بنام خدا

خودآموز PostgreSQL ۷,۳,۲

گروه توسعه جهاني PostgreSQL

فهرست مطالب

سرآغاز

۱. PostgreSQL چيست؟

PostgreSQL يک سيستم مديريتي پايگاه داده(Object-Relational (ORDBMS ميباشد که بر اساس POSTGRES version ٤,٢[1] در بخش علوم کامپيوتري برکلي دانشگاه کاليفرنيا توسعه داده شده است.
پروژه POSTGRES توسط پرفسور Michael Stonebraker هدايت و گروه هايي چون آژانس پروژه هاي تحقيقات پيشرفته دفاع (DARPA)، اداره تحقيقات ارتش(ARO)، بنياد علوم ملي (NFS) ، ESL و اعضاي وابسطه ديگري، حاميان آن بودند .
PostgreSQL نسخه Open-Source ازاين کد اصلي برکلي مي باشد و از زبان SQL۹۲/SQL۹۹ و ديگر ابزارهاي امروزي پشتيباني مي کند.
اکنون POSTGRES بعنوان پيشگام بسياري از مفاهيم Object-Relational، در بعضي از پايگاه داده هاي تجاري عرضه ميگردد. در سيستم مديريت پايگاه دادهRelational (RDBMS) قديمي، از مجموعه نام هاي وابسته، که همگي شامل صفاتي همگون بودند پشتيباني مي شد و در سيستمهاي تجاري فعلي، انواعي شامل Floating Point Number ، Integer، Character String، Money وDate قابل پشتيباني مي باشند. اين مسئله نيز بديهي است که اين مدل براي برنامه هاي Data Processing آينده کافي نيست.
PostgreSQL چند قابليت مهم اضافي را بطريقي که کاربر توانايي توسعه سيستم را دارا باشد در کنارمفاهيم زير عرضه مي دارد:

Inheritance
Data Type
Function

ونيز ابزارهاي ديگري که شامل قابليت ها و انعطاف بيشتري مي باشند:

Constraints
Triggers
Rules
Transactional Integrity

اين قابليت ها PostgreSQL را در زمره پايگاه داده Object-Relational قرار داده است وقابل توجه است که مفاهيم فوق وجه تمايزي با پايگاه هاي داده يي که با عنوان Object-Oriented عرضه شده اند -وبا پايگاه هاي داده وابسته قديمي سازگاري کامل ندارند- محسوب مي شوند. بنا بر اين هر چند که PostgreSQL بعضي از قابليتهاي مدل Object-Oriented را دارد اما در رده پايگاه هاي داده Relational شناخته مي شود.

۲. تاريخچه اي بر PostgreSQL

امروزه سيستم مديريت پايگاه هاي داده Object-Relational که به عنوان PostgreSQL شناخته شده است (و بطور خلاصه Postgres۹۵ خطاب مي شود) ازبسته نرم افزاري POSTGRES که در دانشگاه کاليفرنيا در برکلي توليد شد بر گرفته شده است. PostgreSQL با بيش از يک دهه توسعه، پيشرفته ترين پايگاه داده اپن سورس در سراسر دنياست که ارائه دهنده کنترل همزمان نسخه هاي متنوع، پشتيباني از همه ساختارهايSQL (شامل گزينشهاي تودرتو(Subselects)، Transactions ، توابع و انواع داده اي که کاربر تعريف ميکند) و تعداد بسيار زيادي از زبانهاي قابل اتصال (مانند C، C++، Java، Perl، Tcl و Python) مي باشد.

۲,۱. پروژه PostgreSQL در دانشگاه برکلي

اجراي پروژه POSTGRES DBMS در سال ۱۹۸٦ آغاز گرديد و پس از آن POSTGRES چندين انتشار را پشت سر گذاشت و اولين "نمونه افزار" (demo ware) سيستم در سال ۱۹۸۷ قابل استفاده و در کنفرانس ACM-SIGMOD سال ۱۹۸۸ عرضه گرديد. نسخه ۱ در ماه ژوئن سال ۱۹۸۹ در اختيار تعدادي چند از کاربران آزاد قرار داده شد. سيستم قوانين POSTGRES در واکنش به يک انتقاد از اولين سيستم قوانين (A commentary on the POSTGRES rules system) مجدداً طراحي شد(يعني اصلاحاتي روي Rule ها،procedureها، CachingوViewها در سيستم پايگاه داده) و

[1] http://s۲k-ftp.CS.Berkeley.EDU:۸۰۰۰/postgres/postgres.html

نسخه ۲ آن به همراه سيستم قوانين جديد در ژوئن ۱۹۹۰ عرضه گرديد. در سال ۱۹۹۱ نسخه ۳ با اضافه نمودن پشتيباني از سيستم مديريت ذخايرمتنوع، يک اجرا کننده درخواست بهينه شده و يک سيستم قوانين قابل ويرايش از نو طراحي و وارد بازار شد. همچنين ويرايشهاي بعدي تا نسخهPOSTGRES۹۵، در بسياري از قسمتها، بر روي قابليتهاي جابجايي و پايداري متمرکز شده بودند.
POSTGRES، درگير اجراي تحقيقات و توليد برنامه هايي از قبيل: سيستم تحليل اطلاعات مالي، کنترل بهره وري موتور جت، پايگاه داده براي دنبال کردن ستاره ها، پايگاه داده اطلاعات پزشکي و چندين سيستم اطلاعات جغرافيايي شد. همچنين در بعضي از دانشگاهها POSTGRES تبديل به يک ابزار آموزشي گرديد. در نهايت فناوري اطلاعات Illustrate (که بعده ها به Informix[۲] پيوست و اکنون وابسته به IBM[۳] است.) با در اختيار گرفتن کدهاي مرجع، آن را تجاري ساخت. بدين سان POSTGRES در اواخر ۱۹۹۲، اولين مدير پايگاه داده در پروژه محاسبات علمي ۲۰۰۰[٤] Sequoia گرديد.
در سال ۱۹۹۳ تعداد کاربران آزاد دوبرابرشد و واضح بود که نگهداري از کدهاي اوليه و پشتيباني، زمان بسيار زيادي را که بايد به تحقيقات روي پايگاه داده صرف مي شد به خود اختصاص داده است. بنابراين در تلاشي جهت کاهش باراين پشتيباني ها پروژه Berkeley POSTGRES بصورت رسمي با نسخه ٤,۲ به کار خود پايان بخشيد.

Postgres۹۵ .۲,۲

در سال ۱۹۹٤ Andrew Yu و Jolly Chen يک مترجم زبان SQL به POSTGRES افزودند و متعاقباً Postgres۹۵ رااز طريق وب انتشار دادند.
Postgres۹۵ به طور کامل در ANSI C شکل گرفت و اندازه آن نيز ۲۵ درصد کاهش يافت. همچنين بسياري از تغييرات دروني، به بهره وري و قابليت بازيابي سيستم افزودند. بنا بر استاندارد وسکانسن، نسخه ۱,۰.x، POSTGRES۹۵، ۳۰ تا ۵۰ درصد سريعتر از نسخه ٤,۲، POSTGRES انتشار يافت، همچنين در کنار رفع برخي ازايرادهاي موجود، موارد زير از جمله پيشرفت هاي مهم در POSTGRES۹۵ برشمرده مي شوند:

-زبان درخواست(query) PostQUEL، با SQL تعويض گرديد (در Server تعبيه گرديد.). البته در نسخه هاي قبلي ازقابليت (Subquery) پشتيباني نمي شد و اينن قابليت توسط توابعي که کاربر در SQL تعريف مي کرد شبيه سازي مي شد. Aggregate function ها دوباره پياده سازي شدند و امکاناتي چون عبارات GROUP BY جهت استفاده در Queryها افزوده شد. همچنين اينترفيس Libpq جهت برنامه نويسي در C باقي ماند.

- به منظور مانيتورينگ برنامه ها، يک برنامه جديد به نام Psql براي ارتباط محاوره اي با query هاي SQL فراهم گرديد.

- يک کتابخانه Front-end جديد به نام Libpgtcl، که از سرويس گيرنده هاي مبتني بر TCL پشتيباني مي کند اضافه شد. براي نمونه مي توان به پوسته pgtclsh اشاره نمود که دستورات Tcl جديدي براي ارتباط برنامه Tcl با Back-end Postgres۹۵ فراهم کرده است.

- رابط Large-object بازنگري گرديد. وارونه سازي، تنها راه براي ذخيره مقادير بزرگ گرديد(وارونه سازي فايل سيستم حذف گرديد.).

- instance-level rule system، حذف شد.
- يک خودآموز مختصر، به توصيف ابزارهاي رايج SQL مي پردازد، البته به همان خوبي که POSTGRES۹۵ به همراه کد هاي مرجع منتشر مي کرد.

---

[۲] http://www.informix.com/
[۳] http://www.ibm.com/
[٤] http://meteora.ucsd.edu/s۲k/s۲k_home.html

- از GNU make به جاي BSD make استفاده شد. همچنين POSTGRES۹۵ مي توانست توسط كامپايلر GCC اصلاح نشده كامپايل شود.

۲,۳. PostgreSQL

در سال ۱۹۹٦ بود كه مشخص شد نام Postgres۹۵ براي ادامه كار مناسب نيست بنابراين ما نام PostgreSQL را براي انعكاس رابطه POSTGRES اوليه و نسخه هاي اخير كه قابليتهاي SQL را دارا بودند برگزيديم تا با تمام نسخه ها سازگار باشد و در همين زمان شماره گذاري نسخه ها را نيز از ٦,۰، آغاز نموديم.
تأكيد اصلي براي توسعه Postgres۹۵ بر روي تشخيص و درک مشكلات موجود در كدهاي Back-end بود. با PostgreSQL در حالي كه همچنان به همه جوانب توجه مي شد، تأكيد و اهميت برروي توسعه و تكميل ابزارها و قابليت ها افزايش يافت.
پيشرفتهاي چشمگير PostgreSQL شامل موارد زير مي باشند:
- قفل Table-level با كنترلر هاي چند مفسرهٔ همزمان (multiversion concurrency control) جايگزين شدند، اين تغيير به خواننده ها اجازه مي داد تا به خواندن اطلاعات مشترک كه نويسنده ها در حال نگارش آن بودند به صورت همزمان ادامه دهند و قادر باشند از pg_dump هنگامي كه پايگاه داده براي پاسخ به Query ها آزاد است، پشتيبان بلادرنگ تهيه نمايند.

- ابزارهاي بسيار مهمي از قبيل subselects، defaults،constrains و triggers تعبيه شدند.

- ابزارهاي اضافي سازگار با زبان SQL۹۲ كه به سيستم اضافه شدند شامل Primary key، Quoted identifire، تصحيح كردن تايپ جملات لفظي، Typa casting، ورودي هاي باينري و اعداد صحيح هگزادسيمال بودند.

- Built-in typeها اصلاح شدند و شامل طيف وسيعي از انواع date/timeهاي جديد و انواع جغرافيايي جديد گرديدند.

-هنگامي كه نسخه ٦,۰، انتشار يافت سرعت كليه كدهاي Backe-end بصورت تقريبي۲۰ تا ٤۰ درصد افزايش و زمان آماده به كار شدن آنها تا ۸۰ درصد كاهش يافت .

۳. آنچه در اين كتاب آورده شده

ورود شما را به PostgreSQL و خودآموز آن خوش آمد مي گوئيم، فصلهاي مختصر بعدي، شما را با PostgreSQL آشنا ساخته و براي مبتديان مفاهيم وابسته به پايگاه داده و زبان SQL را بصورت ساده و مقدماتي بيان مي كند. ضمنا ما فقظ فرض كرده ايم كه شما مي توانيدبا كامپيوتر كار كنيد و هيچ نيازي به تجربه كاري خاص با يونيكس يا برنامه نويسي نيست. هدف اصلي كتاب دادن اطلاعات گام به گام در مورد PostgreSQL است بنابر اين شما هم مي توانيده مطالب را كامل دنبال كنيد و هم اينكه يا مراجعه موردي نياز خودرا مرتفع سازيد.
پس از اتمام اين خودآموز مي توانيد PostgreSQL User’s Guide را براي دستيابي به دانش تئوري بيشتر در رابطه با زبان SQL و PostgreSQL Programmer’s Guide را براي توسعه برنامه هاي تحت PostgreSQL مطالعه نماييد، و اگر شما جزو گروهي مي باشيد كه خودتان سرويس دهنده ها را راه اندازي و مديريت مي كنيد، بايد PostgreSQL Administrator’s Guide را مطالعه كنيد.

