپایلر ISO/ANSIC دارید: توصیه می‌شود از آخرین نسخه GCC استفاده شود. ولی PostgreSQL دارای این قابلیت می‌باشد که با تعداد زیادی از کامپایلرهای مختلف سازگار است.

- در مرحله اول به منظور بازکردن بسته توزیع، نیازمند gzip می‌باشید.
- کتابخانه GNU Readline (به منظور سهولت در ویرایش خطوط و بازیابی تاریخچه دستورات) به صورت پیش‌فرض استفاده خواهد شد. چنانچه مایل به استفاده از آن نیستید می‌توانید در پیکربندی، گزینه --without-readline را انتخاب کنید. (بر روی NetBSD کتابخانه libedit با readline سازگار هستند و اگر libreadline موجود نباشد، از آن استفاده می‌شود.)
- به منظور نصب بر روی ویندوز NT یا ویندوز ۲۰۰۰ شما نیازمند بسته‌های نرم‌افزاری Cygwin و cygip هستید. برای اطلاع از جزئیات بیشتر به بخش سوالات رایج مراجعه فرمائید.

بسته‌های نرم‌افزاری که در ادامه بررسی می‌شود اختیاری هستند و در پیکربندی اولیه اجباری نمی‌باشد و پس از نصب گزینه‌های اصلی مورد نیاز هستند. به موارد زیر دقت کنید.

- برای ایجاد زبان برنامه‌نویسی سرور PL/Perl شما نیازمند یک نصب کامل Perl می‌باشید. که شامل کتابخانه‌های libperl و فایلهای سرآیند می‌باشد. این کتابخانه در اکثر بسترهاً جزء کتابخانه‌های مشترک می‌باشد. اگر شما کتابخانه مشترک نداشته باشید پیامی به صورت زیر ظاهر می‌شود:

```
*** 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.
```

در اینصورت شما باید دوباره Perl را به صورت دستی نصب کنید تا PL/Perl ساخته شود.

- به منظور ایجاد یک Tcl یا اجزای Tk (کلاینت و زبان PL/TCL) هستید، نیازمند نصب Tcl هستید.

- به منظور ایجاد گرداننده JDBC شما احتیاج به Ant 1.5 یا بالاتر و یک JDK هستید. Ant یک ابزار ویژه به منظور ایجاد بسته‌های نرم‌افزاری بر پایه جاوا می‌باشد که کاربران می‌توانند آنرا از وب‌سایت 1Ant بارگیری کنند.

اگر چند کامپایلر مختلف جاوا بر روی سیستم خود نصب کرده‌اید، بسته به پیکربندی Ant، یکی از آنها استفاده می‌شود. توزیعهای Ant معمولا به منظور خواندن یک فایل antrc. از فهرست شخصی کاربر جاری برای پیکربندی استفاده می‌شود. به عنوان مثال: برای استفاده از یک JDK متفاوت از آنچه که به‌طور پیش‌فرض وجود دارد، شاید از راه زیر به نتیجه برسید:

```
JAVA_HOME=/usr/local/sun-jdk1.3
JAVACMD=$JAVA_HOME/bin/java
```

توجه: برای ساختن یک گرداننده از ant یا javac استفاده نکنید. از این راه به نتیجه نمی‌رسید. مطابق روش وعمول که در زیر آمرده شده است gmake را اجرا نمائید.

به منظور فعال‌سازی ویژگی پشتیبانی از زبان بومی (NLS)، یعنی قابلیت نمایش پیغامهای برنامه به زبانی غیر از انگلیسی، شما نیازمند یک اجرا از Gettext API هستید. برخی از سیستم‌عاملها دارای این بسته هستند (مانند لینوکس، NetBSD، سولاریس) ولی در باقی سیستم‌عاملها شما می‌توانید آن‌را از آدرس اینترنتی سایت PostgreSQL که: http://www.postgresql.org/~petere/gettext.html دریافت نموده به بسته نرم‌افزاری خود اضافه نمائید. چنانچه شما از اجرای gettext بر روی کتابخانه C در گنو استفاده می‌کنید شما احتیاج به برخی از بسته‌های GNU Gettext برای برخی از برنامه‌های سودمند هستید. برای دیگر پیاده‌سازیها به آن نیازی ندارید.

- اگر شما تمایل به استفاده و پشتیبانی احراز هویت دارید از سروسیهای Kerberos، OpenSSL یا PAM استفاده نمائید.

---

۱. http://jakarta.apache.org/ant/index.html

اگر شما به جای استفاده از بسته نرم‌افزاری منتشر شده توسط منبع، از درخت CVS استفاده می‌کنید و یا چنانچه تصمیم به توسعه نرم‌افزار دارید، نیازمند بسته‌های زیر هستید:

- برای ساختن یک وارسی CVS یا ایجاد تغییر در فایلهای پویشگر یا تجزیه‌گر اصلی، Flex و Bison لازم است. در صورت نیاز به آنها مطمئن شوید که Flex 2.5.4 یا بالاتر و Bison 1.50 و بالاتر را بگیرید. در برخی از موارد دیگر برنامه‌های yacc نیز مورد استفاده قرار می‌گیرند ولی استفاده از آن سخت تر است و به همین دلیل آنها را پیشنهاد نمی‌کنیم.

اگر می‌خواهید بسته گنو را بگیرید می‌توانید آنرا از نزدیک‌ترین سایت آیینه‌ای گنو (برای مشاهده نام آنها به آدرس http://www.gnu.org/order/ftp.html مراجعه کنید) یا از ftp://ftp.gnu.org/gnu/ دریافت نمائید.

همچنین از کافی بودن فضای دیسک نیز اطمینان حاصل نمائید. شما به ۶۵ مگابایت فضا برای درخت منبع و همچنین ۱۵ مگابایت برای نصب فهرستها نیاز دارید. یک پایگاه داده خوشه‌ای خالی ۲۵ مگابایت فضا اشغال می‌کند که این مقدار ۵ برابر فضایی است که یک فایل متنی ساده با همان مقدار داده اشغال می‌کند. اگر شما تصمیم به اجرای آزمایشات برگشتی دارید نیازمند ۹۰ مگابایت حافظه اضافی دارید. از دستور df برای حصول اطمینان از وجود فضای کافی، استفاده نمائید.

۲-۲-۳ تهیه منبع

منبع PostgreSQL 7.3.2 را می‌توانید از آدرس ftp://ftp.postgresql.org/pub/postgresql-7.3.2.tar.gz فراهم نمائید. چنانچه برای شما میسر است از یک سایت آئینه‌ای استفاده نمائید. پس از گرفتن پرونده، باید آنرا با استفاده از دستور زیر باز کنید:

```
gunzip postgresql-7.3.2.tar.gz
tar xf postgresql-7.3.2.tar
```

پس از اجرای این دستورفهرستی با نام PostgreSQL 7.3.2 در شاخه جاری که منبع PostgreSQL 7.3.2 در آن قرار دارد ایجاد می‌گردد. برای انجام ادامه فرآیند نصب به این فهرست وارد شوید.

۲-۲-۴ چنانچه در حال بروزرسانی هستید

در اینصورت قالب ذخیره‌گاه داخلی داده، با نسخه جدید PostgreSQL جایگزین می‌شود. اگر شما می‌خواهید نسخه‌ای را که دارای شماره نگارش "7.3.x" نمی‌باشد را به روزرسانی کنید، باید نسخه پشتیبانی مطابق دستورات زیر از اطلاعاتتان تهیه نمائید. فرض کنید که نسخه نصب شده فعلی شما در شاخه /usr/local/pgsql قرار داشته باشد و محدوده داده‌های شما در اینجا /usr/local/pgsql/data قرار داشته باشد. مسیر متناسب خود را جایگزین کنید.

۱. در ابتدا اطمینان حاصل نمائید که در طول یا پس از فرآیند ایجاد نسخه پشتیبان پایگاه‌داده بروزرسانی نمی‌شود. البته این مسئله تاثیری بر روی صحت ایجاد نسخه پشتیبان ندارد ولی مسلما تغییرات انجام شده بر روی داده‌ها در این زمان اعمال نمی‌گردد. به منظور جلوگیری از ایجاد تغییرات توسط دیگران چنانچه لازم است مجوز کاربری آنها را به فایل /usr/local/pgsql/data/pg_hba.conf تغییر دهید.

۲. برای ایجاد نسخه پشتیبان از نصب پایگاه‌داده خط زیر را اجرا نمائید:

```
pg_dumpall > outputfile
```

اگر تمایل به حفظ OIDها (مانند زمانی که از آنها به عنوان کلید خارجی استفاده می‌کنید) دارید هنگامی که از pg_dumpall استفاده می‌کنید، گزینه o– را نیز به آن بیافزائید.

با استفاده از دستور pg_dumpall شما می‌توانید برای نسخه‌ای که هم‌اکنون از آن استفاده می‌کنید نسخه پشتیبان تهیه کنید.

۳. اگر نسخه جدید را در همان شاخه‌ای که نسخه قدیمی در آن قرار داشت نصب می‌کنید، قبل از نصب فایلهای جدید، سرور قدیمی را با این دستور پائین بیاورید:

```
kill -INT `cat /usr/local/pgsql/data/postmaster.pid`
```

نسخه‌های قبل از 7.0 فایل postmaster.pid را نداشت. اگر شما از چنین نسخه‌ای استفاده می‌کنید، باید شماره شناسه فرآیند سرور را مثالا با استفاده از دستور ps ax | grep پیدا کرده و سپس با استفاده از دستور kill آنرا متوقف سازید.

در سیستمهایی که PostgreSQL در زمان بالا آمدن شروع به کار می‌کند، احتمالا یک فایل شروع کننده وجود دارد که همیین وظیفه را بر عهده دارد. به عنوان مثال در یک سیستم لینوکس ردهت ممکن است این فایل در شاخه /etc/rc.d/init.d/postgresql stop قرار داشته باشد احتمال دیگر pg_ctl stop می‌باشد.

۴. اگر نسخه جدید را در همان شاخه‌ای که نسخه قدیمی در آن قرار داشت نصب می‌کنید بهترین راه‌حل این است که تمامی فایلهای قدیمی نصب شده را با دستور زیر از بین ببرید:

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

پس از نصب PostgreSQL 7.3.2، یک شاخه برای پایگاه‌داده جدید ایجاد کنید و سرور جدید را راه بیاندازید. به خاطر داشته باشید که شما پس از وزود به یک حساب کاربری پایگاه‌داده می‌توانید دستورهای زیر را اجرا نمائید:

```
/usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
/usr/local/pgsql/bin/postmaster -D /usr/local/pgsql/data
```

در نهایت داده‌های خود را با دستور زیر بازگردانید:

```
/usr/local/pgsql/bin/psql -d template1 -f outputfile
```

و سپس از psql جدید خود استفاده نمائید.

۲-۲-۵ فرآیند نصب

۱. پیکربندی

اولین مرحله از نصب، پیکربندی درخت منبع برای سیستم شما و انتخاب گزینه‌هایی است که شما می‌خواهید. که توسط اجرای اسکریپت پیکربندی انجام خواهد شد. در حالت نصب به صورت پیش‌فرض دستور زیر را وارد نمائید.

```
./configure
```

این اسکریپت برای تعیین متغییرهای وابسته، آزمایشهایی را انجام می دهد و نقایص سیستم عامل را شناسایی می‌کند و در نهایت در درخت ساخته شده فایلهایی را ایجاد می‌کند و آنچه را که پیدا کرده در آنجا نگه می‌دارد. (چنانچه شما می‌خواهید شاخه ایجاد را از درخت منبع جدا نگه دارید، می‌توانید پیکربندی را در خارج درخت منبع اجرا کنید.)

پیکربندی پیش‌فرض، سرور و ابزارها را ایجاد می‌کند، نرم‌افزارهای کاربردی و واسط فقط نیاز به یک مترجم C دارند. تمامی فایلها به صورت پیش‌فرض در شاخه /usr/local/pgsql نصب می‌شوند.

شما می‌توانید رویه نصب و ایجاد را سفارسی‌سازی کنید. برای این منظور یک یا چند خط از دستورات زیر را برای ایجاد گزینه‌های پیکربندی اجرا نمائید.

`--prefix=PREFIX`

تمامی فایلها را به جای شاخه /usr/local/pgsql در فهرست PREFIX نصب کنید. فایلهای حقیقی در زیرفهرستهای مختلفی نصب خواهند شد. هیچ فایلی به صورت مستقیم در فهرست PREFIX نصب نخواهد شد.

اگر شما نیازمندیهای خاصی دارید می‌توانید زیرفهرستهای ویژه‌ای را با دستور زیر بسازید:

`--exec-prefix=EXEC-PREFIX`

شما می‌توانید وابستگیهای معماری را در یک prefix ، یعنی EXEC-PREFIX، دیگر نصب کنید. به اشتراک گذاشتن وابستگیهای معماری در بین میزبانها، نیز کار سودمندی می‌باشد. اگر شما این کار را نکنید، EXEC-PREFIX برابر PREFIX قرار گرفته و فایلهای معماری مستقل و وابسته در زیر همان درخت نصب خواهد شد.

`--bindir=DIRECTORY`

برای برنامه‌های اجرایی شاخه جداگانه تعریف کنید. مسیر پیش‌فرض برای اینگونه برنامه‌ها EXEC-PREFIX/bin می‌باشد که معمولا به معنی /usr/local/pgsql/bin می‌باشد.

`--datadir=DIRECTORY`

یک شاخه برای داده‌های فقط خواندنی که توسط برنامه‌های نصب شده استفاده می‌شود، قرار دهید. شاخه پیش‌فرض PREFIX/share می‌باشد. توجه کنید که این موضوع ربطی به مکانی که پایگاه داده شما قرار دارد ندارد.

`--sysconfdir=DIRECTORY`

شاخه پیش‌فرضی که برای فایلهای پیکربندی مختلف قرار دارد PREFIX/etc می‌باشد.

`--libdir=DIRECTORY`

محل پیش فرض برای نصب کتابخانه‌ها و پیمانه‌های قابل بارگذاری پویا EXECPREFIX/lib می‌باشد.

`--includedir=DIRECTORY`

محل پیش فرض برای نصب فایلهای سرآیند C و C++، در PREFIX/include می‌باشد.

`--docdir=DIRECTORY`

فایلهای اسناد به استثنای صفحات "man"در شاخه پیش فرض PREFIX/doc خواهد شد.

`--mandir=DIRECTORY`

صفحات man که همراه با PostgreSQL می‌باشد به صورت پیش‌فرض در زیر شاخه PREFIX/man و زیر فهرستهای مربوطه نصب خواهد شد.

نکته: حتما از نصب PostgreSQL بر روی مکانهای مشترک مانند /usr/local/include ، بدون دخالت namespace باقی سیستم، اطمینان حاصل کنید. در ابتدا رشته ”/postgresql“ به صورت خودکار بهdatadir sysconfdir افزوده می‌شود و docdir با توجه به نام رشته که ”postgres“ یا ”pgsql“ می‌باشد پر خواهد شد و گسترش خواهد یافت. به عنوان مثال اگر شما /usr/local را برای prefix انتخاب کنید، اسناد در /usr/local/doc/postgresql نصب خواهد شد، اما اگر prefix انتخابی شما /opt/postgres باشد، اسناد در /opt/postgres/doc قرار خواهد گرفت. فایلهای سرآیند C با واسط کلاینت در includedir نصب می‌شوند و namespace مرتب می‌شود. فایلهای سرآیند داخلی و سروری در یک شاخه خصوصی در زیر includedir. در نهایت اگر لازم باشد یک زیر فهرست خصوصی برای پیمانه‌های قابل بارگذاری پویا در زیر libdir قرار می‌گیرد.

`--with-includes=DIRECTORIES`

DIRECTORIES یک فهرست کامل از نام شاخه‌هایی است که به فهرست جستجوهای مترجمهای فایلهای سرآیند اضافه می‌شوند، می‌باشد که با colon از یکدیگر جدا می‌شوند. اگر شما بسته‌های اختیاری دارید که در موقعیتهای غیر استاندارد نصب شده‌اند باید از این گزینه استفاده کنید و گزینه‌های مکملی چون with-libraries-- استفاده نمائید. مثال:

```
--with-includes=/opt/gnu/include:/usr/sup/include
```

```
--with-libraries=DIRECTORIES
```

DIRECTORIES یک فهرست از نام شاخه‌ها است برای جستجوی کتابخانه‌ها که با colon از یکدیگر جدا می‌شوند. اگر شما از بسته‌های غیر استاندارد استفاده می‌کنید، احتمالا باید از این گزینه و (گزینه مکمل آن with-includes--) استفاده کنید. مثال:

```
--with-libraries=/opt/gnu/lib:/usr/sup/lib
```

```
--enable-recode
```

Enable نویسه‌گان تک بایتی است که برای خدمات نگهداری قرار داده می‌شود.

```
--enable-nls[=LANGUAGES]
```

تواناساز پشتیبانی از زبانهای ملی (NLS) که امکان نمایش پیامهای برنامه به زبانهای غیر انگلیسی می‌باشد.

LANGUAGES فهرستی از زبانهایی است که شما می‌خواهید پشتیبانی شود و این نامها با فاصله از یکدیگر جدا می‌شوند. به عنوان مثال 'de fr'=enable-nls-- (فصل مشترک فهرست شما و فهرست زبانهای ترجمه شده موجود، به صورت خودکار محاسبه می‌شود.) اگر شما فهرست زبانهای خاصی را تعیین نکنید، تمامی ترجمه‌های ممکن نصب خواهد شد. به منظور استفاده از این گزینه شما باید gettext API را اجرا نمائید.

```
--with-pgport=NUMBER
```

NUMBER را به عنوان شماره درگاه پیش‌فرض برای سرور و کلاینت تنظیم کنید. این شماره ۵۴۳۲ می‌باشد. این شماره در آینده می‌تواند تغییر کند ولی چنانچه شما در اینجا آنرا معین کنید، سرور و کلاینت هردو همین مقدار پیش‌فرض را می‌پذیرند که بسیار مناسب می‌باشد. تنها مزیت انتخاب یک مقدار غیر از مقدار پیش‌فرض برای زمانی است که شما قصد اجرای چند PostgreSQL سرور در یک ماشین را داشته باشید.

```
--with-perl
```

زبان PL/Perl سرور خود را ایجاد کنید.

```
--with-python
```

پیمانه واسط Python و زبان PL/Python سرور را ایجاد نمائید. برای این منظور شما باید توانایی و امکان دسترسی در حد ریشه داشته باشید تا بتوانید پیمانه Python را در محل پیش‌فرض خود کـه /usr/lib/pythonx.y می‌باشد، نصب کنید.

```
--with-tcl
```

اجزایی را که Tcl/Tk نیازدارد ایجاد نمائید. ایـن اجـزا عبارتنـد از libpgtcl, pgtclsh, pgtksh و PL/Tcl. ولی به without-tk-- دقت کنید. اگر شما گزینه with-tcl-- را تعیین کنید، برنامه‌هایی کـه احتیاج به TK(pgtksh) دارند را نادیده گرفته‌اید.

```
--with-tclconfig=DIRECTORY
--with-tkconfig=DIRECTORY
```

TCL/TK فایلهای tclconfig.sh و tkConfig.sh را نصب می‌کند که شـامل اطلاعـات لازم بـرای تنظیمات ایجاد پیمانه‌های واسط TK یا Tcl، مـی‌باشـد. ایـن فایلهـا معمـولا بـه صـورت خودکـار در مکانهای مشور خود پیدا می‌شوند، ولی اگر شما تصمیم دارید که یک نگارش متفاوت از Tcl یـا Tk را استفاده کنید باید شاخه مورد نظر خود را تعیین کنید.

```
--with-java
```

هدایتگر JDBC و دیگر بسته‌های نرم‌افزاری مربوط به جاوا را ایجاد کنید.

```
--with-krb4[=DIRECTORY]
--with-krb5[=DIRECTORY]
```

برای ایجاد پشتیبانی برای تصدیق Kerberos. شما می‌نوانید ازKerberos نگارش ۴ یا ۵ اسـتفاده کنید. آرگومان DIRECTORY برای تعیین شـاخه ریشـهء نصـب Kerberos ؛ کـه /usr/Athena بـه عنوان پیش‌فرض در نظر گرفته شده‌است. اگر فایلهای سرآیند و کتابخانه‌های وابسته در زیر شاخه پدر قرار نگرفته باشد باید از گزینه‌های with-includes-- و with-libraries-- استفاده کنید.

```
--with-krb-srvnam=NAME
```

دستور تعیین نام سرویس Kerberos می‌باشد. Postgres نام پیش‌فرض درنظر گرفته مـی‌شـود و عموما دلیلی برای تعویض آن وجود ندارد.

```
--with-openssl[=DIRECTORY]
```

پشتیبانی از ارتباطات SSL را ایجاد می‌کند. اینکار نیازمند نصب بسته‌های OpenSSL می‌باشد. آرگومان DIRECTORY، شاخه ریشهء OpenSSL را تعیین می‌کند. که به صورت پیش‌فرض /usr/local/ssl در نظر گرفته می‌شود.

```
--with-pam
```

برای ایجاد پشتیبانی از PAM (Pluggable Authentication Modules) استفاده می‌شود.

```
--without-readline
```

از استفادهء کتابخانه Readline جلوگیری می‌کند. این گزینه ایجاد تغییر در خط فرمان و تاریخچه psql را غیر ممکن می‌سازد. بنابراین استفاده از آن توصیه نمی‌شود.

```
--without-zlib
```

از استفادهء کتابخانه Zlib جلوگیری می‌کند. این گزینه امکان فشرده‌سازی در pg_dump را غیر ممکن می‌سازد.

```
--enable-debug
```

تمامی برنامه‌ها و کتابخانه‌ها را ترجمه و اشکال‌زدایی می‌کند. این بدان معناست که شما می‌توانید برنامه‌ها را توسط یک اشکال‌زدا اجرا کرده و مشکلات آنرا تحلیل کنید.

```
--enable-cassert
```

برای آزمایش شرایطی که نباید اتفاق بیافتند به‌کار می‌رود. این گزینه برای اهداف توسعه‌ای بسیار ارزشمند می‌باشد.

```
--enable-depend
```

با این گزینه ویژگی ردیابی خودکار فعال می‌شود.. چنانچه شما توسعه‌دهنده هستید این گزینه بسیار مفید خواهد بود. ولی اگر بخواهید آنرا یکبار نصب و ترجمه کنید زمان سربار زیادی را مصرف می‌کند.

۲. ایجاد

برای شروع به ایجاد نخست بنویسید

```
gmake
```

حتما به خاطر داشته باشید که از GNU make استفاده کنید.) این مرحله بسته به سیستمی شما ممکن است از ۵ تا نیم ساعت به طول انجامد. آخرین خطی که نمایش داده می‌شود باید خط زیر باشد:

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

۳. آزمایشهای بازگشتی

اگر شما می‌خواهید سرور ایجاد شده خود را قبل از نصب امتحان کنید می‌توانید آزمایشهای بازگشتی را اجرا کنید. این آزمایش برای اطمینان از اجرای صحیح PostgreSQL بر روی سیستم مربوطه می‌باشد. برای انجام این آزمایش بنویسید:

```
gmake check
```

(این دستور توسط ریشه اجرا نمی‌شود بلکه توسط یک کاربر غیر ممتاز باید انجام گیرد.) ممکن است برخی آزمایشها مردود شوند.

۴. نصب پرونده‌ها

نکته: اگر شما در حال بروزرسانی یک سیستم موجود هستید و در نظر دارید که فایلهای جدید را بر روی فایلهای قدیمی نصب کنید، حتما باید از داده‌های خود نسخه پشتیبان تهیه کنید و سرور قدیمی را با استفاده از دستوراتی که قبلا گفتیم، خاموش کنید.

برای نصب PostgreSQL خط زیر را وارد کنید:

```
gmake install
```

این دستور فایلها را در شاخه‌هایی که در مرحله قبلی تععین کردیم، نصب خواهد کرد. حتما مطمئن شوید که شما مجوز دسترسی به آن حوزه را داشته باشید. به طور معمول باید این مرحله را از کاربر ریشه اجرا کنید یا به جای ان می‌توانید شاخه‌های هدف ایجاد کرده و مجوزهای دسترسی لازم را در آنجا تعویض کنید.

شما می‌توانید از gmake install-strip به جای gmake install استفاده کنید تا فایلها و کتابخانه‌های اجرائی نصب شده را strip شوند. با این کار در مصرف فضا صرفه‌جوئی می‌کنید. این

دستور برای کارهای منطقی برای صرفه‌جوئی در فضا انجام می‌دهد. ولی درصد خطائی هـم بـرای آن قائل شوید و اگر لازم باشد به صورت دستی این کار را انجام دهید.

اگر شما واسط Python ایجاد کرده‌اید و از کاربر ریشه وارد نشده‌اید، زمانی که دستور بالا را اجـرا می‌کنید، ممکن است قسمتی از نصب انجام نشود. در اینصورت شما باید با کـاربر ریشـه وارد شـده و سپس دستور زیر را اجرا کنید.

```
gmake -C src/interfaces/python install
```

در حالت نصب استاندارد فقط فایلهای سرآیند که برای توسعه نرم‌افزارهای کاربردی کلاینت لازم است نصب می‌شوند. اگر شما تصمیم به توسعه هرگونه برنامه سروری دارید، باید PostgreSQL را بـه طور کامل نصب کنی. برای انجام این کار دستور زیر را اجرا کنید:

```
gmake install-all-headers
```

چنانچه شما می‌خواهید که فقط نرم‌افزارهای کاربردی کلاینت و کتابخانه‌های واسـط آنـرا نصـب کنید باید دستورات زیر را اجرا کنید:

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

لغو نصب: برای انصراف و بازگشت از نصب، دستور gmake uninstall را اجرا نمائید. با این وجـود با این کار، هیچکدام از دایرکتوریهای بوجود آمده از بین نمی‌روند.

پاکسازی: پس از نصب می‌توانید با حذف فایلهای ساخته شده از درخت منبع، اتاق بسازید. بـرای این منظور از دستور gmake clean استفاده کنید. این کار از تولید فایلهای ایجاد شده توسـط برنامـه پیکربندی جلوگیری می‌کند. که بعدا می‌توانید با دستور gmake همه چیـز را بازسـازی کنیـد. بـرای بازنشاندن درخت منبع به حالتی که توزیع شده بود از .use gmake distclean استفاده کنید. اگر در نظر دارید آنرا برای بسترهای مختلف، از یک منبع ایجاد کنید، باید اینکار انجام داده و برای هر ایجاد آنرا پیکربندی دوباره کنید.

۲-۲-۶ تنظیمات پس از نصب

۱. کتابخانه‌های مشترک

در اکثر سیستمهایی که دارای کتابخانه‌های مشترک می‌باشند شما باید به سیستم خود کتابخانه‌های مشترک تازه نصب شده را بشناسانید. سیستمهایی که در آنها این‌کار ضروری نمی‌باشند عبارتند از: BSD/OS, FreeBSD, HP-UX, IRIX, Linux, NetBSD, OpenBSD, Tru64 UNIX و Solaris .

روشهای تنظیم مسیر جستجوی کتابخانه‌های مشترک در بسترهای مختلف، متفاوت می‌باشد. ولی کاربردی‌ترین روش تعیین متغییرهای LD_LIBRARY_PATH مانند: در پوسته Bourne (sh,ksh,bash,zsh)

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

یا در csh یا tcsh:

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

/usr/local/pgsql/lib را با هرچیزی که در مرحله اول برای libdir– تعیین کرده‌اید، جایگزین کنید. این دستورات را در پوسته فایل راه‌انداز مانند /etc/profile یا ~/.bash_profile قرار دهید. اطلاعات بیشتر در رابطه با این روش در آدرس اینترنتی http://www.visi.com/~barr/ldpath.html قرار دارد.

در برخی از سیستمها شاید لازم باشد که متغییرهای LD_RUN_PATH را قبل از ایجاد، تعیین کنیم.

در Cygwin فهرست کتابخانه‌های مشترک را در PATH قرار دهید یا فایل dll.را به شاخه /bin منتقل کنید. اگر شما پیامی مشابه زیر دریافت کردید:

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

آنگاه این مرحله بسیار ضروری می‌باشد.

اگر شما از BSD/OS، Linux یا SunOS 4 استفاده می‌کنید و مجوز دسترسی کاربر ریشه را دارید می‌توانید دستور زیر را پس از نصب، برای توانا سازی پیوند دهنده زمان اجرا و به منظور سرعت در یافتن کتابخانه‌های مشترک اجرا نمائید:

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

در FreeBSD، NetBSD و OpenBSD دستور زیر را اجرا کنید:

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

برای دیگر سیستمهای ناشناخته دستور معادلی وجود ندارد.

