

Playbook - Expurgo de dados em PostgreSQL via particionamento

O que é	Arquivamento e/ou expurgo em bancos RDS PostgreSQL e Aurora PostgreSQL com Bucket's S3 utilizando a extension pg_partman
A quem se destina?	Engenheiros, Cientistas, Administradores de Dados e Analistas de Negócio.
Por que / Para que?	Para armazenar dados históricos e expurgar de dados obsoletos automaticamente além de reduzir custos de dados em Cloud.
Onde aplicar?	Em ambientes de dados com ciclo de vida de negócio expirado.
Quando usar?	Quando o custo de armazenamento e/ou performance do banco de dados for insatisfatória e quando for atualizar a versão dos bancos de dados PostgreSQL com base na obsolescência da tecnologia AWS.

- 1. Contexto:
 - 1.1 Cenário atual
 - 1.2 Riscos
 - 1.3 Objetivos
 - 1.4 Proposta técnica
- 2. Como aplicar
 - 2.1 O que precisamos
 - 2.2 Requisitos
 - 2.3 Execução IAM
 - 2.4 Execução em Banco de Dados
 - 2.5 Restaurando os comandos de definição da tabela a partir do S3
- 3. Conclusão

1. Contexto:

Propor uma alternativa para mitigar custos com dados automatizando a historização, o expurgo e o acesso aos dados menos recentemente usados.

1.1 Cenário atual

Com o aumento dos dados armazenados em Cloud, temos ampliado exponencialmente os custos e se faz necessário realizar uma análise mais minuciosa do dado armazenado.

Na Natura grande parte destes dados estão relacionados à informações menor acesso quando olhamos pela ótica dos negócios da empresa e compreendem de informações com ciclo de vida de negócio expirado e que não precisam mais ser mantidos na base de dados. Exemplos comuns são dados de uma venda realizada, onde o pedido é gerado, comissão processada, pagamento realizado pelo cliente, entrega concluída e que depois de certo tempo os dados poderiam ser movidos para backup e excluídos do banco de dados. Outro exemplo sugere um item inserido pelo cliente no carrinho do E-commerce e que não foi convertido em uma venda além de eventuais dados de cadastro com defeito.

1.2 Riscos

Diante deste cenário há um excesso de dados em Cloud gerando-se um custo adicional quando esses sistemas não conseguem atender por exemplo o Apdex. Basicamente quando um ponteiro de banco de dados precisa realizar uma pesquisa scanando uma imensa massa de dados para encontrar determinado registro, acarreta-se lentidão no tempo de resposta da aplicação e consequentemente adição de mais hardware, seja verticalmente (Processador/Memória/Disco) ou horizontalmente (Clusters, Read Réplicas, Data Cache).

Por mais que tenhamos índices performáticos na base é importante mensurar que o percentual de dados recuperados do banco que atendem as operações de negócio da empresa diante da massa de dados existente deve ter como resposta no máximo um reastreamento de 4% do índice existente. Se por exemplo atendemos o negócio com somente 30% dos dados da base e somados a mais 40% de dados armazenados por legislação, temos margem de expurgo de mais 30% o que pode nos gerar uma economia de custos com recursos computacionais.

1.3 Objetivos

Com a tecnologia proposta buscamos facilitar todo o trabalho de análise acima citado onde antecipamos a análise na arquitetura do banco de dados.

1.4 Proposta técnica

No passado eram utilizadas como técnicas expurgo e melhoria de performance de sistemas com a retirada e o armazenamento de dados em fitas/discos externos, backups históricos com retenção jurídica, servidores DRP etc.

Posteriormente este tema foi evoluindo para a criação de partições com base nas datas e no tempo de retenção, mas isso ainda os mantinha consumindo as Storages do Servidor mesmo que em Cloud.

Hoje em dia há uma recomendação da AWS de que dados com ciclo de vida concluídos em Produção, sejam movimentados para outras unidades de armazenamento de menor custo. Isso consequentemente gera melhoria na performance das aplicações em Produção e também melhoria na manipulação apartada dos dados históricos, para geração de indicadores e relatórios.