٤. نگاهي اجمالي بر منابع و مستندات

مستندات و مراجع PostgreSQL در چند كتاب زير سازماندهي گرديده است :

PostgreSQL Tutorial

اطلاعات اوليه براي كاربران جديد

PostgreSQL User’s Guide

مستند سازي زبان درخواست SQL، مشتمل بر الگوهاي داده و توابع. همه كاربران PostgreSQL بايد اين كتاب را مطالعه نمايند.

PostgreSQL Administrator’s Guide

نصب و مديريت سرويس دهنده، هر فردي كه يک سرويس دهنده PostgreSQL را براي استفاده هاي شخصي يا براي كاربران ديگر راه اندازي و مديريت مي كند بايد اين كتاب را مطالعه كند.

PostgreSQL Programmer's Guide
اطلاعات پيشرفته براي توليد كننده برنامه هاي كاربردي، شامل مطالبي چون: الگوها و توابع قابلهاي گسترش، رابط كتابخانه ها و مباحث طراحي برنامه ها.

PostgreSQL Retfrence Manual
صفحات مرجع براي متن دستورات SQL و برنامه هاي سرويس گيرنده و سرويس دهنده ، اين يک كتاب جانبي براي كتابهاي User's، Administrator's و Programmer's Guide مي باشد.

PostgreSQL Developer's Guide
اطلاعات مورد نياز توسعه دهندگان PostgreSQL. اين كتاب مختص كساني است كه در پروژه هاي PostgreSQL همكاري مي كنند؛ اطلاعات مورد نياز توسعه دهندگان برنامه هاي كاربردي در كتاب Programmer's Guide آورده شده است.
در كنار مجموعه فوق يكسري مرجع براي هدايت شما در نصب و استفاده PostgreSQL وجود دارد:

Man Pages
صفحات راهنماي مرجع كه در قالب صفحات راهنماي كلاسيک UNIX عرضه شده و در محتوا شامل هيچگونه تغييراتي نشده است.

FAQs
مجموعه سوالها يي كه به صورت متناوب پرسيده شده اند و حاوي مباحث كلي و يكسري از مسائل وابسته به نوع سيستم عامل مي باشند، به همراه جوابهايشان در مجموعه اي بنام FAQ فهرست بندي و عرضه شده اند.

READMEs
فايلهاي README در بعضي از بسته هاي منتشر شده موجود مي باشند.

Web Site
وب سايت[5] PostgreSQL مطالبي از قبيل جزئيات آخرين نسخه انتشار يافته، آخرين قابليتهاي افزوده شده و بقيه اطلاعاتي كه باعث بارور شدن فعاليت ها و حتي سروكله زدن هاي شما باPostgreSQL شود را ارائه مي دهد.

Mailing List
براي يافتن پاسخ پرسشها، رد و بدل كردن تجربيات خود با ديگران و برقراري ارتباط با توسعه دهندگان،Mailing List مكان بسيار مناسبي است. براي جزئيات بيشتر مي توانيد به بخش Consult theUser's Lounge[6] در وب سايت مراجعه نماييد.

خودتان!
PostgreSQL نتيجه يک تلاش اُپن سورس مي باشد و براي پشتيباني مداوم به اجتماع كاربران نيازمند خواهد بود. همچنين بعد از شروع كار با PostgreSQL، از طريق اسناد ، مراجع و Mailing List به ديگر كاربران و تجربيات آنها تكيه خواهيد زد، بنا براين انعكاس دانش خود به ديگران را مورد ملاحظه قرار دهيد و اگر مطلبي مي آموزيد كه در مستندات و مراجع ذكر نشده و يا قابليتهاي جديد را به كد اصلي اضافه مي كنيد، حتماً اين اطلاعات ارزشمند را براي ديگران به اشتراک بگذاريد.
حتي آنهايي هم كه تجربيات زيادي ندارند مي توانند با اصلاح اشكالات وانعكاس نتيجه تغييرات در

[5] http://www.postgresql.org/
[6] http://www.postgresgl.org/users-lounge/

اسناد و مراجع نقش مهمي را ايفا نمايند. همچنين مي توانيد فعاليتهاي خود را با ارسال نامه هاي الكترونيكي به آدرس <pgsql_docs@postgresql.org> كه يک Mailing List مي باشد آغاز نماييد.

۵. اصطلاحات و نمادها

بطور كلي شخصي كه نصب و راهبري سرويس دهنده ها را به عهده دارد مدير سيستم (Administrator) خطاب مي شود و كاربر كسي است كه توانايي استفاده از PostgreSQL را داشته و خواستاردسترسي به قسمتهاي مختلف PostgreSQL باشد. البته توجه داشته باشيد اين اصطلاحات نبايد به صورت خيلي محدود تفسير شود و اين مستندات و مراجع نيزحدود كاملاً ثابتي را براي رويه هاي مديريت بيان نمي كند.
ما از /usr/local/pgsql به عنوان شاخه ريشه نصب و از /usr/local/pgsql/data به عنوان شاخه محتوي فايلهاي پايگاه ياد مي شود، اين آدرس ها در سايت شما ممكن است متنوع باشند؛ كه در اين رابطه جزئيات بيشتر در Administrator’s Guide قابل دسترس مي باشند.
درنگارش يک دستور براكتها ([,]) نمايان عبارات اختياري يا Keywordها هستند وهر آنچه كه در آكولادها ({,}) قرار مي گيرد و توسط خطوط عمودي "|" از هم جدا مي شوند، نمايان اين هستند كه شما بايد يكي از آن عبارات را انتخاب نمايد.
مثالها دستوراتي را نشان خواهند داد كه از حسابها و برنامه هاي مختلفي اجرا شده اند، ممكن است دستوراتي كه از يک پوسته UNIX اجرا مي شوند با علامت دلار"$" شروع شوند و دستوراتي كه از حساب يک كاربر ويژه مانند root يا postgres بايد اجرا شوند بگونه اي خاص علامت گذاري و به همراه توضيحات ارائه كند. همچنين ممكن است دستورات SQL با علامت ”=>“ شروع شوند ويا بسته به شرايط هيچگونه علامتي در آغاز نداشته باشند.

توجه:

در همه قسمتهاي مستندات و مراجع نكاتي كه در كنار دستورات قيد شده، به صورت متناوب آورده نشده است، لذا مشكلات را با < pgsql_docs@postgresql.org>، Mailing List گروه مستندات درميان بگذاريد.

۶. راهنماي گزارش دهي Bug ها

ما بسيارعلاقه منديم تا از Bug هايي كه شما در PostgrterSQL مي يابيد آگاه شويم. همچنين گزارشهاي شما نقش بسيار مهمي را درمعتبرتر شدنPostgreSQL ايفا خواهند نمود، زيرا منتهاي دقت و تلاش نيز نمي تواند اجراي موفقيت آميزهمه بخشهايPostgreSQL را در انواع سيستمهاي عامل ودر شرايط مختلف تضمين نمايد.
تنها هدف راهكارهاي زير آن است كه با راهنماييهاي مناسب، Bug ها را سامان داده و باعث تأثير بسزا در ساختار پروژه گردند.
ما نمي توانيم قول برطرف شدن هر Bug را به صورت فوري بدهيم، اما شانس اينكه يک شخص Bugهاي بديهي يا بحراني و يا Bug هايي را كه تعداد زيادي از كاربران را دچار مشكل ساخته است مورد بررسي قرار دهد زياد است. البته گاهي به شما گفته مي شود كه از نسخه جديدتري استفاده نماييد تا متوجه شويم آيا مشكل در نسخه جديد برطرف شده است يا خير.
همينطور ممكن است كه ما بيان كنيم كه Bug تا زماني كه قسمتهاي زيادي از برنامه دوباره نوشته نشود قابل اصلاح شدن نخواهد بود وشايد گفته شود رفع آن بسيار سخت است، و حتي با تعداد زيادي از وظايف مهمتر در ليست كاري مواجه باشيم كه از Bug شما در الويت بالاتري باشند. بنابرين اگر شما واقعاً به كمک و پشتيباني فوري نياز داريد حتما يک قرارداد پشتيباني تجاري را مدنظر قراردهيد.

۱,۶. چگونگي تشخيص Bug ها

قبل از اينكه يک Bug را گزارش دهيد بهتر است مستندات را دوباره و دوباره مطالعه نماييد تا دقيقاً متوجه شويد كه آيا چيزي كه در حال تلاش براي آن هستيد، همان مراحلي است كه واقعاً بايد انجام دهيد يا خير. واگر در مستندات مشخص نيست كه مجاز به انجام كاري هستيد يا نه، اينرا هم گزارش دهيد. و اگر متوجه شديد كه مستندات چيزي را مي گويند كه در برنامه بصورت ديگري اجرا مي شود؛ اين نيز يک Bug محسوب ميشود. Bug هاي شما ممكن است جزء موارد زيرباشد اما اين موارد همه گزينه ها را پوشش نمي دهند:

- يک برنامه ممکن است با يک پيغام fatal يا يک خطا از سوي سيستم عامل که به خطا در برنامه اشاره مي کند پايان بپذيرد.(براي نمونه پيغام "disk full" ميتواند شما را وادار به برطرف نمودن مشکل در فضاي ديسک نمايد.)

- برنامه براي هر ورودي، خروجيهاي اشتباه ارائه مي دهد.
- برنامه ورودي هاي صحيح را (همانطور که درمستندات ارائه شده) مورد پذيرش قرار نمي دهد.
- برنامه ورودي هاي بي اعتبار را بدون اينکه پيغامي دهد مي پذيرد. اما مد نظر داشته باشيد که نگرش شما از يک ورودي بي اعتبارممکن است درنظر ما بسط يا تطبيق يک تمرين قديمي باشد.
- PostgreSQL بر اساس ساختار ارائه شده در سيستم عاملي که قابل پشتيباني است،در compile، Build و نصب دچار اشکال شود.

کند بودن و يا کليه منابع را در اختيار گرفتن الزاماً نمي تواند يک Bug محسوب شود. براي رفع اين مسئله مستندات را بازنگري نماييد و يا درMailing List از يک شخص براي کمک در تنظيم نمودن برنامه هايتان کمک بگيريد. عدم برآورده ساختن براساس استاندارد هاي SQL هم الزاماً Bug محسوب نمي شود، زيرا بوضح برآورده نشدن قابليتهاي ويژه ادعا شده است.
قبل از اينکه بيشتر ادامه دهيد لطفاً ليست TODO و FAQ را به دقت بررسي نماييد تا مطمئن شويد که Bug ها قبلاً گزارش نشده باشد. و اگر نمي توانيد اطلاعات موجود در TODO را کشف نماييد، به ما بگوييد تا حداقل TODO ها را واضحتر نماييم.

