

Guia de Consulta Rápida

PostgreSQL

Juliano Niederauer

Segunda Edição

Novatec Editora

Copyright © 2004 da Novatec Editora Ltda.

Todos os direitos reservados e protegidos pela Lei 9.610 de 19/02/1998. É proibida a reprodução desta obra, mesmo parcial, por qualquer processo, sem prévia autorização, por escrito, do autor e da Editora.

Editor: RUBENS PRATES

ISBN: 85-7522-012-8

NOVATEC EDITORA LTDA.

Rua Cons. Moreira de Barros 1084 – Conj. 01

02018-012 São Paulo SP – Brasil

Tel.: +55 11 6959-6529

Fax: +55 11 6950-8869

E-mail: novatec@novateceditora.com.br

Site: www.novateceditora.com.br

Sumário

Conceitos básicos	5
Arquitetura do PostgreSQL	5
PostgreSQL x MySQL	6
Como obter o PostgreSQL	6
Comandos SQL	8
Tipos de dados do PostgreSQL	30
Numéricos	30
Monetários	30
String	30
Data/Hora	31
Booleanos	32
Geométricos	32
Endereço de rede	33
Tipos de Join	33
CROSS JOIN	33
Qualified JOINS	33
Operadores	35
Operadores numéricos	35
Operadores lógicos	35
Operadores binários	35
Operadores relacionais	36
Operador LIKE	36
Operadores geométricos	37
Operadores de endereços de rede	38
Precedência de operadores	38
Funções	39
Funções de agregação	39
Funções de comparação	39
Funções matemáticas	40
Funções de string	42
Funções de formatação	44
Funções de data e hora	46
Funções geométricas	47
Funções de endereços de rede	49
Miscelânea	49
Sintaxe no PostgreSQL	50
Comentários no código SQL	50
Identificadores	50
Constantes	51
Palavras reservadas do PostgreSQL	54
Recursos avançados	55
Utilizando arrays	55
Herança	57
A Técnica MVCC	59
Programas clientes do PostgreSQL	61
createdb	61
createlang	61
createuser	62
dropdb	62
droplang	62
dropuser	63
ecpg	63
pgaccess	64
pgadmin	64
pg_config	64
pg_dump	65
pg_dumpall	66
pg_restore	67
psql	68
pgtclsh	71
pgtksh	71
vacuumdb	71
Aplicações do servidor PostgreSQL	72
initdb	72
initlocation	72
ipcclean	72
pg_controldata	72

pg_ctl	73
pg_resetlog	73
postgres	73
postmaster	74
Arquivos e catálogos do PostgreSQL	76
O arquivo pg_hba.conf	76
Catálogos do sistema	77
Funções PHP	84
Informações adicionais	90
Versão do PostgreSQL utilizada no Guia	90
Links sobre o PostgreSQL	90
Notação utilizada neste Guia	90
Comentários e sugestões	90
Índice remissivo	91

Conceitos básicos

O PostgreSQL é um SGBD (Sistema Gerenciador de Bancos de Dados) relacional e orientado a objetos. É um software de livre distribuição e tem seu código-fonte aberto. Oferece suporte à linguagem SQL (Structured Query Language) de acordo com os padrões SQL92/SQL99, além de outras características modernas. Em termos de recursos, pode ser comparado aos melhores bancos de dados comerciais existentes, sendo inclusive superior em alguns aspectos. O PostgreSQL introduziu conceitos do modelo objeto-relacional que hoje estão disponíveis em alguns bancos de dados comerciais.

O desenvolvimento do PostgreSQL teve início no ano de 1985, no departamento de Ciência da Computação da Universidade da Califórnia, em Berkeley. Hoje é mantido pela Internet por um grupo de desenvolvedores. Os termos “PostgreSQL” e “Postgres” serão usados no decorrer deste guia, ambos para referenciar esse SGBD, visto que o nome do projeto inicial era Postgres.

Arquitetura do PostgreSQL

Antes de começar a utilizar o PostgreSQL, é importante que haja a compreensão de sua arquitetura básica. Quando é aberta uma sessão do Postgres, três processos (programas em execução) trabalham de forma cooperativa:

- um processo daemon (postmaster);
- a aplicação do cliente (por exemplo, o programa psql);
- um ou mais servidores de banco de dados (o próprio processo postgres).

