



Performance e Otimização

Bootcamp Analista de Banco de Dados

Gustavo Aguilar

2021

Performance e Otimização

Gustavo Aguilar

© Copyright do Instituto de Gestão e Tecnologia da Informação.

Todos os direitos reservados.

Sumário

Capítulo 1. Bancos de Dados Distribuídos	5
1.1. Fundamentos de Sistemas Distribuídos.....	5
1.2. Introdução à Bancos de Dados Distribuídos	13
1.3. Técnicas de Distribuição de Dados.....	17
1.4. Tipos de Replicação de Dados	19
1.5. Tipos de Particionamento de Dados	21
1.6. Teorema de CAP e Propriedades BASE.....	23
Capítulo 2. Distribuição de Dados no SQL Server.....	27
2.1. Partitioned View	27
2.2. Log Shipping e SQL Server Replication.....	27
2.3. Database Mirroring e AlwaysOn Availability Groups	29
Capítulo 3. Banco de Dados Distribuído como Serviço	32
3.1. Azure SQL Database Managed Instance.....	32
3.2. Azure SQL Database	33
3.3. Azure CosmosDB	34
3.4. Bancos de Dados Open Source no Azure	35
Capítulo 4. Distribuição de Dados no MongoDB	36
4.1. Replica Set.....	36
4.2. MongoDB Sharding.....	37
4.3. MongoDB Atlas	39
Capítulo 5. Aspectos Gerais de Performance em Banco de Dados	40
5.1. Aspectos Gerais de Infraestrutura	40
5.2. Aspectos Gerais dos SGBDs	44
5.3. Índices.....	46

5.4 Estatísticas e Plano de Execução de Query	48
Capítulo 6. Otimização de Query no SQL Server	51
6.1. Tipos de Índices	51
6.2. Análise do Plano de Execução de Query	55
Capítulo 7. Otimização de Query no MongoDB	57
7.1. Tipos de Índices	57
7.2. Plano de Execução de Query	62
Capítulo 8. Operações Massivas de Dados	65
8.1. Extração e Carga Massiva de Dados no SQL Server	65
8.2. Extração e Carga Massiva de Dados no MongoDB	73
8.3. Expurgo Massivo de Dados no SQL Server	75
8.4. Expurgo Massivo de Dados no MongoDB	80
Referências	83

Capítulo 1. Bancos de Dados Distribuídos

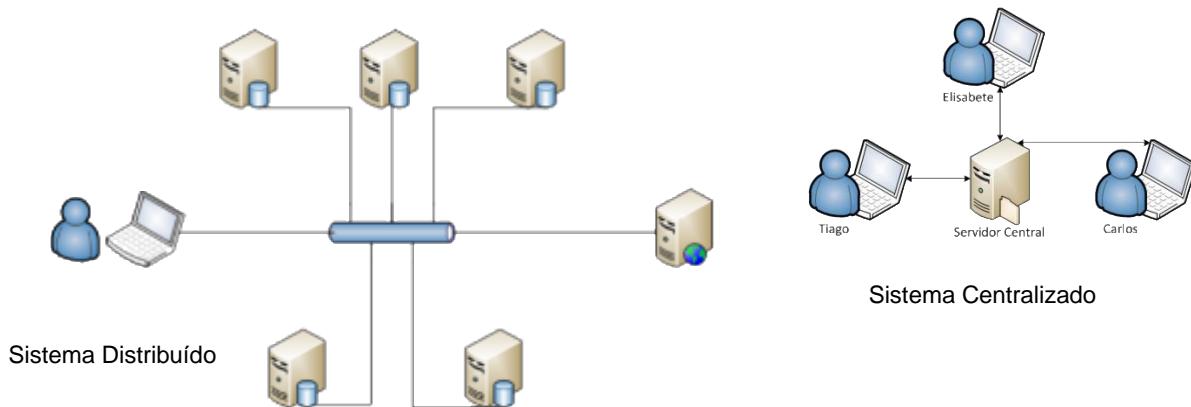
Quando falamos de performance e otimização de banco de dados, não podemos deixar de falar de banco de dados distribuídos, pois suas características de escalabilidade, replicação e particionamento de dados contribuem diretamente para maximizar a performance em um sistema de banco de dados.

Antes de falarmos de banco de dados distribuídos, vamos entender primeiramente o que é um sistema distribuído.

1.1. Fundamentos de Sistemas Distribuídos

Com a crescente onda tecnológica no mercado e a redução dos custos de hardwares e equipamentos tecnológicos, bem como a crescente evolução das redes e da infraestrutura disponíveis para estes tipos de serviços, vemos um cenário cada vez mais favorável à utilização de sistemas distribuídos e soluções que usam as estruturas focadas em várias estações interligadas por uma rede.

Figura 1 – Sistema Distribuído x Sistema Centralizado.



Dessa forma, podemos perceber que, na linha do tempo da evolução macro da computação, começamos a ter a possibilidade de criarmos sistemas distribuídos com a invenção da rede de computadores.

Figura 2 – Macroevolução da computação distribuída.

Sistemas distribuídos, assim como qualquer sistema, devem implementar e seguir algumas definições que podem ser de cunho funcional ou não funcional. Requisitos funcionais são extremamente importantes, principalmente na visão arquitetural, e devem ser entendidos e compreendidos pela solução. As principais características de sistemas distribuídos são:

- **CONECTIVIDADE:** característica voltada para a capacidade do sistema se manter conectado a recursos disponíveis.
- **DISPONIBILIDADE:** em sistemas distribuídos, o grande desafio é reduzir o downtime, que tem como consequência a maximização do uptime, ou seja, o tempo que o sistema permanece disponível para uso.

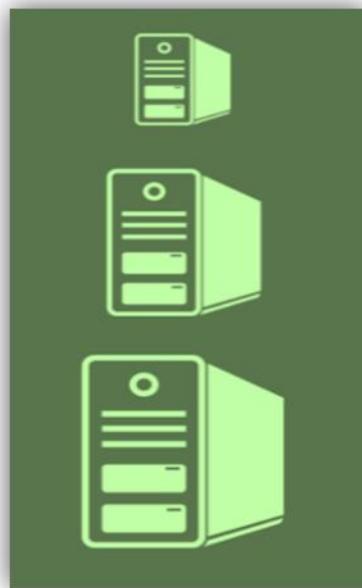
Figura 3 – Disponibilidade de sistemas.

Disponibilidade (%)	Downtime/ano	Downtime/mês
95%	18 dias 6:00:00	1 dia 12:00:00
96%	14 dias 14:24:00	1 dia 4:48:00
97%	10 dias 22:48:00	0 dias 21:36:00
98%	7 dias 7:12:00	0 dias 14:24:00
99%	3 dias 15:36:00	0 dias 7:12:00
99,90%	0 dias 8:45:35.99	0 dias 0:43:11.99
99,99%	0 dias 0:52:33.60	0 dias 0:04:19.20
99,999%	0 dias 0:05:15.36	0 dias 0:00:25.92

- **DESEMPENHO:** tempo no qual um sistema consegue executar uma tarefa. Em sistemas distribuídos, será o tempo total que o ecossistema inteiro leva para processar uma operação.
- **ESCALABILIDADE:** característica que permite aumentar o poder de processamento de um sistema. Em sistemas distribuídos, indica o aumento da quantidade de hardware ou recurso (CPU, RAM, disco, rede, etc.) necessário que possibilite aumentar o poder de processamento, sem interferir na aplicação e nos usuários. Difere-se da **elasticidade**, que é capacidade de um ambiente escalar (aumentar) e depois “desescalar” (reduzir), mas estas duas características estão intrinsecamente ligadas, pois a capacidade de escalar permite que um sistema tenha a possibilidade de ter elasticidade.

Podemos ter a **elasticidade vertical**, que é a capacidade de aumentar recursos aos integrantes de um sistema distribuído, sem alterar a quantidade de integrantes.

Figura 4 – Escalabilidade vertical.



Podemos ter também a **escalabilidade horizontal**, que é a capacidade de adicionar mais componentes (integrantes) à um sistema distribuído, de forma transparente para o usuário.

Figura 5 – Escalabilidade horizontal.

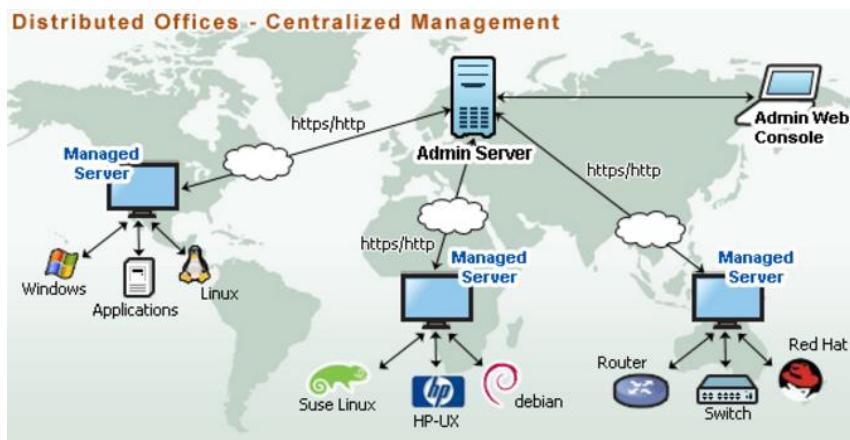


- **TRANSPARÊNCIA:** com essa característica, o objetivo é ocultar do usuário final ou do desenvolvedor, a separação dos componentes em um sistema distribuído, de modo que o sistema seja percebido como um todo, em vez de como uma coleção de componentes independentes e isolados.

Podemos ter as seguintes transparências:

- **De Acesso:** esconde os detalhes dos protocolos e configurações de rede que controlam a comunicação entre as diversas máquinas, ocultando diferenças entre arquiteturas de máquinas.

Figura 6 – Transparência de Acesso.



- **De Localização:** esconde a localização dos recursos no sistema distribuído, de forma que os usuários não são capazes de dizer a localização física do recurso.

Figura 7 – Transparência de localização.



- **De Replicação:** esconde o fato de que múltiplas cópias do mesmo recurso podem estar disponíveis no sistema, permitindo que várias instâncias de recursos sejam usadas para aumentar a confiabilidade e o desempenho. Deve mascarar o conhecimento das réplicas por parte dos usuários, e implica na transparência de localização.