۲,٦. آنچه قابل گزارش دهي است

مهمترين چيزي که در هنگام گزارش دهي مي بايست مد نظر داشته باشيد اينست که در گزارشات هرچيزي را دقيقاً همانطور که وجود دارد گزارش دهيد. آنچه را که شما تصورمي کنيد ايجاد مشکل نموده، آنچه "به نظر مي رسد بايد انجام شود" واينکه مثلاً گفته شود آن بخش از برنامه بايد اصلاح شود رادر گزارش خود وارد ننماييد. در غير اين صورت اگر شما به اجراي پروسه کاملاً تسلط نداشته باشيد ممکن است با حدس اشتباه خود اصلاً کمکي به ما نکنيد، و اگر شما آشنايي کافي و وافي را با مسئله داريد، توضيحات علمي حامي خوبي خواهند بود اما جانشيني براي واقعيات و رخدادهاي پروسه محسوب نمي شوند.
اگر شما خودتان دست به کاررفع Bug شديد، باز هم ما ميخواهيم که از آن مطلع شويم، گزارش دهي شفاف نيز خيلي آسان است (احتمالاً ميتوانيد آنها را ازروي صفحه نمايش Copy/Paste نماييد) اما اغلب اوقات به دليل اينکه افراد تصور مي کنند حذف يک قسمت مشکلي ايجاد نمي کند ويا بدون آن هم گزارش قابل فهم است، جزئيات بسيار کنار گذاشته ميشود.
کليه گزارشات مي بايست شامل موارد زير باشند:

- ترتيب دقيق مراحل از شروع برنامه و بالا آمدن آن براي ارائه مشکل ضروري است. اگر خروجي به اطلاعات داخل table ها وابسته است، اين نيز کافي نيست که بدون اشاره به کارهاي انجام شده قبلي براي ايجاد table و وارد کردن دستورات اشاره اي نشود.

ما براي revers-engineering پايگاه داده شما وقت نداريم و اگر فرض را به ساختن پايگاه داده خود بگذاريم احتمالا در اين صورت با مشکل برخورد نخواهيم کرد. بهترين فرمت براي بررسي يک مورد در رابطه با مشکلات زبان Queryاينست که از يک فايل که در پنجره psql اجرا شده و مشکل را نمايش مي دهد استفاده کنيم. (مطمئن گرديد که در فايل start-up ~/.psqlrc خود چيزي نداشته باشيد). راه بسيار ساده ديگر براي اين فايل اينست که pg_ dump را براي کشف و واضح ساختن اظهارات table و اطلاعات مورد نياز در تنظيم استفاده کنيد و سپس مشکل در خواست را اضافه کنيد.
هر چند که اين موضوع خيلي هم ضروري نيست اما شما را به کاهش اندازه Bug هاي خود تشويق مي کنيم.

اگر Bug امكان توليد مجدد را داشته باشد ما آنرا بر اساس مراحل شما بدست خواهيم آورد. و چنانچه برنامه شما از رابطه هاي كلاينت ديگري هم از قبيل PHP استفاده مي كند ، بنابراين لطفاً قبل از هر چيز سعي كنيد كه Query آزاردهنده را از بقيه جداسازي نمائيد.
قاعدتاً يك سرويس دهنده Web را براي ايجاد و توليد مشكلات شما تنظيم نخواهيم كرد. و در هر موردي بايد توجه داشته باشيد كه فايل ورودي دقيق را ارائه نمائيد واطلاعاتي مانند اندازه فايل و اندازه پايگاه داده كه از روي حدس و گمان بيان شود ، كمكي نخواهند كرد.

- لطفاً در مورد خروجيها به اين شكل كه آن "كار نكرد" يا "به هم ريخت" گزارش دهي نفرمائيد.

- اگر پيغام خطايي داده شده است، حتي اگرمعني اش رانمي فهميديد، آنرا نمايش داده و براي ما ارسال نمائيد. اگر برنامه با يك پيغام سيستم عامل پايان پذيرفت ، براي ما توضيح دهيد كه آن پيغام چيست. و اگر اصلاً چيزي هم اتفاق نيافتاده، آنرا نيز بگوييد.

درمواردي كه نتايج آزمايش، سيستم شما را با مشكل مواجه ساخته و احتمال رخداد مشابه آن روي سيستم ما وجود ندارد، كپي كردن و ارسال خروجي ها از يك ترمينال بسيار ثمر بخش است.

توجه :
در مورد خطاهاي fatal ممكن است پيغام هاي ارائه شده توسط Client شامل اطلاعات كاملي نباشد، پس حتماً نگاهي به خروخي سرور پايگاه داده بيندازيد. چنانچه شما Log ها را نگهداري نمي كنيد و يا تنظيمي براي log گيري نداريد ، اكنون براي شروع و به كار اندازي اين سرويس وقت مناسبي است.

- توصيف انتظاري كه شما از خروجي داشته ايد بسيار پر اهميت است .مثلاً اگر شما فقط بنويسيد كه "اين دستور خروجي درست را نمي دهد" يا "اين چيزي نيست كه من انتظار داشتم" ، در اين صورت ممكن است ما آن دستور را اجرا نموده و خروجي كاملاً درستي را دريافت نمائيم، همينطور ما نبايد وقت زيادي را صرف كشف معاني توضيحات شما نمائيم. مخصوصاً از اينكه به صورت ساده بگوييد "اين چيزي نيست كه SQL مي گويد يا Oracle انجام مي دهد" جدداً خوداري نمائيد.
همانطور كه ميدانيد بررسي رفتارهاي صحيح SQL چيزي نيست كه در حيطه وظايف ما باشد و همة ما هم نمي دانيم كه كليه پايگاه هاي داده object-relational ديگر چگونه رفتار مي كنند. (اگر مشكل شما در به هم ريختن يك برنامه است ، مي توانيد به سادگي از اين مسئله چشم پوشي نمائيد.)

- يك بار ديگر كليه گزينه هاي خط فرمان و گزينه هاي Start- up شامل متغيرهاي محيطي وابسته به فايلهاي پيكر بندي را كه شما از حالت پيش فرض تغيير داده ايد ، با دقت بررسي كنيد. اگر شما در حال استفاده از يك نسخه از قبل بسته بندي شده هستيد كه سرويس دهنده پايگاه داده را در هنگام بوت شدن راه اندازي مي كند، بايد تلاش خود را براي درك اين مسئله كه آن عمليات چگونه صورت مي پذيرد انجام دهيد.

- هر آنچه را كه برخلاف ساختار و مراحل نصب انجام داده ايد .

- شماره نسخه PostgreSQL. براي اينكه نسخه سرويس دهنده اي را كه به آن متصل شده ايد بيابيد مي توانيد از دستور ;( )version SELECT استفاده نمائيد. همچنين بسياري از برنامه هاي اجرايي از گزينه هاي دستور version-- نيز پشتيباني مي كنند يا حداقل دستورات postmaster --version وpsql --version بايد در آنها قابل اجرا باشد .

اگر توابع يا گزينه هاي فوق قابل اجرا نيستند، بنابراين نسخه برنامه شما بسيار پائين تر از آنست كه احتمالاً قابل به روز شدن باشد ، همچنين مي توانيد به فايل README موجود در شاخه مرجع ، يا شاخه همنام با فايل ويرايش و يا شاخه همنام با بسته نگاهي بيانداريد. اگر يك بسته از قبل

آماده شده مانند RPM ها را اجر ا مي كنيد، آنرا بگرديد و كليه نسخه هاي تابعه اي كه بسته ممكن است داشته باشد را هم مورد بررسي قرار دهيد.
اگر در مورد يك CVS snapshot صحبت مي كنيد ، حتماً به آن دقت نمائيد و تاريخ و زمان آنرا نيز قيد كنيد.
در حالي كه نسخه شما ۷,۳,۲ قديمي تر باشد با جديت و اطمينان، به شما توصيه خواهيم كرد كه حتماً عمل به روز سازي را انجام دهيد. بسياري از Bug در هريك از نسخه هاي منتشر شده جديد ، برطرف شده است كه به خاطر همين تغييرات نسخه هاي جديدرا منتشر ساخته ايم.

اطلاعات سيستم شامل اطلاعاتي نظير: نام نسخه كرنل ، كتابخانه هاي C ، نوع پردازنده و اطلاعات حافظه مي باشد. در بسياري از موارد نام توليد كننده ونسخه آن كافي است، ولي اينطور تصور نكنيد كه هر كسي بداند "Debian" واقعاً حاوي چه چيزهاي است. يا انتظار داشته باشيد همه افراد روي پنتيوم كار كنند. اگر شما درنصب دچار شكل هستيد ، اطلاعاتي راجع به compilers، make و ...، بسيار ضروري خواهد بود.

اگر Bug شما خيلي طولاني شده است از ادامه كار خودداري نكنيد، يكي از حقايق زندگي است كه بهتر است همه چيز را در ابتدا بگوئيد تا اينكه ما مجبور باشيم حقايق را از شما بيرون بكشيم، همينطور اگر فايلهاي ورودي شما خيلي بزرگ هستند، اين خوب و منصفانه مي باشد كه بپرسيد آيا كسي تمايل دارد به آن نگاهي بياندازد يا خير.
همه وقت خود را صرف اينكه بفهميد با چه چيزي در فايلهاي ورودي مي توانيد مشكل را مرتفع سازيد، ننمائيد . احتمالاً اين عمل كمكي به حل مشكل نخواهد كرد واگر نتيجه اين بود كه در حال حاضر Bug شما قابل برطرف شدن نيست، همچنان براي يافتن راه حل و ارائه آن به ديگران تلاش كنيد. همچنين يكبار ديگر وقت خود را صرف اينكه حدس بزنيد Bug از كجا سرچشمه گرفته است ننمائيد . ما در اولين زمان ممكن آنرا خواهيم يافت. هنگامي كه در حال نوشتن گزارش براي يك Bug هستيد از هيچگونه اصطلاح مبهم استفاده نمائيد. بسته نرم افزاري بصورت كلي "PostgreSQL" و گاهي بصورت مختصر "Postgres" خوانده مي شود . اگر شما بصورت خاص در مورد Backend server صحبت مي كنيد حتماً آنرا قيد كنيد و فقط نگوئيد كه "PostgreSQL" در هم ريخت .
Crash در پراسس Backend server كاملا متفاوت از Crashدر پراسس والد Postmaster است. پس در آن صورت نگوييد كه Postmaster، Crash كرده است و نه برعكس. همچنين برنامه هاي سرويس گيرنده، مانند محيط كار psql كاملاً از Backend serverها مجزا هستند. لطفا در مورد اينكه مشكل در سمت سرويس دهنده است يا سرويس گيرنده ،دقيق باشيد.

۳,٦. چگونه و به كجا بايد گزارش داد

بصورت كلي گزارش Bug را به Mailing list مربوط به Bug ها <psql_bugs@postgresql.org> بفرستيد. كه در آنجا ازشما خواسته شده تا براي email خود يک موضوع گويا انتخاب نمائيد.
از طريق وب سايتي كه متعلق به پروژه مي باشد يعني: http://www.postgresql.org، خواهشمنديم يک فرم مخصوص را كه براي Bug ها طراحي شده است تكميل و ارسال نمائيد. وارد ساختن Bug ها بدين شيوه باعث ارسال email به كليه اعضاي Mailing list <psql_bugs@postgresql.org> خواهد شد.
گزارش Bug را به هريک از كاربران Mailing list مانند < psql-sql@postgresql.org > يا <psql-general@postgresql.org> ارسال نكنيد، اين آدر س هاي براي پاسخ دهي سوالات كاربران است و صاحبان آنها نمي خواهند كه گزارش Bug ها را دريافت نمايند. ازهمه مهم تر اينكه آنها علاقه اي به بر طرف كردن Bug ها ندارند.
گزارشها را به Mailing list ، توسعه دهندگان<psql_hackers@postgresql.org> نيز ارسال ننمائيد . اين ليست براي بحث در حاشيه توسعه PostgreSQL مي باشد و ارسال به آنها در حالي كه مي توانيم گزارش Bug ها را مجزا نگه داريم، جالب نخواهد بود.
اگر مشكل به بازنگري بيشتري نياز داشته باشد ممكن است كه ما تصميم به مطرح كردن مشكل در Mailing list ، <psql_nackers@postgresql.org> ، گرفته و آنرا به بحث بگذاريم .

اگر مشکلات شما در رابطه با مستندات مي باشد ، بهترين محل براي انعکاس آنها <-pgsql docs@postgresql.org> مي باشد. لطفاً در بيان اينکه کدام بخش از مستندات شما را آزرده کرده است دقيق باشيد.
اگر Bug شما در ارتباط با يکي از سيستم هايي است که پشتيباني نشده اند، مي توانيد با فرستادن يک email به< pgsql-ports@postgresql.org> آنرا دنبال نمائيد. بنابراين، ما (وشما) روي انتقال PostgreSQL به سيستم شما کار خواهيم نمود.