Um único processo postmaster gerencia os bancos de dados existentes em uma máquina. As aplicações do cliente (chamadas de frontend) que desejam acessar determinado banco de dados fazem chamadas a uma biblioteca (chamada LIBPQ). A biblioteca envia a requisição do usuário pela rede para o processo postmaster, que cria um novo processo-servidor (chamado de backend) e conecta o processo-cliente ao servidor criado. A partir daí, os processos-cliente e servidor (frontend e backend) se comunicam sem a intervenção do postmaster. A figura 1 mostra como ocorre uma conexão.

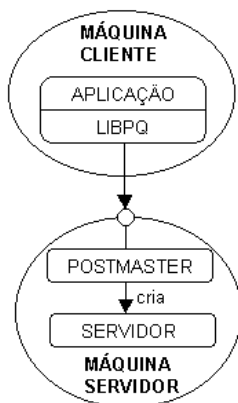


Figura 1 – Estabelecimento de uma conexão.

Portanto o postmaster é o gerenciador de conexões do PostgreSQL. Esse processo estará sempre em execução, esperando por requisições.

PostgreSQL x MySQL

São dois excelentes SGBDs, ambos gratuitos. O MySQL está disponível sob a GPL (licença pública GNU), além de possuir uma licença convencional, para quem não quiser estar limitado aos termos da GPL. Já o PostgreSQL está disponível sob a flexível licença BSD.

O MySQL é mais utilizado no desenvolvimento de aplicações onde a velocidade é importante, enquanto que o PostgreSQL se destaca por ser mais robusto e possuir muito mais recursos. Esses recursos tornam o PostgreSQL um pouco mais qualificado do que o MySQL.

Nas últimas versões do MySQL, os desenvolvedores acrescentaram diversos recursos que já existiam no PostgreSQL, como transações (confirmação ou cancelamento de operações realizadas), triggers (gatilhos), stored procedures (procedimentos armazenados), views (visões), lock de linha (bloqueio em nível de linha) e constraints (cláusulas de integridade).

No entanto, o PostgreSQL continua sendo mais eficiente em vários aspectos. Possui um sofisticado mecanismo de bloqueio (MVCC), suporta tamanhos ilimitados de linhas, bancos de dados e tabelas (até 16TB), aceita vários tipos de subconsultas, possui mais tipos de dados e conta com um bom mecanismo de failsafe (segurança contra falhas, como por exemplo no desligamento repentino do sistema).

Como já foi dito no início desse tópico, a vantagem do MySQL é a velocidade de acesso. Para bases de dados muito grandes, o MySQL faz um acesso mais rápido que o PostgreSQL. Portanto se seu site possuir um banco de dados muito grande, vale a pena analisar a possibilidade de usar o MySQL. Para base de dados menores, não há diferença na velocidade de acesso entre os dois SGBDs.

Como obter o PostgreSQL

O PostgreSQL está disponível para download em qualquer um dos *mirrors* (cópias do servidor principal) apresentados em seu site oficial, no endereço <http://www.postgresql.org>. O download pode ser feito tanto por HTTP como por FTP. Esse site apresenta links para *mirrors* localizados em diversos países, inclusive no Brasil. Para obter sempre as informações mais atualizadas, consulte periodicamente o site oficial do PostgreSQL, no qual você encontrará:

- Última versão do PostgreSQL para download via HTTP ou FTP;
- Informações gerais;
- Últimas notícias sobre o PostgreSQL;
- FAQ (*Frequently Asked Questions*, ou seja, perguntas mais frequentes);
- Documentação para download;
- Documentação interativa;

- Indicação de livros sobre PostgreSQL;
- Listas de discussão;
- Links para artigos e outras publicações sobre o PostgreSQL.

Muitas distribuições do sistema operacional Linux, como por exemplo, o Red Hat ou o Conectiva Linux, já incluem o PostgreSQL entre seus pacotes, e nesse caso você não precisará acessar o site para fazer o download. Basta selecionar, no momento da instalação do Linux, o pacote do PostgreSQL, e o programa de instalação fará sua instalação e configuração automaticamente.