O Coe de Arquitetura na disciplina de Dados, propõe duas estratégias:

- Unidades de armazenamento de menor custo automatizadas;
- Particionamento externo de objetos de dados;

Hoje temos na AWS diversas unidades de armazenamento S3 com base na frequência de acesso e baixo custo conforme apresentamos abaixo:

- **S3 Bucket** - Armazena Dados/Documentos para repositório de dados ou processamento em lote;
- **S3 Standard/One Zone Infrequent Access** - Para dados acessados com menor frequência;
- **S3 Glacier Flexible Retrieval** - Armazenamento de informações em repositório de baixo custo que pode ser recuperado em milissegundos;
- **S3 Glacier Instant Retrieval** - Armazenamento de informações em repositório de baixo custo que pode ser recuperado em até 12 horas;
- **S3 Glacier Deep Archive** - Armazenamento de informações em repositório de baixo custo que pode ser recuperado em até 48 horas;
- **S3 Intelligent Tiering** - Camada de configuração do S3 para movimentação de dados de acordo com as camadas descritas gerando save de custos de armazenamento conforme característica dos buckets.

A tecnologia com o uso dos Buckets S3 ainda que satisfatório, demandava grande esforço da equipe de Engenheiros de Dados para realizar a seleção dos dados manualmente e orientada pelas equipes de negócio e com massivas reuniões de validação, testes, eventuais falhas de acesso aos dados recentemente migrados pelas área por má comunicação além de outros problemas de negócio e/ou técnicos.

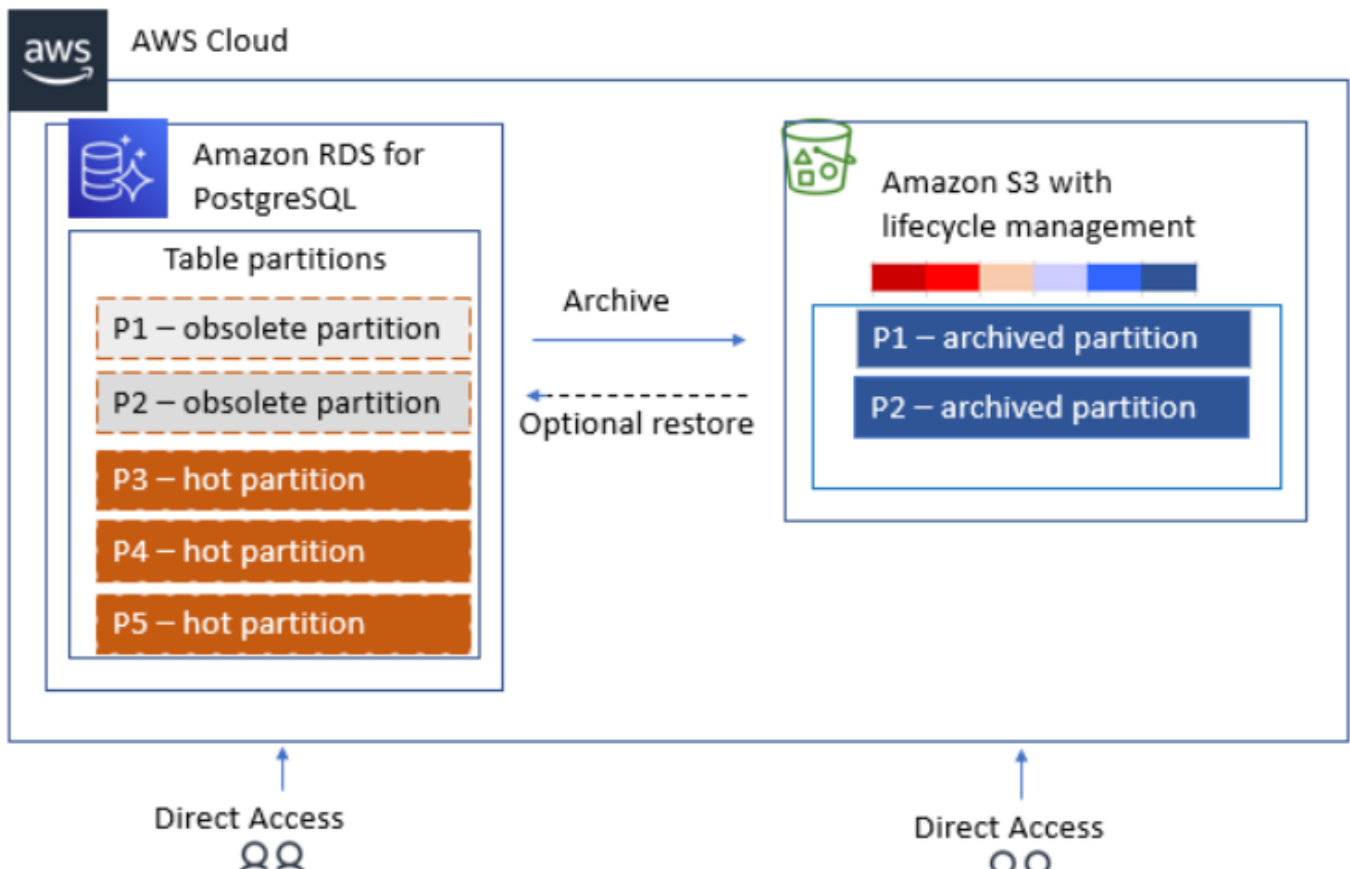
O particionamento de dados no PostgreSQL existe desde a versão 9, porém era pouco utilizado pela comunidade por conta das dificuldades que se apresentava com configurações manuais complexas e também criação de scripts e comandos adicionais para envio de Tuplas para as partições (de herança) a partir de triggers etc.