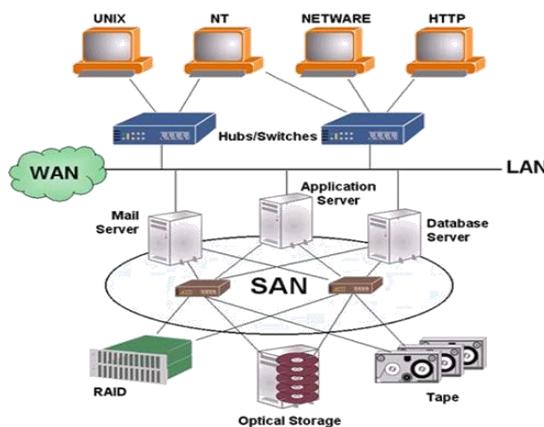
Figura 8 – Transparência de replicação.



Fonte: MongoDB (2018).

- **De Falha:** possibilita que falhas no sistema não sejam percebidas pelo usuário, provendo alta disponibilidade em um sistema distribuído.

Figura 9 – Sist. distrib. SAN com transparência de falha.

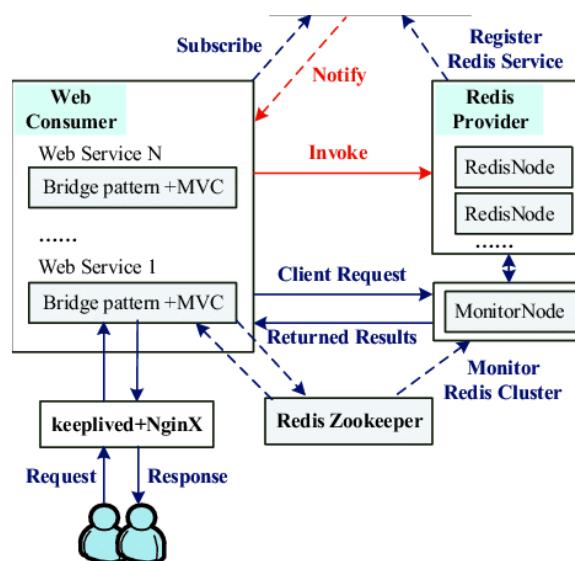


- **De Persistência:** esconde a informação sobre a forma de armazenamento (memória, disco, etc.) dos recursos em um sistema distribuído.
- **De Concorrência:** qualquer recurso compartilhado em um sistema distribuído deve ter garantido o correto funcionamento em um ambiente concorrente com acessos simultâneos.
- **De Migração e Realocação:** escondem a movimentação de recursos em um sistema distribuído, mascarando a movimentação de um objeto de um ponto a outro no sistema. Recursos podem migrar de uma localidade para outra, por questões de desempenho, segurança, etc., devendo isso ser feito de forma automática pelo sistema. Deve manter o nome do recurso que o usuário conhece e garantir a continuidade de comunicação.

Continuando com as características de um sistema distribuído, ainda temos:

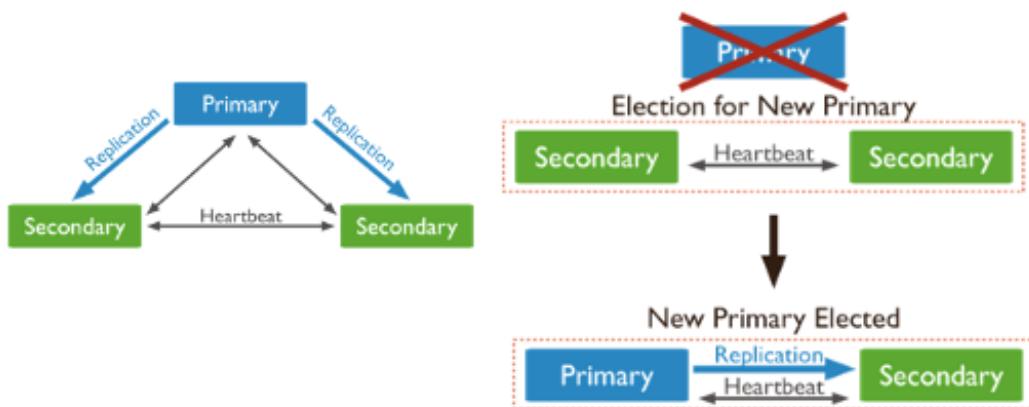
- **SEGURANÇA:** garantir acesso a seus recursos apenas a usuários/sistemas devidamente identificados e autorizados, em qualquer ponto do sistema distribuído.

Figura 10 – Sist. Distrib. com transp. de segurança.



- **INTEROPERABILIDADE:** demonstra a capacidade que o sistema possui de interagir e comunicar com outros sistemas/mecanismos de forma transparente e simples. Abrange a capacidade que o ecossistema tem de integrar tecnologias heterogêneas e apresentá-las de forma homogênea para utilização.
- **TOLERÂNCIA A FALHAS:** propriedade que permite que sistemas continuem a operar adequadamente, mesmo após falha(s) em algum de seus componentes.

Figura 11 – Sist. Distrib. com tolerância a falhas.



Fonte: MongoDB (2018).

- **Tolerância a Falhas x Alta Disponibilidade:** pode-se ter um sistema totalmente tolerante a falhas (que se recupera em situações de falhas), mas com baixa disponibilidade (tempo em que ele se recupera gera indisponibilidade para o usuário).
- **Tolerância a Falhas x Transparência de Falha:** na transparência de falha, o usuário não percebe a falha.
- **USABILIDADE:** é a capacidade que um sistema dispõe de tornar a sua utilização simples e objetiva. Em sistemas distribuídos, essa definição também contempla a transparência de acesso ao usuário, abstraindo-o do conhecimento do ecossistema.

- **HETEROGENEIDADE:** sistemas distribuídos normalmente são construídos a partir de uma variedade de redes, sistemas operacionais, hardwares e linguagens de programação distintas. Para prover essa heterogeneidade de forma transparente para o usuário, utilizam protocolos de comunicação e middlewares para mascarar essa diferença, tornando o sistema homogêneo na visão do usuário.

1.2. Introdução à Bancos de Dados Distribuídos

Os conceitos de banco de dados distribuídos foram definidos por C.J. Date e Dr. E.F. Codd em 1987 (que também foi o autor da teoria relacional de banco de dados). Foram propostas doze regras que um SGBDD (Sistema gerenciador de banco de dados distribuídos) deve seguir para que fosse definido como tal.

- Autonomia Local:** cada nó deve prover seus próprios mecanismos de segurança, bloqueio, acesso, integridade e recuperação após uma falha.

Na prática, se resume a cada nó ser uma instância do SGBDD, cada uma mantendo o controle sobre seus próprios bancos de dados.

- Descentralização:** não dependência de um nó central, o que acarretaria em um único ponto de falha.

Se um nó ficar indisponível, a continuidade das operações no ambiente deve ser garantida.

Figura 12 – Descentralização em SGBDD.



- Operação Contínua:** operações de replicação e distribuição dos dados, backup e recuperação devem ser suportadas de forma on-line.

Rápidas o bastante para não afetarem o funcionamento do sistema.

- Independência de Hardware:** as operações em bancos de dados distribuídos não devem estar condicionadas à recursos físicos ou lógicos de hardware.

- Independência de Sistema Operacional:** as operações em bancos de dados distribuídos não devem estar condicionadas à recursos de um sistema operacional específico.

- Independência de Rede:** deve executar independentemente do protocolo de comunicação e da topologia de rede usada para interligar os nós.

- Independência de SGBD:** um SGBDD ideal deve possuir capacidade(s) para se comunicar com outros SGBDs → interoperabilidade.

- Processamento Distribuído de Consultas:** experiência do usuário, em termos de tempo de resposta, deve ser independente do local de disparo.

- Gerenciamento de Transações Distribuídas.**

- Transparência / Independência de Localização:** os usuários do sistema não necessitam saber a localização do armazenamento dos dados.

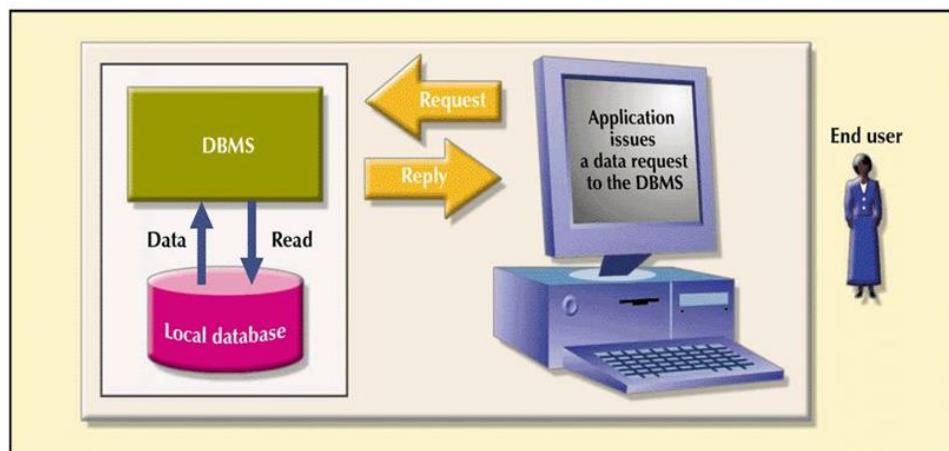
- Independência de Replicação:** dados podem estar replicados em vários nós da rede, sem interferir na visão única do usuário.

As réplicas de dados devem ser mantidas sincronizadas automaticamente pelo SGBDD.