۲. متغییرهای محیط

اگر شما PostgreSQL را در مسیر /usr/local/pgsql یا دیگر مکانهایی که در حالت پیش‌فرض جستجو می‌شوند، نصب کرده‌اید شما باید مسیر /usr/local/pgsql/bin (یا هر مسیر دیگری راکه برای bindir-- در مرحله اول تعیین کرده‌اید) را اضافه کنید. بازهم تاکید می‌کنیم که این کار ضروری نیست ولی باعث می‌شود که استفاده از PostgreSQL راحتتر شود.

برای انجام این‌کار اعمال زیر را برای پوسته فایل راه‌انداز خود، مانند ~/.bash_profile (یا اگر می‌خواهید تمامی کاربران را تحت تاثیر قرار دهید فایل /etc/profile، انجام دهید:

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

اگر شما در حال استفاده از csh یا tcsh هستید، از دستور زیر استفاده کنید:

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

به منظور توانا سازی سیستمتان برای یافتن اسناد راهنما، باید این خطوط را به پوسته فایل راه‌انداز خود اضافه کنید:

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

متغییرهای محیط PGHOST و PGPORT نرم‌افزارهای کاربردی پورت میزبان پایگاه‌داده سرور را تعیین می‌کنند. بهتر است که هر کاربر که می‌خواهد نرم‌افزارهای کاربردی کلاینت را از راه دور اجرا کند، PGHOST پایگاه داده را بازنشاند.

۲-۲-۷ بسترهایی که پشتیبانی می‌شوند

PostgreSQl توسط کمیته توسعه‌دهندگان برای کار کردن بر روی بسترهای زیر آزمایش شده و صحت اجرای آن بر روی این بسترها تائید شده‌است. پشتیبانی از یک بستر یعنی PostgreSQL ایجاد شده، طبق دستورات بالا نصب شده و همچنین آزمایشهای بازگشتی نیز بر روی آنها انجام شده‌است.

نکته: اگر در هنگام نصب مشکلی با یکی از بسترها داشتید با آدرسهای اینترنتی pgsql-bugs@postgresql.org یا pgsql-ports@postgresql.org ارتباط برقرار کنید.

OS	Processor	Version	Reported	Remarks
AIX	RS6000	7.3	2002-11-12, Andreas Zeugswetter (<ZeugswetterA@spardat.at>)	see also doc/FAQ_AIX
BSD/OS	x86	7.3	2002-10-25, Bruce Momjian (<pgman@candle.pha.pa.us>)	4.2
FreeBSD	Alpha	7.3	2002-11-13, Chris Kings-Lynne (<chriskl@familyhealth.com.au>)	
FreeBSD	x86	7.3	2002-10-29, 3.3, Nigel J. Andrews (<nandrews@investsystems.co.uk> 4.7, Larry Rosenman (<ler@lerctr.org>), 5.0, Sean Chittenden (<sean@chittenden.org>)	
HP-UX	PA-RISC	7.3	2002-10-28, 10.20 Tom Lane (<tgl@sss.pgh.pa.us>), 11.00, 11.11, 32 & 64 bit, Giles Lean (<giles@nemeton.com.au>)	gcc and cc; see also doc/FAQ_HPUX
IRIX	MIPS	7.3	2002-10-27, Ian Barwick (<barwick@gmx.net>)	Irix64 Komma 6.5
Linux	Alpha	7.3	2002-10-28, Magnus Naeslund (<mag@fbab.net>)	2.4.19-pre6

OS	Processor	Version	Reported	Remarks
Linux	PlayStation 2	7.3	2002-11-19, Permaine Cheung <pcheung@redhat.com>	#undef HAS_TEST_AND_S remove slock_t typedef
Linux	PPC74xx	7.3	2002-10-26, Tom Lane (<tgl@sss.pgh.pa.us>)	bye 2.2.18; Apple G3
Linux	S/390	7.3	2002-11-22, Permaine Cheung <pcheung@redhat.com>	both s390 and s390x (32 and 64 bit)
Linux	Sparc	7.3	2002-10-26, Doug McNaught (<doug@mcnaught.org>)	3.0
Linux	x86	7.3	2002-10-26, Alvaro Herrera (<alvherre@dcc.uchile.cl>)	2.4
MacOS X	PPC	7.3	2002-10-28, 10.1, Tom Lane (<tgl@sss.pgh.pa.us>), 10.2.1, Adam Witney (<awitney@sghms.ac.uk>)	
NetBSD	arm32	7.3	2002-11-19, Patrick Welche (<prlw1@newn.cam.ac.uk>)	1.6
NetBSD	x86	7.3	2002-11-14, Patrick Welche (<prlw1@newn.cam.ac.uk>)	1.6
OpenBSD	Sparc	7.3	2002-11-17, Christopher Kings-Lynne (<chriskl@familyhealth.com.au>)	3.2
OpenBSD	x86	7.3	2002-11-14, 3.1 Magnus Naeslund (<mag@fbab.net>), 3.2 Christopher Kings-Lynne (<chriskl@familyhealth.com.au>)	

OS	Processor	Version	Reported	Remarks
BeOS	x86	7.2	2001-11-29, Cyril Velter (<cyril.velter@libertysurf.fr>)	needs updates to semaphore code

OS	Processor	Version	Reported	Remarks
DG/UX 5.4R4.11	m88k	6.3	1998-03-01, Brian E Gallew (<geek+@cmu.edu>)	no recent reports
Linux	armv4l	7.2	2001-12-10, Mark Knox (<segfault@hardline.org>)	2.2.x
Linux	MIPS	7.2	2001-11-15, Hisao Shibuya (<shibuya@alpha.or.jp>)	2.0.x; Cobalt Qube2
MkLinux DR1	PPC750	7.0	2001-04-03, Tatsuo Ishii (<t-ishii@sra.co.jp>)	7.1 needs OS update?
NetBSD	Alpha	7.2	2001-11-20, Thomas Thai (<tom@minnesota.com>)	1.5W
NetBSD	m68k	7.0	2000-04-10, Henry B. Hotz (<hotz@jpl.nasa.gov>)	Mac 8xx
NetBSD	MIPS	7.2.1	2002-06-13, Warwick Hunter (<whunter@agile.tv>)	1.5.3
NetBSD	PPC	7.2	2001-11-28, Bill Studenmund (<wrstuden@netbsd.org>)	1.5
NetBSD	Sparc	7.2	2001-12-03, Matthew Green (<mrg@eterna.com.au>)	32- and 64-bit builds
NetBSD	VAX	7.1	2001-03-30, Tom I. Helbekkmo (<tih@kpnQwest.no>)	1.5
NeXTSTEP	x86	6.x	1998-03-01, David Wetzel (<dave@turbocat.de>)	bit rot suspected

OS	Processor	Version	Reported	Remarks
QNX 4 RTOS	x86	7.2	2001-12-10, Bernd Tegge (<tegge@repas-aeg.de>)	needs updates to semaphore code; see also doc/FAQ_QNX4
QNX RTOS v6	x86	7.2	2001-11-20, Igor Kovalenko (<Igor.Kovalenko@motorola.com>)	patches available in archives, but too late for 7.2
SunOS 4	Sparc	7.2	2001-12-04, Tatsuo Ishii (<t-ishii@sra.co.jp>)	
System V R4	m88k	6.2.1	1998-03-01, Doug Winterburn (<dlw@seavme.xroads.com>)	needs new TAS spinlock code
System V R4	MIPS	6.4	1998-10-28, Frank Ridderbusch (<ridderbusch.pad@sni.de>)	no recent reports
Ultrix	MIPS	7.1	2001-03-26	TAS spinlock code not detected
Ultrix	VAX	6.x	1998-03-01	

۲-۲ نصب بر روی ویندوز

با وجود اینکه PostgreSQLبرای سیستم‌عاملهای مانند یونیکس نوشته شده، کتابخانه کلاینت C (libpq) و پایانه تعاملی(psql) آن می‌توانند تحت ویندوز اجرا و ترجمه شوند. فایلهای Make آن که شامل منبع توزیع می‌شوند برای Visual C++ مایکروسافت نوشته شده‌است و ممکن است با دیگر سیستمها کار نکند. ممکن است در برخی موارد ترجمه کتابخانه‌ها به صورت دستی نیز امکان‌پذیر باشد.

توضیح: اگر شما از ویندوز ۹۸ یا جدیدتر از آنرا استفاده می‌کنید می‌توانید از PostgreSQL به همان روشی که در یونیکس بیان شد استفاده کنید.

برای ایجاد همه چیزهایی که در ویندوز داریم، باید به شاخه src منتقل شوید و در آنجا دستور زیر را وارد کنید:

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

این دستور باعث می‌شود که شما Visual C++ را در مسیر خود داشته باشید.

فایلهای زیر ساخته خواهند شد:

```
interfaces\libpq\Release\libpq.dll
```

کتابخانه‌های پویای قابل اتصال frontend .

```
interfaces\libpq\Release\libpqdll.lib
```

واردکردن کتابخانه‌ها به منظور اتصال برنامه شما به libpq.dll .

```
interfaces\libpq\Release\libpq.lib
```

نگارش ایستای کتابخانه‌های frontend

```
bin\psql\Release\psql.exe
```

پایانه تعاملی PostgreSQL .

تنها فایلی که حتما لازم است نصب شود، libpq.dll می‌باشد. این فایل در اکثر مواقع در شاخه WINNT\SYSTEM32 واقع می‌شود. یا در ویندوز 95/98/ME در WINDOWS\SYSTEM قرار می‌گیرد. اگر این فایل با استفاده از یک برنامه اجرایی نصب شده باشد، باید امتحان‌کننده نگارش که در منابع VERSIONINFO که در آن فایل وجود دارد، نیز نصب گردد تا مطمئن شوید که نگارش جدیدتری از کتابخانه در حال نوشته شدن نباشد.

اگر شما در نظر دارید که توسط libpq فرآیند توسعه را بر روی این ماشین انجام دهید، باید دو زیرشاخه src\include و src\interfaces\libpq را به درخت منبع در تنظیمات مترجم خود اضافه کنید.

به منظور استفاده از کتابخانه‌ها باید فایل libpqdll.lib را به پروژه خود اضافه کنید. (در Visual C++ فقط کلیک راست کرده و آن‌را اضافه کنید.)

۲-۳. محیط زمان اجرای سرور

این فصل پیرامون به چگونگی نصب و اجرای پایگاه داده سرور و تاثیر متقابل آن با سیستم عامل، می‌پردازد.

۲-۳-۱ حسابهای کاربر در PostgreSQL

توصیه می‌شود که همانند دیگر سرورهایی که با جهان خارج در ارتباط هستند، حساب کاربری جداگانه‌ای برای PostgreSQL در نظر گرفته شود. این حساب کاربری باید شامل داده‌هایی که توسط سرور مدیریت می‌شوند باشد، و نباید با دیگر کاربران به اشتراک گذاشته شود. توصیه نمی‌شود که دارایهای اجرائی خود را در این حساب کاربری ذخیره کنید زیرا سیستمهای مصالحه شده می‌توانند دودوئیهای آنها را گسترش دهند.

به منظور اضافه کردن یک حساب کاربری یونیکس به سیستمتان، از دستور useradd یا adduser استفاده کنید. نام کاربری postgres معمولا استفاده می‌شود ولی معنی و مفهوم خاصی ندارد.

۲-۳-۲ ایجاد یک پایگاه‌داده خوشه‌ای

قبل از انجام هر کاری شما باید در مرحله اول فضایی برای ذخیره پایگاه‌داده بر روی دیسک در نظر بگیرید. که ما به آن databases cluster می‌گوئیم. یک پایگاه‌داده خوشه‌ای مجموعه‌ای از پایگاه‌داده‌هایی است که توسط یک سرور پایگاه‌داده اجرا شده و قابل دسترس می‌باشند. پس از مرحله اول، یک پایگاه داده خوشه‌ای شامل یک پایگاه‌داده به نام template1 خواهد بود. نام پیشنهاد شده در آینده برای پایگاه‌داده‌های ایجاد شده به‌کار خواهد رفت که برای کارهای حقیقی نباید از آن استفاده شود.

در اصطلاحات فایل سیستمی، یک پایگاه‌داده خوشه‌ای یک دایرکتوری جداگانه در زیر داده‌های ذخیره شده می‌باشد. که ما آنرا data directory یا data area می‌باشد. انتخاب مکان ذخیره داده‌ها کاملا به خود شما بستگی دارد. مکان پیشفرضی برای آن در نظر گرفته نشده‌است، گرچه مکانهایی مانند /usr/local/pgsql/data یا /var/lib/pgsql/ بسیار مناسب و شهور می‌باشند. برای مرحله ابتدائی

ایجاد پایگاه‌داده خوشه‌ای از دستور initdb که همراه PostgreSQL نصب شده‌است استفاده کنید. مکان مطلوب فایل سیستم برای پایگاه‌داده سیستم شما با استفاده از گزینه D– استفاده می‌شودو به عنوان مثال:

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

توجه داشته باشید که شما باید این دستور را پس از وارد شدن به حساب کاربری PstgreSQL که در قسمت قبلی توضیح دا ده شد، استفاده کنید.

توجه: شما می‌توانید به جای گزینه D– ، متغییر محیط PGDATA را ست کنید.

اگر دایرکتوری که شما مشخص کرده‌اید، وجود نداشته باشد Initdb آنرا ایجاد می‌کند. احتمال دارد که اجازه انجام این کار را نداشته باشد (اگر شما توصیه ما را انجام داده باشید و یک حساب غیرممتاز ایجاد کرده باشید) در اینصورت شما باید با استفاده از حساب کاربری ریشه دایرکتوری را ایجاد نمائید و صاحب آنرا کاربر PostgreSQL قرار دهید. مراحل انجام کار در زیر آمده‌است:

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

چنانچه به نظر آید که دایرکتوری قبلا مقداردهی اولیه شده باشد initdb آنرا اجرا نمی‌کند. زیرا دایرکتوری داده شامل تمامی داده‌های ذخیره شده در پایگاه‌داده می‌باشد و حتما باید از دسترس افراد غیر مجاز در امان باشد. بنابراین initdb مجوز تمامی افراد به جز کاربر PostgreSQL، را لغو می‌کند.

به هر حال، زمانی که محتویات دایرکتوری ایمن می‌باشد، تنظیمات پیش‌فرض احراز هویت کلاینت به کاربران محلی اجازه اتصال به پایگاه‌داده را داده و همچنین این امکان را می‌دهد که یک کاربر ممتاز شوند. اگر شما به دیگر کاربران محلی اعتماد ندارید، پیشنهاد می‌شود که از گزینه W- یا pwprompt-- در initdb استفاده کنید که امکان قرار دادن رمزعبور برای کاربران ممتاز را می‌دهد. پس از آن initdb فایل pg_hda.conf را برای استفاده از md5 یا رمزعبور در اولین باری که سرور شروع به کار می‌کند به جای احرازهویت به‌کار می‌برد.

همچنین initdb منطقه پیش‌فرض را برای پایگاه‌داده خوشه‌ای مقداردهی اولیه می‌کند. معمولا فقط تنظیمات محلی در محیط را می‌گیرد و آنها بر روی پایگاه‌داده اعمال می‌کند. تعریف تنظیمات محلی متفاوت امکان‌پذیر است. یکی از اتفاقات غیرمنتظره‌ای که ممکن است در هنگام اجرای initdb بروز کند، دریافت پیامی مشابه زیر است:

```
The database cluster will be initialized with locale de_DE.
This locale setting will prevent the use of indexes for pattern matching
operations. If that is a concern, rerun initdb with the collation order
set to "C". For more information see the Administrator's Guide.
```

این پیغام به شما هشدار می‌دهد که منطقه انتخاب شده جاری ممکن است باعث شود شاخص‌ها به ترتیبی مرتب شوند که LIKE و جستجوهای اصطلاحات باقاعده نتوانند از آنها استفاده کنند. اگر شما می‌خواهید در چنین جستجوهایی بازده بالایی داشته باشید، باید منطقه جاری خود را با C فعال کنید و دوباره initdb را اجرا کنید. با اجرای دستور initdb --lc-collate=C این کار امکان‌پذیر است. ترتیب مرتب شدن در مدتی که پایگاه‌داده خوشه‌ای توسط initdb فعال می‌شوند، تعیین می‌گردد و امکان تغییر آن در آینده وجود ندارد پس ساختن این مورد در اولین بار بسیار مهم می‌باشد.

۲-۳-۳ آغاز بکار یک پایگاه داده سرور

پیش از اینکه کسی بتواند به پایگاه داده دسترسی پیدا کند شما باید پایگاه داده سرور که به آن postmaster گویند را راه‌اندازی کنید. بدین منظور postmaster باید بداند داده‌هایی را که قرار است استفاده کند را از کجا باید پیدا کند. این کار توسط گزینه D– انجام می‌شود بنابراین ساده‌ترین راه برای راه‌اندازی سرور اجرای خط زیر می‌باشد:

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

این کار باعث می‌شود که سرور در foreground اجرا شود. هنگامی که وارد حسابهای کاربری PostgreSQL می‌شوید این اتفاق می‌افتد. بدون D– سرور از متغییر محیط PGDATA به عنوان دایرکتوری داده استفاده خواهد کرد و اگر هیچ‌یک از این دو موفق نشدند، به شکست برخورد می‌کنیم.

برای اجرای postmaster در background معمولا خط زیر اجرا خواهد شد:

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

در اینجا نیز گزینه D– همان معنی که در قبل آورده شده‌است را دارد. Pg_ctl همچنین قادر است سرور را متوقف سازد.

معمولا شما می‌خواهید که سرور در هنگام بوت شدن اجرا شود، اسکریپتهای متعددی در سیستم‌عاملهای مختلف در مسیر contrib/startscripts/ وجود دارد که باید مجوزهای دسترسی کاربر ریشه به آن داده شود.

سیستمهای مختلف روشهای مختلفی برای اجرای سرور در زمان بالا آمدن سیستم دارند مثلا برخی از سیستمها دارای فایلی در مسیر /etc/rc.local یا /etc/rc.d/rc.local دارند و بعضی دیگر از شاخه rc.d استفاده می‌کنند. هرآنچه که شما انجام می‌دهید، سرور باید توسط حساب کاربری PostgreSQL انجام شود، نه با حساب کاربری ریشه یا دیگر حسابها. بنابراین ممکن است شما نیاز داشته باشید که از دستور su -c '...' postgres استفاده نمایید به عنوان مثال دستور زیر را اجرا کنید:

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

در ادامه به معرفی این فایلهای ویژه در سیستم‌عاملهای مختلف می‌پردازیم:

- در FreeBSD این فایل در مسیر contrib/start-scripts/freebsd قرار دارد.
- در OpenBSD خطوط زیر را به فایل /etc/rc.local اضافه کنید:

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

- در سیستمهای لینوکس خطوط زیر را اضافه نمائید:

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

to `/etc/rc.d/rc.local` or look at the file `contrib/start-scripts/linux` in the PostgreSQL source distribution.

- در NetBSD بسته به اینکه از اسکریپتهای شروع FreeBSD یا لینوکس استفاده می‌شود با توجه به اولویت آنها تعیین می‌شود.
- در سولاریس باید فایلی با نام /etc/init.d/postgresql ایجاد شود که شامل خط زیر می‌باشد:

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

- و سپس یک لینک در /etc/rc3.d as S99postgresql با آن ایجاد کنید.

۱. خطاهای شروع به کار سرور

دلایل مختلفی برای شکست در شروع به کار postmaster وجود دارد که برای پی بردن به آن باید فایل log آن را چک کنید. برخی از این پیغامهای خطا علت کاملا بیان شده‌است مانند:

```
FATAL: StreamServerPort: bind() failed: Address already in use
Is another postmaster already running on that port?
```

همان‌طور که در زیر می‌بینید، اگر شما postmaster را در یکی از شماره پورتهای رزرو شده اجرا کنید پیغام زیر ظاهر می‌شود:

```
$ postmaster -i -p 666
FATAL: StreamServerPort: bind() failed: Permission denied
Is another postmaster already running on that port?
```

چند نمونه از پیغامهای خطا در زیر آورده شده‌است:

```
IpcMemoryCreate: shmget(key=5440001, size=83918612, argument
FATAL 1: ShmemCreate: cannot create region
```

که به این معنی است که اندازه هسته شما محدود شده‌است و حجم حافظه مشترک از فضای بافر کمتر می‌باشد.

```
IpcSemaphoreCreate: semget(key=5440026, num=16, 01600) failed: No space left
on device
```

این پیام بدان معنی نیست که فضای دیسک شما پر شده‌است بلکه بدین معنی است که هسته شما توسط تعداد سمافورهای V محدود شده‌اند و تعداد آنها کمتر از آنی است که PostgreSQL می‌خواهد ایجاد کند.

۲. مشکلات برقراری ارتباط در کلاینت

با وجود اینکه بسیاری از مشکلات در سمت سرور اتفاق می‌افتد و علت بروز آنها در وابسته به نرم‌افزارهای کاربردی می‌باشد، ولی برخی از آنها ممکن است مرتبط با چگونگی شروع به کار سرور باشد.

```
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?
```

این پیام یکی از معمول‌ترین خطاهایی است که در هنگام پیدا نشدن سرور برای برقراری ارتباط بروز می‌کند. یکی از خطاهای معمول، فراموش کردن قرار دادن گزینه i– می‌باشد که به postmaster امکان پذیرفتن ارتباطات TCP/IP را می‌دهد.

```
psql: could not connect to server: Connection refused
Is the server running locally and accepting
connections on Unix domain socket "/tmp/.s.PGSQL.5432"?
```

خط آخر برای برقراری ارتباط کلاینت با مکان مناسب، کاربرد دارد.

۲-۳-۴ تنظیمات زمان اجرا

تنظیمات بسیاری وجود دارد که بر روی وضعیت سیستم پایگاه داده تاثیر دارد. که در این بخش به برخی از آنها می‌پردازیم. تمامی پارامترها case-insensitive هستند و از یکی از چهار نوع بولین، عدد صحیح، ممیز شناور و رشته‌ای را دارند. ارزشهای بولین شامل ON, OFF, TRUE, FALSE, YES, NO, 1, یا با پیشوندهای غیر مبهم آنها می‌باشد.

یکی از روشهای فعال کردن این گزینه‌ها ایجاد تغییر در فایل postgresql.conf در دایرکتوری داده می‌باشد (که یک فایل پیش‌فرض در آنجا نصب شده‌است). به عنوان مثال این فایل ممکن است به صورت زیر باشد:

```
# This is a comment
log_connections = yes
syslog = 2
search_path = '$user, public'
```

همانطور که می‌بینید در هر خط یک گزینه قرار گرفته‌است. علامت مساوی در بین نام و مقدار، دلخواه است. فضاهای خالی ارزش ندارند و خطوط خالی حذف می‌شوند. برای قرار دادن توضیحات می‌توانید از علامت "#" استفاده کنید. مقدارهای پارامترها و یا اعداد باید در بین ' ' قرار داده شود.

فایل پیکربندی هنگامی که postmaster یک سیگنال SIGHUP دریافت می‌کند، قرائت می‌شود. Postmaster این سیگنال را به تمامی فرآیندهای backend در حال اجرا منتقل می‌کند و به قسمتهای مختلف مقدار جدید را اعلام می‌کند. البته شما می‌توانید این سیگنال را به صورت مستقیم به فرآیندهای backend ارسال نمائید.

روش دوم برای پیکربندی این پارامترها اجرای دستور زیر می‌باشد:

```
postmaster -c log_connections=yes -c syslog=2
```

اجرای دستور بالا از ایجاد هر ناسازگاری در postgresql.conf جلوگیری می‌کند.

در برخی از موارد اجرای خط دستور فقط برای قسمتهای خاص backend کافی می‌باشد. برای تحقق این منظور از متغییر محیط PGOPTIONS در طرف کلاینت استفاده کنید.

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

با دستور SET می‌توانید برخی از قسمتهای منحصر به فرد SQL را تغییر دهید، به عنوان مثال:

```
=> SET ENABLE_SEQSCAN TO OFF;
```

هر زمان که یک session شروع به کار می‌کند، تنظیمات پیش‌فرض برای کاربران و پایگاه‌داده بارگذاری می‌شود. دستوراتی چون ALTER DATABASE وALTER USER برای ایجاد تغییراتی در این تنظیمات بکار می‌رد.

۲-۳-۵ مدیریت منابع هسته

با نصب کامل PostgreSQL می‌توانید بسیاری از محدودیت منابع سیستم‌عامل را از بین برد. در برخی از سیستمها، پیش‌فرضهای کارخانه بسیار پایین می‌باشد و نیازی به نصب کامل نمی‌باشد.

۲-۳-۶ پایان کار سرور

راه‌های بسیاری برای خاتمه دادن به کار سرور پایگاه داده می‌باشد. شما می‌توانید نوع خاموش کردن سرور را با ارسال سیگنالهای مختلف به فرآیندهای سرور کنترل کرد.

۲-۴ کاربران و مزایای پایگاه داده

هر پایگاه داده خوشهای شامل یکسری از کاربران پایگاه داده می‌باشد. این کاربران با کاربرانی که توسط سیستم‌عامل سرور مدیریت می‌شوند تفاوت دارند. کاربران، اشیاء پایگاه داده را به مالکیت خود درآورده و می‌توانند مزایایی را به دیگر کاربران برای دسترسی به آن اشیاء اختصاص دهند.

این فصل به معرفی مزایای سیستم و چگونگی ایجاد و مدیریت کابران می‌پردازد.

۲-۴-۱ کاربران پایگاه داده

کاربران پایگاه داده کاملا از کاربران سیستم عامل متفاوت هستند. حسابهای کاربری پایگاه داده درسراسر پایگاه داده خوشه‌ای در نظر گرفته می‌شوند. برای ایجاد یک کاربر از دستور زیر استفاده نمائید:

```
CREATE USER name
```

قوانین نامگذاری SQL در اینجا صادق است. از کراکترهای ویژه‌ای چون ″ استفاده نمی‌شود. به منظور حذف یک کاربر از دستور زیر استفاده کنید:

```
DROP USER name
```

به منظور تسهیل در کار برنامه‌های createuser و dropuser به جای دستورهای SQL مربوطه بوجود آمده‌اند که از خط فرمان پوسته فراخوانی می‌شوند:

```
createuser name
dropuser name
```

برای ایجاد ویژگی راه‌اندازی خودکار در سیستم یک کاربر از پیش تعیین شده و ابتدائی در سیستم تعریف می‌شود. این کاربر همواره شماره شناسه ۱ را به خود اختصاص می‌دهد و به طور پیش فرض همان نامی را خواهد داشت که کاربران ابتدائی سیستم‌عامل در پایگاه داده خوشه‌ای به خود اختصاص می‌دهند.

فقط یک شناسه کاربری برای یک ارتباط با سرور پایگاه داده فعال می‌باشد.

۲-۴-۲ خصوصیات کاربران

یک کاربر پایگاه‌داده دارای خصوصیتهایی می‌باشد که تعیین‌کننده مزایا و فعل و انفعالات آن با سیستم احرازهویت کلاینت می‌باشد.

۲-۴-۳ کاربر ممتاز

یک کاربر ممتاز پایگاه دارای تمامی بازرسیهای مجوزها را پشت‌سر گذاشته‌است. تنها یک کاربر ممتاز می‌تواند کاربر جدید ایجاد کند. برای ایجاد یک کاربر ممتاز از دستور CREATE USER name CREATEUSER استفاده کنید.

۲-۴-۴ ایجاد پایگاه‌داده

این نوع کاربر باید مجوز ایجاد پایگاه‌داده را داشته باشد (به استثناء کاربران ممتاز که ابتدا باید مراحل بازرسی مجوزها را پشت‌سر گذارد). برای ایجاد این کاربر از دستور CREATE USER name CREATEDB استفاده کنید.

۲-۴-۵ رمز عبور

یک رمز عبور هنگامی معنی‌دار می‌شود که پس از برقراری ارتباط کاربر با پایگاه‌داده، روشهای احراز هویت از وی درخواست یک رمز عبور کند. روشهای احراز هویت password, md5 و crypt رمزعبور را قابل استفاده می‌سازد. پایگاه‌داده رمزهای عبور از رمزهای عبور سیستم‌عامل جدا هستند. توسط دستور CREATE USER name PASSWORD 'string' می‌توانید رمز عبور مربوط به کاربر را ایجاد نمائید.

