Banco de Dados
– Backup e Particionamento no PostgreSQL –

Prof. Dr. Ives Renê V. Pola
ivesr@utfpr.edu.br

Departamento Acadêmico de Informática – DAINF
UTFPR – Pato Branco DAINF
UTFPR
Pato Branco - PR

Apresenta-se abordgens para Backup e restauração de Bases de Dados.

Outline

Maneiras de realizar um Backup

- Os programas gravam seus dados em discos e outros meios "persistentes", segundo estruturas de dados próprias.
- Mas, erros físicos ou agentes mal-intencionados podem causar perda de dados.
- Os dados devem ser copiados para um destino seguro regularmente.
- Existem algumas maneiras de se realizar um backup, com vantagens e desvantagens:
  - SQL dump
  - Backup no sistema de arquivos
  - Arquivamento contínuo

SQL dump

- A ideia é gerar um arquivo "dump" contendo vários comandos SQL.
- Este dump possui os dados para recriar a base de dados no mesmo estado do momento de geração do dump.
- Para isso, usamos o comando:

SQL DUMP

```
pg_dump NomeBase > Arquivo
```

- É necessário ter permissão de leitura em todas tabelas da base, ou ser root.
- Flags: -h host; -p port; -U user

Vantagens do SQL dump

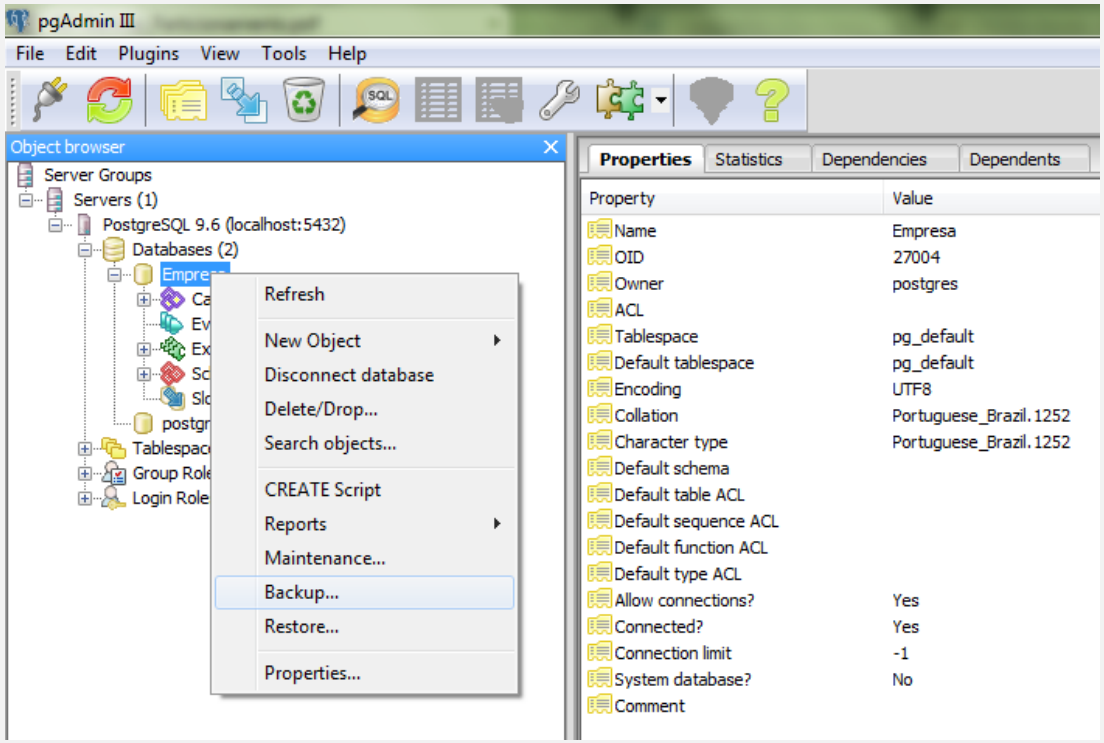
- Pode ser dado como entrada em qualquer versão seguinte do PostgreSQL.
- Também é a melhor alternativa de migrar a base de uma arquitetura para outra (ex: 32-bit para 64-bit).
- O dump representa um snapshot da base de dados no instante que foi emitido o comando.

Restaurando um SQL dump

- Um SQL dump contém todos os comandos SQL para a recriação da base, podendo ser executando usando o psql ou via PGAdmin.
- Importante saber:
  - Deve-se criar a base de dados antes de restaurá-la.
  - Deve-se recriar todos os usuários para os objetos poderem ser associados a eles novamente (owners).
  - Por default, o PG continua a executar o script mesmo havendo erros. Pode-se alterar isso:
    psql --set ON_ERROR_STOP=on nomedb < baseinfile.
  - Mesmo assim a restauração será incompleta, para isso podemos tratá-la como uma transação:
    - psql --single-transaction nomedb < baseinfile

SQL dump via PGAdmin

- Também podemos fazer backup e restauração usando o PGAdmin, de maneira mais fácil.