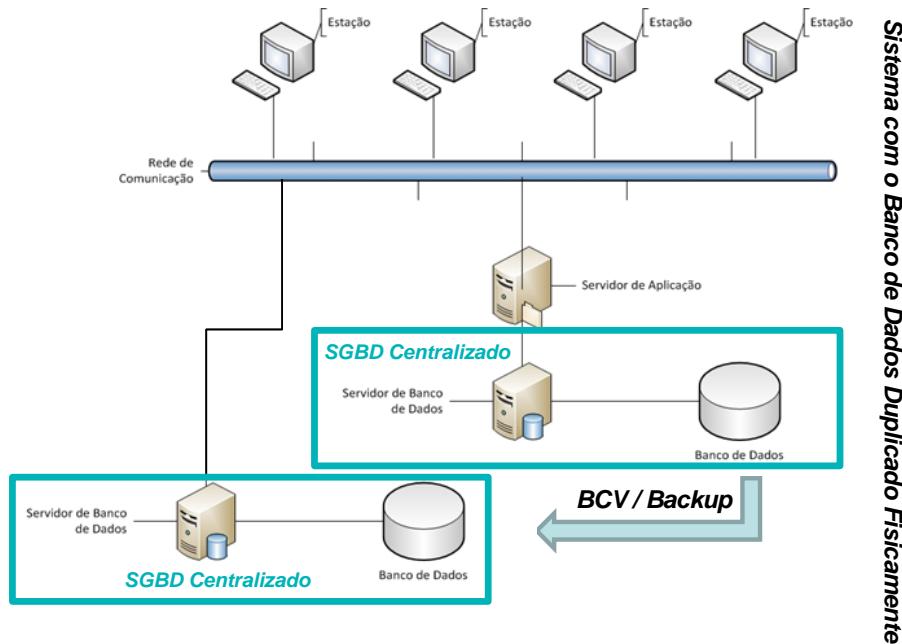
- Independência de Fragmentação:** banco de dados ou tabelas podem estar divididos em fragmentos, localizados fisicamente em diferentes nós, de forma transparente para o usuário.

Diferentemente dessas características, um SGBD centralizado possui um front-end único, com gerenciamento de dados centralizado e processamento centralizado de consultas e transações, o que acarreta em ponto único de falha. Pode ser formado por componentes físicos únicos ou compartilhados (Cluster), mas só permite a escalabilidade vertical.

Figura 13 – SGBD Centralizado.

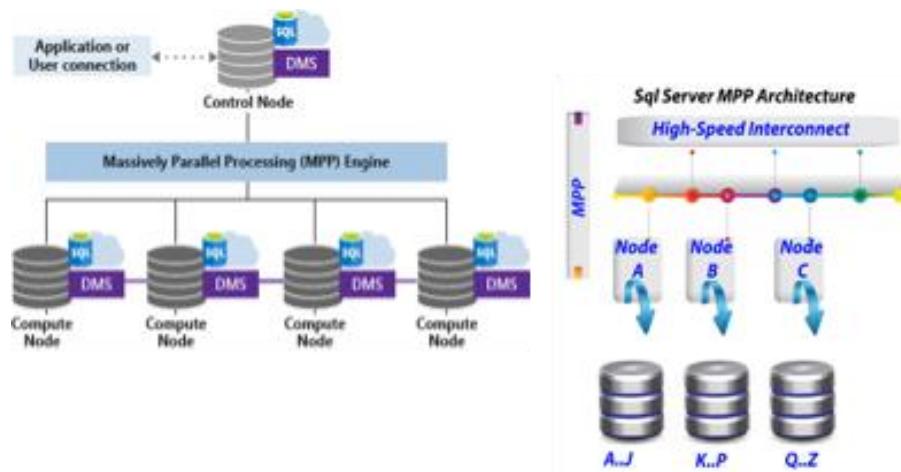


Dessa forma, temos que ter em mente que um sistema gerenciador de banco de dados que não tem a capacidade nativa de distribuir ou replicar o banco de dados, não pode ser considerado um sistema gerenciador de banco de dados distribuído, apesar do banco de dados do sistema poder estar duplicado fisicamente.

Figura 14 – SGBDs Centralizados com Banco Duplicado.

Fonte: Gustavo A. (2018).

Para além disso, temos que ter em mente que banco de dados paralelo (MPP), não é sinônimo de banco de dados distribuído. Esses possuem uma engine única, na maioria dos casos com um hardware único (nodes são lógicos) e com uma comunicação feita entre os nós através de um barramento proprietário.

Figura 15 – SGBD Paralelo (MPP).

Fonte: Microsoft (2018).

Na arquitetura de banco de dados distribuídos, ou a forma como os bancos de dados estão distribuídos na rede, destaca-se o grau de homogeneidade. Definimos como **homogêneos** quando num projeto distribuído temos todos os SGBDDs idênticos, do mesmo fabricante, versão e edição, como por exemplo, todos os bancos Oracle ou todos SQL Server. Neste tipo de arquitetura, a administração do ambiente e as integrações são relativamente simples, visto que os bancos irão utilizar protocolos proprietários e conhecidos. Os bancos cooperam entre si no processamento das transações distribuídas.

Já nos sistemas **heterogêneos**, os SGBDDs são distintos, ou seja, podemos ter um banco de Oracle transacionando junto com um banco de dados MySql. Isto torna o ambiente e a administração complexos. Protocolos de comunicação padrões, como ODBC, por exemplo, podem ser utilizados nestas integrações.

1.3. Técnicas de Distribuição de Dados

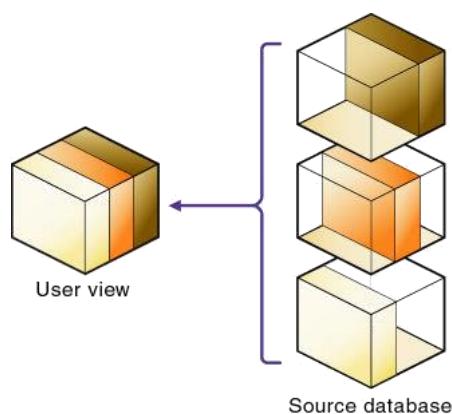
Para distribuir os dados, os SGBDD podem-se valer de duas técnicas:

- ✓ **Replicação de Dados.**
- ✓ **Particionamento de Dados.**

Na replicação de dados, as cópias (réplicas) são armazenadas em mais de um local, não se tratando de backups tradicionais de banco de dados (backupdatabase/dump/export). Essa técnica é muito utilizada para ambientes de disaster recovery (DR) e separação de workloads distintos (escrita/leitura). O SGBDD é quem gerencia a replicação e deve garantir a disponibilidade, consistência, integridade e confiabilidade dos dados. Além disso, ele deve fornecer transparência da replicação e da localização dos dados, não sendo considerado um SGBDD ou um banco de dados distribuído, os bancos de dados que são replicados por softwares de terceiros.

Figura 16 – Replicação de dados.

Já o particionamento de dados, conhecido também como **fragmentação (sharding)**, é uma técnica de distribuição de dados muito utilizada em ambientes que requerem escalabilidade horizontal, onde pode-se usar um hash ou uma faixa de valor para a distribuição dos dados. Da mesma forma que na replicação, o SGBDD é quem gerencia a distribuição e deve garantir a disponibilidade, consistência, integridade e confiabilidade dos dados, além da transparência da distribuição e da localização dos dados, e dispor de mecanismos para resolução de conflitos.

Figura 17 – Particionamento de dados.

1.4. Tipos de Replicação de Dados

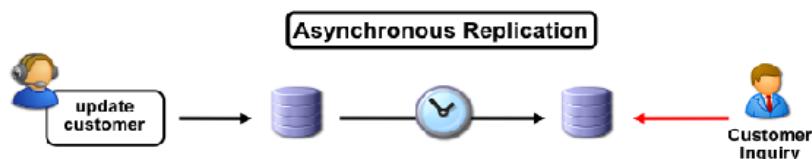
A replicação de dados, em SGBDDs, pode ser classificada quanto à **consistência (assíncrona ou síncrona)** e quanto à **topologia (hierárquica ou peer-to-peer)**.

- **REPLICAÇÃO ASSÍNCRONA:** Transação concluída em um nó e replicada para os demais:

- Nó de origem da transação não aguarda a confirmação dos demais nós do ambiente;
- Nó de origem garante a persistência da transação ocorrida nele;
- SGBDD garante a persistência da transação distribuída em todos os nós.

Mais performática para o usuário, mas possui uma consistência eventual devido ao delay para replicar a transação.

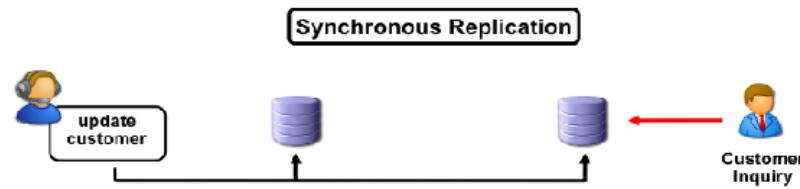
Figura 18 – Replicação Assíncrona.



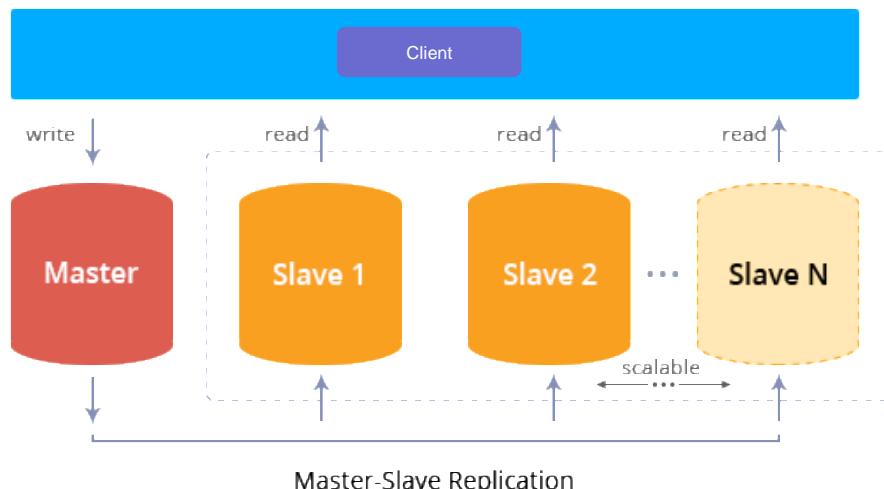
- **REPLICAÇÃO SÍNCRONA:** transação só é concluída após todos os nós confirmarem o commit.

- Nó de origem da transação aguarda a confirmação dos demais nós do ambiente.

Apesar de ser menos performática para o usuário, ela oferece consistência total dos dados, em todos os pontos do SGBDD, ou seja, em qualquer momento, o usuário enxerga o mesmo conjunto de dados em qualquer um dos nós.

Figura 19 – Replicação Síncrona.