خصوصیات مربوط به یک کاربر پس از ایجاد می‌تواند توسط ALTER USER تغییر داده شود. همچنین کاربران می‌توانند بسیاری از پیش‌فرضها را برای تنظیمات پیکربندی زمان اجرا قرار دهند. به عنوان مثال اگر به دلایلی بخواهید index scans را غیرفعال سازید، هر زمان که متصل می‌شوید می‌توانید از خط زیر استفاده کنید.

```
ALTER USER myname SET enable_indexscan TO off;
```

۲-۴-۶ گروه ها

در یونیکس گروه بندی کاربران باعث می‌شود که مدیریت مزایا آسان‌تر شود. می‌توان بـه تمـامی اعضا یک گروه یکسری مزایا را اختصاص داد یا از آنها گرفت. برای ایجاد یک گروه دستور زیر را اجـرا نمائید:

```
CREATE GROUP name
```

به منظور اضافه یا حذف یک کاربر به یک گروه از دستور زیر استفاده نمائید:

```
ALTER GROUP name ADD USER uname1, ...
ALTER GROUP name DROP USER uname1, ...
```

۲-۴-۷ مزایا (امکانات)

هنگامی که یک شئی پایگاه‌داده ایجاد می‌شود، برای آن مالـک اختصـاص داده مـی‌شـود. مالـک کاربری است که فرآیند ایجاد آن‌را بر عهده می‌گیرد. برای ایجاد تغییر در مالک یک جدول، شـاخص، ترتیب یا دید از دستور ALTER TABLE استفاده کنید. در حالت پیش‌فرض فقط مالک می‌تواند بـر روی این اشیاء، اعمال مختلف را انجام دهد. به منظور ایجاد مجوز دسترسی برای دیگر کـاربران بایـد مزایا (امکانات) واگذار گردد.

مزایـای (امکانـات) مختلفـی وجـود دارنـد کـه عبارتنـد از: SELECT, INSERT, UPDATE, DELETE, RULE, REFERENCES, TRIGGER, CREATE, TEMPORARY, EXECUTE, USAGE و غیره. امکان ایجاد تغییر و حذف یک شئی فقط مربوط به مالک آن است.

کاربر ویژه‌ای به نام PUBLIC برای اعطای مزایا به کاربران به‌کار برده می‌شود. بـا نوشـتن کلمـه ALL در مکانی که باید مزایای مورد نظر نوشته شود، باعث می‌شود تمامی مزایا اعطا گردد.

برای لغو یک مزیت از دستور زیر استفاده کنید:

```
REVOKE ALL ON accounts FROM PUBLIC;
```

عملیات ویژه مربوط به جداول مانند: DROP, GRANT, REVOKE فقـط بـه مالـک اختصـاص یافته و واگذار و لغو نمی‌شود. ولی در عملیات معمول مربوط به مالک، امکان تغییر و لغو وجـود دارد. به عنوان مثال وی می‌تواند یک جدول را برای خود و دیگران فقط خواندنی کند.

۲-۵ مدیریت پایگاه داده

۲-۵-۱ مرور

یک پایگاه داده شامل مجموعه‌ای اشیاء نامدار SQL است. عموما هر شئی از پایگاه‌داده (مانند: جداول و توابع و...) فقط مربوط به یک پایگاه‌داده می‌باشد. (فقط چند کاتالوگ سیستم وجود دارد که به نصب مربوط می‌باشد و برای تمام پایگاه‌داده‌های نصب شده قابل دسترسی می‌باشد که pg_database از این نمونه می‌باشد. به طور کلی هر پایگاه‌داده شامل مجوعه‌ای از الگوها می‌باشد و هر الگو شامل جداول، توابع و غیره می‌باشد. بنابراین سلسله مراتب کامل آن به عبارت است از: سرور، پایگاه‌داده، الگوها، جداول (یا دیگر اجزاء تشکیل دهنده پایگاه‌داده.)

هر برنامه کاربردی که به سرور پایگاه‌داده متصل می‌شود، نام پایگاه‌داده مورد نظر خود را مشخص می‌کند. در هر اتصال امکان دسترسی به بیش از یک پایگاه‌داده وجود ندارد. (ولی در تعداد ارتباطاتهایی که یک نرم‌افزار کاربردی در همان پایگاه داده یا دیگر پایگاه‌داده‌ها برقرار می‌کند محدودیتی وجود ندارد.) امکان دسترسی به بیش از یک الگو از طریق همان اتصال وجود دارد. الگوها ساختار کاملا منطقی دارند و توسط مزایای سیستم مدیریت می‌شوند. پایگاه‌داده‌ها به صورت فیزیکی از یکدیگر جدا هستند و کنترل دسترسی توسط سطح ارتباطها مدیریت می‌شود. اگر یک سرور پایگاه‌داده دربردارنده کاربران یا پروژه‌هایی می‌باشد که باید از یکدیگر جدا باشند یا اینکه قسمتهای مختلف از یکدیگر جدا باشند، توصیه می‌شود که آنها را در پایگاه‌داده‌های جداگانه قرار دهید. اگر کاربران و پروژه‌ها با یکدیگر مرتبط باشند یا نیاز باشد که از منابع یکدیگر استفاده کنند حتما باید آنها را در یک پایگاه‌داده ولی در الگوهای متفاوت قرار دهید.

۲-۵-۲ ایجاد یک پایگاه داده

به منظور ایجاد یک پایگاه‌داده، سرور PostgreSQL باید راه‌اندازی شده و در حال اجرا باشد. برای ایجاد یک پایگاه‌داده از دستور زیر استفاده می‌کنیم:

```
CREATE DATABASE name
```

قوانین نام‌گذاری در اینجا همانند شناسه‌های SQL می‌باشد. کاربر جاری به صورت پیش‌فرض به عنوان مالک پایگاه‌داده جدید قرار داده می‌شود. مالک پایگاه‌داده در آینده می‌تواند آنرا حذف کند که در اینصورت تمامی اشیاء مربوط به آن حذف می‌شوند.

به منظور ایجاد اولین پایگاه‌داده باید دستور CREATE DATABASE را اجرا کنیم و به سرور پایگاه‌داده متصل شویم، سوالی که در اینجا مطرح می‌شود این است که: اولین پایگاه‌داده چگونه ایجاد می‌شود. اولین پایگاه‌داده توسط دستور initdb در هنگام مقداردهی اولیه فضای ذخیره‌سازی داده ایجاد می‌شود که این پایگاه‌داده tempate1 نامیده می‌شود. بنابراین جهت ایجاد اولین پایگاه‌داده حقیقی باید به template1 متصل شویم.

ایجاد هر تغییر در template1 باعث ایجاد تغییر در پایگاه‌داده‌های بعدی می‌شود. این بدان معناست که نباید از پایگاه‌داده template برای کارهای حقیقی استفاده نمائید.

برای ایجاد سهولت در کار برنامه‌ای وجود دارد که برای ایجاد پایگاه‌داده جدید از پوسته آنرا اجرا کنید:

```
createdb dbname
```

همانطور که در بالا آمده‌است این دستور به منظور ایجاد پایگاه‌داده جدید به پایگاه‌داده template1 متصل می‌شود و دستور CREATE DATABASE را اجرا می‌کند. توجه داشته باشید که اگر برای createdb آرگومانی قرار ندهید، پایگاه‌داده را با نام کاربر جاری ایجاد می‌کند.

در برخی موارد شما می‌خواهید برای شخص دیگری پایگاه‌داده ایجاد کنید که در اینصورت از دستورهای زیر استفاده کنید:

در محیط SQL ،

```
CREATE DATABASE dbname OWNER username;
```

یا از دستور زیر ازطریق پوسته استفاده کنید:

```
createdb -O username dbname
```

برای این کار شما باید یک کاربر ویژه باشید و مجوز ایجاد پایگاه‌داده برای دیگران را داشته باشید.

۲-۵-۳ قالبهای پایگاه داده

CREATE DATABASE برای ایجاد پایگاه‌داده‌های جدید، پایگاه‌داده‌های موجود را کپی می‌کند. که در حالت استاندارد پایگاه‌داده template1 را کپی می‌کند. اگر شما اشیایی را به template1 افزوده باشید، آنها هم به پایگاه‌داده‌های بعدی افزوده می‌شود.

سیستم پایگاه‌داده استاندارد دیگری وجود دارد که template0 نام دارد. این پایگاه‌داده شامل همان داده‌هایی است که در template1 وجود دارد و اشیاء استاندارد بستگی به نسخه PostgreSQL دارد. پس از اجرای initdb نباید در template0 تغییر ایجاد شود.

به منظور ایجاد پایگاه‌داده‌ای از template0 کپی می‌شود از دستورهای زیر استفاده کنید:

در محیط SQL ،

```
CREATE DATABASE dbname TEMPLATE template0;
```

یا از دستور زیر ازطریق پوسته استفاده کنید:

```
createdb -T template0 dbname
```

۲-۵-۴ تنظیمات پایگاه داده

همانطور که می‌دانید تعداد زیادی متغییر برای تنظیمات زمان اجرا در سرور PostgreSQL وجود دارد. شما می‌توانید یکسری از تنظیمات پیش‌فرض را برای بسیاری از آنها قرار دهید.

به عنوان مثال اگر بنا به دلایلی شما بخواهید بهینه‌ساز GEQO را برای پایگاه‌داده مورد نظر غیر فعال سازید، به صورت معمول باید برای همه پایگاه‌داده‌ها غیرفعال سازید. برای حصول اطمینان از اینکه تمام ارتباطهای کلاینت دقیق و صحیح می‌باشد به ;SET geqo TO off توجه کنید. به منظور پیش‌فرض قرار دادن این تنظیمات دستور زیر را اجرا کنید:

```
ALTER DATABASE mydb SET geqo TO off;
```

این دستور تنظیمات مورد نظر را ذخیره کرده و در ارتباطهای پایگاه‌داده‌های بعدی ظاهر می‌شود و ;SET geqo TO off قبل شروع هر نشست، فراخوانی می‌شود. توجه داشته باشید که کاربران در طول هر نشست می‌توانند این تنظیمات را تغییر دهند و فقط این مقادیر به عنوان پیش‌فرض در نظر

گرفته می‌شوند. جهت ایجاد تغییر در این تنظیمات از دستور ALTER DATABASE dbname RESET varname; استفاده کنید.

۲-۵-۵ موقعیتهای جایگزینی

این امکان وجود دارد که یک پایگاه‌داده را در مکانی غیر از موقعیت پیش‌فرض در هنگام نصب، ایجاد کنیم. به خاطر داشته باشید که دسترسی به تمامی پایگاه‌داده‌ها از طریق سرور پایگاه‌داده امکان‌پذیر می‌باشد بنابراین هر موقعیت ویژه باید توسط سرور قابل دسترسی باشد.

موقعیتهای جایگزینی توسط متغیرهای محیط ارجاع داده می‌شوند که مسیر قطعی را برای فضای ذخیره‌سازی مشخص می‌کند. متغیرهای محیط باید در محیط سرور نمایش داده شوند بنابراین باید آنها را قبل از شروع کار سرور تعریف کنیم. هر متغییر محیط معتبری می‌تواند به عنوان متغیر محیط در نظر گرفته شود. با وجود اینکه این متغیرها همراه با پیشوند PGDATA به کار برده می‌شوند، توصیه می‌شود که از نامهایی که با دیگر متغیرها اشتباه می‌شود، دوری کنید.

به منظور ایجاد متغیرها در محیط سرور، نخست باید سرور را خاموش کرده، متغیر را تعریف کرده، منطقه داده را مقدار دهی اولیه کنید و در نهایت سرور را راه‌اندازی دوباره کنید. جهت ایجاد متغیر محیط، خط زیر را بنویسید:

```
PGDATA2=/home/postgres/data
export PGDATA2
```

در پوسته Bourne، یا

```
setenv PGDATA2 /home/postgres/data
```

در csh یا tcsh.

به منظور ایجاد فضای ذخیره‌سازی داده در PGDATA2، اطمینان حاصل کنید که مسیر مربوطه که در اینجا home/postgres/ می‌باشد، وجود دارد و توسط حساب کاربری که سرور تحت آن اجرا می‌شود، قابل نوشتن باشد. سپس خط زیر را بنویسید:

```
initlocation PGDATA2
```

حال می‌توانید سرور را دوباره راه‌اندازی کنید.

برای ایجاد پایگاه‌داده در موقعیت جدید، از دستور زیر استفاده کنید:

```
CREATE DATABASE name WITH LOCATION = 'location'
```

جهت استفاده از دستور createdb از گزینه D– استفاده کنید.

پایگاه‌داده‌ای که در موقعیتهای جایگزینی ایجاد می‌شوند همانند دیگر پایگاه‌داده‌ها قابل دسترسی و حذف می‌باشد.

۲-۵-۶ حذف یک پایگاه داده

عمل حذف پایگاهcداده توسط دستور زیر انجام می‌شود:

```
DROP DATABASE name
```

فقط مالک پایگاه‌داده یک کاربر ویژه می‌تواندااینکار را انجام دهد. با حذف یک پایگاه‌داده تمامی اشیاء مرتبط با آن حذف خواهد شد. عملیات حذف پایگاه‌داده نباید ناتمام بماند.

امکان استفاده از دستور DROP DATABASE در هنگامیکه به آن متصل هستید، وجود ندارد. اما امکان برقراری ارتباط با دیگر پایگاه‌داده‌ها و حتی template1 وجود دارد.

به منظور ایجاد سهولت در کار برنامه‌ای وجود دارد که برای حذف پایگاه‌داده‌ها از طریق پوسته به‌کار می‌رود:

```
dropdb dbname
```

۲-۶ احراز هویت کلاینت

هنگامی که یک نرم‌افزار کاربردی کلاینت به یک سرور پایگاه‌داده متصل می‌شود باید مشخص شود که کدام حساب کاربری PostgreSQL قصد ورود را دارد.

فرآیند احراز هویت توسط سرور پایگاه‌داده برای شناسایی کلاینت ایجاد می‌شود و در نهایت تصمیم می‌گیرد که به برنامه کاربردی کلاینت تحت نام کاربری مربوطه، اجازه ورود بدهد یا نه.

PostgreSQL دارای تعداد زیادی از روشهای احراز هویت می‌باشد. به طور منطقی حسابهای کاربری PostgreSQL از حسابهای کاربری سیستم‌عامل که سرور تحت آن اجرا می‌شوند، جدا می‌باشند. یک سرور که ارتباطات راه دور را قبول می‌کند، دارای بسیاری از کاربران است که دارای

حساب کاربری محلی نمی‌باشند، در چنین مواردی هیچ ارتباطی بین کاربران پایگاه‌داده و حسابهای کاربری سیستم‌عامل وجود نخواهد داشت.

۲-۶-۱ فایل pg_hba.conf

احراز هویت کلاینت توسط فایل pg_hba.conf که در دایرکتوری داده قرار دارد،کنترل می‌شود. فایل pg_hba.conf پیش‌فرض در هنگام مقداردهی اولیه به دایرکتوری داده توسط initdb انجام می‌شود، صورت می‌گیرد.

فرمت کلی این فایل شامل تعدادی رکورد می‌باشد که در هر خط فقط یک رکورد قرار می‌گیرد. خطوط خالی نادیده گرفته می‌شوند. هر متنی که پس از علامت # قرار می‌گیرد به عنوان توضیح در نظر گرفته می‌شود. هر رکورد شامل چند فیلد می‌باشد که توسط فاصله یا tab از یکدیگر جدا می‌شوند. اگر مقدار یک فیلد، quoted باشد، استفاده از فضای سفید جایز می‌باشد.

هر رکورد یک نوع ارتباط، یک رنج آدرس IP در کلاینت، یک پایگاه‌داده نام، یک حساب کاربری، یک روش احراز هویت که برای تنظیم پارامترهای ارتباطها را تعریف می‌کند.

اولین رکورد با نوع اتصال مشخص شده، آدرس کلاینت، پایگاه‌داده درخواست شده و حساب کاربری برای انجام احرازهویت به‌کار برده می‌شود. اگر هنگام عمل احرازهویت اگر یک رکورد انتخاب شود و عملیات با شکست روبرو شود، هیچ پشتیبان یا fall-through برای آن وجود ندارد و رکوردهای پس‌آیند در نظر گرفته نمی‌شوند. اگر هیچ رکوردی تنظیم نشود، دسترسی امکان پذیر نخواهد بود.

یک رکورد ممکن است یکی از سه قالب زیر را داشته باشد:

```
local database user authentication-method [authentication-option]

host database user IP-address IP-mask authentication-method
[authenticationoption]

hostssl database user IP-address IP-mask authentication-method
[authenticationoption]
```

که معنی فیلدهای آن به‌صورت زیر می‌باشد:

```
local
```

This record matches connection attempts using Unix domain sockets. Without a record of this

type, Unix-domain socket connections are disallowed

`host`

This record matches connection attempts using TCP/IP networks. Note that TCP/IP connections are disabled unless the server is started with the `-i` option or the `tcpip_socket postgresql.conf` configuration parameter is enabled.

`hostssl`

This record matches connection attempts using SSL over TCP/IP. `host` records will match either SSL or non-SSL connection attempts, but `hostssl` records require SSL connections.
To be able make use of this option the server must be built with SSL support enabled. Furthermore, SSL must be enabled by enabling the option `ssl` in `postgresql.conf` (see Section 3.4).

`database`

Specifies which databases this record matches. The value `all` specifies that it matches all databases. The value `sameuser` specifies that the record matches if the requested database has the same name as the requested user. The value `samegroup` specifies that the requested user must a member of the group with the same name as the requested database. Otherwise, this is the name of a specific PostgreSQL database. Multiple database names can be supplied by separating them with commas. A file containing database names can be specified by preceding the file name with `@`. The file must be in the same directory as `pg_hba.conf`.

`user`

Specifies which PostgreSQL users this record matches. The value `all` specifies that it matches all users. Otherwise, this is the name of a specific PostgreSQL user. Multiple user names can be supplied by separating them with commas. Group names can be specified by preceding the group name with `+`. A file containing user names can be specified by preceding the file name with `@`. The file must be in the same directory as `pg_hba.conf`.

`IP-address`

`IP-mask`

These two fields contain IP address/mask values in standard dotted decimal notation. (IP addresses can only be specified numerically, not as domain or host names.) Taken together they specify the client machine IP addresses that this record matches. The precise logic is that
`(`*`actual-IP-address`* `xor` *`IP-address-field`*`) and` *`IP-mask-field`*
must be zero for the record to match. (Of course IP addresses can be spoofed but this consideration is beyond the scope of PostgreSQL.)
These fields only apply to `host` and `hostssl` records.

`authentication-method`

Specifies the authentication method to use when connecting via this record. The possible choices are summarized here; details are in Section 6.2.

`trust`

The connection is allowed unconditionally. This method allows anyone that can connect to the PostgreSQL database to login as any PostgreSQL user they like, without the need for a password. See Section 6.2.1 for details.

`reject`

The connection is rejected unconditionally. This is useful for “filtering out” certain hosts from a group.

`md5`

Requires the client to supply an MD5 encrypted password for authentication. This is the only method that allows encrypted passwords to be stored in `pg_shadow`. See Section 6.2.2 for details.

`crypt`

Like `md5` method but uses older crypt encryption, which is needed for pre-7.2 clients. `md5` is preferred for 7.2 and later clients.

`password`

Same as "md5", but the password is sent in clear text over the network. This should not be used on untrusted networks. See Section 6.2.2 for details.

`krb4`

Kerberos V4 is used to authenticate the user. This is only available for TCP/IP connections.

`krb5`

Kerberos V5 is used to authenticate the user. This is only available for TCP/IP connections.

`ident`

Obtain the operating system user name of the client (for TCP/IP connections by contacting the ident server on the client, for local connections by getting it from the operating system) and check if the user is allowed to connect as the requested database user by consulting the map specified after the `ident` key word.

If you use the map `sameuser`, the user names are assumed to be identical. If not, the map name is looked up in the file `pg_ident.conf` in the same directory as `pg_hba.conf`.

The connection is accepted if that file contains an entry for this map name with the identsupplied user name and the requested PostgreSQL user name.

For local connections, this only works on machines that support Unix-domain socket credentials (currently Linux, FreeBSD, NetBSD, and BSD/OS).

See Section 6.2.4 below for details.

`pam`

`authentication-option`

The meaning of this optional field depends on the chosen authentication method and is described in the next section.

یک مثال از فایل pg_hba در زیر آورده شده‌است:

```
# Allow any user on the local system to connect to any database under
# any user name using Unix-domain sockets (the default for local
# connections).
#
# TYPE DATABASE USER IP-ADDRESS IP-MASK METHOD
local all all trust
```

```
# The same using local loopback TCP/IP connections.
#
# 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 "template1" as the same user name that ident reports for
# the connection (typically the Unix user name).
#
# TYPE DATABASE USER IP-ADDRESS IP-MASK METHOD
host template1 all 192.168.93.0 255.255.255.0 ident sameuser
# Allow a user from host 192.168.12.10 to connect to database
# "template1" if the user's password is correctly supplied.
#
# TYPE DATABASE USER IP-ADDRESS IP-MASK METHOD
host template1 all 192.168.12.10 255.255.255.255 md5
# In the absence of preceding "host" lines, these two lines will
# reject all connection from 192.168.54.1 (since that entry will be
# matched first), but allow Kerberos V connections from anywhere else
# on the Internet. The zero mask means that no bits of the host IP
# address are considered so it matches any host.
#
# TYPE DATABASE USER IP-ADDRESS IP-MASK METHOD
host all all 192.168.54.1 255.255.255.255 reject
host all all 0.0.0.0 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 IP-ADDRESS IP-MASK METHOD
host all all 192.168.0.0 255.255.0.0 ident 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 user name) except for administrators and
# members of group "support" who may connect to all databases. The file
# $PGDATA/admins contains a list of user names. Passwords are required in
# all cases.
#
# TYPE DATABASE USER IP-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, but not groups:
```

```
local db1,db2,@demodbs all md5
```

۲-۶-۲ روشهای احراز هویت

۱. احراز هویت مطمئن

از این روش وقتی استفاده می‌شود که PostgreSQL در نظر می‌گیرد که کسی که به سرور متصل می‌شود و تشخیص هویت داده می‌شود، اجازه دارد که همانطور که تعریف شده‌است به پایگاه‌داده دسترسی پیدا کند. این روش بیشتر هنگامی کاربرد دارد که در اتصالاتی که با postmaster برقرار می‌شود، سطح حفاظت بسیار بالایی وجود داشته باشد.

۲. احرازهویت با رمزعبور

روش احرازهویت بر پایه رمزعبور، شامل md5, crypt و password می‌باشد. این روش مشابه روش قبل عمل می‌کند به جز در مرحله‌ای که رمزعبور در طول ارتباط ارسال می‌شود. اگر شما نگران حمله‌های sniffing رمزعبور هستید از md5 استفاده کنید. اگر می‌خواهید pre-7.2 کلاینتها را پشتیبانی کنید از crypt استفاده کنید. استفاده از رمزعبورهای ساده در ارتباطهای اینترنتی توصیه نمی‌شود.

۳. احرازهویت Kerberos

این روش یک سیستم احرازهویت امن و استاندارد کارخانه می‌باشد و مناسب رایانه‌های توزیع‌شده در یک شبکه عمومی می‌باشد. توصیف کامل این روش در این کتاب جایز نمی‌باشد. در کل این روش بسیار پیچیده و کاملا قدرتمند است. توریعهای مختلفی از Kerberos وجود دارد و برای کسب اطلاعات بیشتر، می‌توانید به دو آدرس http://www.nrl.navy.mil/CCS/people/kenh/kerberos-faq.html و ftp://athena-dist.mit.edu مراجعه کنید.

برای تولید فایل keytab مانند مثال زیر عمل می‌کنیم:

```
kadmin% ank -randkey postgres/server.my.domain.org
kadmin% ktadd -k krb5.keytab postgres/server.my.domain.org
```

۴. احرازهویت بر پایه یکسان‌سازی

این روش احرازهویت با سرکشی به حسابهای کاربری کلاینتها و تعیین حسابهای کاربری مجاز به وسیله mapکردن فایلهایی که فهرست کاربران مجاز را در بر دارند، انجام می‌پذیرند. مرحله تعیین کاربران مجاز کلاینتها یک نکته بسیار حیاتی امنیتی می‌باشد و نحوه عملکرد آن کاملا وابسته به نوع ارتباط می‌باشد.

۵. احرازهویت بر پایه یکسان‌سازی در TCP IP

پروتکل شناسایی مجازا مشابه سیستم‌عامل یونیکس می‌باشد با سرور ident که به پورت پیش‌فرض ۱۱۳ گوش می‌کند، راه‌اندازی می‌شود. تابع اصلی سرور ident بر پایه پاسخ‌گویی به سوال " کدام کاربر در ابتدا برای ارتباطهایی که از پورت X خارج می‌شوند و به پورت Y متصل می‌شوند؟" شکل گرفته‌است. در اینجا PostgreSQL باید دو پورت X و Y را بشناسد و ارتباط فیزیکی بین دو نقطه نیز برقرار شده باشد. در این زمان سرور ident در سمت میزبان ارتباط می‌تواند به صورت فرضی کاربران سیستم‌عامل را برای هر ارتباط مشابه تعیین کند.

۶. احرازهویت بر پایه یکسان‌سازی در Socket های محلی

احرازهویت بر پایه یکسان‌سازی می‌تواند در ارتباطهای محلی نیز به‌کار برده شوند. در این روش هیچ خطر امنیتی ما را تهدید نمی‌کند و مسلما این روش بسیار مناسبی برای اینگونه ارتباطهای محلی در این سیستمها می‌باشد.

۷. نقشه‌های یکسان‌سازی

هنگامی که از روش احرازهویت برپایه ident استفاده می‌کنیم پس از تعیین نام کاربران سیستم‌عامل که در ارتباط مربوطه مقداردهی اولیه شده‌اند، PostgreSQL چک می‌کند که آیا آن کاربر اجازه اتصال به عنوان کاربر پایگاه‌داده را دارد. این کار با map شدن مقدارهایی که در لغات کلیدی ident در فایل pg_hba.conf می‌آید، انجام خواهد شد. یک نقشه از پیش تعریف شده به نام sameuser وجود دارد که به کاربران محلی اجازه متصل شدن به عنوان یک کاربر پایگاه‌داده با همان نام را می‌دهد. دیگر نقشه‌ها باید به صورت دستی ایجاد گردند.

نقشه‌های ident به جز sameuser در فایل pg_ident.conf در دایرکتوری داده قرار دارند که شامل خطوط با قالب کلی زیر می‌باشند:

```
map-name ident-username database-username
```

با توضیحات و فضاهای خالی مانند قبل برخورد می‌شود. Map-name شامل نام قراردادی نقشه ایجاد شده در pg_hba.conf می‌باشد. دو فیلد دیگر تعیین می‌کنند که کدام کاربر سیستم‌عامل اجازه دارد به عنوان کدام کاربر پایگاه‌داده ارتباط را برقرار کند.

مثالی از فایل pg_ident در زیر آمده است:

```
# MAPNAME IDENT-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
```

۸. احرازهویت به روش PAM

در این روش عملیات کاملا مشابه روش رمزعبور می‌باشد به جز اینکه از مکانیزم PAM (Pluggable Authentication Modules) برای تشخیص هویت استفاده می‌کند.

۲-۶-۳ مشکلات احزار هویت

اینگونه خطاها و مشکلات خود را با ارسال پیامی مشابه زیر نشان می‌دهند:

```
No pg_hba.conf entry for host 123.123.123.123, user andym, database testdb
```

در این حالت شما ارتباط را با سرور برقرار کرده‌اید اما سرور نمی‌خواهد به آن پاسخ بگوید. همانگونه که در متن پیام آمده است، سرور برقراری ارتباط را رد کرده‌است زیرا وی هیچ داده‌ای برای احرازهویت در تنظیمات فایل pg_hba.conf خود پیدا نکرده‌است.

```
Password authentication failed for  user 'andym'
```

این پیغام بدان معناست که شما با سرور ارتباط برقرار کرده‌اید و سرور هم تمایل به برقراری ارتباط دارد ولی تا قبل از اینکه شما روش تشخیص هویت آورده شده در تنظیمات pg_hba.conf را

پشت سر نگذارید امکان‌پذیر نمی‌باشد. رمز عبور یا Kerberos یا نرم‌افزار ident خود را چک کنید. که البته بسته به نوع احرازهویت آنرا پیگیری کنید.

```
FATAL 1: user "andym" does not exist
```

کاربر مربوطه یافت نشده‌است.

```
FATAL 1: Database "testdb" does not exist in the system catalog.
```

پایگاه‌داده‌ای که شما سعی در برقراری ارتباط با آن را دارید، پیدا نشده‌است. دقت کنید که نام پایگاه‌داده یا حساب کاربری پایگاه‌داده درست تعریف شده باشد.