No entanto, se você quiser ter sempre a versão mais atualizada do PostgreSQL, fique atento às atualizações publicadas no site oficial.

Comandos SQL

ABORT

Aborta a transação corrente, descartando todas as atualizações feitas desde o início da transação. É equivalente ao comando ROLLBACK.

ABORT [WORK | TRANSACTION]

ALTER AGGREGATE

Altera a definição de uma função agregada. Nessa versão, a única funcionalidade é a mudança do nome da função.

ALTER AGGREGATE *nome* (*tipo*) **RENAME TO** *novo_nome*

ALTER CONVERSION

Altera a definição de uma conversão. Nessa versão, a única funcionalidade é a mudança do nome da conversão.

ALTER CONVERSION *nome* **RENAME TO** *novo_nome*

ALTER DATABASE

Altera um banco de dados. Pode ser usado de três formas. As duas primeiras alteram o valor de uma variável de configuração, enquanto a terceira é usada para alterar o nome do banco de dados.

ALTER DATABASE *nome* **SET** *parâmetro* {**TO** | **=**} {*valor* | **DEFAULT**}

ALTER DATABASE *nome* **RESET** *parâmetro*

ALTER DATABASE *nome* **RENAME TO** *novo_nome*

ALTER DOMAIN

Altera a definição de um domínio. Pode ser usado para adicionar ou remover cláusulas de integridade (*constraints*), definir um valor padrão ou alterar o proprietário do domínio.

ALTER DOMAIN *nome*
{**SET DEFAULT** *expressão* | **DROP DEFAULT**}

ALTER DOMAIN *nome*
{**SET** | **DROP**} **NOT NULL**

ALTER DOMAIN *nome*
ADD *cláusula_integridade*

ALTER DOMAIN *nome*
DROP CONSTRAINT *cláusula_integridade* [**RESTRICT** | **CASCADE**]

ALTER DOMAIN *nome*
OWNER TO *novo_proprietário*

ALTER FUNCTION

Altera a definição de uma função. Nessa versão, a única funcionalidade é a mudança do nome da função.

ALTER FUNCTION *nome* ([*tipo* [, ...]]) **RENAME TO** *novo_nome*

ALTER GROUP

Adiciona ou remove usuários a um grupo. Os usuários devem existir, pois este comando não cria ou exclui usuários, ele apenas controla quais usuários farão parte do grupo. Também pode ser usado pelo superusuário para renomear um grupo.

ALTER GROUP *nome_grupo* **ADD USER** *usuário* [, ...]

ALTER GROUP *nome_grupo* **DROP USER** *usuário* [, ...]

ALTER GROUP *nome_grupo* **RENAME TO** *novo_nome*

ALTER LANGUAGE

Altera a definição de uma linguagem procedural. Nessa versão, a única funcionalidade é a mudança do nome da linguagem.

ALTER LANGUAGE *nome* **RENAME TO** *novo_nome*

ALTER OPERATOR CLASS

Altera a definição do operador de uma classe. Nessa versão, a única funcionalidade é a mudança do nome do operador.

```
ALTER OPERATOR CLASS nome USING método_índice RENAME TO  
novo_nome
```

ALTER SCHEMA

Altera a definição de um esquema. Para executar esse comando você deve ser proprietário do esquema e possuir o privilégio CREATE no banco de dados.

```
ALTER SCHEMA nome RENAME TO novo_nome
```

ALTER SEQUENCE

Altera a definição de um gerador de seqüência. Os parâmetros não especificados nesse comando continuam com o mesmo valor.

```
ALTER SEQUENCE nome [INCREMENT [BY] incremento]  
[MINVALUE valor_min | NO MINVALUE]  
[MAXVALUE valor_max | NO MAXVALUE]  
[RESTART [WITH] início] [CACHE cache] [[NO] CYCLE]
```