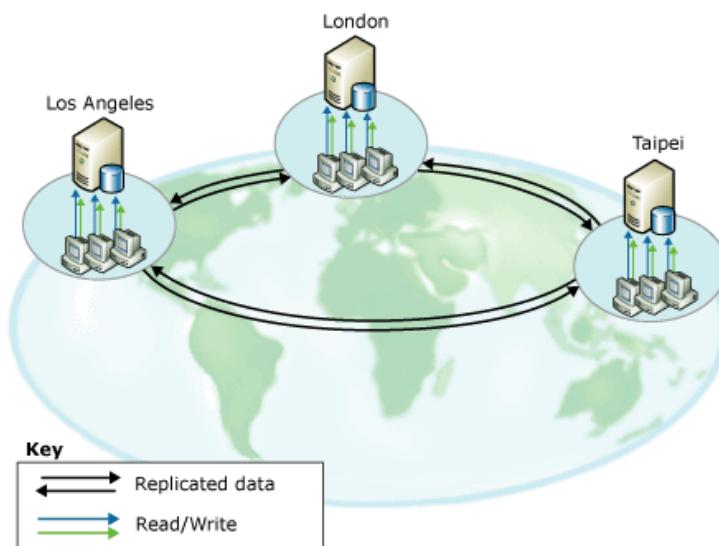
- **REPLICAÇÃO HIERÁRQUICA:** também conhecida como **single-master** ou **master-slave**, possui uma topologia com um nó principal (replica primária/master) e N nós secundários (slaves). As operações de escrita (insert/update/delete) ocorrem somente no nó primário, e é feita a replicação do log com as transações, de forma síncrona ou assíncrona. Já as operações de leitura na(s) réplica(s) secundária(s), depende do SGBDD em questão.

Figura 20 – Exemplo de replicação hierárquica.

Fonte: Gustavo A. (2020).

- **REPLICAÇÃO PEER-TO-PEER:** também conhecida como **multi-master** ou **update-anywhere**, esse tipo de replicação permite operações de escrita (insert/update/delete) em todos os nós. Pode possuir uma topologia circular, cruzada, etc.

Figura 21 – Exemplo de replicação peer-to-peer.

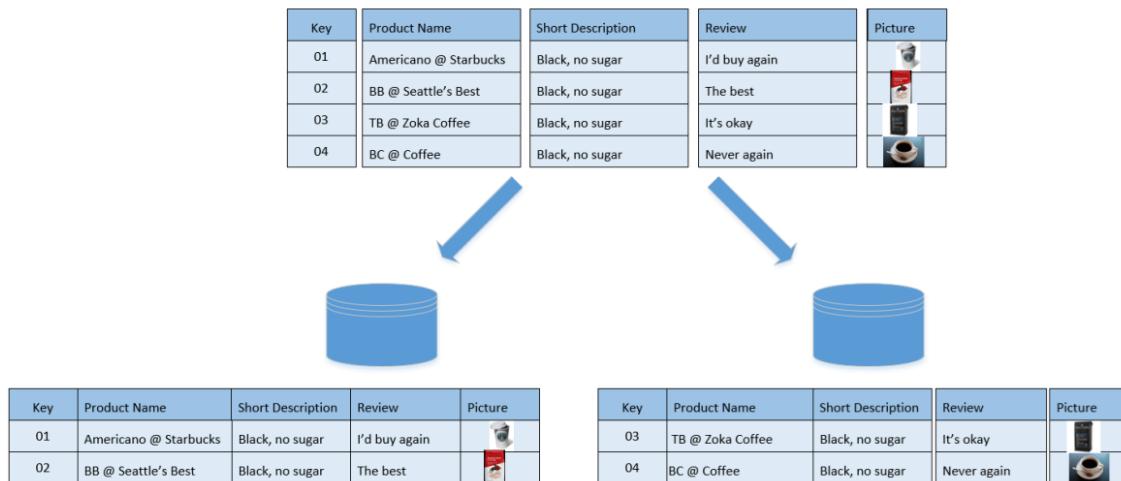


1.5. Tipos de Particionamento de Dados

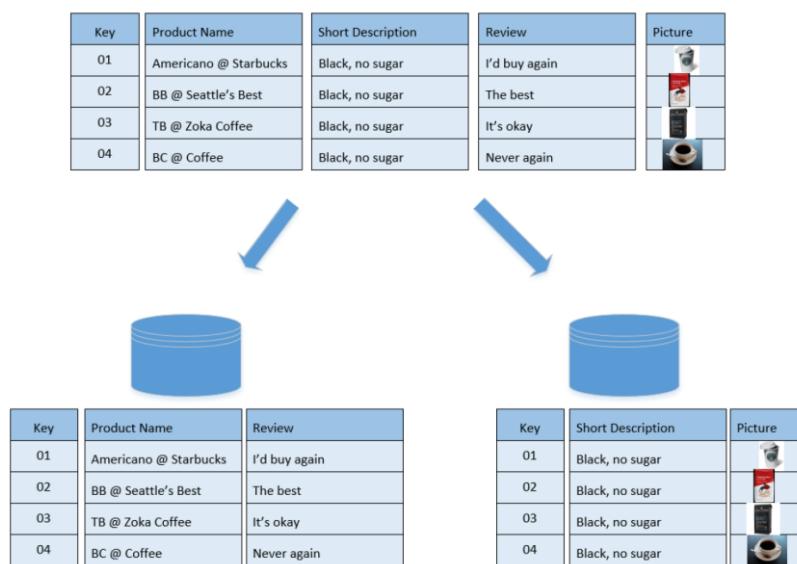
O particionamento de dados, em SGBDDs, pode ser feito de forma **horizontal, vertical e misto**.

- **PARTICIONAMENTO HORIZONTAL:** cada fragmento (shard) consiste em um subconjunto de linhas (tuplas), definido pela operação de restrição (filtro), pode ser feita por:
 - Hash (hash key);
 - Faixa de valores.

Muito utilizado em conjunto com a distribuição geográfica dos dados.

Figura 22 – Particionamento horizontal de dados.

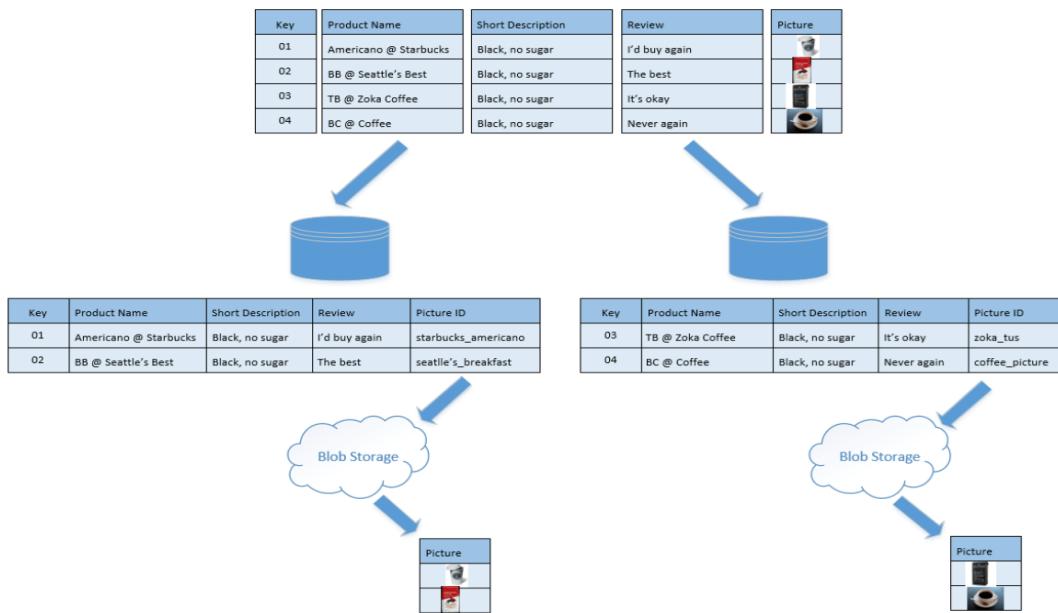
- **PARTICIONAMENTO VERTICAL:** cada fragmento (shard) consiste em um subconjunto de atributos (colunas), definidos pela operação de projeção (select), com base em na afinidade dos atributos, confidencialidade dos dados, armazenamento ou localização.

Figura 23 – Particionamento vertical de dados.

- **PARTICIONAMENTO MISTO:** utiliza as técnicas de particionamento horizontal e vertical juntas, sendo definido pela operação de restrição e pela de projeção.

Muito comum de ser usado em grandes conjuntos de dados com diferentes tipos de dados, como por exemplo, CLOB, BLOB e XML.

Figura 24 – Particionamento misto de dados.



1.6. Teorema de CAP e Propriedades BASE

Um ponto importante ao se trabalhar com bancos de dados escaláveis e, portanto, distribuídos, é compreender o **Teorema de CAP** (Consistency, Availability, Partition tolerance). Esse teorema, criado por Eric Brewer em 2000, diz que existem **três principais requisitos sistêmicos** em um ambiente distribuído:

- **Consistência:** cada usuário deve ter uma visão consistente ou igual dos dados, ou seja, todos os nós dentro de um cluster veem os mesmos dados.
- **Disponibilidade:** o sistema está disponível quando solicitado, independentemente de ter ocorrido uma falha física ou lógica em alguns dos servidores.

- **Tolerância à Partição:** o sistema continua a operar como esperado, mesmo sob circunstâncias de perda de conexão entre os nós ou perda de dados em um determinado nó.

A conclusão do Teorema de CAP é que um sistema distribuído pode garantir **apenas dois desses requisitos** e, ignorar isso, pode ter resultados catastróficos, que incluem a possibilidade de em determinada situação, nenhum dos três requisitos serem contemplados.

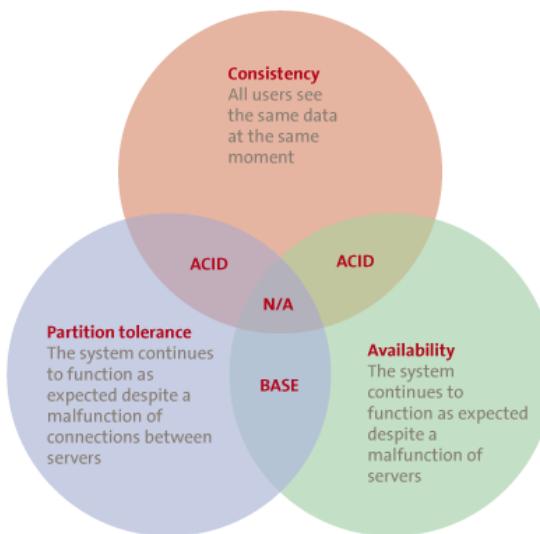
As limitações colocadas pelo Teorema CAP, acerca da confiabilidade do banco de dados, trouxeram consequências significantes para os SGBDs não relacionais distribuídos de grande escala. Como eles miravam na disponibilidade e tolerância à partição, deixavam a consistência em segundo plano, o que fazia com que as **propriedades ACID não fossem aplicáveis em sua totalidade**.