۲-۷ فعالیتهای معمول برای نگهداری پایگاه داده

۲-۷-۱ مباحث عمومی

یکی از معمول‌ترین فعالیتهای معمول برای نگهداری PostgreSQL نصب شده، ایجاد نسخه‌های پشتیبان از داده‌ها در یک زمانبندی مشخص می‌باشد. بدون نسخهٔ پشتیبان پس از بروز یک مشکل مانند خطای دیسک، آتش‌سوزی، از بین رفتن اشتباهی جداول و... شانسی برای بازیابی اطلاعات وجود نخواهد داشت.

چند فعالیت دیگر وجود دارد که باید به صورت دوره‌ای انجام شود که در زیر آورده شده است و در قسمتهای بعدی به شرح آنها می‌پردازیم.

مورد دیگر که باید انجام شود عمل vacuuming دوره‌ای پایگاه‌داده‌ها می‌باشد. که شامل سه مرحله می‌باشد:

۱- بازیابی فضای دیسک توسط بروزرسانی یا حذف سطرها

۲- بروزرسانی داده‌های ایستا توسط PostgreSQL query planner.

۳- حفاظت از داده‌های قدیمی مربوط به transaction ID wraparound

عمل دیگری که باید به صورت دوره‌ای انجام گردد مدیریت و نگهداری فایلهای log می‌باشد. یکی از بهترین فعالیتها نگهداری خروجی فایل log مربوط به پایگاه‌داده سرور در جایی غیر از مسیر اصلی یعنی /dev/null، می‌باشد. البته خروجی فایل log در مراحل بالای debug نگهداری می‌شود و دیگر

نیازی به نگهداری همیشگی آنها نیست. شما باید این فایلها را به گردش در آورید یعنی فایلهـای log جدید را نگه داشته و قدیمی‌ترها را هر چند وقت یکبار از بین ببرید.

۲-۸ فرآیند بازیابی و ایجاد نسخه پشتیبان

PostgreSQL باید از تمامی داده‌های ارزشمند نسخه پشتیبان تهیه کند. باوجودیکه فرآیند ایجاد نسخه پشتیبان ساده می‌باشد، داشتن فهم کاملی از تکنیکها و فرضیات آن ضروری به نظر می‌رسد.

دو روش برای ایجاد نسخه پشتیبان از داده‌های PostgreSQL وجود دارد:

- SQL dump
- File system level backup

۲-۸-۱ SQL Dump

ایده اصلی در این روش، ایجاد یک فایل متنی از طریق دستورهای SQL است کـه در زمـان لازم پایگاه داده را در همـان وضـعیت، مجـددا بسـازد. بـرای ایـن منظـور PostgreSQL برنامـه سـودمند pg_dump را ارائه می‌دهد. کاربرد اصلی این دستور به صورت زیر است:

```
pg_dump dbname > outfile
```

pg_dump یک برنامه کاربردی مشتری است (البته از نوع هوشمند آن) و همان طور که مشاهده می‌کنید، pg_dump نتایج را در یک خروجی استاندارد می‌نویسد. منظور از هوشمند بودن این برنامه، آنست که شما می‌توانید رویه پشتیبان‌گیری را از طریق کنترل از راه دور انجام دهید. بـه یـاد داشـته باشید که برای این کار نیاز به داشتن اجازه خاصی نیست. در واقع کافی است شـما اجـازه خوانـدن و دسترسی به پایگاه داده از این طریق را داشته باشید.

۲-۸-۲ بازیابی Dump

فایل متنی که از طریق pg_dump ایجاد می‌شود، بـه وسـیله برنامـه psql قابـل خوانـدن اسـت. دستور معمول برای بازیابی یک dump به صورت زیر است:

```
psql dbname < infile
```

که در آن infile همان است که شما در قسمت قبل به عنوان outfile در دستور pg_dump استفاده کردید. پایگاه داده dbname از طریق این دستور ایجاد نمی‌گردد، شما باید خودتان قبل از اجرای psql، از template0 آن را بسازید:

```
createdb -T template0 dbname
```

یک ویژگی بسیار مهم pg_dump و psql، امکان خواندن و یا نوشتن از طریق لوله‌ها است که dump کردن پایگاه داده را مستقیما از طریق یک سرور به سرور دیگر به ما می‌دهد. برای مثال:

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

۲-۸-۳ سطوح مختلف پشتیبانی

راهبرد دیگر برای تهیه نسخه‌های پشتیبان، اینست که فایلهایی که PostgreSQL برای ذخیره داده‌های پایگاه داده استفاده می‌کند را مستقیما کپی کنیم. شما می‌توانید هر روشی که خود ترجیح می‌دهید برای انجام پشتیبانگیری از سیستم فایلها را به کار برید. برای مثال:

```
tar -cf backup.tar /usr/local/pgsql/data
```

وجود دو محدودیت سبب غیر عملی شدن و در برخی موارد نامرتب شدن روش pg_dump می‌گردد:

برای گرفتن یک نسخه پشتیبان کارآمد، سرور پایگاه داده باید حتما خاموش شود. به همین دلیل توصیه نمی‌شود که به فایل سیستمهایی که از "snapshot های پایدار" پشتیبانی می‌کنند، اعتماد کنید. و البته نیازی به گفتن این مسئله نیست که سرور باید قبل از بازیابی داده‌ها نیز خاموش شود.

۱. چنانچه شما مجبور به dug into جزئیات سیستم فایل باشید، ممکن است بخواهید فقط جدولها یا پایگاه داده‌های خاصی را از فایلها یا دایرکتوریهای مربوطه، بازیابی کرده یا از آنها پشتیبانگیری کنید. چنین کاری عملی نیست زیرا اطلاعات موجود در این فایلها تنها نیمی از واقعیت است. نیمی دیگر از آنها در فایلهای ثبت pg_clog/* قرار داند.

۲-۸-۴ مهاجرت بین نسخه‌ها

به عنوان یک قانون عمومی، قالب ذخیره داده‌ای داخلی۱ در نسخه‌های مختلف ostgreSQL قابل تغییر است. این موضوع برای "patch level"های مختلف کاربرد ندارد و فقط با قالبهای ذخیره‌سازی، سازگار است. به عنوان مثال، نسخه‌های ۷،۰،۱، ۷،۱،۲ و ۷،۲ با یکدیگر سازگار نیستند، درحالیکه ۷،۱،۱ و ۷،۱،۲ سازگارند. هنگامی که به یک نسخه سازگار به روز رسانی می‌کنید، به سادگی قادر خواهید بود مجددا از داده‌های موجود روی دیسک، با اجرای نسخه جدید استفاده نمایید. در غیر این صورت باید با استفاده از pg_dump ابتدا یک نسخه پشتیبان از داده‌ها تهیه کرده و سپس روی سرور جدید بازیابی کنید.

با نصب سرور جدید در یک دایرکتوری متفاوت و راه‌اندازی دو سرور جدید و قدیمی به صورت موازی، بر روی پورتهای جداگانه، آخرین زمان از کار افتادگی قابل دسترسی خواهد بود. آنگاه شما می‌توانید در صورت تمایل، مشابه زیر برای انتقال داده یا استفاده از فایل میانی عمل نمایید:

```
pg_dumpall -p 5432 | psql -d template1 -p 6543
```

سپس می‌توانید سرور قدیمی را خاموش کرده و سرور جدید را روی پورتی که سرور قبل کار می‌کرد راه‌اندازی کنید. شما باید مطمئن باشید که پایگاه داده پس از اجرای pg_dumpall به‌روز رسانی نمی‌شود، در غیر این صورت شما قطعا اطلاعاتی را از دست خواهید داد.

چنانچه نمی‌توانید یا نمی‌خواهید دو سرور را به صورت موازی راه‌اندازی کنید، می‌توانید مرحله پشتیبان‌گیری را قبل از نصب نسخه جدید انجام دهید؛ نسخه قدیمی را غیر قابل دسترس کرده، نسخه جدید را نصب و سرور جدید را راه‌اندازی نمایید، سپس داده را بازیابی کنید. برای مثال:

```
pg_dumpall > backup
pg_ctl stop
mv /usr/local/pgsql /usr/local/pgsql.old
cd /usr/src/postgresql-7.3.2
gmake install
```

---

۱. http://jakarta.apache.org/ant/index.html

[1] internal data storage format

```
initdb -D /usr/local/pgsql/data
postmaster -D /usr/local/pgsql/data
psql template1 < backup
```

۲-۹ مانیتور کردن فعالیتهای پایگاه داده

در بسیاری مواقع برای راهبران مهم است که بدانند سیستم در حال حاضر چه می‌کند. در این بخش به توضیح این مسئله خواهیم پرداخت.

ابزارهای متعددی برای مانیتور کردن فعالیتهای پایگاه داده در دسترس است. در این بخش به توضیح کلکتور استاتیک[1] PostgreSQL می‌پردازیم. البته نباید برنامه‌های مانیتورینگ یونیکس مانند ps و top را فراموش کرد.

۲-۹-۱ ابزارهای استاندارد یونیکس

در بیشتر بسترها، PostgreSQL عنوان دستورات خود را طبق آنچه در ps گزارش می‌شود، تغییر می‌دهد، تا فرآیندهای مخصوص یک سرور به آسانی شناسایی شوند. یک مثال به صورت زیر است:

```
$ ps auxww | grep ^postgres
postgres 960 0.0 1.1 6104 1480 pts/1 SN 13:17 0:00 postmaster
-i
postgres 963 0.0 1.1 7084 1472 pts/1 SN 13:17 0:00 postgres: stats buffer
postgres 965 0.0 1.1 6152 1512 pts/1 SN 13:17 0:00 postgres: stats collector
process
postgres 998 0.0 2.3 6532 2992 pts/1 SN 13:18 0:00 postgres: tgl runbug
127.0.0.1 idle
postgres 1003 0.0 2.4 6532 3128 pts/1 SN 13:19 0:00 postgres: tgl regression
[local] SELECT waiting
postgres 1016 0.1 2.4 6532 3080 pts/1 SN 13:19 0:00 postgres: tgl regression
[local] idle in transaction
```

---

۱. http://jakarta.apache.org/ant/index.html

[1] statistics collector

(احضار مناسب ps و جزئیات آنچه نشان داده شده است، در بسترهای مختلف تغییر می‌کند.) پروسه اولی که در اینجا لیست شده است postmaster می‌باشد، که پروسه ارشد است. دو پروسه بعد استاتیک کلکتورها را اجرا می‌نمایند که در بخش بعد درباره آنها توضیح داده خواهد شد. هر یک از این پروسسها، یک پروسس سرور است که یک ارتباط کارخواه را اداره می‌کنند. هر یک از آنها خط فرمان خود را به صورت زیر نمایش می‌دهند:

```
postgres: user database host activity
```

کاربر، پایگاه داده و اقلام ارتباط منبع میزبان، در طول حیات ارتباط با کارخواه، به همان شکل باقی می‌مانند، اما نشانه‌گر[1] فعالیت تغییر می‌کند. یک فعالیت ممکن است idle (در انتظار یک دستور کارخواه)، idle in transaction (در انتظار وارد کردن بلوک BEGIN توسط کارخواه) و یا نام یک نوع دستور مانند SELECT باشد. در مثال بالا دیده می‌شود که پروسس ۱۰۰۳ منتظر پروسس ۱۰۱۶ است تا یک تراکنش را کامل نماید.

۲-۹-۲ کلکتور استاتیک

کلکتور استاتیک PostgreSQL در حقیقت زیرسیستمی است که از collection پشتیبانی می‌نماید و اطلاعاتی درباره فعالیتهای سرور گزارش می‌دهد. در حال حاضر یک کلکتور امکان شمارش تعداد دسترسیها به جداول و ایندکسها را، هم در شرایط بلوک دیسک و هم در مورد ردیفهای مشخص و منحصر به فرد دارد.

۲-۹-۳ پیکربندی استاتیک کلکتور

از آنجا که collectionهای استاتیک سربار اضافی در اجرای یک پرس‌وجو ایجاد می‌کنند، می‌توان سیستم را به طریقی پیکربندی کرد که اطلاعات را جمع‌آوری کند یا جمع‌آوری نکند. این کار از طریق پیکربندی متغیرهایی که معمولا در postgresql.conf قرار دارند، کنترل می‌گردد.

---

۱. http://jakarta.apache.org/ant/index.html

[1] indicator

متغیر STATS_START_COLLECTOR باید برای کلکتور استاتیک دارای مقدار true باشد تا به هیچ وجه اجرا نگردد. این تنظیم پیش‌فرض و توصیه شده است. شما می‌توانید در صورت عدم علاقه به استاتیک آن را غیرفعال کنید. البته توجه به این نکته مهم است که نمی‌توان این گزینه را در هنگامی که سرور در حال کار است، تغییر داد.

متغیرهای STATS_COMMAND_STRING، STATS_BLOCK_LEVEL و STATS_ROW_LEVEL مقدار اطلاعاتی که واقعا به کلکتور ارسال می‌شود را کنترل می‌کنند و بدین سان تعیین می‌کنند چه مقدار سربار در زمان اجرا اتفاق افتاده است. این متغیرها به ترتیب تعیین می‌کنند که آیا پروسس سرور رشته دستور جاری خود را ارسال کرده است، آمار دسترسی به سطح بلوک دیسک و آمار دسترسی به سطح ردیف را به کلکتور نشان می‌دهند. معمولا این متغیرها در postgresql.conf قرار دارند و از این رو می‌توانند برای تمامی پروسسهای سرور به کار گرفته شوند.

متغیرهای STATS_COMMAND_STRING، STATS_BLOCK_LEVEL و STATS_ROW_LEVEL به صورت پیش‌فرض دارای مقدار false هستند. شما باید برای گرفتن نتایج مفید و قابل استفاده قبلا یکی از آنها یا همه را فعال نمایید.

Viewing locks ۴-۹-۲

ابزار کاربردی دیگری که برای مانیتور کردن اعمال پایگاه‌داده استفاده از کاتالوگ سیستم pg_locks می‌باشد. این ابزار به راهبر پایگاه‌داده امکان با خبر شدن از اطلاعات در رابطه با outstanding locks در مدیریت قفل را می‌دهد. برای مثال از قابلیتهای زیر می‌توان استفاده کرد:

- مشاهده تمامی قفلهایی که در حال حاضر outstanding هستند، تمام قفلهای روی ارتباطهای موجود در یک پایگاه داده، تمام قفلهای روی یک ارتباط مشخص، یا تمام قفلهایی که توسط یک session مشخص PostgreSQL نگه داشته می‌شوند.

- مشاهده رابطه در پایگاه داده جاری به همراه بیشتر قفلهای داده نشده.
- تعیین تاثیر رقابت قفل[۱] بر روی کارایی کل پایگاه داده.

جدول زیر تعریف ستونهای pg_locks را نشان می‌دهد. pg_locks محتوای یک سطر به ازای هر شیئ قابل قفل کردن و حالت قفل درخواست شده را می‌بیند. یک شیئ قابل قفل شدن یا یک رابطـه است یا یک ID تراکنش.

۱. http://jakarta.apache.org/ant/index.html

[1] lock contention

Column Name	Type	Description
relation	oid	The OID of the locked relation, or null if the lockable object is a transaction ID. This column can be joined with the pg_class system catalog to get more information on the locked relation. Note however that this will only work for relations in the current database (those for which the database column is either the current database's OID or zero).
database	oid	The OID of the database in which the locked relation exists, or null if the lockable object is a transaction ID. If the lock is on a globally-shared table, this field will be zero. This column can be joined with the pg_database system catalog to get more information on the locked object's database.
transaction	xid	The ID of a transaction, or null if the lockable object is a relation. Every transaction holds an exclusive lock on its transaction ID for its entire duration. If one transaction finds it necessary to wait specifically for another transaction, it does so by attempting to acquire share lock on the other transaction ID. That will succeed only when the other transaction terminates and releases its locks.
pid	integer	The process ID of the PostgreSQL backend belonging to the session that has acquired or is attempting to acquire the lock. If you have enabled the statistics collector, this column can be joined with the pg_stat_activity view to get more information on the backend holding or waiting to hold the lock.

Column Name	Type	Description
mode	text	The mode of the requested or held lock on the lockable object. For more information on the different lock modes available in PostgreSQL, refer to the *PostgreSQL User's Guide*.
isgranted	boolean	True if this lock has been granted (is held by this session). False indicates that this session is currently waiting to acquire this lock, which implies that some other session is holding a conflicting lock mode on the same lockable object. This backend will sleep until the other lock is released (or a deadlock situation is detected). A single backend can be waiting to acquire at most one lock at a time.

۲-۱۰ مانیتور کردن فضای دیسک

در این بخش چگونگی مانیتور کردن فضای استفاده شده دیسک در سیستم پایگاه داده PostgreSQL توضیح داده خواهد شد. در نسخه مورد نظر راهبر پایگاه داده کنترل زیادی بر روی فضای ذخیره‌سازی دیسک ندارد. بنابراین این بخش شامل اطلاعات مفید و آموزنده است و می‌تواند به شما ایده‌هایی در چگونگی مدیریت فضای دیسک با استفاده از ابزارهای سیستم‌عامل دهد.

۲-۱۰-۱ عامل تعیین کننده مصرف دیسک

هر جدولی یک فایل دیسک heap اولیه دارد که بیشتر داده‌ها در آن ذخیره می‌شوند. برای ذخیره مقادیر طولانی ستون نیز یک فایل TOAST به همراه جدول، که نام آن بر اساس OID جدول مشخص می‌گردد (در حقیقت pg_class.relfilenode) و یک ایندکس بر روی جدول TOAST، وجود دارد. ممکن است ایندکسهایی نیز به همراه جدول پایه وجود داشته باشد.

شما می‌توانید فضای دیسک را از سه جا مانیتور کنید: از psql با استفاده از اطلاعات VACUUM، از psql با استفاده از contrib/dbsize و از خط فرمان با استفاده از contrib/oid2name. با استفاده از psql بر روی فضای خالی پایگاه داده می‌توانید پرس‌وجوهایی جهت دیدن فضای استفاده شده توسط هر جدول، ایجاد کنید:

```
play=# SELECT relfilenode, relpages
play-# FROM pg_class
play-# WHERE relname = 'customer';
relfilenode | relpages
-------------+----------
16806 | 60
(1 row)
```

هر صفحه حدودا ۸ کیلوبایت است. برای دیدن فضای استفاده شده توسط جدول TOAST از پرس‌وجویی بر مبنای heap relfilenode که در بالا نشان داده شده استفاده کنید:

```
play=# SELECT relname, relpages
play-# FROM pg_class
play-# WHERE relname = 'pg_toast_16806' OR
play-# relname = 'pg_toast_16806_index'
play-# ORDER BY relname;
relname | relpages
----------------------+----------
pg_toast_16806 | 0
pg_toast_16806_index | 1
```

همچنین می‌توانید فضای استفاده شده توسط ایندکس را نیز مشاهده کنید:

```
play=# SELECT c2.relname, c2.relpages
play-# FROM pg_class c, pg_class c2, pg_index i
play-# WHERE c.relname = 'customer' AND
play-# c.oid = i.indrelid AND
play-# c2.oid = i.indexrelid
play-# ORDER BY c2.relname;
relname | relpages
----------------------+----------
customer_id_indexdex | 26
```

پیدا کردن بزرگترین فایلهایی که از psql استفاده می‌کنند نیز به سادگی ممکن است:

```
play=# SELECT relname, relpages
play-# FROM pg_class
play-# ORDER BY relpages DESC;
relname | relpages
```

```
----------------------+----------
bigtable | 3290
customer | 3144
```

contrib/dbsize توابع را به پایگاه داده شما لود می‌کند که امکان پیدا کردن اندازه جدول یا پایگاه داده را از داخل psql بدون نیاز به VACUUM/ANALYZE می‌دهد.

همچنین می‌توانید از contrib/oid2name جهت نشان دادن فضای استفاده شده دیسک استفاده کنید. برای دیدن مثالهایی در این رابطه فایل README.oid2name را ببینید.

۲-۱۰-۲ خطای پرشدن دیسک

یکی از مهمترین اعمال مانیتورینگ دیسک توسط راهبران پایگاه‌داده حصول اطمینان از وجود فضای خالی در دیسک می‌باشد. دیسک پر شده از داده سبب بروز اختلالات و خرابی فراوان در ایندکسهای پایگاه داده می‌شود، اما اطلاعات اساسی و بنیادی داده‌های جداول را خراب نمی‌کند. اگر فایلهای WAL بر روی یک دیسک باشند (تنظیمات پیش‌فرض چنین است) آنگاه پر بودن دیسک در هنگام مقداردهی اولیه پایگاه داده ممکن است سبب گردد خرابی یا ناقص شدن فایلهای WAL گردد. این شرایط خطا شناسایی شده و سرور پایگاه داده مانع از راه‌اندازی می‌گردد.

اگر نمی‌توانید فضای خالی کافی از طریق حذف کردن سایر چیزها به وجود آورید، می‌توانید برخی فایلهای پایگاه داده را به فایل سیستمهای دیگر برده و یک symlink از مکان اصلی ایجاد کنید. اما توجه کنید که pg_dump نمی‌تواند اطلاعات ترکیب اصلی مکان این آماده‌سازی را ذخیره نماید. با یک بازیابی همه چیز را به یک مکان برمی‌گرداند. برای اجتناب از تمام شدن فضای خالی دیسک، می‌توانید فایلهای WAL و یا پایگاه داده‌های مشخصی را در مکان دیگری در هنگام ایجاد آنها قرار دهید. برای کسب اطلاعات بیشتر اسناد initdb را مطالعه کنید.

۳- راهنما برای برنامه نویسان PostgreSQL

۳-۱ واسط کارخواه

در این بخش از راهنما به توضیح برنامه‌نویسی واسط سمت کارخواه و کتابخانه‌هایی که برای زبانهای مختلف پشتیبانی می‌شوند، خواهیم پرداخت.

۳-۱-۱ کتابخانه Libpq-c

Libpq یک واسط برنامه‌نویسی کاربردی زبان C برای PostgreSQL می‌باشد. Libpq یک مجموعه از روالهای کتابخانه‌ای می‌باشد که به برنامه‌های Client اجازه فرستادن پرس و جو به کارساز PostgreSql و دریافت نتیجه آن را می‌دهند. Libpq همچنین یک موتور اساسی برای واسط برنامه‌های کاربردی دیگر PostgreSQL از قبیل(++c) Libpq ، Libpgtcl(tc)، perl ،ecpg می‌باشد. اگر بخواهید از یکی از آن بسته‌ها استفاده کنید برخی از رفتارهای Libpq مهم می شوند.

سه برنامه کوچک در انتهای این بخش برای نمایش چگونگی برنامه‌نویسی با Libpq آورده شده‌است.

چندین مثال کامل برای برنامه‌های کاربردی با Libpq در مسیرهای زیر آورده شده‌است:

```
src/test/examples
src/bin/psql
```

۳-۱-۲ توابع برقراری ارتباط پایگاه داده

برنامه‌های آغازین باید دارای فایل سرآیند libpq-fe.d باشند و همچنین به کتابخانه libpq پیوند زده شوند.

روالهای زیر با ساخت یک ارتباط با سرور libpq مقداردهی می‌شود.

برنامه‌های کاربردی می‌توانند در یک زمان چندین ارتباط باز داشته باشند (یکی از دلایل آن ارتباط به بیش از یک پایگاه داده می‌باشد.) هر ارتباط با یک شی PGconn نمایش داده شده‌است که از PQconnectdb یا PQsetdbLogin احراز شده‌است. دقت داشته‌باشید که این توابع همیشه باید

یک اشاره‌گر شیی غیر پوچ را بازگشت دهند مگر اینکه حافظه خیلی کم برای اختصاص دادن به شی PGconn موجود باشد.

تابع PQstatus قبل از ارسال پرس و جو از طریق شی ارتباط بررسی می‌کند که آیا ارتباط به درستی بر قرار شده است یا خیر .

PQconnectdb یک ارتباط جدید با خادم پایگاه‌داده برقرار می‌کند.

```
PGconn *PQconnectdb(const char *conninfo)
```

این روال یک ارتباط پایگاه‌داده را به وسیله پارامترهای استفاده شده از طریق رشته conninfo را امکان‌پذیر می‌کند.برعکس در PQsetdbLogin مجموعه پارامترها بدون تغییر در صحت تابع می‌توانند گسترش پیدا کنند. در نتیجه استفاده از این روال یا PQconnectStart و PQconnectPoll برای برنامه‌های کاربردی ترجیح داده شده است. رشته عبوری می‌تواند برای تمام پارامترهای پیش‌فرض خالی باشد یا می‌تواند شامل یک یا چندین حالت پارامتر که با فاصله سفید مجزا شده‌اند باشد.

هر حالت پارامتر به فرم keyword=value می‌باشد.(برای نوشتن یک مقدار خالی یا مقداری که شامل فاصله باشد باید آن را بوسیله single quotes محصور کرد ،e.g.، keyword='a value'.

Single quotes و backslash در یک مقدار با یک backslash گریز داده می‌شود، e.g.، \\ یا \') فاصله اطراف علامت برابری، اختیاری می‌باشد. کلید واژه‌های پارامتر عبارتند از:

Host

نام میزبانی که به آن متصل می‌شود.

Hostaddr

نشانی IP میزبانی که به آن متصل می‌شود.

Port

شماره درگاهی که از آن برای اتصال به میزبان کارساز استفاده می‌شود، یا پسوند نام پرونده سوکت برای ارتباط دامنه- UNIX

dbname

نام پایگاه‌داده

user

نام کاربری که از آن برای اتصال استفاده می‌شود.

Password

رمز عبوری که استفاده می‌شود در صورتی که کارساز اعتبارسنجی کلمه عبور را درخواست کرد.

connect_timeout

فاصله زمانی به ثانیه که ارتباط بر قرار است. اگر مقدار آن صفر باشد یا مقداردهی نشود به معنی نامحدود بودن آن می‌باشد.

option

انتخابهای trace/debug که به کارساز فرستاده می‌شود.

Tty

یک فایل یا tty برای خروجی اشکال‌زدایی اختیاری.

Requiressl

برای درخواست اتصال SSL به کارساز به ۱ مقداردهی می‌شود. libpq اگر کارساز اتصال SSL را نپذیرد، اتصال را نمی‌پذیرد. با صفر برای مبادلات با کارساز مقداردهی می‌شود. این گزینه فقط در صورتی در دسترس می‌باشد که postgreSQL با پشتیبانی SSL ترجمه شود. PQsetdbLogin یک ارتباط جدید با کارساز بر قرار می‌کند.

```
PGconn *PQsetdbLogin(const char *pghost,
const char *pgport,
const char *pgoptions,
const char *pgtty,
const char *dbName,
const char *login,
const char *pwd)
```

PQsetdb یک ارتباط جدید با کارساز پایگاه‌داده ایجاد می‌کند.

```
PGconn *PQsetdb(char *pghost,
char *pgport,
char *pgoptions,
char *pgtty,
char *dbName)
```

PQconnectStart و PQconntectPoll یک ارتباط non-blocking را برای اتصال به کارساز پایگاه‌داده برقرار می‌سازند.

```
PGconn *PQconnectStart(const char *conninfo)
PostgresPollingStatusType PQconnectPoll(PGconn *conn)
```

برای شروع،

```
conn=PQconnectStart("connection_info_string")
```

را فراخوانی کنید. اگر مقدار conn پوچ باشد، آنگاه libpq قادر به تخصیص دادن یک ساختار PGconn جدید نمی‌باشد. در غیر این صورت یک مقدار معتبر

PGconn برگشت داده می‌شود. در فراخوانی از PQconnectStart، status=PQstatus(conn) فراخوانی می‌شود. اگر status برابر CONNECTION_BAD باشد، PQconnectStart مردود شده‌است.

اگر PQconnectStart با موفقیت انجام شود،

اگر آخرین فراخوانی PQconnectPoll، PGTES_POLLING_ACTIVE باشد، به صورت "active" ملاحظه می‌شود.

اگر آخرین فراخوانی PQconnectPol(conn)، PGRES_POLLING_READING باشد، یک select() برای خواندن روی PQsocket(conn) انجام می‌شود.

اگر آخرین فراخوانی آن، PGRES_POLLING_WRITING باشد، یک select() برای نوشتن روی PQsocket(conn) انجام می‌شود.

۱- واسط میانبر(fast-path).

PostgreSQL یک واسط میانبر برای فرستادن فراخوانی تابع فراهم کرده‌است. این یک trapdoor در د اخل سیستم می‌باشد و می‌تواند یک حفره امنیتی باشد. اغلب کاربرها به این ویژگی نیازی نخواهند داشت.