توجه :

در اين اثنا ، جهت جلوگيري از گسترش spam ، کليه آدرس هاي emil فوق ، محدود شده و شما براي ارسال email ، ابتدا مي بايست به عضويت ليست در آييد. (هر چند که فرم هاي وب جهت گزارش Bug ها نيازي به عضويت ندارند). همينطوراگر شما تمايلي به دريافت email ها و ترافيک Mailing list نداريد اما مي خواهيد که email هايتان ارسال شود مي توانيد در هنگام تکميل فرم عضويت گزينه noemail را فعال نمائيد. جهت دريافت اطلاعات بيشتر يک email به آدرس <majordomo@postgresql.org> ارسال نماييد و در بدنه آن تنهاکلمه help را بنويسيد.

فصل اول
شروع

۱,۱. نصب PostgreSQL

بديهي است كه براي استفاده از PostgreSQL شما بايد آنرا برروي ماشين خود نصب نمائيد اما قبل از اين كار بهتراست كه نگاهي به اسناد سيستم خود انداخته و با مدير سيستم تماس بگيريد، زيرا گاهي مديران سيستم اين بسته را از قبل نصب و آماده نموده اند. همچنين اين بسته در بعضي از نسخه هاي سيستم عامل موجود بوده و در فرآيند نصب سيستم عامل برروي ماشين نصب مي گردد. در صورتي كه از وجود PostgreSQL روي سيستم خود مطمئن نبوده يا جهت كسب تجربه و آزمايشات شخصي قصد استفاده از آن را داريد مي توانيد خودتان آنرا نصب نمائيد .

نصب PostgreSQL مشكل نبوده و مي تواندتمرين بسيار خوبي باشد. براي اين كار دسترسي يك كاربر عادي كافي است و نيازي به دسترسي Superuser (مانند root) نمي باشد.

اگردر نهايت خودتان تصميم به نصب PostgreSQL گرفتيد بهتر است كه جهت نصب كتابPostgreSQL Adaministrato's guide مراجعه نمائيد و پس از سپري نمودن مراحل نصب بطور كامل مجدداً به اين راهنما برگرديد. توجه داشته باشيد كه متغيرهاي اختصاصي محيط بدرستي و با دقت تنظيم گردند(Appropriate Environment Variables).

اگر مدير سيستم شما تنظيمهاي پيش فرض را هنگام نصب تغيير داده باشد ، شما متحمل كار بيشتري خواهيد شد. براي مثال چنانچه سرويس دهنده پايگاه داده شما يك سرويس دهنده دور (Remote) باشد در اينصورت شما مجبور خواهيد شد كه متغير محيطي PGHOST را متناسب با نام سرويس دهنده پايگاه داده تنظيم نمائيد و گاهي اوقات متغير محيطي PGPORT لازم به تنظيم مي باشد. بطور كلي در صورتي كه شما در حال راه اندازي يك برنامه كاربردي مي باشيد و برنامه از عدم توانايي اتصال به پايگاه داده شكايت مي كند در اين صورت با مدير سيستم خود مشورت كنيد، در صورتي كه مدير سيستم خود شما مي باشيد براي اطمينان از درستي تنظيم متغيرهاي محيطي به مستندات مراجعه نمائيد وچنانچه بطور كامل متوجه مطالب آورده شده در بند اخير نشده ايد بهتراست كه بخش بعدي را بررسي كنيد.

۱,۲. مباني معماري

قبل از شروع كار بايد با معماري و ساختار PostgreSQL آشنا شد. آشنايي با چگونگي عملكرد داخلي بخش هاي گوناگون PostgreSQL در درک ساده ترمطالب اين بخش كمک شاياني خواهد نمود.

از نظر فني، PostgreSQL از مدل سرويس گيرنده / سرويس دهنده (Client / Server) پيروي مي كند. در واقع يك نشست PostgerSQL از فرآيند هاي زير كه با هم همكاري مي كنند، تشكيل مي گردد:

- فرآيند سرويس دهي: اين فرآيند مديريت فايلهاي پايگاه داده، پذيرفتن اتصال به پايگاه داده از سوي برنامه هاي سرويس گيرنده و اعمال در خواست هاي سرويس گيرنده ها روي پايگاه داده را انجام مي دهد. اين برنامه سرور پايگاه داده، Postmaster ناميده مي شود.

- برنامه كابردي سرويس گيرنده (Front-end): اين برنامه عملياتي را با پايگاه داده انجام مي دهد. انواع متفاوتي از برنامه هاي سرويس گيرنده وجود دارد.مثلا مي تواند يک ابزار متن گرا

باشد يا يک برنامه گرافيکي يا يک وب سرور که براي نمايش صفحات وب به پايگاه داده دسترسي دارد و يا يک ابزار خاص براي نگهداري پايگاه داده. به همراه انتشار PostgreSQL تعدادي برنامه سرويس گيرنده نيز ارائه گرديده که اغلب آنها توسط کاربران توسعه داده شده و قابل دسترس مي باشند.
معمولا در مدلهاي Client/Server سرويس گيرنده و سرويس دهنده در ماشينهاي مجزا قرار مي گيرند. دراين صورت آنها براساس پرتکلTCP/TP با يکديگر ارتبا ط برقرار خواهند کرد و مطلب مهمي که شما بايد به آن توجه داشته باشيد اين است که فايلهايي که روي ماشين سرويس گيرنده قابل دسترسي هستند، ممکن است برروي ماشين سرويس دهنده پايگاه داده قابل دسترس نبوده ويا اينکه با اسامي متفاوت موجود مي باشند.
سرويس دهنده PostgreSQL مي تواند به چندين اتصال همزمان ازسوي سرويس گيرنده ها پاسخ دهد براي اين منظور سرويس دهنده براي هريك از اتصالات يك فرآيند مجزا راه اندازي مي کند که اصطلاحاً "Fork" ناميده مي شود و براساس اين ساختار، سرويس گيرنده و فرآيند جديد سرويس دهنده بدون ايجاد مزاحمت مستقيم براي Postmaster اصلي با يکديگر به فعاليت مي پردازند. بدين صورت Postmaster همواره در حالت اجرا شده باقي است و در انتظار تقاضاي اتصال از سوي سرويس گيرنده خواهد بود. البته چنانچه فرآيند بدين صورت طراحي نمي شد، فرآيندهاي سرويس گيرنده و سرويس دهنده مخاطب آن مدام در حال آمدو رفت مي بودند (لازم به ذکر است که کليه فعاليت هاي فوق از نظر کاربران مخفي بوده و ما تنها به دليل کامل ساختن ديد شما از ساختار PostgreSQL به آن اشاره نموديم.).

۱,۳.ايجاد يك پايگاه داده:

اولين راه امتحان اينکه شما به سرور پايگاه داده دسترسي داريد يا نه اينست که سعي کنيد يک پايگاه داده ايجاد کنيد. يک سرور PostgreSQL در حال اجرا مي تواند پايگاه داده هاي زيادي را مديريت کند. معمولا براي هر کاربر يا هر پروژه يک پايگاه داده جدا در نظر گرفته ميشود. غالبا مدير سيستم شما يک پايگاه را براي شما ايجاد کرده است که بايد نام آن را به شما بگويد. در آنصورت شما نيازي به ايجاد پايگاه داده نداريد و مي توانيد از اين مرحله صرفنظر کنيد.
براي ايجاد يك پايگاه داده جديد که در اين مثال mydb ناميده شده است بايد از دستور زير استفاده نمائيد :

```
$ createdb mydb
```

در صورت ايجاد موفقيت آميز اين پايگاه داده چنين پاسخي را دريافت خواهيد نمود:

```
CREATE   DATABASE
```

با دريافت اين پيغام شما پايگاه داده خود را با موفقيت ايجاد نموده اید و مي توانيد باقي مطالب اين بخش را رها نموده و به بخش بعدي مراجعه کنيد .
چنانچه PostgreSQL بدرستي نصب نشده باشد يا اصلاً نصب نگرديده و يا مسيري که بايد به آن دسترسي پيدا کرد بدرستي وارد نشده باشد. پيغام زير در صفحه نمايش ظاهر خواهد شد.

```
createdb: command  not  found
```

در اين صورت سعي کنيد که دستور را با ذکر دقيق آدرس اجرا نمائيد:

```
$/usr/Local/pgsgl/bin/createdb mydb
```

البته اين آدرس روي سيستم شما مي تواند متفاوت باشد. بنابراين با مدير سيستم تماس گرفته و آدرس را همانگونه که او به شما مي گويد وارد نمائيد و يا اينکه مراحل نصب را مجدداً مورد بررسي قرار دهيد.
ممکن است پيغام ديگري به شکل زير دريافت نمائيد :

```
psql: could not connect to server: Connection refused
```

```
        Is the server running locally and accepting
        connections on Unix domain socket "/tmp/.s.PGSQL.٥٤٣٢"?
createdb: database creation failed
```

اين پيغام بدان معني است كه سرويس دهنده آغاز به كار نكرده و يا آغاز به كار آنگونه كه فرمان createdbانتظارش را داشته نبوده است ، بنابراين مجدداً مراحل نصب را بررسي و يا مدير سيستم مشورت نمائيد .
چنانچه شما مجوز لازم جهت ايجاد پايگاه داده را نداشته باشيد پيغامي شبيه به خطوط زير دريافت مي كنيد :

```
        ERROR: CREATE DATABASE: permission denied
        createdb: database creation failed
```

همه كاربران نميتوانند پايگاه داده ايجاد كنند. بنابراين چنانچه PostgreSQL از ايجاد پايگاه داده براي شما خودداري نمود، حتماً با مدير سيستم تماس بگيرد و در صورتي كه مدير سيستم خود شما مي باشيد و PostgreSQL را خودتان نصب و راه اندازي نموده ايد.در اين صورت براي رفع مشكل مي بايست كه با حساب كاربري كه سرويس دهنده را با آن راه اندازي نموده ايد Login نمائيد.[۷]

البته شما مي توانيد پايگاه داده را با اسامي ديگري نيز ايجاد نمائيد ، همچنين PostgreSQL به شما اجازه مي دهد تا تعداد دلخواه پايگاه داده را در يك سيستم ايجاد نمائيد. اسامي پايگاه هاي داده مي بايست كه با حروف الفبا آغاز شوند و حداكثر ٦٣ كاراكتر را در برگيرند. خوب است كه پايگاه داده را همنام با نام كاربري خود كه در حال استفاده از آن مي باشيد ايجاد نمائيد.
در اين صورت ابزارهاي زيادي آن پايگاه داده را بصورت پيش فرض در نظر خواهند گرفت و شما وقت زيادي هنگام تايپ نام پايگاه خود صرفه جويي مي كنيد .
براي ايجاد چنين پايگاه داده اي فرمان ساده زير را تايپ نمائيد.

```
$ createdb
```

براي حذف يك پايگاه داده .چنانچه مالك آن باشيد مي توانيد از فرمان زير بهره بجوئيد:

```
$ dropdb  mydb
```

(در اين دستور، هيچگاه نام كاربري به عنوان نام پيش فرض پايگاه داده تلقي نشده و شما همواره ملزم به وارد نمودن نام پايگاه داده مورد نظر مي باشيد) اين دستور كليه فايلها و اطلاعات موجود در رابطه با پايگاه داده از بين مي برد و آنها قابل بازيافت نمي باشند. بنابراين در اجراي اين فرمان صبور باشيد و پس از دقت كافي آنرا اجرا نمائيد.

۱,٤. دسترسي به پايگاه داده:

بعد از ايجاد داده دلخواه اكنون مي توانيد توسط يكي از طرق زير به آن دسترسي پيدا كنيد :

- psql كه به طور محاوره اي در حال اجرا مي باشد و به شما اجازه وارد كردن، ويرايش و اجراي دستورات SQL را بر روي پايگاه داده خواهد داد.
- استفاده از ابزار گرافيكي موجود مانند PgAccess و يا مجموعه هاي اداري كه داراي ODBC هم مي باشند و توانايي پشتيباني از ايجاد و بازيافت و نگهداري پايگاه هاي داده را فراهم مي آورند. (اين ابزارها در اين خودآموز توضيح داده نشده اند.)
- نوشتن برنامه اي مناسب با نيازهاي شخصي با استفاده از زبانهاي برنامه نويسي.