ALTER TABLE

Modifica propriedades de uma tabela. Possui diversas sintaxes, permitindo adicionar ou remover colunas e cláusulas de integridade de uma tabela, definir o modo de armazenamento dos dados, alterar o proprietário, etc. Você deve ser o dono (*owner*) da tabela para poder alterá-la.

```
ALTER TABLE [ONLY] tabela [*]  
  ADD [COLUMN] coluna tipo [cláusula_integridade [...]]  
ALTER TABLE [ONLY] tabela [*]  
  DROP [COLUMN] coluna [RESTRICT | CASCADE]  
ALTER TABLE [ONLY] tabela [*]  
  ALTER [COLUMN] coluna  
  {SET DEFAULT expressão | DROP DEFAULT}  
ALTER TABLE [ONLY] tabela [*]  
  ALTER [COLUMN] coluna { SET | DROP} NOT NULL  
ALTER TABLE [ONLY] tabela [*]  
  ALTER [COLUMN] coluna SET STATISTICS inteiro  
ALTER TABLE [ONLY] tabela [*]  
  ALTER [COLUMN] coluna  
  SET STORAGE {PLAIN | EXTERNAL | EXTENDED | MAIN}  
ALTER TABLE [ONLY] tabela [*]  
  SET WITHOUT OIDS  
ALTER TABLE [ONLY] tabela [*]  
  RENAME [COLUMN] coluna TO nova_coluna  
ALTER TABLE tabela  
  RENAME TO novo_nome_tabela  
ALTER TABLE [ONLY] tabela [*]  
  ADD cláusula_integridade  
ALTER TABLE [ONLY] tabela [*]  
  DROP CONSTRAINT cláusula_integridade  
  [RESTRICT | CASCADE ]  
ALTER TABLE tabela  
  OWNER TO novo_proprietário  
ALTER TABLE tabela  
  CLUSTER ON nome_índice
```

ALTER TRIGGER

Altera a definição de um gatilho (*trigger*). Para executar esse comando, você deve ser o proprietário da tabela para a qual o gatilho é ativado.

```
ALTER TRIGGER nome ON tabela RENAME TO novo_nome
```

ALTER USER

Modifica informações da conta do usuário. Usuários comuns podem alterar apenas sua própria senha, enquanto o superusuário do banco de dados pode alterar as senhas e privilégios de todos os usuários.

ALTER USER *usuário* [[**WITH**] *opção* [...]]

O parâmetro "*opção*" pode ser:

[**ENCRYPTED** | **UNENCRYPTED**] **PASSWORD** '*senha*'
 | **CREATEDB** | **NOCREATEDB**
 | **CREATEUSER** | **NOCREATEUSER**
 | **VALID UNTIL** '*tempo*'

Outras sintaxes possíveis:

ALTER USER *nome* **RENAME TO** *novo_nome*

ALTER USER *nome* **SET** *parâmetro* {**TO** | =} {**value** | **DEFAULT**}

ALTER USER *nome* **RESET** *parâmetro*

Parâmetro	Descrição
<i>usuário</i>	O <i>username</i> do usuário cujos dados serão alterados.
<i>senha</i>	A nova senha a ser usada nessa conta.
ENCRYPTED, UNENCRYPTED	Definem se a senha será armazenada criptografada (<i>encrypted</i>) no arquivo <i>pg_shadow</i> .
CREATEDB, NOCREATEDB	Definem se o usuário está habilitado a criar bancos de dados. Se CREATEDB for especificado, o usuário poderá criar seus próprios bancos de dados.
CREATEUSER, NOCREATEUSER	Definem se o usuário está habilitado a criar novos usuários.
<i>tempo</i>	A data (opcionalmente a hora) que a senha expirará.

ANALYZE

Obtém estatísticas sobre um banco de dados. Os resultados são armazenados na tabela *pg_statistic*, e posteriormente são utilizados pelo sistema no plano de execução das consultas.

ANALYZE [**VERBOSE**] [*tabela* [(*coluna* [, ...])]]

BEGIN

Inicia uma transação. O Postgres, por padrão, realiza o "autocommit", isto é, cada comando é considerado uma transação, que é confirmada de forma implícita. O comando **BEGIN** inicia uma transação em que vários comandos podem ser executados, para posteriormente serem confirmados ou descartados.

BEGIN [**WORK** | **TRANSACTION**]

Parâmetro	Descrição
WORK, TRANSACTION	São palavras opcionais e não produzem nenhum efeito.