PQfn اجرای تابع را از طریق واسط میانبر درخواست می‌کند.

```
PGresult* PQfn(PGconn* conn,
int fnid,
```

```
int *result_buf,
int *result_len,
int result_is_int,
const PQArgBlock *args,
int nargs);
```

نشانوند fnid شناسه شیی تابعی است که اجرا می‌شود.result_buf میانگیر موقتی است که مقدار برگشتی در آن قرار می‌گیرد و باید دارای فضای کافی برای ذخیره مقدار بازگشتی باشد. result_len طول مقدار برگشتی را نشان می‌دهد. اگر یک نتیجه صحیح ۴ بایت مورد انتظار بود مقدار result_is_int برابر یک و در غیر این صورت برابر صفر خواهد بود. args و nargs نشانوندهای تابع را مشخص می‌کنند

```
typedef struct {
int len;
int isint;
union {
int *ptr;
int integer;
} u;
} PQArgBlock;
```

pqfn یک *PGresult معتبر بازگردانی می‌کند.

۲- توابع ردیابی libpq

PQtrace ردیابی ارتباطات در یک جریان پرونده اشکال‌زدا را فعال می‌کند.

```
void PQtrace(PGconn *conn
FILE *debug_port)
```

PQuntrace ردیابی آغاز شده بوسیله PQtrace را غیرفعال می‌کند.

```
void PQuntrace(PGconn *conn)
```

۳- توابع کنترلی libpq

```
typedef void (*PQnoticeProcessor) (void *arg, const char *message);
PQnoticeProcessor
PQsetNoticeProcessor(PGconn *conn,
PQnoticeProcessor proc,
void *arg);
```

۴- پرونده‌ها

پرونده pgpass. در فهرست شخصی، پرونده‌ای است که می‌تواند شامل اسم رمزهایی است اگر در اتصال اسم رمز درخواست شود. این پرونده باید قالب زیر را داشته باشد:

hostname:*port*:*database*:*username*:*password*

هر کدام از آنها ممکن یک اسم واقعی، یا * ، که همه چیز را شامل می‌شود.

مجوزها در pgpass. بایستی هرگونه دسترسی غیر مجاز را ممنوع کنند; برای این کار از فرمان chmod 0600 .pgpass استفاده می‌کنیم.

برنامه‌های نمونه

مثال ۱ مربوط به libpq

```
/*
* testlibpq.c
*
* Test the C version of libpq, the PostgreSQL frontend
* library.
*/
#include <stdio.h>
#include <libpq-fe.h>
void
exit_nicely(PGconn *conn)
{
PQfinish(conn);
exit(1);
}
main()
{
char *pghost,
*pgport,
*pgoptions,
*pgtty;
char *dbName;
int nFields;
int i,
j;
/* FILE *debug; */
PGconn *conn;
PGresult *res;
/*
* begin, by setting the parameters for a backend connection if the
* parameters are null, then the system will try to use reasonable
```

```
* defaults by looking up environment variables or, failing that,
* using hardwired constants
*/
pghost = NULL; /* host name of the backend server */
pgport = NULL; /* port of the backend server */
pgoptions = NULL; /* special options to start up the backend
* server */
pgtty = NULL; /* debugging tty for the backend server */
dbName = "template1";
/* make a connection to the database */
conn = PQsetdb(pghost, pgport, pgoptions, pgtty, dbName);
/*
* check to see that the backend connection was successfully made
*/
if (PQstatus(conn) == CONNECTION_BAD)
{
fprintf(stderr, "Connection to database '%s' failed.\n", dbName);
fprintf(stderr, "%s", PQerrorMessage(conn));
exit_nicely(conn);
}
/* debug = fopen("/tmp/trace.out","w"); */
/* PQtrace(conn, debug); */
/* start a transaction block */
res = PQexec(conn, "BEGIN");
if (!res || PQresultStatus(res) != PGRES_COMMAND_OK)
{
fprintf(stderr, "BEGIN command failed\n");
PQclear(res);
exit_nicely(conn);
}
/*
* should PQclear PGresult whenever it is no longer needed to avoid
* memory leaks
*/
PQclear(res);
/*
* fetch rows from the pg_database, the system catalog of
* databases
*/
res = PQexec(conn, "DECLARE mycursor CURSOR FOR SELECT * FROM pg_database");
if (!res || PQresultStatus(res) != PGRES_COMMAND_OK)
{
fprintf(stderr, "DECLARE CURSOR command failed\n");
PQclear(res);
exit_nicely(conn);
}
```

```
PQclear(res);
res = PQexec(conn, "FETCH ALL in mycursor");
if (!res || PQresultStatus(res) != PGRES_TUPLES_OK)
{
fprintf(stderr, "FETCH ALL command didn't return tuples properly\n");
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 cursor */
res = PQexec(conn, "CLOSE mycursor");
PQclear(res);
/* commit the transaction */
res = PQexec(conn, "COMMIT");
PQclear(res);
/* close the connection to the database and cleanup */
PQfinish(conn);
/* fclose(debug); */
return 0;
}
```

مثال ۲ مربوط به libpq

```
/*
* testlibpq2.c
* Test of the asynchronous notification interface
*
* Start this program, then from psql in another window do
```

```
* NOTIFY TBL2;
*
* Or, if you want to get fancy, try this:
* Populate a database with the following:
*
* 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
*
* INSERT INTO TBL1 values (10);
*
*/
#include <stdio.h>
#include "libpq-fe.h"
void
exit_nicely(PGconn *conn)
{
PQfinish(conn);
exit(1);
}
main()
{
char *pghost,
*pgport,
*pgoptions,
*pgtty;
char *dbName;
int nFields;
int i,
j;
PGconn *conn;
PGresult *res;
PGnotify *notify;
/*
* begin, by setting the parameters for a backend connection if the
* parameters are null, then the system will try to use reasonable
* defaults by looking up environment variables or, failing that,
* using hardwired constants
*/
pghost = NULL; /* host name of the backend server */
pgport = NULL; /* port of the backend server */
```

```
pgoptions = NULL; /* special options to start up the backend
* server */
pgtty = NULL; /* debugging tty for the backend server */
dbName = getenv("USER"); /* change this to the name of your test
* database */
/* make a connection to the database */
conn = PQsetdb(pghost, pgport, pgoptions, pgtty, dbName);
/*
* check to see that the backend connection was successfully made
*/
if (PQstatus(conn) == CONNECTION_BAD)
{
fprintf(stderr, "Connection to database '%s' failed.\n", dbName);
fprintf(stderr, "%s", PQerrorMessage(conn));
exit_nicely(conn);
}
res = PQexec(conn, "LISTEN TBL2");
if (!res || PQresultStatus(res) != PGRES_COMMAND_OK)
{
fprintf(stderr, "LISTEN command failed\n");
PQclear(res);
exit_nicely(conn);
}
/*
* should PQclear PGresult whenever it is no longer needed to avoid
* memory leaks
*/
PQclear(res);
while (1)
{
/*
* wait a little bit between checks; waiting with select()
* would be more efficient.
*/
sleep(1);
/* collect any asynchronous backend messages */
PQconsumeInput(conn);
/* check for asynchronous notify messages */
while ((notify = PQnotifies(conn)) != NULL)
{
fprintf(stderr,
"ASYNC NOTIFY of '%s' from backend pid '%d' received\n",
notify->relname, notify->be_pid);
free(notify);
}
}
```

```
/* close the connection to the database and cleanup */
PQfinish(conn);
return 0;
}
```

مثال ۳ مربوط به libpq

```
/*
* testlibpq3.c Test the C version of Libpq, the PostgreSQL frontend
* library. tests the binary cursor interface
*
*
*
* populate a database by doing the following:
*
* CREATE TABLE test1 (i int4, d real, p polygon);
*
* INSERT INTO test1 values (1, 3.567, polygon '(3.0, 4.0, 1.0, 2.0)');
*
* INSERT INTO test1 values (2, 89.05, polygon '(4.0, 3.0, 2.0, 1.0)');
*
* the expected output is:
*
* tuple 0: got i = (4 bytes) 1, d = (4 bytes) 3.567000, p = (4
* bytes) 2 points boundbox = (hi=3.000000/4.000000, lo =
* 1.000000,2.000000) tuple 1: got i = (4 bytes) 2, d = (4 bytes)
* 89.050003, p = (4 bytes) 2 points boundbox =
* (hi=4.000000/3.000000, lo = 2.000000,1.000000)
*
*
*/
#include <stdio.h>
#include "libpq-fe.h"
#include "utils/geo_decls.h" /* for the POLYGON type */
void
exit_nicely(PGconn *conn)
{
PQfinish(conn);
exit(1);
}
main()
{
char *pghost,
```

```
*pgport,
*pgoptions,
*pgtty;
char *dbName;
int nFields;
int i,
j;
int i_fnum,
d_fnum,
p_fnum;
PGconn *conn;
PGresult *res;
/*
* begin, by setting the parameters for a backend connection if the
* parameters are null, then the system will try to use reasonable
* defaults by looking up environment variables or, failing that,
* using hardwired constants
*/
pghost = NULL; /* host name of the backend server */
pgport = NULL; /* port of the backend server */
pgoptions = NULL; /* special options to start up the backend
* server */
pgtty = NULL; /* debugging tty for the backend server */
dbName = getenv("USER"); /* change this to the name of your test
* database */
/* make a connection to the database */
conn = PQsetdb(pghost, pgport, pgoptions, pgtty, dbName);
/*
* check to see that the backend connection was successfully made
*/
if (PQstatus(conn) == CONNECTION_BAD)
{
fprintf(stderr, "Connection to database '%s' failed.\n", dbName);
fprintf(stderr, "%s", PQerrorMessage(conn));
exit_nicely(conn);
}
/* start a transaction block */
res = PQexec(conn, "BEGIN");
if (!res || PQresultStatus(res) != PGRES_COMMAND_OK)
{
fprintf(stderr, "BEGIN command failed\n");
PQclear(res);
exit_nicely(conn);
}
/*
* should PQclear PGresult whenever it is no longer needed to avoid
```

```
* memory leaks
*/
PQclear(res);
/*
* fetch rows from the pg_database, the system catalog of
* databases
*/
res = PQexec(conn, "DECLARE mycursor BINARY CURSOR FOR SELECT * FROM test1");
if (!res || PQresultStatus(res) != PGRES_COMMAND_OK)
{
fprintf(stderr, "DECLARE CURSOR command failed\n");
PQclear(res);
exit_nicely(conn);
}
PQclear(res);
res = PQexec(conn, "FETCH ALL in mycursor");
if (!res || PQresultStatus(res) != PGRES_TUPLES_OK)
{
fprintf(stderr, "FETCH ALL command didn't return tuples properly\n");
PQclear(res);
exit_nicely(conn);
}
i_fnum = PQfnumber(res, "i");
d_fnum = PQfnumber(res, "d");
p_fnum = PQfnumber(res, "p");
for (i = 0; i < 3; i++)
{
printf("type[%d] = %d, size[%d] = %d\n",
i, PQftype(res, i),
i, PQfsize(res, i));
}
for (i = 0; i < PQntuples(res); i++)
{
int *ival;
float *dval;
int plen;
POLYGON *pval;
/* we hard-wire this to the 3 fields we know about */
ival = (int *) PQgetvalue(res, i, i_fnum);
dval = (float *) PQgetvalue(res, i, d_fnum);
plen = PQgetlength(res, i, p_fnum);
/*
* plen doesn't include the length field so need to
* increment by VARHDSZ
*/
```

```
pval = (POLYGON *) malloc(plen + VARHDRSZ);
pval->size = plen;
memmove((char *) &pval->npts, PQgetvalue(res, i, p_fnum), plen);
printf("tuple %d: got\n", i);
printf(" i = (%d bytes) %d,\n",
PQgetlength(res, i, i_fnum), *ival);
printf(" d = (%d bytes) %f,\n",
PQgetlength(res, i, d_fnum), *dval);
printf(" p = (%d bytes) %d points \tboundbox = (hi=%f/%f, lo = %f,%f)\n",
PQgetlength(res, i, d_fnum),
pval->npts,
pval->boundbox.xh,
pval->boundbox.yh,
pval->boundbox.xl,
pval->boundbox.yl);
}
PQclear(res);
/* close the cursor */
res = PQexec(conn, "CLOSE mycursor");
PQclear(res);
/* commit the transaction */
res = PQexec(conn, "COMMIT");
PQclear(res);
/* close the connection to the database and cleanup */
PQfinish(conn);
return 0;
}
```

۳-۱-۲ اشیای گسترده

۱. دسترسی به اشیای بزرگ از طریق Libpq

مثال ۲-۱ مثالی است که نشان می‌دهد واسط شیی بزرگ در libpq چطور می‌تواند مورد استفاده قرار گیرد.

Example 2-1. Large Objects with Libpq Example Program

```
/*--------------------------------------------------------------
*
* 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, "can't open unix file %s\n", filename);
}
/*
* create the large object
*/
lobjId = lo_creat(conn, INV_READ | INV_WRITE);
if (lobjId == 0)
fprintf(stderr, "can't 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, "can't 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_READ);
if (lobj_fd < 0)
{
fprintf(stderr, "can't 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;
/*
* create an inversion "object"
*/
lobj_fd = lo_open(conn, lobjId, INV_READ);
if (lobj_fd < 0)
{
fprintf(stderr, "can't 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, "can't open unix file %s\n",
filename);
}
```

```
/*
* read in from the Unix file and write to the inversion 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);
}
```

Pgtcl-Tcl ۳-۱-۳

Pgtcl یک بسته Tcl برای برنامه‌های کارخواه برای واسط بودن با کارسازهای PostgreSQL می‌باشد.

جدول ۳-۱ مروری بر دستورات موجود در pgtcl می‌باشد.

فرمان	توضیح
pg_connect	یک اتصال به کارساز backend را باز می‌کند.
pg_disconnect	یک اتصال را می‌بندد.
pg_conndefaults	گزینه‌های اتصال و پیش‌فرض‌های آن را می‌گیرد.

pg_exec	یک درخواست به backend می‌فرستد.
pg_result	نتیجه یک درخواست را اداره می‌کند.
pg_select	بر نتیجه یک جمله SELECT چرخش می‌کند.
pg_execute	یک درخواست می‌فرستد و بصورت اختیاری بر روی نتیجه چرخش می‌کند.
pg_listen	یک فراخوانی برگشتی برای پیغام‌های اطلاع رسانی برقرار می‌کند.
pg_on_connection_loss	یک فراخوانی بازگشتی برای افت اتصالات غیر منتظره برقرار می‌سازد
pg_lo_creat	یک شی گسترده می‌سازد
pg_lo_open	یک شی گسترده را باز می‌کند
pg_lo_close	یک شی گسترده را می‌بندد.
pg_lo_read	یک شی گسترده را می‌خواند
pg_lo_write	یک شی گسترده را می‌نویسد
pg_lo_lseek	یک موقعیت در شی گسترده را جستجو می‌کند

pg_lo_tell	موقعیت جستجوی فعلی از شی گسترده را بر می‌گرداند
pg_lo_unlink	یک شی گسترده را حذف می‌کند
pg_lo_import	یک پرونده Unix را وارد شی گسترده می‌کند
pg_lo_export	یک شی گسترده را وارد یک پرونده Unix می‌کند

Example 3-1. pgtcl Example Program

```
# getDBs :
# get the names of all the databases at a given host and port number
# with the defaults being the localhost and port 5432
# return them in alphabetical order
proc getDBs { {host "localhost"} {port "5432"} } {
# datnames is the list to be result
set conn [pg_connect template1 -host $host -port $port]
set res [pg_exec $conn "SELECT datname FROM pg_database ORDER BY datname"]
set ntups [pg_result $res -numTuples]
for {set i 0} {$i < $ntups} {incr i} {
lappend datnames [pg_result $res -getTuple $i]
}
pg_result $res -clear
pg_disconnect $conn
return $datnames
}
```

۳-۲ برنامه‌نویسی سرور

در این بخش از راهنما روش توسعه‌پذیری PostgreSQL را توضیح داده و شرح می‌دهیم چگونه کاربران از طریق اضافه کردن انواع، عملگرها و توابع زبان برنامه‌نویسی و نیز پرس و جوهای زبان که توسط خود کاربر تعریف شده‌اند، برای توسعه PostgreSQL اقدام کرد.

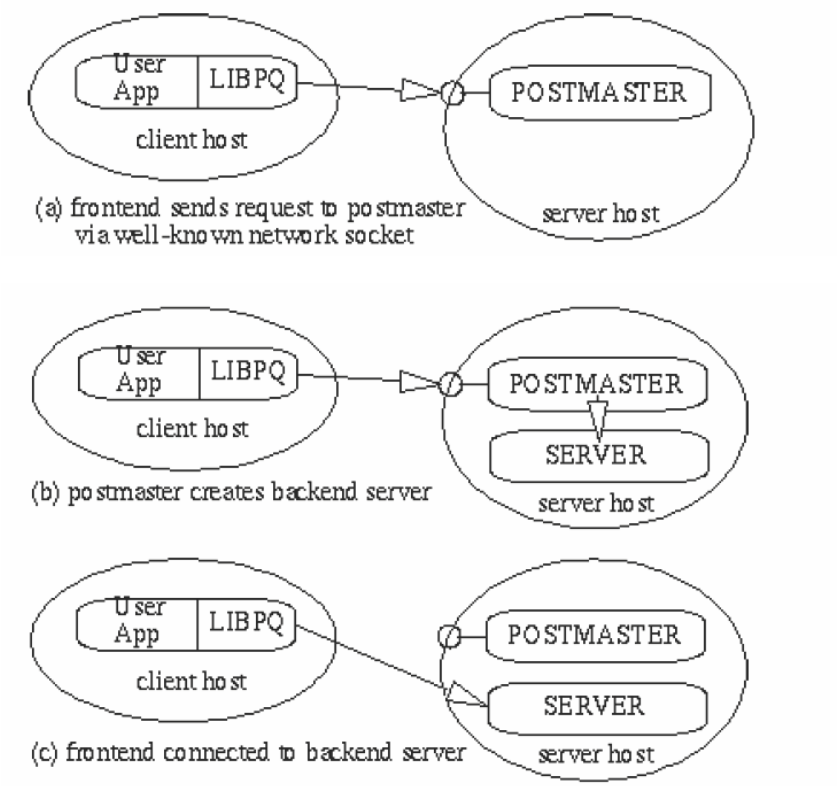
۳-۲-۱ معماری

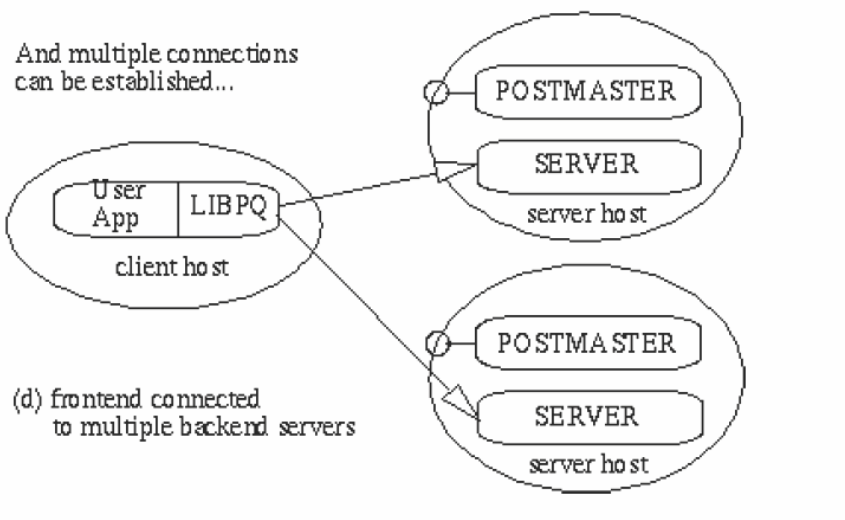
قبل از اینکه شروع کنیم، شما باید معماری سیستم PostgreSQL را فهمیده باشید. فهمیدن بخش‌های PostgreSQL ، درک قسمت بعد را آسانتر می‌سازد. نشست PostgreSQL شامل برنامه‌های زیر می‌باشد:

۱- یک پردازه نظارتی (postmaster)

۲- برنامه کاربردی user's frontend

۳- یک یا چند کارساز backend database





۳-۲-۲ مرور

در این بخش، ما به این بحث می‌پردازیم که چگونه می‌توان با اضافه کردن بخش‌های زیر زبان پرس‌وجوی SQL، PostgreSQL را گسترش داد:

۱- توابع

۲- انواع داده

۳- عملگرها

۴- اتحاد

۳-۲-۳ توابع

PostgreSQL چهار نوع تابع را در اختیار گذاشته است:

۱- توابع زبان پرس‌وجو (توابع نوشته شده در SQL)

۲- توابع زبان رویه‌ای (به‌طورمثال توابع در PL/TCL یا PL/pgSQL)

۳- توابع داخلی

۴- توابع زبان C

توابع زبان پرس‌وجو

مثال: برای شرح دادن یک تابع SQL ساده، از مثال زیر استفاده شده که ممکن است برای یک حساب بانک مورد استفاده قرار بگیرد:

```
CREATE FUNCTION tp1 (integer, numeric) RETURNS integer AS '
UPDATE bank
SET balance = balance - $2
WHERE accountno = $1;
SELECT 1;
' LANGUAGE SQL;
```

یک کاربر می‌تواند این تابع را برای حساب بدهی ۱۷ با ۱۰۰،۰۰۰ دلار اجرا کند:

```
SELECT tp1(17, 100.0);
```

در تمرین یک احتمالاً تمایل گرفتن نتایج سودمند بیشتری از تابع غیر از ثابت "۱" داریم، پس یک تعریف بهتر بصورت زیر می‌باشد:

```
CREATE FUNCTION tp1 (integer, numeric) RETURNS numeric AS '
UPDATE bank
SET balance = balance - $2
WHERE accountno = $1;
SELECT balance FROM bank WHERE accountno = $1;
' LANGUAGE SQL;
```

توابع زبان رویه‌ای

زبان‌های رویه‌ای در کارساز PostgreSQL ساخته نشده است، این زبان‌ها با استفاده از پیمانه‌های قابل بارگیری عرضه می‌شوند. چهار زبان رویه‌ای درتقسیم PostgreSQL استاندارد در دسترس می‌باشد:PL/pgSQL ، PL/Tcl ، PL/Perl و PL/Python . زبان‌های دیگر می‌تواند به وسیله کاربران تعریف شود.

توابع داخلی

توابع داخلی توابعی هستند که به زبان C نوشته شده‌اندو بصورت ایستا به کارساز PostgreSQL پیوند زده شده‌اند. بدنه تعریف تابع ، نام تابع را مشخص می‌کند که نباید یکی از اسم‌هایی باشد که SQL استفاده می‌کند. بطور عادی تمام توابع داخلی که در backend حاضر می‌شوند در مدت مقداردهی آغازی خوشه (cluster) پایگاه داده تعریف شده‌اند، اما یک کاربر می‌تواند از CREATE FUNCTION برای ساخت نام‌های مستعار اضافی برای تابع داخلی استفاده کند. توابع داخلی در CREATE FUNCTION با نام زبان داخلی تعریف می شود. بطور مثال برای ساخت یک مستعار برای تابع sqrt بصورت زیر عمل می‌کنیم:

```
CREATE FUNCTION square_root(double precision) RETURNS double precision
AS 'dsqrt'
LANGUAGE INTERNAL
WITH (isStrict);
```

(اکثر توابع داخلی انتظار می‌رود که بصورت "strict" تعریف شوند.)

توابع زبان C

توابع تعریف شده توسط کاربر می‌توانند در C نوشته شوند. این چنین توابع می‌توانند در اشیا قابل بارگیری پویا ترجمه شوند و بوسیله کارساز بار شوند.

۳-۲-۴ انواع

طبق توضیحات قبل ، دو گونه از انواع در PostgreSQL موجود می باشد: انواع پایه(تعریف شده در زبان برنامه نویسی)و انواع مرکب. در این قسمت چگونگی تعریف انواع پایه‌ای جدید توضیح داده شده است.

یک نوع تعریف شده توسط کاربر(user-defined) باید همیشه توابع ورودی و خروجی داشته باشند. این توابع تعیین می کنند که چطور نوع در رشته‌ها حاضر می‌شوند(برای ورودی به وسیله user و خروجی به user) و چطور نوع در حافظه سازمان‌دهی می‌شود. تابع ورودی یک رشته کاراکتری null-terminated را به عنوان ورودی گرفته و نوع نمایشی داخلی (در حافظه) را برمی‌گرداند. تابع خروجی یک نوع نمایش(representation) ورودی می‌گیرد و یک رشته کاراکتری null-terminate بر می‌گرداند. فرض کنید می‌خواهیم یک مجموعه نوع مجموعه اعداد را نمایش می‌دهند را تعریف کنیم. طبیعتاً ، ما انتخاب C structure زیر را برای نمایش یک مجموعه در حافظه:

```
typedef struct Complex {
double x;
double y;
} Complex;
```

و یک رشته از فرم (x,y) به عنوان نمایش رشته خارجی . توابع معمولاً برای نوشتن مشکل نیستند،بخصوص توابع خروجی. نکات زیر را به یاد داشته باشید:

۱- زمان تعریف نمایش رشته خارجی ، به خاطر داشته باشید که شما باید سرانجام یک تجزیه

کننده نیرومند و کامل برای نمایش تابع ورودی بنویسید

برای مثال:

For instance:

```
Complex *
complex_in(char *str)
{
double x, y;
Complex *result;
if (sscanf(str, " ( %lf , %lf )", &x, &y) != 2) {
elog(ERROR, "complex_in: error in parsing %s", str);
return NULL;
}
result = (Complex *)palloc(sizeof(Complex));
result->x = x;
result->y = y;
return (result);
}
```

تابع خروجی به سادگی می‌تواند بصورت زیر باشد:

```
char *
complex_out(Complex *complex)
{
char *result;
if (complex == NULL)
return(NULL);
result = (char *) palloc(60);
sprintf(result, "(%g,%g)", complex->x, complex->y);
return(result);
}
```

شما باید سعی کنید که توابع ورودی و خروجیتان را عکس یکدیگر بسازید.

برای تعریف یک نوع مجموعه‌ای، ما نیاز داریم دو تابع تعریف شده توسط کاربر complex_in و complex_out را قبل از ساخت نوع بسازیم:

```
CREATE FUNCTION complex_in(cstring)
RETURNS complex
AS 'PGROOT/tutorial/complex'
LANGUAGE C;
CREATE FUNCTION complex_out(complex)
RETURNS cstring
AS 'PGROOT/tutorial/complex'
LANGUAGE C;
```

در آخر ما نوع را تعریف می‌کنیم:

```
CREATE TYPE complex (
internallength = 16,
input = complex_in,
output = complex_out
);
```

۳-۲-۵ عملگر

PostgreSQL از left unary ، right unary و عملگرهـای دودویـی پشـتیبانی میکنـد. عملگرهـا میتوانند بار اضافی بگیرند، که همان نام عملگر می باشد که می‌تواند بـرای عملگرهـای مختلفـی کـه اعداد و انواع عملوندها را دارا می‌باشند استفاده شود. اگر یک شکل مبهم وجود داشته باشد و سیستم نتواند عملگر صحیح را برای استفاده انتخاب کند، یک پیغام خطا بازگردانی می شود.شما ممکن است مجبور شوید از type-cast کنید عملوندهای چپ و/یا راست برای کمک به اینکه بفهمید منظور شـما کدام عملگر برای استفاده می باشد.

هر عملگر "syntactic sugar" برای فراخوانی یک تابع underlying می‌باشـد کـه کـار واقعـی را انجام می‌دهد، پس شما باید قبل از ساخت عملگر، تابع underlying را بسازید. بـه هـر صـورت، یـک عملگر صرفاً syntactic sugar نمی باشد، زیرا این اطلاعات اظافی را که می‌تواند به طـراح پـرس‌وجـو برای بهینه سازی کردن پرس‌وجوهایی که عملگر استفاده می‌کند، کمک می‌کند.