SQL dump como replicação

- Pode-se utilizar o pg_dump como uma ferramenta de replicação manual.
- A replicação pode ser feita de um servidor para outro, com a base alvo devidamente limpa e preparada.
- Para isso usamos o seguinte comando. Note que é importante executar o ANALYZE em cada base copiada, para o otimizador de planos ter acesso a estatísticas úteis.

DUMPING um server para outro

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

SQL dump ALL

- O comando pg_dump apenas copia a estrutura e os dados de uma database passada como parâmetro.
- Ele não importa os dados dos usuários nem as definições das tablespaces criadas.
  - Isso porque essas são informações do cluster, e não de uma base individual.
- Para realizar um dump completo de um cluster inteiro, usamos o comando pg_dumpall.

DUMPING completo de um cluster

```
pg_dumpall > outfile
```

Restauração de DUMPING completo

```
psql -f infile postgres
```

Backup de Grandes Databases

Comprimir um dump

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

Carregue com

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

Dividir em Arquivos

```
pg_dump dbname | split -b 500M bkp
```

Carregue com

```
cat bkp* | psql dbname
```

Backup Customizado

- Caso o servidor esteja instalado em um sistema operacional com suporte à compressão zlib. Pode-se utilizar o formato de dump customizado pelo PostgreSQL.
- Ele produz um dump de tamanho parecido com o gzip, sem gerar script, e possui vantagens:
  - Cada tabela pode ser descompactada individualmente sem precisar descompactar todo o dump.

Dump comprimido seletivo

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

Carregue com

```
pg_restore -d dbname filename
```

Backup dos Arquivos

- Uma alternativa é realizar uma cópia dos arquivos da base de dados diretamente do sistema de arquivos, caso seu usuário tenha permissão de leitura a estes arquivos.
- Em cada sistema operacional, o PostgreSQL salva os arquivos em caminhos diferentes. Por exemplo em linux:
  - tar -cf backup.tar /usr/local/pgsql/data
- Restrições que podem tornar este método pouco útil:
  - O SGBD não pode estar operando para realizar este backup/recuperação (shut down).
  - Além disso, não se pode copiar apenas algumas (tabelas), porque a restauração também precisa de vários outros arquivos, por exemplo os commit log files (pc_clog/*), que contém os status de todas transações ativas.

Alternativas para Backup dos Arquivos

- Uma segunda opção é utilizar o rsync para realizar o backup.
- rsync: comando em sistemas Linux/Unix/Mac que sincroniza arquivos remotamente e localmente para backup de dados e espelhamento em um ou mais computadores.
  1. Mais rápido que o scp, pois usa um protocolo de atualização que transmite apenas as diferenças entre arquivos.
  2. Utiliza um método de compressão e descompressão durante o envio/recebimento dos arquivos.
  3. Herda a propriedade de criptografia do SSH.

Sincronização de Arquivos

- Deve ser feitas as seguintes etapas, para sincronizar via rsync.

1. Execute rsync enquanto o servidor estiver operando (realizar uma primeira cópia de toda a base na primeira execução).
2. Interrompa o serviço do servidor o tempo suficiente para executar um segundo rsync --checksum
   - necessário o checksum porque a granularidade de modificações do rsync é no mínimo 1 segundo.
   - O segundo rsync será mais rápido, pois somente as diferenças serão copiadas.
   - O resultado será consistente pois o servidor estará inoperante até terminar a sincronização.

Particionamento básico no PostgreSQL

- O particionamento de uma tabela é a divisão suas tuplas em diversas outras tabelas. Útil para tabelas com grandes quantidades de tuplas.
- Normalmente deve-se particionar uma tabela quando seu tamanho excede a quantidade de memória RAM do servidor. Os benefícios são:

1. Desempenho melhorado em consultas, principalmente quando a maioria das tuplas retornadas envolvem uma ou poucas tabelas de partição.
2. Redução do tamanho do índice nas tabelas particionadas. Um índice menor em cada tabela levará a um desempenho melhor nas consultas individuais, e também em inserções nela.
3. Dados pouco acessados (algumas partições) podem ser movidos para dispositivos mais lentos/baratos.

Como realizar o Particionamento no PostgreSQL

- Para criar o particionamento automático, primeiro devemos criar a tabela "master", na qual todas partições vão herdar. Ela não conterá nenhuma tupla. Não há necessidade de constraints ou índices nesta tabela.
- Crie várias tabelas de partições conforme necessário, fazendo-as herdar a tabela master. Estas tabelas podem ter atributos adicionais diferentes dos herdados.
- Adicione restrições de tabela nas tabelas de partição para definir os valores permitidos em cada uma delas. Exemplos:

Exemplos de check constraint das Partições

CHECK (RA $>$ 0 AND RA $<$ 1000)
CHECK (ANO $=$ 2017)