---

[۷] بعنوان يك توضيح در اين رابطه بدنيست بدانيد كه نام هاي كاربري در PostgreSQL با حساب كاربران در سيستم عامل متفاوت مي باشند. چنانچه شما به يک پايگاه داده متصل شويد. در اين صورت مي توانيد نام كاربري، مختص به PostgreSQL را جهت اتصال انتخاب نمائيد. در غير اينصورت بصورت پيش فرض شما با نام كاربري در PostgreSQL كه شبيه به همان نامي است كه به سيستم Login نموده ايد به پايگاه داده متصل خواهيد شد. چنانچه اين اتفاق روی دهد شما همواره حساب PostgreSQL ی خواهيد داشت كه نامی شبيه به حساب كاربر سيستم عامل داشته، سرويس توسط آن راه اندازی می گردد و اجازه ايجاد پايگاه داده را خاهد داشت. البته بجای اينكه همواره با اين نام كاربری به سيستم Login كنيد می توانيد از گزينه "U-" استفاده كنيد و با نام كاربری مورد نظر خود به پايگاه داده متصل شويد.

در اين رابطه مي توانيد به كتاب راهنمايي برنامه نويسان PostgreSQL Programmer's Guide مراجعه نمائيد.
احتمالا شما تمايل داريد براي بررسي مثالهاي اين خود آموز ازpsql استفاده نمائيد دراين صورت جهت فعال كردن پايگاه داده mydb روي آن از دستور زير مي توانيد بهره بجوئيد:

```
$ psql  mydb
```

چنانچه نام پايگاه داده خالي گذاشته شود ، بطور پيش فرض پايگاه داده اي همنام با نام كاربري شما در نظر گرفته مي شود.
ورود شما بهpsql با پيغام زير همواره است:

```
Welcome to psql ٧,٣,٢, the PostgreSQL interactive terminal.

Type: \copyright for distribution terms
\h for help with SQL commands
\? for help on internal slash commands
\g or terminate with semicolon to execute query
\q to quit

mydb=>
```

به جاي خط آخر ممكن است پيغام زير داشته باشيد.

```
mydb=#
```

معني اين پيغام آنست كه شما براي اين پايگاه داده يك كاربر قدرتمند (Superuser) محسوب مي شويد. در واقع دسترسي شما به اين پايگاه داده شباهت بسيار زيادي به دسترسي كسي دارد كه PostgreSQL رانصب نموده باشد. به صورت كلي يك كاربر قدرتمند (Superuser) كسي است كه تحت كنترل قوانين و محدوديت هاي دسترسي و كنترلي قرار نمي گيرد.
چنانچه شما در استفاده از psql دچار مشكلاتي مي باشيد ، بهتر است كه بخش قبلي برگرديد زيرا علائم تشخيص ايراد در psql با createdb كاملاً شبيه بوده و اگر آنها قبلا بدرستي عمل كرده باشند اكنون نيز پاسخگوي نياز شما خواهند بود.
آخرين پيغامي كه توسط psql نمايش داده شده است علامت آماده باش psql (prompt) آن محسوب مي شود، اين نشانه گوياي اينست كه psql آماده اجراي فرامين و درخواست هاي SQL شما روي پايگاه داده مورد نظرو در محيط آماده شده توسط psql مي باشد.
اكنون دستورات زير را به نوبت وارد نمائيد:

```
mydb=> SELECT version();
version
------------------------------------------------------------
PostgreSQL ٧,٣devel on i٥٨٦-pc-linux-gnu, compiled by GCC ٢,٩٦
(١ row)

mydb=> SELECT current_date;
date
------------
٢٠٠٢-٠٨-٣١
(١ row)

mydb=> SELECT ٢ + ٢;
?column?
----------
٤
```

(١ row)

برنامهpsql داراي تعدادي دستور داخلي است كه جزء دستورات SQL محسوب نمي شوند ، اين دستورات با " \ " آغاز مي گردند. بعضي از اين دستورات همراه با پيغام خوش آمد گويي psql آورده شده است. براي مثال جهت دريافت كمك در رابطه با شكل(syntax) دستورات SQL مي توانيد فرمان زير را وارد نمائيد.

Mydb=> \h

دستور زير براي خروج از psql مي باشد:

Mydb=> \q

اين فرمان باعث خروج ازpsql و نمايش prompt ،پوسته خواهد شد (جهت آشنايي بيشتر با دستورات داخلي psql مي توانيد از فرمان "\?" در prompt، psql بهره بگيريد.) قابليتهاي كامل psql در PostgreSQL Reference Manual آورده شده است و چنانچه PostgreSQL به درستي نصب شده باشد مي توانيد از دستور man psql در پوسته سيستم عامل خود براي دسترسي به مستندات استفاده كنيد.
در اين خود آموز ما قصد نداريم كه به بحث عميق در مورد اين ابزارها بپردازيم بنابراين چنانچه شما خودتان علاقه مند باشيد مي توانيد بطور كامل بر روي آنها تحقيق نماييد.

فصل دوم
زبان SQL

۱,۲. معرفي

اين بخش به شما چگونگي استفاده از SQL، براي انجام عمليات هاي ساده را خواهد آموخت .اين خود آموز تنها يكسري مقدمات را بيان مي كند و به هيچ وجه خودآموز كاملي درباره SQL نخواهد بود. تعداد بسيار زيادي كتاب در رابطه با SQL موجود است كه مي توان به Understanding the New SQL و A Guide to the SQL Standard اشاره نمود.
در مثال زير اينگونه فرض شده است كه شما پايگاه داده mydb را ايجاد نموده ايد و دستور psql را جهت اتصال به پايگاه داده خود اجرا كرده ايد. همچنين مثالهاي اين خود آموز در آدرس src/tutorial از نسخه مرجع منتشر شده PostgreSQL موجود خواهند بود، مي توانيد به فايل README جهت چگونگي استفاده از PostgreSQL مراجعه نمائيد.
اكنون براي شروع كار با مثالها، دستورات زير را وارد نمائيد.

```
$ cd  ..../src/tutorial
$ psql -s mydb
...
mydb=> \i basics.sql
```

دستور i\ باعث مي شود تا دستورات ورودي، از يك فايل خوانده شوند و سوئيچ s- شما را در حالت single step mode قرار مي دهد كه در اين حالت شما را قبل از ارسال هر دستور به سرويس دهنده با مكث مواجه خواهد كرد. دستورات مورد استفاده در اين فصل در فايل basics.sql آورده شده اند.

۲,۲.مفاهيم

PostgreSQL، يك سيستم مديريت پايگاه داده رابطه اي (RDBMS) (Relational Database Management System) مي باشد و اين به معني آن است كه يک سيستم براي مديريت داده در Relations موجود مي باشد. Relation اساسا يك اصطلاح رياضي براي table تلقي مي شود. ذخيره و داده ها بصورت table در اكثر پايگاه هاي داده امروزي بصورت امري بديهي درآمده است، اما خوب است بدانيد كه راه هاي گوناگون و متفاوت ديگري جهت سازماندهي داده ها وجود دارد. فايلها و دايركتوريهاي سيستم عامل هاي خانواده Unix مثال خوبي براي پايگاه داده سلسله مراتبي (hierarchical) مي باشند. پايگاه هاي داده (Object-Oriented) نمونه متفاوت و پيشرفته تري براي مديريت داده ها مي باشند.
هر Table مجموعة نامگذاري شده اي از row ها مي باشد و هر يك از row هاي يك table داراي مجموعه مشخص و مشتركي از column ها است و هريك از column وضعيت مشخصي را در هر row دارا مي باشد.
Table ها داخل پايگاه هاي داده گروه بندي مي شوند و مجموعه اي از يك پايگاه هاي داده توسط يك سرويس دهنده PostgreSQL مديريت مي شود، و نيازي به راه اندازي يك پايگاه داده دسته اي (Cluster) نمي باشد.

۳,۲. ايجاد يک table جديد

Table با انتصاب يك نام مناسب براي آن و درادامه، نامگذاري column ها و مشخص نمودن نوع آنها ايجاد خواهد شد.

```
CREATE TABLE weather (
        City            varchar(٨٠),
        temp_lo         int,            -- low temperature
        temp_hi         int,            -- high temperature
        prcp            real,           -- precipitation
        date            date
);
```

همچنين مي توانيد دستورات فوق را به شكل خطوط نا تمام در psql وارد نمائيد. psql مي تواند تشخيص دهد كه دستور تا وارد كردن ";" به پايان نرسيده است.

كاراكترهاي خالي (كليدهاي newline,tabs,space) به طور كاملا آزادانه مي توانند در دستورات SQL به كار روند. اين بدان معني است كه شما مي توانيد دستورات را به شكلي كاملاً متفاوت با مثال فوق و يا حتي همگي را در يك خط وارد نمائيد. توضيحات را بايد بعد از دو خط فاصله متوالي("--") قرار دهيد، در اين صورت تا پايان خط هر آنچه‌كه بعد از اين علامت بيايد، ناديده گرفته شده و دستور فرض نمي شود. SQL نسبت به حروف كليدي و تشخيص دهنده ها (identifier) حساس نمي باشد، مگر اينكه براي صرفه‌جويي در گيومه (double-quoted"") قرار داده شده باشند (اين كار در مثال فوق صورت نگرفته است.)

Varchar(٨٠) تعيين كننده نوعي از داده مي باشد كه قابليت ذخيره داده هاي رشته تا سقف ٨٠ كاراكتر را دارا مي باشند. int همان نوع صحيح مي باشد. real نيز براي ذخيره اعداد درست با يك نقطه شناور استفاده مي شود. و در نهايت date هم كه خودش كاملاً گويا به نظر مي‌رسد (البته اين مسأله كه column هاي نوع date نيز date ناميده مي‌شود، ممكن است كمي گيج كننده به نظر برسد.)

PostgreSQL از انواع مختلف SQL نظير int، smallint، real، double، precision، char(N)، vachar(N)، date، time، timestamp و interval حمايت مي كند. PostgreSQL مي تواند با استفاده از تعداد دلخواه انواع داده كه كاربر تعريف نموده است، سفارشي شود. در نهايت به جز مواردي كه نياز به پشتيباني خاص در استانداردهاي SQL مي باشد، نام type ها لغات كليدي متن محسوب نمي شوند.

مثال دوم : اطلاعات مربوط به شهرها و موقعيت جغرافيايي آنها را ذخيره مي كند :

```
CREATE TABLE cities (
        name            varchar(٨٠),
        location                point
);
```

نوع point نمونه اي از انواع داده اختصاصي PostgreSQL مي باشد.