Para contornar isso, foi proposto um modelo alternativo de consistência denominado **BASE (Basically Available, Soft state, Eventual consistency)**.

- **Basicamente Disponível (Basically Available):** essa restrição afirma que o sistema garante a disponibilidade dos dados no que diz respeito ao Teorema de CAP, ou seja, haverá uma resposta a qualquer solicitação. Entretanto, pode retornar uma "falha" para obter os dados solicitados ou os dados podem estar em um estado inconsistente ou em transição.
- **Estado Não Rígido (Soft State):** o estado do sistema pode mudar ao longo do tempo, mesmo durante momentos sem entrada de dados, o que é necessário para a consistência eventual (dados sendo replicados assincronamente para os outros nós).
- **Consistência Eventual (Eventual Consistency):** os dados são propagados para todos os nós, mais cedo ou mais tarde, ou seja, assincronamente. Entretanto, o sistema continuará a receber dados e não estará verificando a consistência de todas as transações antes de passar para a próxima transação.

Com esse novo modelo de consistência no processamento das transações, permitiu-se uma clareza maior acerca do nível de confiabilidade de cada SGBD NOSQL. Além disso, foi possível um dimensionamento horizontal mais eficiente em termos financeiros, pois verificar a consistência dos dados, a cada transação, sempre acarretou em custos de processamento enormes em sistemas que realizam milhões de transações concorrentes.

Figura 25 – Teorema CAP e as Propriedades ACID e BASE.



Fonte: Hatoutdoor (2019).

A consistência eventual teve papel fundamental no mundo computacional, pois trouxe às organizações como *Google*, *Twitter* e *Facebook*, a capacidade de suportar milhões de usuários espalhados ao redor do mundo, milhares de transações por segundo, com a disponibilidade e tolerância a falhas necessárias, ao mesmo tempo em que mantiveram seus custos baixos, sem prejuízos aos dados das suas soluções.

Obviamente que o mundo perfeito seria conseguir consistência total dos dados sempre, juntamente com alta disponibilidade e tolerância à partição, mas como Dan Pritchett expôs em seu artigo “**BASE: Uma Alternativa Ácida**”, para atender ao crescimento exponencial de dados devido à era da **IOT (Internet Das Coisas)**, redes sociais, computação em nuvem e projetos de Big Data, eram necessárias mudanças

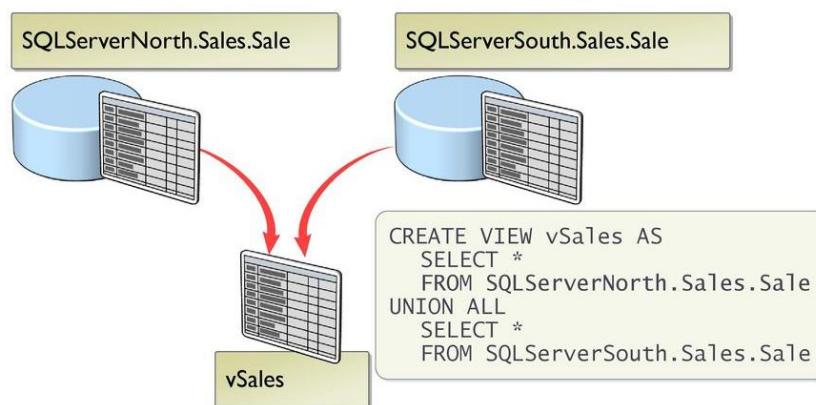
de paradigmas para se aceitar certa consistência eventual dos dados, em detrimento dos outros benefícios.

Capítulo 2. Distribuição de Dados no SQL Server

2.1. Partitioned View

Tipo de objeto do SQL Server que permite ter um banco de dados distribuído homogêneo ou heterogêneo (acessa outros SGBDs via linked server). Esse tipo de view pode ser atualizável, caso atenda algumas restrições, e pode-se fazer o particionamento horizontal ou vertical.

Figura 26 – Partitioned view.



Fonte: Microsoft (2018).

2.2. Log Shipping e SQL Server Replication

▪ LOG SHIPPING:

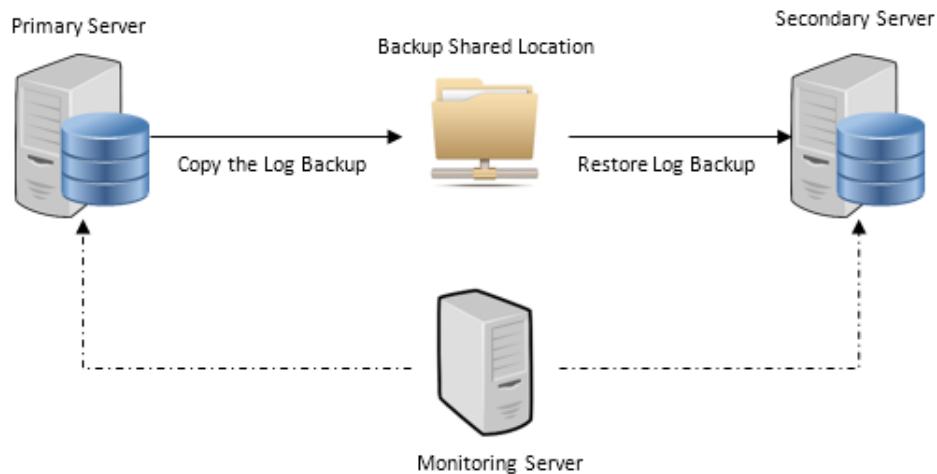
- Escrita apenas no servidor primário;
- Assíncrono;
 - Faz a replicação através de backup dos logs com o restore destes logs na réplica secundária, requerendo um local compartilhado para armazenar esses backups de log.

O servidor secundário pode ser configurado em 2 modos:

- **No recovery mode:** sem leituras;
- **Standby mode:** permite leituras.

Ideal é que as versões do SQL sejam iguais, mas a versão do secundário pode ser superior. Entretanto, em caso de failover, não tem possibilidade de fazer fallback.

Figura 27 – SQL Server Log Shipping.

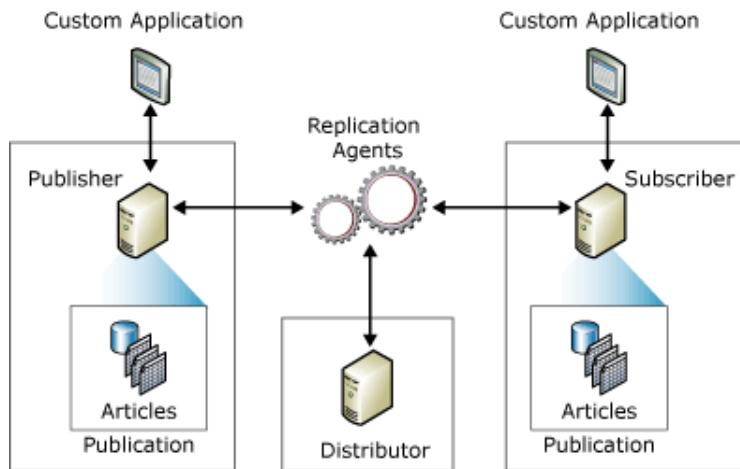


Fonte: Microsoft (2018).

- **SQL SERVER REPLICATION:**

- Replica/Distribui: banco inteiro/tabela inteira/linhas/colunas;
- Pode ser do tipo **snapshot/transacional**, permitindo leituras nas réplicas secundários, ou do tipo **merge**, que permite escrita em qualquer nó;
- Publisher: SQL e Oracle;
- Subscriber: SQL, Oracle e DB2.

Figura 28 – SQL Server Replication.



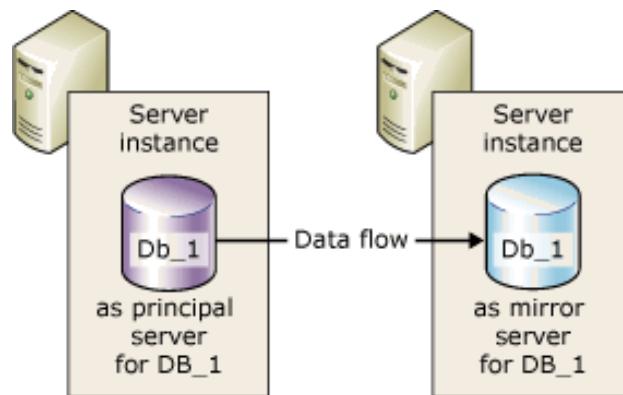
Fonte: Microsoft (2018).

2.3. Database Mirroring e AlwaysOn Availability Groups

▪ DATABASE MIRRORING:

- Replicação das transações existentes no transaction log, assim como no Log Shipping, mas não requer pasta compartilhada;
- Escrita apenas no servidor primário;
- Servidor secundário em recovery mode (não acessível para leituras);
- Replicação assíncrona ou síncrona;
 - Assíncrona: mais performática, mas posso apenas failover manual;

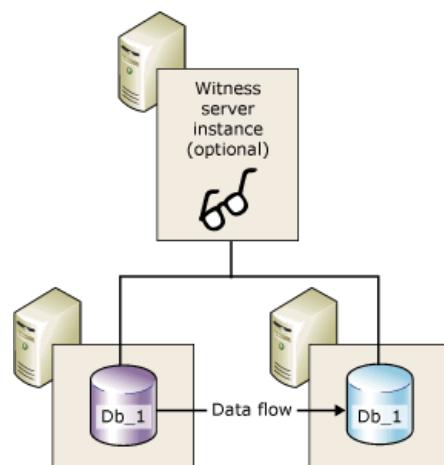
Figura 29 – Database mirroring assíncrono.



Fonte: Microsoft (2018).

- Síncrona
 - Mais segura;
 - Menos performática;
 - Failover manual;
 - Failover automático:
 - ✓ Requer witness.

Figura 30 – Database mirroring síncrono com Witness.