Exemplo para Particionamento no PostgreSQL

- Considere uma tabela que registra dados de um sensor de temperatura em cada cidade coberta, e que registra as vendas de uma rede de sorveteria por datas.
- Deste modo, definimos a tabela master:

Master Table (Tabela a ser otimizada)

```
CREATE TABLE medidas (
    cidade_id int not null,
    logdata date not null,
    tempmin int,
    tempmax int,
    vendas int
);
```

Exemplo para Particionamento no PostgreSQL

- A granularidade do particionamento vai depender do período que a aplicação faça consultas. Por exemplo, vamos supor que os relatórios são feitos em média de 1 a 3 meses, podemos manter as partições em cada mês.
- A sintaxe para a criação das partições mensais com herança é:

Criação das Partições

```
CREATE TABLE medidas_ano2016m01 ( ) INHERITS (medidas);
CREATE TABLE medidas_ano2016m02 ( ) INHERITS (medidas);
CREATE TABLE medidas_ano2016m03 ( ) INHERITS (medidas);
...
CREATE TABLE medidas_ano2017m09 ( ) INHERITS (medidas);
```

Exemplo para Particionamento no PostgreSQL

- O ideal é criar restrições para checar os valores das tuplas antes de inserir nas partições.
- As condições não devem se sobrepor.

Partições com restrições para agilizar consultas

```
CREATE TABLE medidas_ano2016m01 (
   CHECK ( logdata >= DATE '2016-01-01' AND logdata < DATE '2016-02-01' )
) INHERITS (medidas);
CREATE TABLE medidas_ano2016m02 (
   CHECK ( logdata >= DATE '2016-02-01' AND logdata < DATE '2016-03-01' )
) INHERITS (medidas);
...
CREATE TABLE medidas_ano2017m09 (
   CHECK ( logdata >= DATE '2017-09-01' AND logdata < DATE '2017-10-01' )
) INHERITS (medidas);
```

Exemplo para Particionamento no PostgreSQL

- Devemos criar índices nos atributos de particionamento, para agilizar o controle do check nas partições.

Criação de Índices nas Partições

```
CREATE INDEX Idx2016m01log ON medidas_ano2016m01 (logdata);
CREATE INDEX Idx2016m02log ON medidas_ano2016m02 (logdata);
...
CREATE INDEX Idx2017m09log ON medidas_ano2017m09 (logdata);
```

Redirecionando dados nas partições

- Uma alternativa simples é fazer com que os dados sejam inseridos sempre em uma partição mais recente, como por exemplos os dados referentes ao mês atual:

Função para redirecionamento

```
CREATE OR REPLACE FUNCTION medidas_insert()
RETURNS TRIGGER AS $$
BEGIN
  INSERT INTO medidas_ano2017m09 VALUES (NEW.*);
  RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

Trigger para redirecionamento

```
CREATE TRIGGER insert_medidas_trigger
  BEFORE INSERT ON medidas
  FOR EACH ROW EXECUTE PROCEDURE medidas_insert();
```

Redirecionando dados nas partições

- O ideal é ter um método automático para preencher as partições, podendo haver partições previamente criadas para o futuro:

```
CREATE OR REPLACE FUNCTION medidas_insert()
RETURNS TRIGGER AS $$
BEGIN
 IF (NEW.logdata >= DATE '2016-01-01' AND NEW.logdata < DATE '2016-02-01') THEN
        INSERT INTO medidas_ano2016m01 VALUES (NEW.*);
 ELSIF (NEW.logdata >= DATE '2016-02-01' AND NEW.logdata < DATE '2016-03-01') THEN
        INSERT INTO medidas_ano2016m02 VALUES (NEW.*);
 ...
 ELSIF (NEW.logdata >= DATE '2017-09-01' AND NEW.logdata < DATE '2017-10-01') THEN
        INSERT INTO medidas_ano2017m09 VALUES (NEW.*);
 ELSE
        RAISE EXCEPTION 'Erro, data não definida na função de trigger: medidas_insert.';
 END IF;
 RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

Gerenciamento das partições

- Pode ocorrer que dados antigos de uma grande tabela não sejam mais necessários no sistema atual, mas não queremos perdê-los. Podemos guardá-los como backup facilmente usando o particionamento, onde essa tarefa pode ser feita de maneira quase instantânea sem sobrecarregar as outras partições.
- Após ser feito o backup, para remover uma partição basta dropar a tabela de partição:

```
DROP TABLE medidas_ano2000m01;
```

- Desta forma podemos excluir milhares de tuplas rapidamente e o banco exclui a tabela da herança na master table.
- Caso queremos apenas retirá-lo da herança (excluir da master table mas manter a tabela dos dados na partição):

```
ALTER TABLE medidas_ano2000m01 NO INHERIT medidas;
```

Banco de Dados
– Backup e Particionamento no PostgreSQL –

Prof. Dr. Ives Renê V. Pola
ivesr@utfpr.edu.br

Departamento Acadêmico de Informática – DAINF
UTFPR – Pato Branco DAINF
UTFPR
Pato Branco - PR

FIM