مثال: اینجا یک مثال برای ساخت یک عملگر که دو مجموعه عدد را بـا یکـدیگر جمـع مـی‌کنـد می‌باشد. فرض می‌کنیم ما همچنین تعریف مجموعه انواع را نیز انجام داده‌ایم .در ابتدا ما تابعی نیـاز داریم که کار انجام دهد وآنگاه می‌توانیم عملگر تعریف کنیم:

```
CREATE FUNCTION complex_add(complex, complex)
RETURNS complex
AS 'PGROOT/tutorial/complex'
LANGUAGE C;
CREATE OPERATOR + (
leftarg = complex,
rightarg = complex,
procedure = complex_add,
commutator = +
```

```
);
```

حالا می‌توانیم کار زیر را انجام دهیم:

```
CREATE FUNCTION complex_add(complex, complex)
RETURNS complex
AS 'PGROOT/tutorial/complex'
LANGUAGE C;
CREATE OPERATOR + (
leftarg = complex,
rightarg = complex,
procedure = complex_add,
commutator = +
);
```

۳-۲-۶ اتحاد

توابع اتحاد در PostgreSQL بعنوان مقادیر حالت و توابع انتقال حالت بیان شده‌اند. اتحاد می‌تواند در عبارات حالت اصلاح می‌شوند هر زمان که یک گزینه ورودی مورد جریان قرار بگیرد تعریف شود. برای تعریف یک تابع اتحاد جدید، یک نوع داده برای مقدار حالت انتخاب می شود، یک مقدار اولیه برای حالت و تابع انتقال حالت. تابع انتقال حالت فقط یک تابع معمولی می باشد که می‌تواند خارج از متن اتحاد مورد استفاده قرار بگیرد.یک تابع نهایی می‌تواند بخصوص باشد، خروجی مورد انتظار اتحاد با داده ای که نیاز به نگهداری در هنگام اجرای حالت اجرا دارد متفاوت می باشد. بنابراین در مجموع ورودی و انواع داده نتیجه‌ای از نظر کاربر اتحاد، یک نوع داده مقدار حالت داخلی وجود دارد که ممکن است با هر دوی ورودی و انواع نتیجه متفاوت باشد. اگر ما یک اتحاد که از تابع نهایی استفاده نمی کند تعریف کنیم، ما اتحادی داریم که تابع اجرای مقادیر ستون از هر سطر را محاسبه می کند. sum مثالی از این نوع اتحاد است. sum از صفر شروع می‌شود و همیشه مقادیر فعلی را با مجموع در حال اجرای خود جمع می‌کند.برای مثال، اگر ما بخواهیم یک اتحاد جمع برای کار روی نوع داده برای اعداد مجموعه بسازیم، ما فقط به تابع جمع برای آن نوع داده احتیاج داریم. تعریف اتحاد بصورت زیر می‌باشد:

```
CREATE AGGREGATE complex_sum (
sfunc = complex_add,
basetype = complex,
stype = complex,
```

```
initcond = '(0,0)'
);
SELECT complex_sum(a) FROM test_complex;
complex_sum
-------------
(34,53.9)
```

۳-۲-۷ قوانین سیستم

قوانین سیستم مفهوماً ساده می باشند ، اما نکات ریزی در استفاده از آنها وجود دارد. تعـدادی از این نکات و ساختار تئـوری سیسـتم قـوانین PostgreSQL مـی‌توانـد در On Rules ، Procedures ، Caching و Views یافت شود.

تعدادی از سیستم های پایگاه داده‌ای دیگر ، قوانین پایگاه داده فعال را تعریف می‌کنند.روال‌هـا و محرک‌ها (triggers) می‌توانند در PostgreSQL به عنوان توابع و triggers پیاده سـازی شـوند. یـک پرس‌وجو، سیستم قانون را بازنویسی می‌کند و در مجموع با روال‌های ذخیره شده و triggers متفاوت می باشد و پرس و جوها را برای گرفتن قوانین داخل ملاحظات اصلاح می کنند، آنگـاه پـرس‌وجـوی اطلاح شده به طراح پایگاه داده برای طرح‌ریزی و اجرا ارسال می شود و خیلی قدرتمنـد مـی‌باشـد و می‌تواند برای بسیاری از موارد از قبیل روالهای زبان پرس‌وجو، views و نسخه‌ها مـورد اسـتفاده قـرار بگیرد .

۳-۳ زبانهای رویه‌ای

در این بخش زبانهای رویه‌ای در دسترس برای توزیع PostgreSQL توضیح داده خواهد شد.

PostgreSQL به کاربران امکان اضافه کردن زبانهای برنامه‌نویسی جدید را جهت نوشتن توابـع و رویه‌ها می‌دهد، که به آنها زبانهای رویه‌ای[1] (PL) می‌گویند. در مورد یک تـابع یـا یـک رویـه تریگـر نوشته شده به یک زبان رویه‌ای، سرور پایگاه داده دانشی درباره چگونگی تفسیر متن منبع تابع ندارد.

---

۱. http://jakarta.apache.org/ant/index.html

[1] procedural languages

در عوض این وظیفه به گرداننده[1] ویژه‌ای فرستاده می‌شود که با جزئیات زبان آشنا است. این گرداننده تمامی کارهای مربوط به عملیات پارس، تجزیه و تحلیل نحوی، اجرا و غیره را خود انجام می‌دهد و یا به عنوان glue بین PostgreSQL و پیاده‌سازی موجود در زبان برنامه‌نویسی، خدمت نماید. خود این گرداننده یک تابع زبان برنامه‌نویسی ویژه است که به عنوان یک شیئ مشترک کامپایل شده و بنا بر تقاضا بارگذاری می‌شود.

۳-۳-۱ زبان رویه‌ای PL/pgSQL – SQL

زبان PL/pgSQL یک زبان رویه‌ای قابل بارگذاری برای سیستم پایگاه داده PostgreSQL می‌باشد. هدف از طراحی PL/pgSQL، ایجاد زبان رویه‌ای قابل بارگذاری است که:

- جهت ایجاد توابع و رویه‌های تریگر قابل استفاده باشد
- ساختارهای کنترلی را به زبان SQL اضافه کند
- بتواند محاسبات پیچیده را انجام دهد
- تمامی انواع، توابع و عملگرهای تعریف شده توسط کاربر را به ارث برد
- به طریقی که برای سرور قابل اعتماد باشد تعریف گردد
- استفاده از آن ساده و آسان باشد

گرداننده PL/pgSQL فراخوانی شده متن منبع تابع را پارس کرده و یک درخت دستورالعمل دودویی داخلی را در اولین باری که تابع فراخوانی می‌شود، تولید می‌کند. درخت دستورالعمل به طور کامل ساختار دستورات PL/pgSQL را ترجمه می‌کند اما عبارات منطقی خاص SQL و پرس‌وجوهای SQL استفاده شده در توابع فورا ترجمه نمی‌شوند.

---

۱. http://jakarta.apache.org/ant/index.html

[1] handler

چنانچه هر یک از عبارات و پرس‌وجوهای SQL برای اولین بار در تابع استفاده گردند، مفسر PL/pgSQL یک نقشه[1] اجرایی آماده ایجاد می‌نماید (با استفاده از توابع SPI_prepare و SPI_saveplan). در بازدید ثانویه آن عبارات یا پرس‌وجوها مجددا در نقشه استفاده می‌شوند. بنابراین تنها یک تابع با کدهای شرطی که شامل عبارات بسیاری است که نقشه اجرایی ممکن است نیاز داشته باشد، آماده شده و آن نقشه‌هایی را که واقعا در طول ارتباط با پایگاه داده استفاده می‌شوند را ذخیره می‌نماید.

این امر می‌تواند اساسا میزان زمان کلی لازم جهت پارس را کاهش دهد و نقشه‌های پرس‌وجو برای عبارات موجود در تابع زبان رویه‌ای را تولید کند. اشکال این روش اینست که خطاهای موجود در یک عبارت یا پرس‌وجوی خاص، تا زمانیکه آن قسمت از تابع به مرحله اجرا نرسد، از بین نمی‌رود.

هنگامی که PL/pgSQL یکبار نقشه پرس‌وجو برای پرس‌وجوی خاصی در یک تابع می‌سازد، مجددا از آن نقشه در طول زمان برقراری ارتباط با پایگاه داده استفاده می‌کند. این امر سبب بهبود کارایی می‌گردد، اما اگر به صورت پویا شِمای پایگاه داده خود را تغییر دهید، ممکن است مشکلاتی پیش آید. برای مثال:

```
CREATE FUNCTION populate() RETURNS INTEGER AS '
DECLARE
-- Declarations
BEGIN
PERFORM my_function();
END;
' LANGUAGE 'plpgsql';
```

در صورتیکه تابع بالا را اجرا نمایید، به OID مربوط به my_function() در نقشه پرس‌وجوی ساخته شده برای عبارت PERFORM ارجاع داده می‌شود. چنانچه بعدا شما my_function() را drop کرده و مجددا آن را بسازید، آنگاه populate() نمی‌تواند my_function() را پیدا کند. در این

---

۱. http://jakarta.apache.org/ant/index.html

[1] plan

صورت شما باید مجددا ()populate را ایجاد کنید و یا یک session جدید برای پایگاه داده را آغاز کنید و در نتیجه باید آن را از نو کامپایل نمایید.

از آنجا که PL/pgSQL بدین طریق نقشه‌های اجرایی را ذخیره می‌کند، پرس‌وجوهایی که به صورت مستقیم در تابع PL/pgSQL ظاهر می‌شوند باید به جداول و فیلدهای مشابهی در هر اجرا مراجعه کنند؛ از این رو نمی‌توانید پارامتری با همان نام جدول یا فیلد در پرس‌وجو استفاده کنید. برای دوری از این محدودیت، می‌توانید پرس‌وجوهای پویای خود را طوری طرح‌ریزی کنید که از عبارت EXECUTE استفاده کنند (البته با این هزینه که برای هر اجرا یک نقشه پرس‌وجوی جدید ایجاد نمایید).

مزایای استفاده از PL/pgSQL عبارتند از:

- کارایی بالاتر
- پشتیبانی از SQL
- قابلیت انتقال و جابجایی

۳-۳-۲ زبان رویه‌ای PL/Tcl – Tcl

PL/Tcl یک زبان رویه‌ای قابل بارگذاری برای پایگاه داده PostgreSQL می‌باشد که زبان Tcl را قادر می‌سازد برای نوشتن توابع و رویه‌های تریگر استفاده شود.

PL/Tcl بیشتر امکاناتی را که یک نویسنده تابع در زبان C در اختیار دارد، ارائه می‌دهد، به استثنای تعداد کمی محدودیت.

یک محدودیت خوب اینست که همه چیز در یک مفسر ایمن Tcl اجرا می‌شود. به علاوه در مجموعه دستورات محدود شده Tcl ایمن، تنها تعداد کمی از دستورات جهت دسترسی به پایگاه داده از طریق SPI و تولید پیغام از طریق ()elog، در دسترس می‌باشند. هیچ راهی برای دسترسی به داخل پایگاه داده backend و یا دسترسی به OS-level، تحت مجوزهای user ID، PostgreSQL وجود

ندارد، آنگونه که توابع C می‌توانند. از این رو هیچ کاربر غیر ویژه پایگاه داده، امکان استفاده از این زبان را دارد.

دیگر محدودیت پیاده‌سازی شده این است که رویه‌های Tcl نمی‌توانند جهت ایجاد توابع ورودی/خروجی برای انواع داده‌ای جدید استفاده شوند.

گاهی اوقات لازم است توابعی نوشته شوند که به Tcl ایمن، محدود نشده باشند. مثلا یک نفر ممکن است به تابع Tcl که mail ارسال می‌کند، نیاز داشته باشد. در چنین مواردی گزینه‌های متعددی از PL/Tcl وجود دارند که PL/TclU نامیده می‌شوند (untrusted Tcl). این دقیقا همان زبان می‌باشد به جز اینکه یک مفسر Tcl کامل استفاده می‌شود. چنانچه PL/TclU استفاده می‌گردد، باید به عنوان یک زبان رویه‌ای غیر قابل اعتماد نصب گردد که تنها superuser ها امکان ایجاد تابع در آن را دارند. نویسنده تابع PL/TclU باید توجه داشته باشد که تابع نمی‌تواند هیچ کاری انجام دهد مگر اینکه کاربر به عنوان یک راهبر پایگاه داده وارد شود.

۳-۳-۳ زبان رویه‌ای PL/Perl

PL/Perl یک زبان رویه‌ای دیگر است که به شما امکان نوشتن توابع PostgreSQL را به زبان برنامه‌نویسی Perl می‌دهد.

برای نصب PL/Perl در پایگاه داده باید از createlang plperl dbname استفاده نمایید. توجه به این نکته مهم است که چنانچه یک زبان در template1 نصب شده باشد، تمامی پایگاههای داده‌ای که پس از این ایجاد گردند، به صورت خودکار این زبان را خواهند داشت.

معمولا PL/Perl به عنوان یک زبان برنامه‌نویسی قابل اعتماد، تحت نام plperl نصب می‌گردد. در این روش نصب برای حفظ امنیت، برخی از عملیاتهای Perl غیر فعال می‌گردند. به صورت کلی عملیاتهایی که محدود می‌شود آنهایی هستند که با محیط در تعاملند. . هیچ راهی برای دسترسی به داخل پایگاه داده backend و یا دسترسی به OS-level، تحت مجوزهای user ID، PostgreSQL

وجود ندارد، آنگونه که توابع C می‌توانند. از این رو هیچ کاربر غیر ویژه پایگاه داده، امکان استفاده از این زبان را دارد.

ویژگیهای مطرح شده در زیر در حال حاضر در PL/Perl موجود نمی‌باشند و ممکن است مورد توجه علاقمندان جهت مشارکت واقع گردند:

- توابع PL/Perl نمی‌توانند مستقیما یکدیگر را فراخوانی کنند (زیرا آنها زیرروالهای بی‌نام در Perl می‌باشند). در حال حاضر راهی برای آنها جهت به اشتراک گذاشتن متغیرهای سراسری وجود ندارد.
- PL/Perl را نمی‌توان برای نوشتن توابع تریگر استفاده کرد.
- DBD::PgSPI و یا قابلیتهای مشابه باید در توزیع استاندارد PostgreSQL جمع گردند.

۴-۳-۳ زبان رویه‌ای PL/Python

PL/Python زبان رویه‌ای است امکان نوشتن توابع PostgreSQL به زبان برنامه‌نویسی Python را می‌دهد.

نسخه موجود PL/Python در حال حاضر تنها به عنوان یک زبان قابل اعتماد عمل می‌نماید؛ دسترسی به فایل سیستم و دیگر منابع محلی غیرممکن است. به ویژه PL/Python از محیط اجرایی محدود Python استفاده می‌کند، که این خود سبب اعمال محدودیت بیشتر در استفاده از فراخوانی open فایل می‌گردد و فقط به ماژولهایی از یک لیست ویژه اجازه ورود می‌دهد. این لیست در حال حاضر شامل موارد زیر می‌باشد:

array ،bisect ،binascii ،calendar ،cmath ،codecs ،errno ،marshal ،math ،md5 ،mpz ، operator ،pcre ،pickle ،random ،re ،regex ،sre ،sha ،string ،StringIO ،struct ،time ، whrandom و zlib..

۴- راهنما برای توسعه دهندگان PostgreSQL

۴-۱ قالب کد منبع PostgreSQL

قالب کد منبع چهار ستون و چند tab به عنوان رزرو به کار می‌برد. برای Emacs، قسمتهای زیر را به فایل emacs./~ خود اضافه کنید:

```
;; check for files with a path containing "postgres" or "pgsql"
(setq auto-mode-alist
(cons '("\\(postgres\\|pgsql\\).*\\.[ch]\\'" . pgsql-c-mode)
auto-mode-alist))
(setq auto-mode-alist
(cons '("\\(postgres\\|pgsql\\).*\\.cc\\'" . pgsql-c-mode)
auto-mode-alist))
(defun pgsql-c-mode ()
;; sets up formatting for PostgreSQL C code
(interactive)
(c-mode)
(setq-default tab-width 4)
(c-set-style "bsd") ; set c-basic-offset to 4, plus other stuff
(c-set-offset 'case-label '+) ; tweak case indent to match PG custom
(setq indent-tabs-mode t)) ; make sure we keep tabs when indenting
```

برای vi، vimrc./~ یا فایل مشابه آن باید شامل موارد زیر باشد:

```
set tabstop=4
```

و یا معادل vi:

```
:set ts=4
```

۴-۲ مروری بر دید داخلی PostgreSQL

این فصل دیدی کلی از ساختار داخلی backend، PostgreSQL ارائه می‌دهد. بعد از خواندن این بخش شما ایده‌ای کامل در رابطه با چگونگی عملکرد و پردازش یک پرس و جو خواهید داشت. از آنجا که توضیح کامل و پر جزئیات ساختار داده و توابع مورد استفاده به همراه PostgreSQL بسیار حجیم بوده و شاید بیش از ۱۰۰۰ صفحه نیاز داشته باشد، در اینجا از پرداختن به جزئیات خودداری کرده‌ایم. هدف این فصل، کمک به درک کنترل عمومی و جریان داده مطابق بر backend، از زمان دریافت یک پرس و جو تا ارسال نتایج می‌باشد.

۴-۲-۱ شاخه های پرسوجو

در این بخش مروری کوتاه بر مراحل پردازش یک پرس و جو تا به دست آمدن نتایج خواهیم داشت.

۱. ارتباطی از طریق یک برنامه کاربردی با سرور PostgreSQL برقرار می‌گردد. این برنامه کاربردی پرس و جویی را به سرور ارسال نموده و نتایج را از آن دریافت می‌نماید.
۲. در مرحله پارسر، پرس و جوی ارسال شده از طریق برنامه کاربردی (سرویس گیرنده۱) از نظر درستی نحوی۲ دستورات بررسی می‌گردد و یک درخت پرس و جو۳ ایجاد می‌شود.
۳. سیستم بازنویسی۴ درخت پرس و جوی ساخته شده توسط مرحله پارسر را گرفته و در بین قوانین ذخیره شده در کاتالوگ سیستم جستجو می‌کند تا قانون لازم را بر درخت پرس و جو اعمال کند و تبدیلات موجود در بدنه این قوانین را اجرا می‌نماید. یک کاربرد سیستم بازنویسی در درک و تحقق دیدها می‌باشد.

هر زمان که یک پرس و جو بر خلاف یک دید ساخته شود، سیستم بازنویسی پرس و جوی مورد درخواست کاربر را مطابق بر تعاریف موجود در دید و به طریقی که به جداول پایه دسترسی داشته باشد، مجددا بازنویسی می‌کند.

۴. برنامه‌ساز/بهینه‌ساز، درخت پرس و جوی بازنویسی شده را گرفته و یک queryplan می‌سازد که به عنوان ورودی اجراکننده۵ خواهد بود.

در این قسمت ابتدا تمامی مسیرهای ممکنی که به نتیجه مورد نظر ختم می‌شوند، ایجاد می‌گردند. برای مثال اگر قرار است یک ایندکس موجود بر روی یک ارتباط، بررسی و پویش۶ شود،

---

۱. http://jakarta.apache.org/ant/index.html

[1] client
2 syntax
3 query tree
[4] rewrite system
5 executor
[6] Scan

دو راه برای بررسی وجود خواهد داشت. یک امکان بررسی و پویش ترتیبی و امکان دیگر به کار گرفتن ایندکس است. سپس هزینه اجرای هر کدام تخمین زده می‌شود و مسیری که کمترین هزینه را دارد انتخاب می‌شود.

۵. اجراکننده مراحل را به صورت بازگشتی از میان plan tree طی می‌کند و تاپلها را با توجه به برنامه بازیابی می‌نماید. اجراکننده در حین پویش ارتباطات از سیستم ذخیره‌سازی استفاده می‌کند، مرتب‌سازیها و پیوندها را انجام می‌دهد، شرایط لازم را بررسی کرده و در آخر تاپلهای مربوطه را برمی‌گرداند.

در ادامه ما تمامی موارد ذکر شده در بالا را با جزئیات بیشتر توضیح خواهیم داد تا درک بهتری از کنترل داخلی و ساختار داده PostgreSQL ارائه دهیم.

۴-۲-۲ چگونگی ایجاد ارتباطات

اجرای PostgreSQL در حقیقت از مدل «مشتری/خدمتگذار» استفاده می‌کند. در این مدل یک پروسه مشتری۱ مشخص وجود دارد که دقیقا به یک پروسه خدمتگذار۲ متصل می‌شود. از آنجا که ما از تعداد اتصالات بخودی خود اطلاع نداریم، باید از یک پروسه ارشد۳ استفاده می‌کنیم که در زمان رسیدن هر درخواست برقراری ارتباط، یک پروسه خدمتگذار تولید کند. این پروسه ارشد، postmaster نامیده می‌شود و به پورت TCP/IP مشخص شده برای ارتباطات وارد شونده، گوش می‌دهد. هر زمان که یک درخواست ارتباط مشخص شود، postmaster یک پروسس خدمتگذار جدید به نام postgres تولید می‌کند. پروسه‌های postgres از طریق سمافورها و حافظه اشتراکی با یکدیگر ارتباط برقرار می‌کنند تا از جامعیت داده‌ای در تمام مدت دسترسی به داده اطمینان حاصل شود.

---

۱. http://jakarta.apache.org/ant/index.html

1 client process
2 server process
3 master process

شـکل {connection}\ref\ تعامـل بـین پروسـه ارشـد postmaster پروسـه خدمتگـذار postgres و برنامه‌کاربردی مشتری را نشان می‌دهد.

پروسه مشتری می‌تواند نرم‌افزار نهایی psql ۱ (در پرس و جوهای محاوره‌ای۲) یا هر برنامه‌کابردی کاربر که از کتابخانه libpg استفاده می‌کند باشد. توجه داشته باشید که برنامـه‌هـای کـاربردی کـه از ecpg استفاده می‌کنند نیز این کتابخانه را به کار می‌گیرند.

با برقراری ارتباط پروسه مشتری می‌تواند یک پـرس و جـو را بـه backend (خدمتگـذار) ارسـال نماید. پرس و جوی ارسال شده، متن واضح و آشکار را استفاده می‌کند؛ هیچ عمـل پارسـی در سـمت frontend (مشتری) انجام نمی‌شود. خدمتگذار پرس و جو را پارس می‌نمایـد، یـک طـرح قابـل اجـرا ایجاد می‌کند، طرح را اجرا کرده و تاپلهای مربوطه را بـه مشـتری از طریـق ارتبـاط موجـود، ارسـال می‌کند.

۴-۲-۳ مرحله پارسر

مرحله پارسر از دو قسمت تشکیل شده است:

- پارسر تعریف شده در gram.y و scan.l که از ابزارهای یونیکس yacc و lex استفاده می‌کند.
- پروسه تبدیل و انتقال که تغییر وتبدیلات لازم برای تقویت ساختارهای داده‌ای که توسط پارسر برمی‌گردد را انجام می‌دهد.

پیش از این چگونگی ایجاد یک درخـت پـارس۳ توضـیح داده شـد؛ بـرای انجـام آن از ابزارهـای شناخته شده lex و yacc یونیکس استفاده می‌شود. در فایل scan.l، lexer تعریف شـده کـه مسـئول تعیین شناسه‌ها، لغات کلیدی SQL و غیره است. برای هر لغت کلیدی یا شناسه‌ای که پیدا می‌شـود، یک توکن ایجاد و به پارسر داده می‌شود.

---

۱. http://jakarta.apache.org/ant/index.html

[1] frontend
[2] interactive
[3] parse tree

پارسر در فایل gram.y تعریف گردیده و شامل مجموعه‌ای از قوانین گرامر و اعمالی است که در زمان اجرای دستورات اجرا می‌شوند. کد این اعمال (که در حقیقت به زبان C است) برای ساختن درخت پارس کاربرد دارند.

فایل scan.l به فایل منبع C تبدیل شده و از برنامه lex استفاده می‌کند و gram.y با استفاده از yacc به gram.c تبدیل شده است. پس از انجام این تبدیلات، یک کامپایلر C معمولی می‌تواند برای ساختن پارسر به کار گرفته شود.

توجه: تبدیلات، کامپایلها و جمع‌آوری اطلاعات ذکر شده در بالا، معمولا به صورت خودکار و با استفاده از makefiles موجود در منبع PostgreSQL انجام می‌گیرند.

توضیح جزئیات بیشتر yacc و قوانین گرامر موجود در gram.y فراتر از سطح این کتاب می‌باشد. کتابها و منابع زیادی در رابطه با lex و yacc وجود دارند. پیش از آنکه مطالعه گرامر موجود در gram.y را آغاز کنید لازم است با yacc آشنایی داشته باشید، در غیر این صورت متوجه نخواهید شد که چه اتفاقاتی در آنجا رخ می‌دهد.

به منظور درک بهتر ساختار داده مورد استفاده در PostgreSQL برای پردازش یک پرس و جو در زیر مثالی آورده‌ایم. مثال زیر شامل پرس و جوی ساده‌ای است که در قسمتهای بعدی نیز از آن استفاده خواهیم کرد. در این مثال فرض بر این است که جداول داده شده در پایگاه داده Supplier قبلا تعریف شده است.

مثال ۲-۱: یک select ساده

```
select s.sname, se.pno
from supplier s, sells se
where s.sno > 2 and s.sno = se.sno;
```

شکل {parsetree}\ref یک درخت پارس ساخته شده توسط قوانین گرامر و اعمال داده شده در gram.y را برای پرس و جوی مثال ۲-۱ نشان می‌دهد (بدون درخت عامل۱ برای کلاز where که در شکل {where_clause}\ref نشان داده شده است، زیرا فضای کافی برای نمایش هر دو ساختار داده در یک شکل وجود ندارد).

بالاترین گره در درخت گره SelectStmt است. برای هر مدخلی که در کلاز from پرس و جوی SQL ظاهر می‌شود، یک گره RangeVar که alias نامیده شده و یک اشاره‌گر به گره RelExpr با نام relation ایجاد می‌گردد. تمامی گره‌های RangeVar در لیستی که به فیلد fromClause گره SelectStmt پیوست می‌شود، جمع گردیده‌اند.

برای هر مدخل ظاهر شده در لیست select پرس و جوی SQL یک گره ResTarget با یک اشاره‌گر به گره Attr ساخته می‌شود. تمامی گره‌های ResTarget در لیستی که به فیلد targetList گره SelectStmt وصل است، جمع گردیده‌اند.

شکل {where_clause}\ref نشان داده شده در درخت عامل برای کلاز where پرس و جوی SQL داده شده در مثال ۲-۱ ساخته شده که به فیلد qual گره SelectStmt متصل شده است. بالاترین گره درخت عامل، نماینده گره A_Expr و عمل AND می‌باشد. این گره دارای دو راس جانشین به نامهای lexpr و rexpr می‌باشد که به دو زیر درخت اشاره می‌کنند. یکی از زیر درختها به lexpr متصل بوده و بیانگر شرط s.sno > 2 است و زیر درخت دیگر، متصل به rexpr و بیانگر s.sno = se.sno می‌باشد.

پروسه تبدیل و انتقال درختی که قرار است توسط پارسر برگردانده شود را به عنوان ورودی گرفته و به صورت بازگشتی آن را پیمایش می‌کند. چنانچه گره SelectStmt پیدا شود، به یک گره Query و

---

۱. http://jakarta.apache.org/ant/index.html

[1] operator tree

بالاترین گره در ساختار داده جدید تبدیل می‌گردد. شکل \ref{transformed} ساختار داده تغییر یافته را نشان می‌دهد.

حال یک بررسی صورت می‌گیرد تا مشخص شود آیا نامهای relation در کلاز FROM به سیستم شناسانده شده‌اند یا نه. برای هر نام relation که در کاتالوگ سیستم موجودند، یک گره RTE شامل نام relation، نام alias و relation id ایجاد می‌گردد. از اینجا به بعد relation idها برای رجوع به relationهای داده شده در پرس و جو استفاده می‌شوند. تمامی گره‌های RTE در لیست مدخلهای جدول range که به فیلد rtable گره Query متصل است، جمع‌آوری شده‌اند. چنانچه یک نام relation در پرس و جو برای سیستم ناشناخته باشد، یک خطا رخ می‌دهد و پردازش پرس و جو متوقف می‌گردد.

سپس بررسی می‌شود که آیا نام صفات استفاده شده در رابطه‌های داده شده در پرس و جو موجود هستند یا نه. برای هر صفتی که پیدا می‌شود، یک گره TLE به همراه یک اشاره‌گر به گره Resdom (که نام ستون می‌باشد) و یک اشاره‌گر به گره VAR ایجاد می‌گردد.

۴-۲-۴ سیستم قوانین PostgreSQL

PostgreSQL از سیستم قوانین قدرتمندی جهت تعیین ویژگیهای دیدها و ابهامات به روزرسانی دید، پشتیبانی می‌کند. اصولا سیستم قوانین PostgreSQL دو پیاده‌سازی (کاربرد) دارد:

- یک عملکرد از پردازش سطحی تاپلها استفاده کرده و در اجراکننده به صورت عمقی عمل می‌نماید. سیستم قواعد در هر زمان که دستیابی به یک تاپل منحصر به فرد مورد نظر باشد، فراخوانی می‌گردید. این پیاده‌سازی در سال ۱۹۹۵ برچیده شد.
- پیاده‌سازی دوم سیستم قواعد، تکنیکی است که بازنویسی مجدد پرس و جو[1] نامیده می‌شود. سیستم بازنویسی ماژولی است که بین مرحله پارسر و بهینه‌ساز واقع شده است. این روش

---

۱. http://jakarta.apache.org/ant/index.html

[1] query rewriting

همچنان در حال استفاده است.