CHECKPOINT

Insere um checkpoint no log da transação. O checkpoint define até onde os arquivos de dados devem ser atualizados.

CHECKPOINT

CLOSE

Fecha um cursor declarado pelo comando **DECLARE**, liberando os recursos associados a ele.

CLOSE *nome_cursor*

CLUSTER

Envia um aviso de clustering ao servidor. Assim a tabela é fisicamente reordenada, baseada na informação do índice.

CLUSTER *índice* **ON** *tabela*

CLUSTER *tabela*

CLUSTER

COMMENT

Adiciona um comentário a um objeto. Se um objeto comentado for excluído, os comentários referentes a ele também serão.

COMMENT ON

```
{
  TABLE nome_objeto |
  COLUMN tabela.coluna |
  AGGREGATE nome_agreg (tipo_agreg) |
  CONSTRAINT cláusula_integridade ON tabela |
  DATABASE nome_objeto |
  DOMAIN nome_objeto |
  FUNCTION nome_função (tipo_arg1, tipo_arg2,
  ...) |
  INDEX nome_objeto |
  OPERATOR op (tipo_op_esq, tipo_op_dir) |
  RULE nome_regra ON tabela |
  SCHEMA nome_objeto |
  SEQUENCE nome_objeto |
  TRIGGER nome_trigger ON tabela |
  TYPE nome_objeto |
  VIEW nome_objeto
} IS 'comentário'
```

Parâmetro	Descrição
<i>nome_objeto</i> , <i>tabela</i> , <i>coluna</i> , <i>nome_agreg</i> , <i>nome_função</i> , <i>op</i> , <i>nome_trigger</i>	O nome do objeto a ser comentado.
<i>comentário</i>	O texto do comentário.

COMMIT

Confirma a transação atual, fazendo com que as mudanças feitas possam ser vistas por quem consultar a base de dados.

COMMIT [WORK | TRANSACTION]

Parâmetro	Descrição
WORK , TRANSACTION	São palavras opcionais e não produzem nenhum efeito.

COPY

Copia dados entre arquivos e tabelas. Para ler ou gravar dados em um arquivo é necessário que o usuário possua as devidas permissões de leitura e escrita.

```
COPY tabela [(coluna [,...])] FROM {'arquivo' | STDIN}
  [[WITH] [BINARY] [OIDs]
  [DELIMITER [AS] 'delimitador']
  [NULL [AS] 'string_nula']]
```

```
COPY tabela [(coluna [, ...])] TO {'arquivo' | STDOUT}
  [[WITH] [BINARY] [OIDs]
  [DELIMITER [AS] 'delimitador']
  [NULL [AS] 'string_nula']]
```

Parâmetro	Descrição
<i>tabela</i>	Nome de uma tabela existente.
<i>coluna</i>	Nome da coluna a ser obtida.
<i>arquivo</i>	Caminho absoluto para o arquivo de entrada ou de saída.
BINARY	Força todos os dados a serem armazenados ou lidos no formato binário. As opções DELIMITERS e WITH NULL são irrelevantes para o formato binário.
OIDs	Copia o identificador do objeto (OID) para cada linha.
STDIN	Define que a entrada vem da aplicação cliente.
STDOUT	Define que a saída vai para a aplicação cliente.
<i>delimitador</i>	O caractere que separa os campos em uma linha do arquivo.
<i>string_nula</i>	A string que representa o valor NULL.

CREATE AGGREGATE

Define uma nova função de agregação. Este comando torna possível o acréscimo de novas funcionalidades por parte dos usuários.

```
CREATE AGGREGATE nome
  (BASETYPE = tipo_entrada,
  SFUNC = funcao_estado, STYPE = tipo_estado
  [, FINALFUNC = função_final]
  [, INITCOND = condição_inicial])
```