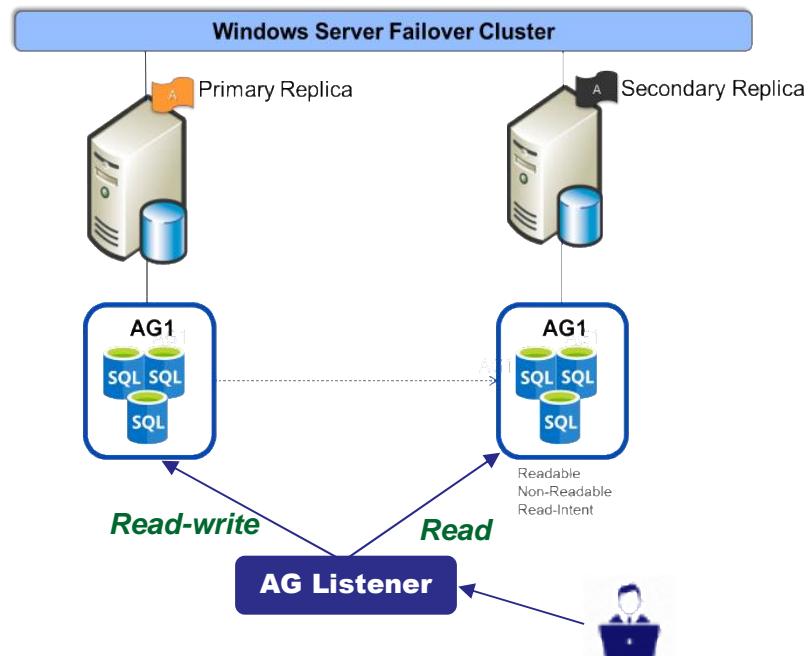


Fonte: Microsoft (2018).

- **ALWAYSON AVAILABILITY GROUPS:**

- Replicação das transações:
 - Modo Síncrono → mais seguro;
 - Modo Assíncrono → mais performático.
- Escrita apenas no servidor primário;
- Servidor secundário permite leitura:
 - Backup no secundário.
- Listener: fornece a transparência de localização.

Figura 31 – SQL Server AlwaysOn Availability Groups.



Fonte: Microsoft (2018).

Capítulo 3. Banco de Dados Distribuído como Serviço

3.1. Azure SQL Database Managed Instance

Instância de banco de dados SQL Server gerenciada (SQL Database Managed Instance) é fornecida em duas camadas de serviço: **Propósito Geral (General Purpose)** e **Negócios Críticos (Business Critical)**. Ambas as camadas de serviço suportam o mesmo conjunto de recursos, e as principais diferenças entre as duas estão relacionadas ao desempenho e disponibilidade. As únicas diferenças de recursos entre as duas camadas são que a Business Critical suporta OLTP In-Memory e leitura nas réplicas secundárias. Ela também inclui mais memória por núcleo (vCore) e usa armazenamento atachado direto (ao contrário de SAN), o que oferece menor latência de armazenamento.

É possível aproveitar as licenças de SQL Server já adquiridas ao se migrar para SQL Managed Instacen. Para cada núcleo do Enterprise Edition com Active Software Assurance, você é elegível para um vCore do Azure SQL Database ou Managed Instance Business Critical e oito vCores de General Purpose. Para cada núcleo da Standard Edition com Software Assurance que você possui, você é elegível para um vCore da General Purpose. Isso pode reduzir o custo total da licença em até 40%.

O banco de dados SQL do Azure e a instância gerenciada têm arquiteturas semelhantes de alta disponibilidade, que garantem 99,99% por cento de tempo de atividade. As atualizações do Windows e do SQL Server são tratadas pela infraestrutura de back-end e de responsabilidade do Azure. A solução de alta disponibilidade é automática e integrada à plataforma e foi projetada para que os dados comprometidos nunca sejam perdidos por falhas e os bancos de dados não tenham um ponto único de falha.

O backup gerenciado fornece um serviço de backup totalmente gerenciado que realiza backups completos, diferenciais e de log regularmente. É possível também realizar manualmente backups “copy only” dos bancos de dados no Azure.

3.2. Azure SQL Database

O banco de dados SQL do Azure é um mecanismo de banco de dados PaaS (plataforma como serviço) totalmente gerenciado que lida com a maioria das funções de gerenciamento de banco de dados, como atualização, aplicação de patches, backups e monitoramento sem envolvimento do usuário. O banco de dados SQL do Azure está sempre em execução na versão estável mais recente do SQL Server e de patches do sistema operacional, com 99,99% de disponibilidade.

Possui 2 modelos de implantação:

- **Single Database:**

- Banco de dados isolado;
 - Totalmente gerenciado pelo Azure.

- **Elastic Pool:**

- Coleção de single databases;
 - Com um conjunto compartilhado de recursos (CPU / RAM).

Modelos de contratação

- **Modelo de compra baseado em vCore:** permite escolher o número de vCores, a quantidade de memória e a quantidade e velocidade do storage.
- **Modelo de compra baseado em DTU:** oferece uma combinação de recursos de computação (CPU), memória RAM e I/O.
- **Modelo serverless (sem servidor):** dimensiona automaticamente a configuração necessária de recursos com base na demanda da carga de trabalho e cobra pela quantidade de computação usada por segundo. Neste modelo, os bancos de dados também são pausados automaticamente durante os períodos inativos. Dessa forma, apenas o armazenamento é cobrado, e os bancos de dados são ativados automaticamente quando a atividade retorna.

3.3. Azure CosmosDB

Antes de se criar um banco de dados CosmosDB, é preciso criar uma conta do CosmosDB. Uma conta do Azure Cosmos DB é um recurso do Azure que atua como uma entidade organizacional para seus bancos de dados. Ele conecta seu uso à sua assinatura do Azure para fins de cobrança. Cada conta do Azure Cosmos DB está associada a um dos vários modelos de dados aos quais o Azure Cosmos DB oferece suporte podem ser criadas quantas contas precisar.

Figura 32 – Conta Azure CosmosDB.



Fonte: Microsoft (2019).

O Azure CosmosDB mede a taxa de transferência usando uma métrica chamada unidade de solicitação (Requisition Unit - RU). O uso da unidade de solicitação é medido por segundo, portanto, a unidade de medida para o CosmosDB é unidades de solicitação por segundo (RU/s). Deve-se reservar o número de RU/s que deseja que o Azure CosmosDB provisione com antecedência, para que ele possa lidar com a carga estimada, mas pode-se aumentar ou diminuir a RU/s a qualquer momento.

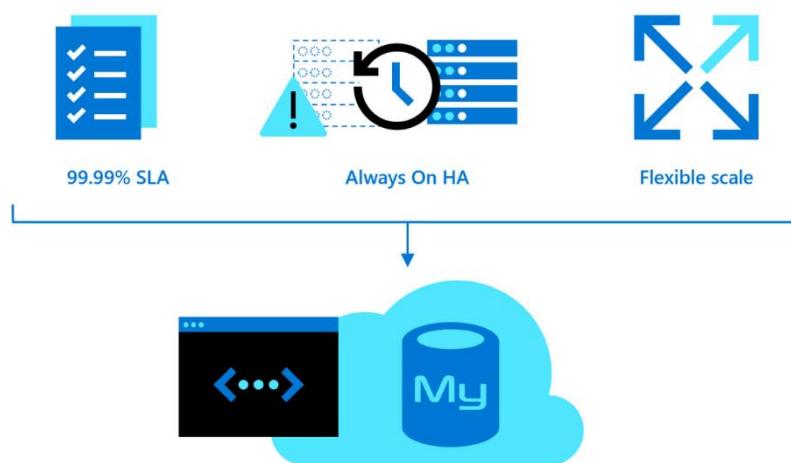
3.4. Bancos de Dados Open Source no Azure

Com o mesmo viés e implicações da modalidade de PaaS oferecida com o Azure SQL Database, é possível criar os seguintes bancos de dados open source no Azure:

- Azure Database for MySQL;
- Azure Database for MariaDB;
- Azure Database for PostgreSQL.

Para o MySQL e MariaDB, há o recurso de replicação (master/slave), mas também existindo a opção de criação de um database único (single database).

Figura 33 – Azure Database para MySQL.



Fonte: Microsoft (2019).

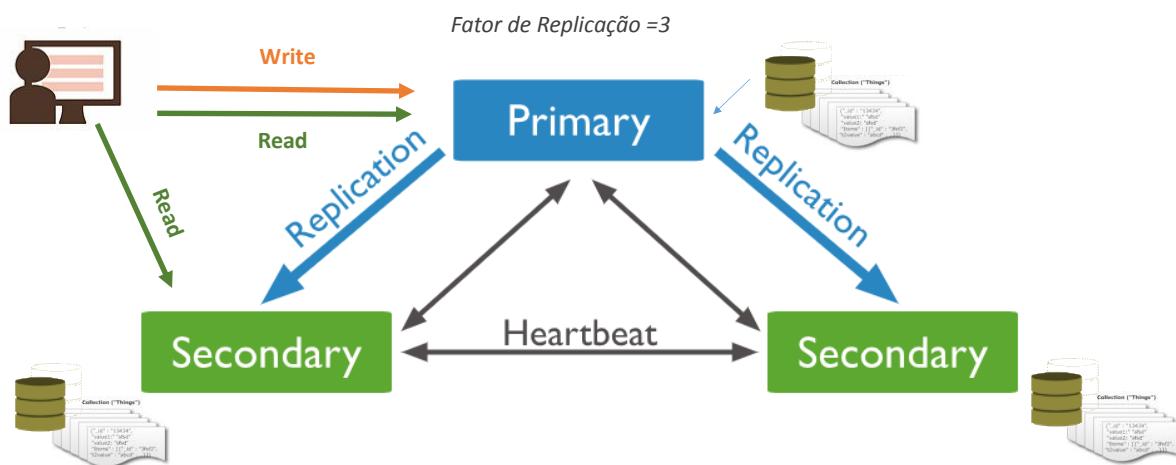
Capítulo 4. Distribuição de Dados no MongoDB

4.1. Replica Set

O recurso *replica set* do MongoDB fornece redundância e aumenta a disponibilidade dos dados. Com várias cópias de dados em diferentes servidores de banco de dados, sendo replicados de forma assíncrona, a replicação fornece um nível de tolerância a falhas contra a perda de um único servidor de banco de dados.

Em alguns casos, a replicação pode fornecer maior capacidade de leitura, pois os clientes podem enviar operações de leitura para servidores diferentes, mas se falando de operações de escrita, as mesmas só podem ser feitas no servidor primário.