E recentemente foi disponibilizado pela AWS, o PG_PARTMAN para os bancos de dados Amazon RDS for PostgreSQL 12.5 e Amazon Aurora PostgreSQL 12.6, melhorando a funcionalidade de particionamento de tabelas com automatizações de movimentação de dados para S3.

Unindo essas duas tecnologias é possível realizar a cópia de uma partição específica para um Bucket S3, liberando o espaço na Storage do Banco de Dados e permitindo o acesso aos dados menos frequentemente usados pelo negócio quando necessário.

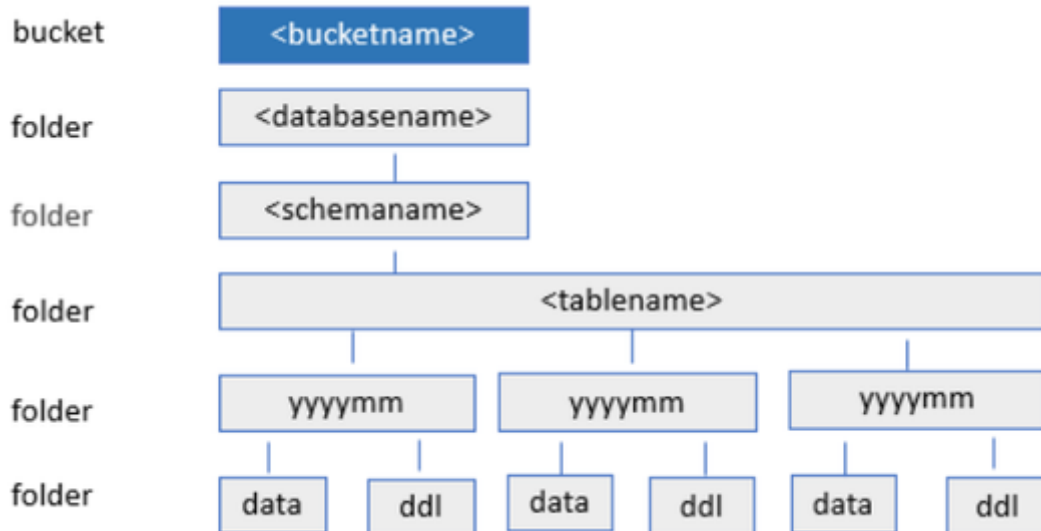
Essa técnica provê menor custo com armazenamento de dados uma vez que o S3 é o menor custo de manipulação de dados da AWS.