در آخر چنانچه به يک table نيازي نداريد و با اينكه مي خواهيد آنرا با table ديگري جايگزين نمائيد بايد از دستور زير استفاده كنيد:

```
DROP TABLE table-name;
```

٤,٢. گسترش Table با Row ها

دستور INSERT براي گسترش يک table توسط رديفها قابل استفاده مي باشد.

```
INSERT INTO weather VALUES ('San Francisco', ٤٦, ٥٠, ٠,٢٥, '١٩٩٤-١١-٢٧');
```

توجه داشته باشيد كه همه انواع داده ها تقريباً از شكل ورودي واضح و آشكاري استفاده مي كنند. و همانطور كه در مثال ديده شد، ثوابتي كه متغييرهاي عددي ساده مي باشند ما بين

علامت (') آورده شده است. نوع date از اين نظر كه چه مقاديري را بپذيرد، كاملاً انعطاف پذير بنظر مي رسد. در اين خود آموز ما به دنبال فرمت هاي واضح و بدون ابهام بوده ايم.
نوع point همانطور كه در مثال زير آورده شده است نيازمند ورودي هايي بصورت جفت مي باشد:

```
INSERT INTO cities VALUES ('San Francisco', '(-١٩٤,٠, ٥٣,٠)');
```

با دستوري كه براي ايجاد table مورد استفاده قرار گرفت، نيازمند آن است كه شما ترتيب column ها را به خاطر بسپاريد. نمونه ديگري وجود دارد كه به شما اجازه مي دهد كه column را بصورت بسيار واضح ليست نمائيد:

```
INSERT INTO weather (city, temp_lo, temp_hi, prcp, date)
VALUES ('San Francisco', ٤٣, ٥٧, ٠,٠, '١٩٩٤-١١-٢٩');
```

همچنين مي توانيد column را بترتيب ديگري ليست نموده وبعضي از آنها را حذف كنيد:

```
INSERT INTO weather (date, city, temp_hi, temp_lo)
VALUES ('١٩٩٤-١١-٢٩', 'Hayward', ٥٤, ٣٧);
```

بسياري از توسعه دهندگان روش دوم را بهتر از روش اول مي دانند. اكنون دستورات فوق را به همراه مقادير آنها براي كاربرد در بخش بعدي وارد نماييد.
شما همچنين مي توانيد از دستور copy به منظور وارد نمودن داده هاي انبوه از يک فايل flat-text استفاده نمائيد. اين شيوه معمولا به اين دليل كه دستور copy سريعتر عمل مي كند، خوب است اما از ديدگاه انعطاف پذيري، نسبت به دستور INSERT انعطاف كمتري را داراست.
به مثال زير در اين رابطه توجه نمائيد:

```
COPY weather FROM '/home/user/weather.txt';
```

توجه داشته باشيد در هنگامي كه سرويس دهنده بصورت مستقيم در حال خواندن فايل است، فايل مورد نظر مي بايست براي سرويس دهنده backend قابل دسترس باشد نه براي سرويس گيرنده. در مورد دستور COPY مي توانيد اطلاعات بيشتري را در PostgreSQL Reference Manual بيابيد.

٥,٢. گرفتن اطلاعات از يک table

براي باز يابي اطلاعات از يك table به آن table، query زده مي‌شود. دستور SELECT در SQL به همين منظور مي باشد.
اين دستور تحت بخش هاي زيرقابل بررسي است:
ليست انتخابي (column هايي را كه مي بايست برگردانده شوند ليست مي كند) ، ليست tableها (table هايي كه اطلاعات از آن باز يابي مي شود را ليست مي كند)، و گزينه هاي شرطي (جهت تعيين محدوديت ها).
براي مثال جهت بازيابي كليه row هاي متعلق به جدول weather دستور زير را تايپ كنيد :

```
SELECT * FROM weather;
```

(در اينجا "*" به معني تمامي ستونها مي‌باشد) خروجي اين دستور به شكل زير خواهد بود:

```
   city           | temp_lo | temp_hi | prcp |   date
------------------+---------+---------+------+------------
San Francisco |      ٤٦ |      ٥٠ | ٠,٢٥ | ١٩٩٤-١١-٢٧
San Francisco |      ٤٣ |      ٥٧ |    ٠ | ١٩٩٤-١١-٢٩
Hayward       |      ٣٧ |      ٥٤ |      | ١٩٩٤-١١-٢٩
```

(٣ rows)

ممكن است كه شما عبارتهاي دلخواهي را براي ليست نهايي مشخص كنيد. براي مثال مي توانيد دستوري به شكل زير وارد نمائيد:

SELECT city, (temp_hi+temp_lo)/٢ AS temp_avg, date FROM weather;

كه نتيجه اين چنين خواهد بود:

```
        city         | temp_avg |      date
---------------------+----------+------------
San Francisco  |         ٤٨    | ١٩٩٤-١١-٢٧
San Francisco  |         ٥٠    | ١٩٩٤-١١-٢٩
Hayward           |       ٤٥    | ١٩٩٤-١١-٢٩
(٣ rows)
```

توجه داشته باشيد كه گزينه AS چگونه براي لقب دادن مجدد يك column خروجي مورد استفاده قرار گرفته است (اين گزينه اختياري است) .
عملگرهاي بولين(Boolean) مثل AND,OR and NOT در مشروط ساختن يك Query قابل استفاده مي باشند. براي نمونه، مثال زير اطلاعات مربوط به آب وهوا را در روزهاي باراني فرانيسكو باز يابي خواهد نمود :

```
SELECT * FROM weather
WHERE city = 'San Francisco'
AND prcp > ٠,٠;
```

نتيجه:

```
  City              | temp_lo | temp_hi |    prcp   | date
--------------------+---------+---------+-----------+------------
San Francisco  |      ٤٦    |      ٥٠  |     ٠,٢٥  | ١٩٩٤-١١-٢٧
      (١ row)
```

بعنوان آخرين نكته اين مطلب را به ياد داشته باشيد كه شما مي توانيد توسط select اطلاعات را بصورت ترتيب ذخيره سازي، و يا با حذف row هاي تكراري باز آوري نمائيد .

```
SELECT DISTINCT city
FROM weather
ORDER BY city;
city
--------------
Hayward
San Francisco
(٢ rows)
```

ORDER BY, DISTINCT مي توانند بصورت جداگانه نيز مورد استفاده قرار گيرند.

٦,٢. ايجاد ارتباط ما بين table ها

بخش قبلي به ما آموخت كه در يک زمان ، Query هاي ما مي توانند داده هاي يك table را بازيابي نمايند و يا اينكه به يك table به گونه اي دسترسي پيدا كنند كه row هاي گوناگوني از آن در آن واحد قابل پردازش باشد. Query هايي كه دريك زمان واحد به چندين row از يك table يا چندين table متفاوت دسترسي مي يابند را Query اتصال (join query) مي نامند بعنوان يك مثال، ممكن است شما بگوئيد كه مي خواهيد همه اطلاعات ثبت شده در table ، weather را همراه با مكان مرتبط با هريك از وضعيت هاي آب و هوايي ليست نمائيد. براي اين منظور ما بايد column هاي city متعلق به هر يك از row هاي table، weatherرابا نام column متعلق به همه row ها در table ،cities مقايسه نموده و جفت row هايي كه اين اطلاعات بين آن يكسان است را انتخاب نمائيم.

توجه :

مثال آورده شده تنها نمونه اي است براي درك بهتر مفهوم query هاي اتصال ،اتصال هاي واقعي هر چند كه از ديدكاربر مخفي مي باشند اما مي توانند بصورت بسيار مؤثر تر و كاربردي تر شكل داده شوند .

نمونه توضيح داده شد توسط Query زير قابل دسترسي مي باشد :

```
SELECT *
FROM weather, cities
WHERE city = name;
```

```
      city       | temp_lo | temp_hi | prcp |    date    |     name      | location
-----------------+---------+---------+------+------------+---------------+----------
San Francisco |      ٤٦ |      ٥٠ | ٠,٢٥ | ١٩٩٤-١١-٢٧ | San Francisco | (-
١٩٤،٥٣)
San Francisco |      ٤٣ |      ٥٧ |    ٠ | ١٩٩٤-١١-٢٩ | San Francisco | (-
١٩٤،٥٣)
```

در مورد اطلاعات خروجي همواره به ٢ مسأله توجه داشته باشيد:

- Query اتصال، row هاي تطبيق داده نشده را ناديده مي گيرد و از آنجائي كه شهر Hayward هيچ گونه تطبيق ورودي در جدول cities ندارد ، در ليست row هاي به دست آمده هيچ گونه داده اي در مورد Hayward ديده نمي شود. ما بطور خلاصه كه با راه حل اين مورد آشنا خواهيم شد .

- دو column وجود دارد كه حاوي نام شهر است ، اين مطلب بدون اشكال است و دليل آن اينست كه ليست column هاي جدول weather و جدول cities به يكديگر افزوده شده اند. در تمرينات اين مسئله بطور نا خواسته پيش مي آيد بنابراين چنانچه شما column هاي خروجي را به جاي اينكه با "*" مشخص نمائيد به طور صريح مشخص كنيد، مي توانيد به شكل زير عمل نمائيد:

```
SELECT city, temp_lo, temp_hi, prcp, date, location
FROM weather, cities
WHERE city = name;
```

تمرين: تلاش كنيد كه مفهوم Query زير را هنگامي كه عبارت WHERE حذف شده است بيابيد.

تا زماني كه همهcolumn ها داراي نامهاي مختلف مي باشند ، مفسر گرامري توانايي آن را دارد كه بصورت خود كار تشخيص دهد column ها به كدام يك از جدول ها تعلق دارند، اما روش بهتر اينست كه براي Query هاي اتصال نام column كاملاً مشخص شوند .

```
SELECT weather.city, weather.temp_lo, weather.temp_hi,
weather.prcp, weather.date, cities.location
FROM weather, cities
WHERE cities.name = weather.city;
```

Query هاي اتصالي كه با هم ديديم مي تواند به فرم زير هم درخواست شوند:

```
SELECT *
FROM weather INNER JOIN cities ON (weather.city = cities.name);
```

اين فرم دستور به انداز فرم اوليه كه درخطوط بالا با هم ديديم رايج نيست، اما آوردن اين نحوة دستور نويسي براي درك بهتر مفاهيم زير لازم است.
در اينجا ما مي خواهيم ياد بگيريم كه چگونه داده هاي ثبت شده Hayward را برگردانيم. كاري كه از Query مي خواهيم اينست كه جدول weather را بررسي نموده و براي هر يك از رديفهاي آن، رديفهاي جدول cities منطبق شده را بيابيد، چنانچه هيچ رديف منطبقي يافت نشد، "مقدار خالي (empty value)" را به جاي ستونهاي جدول cities قرار دهيد. اين نوع از Query ها اتصالهاي بيروني (outer joins) ناميده مي شوند. (اتصال‌هاي قبلي از نوع اتصالهاي دروني (inner joins) بودند.)
بدين منظور بايد از دستور زير بهره بگيريد :

```
SELECT *
FROM weather LEFT OUTER JOIN cities ON (weather.city = cities.name);
```

```
City            | temp_lo | temp_hi | prcp  |    date       |  name          | location
----------------+---------+---------+-------+---------------+----------------+----------
Hayward         |      ٣٧ |      ٥٤ |       | ١٩٩٤-١١-٢٩ |                |
San Francisco |      ٤٦ |      ٥٠ |  ٠,٢٥ | ١٩٩٤-١١-٢٧ | San Francisco| (-
١٩٤،٥٣)
San Francisco |      ٤٣ |      ٥٧ |     ٠ | ١٩٩٤-١١-٢٩ | San Francisco| (-
١٩٤،٥٣)
(٣ rows)
```

اين يك Query بيروني چپ (left outer query) ناميده مي شود و دليل اين نامگذاري اينست كه هريك از رديفهاي سمت چپ اتصال دهنده در جدول اشاره شده، حداقل يكبار در خروجي ديده خواهد شد. يا اينكه جدول سمت راست، فقط رديفهايي در خروجي خواهد داشت كه با تعدادي از رديفهايي سمت چپ منطبق شده باشند هنگامي كه در خروجي براي رديفهاي جدول چپي هيچ گونه انطباقي در جدول راستي وجود نداشته باشند، مقادير خالي براي ستونهاي جدول راستي جايگزين خواهد شد.
تمرين: در اينجا اتصال دهنده‌هاي بيروني راست (right outer joins) و اتصال دهنده‌هاي بيروني كامل (full outer joins) موجود مي باشد. تلاش كنيد تا بيابيد كه هريك از آنها چه كاري انجام مي دهند.
همچنين ما ميتوانيم يك جدول را با خودش اتصال دهيم ، كه اين پديده را اتصال به خود (self join) مي‌نامند . به عنوان يك مثال ما مي خواهيم تمام داده هاي ثبت شده آب و هوايي را كه در رنج دماي داده هاي ثبت شده آب وهواي ديگران مي باشد بيابيم . بنابراين نياز داريم تا ستونهاي temp-lo و temp-hi مربوط به هرردیف در جدول weather را با ستونهاي temp-lo و temp-hi در رديفهاي ديگر مقايسه نمائيم، با Query زير اين عمل امكان پذير خواهد بود:

```
SELECT W١.city, W١.temp_lo AS low, W١.temp_hi AS high,
W٢.city, W٢.temp_lo AS low, W٢.temp_hi AS high
FROM weather W١, weather W٢
WHERE W١.temp_lo < W٢.temp_lo
AND W١.temp_hi > W٢.temp_hi;

        City       |    low     |  high      |         city         |   low    | high
-------------------+------------+------------+----------------------+----------+------
San Francisco |      ٤٣   |      ٥٧    | San Francisco |      ٤٦  |    ٥٠
Hayward          |      ٣٧   |      ٥٤    | San Francisco |      ٤٦  |    ٥٠
(٢ rows)
```

در اينجا ما براي تشخيص طرف سمت راست و چپ اتصال، جدول weather را به w١و w٢ تغيير نام داده ايم. اينكار باعث صرفه جويي در تايپ نيز مي شود. راه ساده تري نيز براي جلوگيري از تايپ طولاني وجود دارد كه در اينجا با هم مي بينيم:

```
SELECT *
FROM weather w, cities c
WHERE w.city = c.name;
```

در طول كار با اين شيوه مختصر نويسي بارها برخورد كرده و آشنايي كافي را كسب خواهيد كرد .