برای کسب اطلاع از دستور نحو و اطمینان از قوانین در سیستم PostgreSQL، به راهنمای کاربر PostgreSQL مراجعه نمایید.

۴-۲-۵ بهینه‌ساز

وظیفه بهینه‌ساز ایجاد یک طرح اجرایی بهینه است. بهینه‌ساز ابتدا تمامی راههای ممکن پویش و پیوند رابطه‌های ظاهر شده در پرس و جو را ترکیب می‌کند. همه مسیرهای تولید شده به یک نتیجه می‌رسند و وظیفه بهینه‌ساز برآورد هزینه هر مسیر و پیدا کردن کم هزینه‌ترین آنها می‌باشد.

بهینه‌ساز بر اساس انواع ایندکسهای تعریف شده روی رابطه‌های ظاهر شده در پرس و جو، تصمیم می‌گیرد که چه طرحهایی باید ایجاد گردند. همیشه احتمال ایجاد پیمایش ترتیبی رابطه وجود دارد؛ بنابراین همواره یک طرح که فقط از پیمایش ترتیبی استفاده می‌کند، ایجاد می‌گردد.

بعد از آنکه تمام طرحهای ممکن و عملی جهت پویش رابطه‌های واحد پیدا شدند، طرحهایی برای پیوند رابطه‌ها ایجاد می‌گردد. بهینه‌ساز بررسی پیوندهای بین هر دو رابطه‌ای که در شرط where متناظر هم باشند را آغاز می‌نماید (برای هر محدودیت مشابه where rel1.attr1=rel2.attr2 وجود داشته باشد).

سه پیوند ممکن عبارتند از:

- پیوند تکرار تودرتو[1]: رابطه سمت راست یکبار برای هر تاپل پیدا شده در رابطه سمت چپ پویش می‌شود. پیاده‌سازی این راهبرد ساده اما بسیار وقت‌گیر است.
- پیوند با مرتب‌سازی ادغام[2]: پیش از شروع پیوند، هر رابطه در صفات پیوند ذخیره می‌گردد. سپس دو رابطه با صفات مرتب شده در پیوند با یکدیگر ادغام می‌شوند. این روش پیوند بهتر

---

۱. http://jakarta.apache.org/ant/index.html

[1] nested iteration join
[2] merge sort join

است زیرا هر رابطه تنها یکبار پویش می‌شود.

- پیوند درهم[1]: ابتدا رابطه سمت راست به صورت نامرتب بر روی صفات پیوند تنظیم می‌شود. سپس رابطه سمت چپ پویش شده و ارزشهای مناسب هر تاپل به عنوان کلید hash برای تعیین مکان تاپلها در رابطه سمت راست، ایجاد می‌شوند.

۴-۲-۶ اجراکننده

اجراکننده طرح برگردانده شده توسط بهینه‌ساز را گرفته و شروع به پردازش بالاترین گره می‌کند. در مثال ما بالاترین گره، گره Merge-Join می‌باشد.

پیش از هر ادغام دو تاپل باید واکشی شوند. بنابراین اجراکننده، جهت پردازش subplanها، به صورت بازگشتی خود را فراخوانی می‌کند (این عمل با subplan پیوست شده به lefttree آغاز می‌شود). گره بالایی جدید (گره بالایی subplan سمت چپ) یک گره SeqScan است و مجددا پیش از آنکه خود گره پردازش شود، یک تاپل باید واکشی شود. اجراکننده یکبار دیگر خود را برای پردازش subplan پیوست شده به lefttree گره SeqScan، به صورت بازگشتی فراخوانی می‌کند.

حال گره جدید، گره Sort می‌باشد. هر زمان که پردازش گره Sort نیاز به تاپلی داشته باشد، اجراکننده به صورت بازگشتی فراخوانی می‌شود. اگر تاپل با شرایط موجود مطابقت کند، برگردانده می‌شود؛ در غیر این صورت تاپل بعدی بررسی می‌گردد تا زمانیکه یک تاپل مناسب پیدا شود. در صورتی که تاپلی مطابق با شرایط یافت نشود، مقدار NULL برمی‌گردد.

۴-۳ کاتالوگهای سیستم

کاتالوگهای سیستم مکانی است که سیستم مدیریت پایگاه داده رابطه‌ای، شمای ابرداده، اطلاعاتی درباره جدولها و ستونها و همچنین اطلاعات ساماندهی داخلی را در آن ذخیره می‌نماید. کاتالوگهای سیستمی PostgreSQL معمولا به صورت جدول می‌باشند. شما می‌توانید یک جدول را حذف و

---

۱. http://jakarta.apache.org/ant/index.html

[1] hash join

مجددا ایجاد نمایید، ستونی اضافه کنید و یا اطلاعات جدید وارد و به روزرسانی کنید. به طور معمول ایجاد تغییرات دستی در کاتالوگ سیستم ممکن نیست، برای این امر دستورات SQL ویـژه‌ای وجـود دارد.

بیشتر کاتالوگهای سیستم در هنگام ایجـاد پایگـاه داده جدیـد، از یـک پایگـاه داده موقـت کپـی می‌شوند. تعداد کمی از کاتالوگها از نظر فیزیکی، بین همه پایگاه داده‌ها به اشتراک گذاشته می‌شوند.

Catalog Name	Purpose
pg_aggregate	aggregate functions
pg_am	index access methods
pg_amop	access method operators
pg_amproc	access method support procedures
pg_attrdef	column default values
pg_attribute	table columns (“attributes”, “fields”)
pg_cast	casts (data type conversions)
pg_class	tables, indexes, sequences (“relations”)
pg_constraint	check constraints, unique / primary key constraints, foreign key constraints
pg_conversion	encoding conversion information
pg_database	databases within this database cluster
pg_depend	dependencies between database objects
pg_description	descriptions or comments on database objects
pg_group	groups of database users
pg_index	additional index information
pg_inherits	table inheritance hierarchy
pg_language	languages for writing functions
pg_largeobject	large objects
pg_listener	asynchronous notification
pg_namespace	namespaces (schemas)
pg_opclass	index access method operator classes
pg_operator	operators
pg_proc	functions and procedures
pg_rewrite	query rewriter rules
pg_shadow	database users
pg_statistic	optimizer statistics
pg_trigger	triggers
pg_type	data types

جدول ۳-۱: کاتالوگهای سیستم

در ادامه توضیحات بیشتری درباره هر یک آمده است.

pg_aggregate ۱-۳-۴

pg_aggregate اطلاعاتی در رابطه با توابع جمعی ذخیره می‌کند. یک تابع جمعی، تابعی است که بر روی مجموعه‌ای از مقادیر عمل می‌کند (به ویژه یک ستون از هر سطر که با شرایط پرس و جو مطابقت دارد) و یک مقدار واحد از کلیه این مقادیر برمی‌گرداند. این توابع عبارتند از sum، count و max. توابع جمعی جدید با دستور CREATE AGGREGATE همراه می‌باشند.

Name	Type	References	Description
aggfnoid	regproc	pg_proc.oid	pg_proc OID of the aggregate function
aggtransfn	regproc	pg_proc.oid	Transition function
aggfinalfn	regproc	pg_proc.oid	Final function (zero if none)
aggtranstype	oid	pg_type.oid	The type of the aggregate function's internal transition (state) data
agginitval	text		The initial value of the transition state. This is a text field containing the initial value in its external string representation. If the field is NULL, the transition state value starts out NULL.

جدول ۳ -۲: ستونهای pg_aggregate

pg_am ۲-۳-۴

pg_am اطلاعاتی درباره روشهای دسترسی ایندکسها را ذخیره می‌کند. برای هر روش دسترسی به ایندکس یک سطر وجود دارد که توسط سیستم پشتیبانی می‌شود.

Name	Type	References	Description
amname	name		name of the access method
amowner	int4	pg_shadow.usesysid	user ID of the owner (currently not used)
amstrategies	int2		number of operator strategies for this access method
amsupport	int2		number of support routines for this access method
amorderstrategy	int2		zero if the index offers no sort order, otherwise the strategy number of the strategy operator that describes the sort order
amcanunique	bool		does AM support unique indexes?
amcanmulticol	bool		does AM support multicolumn indexes?
amindexnulls	bool		does AM support NULL index entries?
amconcurrent	bool		does AM support concurrent updates?
amgettuple	regproc	pg_proc.oid	“next valid tuple” function
aminsert	regproc	pg_proc.oid	“insert this tuple” function
ambeginscan	regproc	pg_proc.oid	“start new scan” function
amrescan	regproc	pg_proc.oid	“restart this scan” function
amendscan	regproc	pg_proc.oid	“end this scan” function
ammarkpos	regproc	pg_proc.oid	“mark current scan position” function
amrestrpos	regproc	pg_proc.oid	“restore marked scan position” function
ambuild	regproc	pg_proc.oid	“build new index” function
ambulkdelete	regproc	pg_proc.oid	bulk-delete function
amcostestimate	regproc	pg_proc.oid	estimate cost of an indexscan

جدول ۳ -۲: ستونهای pg_am

pg_amop ۳-۳-۴

pg_amop اطلاعاتی درباره عملگرهای شرکت‌پذیر با کلاس عملگرهای روش دستیابی ایندکس، ذخیره می‌کند. برای هر عملگر یک سطر وجود دارد که عضو یک کلاس عملگر می‌باشد.

Name	Type	References	Description
amopclaid	oid	pg_opclass.oid	the index opclass this entry is for
amopstrategy	int2		operator strategy number
amopreqcheck	bool		index hit must be rechecked
amopopr	oid	pg_operator.oid	the operator's pg_operator OID

جدول ۳-۳: ستونهای pg_amop

pg_amproc ۴-۳-۴

pg_amproc اطلاعاتی درباره رویه‌های پشتیبان شرکت‌پذیر به همراه روش دستیابی به کلاسهای عملگر ایندکس را ذخیره می‌کند. برای هر رویه پشتیبان وابسته به کلاس عملگر، یک سطر وجود دارد.

Name	Type	References	Description
amopclaid	oid	pg_opclass.oid	the index opclass this entry is for
amprocnum	int2		support procedure index
amproc	regproc	pg_proc.oid	OID of the proc

pg_attrdef ۵-۳-۴

در این کاتالوگ مقادیر پیش‌فرض ستونها ذخیره مـی‌گـردد. اطلاعـات اصـلی دربـاره سـتونها در pg_attribute ذخیره می‌گردد. تنها ستونهایی که در هنگام ایجاد جدول برایشان مقـدار پـیش‌فـرض تعیین شده است، در اینجا دیده می‌شوند.

Name	Type	References	Description
adrelid	oid	pg_class.oid	The table this column belongs to
adnum	int2	pg_attribute.attnum	The number of the column
adbin	text		An internal representation of the column default value
adsrc	text		A human-readable representation of the default value

pg_attribute ۶-۳-۴

pg_attribute اطلاعاتی درباره ستونهای جدول نگه می‌دارد. برای هر ستون از جداول موجـود در پایگاه داده، یک سطر در pg_attribute وجود خواهد داشت.

Name	Type	References	Description
attrelid	oid	pg_class.oid	The table this column belongs to
attname	name		Column name
atttypid	oid	pg_type.oid	The data type of this column

attstattarget	int4		attstattarget controls the level of detail of statistics accumulated for this column by ANALYZE. A zero value indicates that no statistics should be collected. A negative value says to use the system default statistics target. The exact meaning of positive values is datatype-dependent. For scalar datatypes, attstattarget is both the target number of "most common values" to collect, and the target number of histogram bins to create.
attlen	int2		This is a copy of pg_type.typlen of this column's type.
attnum	int2		The number of the column. Ordinary columns are numbered from 1 up. System columns, such as oid, have (arbitrary) negative numbers.
attndims	int4		Number of dimensions, if the column is an array type; otherwise 0. (Presently, the number of dimensions of an array is not enforced, so any nonzero value effectively means "it's an array".)
attcacheoff	int4		Always -1 in storage, but when loaded into a tuple descriptor in memory this may be updated to cache the offset of the attribute within the tuple.

atttypmod	int4		atttypmod records type-specific data supplied at table creation time (for example, the maximum length of a varchar column). It is passed to type-specific input functions and length coercion functions. The value will generally be -1 for types that do not need typmod.
attbyval	bool		A copy of pg_type.typbyval of this column's type
attstorage	char		Normally a copy of pg_type.typstorage of this column's type. For TOASTable datatypes, this can be altered after column creation to control storage policy.
attisset	bool		If true, this attribute is a set. In that case, what is really stored in the attribute is the OID of a tuple in the pg_proc catalog. The pg_proc tuple contains the query string that defines this set - i.e., the query to run to get the set. So the atttypid (see above) refers to the type returned by this query, but the actual length of this attribute is the length (size) of an oid. --- At least this is the theory. All this is probably quite broken these days.
attalign	char		A copy of pg_type.typalign of this column's type

attnotnull	bool		This represents a NOT NULL constraint. It is possible to change this field to enable or disable the constraint.
atthasdef	bool		This column has a default value, in which case there will be a corresponding entry in the pg_attrdef catalog that actually defines the value.
attisdropped	bool		This column has been dropped and is no longer valid. A dropped column is still physically present in the table, but is ignored by the parser and so cannot be accessed via SQL.
attislocal	bool		This column is defined locally in the relation. Note that a column may be locally defined and inherited simultaneously.
attinhcount	int4		The number of direct ancestors this column has. A column with a nonzero number of ancestors cannot be dropped nor renamed.

pg_cast ۷-۳-۴

pg_cast راه و روشهای تبدیل انواع داده را ذخیره می‌نماید.

Name	Type	References	Description
castsource	oid	pg_type.oid	OID of the source data type
casttarget	oid	pg_type.oid	OID of the target data type

Name	Type	References	Description
castfunc	oid	pg_proc.oid	The OID of the function to use to perform this cast. Zero is stored if the data types are binary coercible (that is, no run-time operation is needed to perform the cast).
castcontext	char		Indicates what contexts the cast may be invoked in. e means only as an explicit cast (using CAST, ::, or function-call syntax). a means implicitly in assignment to a target column, as well as explicitly. i means implicitly in expressions, as well as the other cases.

pg_class ۸-۳-۴

جداول و هر چیز دیگری که ستونهایی دارد و یا به طریقی مشابه جدول است را کاتالوگ می‌کند.

ایندکسها، ترتیبها، دیدها و برخی از رابطه‌های خاص دیگر، مثالهایی از این دسته‌اند.

Name	Type	References	Description
relname	name		Name of the table, index, view, etc.
relnamespace	oid	pg_namespace.oid	The OID of the namespace that contains this relation
reltype	oid	pg_type.oid	The OID of the data type that corresponds to this table, if any (zero for indexes, which have no pg_type entry)
relowner	int4	pg_shadow.usesysid	Owner of the relation
relam	oid	pg_am.oid	If this is an index, the access method used (B-tree, hash, etc.)

Name	Type	References	Description
relfilenode	oid		Name of the on-disk file of this relation; 0 if none
relpages	int4		Size of the on-disk representation of this table in pages (size BLCKSZ). This is only an estimate used by the planner. It is updated by VACUUM, ANALYZE, and CREATE INDEX.
reltuples	float4		Number of tuples in the table. This is only an estimate used by the planner. It is updated by VACUUM, ANALYZE, and CREATE INDEX.
reltoastrelid	oid	pg_class.oid	OID of the TOAST table associated with this table, 0 if none. The TOAST table stores large attributes "out of line" in a secondary table.
reltoastidxid	oid	pg_class.oid	For a TOAST table, the OID of its index. 0 if not a TOAST table.
relhasindex	bool		True if this is a table and it has (or recently had) any indexes. This is set by CREATE INDEX, but not cleared immediately by DROP INDEX. VACUUM clears relhasindex if it finds the table has no indexes.
relisshared	bool		True if this table is shared across all databases in the cluster. Only certain system catalogs (such as pg_database) are shared.

Name	Type	References	Description
relkind	char		'r' = ordinary table, 'i' = index, 'S' = sequence, 'v' = view, 'c' = composite type, 's' = special, 't' = TOAST table
relnatts	int2		Number of user columns in the relation (system columns not counted). There must be this many corresponding entries in pg_attribute. See also pg_attribute.attnum.
relchecks	int2		Number of check constraints on the table; see pg_constraint catalog
reltriggers	int2		Number of triggers on the table; see pg_trigger catalog
relukeys	int2		unused (*Not* the number of unique keys)
relfkeys	int2		unused (*Not* the number of foreign keys on the table)
relrefs	int2		unused
relhasoids	bool		True if we generate an OID for each row of the relation.
relhaspkey	bool		True if the table has (or once had) a primary key.
relhasrules	bool		Table has rules; see pg_rewrite catalog
relhassubclass	bool		At least one table inherits from this one
relacl	aclitem[]		Access permissions. See the descriptions of GRANT and REVOKE for details.

pg_constraint ۹-۳-۴

در این کاتالوگ سیستم قیود CHECK، PRIMARY KEY، UNIQUE و FOREIGN KEY روی جدولها ذخیره می‌شود.

نکته: قیود هیچ‌مقدار ناپذیری در کاتالوگ pg_attribute آورده می‌شوند.

Name	Type	References	Description
conname	name		Constraint name (not necessarily unique!)
connamespace	oid	pg_namespace.oid	The OID of the namespace that contains this constraint
contype	char		'c' = check constraint, 'f' = foreign key constraint, 'p' = primary key constraint, 'u' = unique constraint
condeferrable	boolean		Is the constraint deferrable?
condeferred	boolean		Is the constraint deferred by default?
conrelid	oid	pg_class.oid	The table this constraint is on; 0 if not a table constraint
contypid	oid	pg_type.oid	The domain this constraint is on; 0 if not a domain constraint
confrelid	oid	pg_class.oid	If a foreign key, the referenced table; else 0
confupdtype	char		Foreign key update action code
confdeltype	char		Foreign key deletion action code
confmatchtype	char		Foreign key match type
conkey	int2[]	pg_attribute.attnum	If a table constraint, list of columns which the constraint constrains
confkey	int2[]	pg_attribute.attnum	If a foreign key, list of the referenced columns
conbin	text		If a check constraint, an internal representation of the expression

Name	Type	References	Description
consrc	text		If a check constraint, a human-readable representation of the expression

pg_conversion ۱۰-۳-۴

در این کاتالوگ اطلاعات تبدیلات رمزگذاری ذخیره می‌شود.

Name	Type	References	Description
conname	name		Conversion name (unique within a namespace)
connamespace	oid	pg_namespace.oid	The OID of the namespace that contains this conversion
conowner	int4	pg_shadow.usesysid	Owner (creator) of the namespace
conforencoding	int4		Source(for) encoding ID
contoencoding	int4		Destination(to) encoding ID
conproc	regproc	pg_proc.oid	Conversion procedure
condefault	boolean		true if this is the default conversion

pg_database ۱۱-۳-۴

این کاتالوگ اطلاعاتی درباره پایگاه داده‌های موجود ذخیره می‌کند. پایگاه داده با دستور CREATE DATABASE ایجاد می‌شود. برای درک جزئیات بیشتر راهنمای Adminstrator را مطالعه نمایید.

بر خلاف بیشتر کاتالوگهای سیستم، pg_database بین پایگاه داده‌های یک خوشه[۱] به اشتراک گذاشته می‌شود.

---

۱. http://jakarta.apache.org/ant/index.html

[1] cluster

Name	Type	References	Description
datname	name		Database name
datdba	int4	pg_shadow.usesysid	Owner of the database, usually the user who created it
encoding	int4		Character/multibyte encoding for this database
datistemplate	bool		If true then this database can be used in the "TEMPLATE" clause of CREATE DATABASE to create a new database as a clone of this one.
datallowconn	bool		If false then no one can connect to this database. This is used to protect the template0 database from being altered.
datlastsysoid	oid		Last system OID in the database; useful particularly to pg_dump
datvacuumxid	xid		All tuples inserted or deleted by transaction IDs before this one have been marked as known committed or known aborted in this database. This is used to determine when commit-log space can be recycled.
datfrozenxid	xid		All tuples inserted by transaction IDs before this one have been relabeled with a permanent ("frozen") transaction ID in this database. This is useful to check whether a database must be vacuumed soon to avoid transaction ID wraparound problems.

Name	Type	References	Description
datpath	text		If the database is stored at an alternative location then this records the location. It's either an environment variable name or an absolute path, depending how it was entered.
datconfig	text[]		Session defaults for run-time configuration variables
datacl	aclitem[]		Access permissions

pg_depend ۱۲-۳-۴

جدول pg_depend وابستگی رابطه‌ای بین اشیای پایگاه داده را حفظ می‌کند. این اطلاعات در هنگام اجرای دستوری مانند DROP، کمک می‌کند تا سایر اطلاعات وابسته توسط DROP CASCADE حذف شوند و همچنین از حذف ناخواسته اطلاعات وابسته از طریق DROP RESTRICT جلوگیری می‌کند.

Name	Type	References	Description
classid	oid	pg_class.oid	The oid of the system catalog the dependent object is in
objid	oid	any oid attribute	The oid of the specific dependent object
objsubid	int4		For a table attribute, this is the attribute's column number (the objid and classid refer to the table itself). For all other object types, this field is presently zero.
refclassid	oid	pg_class.oid	The oid of the system catalog the referenced object is in
refobjid	oid	any oid attribute	The oid of the specific referenced object
refobjsubid	int4		For a table attribute, this is the attribute's column number (the refobjid and refclassid refer to the table itself). For all other object types, this field is presently zero.

Name	Type	References	Description
deptype	char		A code defining the specific semantics of this dependency relationship.

pg_description ۱۳-۳-۴

در این جدول می‌توان توضیحات و تفسیرهایی به صورت اختیـاری بـرای هـر شـیئ پایگـاه داده تعریف نمود.

Name	Type	References	Description
objoid	oid	any oid attribute	The oid of the object this description pertains to
classoid	oid	pg_class.oid	The oid of the system catalog this object appears in

Name	Type	References	Description
objsubid	int4		For a comment on a table attribute, this is the attribute's column number (the objoid and classoid refer to the table itself). For all other object types, this field is presently zero.
description	text		Arbitrary text that serves as the description of this object.

pg_group ۱۴-۳-۴

در این کاتالوگ گروهها تعریف می‌شوند و اطلاعاتی که نشان می‌دهند چه کاربری به چه گروهی تعلق دارد، در آن ذخیره می‌گردد. گروه را می‌توان با استفاده از دستور CREATE GROUP ایجاد کرد.

این کاتالوگ نیز بین تمام پایگاه داده‌های یک خوشه به اشتراک گذاشته می‌شود.

Name	Type	References	Description
groname	name		Name of the group
grosysid	int4		An arbitrary number to identify this group
grolist	int4[]	pg_shadow.usesysid	An array containing the ids of the users in this group

pg_index ۱۵-۳-۴

این کاتالوگ شامل برخی از اطلاعات درباره ایندکسها می‌باشد. باقی اطلاعات در pg_class می‌باشند.

Name	Type	References	Description
indexrelid	oid	pg_class.oid	The OID of the pg_class entry for this index
indrelid	oid	pg_class.oid	The OID of the pg_class entry for the table this index is for
indproc	regproc	pg_proc.oid	The function's OID if this is a functional index, else zero

Name	Type	References	Description
indkey	int2vector	pg_attribute.attnum	This is a vector (array) of up to INDEX_MAX_KEYS values that indicate which table columns this index pertains to. For example a value of 1 3 would mean that the first and the third column make up the index key. For a functional index, these columns are the inputs to the function, and the function's return value is the index key.
indclass	oidvector	pg_opclass.oid	For each column in the index key this contains a reference to the "operator class" to use. See pg_opclass for details.
indisclustered	bool		If true, the table was last clustered on this index.
indisunique	bool		If true, this is a unique index.
indisprimary	bool		If true, this index represents the primary key of the table. (indisunique should always be true when this is true.)
indreference	oid		unused
indpred	text		Expression tree (in the form of a nodeToString representation) for partial index predicate. Empty string if not a partial index.

pg_inherits ۱۶-۳-۴

این کاتالوگ اطلاعاتی درباره سلسله مراتب وراثتی جدول ثبت می‌نماید.

Name	Type	References	Description
inhrelid	oid	pg_class.oid	The OID of the child table.
inhparent	oid	pg_class.oid	The OID of the parent table.
inhseqno	int4		If there is more than one parent for a child table (multiple inheritance), this number tells the order in which the inherited columns are to be arranged. The count starts at 1.

pg_language ۱۷-۳-۴

pg_language واسطهای فراخوانی یا زبانها را برای نوشتن تابع یا رویه ذخیره شده ذخیره می‌کند.

Name	Type	References	Description
lanname	name		Name of the language (to be specified when creating a function)
lanispl	bool		This is false for internal languages (such as SQL) and true for user-defined languages. Currently, pg_dump still uses this to determine which languages need to be dumped, but this may be replaced by a different mechanism sometime.
lanpltrusted	bool		This is a trusted language. See under CREATE LANGUAGE what this means. If this is an internal language (lanispl is false) then this field is meaningless.

Name	Type	References	Description
lanplcallfoid	oid	pg_proc.oid	For non-internal languages this references the language handler, which is a special function that is responsible for executing all functions that are written in the particular language.
lanvalidator	oid	pg_proc.oid	This references a language validator function that is responsible for checking the syntax and validity of new functions when they are created. See under CREATE LANGUAGE for further information about validators.
lanacl	aclitem[]		Access permissions

pg_largeobject ۱۸-۳-۴

برای ذخیره اطلاعات در "اشیا بزرگ[1]" استفاده می‌شود. یک شیئ بزرگ توسط یک OIDای که در زمان ساخت به آن داده می‌شود، شناخته می‌گردد. هر شیئ بزرگ به تعدادی قطعه یا صفحه شکسته می‌شود تا به اندازه‌ای کوچک شود که ذخیره آن به صورت سطرهایی در pg_largeobject به راحتی صورت پذیرد.

۱. http://jakarta.apache.org/ant/index.html

[1] large objects

Name	Type	References	Description
loid	oid		Identifier of the large object that includes this page
pageno	int4		Page number of this page within its large object (counting from zero)
data	bytea		Actual data stored in the large object. This will never be more than LOBLKSIZE bytes, and may be less.

pg_listener ۱۹-۳-۴

پشتیبانی دستورات LISTEN و NOTIFY توسط pg_listener صورت می‌گیرد.

Name	Type	References	Description
relname	name		Notify condition name. (The name need not match any actual relation in the database; the term "relname" is historical.)
listenerpid	int4		PID of the backend process that created this entry.
notification	int4		Zero if no event is pending for this listener. If an event is pending, the PID of the backend that sent the notification.

pg_namespace ۲۰-۳-۴

namespace ساختاری اصولی در شمای SQL92 است. هر namespace می‌تواند کلکسیونی از ارتباطات، انواع و ... داشته باشد، بدون هیچ‌گونه ناسازگاری در نام آنها.

Name	Type	References	Description
nspname	name		Name of the namespace
nspowner	int4	pg_shadow.usesysid	Owner (creator) of the namespace
nspacl	aclitem[]		Access permissions

pg_opclass ۲۱-۳-۴

روشهای دستیابی ایندکس به کلاس عملگرها را تعریف می‌کند. هر کلاس عملگر معنایی برای ستونهای ایندکس یک نوع داده ویژه و نیز یک روش دستیابی به ایندکس خاص تعریف می‌کند.

Name	Type	References	Description
opcamid	oid	pg_am.oid	index access method opclass is for
opcname	name		name of this opclass
opcnamespace	oid	pg_namespace.oid	namespace of this opclass
opcowner	int4	pg_shadow.usesysid	opclass owner
opcintype	oid	pg_type.oid	type of input data for opclass
opcdefault	bool		true if opclass is default for opcintype
opckeytype	oid	pg_type.oid	type of index data, or zero if same as opcintype

pg_operator ۲۲-۳-۴

برای کسب اطلاعات کامل درباره این کاتالوگ دستور CREATE OPERATOR و راهنمای برنامه‌نویسان را مطالعه نمایید.