Parâmetro	Descrição
<i>nome</i>	Nome da função de agregação a ser criada.
<i>tipo_entrada</i>	Tipo do dado de entrada com o qual essa função opera. Se for usado o valor ANY, o valor de entrada não será analisado.
<i>função_estado</i>	O nome da função de transição de estado que será chamada para cada dado de entrada. Recebe o valor do estado corrente e o dado de entrada corrente, e retorna o próximo valor do estado.
<i>tipo_estado</i>	O tipo de dados para o valor do estado agregado.
<i>função_final</i>	O nome da função final chamada para calcular o resultado do agregado após todas as entradas terem sido fornecidas. Se essa função não for especificada, o valor do estado final é usado como resultado do agregado, e o tipo de saída é <i>tipo_estado</i> .
<i>condição_inicial</i>	A configuração inicial para o valor do estado. Deve ser uma constante em uma forma aceita pelo tipo <i>tipo_estado</i> . Se não for especificada, o valor do estado inicia em NULL.

CREATE CAST

Define um novo cast. Um cast define como deve ser executada a conversão entre dois tipos de dados.

```
CREATE CAST (tipo_origem AS tipo_destino)
  WITH FUNCTION nome_função (tipo_arg)
  [AS ASSIGNMENT | AS IMPLICIT]
CREATE CAST (tipo_origem AS tipo_destino)
  WITHOUT FUNCTION [AS ASSIGNMENT | AS IMPLICIT]
```

CREATE CONSTRAINT TRIGGER

Cria um trigger para suportar integridade de dados. É usado nos comandos de criação e alteração de tabelas.

```
CREATE CONSTRAINT TRIGGER nome
  AFTER eventos
  ON relação constraint atributos
  FOR EACH ROW EXECUTE PROCEDURE função ('(' args ')')
```

Parâmetro	Descrição
<i>nome</i>	O nome do trigger constraint.
<i>eventos</i>	As categorias de evento para as quais o trigger deve ser disparado.
<i>relação</i>	Nome da tabela relacionada com o trigger.
<i>constraint</i>	Especificação da constraint.
<i>atributos</i>	Atributos da constraint.
<i>função</i> (<i>args</i>)	Função a ser chamada como parte do processamento do trigger.

CREATE CONVERSION

Define uma nova conversão. As conversões podem ser usadas na função *convert* para especificar um determinado tipo de codificação.

```
CREATE [DEFAULT] CONVERSION nome
  FOR codif_origem TO codif_destino FROM nome_função
```

CREATE DATABASE

Cria um novo banco de dados. Opcionalmente, pode-se especificar um local (caminho) onde os dados serão armazenados. Se for especificado, por exemplo, “/var/lib/pgsql”, os dados serão gravados nesse diretório. O usuário que executar esse comando se tornará o dono (*owner*) do banco de dados.

```
CREATE DATABASE nome
  [WITH [OWNER [=] proprietário]
  [LOCATION = 'caminho']
  [TEMPLATE = modelo ]
  [ENCODING = codificação]]
```

Parâmetro	Descrição
<i>nome</i>	O nome do banco de dados a ser criado.
<i>proprietário</i>	Usuário que será o proprietário do banco de dados.
<i>caminho</i>	A localização do sistema de arquivos onde será armazenado o banco de dados.
<i>modelo</i>	Nome do modelo a partir do qual será criado o banco de dados. Se não for especificado, será usado o modelo-padrão (template1).
<i>codificação</i>	Método de codificação que será usado no banco de dados. Pode ser especificado por um string (como, por exemplo, "SQL_ASCII"), ou por um número inteiro codificado. Se não for especificado, será usado o método-padrão.

CREATE DOMAIN

Define um novo domínio, que consiste em um tipo de dado personalizado criado a partir de um tipo já existente. Com um domínio podemos definir regras e fazer validações de dados sobre determinadas colunas em uma tabela.

```
CREATE DOMAIN nome [AS] tipo_dado
  [DEFAULT expressão][ constraint [...]]
```

O parâmetro *constraint* pode ser:

```
[CONSTRAINT cláusula_integridade]
{NOT NULL | NULL | CHECK (expressão)}
```