٧,٢. توابع محاسباتي (Aggregate Functions)

مشابه پايگاه هاي داده رابطه اي (Relational) ديگر، PostgreSQL نيز از توابع محاسباتي پشتيباني مي نمايد. يك تابع پيوند از چندين رديف وروردي يك نتيجه را محاسبه مي نمايد. براي مثال يك تابع محاسباتي براي محاسبه avg(average)، sum، count، min(minimum) و max(maximum) برروي مجموعه اي از رديفها وجو دارد . برا ي نمونه ما ميتوانيم بالاترين دما را از مجموعه دماهاي پايين بيابيم :

```
SELECT max(temp_lo) FROM weather;
max
-----
٤٦
(١ row)
      )
```

چنانچه بخواهيم بدانيم خواندن اطلاعات از كدام شهر يا شهرها صورت گرفته است، احتمالاً مي بايست دستور زير را به كار بنديم:

```
SELECT city FROM weather WHERE temp_lo = max(temp_lo); WRONG
```

اما اين مسأله تا زماني كه تابع max نتواند در عبارت WHERE استفاده شود عملي نخواهد شد. (اين محدوديت بدليل اينكه عبارت WHERE رديفهايي كه به مرحله محاسبات وارد مي شوند را تشخيص مي دهد،وجود خواهد داشت. بنابراين قبل از اينكه محاسبات توابع كامل شوند بايد سنجيده شوند.) هرچند در اغلب موارد مشابه، Query براي دسترسي به نتايج دلخواه بازنگري مي شود اما در اينجا ما توسط يك subquery اين كار را انجام داديم:

```
SELECT city FROM weather
WHERE temp_lo = (SELECT max(temp_lo) FROM weather);
city
```

```
---------------
San Francisco
(١ row)
```

اين دستور صحيح بوده و خروجي درستي را ارائه مي دهد. و دليل اين امر نيز آنست كه subquery استفاده شده محاسبات موجود در تابع خود را كاملاً مستقل از آنچه كه در Queryهاي ديگر رخ مي دهد انجام خواهد داد.
توابع محاسباتي براي تركيب با عبارات Group BY خيلي كاربردي و مفيد مي‌باشند.
براي مثال ما مي توانيم بالاترين دما در ردة دماهاي پائين ثبت شده براي هر شهر را از طريق زير نيز بدست آوريم:

```
SELECT city, max(temp_lo)
FROM weather
GROUP BY city;
```

```
    city          | max
------------------+-----
Hayward           | ٣٧
San Francisco | ٤٦
(٢ rows)
```

اين دستور يك رديف خروجي براي هريك ازشهرها خواهدداد. هريك از نتايج محاسبات براساس رديف جدولهاي منطبق شده با شهرها محاسبه مي شود. اين رديف هاي دسته شده توسط عبارت HAVING قابل فيلتر شدن مي باشد.

```
SELECT city, max(temp_lo)
FROM weather
GROUP BY city
HAVING max(temp_lo) < ٤٠;
```

```
  City          |    max
----------------+---------
Hayward     |    ٣٧
(١ row)
```

اين دستور فقط براي شهرهايي كه تمام دماي پا يين آن زير٤٠ باشد، نتيجه مشابهي را فراهم مي آورد، درنهايت چنانچه فقط اطلاعات مربوط به شهرهايي را كه نام آنها با "S" شروع مي شود بخواهيم دستور زيررا به كارگيريم:

```
SELECT city, max(temp_lo)
        FROM weather
        WHERE city LIKE 'S%'
        GROUP BY city
        HAVING max(temp_lo) < ٤٠;
```

عملگر LIKE عمل انطباق را انجام ميدهد و در كتابPostgreSQL User's Guide كاملاً تشريح شده است.

درك ارتباط محاوره اي بين توابع محاسباتي و عبارات WHERE و HAVING ، در SQL خيلي مهم است. تفاوت اصلي بين WHERE و HAVING از اين قر ار مي باشد: WHERE،رديفهاي ورودي راقبل از دسته ها (groups) انتخاب كرده و محا سبات را بد ين سان انجام مي دهد، (بدين صورت كنترل اينكه كدام رديف وارد محاسبات شود، دردست WHERE خواهد بود) ازطرف ديگر HAVING محاسبات را با انتخاب رديفهاي دسته اي بعد از دسته ها، انجام مي دهد. بنابر ين قاعده WHERE نبايد حاوي توابع محاسباتي باشد.

چنانچه بخواهيد مي توانيد از يك محاسبات همراه با WHERE استفاده كنيد وببينيد كه كدام يك ازرديفها بعنوان ورودي محاسبات وارد خواهد شد .به عبارت ديگر در مورد HAVING مي توان گفت كه آن هميشه حاوي توابع محاسبات مي باشد.

(در اين مورد خيلي بادقت بايد صحبت كرد ،زيرا هرچند كه بي ارزش خواهد بود اما شما اجازه ي نوشتن قاعده HAVING ي راداريد كه در آن از توابع محاسبات استفاده نمي شود. در اين رابطه شرايط مشابهي بصورت خيلي مؤثر در مورد مراحل WHERE نيز صدق مي كند.)

توجه داشته باشيد كه مي توان محدوديت هاي وابسته به نام شهرها رابه WHEREاضافه‌كنيم. (هنگامي كه به محاسبات نيازمند هستيم اين شيوه نسبت به اضافه كردن محدوديت ها بهHAVING ثمربخش تراست . زيرا از دسته بندي ومحاسبات براي همه رديفها يي كه در بررسي WHERE مردود شده اند خودداري كرده ايم.)

۷,۲. به روز سازي ها

با استفاده از دستور UPDATE قادر خواهيد بود كه رديفهاي موجود را به روز نما ئيد. مثلاً چنانچه از شما خواسته شد كه دماهاي موجود در۲۰ نوامبر را با ۲ درجه كاهش ثبت كنيد، بايد داده ها را با دستور زير به روز سازيد.

```
UPDATE weather
    SET temp_hi = temp_hi - ۲, temp_lo = temp_lo - ۲
    WHERE date > '۱۹۹٤-۱۱-۲۸';
```

اكنون به وضعيت جديد داده ها نگاهي بيند ازيد:

```
SELECT * FROM weather;

        city        | temp_lo | temp_hi |  prcp    | date
--------------------+---------+---------+----------+------------
San Francisco |        ٤٦ |       ٥۰ |   ۰,۲٥  | ۱۹۹٤-۱۱-۲۷
San Francisco |        ٤۱ |       ٥٥ |      ۰   | ۱۹۹٤-۱۱-۲۹
Hayward         |        ۳٥ |      ٥۲  |          | ۱۹۹٤-۱۱-۲۹
(۳ rows)
```

۹,۲. پاكسازي

در شرايط خاصي ممكن است كه ديگر نيازي به وضعيت آب و هوايي Hayward نداشته باشيد. در اين شرايط شما بايد دستورات زيررا وارد نماييد تا رديفهاي مورد نظر را از جدول مربوط پاك نماييد.

پاكسازي با استفاده از دستور DELETE قابل اجرا خواهد بود.

```
DELETE FROM weather WHERE city = 'Hayward';
```

با وارد كردن اين دستور كليه داده هاي ثبت شده كه متعلق به Hayward مي باشند، حذف خواهد شد.

```
SELECT * FROM weather;

        city          | temp_lo  | temp_hi |  prcp  | date
```

```
-------------------+------------+-----------+--------+-----------
San Francisco |      ٤٦ |      ٥٠ |   ٠,٢٥ | ١٩٩٤-١١-٢٧
San Francisco |      ٤١ |      ٥٥ |      ٠ | ١٩٩٤-١١-٢٩
 (٢ rows)
```

فقط يك مطلب نگران كننده در مورد شكل كلي دستور وجود دارد:

```
DELET  FORM  table-name;
```

وآن اينست كه بدون مشروط ساختن، DELETE كليه رديفهاي يك جدول را دور ريخته و آن را خالي مي كند و سيستم بدون پرسيدن هيچگونه تاييد اين كار را انجام مي دهد.

فصل ٣
ابزارهاي پيشرفته

١,٣. معرفي

در فصل قبلي مباني استفاده از زبان SQL ، طريقه ايجاد، ذخيره سازي و دسترسي به داده ها را در PostgreSQL آموختيم. اكنون با تعدادي از ابزارهاي پيشرفته SQL كه مديريت را ساده تر كرده و باعث جلوگيري از آسيب پذيري داده ها مي شوند، آشنا مي شويم. در آخر نيز به تعدادي از متعلقات PostgreSQL نگاهي خو اهيم انداخت.
در بعضي موارد، تغييراتي را در مثالها آورده شده در فصل ٢ ايجاد مي كنيم و به توسعه و تغيير در آنها مي پردازيم، لذا مطالعه فصل ٢ در درک بهتر و مفيدتراين فصل كمک شاياني خواهد نمود. بعضي مثالهاي اين فصل نيز در فايل advanced.sql و در شاخه tutorial قابل دسترسي هستند. همچنين اين فايل حاوي يكسري داده براي بارگذاري مي باشد كه در اين خودآموز اشاره اي به آنها نشده است. (به قسمت ١-٢ براي چگونگي استفاده از اين فايل مراجعه نمائيد .)

٢,٣. views

مجدداً به Query هاي ٦-٢، مراجعه نمائيد ، در اينجا شما با ركوردهاي آب وهوا كه با موقعيت شهرها برحسب كاربرد شما ليست شده اند مواجه ايد، اما شما نمي خواهيد هر موقع كه به درخواست نياز داريد آنرا مجدداً تايپ كنيد، براي اين مقصود شما مي توانيد درخواست خود را با ايجاد يک view پوشش دهيد، اين پوشش يک نام را به Query شما نسبت مي دهد و شما مي توانيد هر موقع كه خواستيد به آن مراجعه كنيد (دقيقاً همانطوري كه يک جدول را صدا مي زديد.)

```
CREATE VIEW myview AS
        SELECT city, temp_lo, temp_hi, prcp, date, location
        FROM weather, cities
        WHERE city = name;
        SELECT * FROM myview;
```

ايجاد view هاي نامحدود يكي از كليدي ترين خصوصيات خوب SQL در طراحي پايگاه داده مي باشد . view ها به شما اجازه مي دهند تا جزئيات ساختار جدول هاي خود را در پشت رابط هاي پيش بيني شده جاي دهيد.
view در اكثر مواردي كه يک جدول واقعي استفاده شود، قابل اجرا مي باشد. ساختن view هاي جديد تحت viewهاي موجود نيز معمول است.