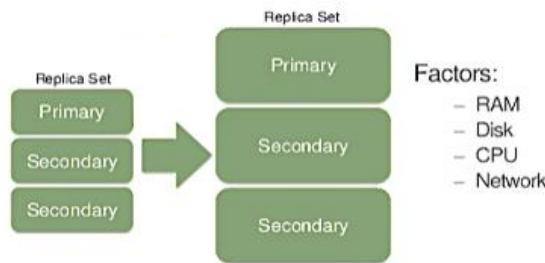
Figura 34 – MongoDB replica set.



Fonte: MongoDB (2018).

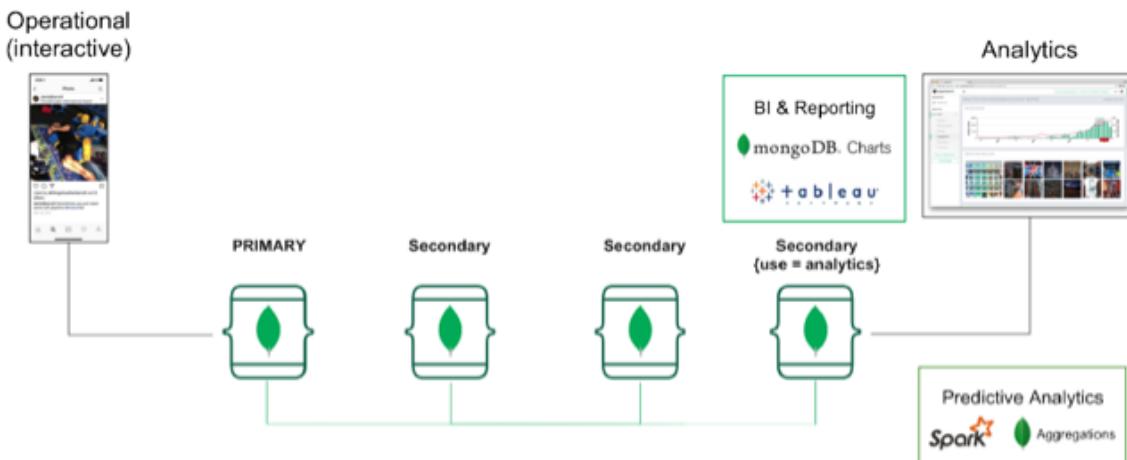
O direcionamento das operações de leitura para as réplicas secundárias pode ser feito via parâmetro, na string de conexão, usando **secondary Preferred** (leitura preferencialmente nas secundárias) ou **secondary** (leitura somente nas secundárias).

Com esse tipo de distribuição de dados, é possível ter escalabilidade vertical na camada de banco de dados MongoDB.

Figura 35 – Escalabilidade Vertical no MongoDB.

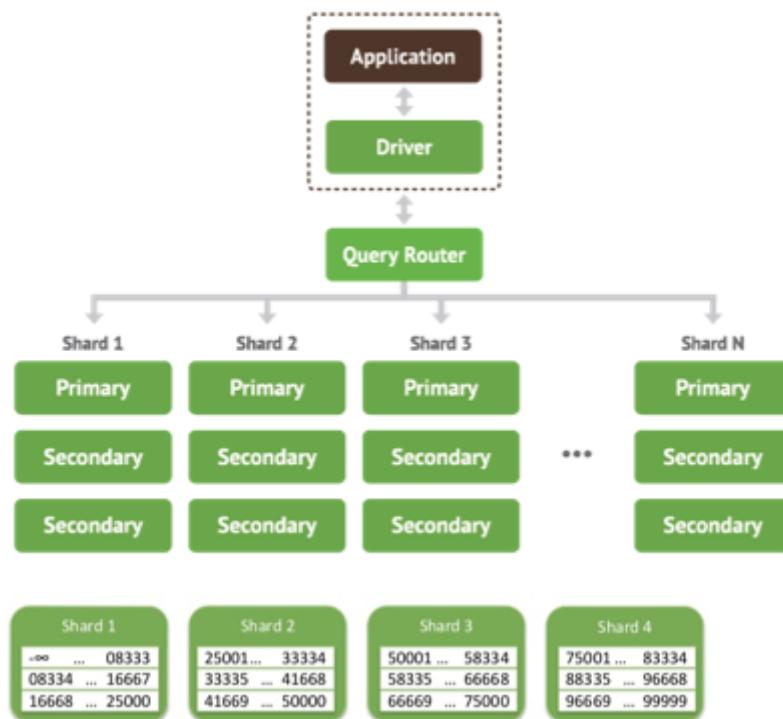
Fonte: Gustavo (2018).

Usando esse mecanismo de replica set, pode-se também obter ganhos de performance ao separar os diversos workloads de um ambiente, do workload de escrita.

Figura 36 – Separação de workloads no MongoDB.

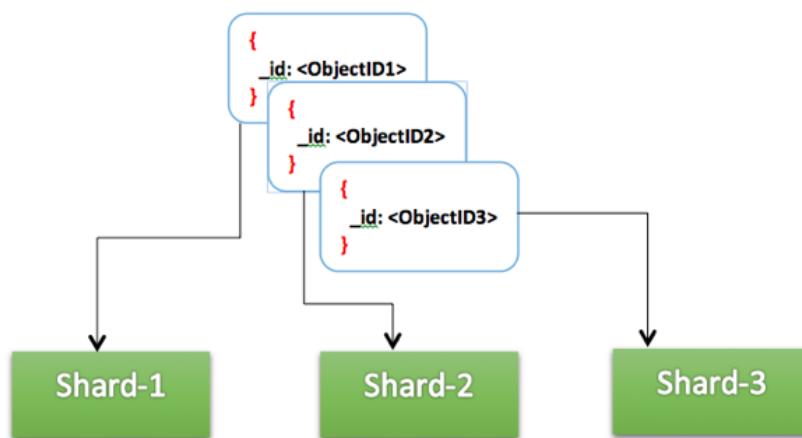
4.2. MongoDB Sharding

Com este recurso, que usa técnicas de particionamento horizontal (sharding) para permitir leitura e escrita em todos nós, é possível ter escalabilidade horizontal e vertical. Além disso, como cada shard é replicado, tem-se também alta disponibilidade para a camada de banco de dados.

Figura 37 – MongoDB Sharding.

Fonte: MongoDB (2018).

Os shards podem ser definidos chave hash ou por uma chave com intervalo de valores, sendo a primeira, a mais recomendada para ambientes críticos e de altos volumes de dados.

Figura 38 – MongoDB Sharding com Hash.

Fonte: MongoDB (2018).

4.3. MongoDB Atlas

MongoDB Atlas (www.mongodb.com/cloud/atlas) é um banco de dados em nuvem totalmente gerenciado, desenvolvido e fornecido pela MongoDB como plataforma como serviço (PaaS).

O Atlas lida com toda a complexidade de implantação, gerenciamento e monitoramento, e usa provedor de serviços de nuvem de sua escolha (AWS, Azure ou GCP). Com ele, é possível criar facilmente clusters de bancos de dados replicados ou particionados, incrementando a performance da camada de banco de dados.

Figura 39 – MongoDB Atlas.

The screenshot shows the MongoDB Atlas web interface. At the top, there's a navigation bar with 'M001-MongoDB Basic...', 'Access Manager', 'Support', 'Billing', 'All Clusters' (set to 'GUSTAVO A A'), and a user icon. Below the navigation is a secondary header with 'Atlas' (highlighted in green), 'Realm', 'Charts', and a 'Create a New Cluster' button. On the left, a sidebar has 'DATA STORAGE' with 'Clusters' selected (highlighted in green) and other options like 'Triggers', 'Data Lake'. Under 'SECURITY', it lists 'Database Access', 'Network Access', and 'Advanced'. The main content area shows 'M001-MONGO000 BASICS (ORGANIZATION) > M001-MONGO000 BASICS' and the 'Clusters' section. It displays a cluster named 'Cluster0' (Version 4.2.8) under a 'SANDBOX' tier. The cluster details include 'CONNECT', 'METRICS', 'COLLECTIONS', and a 'Monitoring for Cluster0 is Paused' message. The monitoring status says 'Monitoring will automatically resume when you connect to your cluster. Visit the documentation for more info.' There are also 'REGION' (AWS / N. Virginia (us-east-1)) and 'TYPE' (Replica Set - 3 nodes) details, along with a 'LINKED REALM APP' section.

Fonte: Gustavo (2020).

Capítulo 5. Aspectos Gerais de Performance em Banco de Dados

5.1. Aspectos Gerais de Infraestrutura

Ao se falar de performance em banco de dados, muitos aspectos devem ser verificados, a começar pela infraestrutura montada para acomodar o SGBD e os bancos de dados. Alguns componentes impactam diretamente na performance das queries e precisam ser bem escolhidos, de acordo com os requisitos não funcionais da aplicação.

Dentre os mais importantes, temos:

- **Storage**

- Tipo dos discos impacta diretamente na performance do banco de dados, dada a **taxa de IOPS** de cada um deles, ou seja, a quantidade de operações de leitura e escrita que conseguem fazer por segundo.

Figura 40 – Tipos de Discos x IOPS.

	3,5"	2,5"
5900 RPMs	50 IOPS	-
7200 RPMs	75 IOPS	95 IOPS
10000 RPMs	110 IOPS	140 IOPS
15000 RPMs	150 IOPS	180 IOPS
SSD	1500 IOPS	1500 IOPS

Fonte: Gustavo (2020).

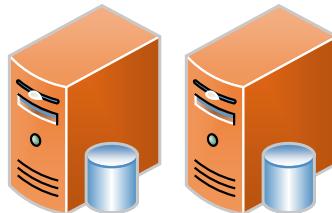
- A arquitetura ou estratégia de alocação dos discos também podem impactar na performance do banco de dados.

- **Disco Atachado Diretamente no Servidor:**

- ✓ Boa performance;

- ✓ “Ilhas” isoladas de storage nos servidores;
- ✓ Baixa tolerância a falhas (exceto se for virtual);
- ✓ Expansão fica limitada.