CREATE FUNCTION

Define uma nova função. Posteriormente, essa função poderá ser usada por outros comandos, como, por exemplo, o CREATE TYPE.

```
CREATE [OR REPLACE] FUNCTION nome ([tipo_arg [...]])
  RETURNS tipo_ret
  {LANGUAGE nome_ling | IMMUTABLE | STABLE | VOLATILE
  | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT
  | STRICT | [EXTERNAL] SECURITY INVOKER | [EXTERNAL]
  SECURITY DEFINER | AS 'definição'
  | AS 'arq_obj', 'link_simbol'} ...
  [WITH (atributo [...])]
```

Parâmetro	Descrição
<i>nome</i>	O nome da função a ser criada.
<i>tipo_arg</i>	Os tipos de dados dos argumentos da função (se houver). Os tipos de entrada devem ser básicos, complexos ou opacos (argumentos que não são do tipo SQL).
<i>tipo_ret</i>	O tipo do dado que a função retorna. Deve ser especificado como um tipo básico, complexo, setof (indica que a função retorna um conjunto de itens) ou opaco.
<i>atributo</i>	Parte de uma informação opcional usada para otimização.
<i>definição</i>	Uma definição (em string) da função. Pode ser o nome de uma função interna, o caminho para um arquivo-objeto, uma consulta SQL ou texto em linguagem procedural.
<i>arq_obj</i> , <i>link_simbol</i>	Usado para criar links dinamicamente. Às vezes o nome da função na linguagem C não possui o mesmo nome da função SQL. O parâmetro <i>arq_obj</i> é o nome do arquivo contendo o objeto que pode ser carregado dinamicamente, e <i>link_simbol</i> é o link simbólico do objeto, que é o nome da função no código-fonte da linguagem C.
<i>nome_ling</i>	Nome da linguagem usada. Pode ser "sql", "C", "internal" ou "plname" (para linguagem procedural).

CREATE GROUP

Cria um novo grupo. Um grupo é uma maneira de unir logicamente determinados usuários do banco de dados.

```
CREATE GROUP nome
  [WITH [SYSID gid ] [USER usuário [,...]]]
```

Parâmetro	Descrição
<i>nome</i>	O nome do grupo.
<i>gid</i>	A identificação do grupo (group id) no Postgres. Pode ser usada a cláusula SYSID para escolher a identificação do novo grupo. Se não for especificado, o maior número de grupo atribuído é incrementado e usado como padrão.
<i>usuário</i>	Lista de usuários a serem incluídos no grupo. Esses usuários devem existir.

CREATE INDEX

Cria um índice secundário. Os índices são utilizados para melhorar o desempenho do banco de dados, possibilitando que as consultas sejam executadas mais rapidamente.

```
CREATE [UNIQUE] INDEX nome ON tabela
  [USING método] ({coluna | (expressão)} [op_classe]
  [, ...])
  [WHERE predicado ]
```

Parâmetro	Descrição
UNIQUE	Não permite a duplicação de valores na tabela quando o índice é criado e cada vez que novos dados são acrescentados. Inserções ou atualizações de dados que geram entradas duplicadas causarão um erro.
<i>nome</i>	O nome do índice a ser criado.
<i>tabela</i>	O nome da tabela a ser indexada.
<i>método</i>	O nome do método de acesso a ser usado pelo índice. Os métodos disponíveis no Postgres são <i>btree</i> , <i>hash</i> , <i>rtree</i> e <i>gist</i> . O padrão é BTREE.
<i>coluna</i>	Nome de uma coluna da tabela.
<i>expressão</i>	Expressão baseada em uma ou mais colunas da tabela.
<i>op_classe</i>	Um operador de classe associado.
<i>predicado</i>	Cláusula de integridade para definir uma indexação parcial.

CREATE LANGUAGE

Define uma nova linguagem para funções. Posteriormente poderão ser criadas funções e *triggers* com o uso dessa linguagem.

```
CREATE [TRUSTED] [PROCEDURAL] LANGUAGE nome_ling
  HANDLER função_manip [VALIDATOR função_valid ]
```

Parâmetro	Descrição
TRUSTED	Especifica se a função de manipulação da linguagem é segura, isto é, se haverá restrições de acesso para um usuário não privilegiado. Se essa palavra-chave for omitida ao registrar uma linguagem, somente os usuários com privilégio de superusuário do Postgres poderão usar essa linguagem para criar funções.
<i>nome_ling</i>	O nome (sensível a letras maiúsculas e minúsculas) da nova linguagem procedural. O nome dessa linguagem não pode anular uma das linguagens predefinidas do Postgres.
<i>função_manip</i>	Nome de uma função previamente registrada que será chamada para executar os procedimentos PL.
<i>função_valid</i>	Nome de uma função já registrada, que será chamada para validar a criação de uma função nessa linguagem.