Abaixo a figura demonstra o uso dessa técnica de movimentação de dados:





Abaixo um diagrama representando a organização dos dados no Bucket S3:



A aplicação consiste da criação de políticas de expurgo, customização de ambiente de banco de dados para que possam atender os requisitos de negócio.

- AWS Bucket S3;
- Configuração do recurso no Banco de Dados PostgreSQL;
- Agendamento de script de expurgo;
- Ferramenta de acesso ao arquivamento(awscli/athena).

- Escolha da tabela candidata ao particionamento;
- Definição do ciclo de vida dos dados;
- Definição da frequência de execução das rotinas de arquivamento e expurgo;

- ## 2.3 Execução IAM

`aws iam create-policy --policy-name pg-s3-policy --policy-document file:///s3-exp-policy` onde o arquivo `s3-exp-policy` armazenado no diretório corrente tem o seguinte conteúdo:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
```

```

        "s3:GetObject",
        "s3:AbortMultipartUpload",
        "s3:DeleteObject",
        "s3:ListMultipartUploadParts",
        "s3:PutObject",
        "s3:ListBucket"
    ],
    "Resource": [
        "arn:aws:s3:::<bucketname>/<databasename>*",
        "arn:aws:s3:::<bucketname>"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "kms:Decrypt",
        "kms:DescribeKey",
        "kms:GenerateDataKey",
        "kms:Encrypt",
        "kms:ReEncrypt*"
    ],
    "Resource": "<yourKMSkeyarn>"
}
]
}

```

2.3.2 Criação de 2 roles de acesso no IAM para exportação e importação respectivamente

```

aws iam create-role --role-name pg-s3-export-role --assume-role-
policy-document file://s3-exp-assumerole-policy

aws iam create-role --role-name pg-s3-import-role --assume-role-
policy-document file://s3-exp-assumerole-policy

```

onde o arquivo s3-exp-assumerole-policy armazenado no diretório corrente tem o seguinte conteúdo:

```

{
  "Version": "2012-10-17", "Statement": [
    {
      "Effect": "Allow", "Principal": {
        "Service": "rds.amazonaws.com"
      }, "Action": "sts:AssumeRole"
    }
  ]
}

```

2.3.3 Anexe as políticas criadas no item 2.3.1 com os papéis criados no item 2.3.2:

```

aws iam attach-role-policy --policy-arn <yourpolicyarn> --role-name pg-
s3-export-role

```

```
aws iam attach-role-policy --policy-arn <yourpolicyarn> --role-name pg-s3-import-role
```

2.3.4 Adicione os papéis na sua instância a partir do AWS Management Console. Vá na aba Connectivity & Security, preencha com as informações dos papéis de exportação e importação acima criados e selecione Add Role.

Manage IAM roles

Add IAM roles to this cluster

pg-s3-export-role ▼

Feature

s3Export ▼

Add role

Figure 3 – Add S3 export role to Aurora cluster

Add IAM roles to this cluster

pg-s3-import-role ▼

Feature

s3Import ▼

Add role

2.4 Execução em Banco de Dados

2.4.1 Criação de um usuário e atribuição de grants

O usuário deverá ser exclusivo para a execução dos procedimentos de arquivamento com privilégios de manipulação dos dados de todos os schemas envolvidos na solução:

```
CREATE USER <archive_user> WITH PASSWORD '<secret_archive_user>';

GRANT CONNECT ON DATABASE <db_name> TO <archive_user>;

GRANT USAGE ON SCHEMA <schema_name> TO <archive_user>;

GRANT SELECT ON <tablename> TO <archive_user>;

GRANT SELECT ON <table_partition_name> TO <archive_user>;
```

2.4.2 Configurando o particionamento no banco de dados

Vamos criar um schema chamado partman e adicionar a extensão pg_partman:

```
CREATE SCHEMA partman;
```

```
CREATE EXTENSION pg_partman WITH SCHEMA partman;
```

** Para realizar esta atividade você tem que ser um superuser (ex: rds_superuser) no banco de dados ou possuir tais privilégios.*

2.4.3 Crie a extensão AWS_S3 que permitirá que os dados sejam movidos entre o banco de dados e o Bucket S3. Conceda as permissões necessárias para o usuário que vai realizar a o arquivamento:

```
CREATE EXTENSION aws_s3 CASCADE;

GRANT USAGE ON SCHEMA aws_s3 TO <archive_user>;
```

```
GRANT EXECUTE ON ALL FUNCIONTS IN SCHEMA aws_s3 TO <archive_user>;
```

2.4.4 Particionamento da tabela candidata

Um método usual para o particionamento da sua tabela é usar o critério de expurgo com sendo este, um bom caminho para definir como podemos criar partições candidatas a historização e/ou expurgo após finalização do ciclo de vida dos dados e escolher o intervalo de particionamento(dia/mês/ano) para o plano de ação sobre as tabelas.

```
CREATE TABLE dms_sample.ticket_purchase_hist (  
    sporting_event_ticket_id int NOT NULL,  
    purchased_by_id int NOT NULL,  
    transaction_date_time timestamp(0) NOT NULL,  
    transferred_from_id int NULL,  
    purchase_price numeric(8, 2) NOT NULL,  
    CONSTRAINT ticket_purchase_hist_pk PRIMARY KEY  
    (sporting_event_ticket_id, purchased_by_id, transaction_date_time)
```

Neste modelo de exemplo cada venda ou transferência de um tickete, um registro é inserido dentro esta tabela com o timestamp corrente.

asc month ↕	123 count ↕
2021-01	64,946
2021-02	35,422
2021-03	98,110
2021-04	413,276
2021-05	269,706
2021-06	3,383,712
2021-07	210,959
2021-08	316,013
2021-09	24,411
2021-10	451,678
2021-11	439,333
2021-12	34,712

Vamos assumir como política uma retenção de 12 meses e usamos a coluna transaction_date_time com armazenamento em intervalos mensais que podem ser a estratégia de particionamento e processamento da política.

2.4.5 Criação da tabela de partição com intervalo mensal orientada a partir da tabela original acima criada e tenha como chave de partição a coluna transaction_date_time.

```
CREATE TABLE dms_sample.ticket_purchase_hist_temp (  
    sporting_event_ticket_id int NOT NULL,  
    purchased_by_id int NOT NULL,  
    transaction_date_time timestamp(0) NOT NULL,  
    transferred_from_id int NULL,  
    purchase_price numeric(8, 2) NOT NULL)  
PARTITION BY RANGE(transaction_date_time);
```

2.4.6 Renomear a tabela ticket_purchase_hist para old e a tabela ticket_purchase_hist_temp para a tabela principal sem o final "temp):

```
ALTER TABLE dms_sample.ticket_purchase_hist RENAME TO
ticket_purchase_hist_old;

ALTER TABLE dms_sample.ticket_purchase_hist_temp RENAME TO
ticket_purchase_hist;
```

2.4.7 Registrar e ativar a tabela pai com o pg_partman para que inicie a criação das partições.

```
SELECT partman.create_parent( p_parent_table => 'dms_sample.
ticket_purchase_hist', p_control => 'transaction_date_time', p_type =>
'native', p_interval=> 'monthly', p_premake => 7,p_start_partition =>
'2021-01-01' );
```

** Para realizar esta atividade você tem que ser um superuser (ex: rds_superuser) no banco de dados ou possuir tais privilégios.*

2.4.8 Dropando partição automática para valores não contemplados

Durante a criação o PostgreSQL cria também uma partição padrão para que sejam armazenados dados que não contemplem nenhuma das partições. Isso geralmente acontece quando temos falha na aplicação no momento do insert dos dados. O ideal é remover esta partição para que neste caso ocorra uma falha na aplicação para inserir um registro.