Name	Type	References	Description
oprname	name		Name of the operator
oprnamespace	oid	pg_namespace.oid	The OID of the namespace that contains this operator
oprowner	int4	pg_shadow.usesysid	Owner (creator) of the operator
oprkind	char		'b' = infix ("both"), 'l' = prefix ("left"), 'r' = postfix ("right")
oprcanhash	bool		This operator supports hash joins.
oprleft	oid	pg_type.oid	Type of the left operand
oprright	oid	pg_type.oid	Type of the right operand
oprresult	oid	pg_type.oid	Type of the result

Name	Type	References	Description
oprcom	oid	pg_operator.oid	Commutator of this operator, if any
oprnegate	oid	pg_operator.oid	Negator of this operator, if any
oprlsortop	oid	pg_operator.oid	If this operator supports merge joins, the operator that sorts the type of the left-hand operand (L<L)
oprrsortop	oid	pg_operator.oid	If this operator supports merge joins, the operator that sorts the type of the right-hand operand (R<R)
oprltcmpop	oid	pg_operator.oid	If this operator supports merge joins, the less-than operator that compares the left and right operand types (L<R)
oprgtcmpop	oid	pg_operator.oid	If this operator supports merge joins, the greater-than operator that compares the left and right operand types (L>R)
oprcode	regproc	pg_proc.oid	Function that implements this operator
oprrest	regproc	pg_proc.oid	Restriction selectivity estimation function for this operator
oprjoin	regproc	pg_proc.oid	Join selectivity estimation function for this operator

pg_proc ۲۳-۳-۴

اطلاعاتی را درباره توابع و یا رویه‌ها ذخیره می‌کند. برای اطلاع بیشتر توضیحات مربوط به دستور CREATE FUNCTION و راهنمای برنامه‌نویسان را مطالعه نمایید.

Name	Type	References	Description
proname	name		Name of the function
pronamespace	oid	pg_namespace.oid	The OID of the namespace that contains this function
proowner	int4	pg_shadow.usesysid	Owner (creator) of the function
prolang	oid	pg_language.oid	Implementation language or call interface of this function
proisagg	bool		Function is an aggregate function
prosecdef	bool		Function is a security definer (i.e., a "setuid" function)
proisstrict	bool		Function returns null if any call argument is null. In that case the function won't actually be called at all. Functions that are not "strict" must be prepared to handle null inputs.
proretset	bool		Function returns a set (ie, multiple values of the specified data type)

Name	Type	References	Description
provolatile	char		provolatile tells whether the function's result depends only on its input arguments, or is affected by outside factors. It is i for "immutable" functions, which always deliver the same result for the same inputs. It is s for "stable" functions, whose results (for fixed inputs) do not change within a scan. It is v for "volatile" functions, whose results may change at any time. (Use v also for functions with side-effects, so that calls to them cannot get optimized away.)
pronargs	int2		Number of arguments
prorettype	oid	pg_type.oid	Data type of the return value
proargtypes	oidvector	pg_type.oid	A vector with the data types of the function arguments
prosrc	text		This tells the function handler how to invoke the function. It might be the actual source code of the function for interpreted languages, a link symbol, a file name, or just about anything else, depending on the implementation language/call convention.
probin	bytea		Additional information about how to invoke the function. Again, the interpretation is language-specific.
proacl	aclitem[]		Access permissions

pg_rewrite ۲۴-۳-۴

این کاتالوگ سیستم قوانین بازنویسی جدولها و دیدها را ذخیره می‌کند.

Name	Type	References	Description
rulename	name		Rule name
ev_class	oid	pg_class.oid	The table this rule is for
ev_attr	int2		The column this rule is for (currently, always zero to indicate the whole table)
ev_type	char		Event type that the rule is for: '1' = SELECT, '2' = UPDATE, '3' = INSERT, '4' = DELETE
is_instead	bool		True if the rule is an INSTEAD rule
ev_qual	text		Expression tree (in the form of a nodeToString representation) for the rule's qualifying condition
ev_action	text		Query tree (in the form of a nodeToString representation) for the rule's action

pg_shadow ۲۵-۳-۴

این کاتالوگ شامل اطلاعاتی درباره کاربران پایگاه داده است. از آنجا که این جدول شامل کلمه عبور افراد است، نباید برای عموم قابل خواندن باشد و علت این نامگذاری نیز همین مسئله است.

Name	Type	References	Description
usename	name		User name
usesysid	int4		User id (arbitrary number used to reference this user)
usecreatedb	bool		User may create databases
usesuper	bool		User is a superuser
usecatupd	bool		User may update system catalogs. (Even a superuser may not do this unless this attribute is true.)
passwd	text		Password
valuntil	abstime		Account expiry time (only used for password authentication)
useconfig	text[]		Session defaults for run-time configuration variables

pg_statistic ۲۶-۳-۴

این کاتالوگ اطلاعات آماری درباره محتویات پایگاه داده را ذخیره می‌نماید. برای هر ستون جدول که آنالیز و تحلیل شده باشد، یک مدخل[1] در آن وجـود دارد. pg_statistic نبایـد بـرای عمـوم قابـل خواندن باشد، زیرا اطلاعات آماری درباره محتویات جدولها ممکن است حساس و مهم باشند. مثلا در رابطه با بیشترین و کمترین میزان ستون حقوق این مسئله اهمیت پیدا می‌کند.

Name	Type	References	Description
starelid	oid	pg_class.oid	The table that the described column belongs to

۱. http://jakarta.apache.org/ant/index.html

[1] entry

Name	Type	References	Description
staattnum	int2	pg_attribute.attnum	The number of the described column
stanullfrac	float4		The fraction of the column's entries that are NULL
stawidth	int4		The average stored width, in bytes, of non-NULL entries
stadistinct	float4		The number of distinct non-NULL data values in the column. A value greater than zero is the actual number of distinct values. A value less than zero is the negative of a fraction of the number of rows in the table (for example, a column in which values appear about twice on the average could be represented by stadistinct = -0.5). A zero value means the number of distinct values is unknown.
stakindN	int2		A code number indicating the kind of statistics stored in the Nth "slot" of the pg_statistic row.
staopN	oid	pg_operator.oid	An operator used to derive the statistics stored in the Nth "slot". For example, a histogram slot would show the < operator that defines the sort order of the data.
stanumbersN	float4[]		Numerical statistics of the appropriate kind for the Nth "slot", or NULL if the slot kind does not involve numerical values.

Name	Type	References	Description
stavaluesN	text[]		Column data values of the appropriate kind for the Nth "slot", or NULL if the slot kind does not store any data values. For data-type independence, all column data values are converted to external textual form and stored as TEXT datums.

pg_trigger ۲۷-۳-۴

در این کاتالوگ تریگرهای جداول ذخیره می‌شوند. برای اطلاع بیشتر دستور CREATE TRIGGER مطالعه گردد.

Name	Type	References	Description
tgrelid	oid	pg_class.oid	The table this trigger is on
tgname	name		Trigger name (must be unique among triggers of same table)
tgfoid	oid	pg_proc.oid	The function to be called
tgtype	int2		Bitmask identifying trigger conditions
tgenabled	bool		True if trigger is enabled (not presently checked everywhere it should be, so disabling a trigger by setting this false does not work reliably)
tgisconstraint	bool		True if trigger implements an RI constraint
tgconstrname	name		RI constraint name
tgconstrrelid	oid	pg_class.oid	The table referenced by an RI constraint
tgdeferrable	bool		True if deferrable
tginitdeferred	bool		True if initially deferred

Name	Type	References	Description
tgnargs	int2		Number of argument strings passed to trigger function
tgattr	int2vector		Currently unused
tgargs	bytea		Argument strings to pass to trigger, each null-terminated

pg_type ۲۸-۳-۴

این کاتالوگ اطلاعاتی درباره انواع داده نگهداری می‌کند.

Name	Type	References	Description
typname	name		Data type name
typnamespace	oid	pg_namespace.oid	The OID of the namespace that contains this type
typowner	int4	pg_shadow.usesysid	Owner (creator) of the type
typlen	int2		For a fixed-size type, typlen is the number of bytes in the internal representation of the type. But for a variable-length type, typlen is negative. -1 indicates a "varlena" type (one that has a length word), -2 indicates a null-terminated C string.

Name	Type	References	Description
typrelid	oid	pg_class.oid	If this is a complex type (see typtype), then this field points to the pg_class entry that defines the corresponding table. (For a free-standing composite type, the pg_class entry doesn't really represent a table, but it is needed anyway for the type's pg_attribute entries to link to.) Zero for non-complex types.
typelem	oid	pg_type.oid	If typelem is not 0 then it identifies another row in pg_type. The current type can then be subscripted like an array yielding values of type typelem. A "true" array type is variable length (typlen = -1), but some fixed-length (typlen > 0) types also have nonzero typelem, for example name and oidvector. If a fixed-length type has a typelem then its internal representation must be N values of the typelem data type with no other data. Variable-length array types have a header defined by the array subroutines.
typinput	regproc	pg_proc.oid	Input conversion function
typoutput	regproc	pg_proc.oid	Output conversion function

Name	Type	References	Description
typalign	`char`		`typalign` is the alignment required when storing a value of this type. It applies to storage on disk as well as most representations of the value inside PostgreSQL. When multiple values are stored consecutively, such as in the representation of a complete row on disk, padding is inserted before a datum of this type so that it begins on the specified boundary. The alignment reference is the beginning of the first datum in the sequence. Possible values are: • 'c' = CHAR alignment, i.e., no alignment needed. • 's' = SHORT alignment (2 bytes on most machines). • 'i' = INT alignment (4 bytes on most machines). • 'd' = DOUBLE alignment (8 bytes on many machines, but by no means all). **Note:** For types used in system tables, it is critical that the size and alignment defined in `pg_type` agree with the way that the compiler will lay out the field in a struct representing a table row.

Name	Type	References	Description
typstorage	char		typstorage tells for varlena types (those with typlen = -1) if the type is prepared for toasting and what the default strategy for attributes of this type should be. Possible values are • 'p': Value must always be stored plain. • 'e': Value can be stored in a "secondary" relation (if relation has one, see pg_class.reltoast • 'm': Value can be stored compressed inline. • 'x': Value can be stored compressed inline or in "secondary". Note that 'm' fields can also be moved out to secondary storage, but only as a last resort ('e' and 'x' fields are moved first).
typnotnull	bool		typnotnull represents a NOT NULL constraint on a type. Presently used for domains only.
typbasetype	oid	pg_type.oid	If this is a derived type (see typtype), then typbasetype identifies the type that this one is based on. Zero if not a derived type.

Name	Type	References	Description
typtypmod	int4		Domains use typtypmod to record the typmod to be applied to their base type (-1 if base type does not use a typmod). -1 if this type is not a domain.
typndims	int4		typndims is the number of array dimensions for a domain that is an array (that is, typbasetype is an array type; the domain's typelem will match the base type's typelem). Zero for non-domains and non-array domains.
typdefaultbin	text		If typdefaultbin is not NULL, it is the nodeToString representation of a default expression for the type. Currently this is only used for domains.
typdefault	text		typdefault is NULL if the type has no associated default value. If typdefaultbin is not NULL, typdefault must contain a human-readable version of the default expression represented by typdefaultbin. If typdefaultbin is NULL and typdefault is not, then typdefault is the external representation of the type's default value, which may be fed to the type's input converter to produce a constant.

۴-۴ پروتکلهای frontend/backend

PostgreSQL از پروتکل مبتنی بر پیغام برای برقراری ارتباط بین frontend ها (مشتری) و backend ها (خدمت‌گزار) استفاده می‌کند. این پروتکل بر روی TCP/IP و نیز سوکتهای domain یونیکس[۱] اجرا می‌شود. در اینجا به توضیح نسخه ۲،۰ پروتکل، که در PostgreSQL 6.4 و نسخه‌های بعدی پیاده‌سازی شده است را توضیح خواهیم داد.

یک frontend اتصالی را با خدمتگذار برقرار می‌کند و بسته[۲] شروع و ابتدایی را ارسال می‌کند. این بسته شامل نام کاربری از پایگاه داده است که تقاضای برقراری ارتباط را دارد. خدمتگذار از این بسته و اطلاعات فایل pg_hba.conf، جهت تعیین اعتبار استفاده می‌کند تا مشخص شود چه اطلاعاتی باید برای frontend ارسال گردد.

سپس frontend اطلاعات مورد نیاز برای تعیین اعتبار ارسال می‌نماید. خدمتگذار نیز پس از بررسی پاسخها در صورت تایید آنها پیغام موفقیت‌آمیز بودن شروع را برای آن ارسال می‌کند و در صورت نامعتبر بودن، عدم موفقیت را به او اعلام می‌نماید.

به منظور پاسخگویی موثر به چند مشتری، خدمتگذار به ازای هر مشتری یک پروسس «backend» تولید می‌کند. در پیاده‌سازی موجود، یک پروسس فرزند جدید بلافاصله پس از برقراری ارتباط ایجاد می‌گردد.

با تمام شدن کار frontend، ارتباط باید قطع شود، frontend برای این امر بسته مناسب را ارسال کرده و ارتباط بدون انتظار برای دریافت جواب از خدمتگذار پایان می‌یابد.

در این پروتکل بر اساس وضعیت و حالت ارتباط، چهار نوع جریان پیام داریم: شروع، پرس و جو، فراخوانی تابع و خاتمه. در چرخه پرس و جو، frontend یک پیغام پرس و جو برای backend ارسال می‌کند. backend با توجه به رشته دستورات موجود در پرس و جو، یک یا چند پیام پاسخ ارسال

---

۱. http://jakarta.apache.org/ant/index.html

[1] Unix domain sockets
[2] packet

می‌کند و بالاخره یک پیام پاسخ ReadyForQuery می‌فرستد. ReadyForQuery به frontend اطلاع می‌دهد که می‌تواند یک پرس و جوی جدید فرستاده یا یک تابع را فراخوانی کند. چرخه فراخوانی تابع نیز توسط frontend و با ارسال یک پیغام FunctionCall برای backend آغاز می‌گردد. backend نیز با توجه به درخواست پاسخ مناسب را با یک یا چند پیغام برای frontend ارسال می‌دارد. همچنین مشابه قبل با فرستادن یک پیام ReadyForQuery به frontend اطلاع می‌دهد که می‌تواند یک پرس و جوی جدید فرستاده یا یک تابع را فراخوانی کند.

علاوه بر اینها، مقررات ویژه‌ای برای اخطارها و تذکرات و نیز لغو دستورات وجود دارد که بعد از شروع ارتباط در هر زمان امکان رخ دادن آنها وجود داد.

نسخه‌های اخیر PostgreSQL به ارتباط frontend/backend امکان رمز کردن با استفاده از SSL را می‌دهند. برای این منظور frontend یک پیغام SSLRequest را پیش از بسته شروع ارسال می‌نماید. خدمتگذار در پاسخ یک بایت ساده شامل Y یا N برای آن ارسال می‌کند که تمایل و یا عدم تمایل ایجاد SSL را در آن بیان می‌دارد. ممکن است در صورت رد تقاضا از سوی خدمتگذار، frontend ارتباط را قطع نماید. در صورت تمایل خدمتگذار توافقهای اولیه برای شروع SSL با آن انجام می‌گیرد. پس از آن بسته شروع ارسال می‌گردد. از اینجا به بعد تمامی داده‌ها به صورت رمز شده ردوبدل می‌شوند.

۴-۵ بهینه سازی پرس و جوهای عمومی

۴-۵-۱ بررسی پرس و جو، مشکل پیچیدگی بهینه‌سازی

از بین همه عملگرهای رابطه‌ای، پیچیده‌ترین آنها در پردازش و بهینه‌سازی، پیوند[۱] می‌باشد. تعداد راههای جانشین برای جواب پرس و جوهایی که شامل پیوند هستند، به صورت نمایی رشد

---

۱. http://jakarta.apache.org/ant/index.html

[1] join

می‌کند. در نتیجه فعالیت بیشتری باید توسط بهینه‌ساز در هر یک از روشهای پیوند صورت گیرد (حلقه‌های تودرتو، پیوند hash، پیوند ادغامی در PostgreSQL).

پیاده‌سازی بهینه‌ساز PostgreSQL موجود یک جستجوی شبه-کامل[1] بر روی فضای راهبردهای جانشین انجام می‌دهد. تکنیک بهینه‌سازی پرس و جو برای پشتیبانی از برنامه‌های کاربردی پایگاههای داده حوزه‌های گسترده پرس و جو، مانند هوش مصنوعی، ناقص می‌باشد.

در ادامه ما پیاده‌سازی الگوریتم وراثت را برای مشکل بهینه‌سازی پرس و جوی پایگاه داده ارائه می‌دهیم.

۴-۵-۲ الگوریتمهای وراثت[2]

الگوریتم وراثت، یک روش بهینه‌سازی اکتشافی است که از طریق جستجوهای تصادفی، قطعی[3] عمل می‌نماید. مجموعه راه‌حلهای ممکن برای مساله بهینه‌سازی به عنوان population of individuals بررسی می‌گردند. درجه تطبیق یک individual با محیطش به وسیله fitness آن تعیین می‌گردد.

هم‌پایه‌های[4] یک individual در فضای جستجو توسط کروموزومها و در اصل با مجموعه‌ای از رشته‌های کاراکتری نمایش داده می‌شوند. "ژن" بخشی از یک کروموزوم است که مقدار یک پارامتر ساده بهینه شده را کدگذاری می‌کند. کدگذاری ژنها می‌تواند به صورت دودویی یا integer باشد.

با توجه به پرسش و پاسخهای مطرح شده در comp.ai.genetic، نمی‌توان قاطعانه اظهار داشت که GA نمی‌تواند روش جستجوی تصادفی کامل و مطلق برای راه‌حل یک مسئله باشد. در واقع GA از طریق فرآیندهای احتمالی عمل می‌کند، اما نتیجه به وضوح نتیجه‌ای غیر تصادفی و نسبتا خوب است.

---

۱. http://jakarta.apache.org/ant/index.html

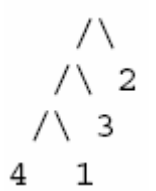
[1] near-exhaustive search
[2] genetic algorithm (GA)
[3] determined
[4] coordinates

P(t)	generation of ancestors at a time t
P”(t)	generation of descendants at a time t

```
+=========================================+
|>>>>>>>>>>>  Algorithm GA  <<<<<<<<<<<<<|
+=========================================+
| INITIALIZE t := 0                       |
+=========================================+
| INITIALIZE P(t)                         |
+=========================================+
| evaluate FITNESS of P(t)                |
+=========================================+
| while not STOPPING CRITERION do         |
|   +-------------------------------------+
|   | P'(t)  := RECOMBINATION{P(t)}       |
|   +-------------------------------------+
|   | P"(t) := MUTATION{P'(t)}            |
|   +-------------------------------------+
|   | P(t+1) := SELECTION{P"(t) + P(t)}   |
|   +-------------------------------------+
|   | evaluate FITNESS of P"(t)           |
|   +-------------------------------------+
|   | t := t + 1                          |
+===+=====================================+
```

۳-۵-۴ بهینه سازی پرس و جوی عمومی (GEQO)[1] در PostgreSQL

ماژول GEQO برای حل مساله بهینه‌سازی پرس و جو مشابه مساله فروشنده دوره‌گرد (TSP) عمل می‌نماید. راههای ممکن پرس و جو به رشته‌های integer رمز شده‌اند. هر رشته نشان‌دهنده یک پیوند از یک ارتباط پرس و جو به بعدی می‌باشد.



۱. http://jakarta.apache.org/ant/index.html

[1] Genetic Query Optimization

درخت پرس و جو با رشته "۲-۳-۱-۴" رمز شده و بدین معنی است کـه اولـین پیونـد بـا ۴ و ۱، سپس ۳ و بعد ۲ ارتباط دارد که ۱،۲،۳،۴ IDهای ارتباط در بهینه‌ساز PostgreSQL هستند.

ماژولهای GEQO به بهینه‌سـاز پـرس و جـوی PostgreSQL امکـان پشـتیبانی کـارا از پـرس و جوهای پیوندی بزرگ را از طریق جستجوی ناکامل می‌دهد.

برای کسب اطلاعات بیشتر درباره الگوریتمهای وراثت می‌توانید به منابع زیر رجوع نمایید:

- راهنمای Hitch-Hiker۱ (FAQ برای 2comp.ai.genetic)
- محاسبات تکاملی و برنامه‌های کاربردی آن جهت هنر و طراحی۳، نوشته Craig Reynolds
- مهارتهای اصولی سیستمهای پایگاه داده4
- طراحی و پیاده‌سازی بهینه‌ساز پرس و جوی POSTGRES5

۴-۶ پشتیبانی از زبانهای ملی

۴-۶-۱ برای مترجم

برنامه‌های PostgreSQL (مشتری و خدمتگذار) می‌توانند پیغامهای خود را به زبان مـورد علاقـه شما صادر کنند، البته چنانچه پیغام ترجمه شود. ایجاد و نگهداری مجموعه پیامها به کمک افرادی که با زبان خود به خوبی آشنا هستند و می‌خواهند در گسترش PostgreSQL سـهیم باشـند، نیـاز دارد. برای این منظور نیازی نیست که شما یک برنامه‌نویس باشـید. در ادامـه روش انجـام آن توضـیح داده می‌شود.

۱. نیازمندیها

---

۱. http://jakarta.apache.org/ant/index.html

[1] http://surf.de.uu.net/encore/www/
[2] news://comp.ai.genetic
[3] http://www.red3d.com/cwr/evolve.html
[4] Ramez Elmasri and Shamkant Navathe, Fundamentals of Database Systems, 3rd Edition, Addison-Wesley, ISBN 0-805-31755-4, August 1999.
[5] Zelaine Fong, The design and implementation of the POSTGRES query optimizer2, University of California, Berkeley, Computer Science Department. (http://s2k-ftp.CS.Berkeley.EDU:8000/postgres/papers/UCB-MS-zfong.pdf)

در این قسمت درباره مهارت شما در زبان قضاوتی نمی‌شود؛ این بخش مربوط به ابزارهای نرم‌افزار است. از نظر عملی شما تنها به یک ویرایشگر متن نیاز دارید. هنگامی که قصد پیکربندی درخت منبع خود را دارید، مطمئن شوید که گزینه enable-nls-- را به کار می‌برید. این امر همچنین برای کتابخانه libintl و برنامه msgfmt باید بررسی گردد. برای آزمایش کامل کار خود، بخشهای قابل اجرای دستورالعمل نصب را دنبال کنید.

چنانچه قصد دارید یک ترجمه جدید را آغاز نمایید و یا قصد ادغام کاتالوگ پیغام را دارید، به برنامه‌های xgettext و msgmerge، به ترتیب، در پیاده‌سازی سازگار GNU نیاز دارید. بعدا برای مرتب‌سازی آن اقدام می‌کنیم، چنانچه شما از توزیع منبع بسته‌بندی استفاده کنید، نیازی به xgettext ندارید (برای CVS شما همچنان به آن نیاز دارید). gettext 0.10.36 گنو به بعد همچنان توصیه می‌شوند.

شما برای کسب جزئیات بیشتر نیاز دارید که پیاده‌سازی gettext محلی‌تان همراه با سندش آورده شود.

۲. مفاهیم

پیام اصلی (انگلیسی) و معادل ترجمه شده آن در کاتالوگ پیغام[1] نگهداری می‌شوند، یکی برای هر برنامه (همچنین برنامه‌های وابسته می‌توانند کاتالوگ پیغام را به اشتراک گذارند) و برای هر زبان مقصد. دو نوع قالب فایل برای کاتالوگ پیغام وجود دارد: اولی فایل «2PO» (برای اشیا قابل جابه‌جایی)، که یک فایل plain text با گرامر و نحو خاص است که مترجمها ویرایش می‌کنند. دومی فایل «3MO» (برای اشیا ماشین)، که یک فایل دودویی از فایل PO مربوطه است و در هنگام بین‌المللی کردن برنامه استفاده می‌شود.

---

۱. http://jakarta.apache.org/ant/index.html

[1] Message catalogs
[2] Portable Object
[3] Machine Object

پسوند فایل کتالوگ پیام po. و یا mo. می‌باشد. نام اصلی یا نام برنامه همراه آن است و یـا زبـانی که فایل برای آن ساخته شده است. مثلا psql.po (فایل PO برای psql) و یا fr.mo (فایل MO بـرای فرانسه).

قالب فایل PO در اینجا نشان داده شده است:

```
# comment
msgid "original string"
msgstr "translated string"
msgid "more original"
msgstr "another translated"
"string can be broken up like this"
...
```

msgid ها از منبع برنامه استخراج شده‌اند. (البته نیازی نیست اما این روش عمومی است.) مقـدار اولیه خطوط msgstr خالی است و توسط مترجم با رشته‌های کاربردی مفید پر می‌شوند.

کاراکتر # نشان‌دهنده توضـیحات اسـت. بلافاصـله بعـد از کـاراکتر # یـک whitespace ظـاهر می‌شود که نشان می‌دهد این توضیح توسط مترجم اضافه شده است. همچنین برخـی توضـیحات بـه صورت خودکار اضافه می‌شوند که در این صورت پس از کاراکتر # هیچ whitespace قرار نمی‌گیرد.

```
#. automatic comment
#: filename.c:1023
#, flags, flags
```

۴-۶-۲ ایجاد و نگهداری کاتالوگهای پیغام

خوب، چگونه می‌توان یک کاتالوگ پیغام «خالی» ساخت؟ ابتدا به دایرکتوری مورد نظر خود کـه شامل برنامه‌ای است که قصد دارید پیغامهایش را ترجمه کنیـد، برویـد. چنانچـه فایـل nls.mk وجـود داشته باشد، این برنامه برای ترجمه آماده است.

اگر تعدادی فایل po. وجود دارد دلیل آنست که کسی قبلا کمی از کارهای ترجمه را انجـام داده است. فایلها با اسم language.po نامگـذاری مـی‌شـوند کـه language کـد دو حرفـی 1ISO 639-1 زبانهاست، مثلا برای فرانسه fr.po است. چنانچه به دلیلی نیاز است که مجددا برای یک زبان ترجمـه انجام شود، در این صورت این فایلها با اسم language_region.po ایجاد می‌شود کـه region کـد دو حرفی 2ISO 3166-1 کشور مورد نظر است، مثلا pt_BR.po برای زبان پرتقالی در برزیل.

اما اگر شما اولین تلاش را برای ترجمه آغاز می‌کنید، ابتدا باید دستور زیر را اجرا نمایید:

```
gmake init-po
```

این دستور یک فایل progname.pot می‌سازد. (pot. برای تمییز دادن از فایلهای PO استفاده شده که در آن T به «الگو[3]» بودن آن اشاره دارد.) این فایل را در language.po کپی و ویـرایش کنیـد. بـرای اظهار این مطلب که یک زبان جدید در دسترس است، فایل nls.mk را ویرایش کرده و کد زبان (یا کد زبان و کد کشور) را مشابه زیر اضافه کنید:

```
AVAIL_LANGUAGES := de fr
```

درصورتیکه تغییری در زیربرنامه یا کتابخانـه‌ای بـه وجـود آیـد، ممکـن اسـت پیامهـایی توسـط برنامه‌نویس اضافه شده یا تغییر کنند. در این صورت نیازی نیست که شما از صفر شروع کنید. کـافی است دستور زیر را اجرا کنید:

```
gmake update-po
```

این دستور یک فایل کاتالوگ پیغام خالی جدید ایجاد کرده (فایل pot کـه بـا آن شـروع بـه کـار کردید) و آن را با فایلهای PO موجود ادغام می‌نماید.

---

۱. http://jakarta.apache.org/ant/index.html

[1] http://lcweb.loc.gov/standards/iso639-2/englangn.html
[2] http://www.din.de/gremien/nas/nabd/iso3166ma/codlstp1/en_listp1.html
[3] template

۴-۶-۳ برای برنامه‌نویسان

این بخش به توضیح چگونگی پشتیبانی از زبان اختصاصی۱ در برنامه یا کتابخانـه کـه بخشـی از توزیع PostgreSQL است می‌پردازد. در حال حاضر این امر تنها برای برنامه‌های C قابل اجراست.

- **اضافه کردن پشتیبانی NLS به یک برنامه**

۱. در ابتدای برنامه قطعه کد زیر را اضافه نمایید: (progname را می‌توان آزادانه انتخاب کرد)

```
#ifdef ENABLE_NLS
#include <locale.h>
#endif
...
#ifdef ENABLE_NLS
setlocale(LC_ALL, "");
bindtextdomain("progname", LOCALEDIR);
textdomain("progname");
#endif
```

۲- در هر جاییکه پیغامی برای ترجمه پیدا شد، ()gettext باید فراخوانی شود. مثلا:

```
fprintf(stderr, "panic level %d\n", lvl);
```

به شکل زیر تغییر می‌یابد:

```
fprintf(stderr, gettext("panic level %d\n"), lvl);
```

۳- یک فایل nls.mk در دایرکتوری منبع برنامه اضافه نمایید. این فایل به عنوان یک makefile خوانده خواهد شد. مقداردهیهای زیر در این قسمت باید صورت گیرد:

CATALOG_NAME

نام برنامه، که با فراخوانی ()textdomain فراهم می‌شود.

AVAIL_LANGUAGES

---

۱. http://jakarta.apache.org/ant/index.html

[1] native language