CREATE OPERATOR

Define um novo operador, ou seja, permite que o usuário defina um símbolo para representar determinada operação.

```

CREATE OPERATOR nome
  (PROCEDURE = nome_func
  [, LEFTARG = tipo1] [, RIGHTARG = tipo2]
  [, COMMUTATOR = com_op]
  [, NEGATOR = neg_op]
  [, RESTRICT = func_res] [, JOIN = func_join]
  [, HASHES] [, MERGES] [, SORT1 = esq_op_sort]
  [, SORT2 = dir_op_sort]
  [, LTCMP = op_menor_que] [, GTCMP = op_maior_que])

```

Parâmetro	Descrição
<i>nome</i>	Nome do operador que será definido. Pode conter os caracteres + - * / < > = ~ ! @ # % ^ & ' ? \$.
<i>nome_func</i>	A função usada para implementar este operador.
<i>tipo1</i>	O tipo do argumento da esquerda do operador (se existir). Esta opção deve ser omitida para um operador unário que não requer o argumento da esquerda.
<i>tipo2</i>	O tipo do argumento da direita do operador (se existir). Esta opção deve ser omitida para um operador unário que não requer o argumento da direita.
<i>com_op</i>	O comutador deste operador.
<i>neg_op</i>	O negador deste operador.
<i>func_res</i>	A função de restrição para este operador.
<i>func_join</i>	A função de união para este operador.
HASHES	Indica que este operador pode suportar uma união (<i>join</i>) do tipo <i>hash</i> .
MERGES	Indica que este operador pode suportar uma união (<i>join</i>) do tipo <i>merge</i> .
<i>esq_op_sort</i>	Se este operador pode suportar uma junção, este argumento é o operador que classifica o tipo de dados à esquerda.
<i>dir_op_sort</i>	Se este operador pode suportar uma junção, este argumento é o operador que classifica o tipo de dados à direita.
<i>op_menor_que</i>	Operador "menor que" que compara os tipos de dados de entrada do operador, caso ele suporte um <i>merge join</i> .
<i>op_maior_que</i>	Operador "maior que" que compara os tipos de dados de entrada do operador, caso ele suporte um <i>merge join</i> .

CREATE OPERATOR CLASS

Define um novo operador de classe, que permite definir como um determinado tipo de dado pode ser usado com um índice.

```

CREATE OPERATOR CLASS nome [DEFAULT] FOR TYPE tipo_dado
  USING método_índice AS
  {OPERATOR núm_estratégia operador
  [(tipo_op, tipo_op)] [RECHECK]
  | FUNCTION núm_suporte nome_func (tipo_arg [, ...])
  | STORAGE tipo_armaz } [, ...]

```

Parâmetro	Descrição
<i>nome</i>	Nome do operador de classe a ser criado.
<i>tipo_dado</i>	Tipo de dado para o qual o operador será criado.
<i>método_índice</i>	Nome do método de índice a ser usado no operador.
<i>núm_estratégia</i>	Número de estratégia do método de índice para um operador associado com o operador de classe.
<i>operador</i>	Nome de um operador associado com o operador de classe.
<i>tipo_op</i>	Tipo de dado do operando.
RECHECK	Indica se as linhas retornadas usando o índice devem ser checadas novamente, para verificar se elas satisfazem a cláusula de qualificação que envolve esse operador.
<i>núm_suporte</i>	Número do procedimento de suporte do método de índice, para uma função associada com o operador de classe.
<i>nome_func</i>	Nome de uma função que é o procedimento de suporte do método de índice para o operador de classe.
<i>tipo_arg</i>	Tipo de dado dos parâmetros da função.
<i>tipo_armaz</i>	Permite definir o tipo de dado a ser armazenado no índice (que normalmente é mesmo tipo de dado da coluna).