```
DROP TABLE dms_sample.ticket_purchase_hist_default;
```

2.4.9 Carga de Dados

Insira dados na "nova tabela principal" a partir da tabela histórica renomeada para old:

```
INSERT INTO dms_sample.ticket_purchase_hist SELECT * FROM dms_sample.
ticket_purchase_hist_old;
```

Valide o resultado desta ação na tabela:

```
Select count(1) from dmbs_sambple.ticket_purchase_hist_p2021_01;

Output: 64946
```

2.4.10 Adicione chaves e índices à tabela particionada:

```
ALTER TABLE dms_sample.ticket_purchase_hist ADD CONSTRAINT
ticket_purchase_hist_pk1 PRIMARY KEY (sporting_event_ticket_id,
purchased_by_id, transaction_date_time);
```

2.4.11 Configurando a política com pg_partman

Com privilégio de superusuário configure a política com o pg_partman. Vamos usar o exemplo definido para 12 meses:

```
UPDATE partman.part_config SET infinite_time_partitions = true,
retention = '12 months', retention_keep_table=true WHERE parent_table =
'dms_sample.ticket_purchase_hist';
```

Como superusuário e a partir de linha de comando do PL/SQL execute periodicamente (mensalmente para este exemplo) a query para identificar em qual mês você está trabalhando. O ideal é automatizar esta execução para todo o primeiro dia do período de cada mês:

```
CALL partman.run_maintenance_proc();
```

i **O procedimento cria as partições futuras e separa as obsoletas seguindo uma configuração de retenção.

2.4.12 Movendo a partição obsoleta para o AWS S3 e eliminando a partição armazenada dentro do banco de dados

Vamos compreender uma partição de determinada tabela como sendo uma “tabela filha” da tabela particionada pelo pg_partman. Com o tempo esta partição ficará obsoleta e conforme a configuração da política de retenção será automaticamente desativada da tabela pela procedure partman.run_maintenance_proc. Após isso o responsável tem que identificar esta partição e enviá-la para o arquivamento no Bucket S3.

2.4.13 A partir do comando abaixo execute a query para identificar quais partições foram desativadas da tabela e necessitam ser migradas para o S3 Bucket.

```
select relname, n.nspname
from pg_class
      join pg_namespace n on n.oid = relnamespace
where relkind = 'r' and relispartition = 'f'
and relname like 'ticket_purchase_hist_p%' and n.nspname = 'dms_sample';
```

output:

```
relname          nspname
-----
ticket_purchase_hist_p2021_01  dms_sample
```

2.4.14 Exporte os dados da partição desativada para o S3 bucket. Neste passo você executa um comando plsql para arquivar os dados para o S3 utilizando duas funções PostgreSQL:

- create_s3_uri que define o destino do bucket S3 e no exemplo vamos orientar para o nome de 'dbarchive-test', no caminho 'testdb/dms_sample/ticket_purchase_hist/202101/data' e a região será "us-east-1";
- query_export_to_s3 fará a extração dos dados da partição;

Veja o exemplo:


```
SELECT * FROM aws_s3.query_export_to_s3(
  'SELECT * FROM dms_sample.ticket_purchase_hist_p2021_01',
  aws_commons.create_s3_uri('dbarchive-test','testdb/dms_sample
/ticket_purchase_hist/202101/data','us-east-1'));
```

2.4.15 Valide comparando a quantidade de registros no bucket S3 e na tabela do banco de dados. A partir do AWSCLI execute a query para contar quantos registros na tabela dentro do bucket S3. No exemplo abaixo o número de registros vai ser armazenado em um arquivo de saída chamado recordcount.csv:

```
aws s3api select-object-content \
  --bucket dbarchive-test \
  --key testdb/dms_sample/ticket_purchase_hist/202101/data \
  --expression "select count(*) from s3object" \
  --expression-type 'SQL' \
  --input-serialization '{"CSV": {}, "CompressionType": "NONE"}' \
  --output-serialization '{"CSV": {}}' "recordcount.csv"
```

2.4.16 Recomenda-se gerar também a estrutura (DDL) da tabela a qual será arquivada no Bucket S3. Assim em caso de restauração será possível compor a estrutura da tabela semelhante ao dado arquivado no passado sem o risco de seu funcionamento.

Primeiro vamos gerar o script de criação da tabela em um arquivo SQL:

```
pg_dump -h <dbhost> -d <dbname> -U <dbuser> -s -t dms_sample.
ticket_purchase_hist_p2021_01 > ticket_purchase_hist_p2021_01.sql
```

2.4.17 Armazenamos o arquivo criado com o DDL da tabela para o Bucket S3 utilizando o AWSCLI:

```
aws s3 cp ticket_purchase_hist_p2021_01.sql s3://dbarchive-test/testdb
/dms_sample/ticket_purchase_hist/202101/ddl
/ticket_purchase_hist_p2021_01.sql
```

2.4.18 Exclua a tabela do banco de dados arquivada no bucket S3 com o comando abaixo:

```
Drop table dms_sample.ticket_purchase_hist_p2021_01 cascade;
```

Opcionalmente você pode também recuperar a tabela arquivada no bucket S3 novamente para o Banco de Dados usando as ferramentas de select em bucket S3 usando o S3 select ou também o Amazon S3 Glacier Select ou o Amazon Athena.

2.5 Restaurando os comandos de definição da tabela a partir do S3

2.5.1 gere o script sql a partir do AWSCLI

```
aws s3 cp s3://dbarchive-test/testdb/dms_sample/ticket_purchase_hist/202101/ddl/ticket_purchase_hist_p2021_01.sql ticket_purchase_hist_p2021_01.sql
```

2.5.1 A partir do comando psql, crie a tabela usando o arquivo recuperado:


```
psql "dbname=<dbname> host=<dbhost> user=<dbuser> port=5432" -f ticket_purchase_hist_p2021_01.sql
```

2.5.2 Restaure os dados arquivados a partir das functions aws_s3.table_import_from_s3 e aws_commons.create_s3_uri

- aws_commons.create_s3_uri define a fonte de informação do bucket S3 onde no exemplo abaixo o bucket chama-se 'dbarchive-test', no caminho 'testdb/dms_sample/ticket_purchase_hist/202101/data', and AWS regions as 'us-east-1'
- aws_s3.table_import_from_s3 fará a importação dos dados a partir do bucket S3 para dentro da tabela criada anteriormente. Vamos usar como tabela destino a dms_sample.ticket_purchase_hist_p2021_01 e orientamos o formato "text".

```
SELECT *
FROM aws_s3.table_import_from_s3('dms_sample.
ticket_purchase_hist_p2021_01','',
'(format text)',aws_commons.create_s3_uri
('dbarchive-test','testdb/dms_sample/ticket_purchase_hist/202101
/data','us-east-1'))
```

```
Output: 64946 rows imported into relation "dms_sample.
ticket_purchase_hist_p2021_01 "
from file testdb/dms_sample/ticket_purchase_hist/202101/data of 3003108
bytes
```

 *Você pode automatizar todos os comandos acima utilizando ferramentas de execução automática da Cloud como por exemplo o Lambda e depois de tudo isso, ainda utilizar-se dos recursos do S3 Intelligent Tiering os quais também pode movimentar dados entre as diversas configurações de Bucket's disponíveis na AWS.*

3. Conclusão

Inicialmente a proposta será substancial para o negócio, que hoje gera muito de armazenamento de dados. Com o passar do tempo, a mudança de estratégia na arquitetura para composição da camada de dados, trará benefícios para a empresa no controle dos dados.

A recomendação de uso é para quando a AWS orientar a troca de uma versão do PostgreSQL por término do suporte, que já se atualize este banco para versões 12.5 e ativem o pg_partman por período em tabelas de negócio para eliminar ao máximo o trabalho manual, de análise e expurgo de dados.

A simples mudança na forma de compor um servidor de banco de dados com a visão de arquitetura de dados voltada para o menor consumo de recursos pode ser substancial para a empresa.