٣,٣. كليدهاي خارجي (Foreign keys)

دوباره جدول هاي weather و cities را از فصل ٢ مورد بررسي كنيد و اين مساله را مورد توجه قرار دهيد: شما تصميم داريد مطمئن شويد كه كسي نمي تواند رديفهايي را كه هيچگونه

انطباقي در جدول cities ندارد وارد جدول weather نمايد، اين امر نگهداري از سلامت ارجاعي (referential integrity)داده هاي شما ناميده مي شود. در سيستم پايگاههاي داده پيچيده نيز در ابتدا با مراجعه به جدول cities و چک کردن اين مطلب که آيا رکورد منطبقي يافت مي شود يا خير؛ کار آغاز و سپس رکوردهاي weather وارد يا خارج مي شوند. اين طريقه شروع با يکسري معايب همراه بوده و بسيار پر زحمت است، از اينرو PostgreSQL اين کار را براي شما انجام مي دهد. بنابر مطالب جديد فوق جدولها به شکل زير خواهند بود:

```
CREATE TABLE cities (
        city            varchar(٨٠)   primary      key,
        location point
);


CREATE TABLE weather (
        city                    varchar(٨٠)   references   cities,
        temp_lo                 int,
        temp_hi                 int,
        prcp                    real,
        date                    date
);
```

اکنون تلاش خود را براي وارد کردن رکوردهاي صحيح بکار مي بنديم:

```
INSERT INTO weather VALUES ('Berkeley', ٤٥, ٥٣, ٠.٠, '١٩٩٤-١١-٢٨');

ERROR: <unnamed> referential integrity violation - key referenced from weather not
```

رفتارکليدهاي خارجي مي تواندکاملاً به برنامه کاربردي شما برگردد.
ما به اين مثال کوچک بسنده کرده و شما را به مطالعهPostgreSQL User's Guide تشويق مي کنيم. استفادهٔ صحيح از کليدهاي خارجي کيفيت برنامه هاي کاربردي پايگاه داده را واقعاً افزايش مي دهند، لذا شما را به يادگيري و تسلط به آن تشويق مي کنيم.

٤,٣. Transaction

Transaction يک مفهوم پايه براي همه سيستم هاي پايگاه داده مي باشد. مقدماتي ترين نکته در مورد Transaction اين است که آن چندين عمل را طي يک عملگر انجام مي دهد (يا همه با موفقيت انجام مي شوند يا هيچکدام انجام نمي شود). وضعيت عملکرد يک Transaction بر روي گام هاي آن براي Transactionهاي ديگري که بصورت همزمان در حال اجرا مي باشند، قابل رويت نبوده و چنانچه تعدادي از آنها در عمليات دچار مشکل شده و از کامل شدن Transaction جلوگيري نمايند، تأثير هيچ يک از گام ها برروي پايگاه داده اعمال نخواهد شد. براي مثال فرض کنيد با پايگاه داده يک بانک مواجه هستيد. اين بانک همانطور که براي شعبات گوناگون داراي توازن در سپرده هاي بانکي است، از حساب هاي متوازن براي مشتريهاي گوناگون نيز پشتيباني مي کند. فرض کنيد که ما يک رکود حاوي برداشت مبلغ ١٠٠٠٠٠ دلار را از حساب Alice و ارسال آن به حساب Bob داريم. دراين رابطه دستورات SQL به شکل زير صورت مي گيرند.

```
UPDATE accounts SET balance = balance - ١٠٠.٠٠
WHERE name = 'Alice';

UPDATE branches SET balance = balance - ١٠٠.٠٠
WHERE name = (SELECT branch_name FROM accounts WHERE name = 'Alice');

UPDATE accounts SET balance = balance + ١٠٠.٠٠
WHERE name = 'Bob';
```

```
UPDATE branches SET balance = balance + ۱۰۰,۰۰
WHERE name = (SELECT branch_name FROM accounts WHERE name = 'Bob');
```

جزئيات اين دستورات در اينجا اهميت زيادي ندارد؛ مهم اينست كه براي اعمال اين تغيير در پايگاه داده نياز به چندين روند به روز سازي مجزا مي باشد. ممكن است مسئولين بانک بخواهند مطمئن شوند كه آيا همه اين تغيرات اعمال شده، يا هيچ يک انجام نپذيرفته است. مطمئناً چنين اتفاقي رخ نخواهد داد كه بدون كم شدن ۱۰۰۰۰۰ دلار از حساب Alice همان مبلغ به حساب Bob وارد شود، همچنين Alice از اينكه بدون هيچگونه اعتبار واطميناني، پول را به Bob پرداخت كند خشنود نخواهد بود. بنابراين ما نياز به اين تضمين داريم كه اگر يكي از عملگرها و گام هاي پروسه با مشكل مواجه شد، گامهايي كه با موفقيت انجام پذيرفته اند نيز هيچگونه تغييري روي پايگاه داده اعمال نكند. به روز سازي گروهي با استفاده از Transactionها اين تضمين را به ما خواهند داد. يک Transaction،از نگاه Transactionهاي ديگرatomic است.

تضمين ديگري كه ما دراين رابطه به آن نيازداريم اينست كه هنگامي كه يک Transaction كامل شد و از سوي سيستم پايگاه داده تأئيد گرديد، اين ركورد بصورت دائمي ثبت شود و هر گونه crash در سيستم اين ركورد را از بين نبرد. براي مثال Bob از حساب خود يک برداشت نقدي دارد، به هيچ عنوان خوشايند نيست كه با بيرون رفتن از بانک و در اثر يک crash اسناد اين پرداخت ناپديد شوند. يک سيستم transactional پايگاه داده با log كردن كليه به روز سازي ها در يک storage پايدار و هميشگي قبل از اينكه پيغام تكميل را بفرستد سيستم را تضمين مي كند.

ويژگي مهم ديگر كه پايگاههاي داده اي كه از Transaction پشتيباني مي كنند كاملاً به نظريه به روز سازي atomic وابسته است: هنگامي كه چندين Transaction به صورت همزمان در حال اجرا مي باشند هيچكدام نبايد قادر به ديدن تغييرات ايجاد شده و تكميل نشده توسط ديگران باشند براي مثال ، چنانچه يكTransaction در حال ايجاد توازن بين همه شعبات است، آن Transaction اين كار را به عهده نخواهد گرفت كه بدون دادن اعتبار به شعبه Bob از شعبه Alice برداشت كند (و نه بر عكس اين قضيه).

بنابراين Transaction ها علاوه بر تضمين ثبت تغييرات به شكل پايدار، بايد بصورت همه يا هيچ يک باشند، يعني يا همه درست اجرا شده و تغييرات را ثبت كنند و يا اينكه در صورت بروز خطا در يكي از گامهاي يک Transaction هيچ يک از تغييرات مربوط به گامهاي ديگر روي پايگاه داده اعمال نشود. اين مطلب در مورد Transaction ها قابل رويت است.

مراحل به روزسازي توسط يک Transaction براي ديگر Transactionها تا هنگامي كه كليه گا مراحلها با موفقيت به اتمام برسند و تغييرات اعمال شود قابل رويت نيستند.

در PostgreSQL يک Transaction با دستورات SQL ، COMMIT, BEGIN احاطه مي شود. بنابراين در ارتباط با مثال بانک شكل دستور مورد نياز اينگونه است:

```
BEGIN;
        UPDATE accounts SET balance = balance - ۱۰۰,۰۰
        WHERE name = 'Alice';
        -- etc etc
COMMIT;
```

چنانچه در ميان اجراي Transaction تصميم بگيريم كه از تكميل از صرفنظر كنيم (براي مثال چنانچه متوجه شديم كه شعبه Alice با مشكل مواجه است) در اين جا مي توانيم از دستور ROLL BACK به جاي COMMIT استفاده كنيم و به اين شكل همه به روز سازي ها لغو خواهند شد.

PostgreSQL دقيقاً مطابق يكايک دستورات SQL كه در Transaction آورده شده رفتار مي كند و آنها را به اجرا مي گذارد. در صورتي كه شما از دستور BEGIN استفاده نكنيد، هريک از دستورات با يک BEGIN ضمني (چنانچه دستور با موفقيت همراه باشد) و COMMIT احاطه خواهند شد . بلوک Transactionاصطلاحي است كه گاهي به اين گونه دستورات كه با BEGIN و COMMIT احاطه شده اند اطلاق مي شود.

نکته : در library بعضي از رابطه هاي سرويس گيرنده دستورات BEGIN و COMMIT بصورت خودکار الصاق مي شوند. بنابراين ممکن است شما بدون هيچگونه در خواستي از مزاياي Transaction بهره مند شويد.

۵,۳. وراثت (Inheritence)

وراثت مفهومي است که در پايگاههاي داده مستقل (Object-Oriented) کاربرد دارد و قابليت هاي جديدي را جهت طراحي پايگاه داده در اختيار طراحان مي گذارد.
بياييد دوجدول ايجاد نمائيم: يک جدول با عنوان cities و ديگري با عنوان capitals.
قاعدتاً مراکز شهرها نيز شهر محسوب مي شوند، بنابراين شما نياز به راهي داريد که هنگامي که ليست شهرها را درخواست مي کنيد مراکز شهرها نيز در ليست موجود باشد. اگر شما واقعاً زيرک باشيد مي توانيد از تمهيدات زير بهره بجوئيد:

```
CREATE TABLE capitals (
        name text,
        population real,
        altitude int, -- (in ft)
        state char(۲)
);

CREATE TABLE non_capitals (
        name text,
        population real,
        altitude int -- (in ft)
);

CREATE VIEW cities AS
        SELECT name, population, altitude FROM capitals
        UNION
        SELECT name, population, altitude FROM non_capitals;
```

اين تدبير به خوبي Query ها عمل مي کند. اما بدي آن زماني است که مي خواهيد چندين رديف را با نامگذاري يک چيز به روز سازيد.

```
CREATE TABLE cities (
        name text,
        population real,
        altitude int -- (in ft)
);


CREATE TABLE capitals (
        state char(۲)
        ) INHERITS (cities);
```

در اين مورد ،يک رديف از جدول capitals همه ستونها(name,population,...) را از والد خود ،cities به ارث مي برد. نوع ستون name از نوع متن مي باشد که يک نوع ذاتي در PostgreSQL براي، کارکترهاي رشته اي با طولهاي متغير محسوب مي شود. جدول capital حاوي يک ستون اضافه با عنوان state است که نمايانگر وضعيت آنها مي باشد. در PostgreSQL يک جدول مي تواند از صفر يا تعداد بيشتري از جدولهاي ديگر به ارث ببرد.

مثال: Query زير نام همه شهر را به همراه وضعيت مراكز استان هايي كه در ارتفاع بيش از ٥٠٠ft واقع هستند را نمايش مي دهد.

```
SELECT name, altitude
        FROM cities
        WHERE altitude > ٥٠٠;
```

كه نتيجه چنين خواهد بود:

```
     name    | altitude
-------------+----------
Las Vegas | ٢١٧٤
Mariposa   | ١٩٥٣
Madison    | ٨٤٥
(٣ rows)
```

به عبارت ديگر Query زير همه شهرهايي را كه مراكز استان نبوده و در موقعيت با ارتفاع بيش از ٥٠٠ ft هستند را جستجو كرده و نمايش مي دهد.

```
SELECT name, altitude
        FROM ONLY cities
        WHERE altitude > ٥٠٠;
```

```
   name     | altitude
--------------+----------
Las Vegas  | ٢١٧٤
Mariposa    | ١٩٥٣
(٢ rows)
```

در اينجا واژه ONLY كه قبل از cities آورده شده گوياي اين است كه اين Query فقط در جدول cities بايد صورت پذيرد و نبايد شامل جدولهاي ورثه اي وسلسله مراتبي پائين تر از cities شود. همچنين بسياري از دستوراتي كه در اينجا راجع به آنها صحبت شد از عبارت ONLY پشتيباني مي كنند.

٦,٣. نتيجه گيري

PostgreSQL قابليت هاي بسيار زيادي را داراست كه دراين خودآموز به آنها پرداخته نشد اين ابزارها با جزئيات بيشتري در كتب PostgreSQL User’s Guide و PostgreSQL Programmer’s Guide مورد بحث و بررسي واقع شده اند.

چنانچه احساس مي كنيد كه نياز به آشنايي و معرفي ابزار بيشتري را داريد، مي توانيد به سايت PostgreSQL جهت ارتباط با لينكهاي متنوع تر در اين زمينه مراجعه بفرمائيد .