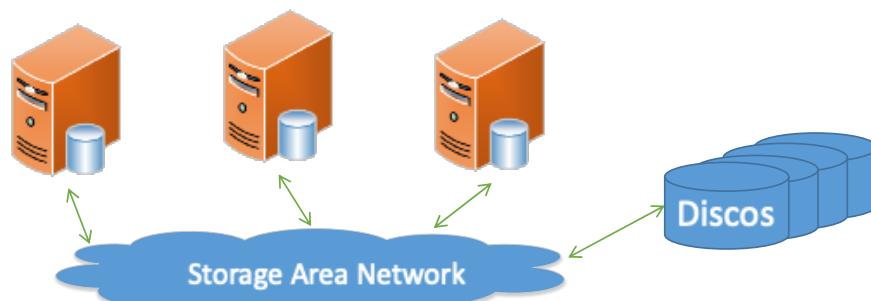
Figura 41 – Disco atachado diretamente no servidor.



- **Storage Area Network (SAN)**

- ✓ Grande “ilha” de discos ligada ao parque de servidores, que pode ser facilmente expandida;
- ✓ Montado com rede dedicada que fornece acesso rápido aos discos;
- ✓ Incrementa a utilização da capacidade de armazenamento, com recursos de alocação do espaço físico sob demanda, como o thin provision;
- ✓ Discos “virtuais” (luns) são dedicados (não compartilhados com outro servidor).

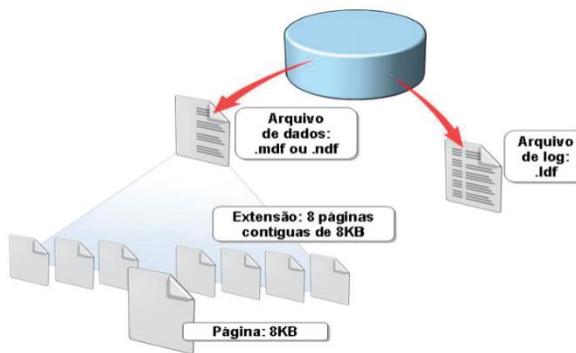
Figura 42 – Storage Area Network (SAN).



- O tipo de formatação do disco impacta diretamente na performance.

No SQL Server, por exemplo, que possui páginas de dados de 8 KB e faz leitura de *extents* (segmento contíguo de 8 páginas) de 64 KB, a formatação recomendada é a NTFS com blocos de 64KB (➔ uma operação de I/O = um extent de 64KB). Com blocos de 4 KB por exemplo, para a leitura de 1 extent seriam necessárias 16 operações de I/O para ler os 64KB de páginas.

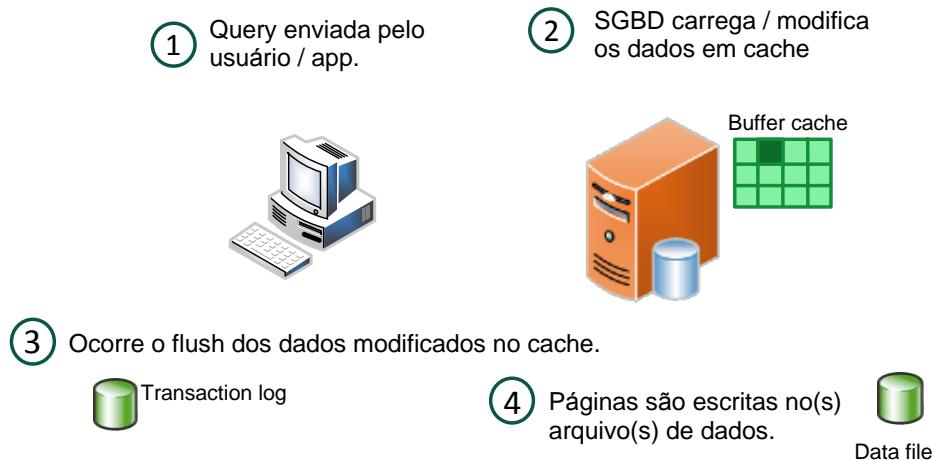
Figura 43 – Esquema de alocação interno do SQL Server.



Fonte: Microsoft (2018).

- **Memória RAM**

- Normalmente, banco de dados usa mais memória (cache) que CPU. Devido a isso, é importante fazer uma projeção baseada na relação **Total de Storage para Dados x Memória RAM para SGBD**.

Figura 44 – Utilização de cache no SQL Server.

Fonte: Gustavo (2020).

- **Unidades de Processamento (CPU)**

- Necessário avaliar a quantidade de CPU x Paralelismo Desejado;
- Alto consumo constante de CPU normalmente está relacionado à falta de índice, fragmentação, estatística desatualizada, ou excesso de concorrência/paralelismo em detrimento do número de CPUs.

Nessa seara de recursos de infraestrutura, temos sempre que ter em mente as seguintes premissas:

- Recurso é finito:
 - Deve ser bem dimensionado bem para o projeto;
 - Planejar o Crescimento Vegetativo (CVT) anual/semestral, etc.;
 - Acompanhar e tomar medidas rápidas para os desvios.
- Queries ineficientes podem ser compensadas quando há recursos de sobra, até que um belo dia começam a apresentar problemas de performance sem que ela tenha sofrido alterações;

- Queries eficientes podem estar sendo impactadas por escassez de recursos/topologia da solução:
 - Ausência de escalabilidade horizontal;
 - Estrutura/formatação dos discos;
 - Quantidade de CPU e memória.

5.2. Aspectos Gerais dos SGBDs

Se tratando de performance na camada interna do SGBD, é preciso ter uma atenção aos aspectos a seguir que, assim como os aspectos de infraestrutura, pode impactar diretamente no desempenho das queries nos bancos de dados.

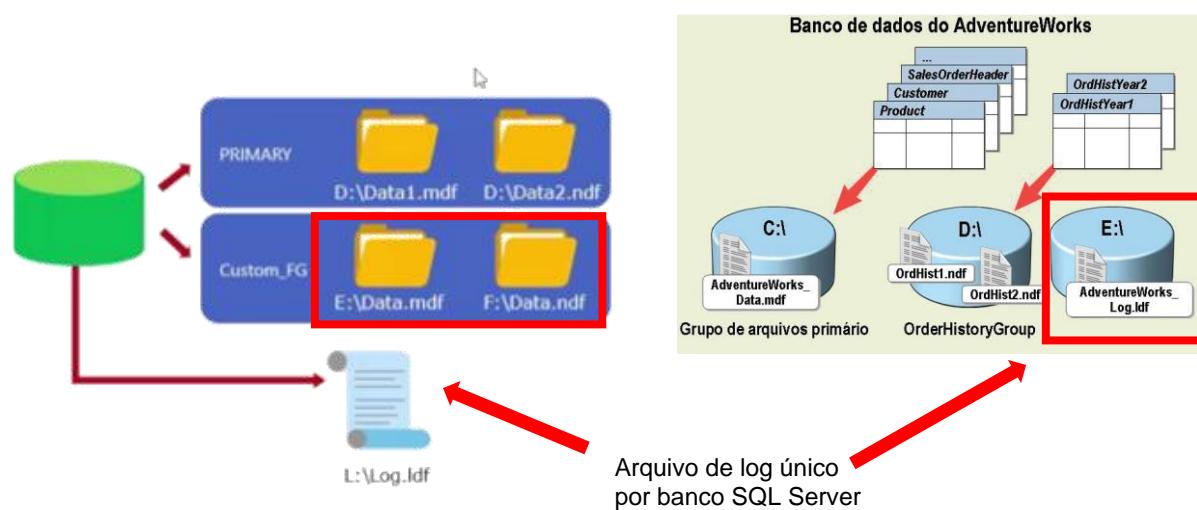
▪ **Aspectos da Instância**

- Configuração correta dos parâmetros que afetam diretamente a performance dos bancos de dados, como por exemplo:
 - Memória: mínimo e máximo;
 - CPU: usar todas? Afinidade de CPU?
 - Paralelismo: ativado/desativado? Métrica para paralelizar?
 - Número máximo de sessões abertas;
 - Quantidade máxima de sessões concorrentes;
 - Configurar algum mecanismo de gerenciamento dos recursos internos da instância entre os bancos (Resource Governor)?
- Cada instância possui seu conjunto de discos, de forma a estarem isoladas uma das outras (nos cenários de servidores compartilhados) em termos de I/O e capacidade de armazenamento?

▪ Aspectos da Estrutura do Banco de Dados

- Separação dos bancos de dados de sistema (dicionário de dados do SGBD) dos bancos de usuário (aplicação);
- Separação das estruturas peculiares de cada banco de dados, em discos distintos.
- No caso do SQL Server, por exemplo:
 - Distribuição do filegroup em vários discos → paralelismo de gravação e leitura.
 - Datafiles com mesmo size e crescimento → crescimento uniforme ao longo de todos os discos.
 - Separação de dados históricos em outros discos.
 - Arquivo único para o transaction log.

Figura 45 – Estruturas de Bancos de Dados SQL Server.



Fonte: Gustavo (2020).

- **Aspectos da Estrutura da Criação de Tabelas**

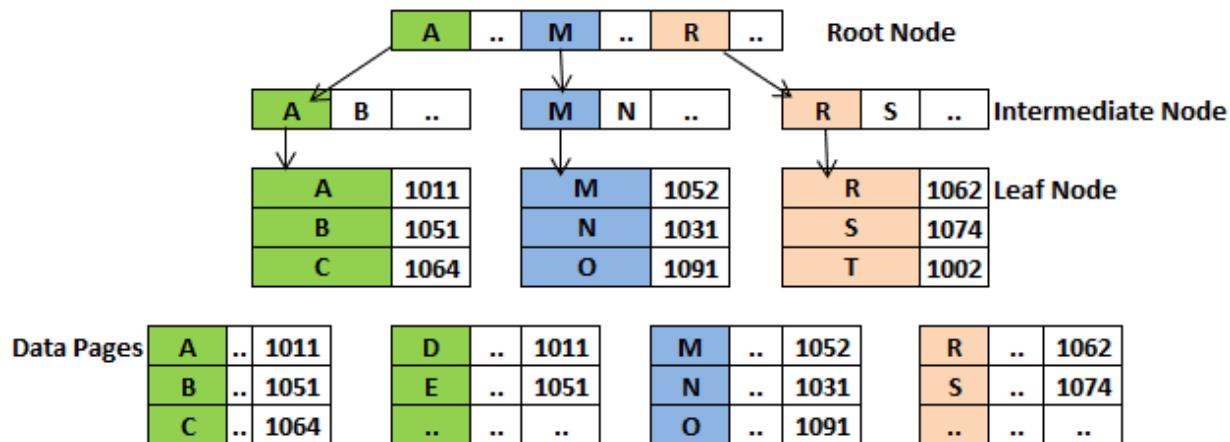
- Segregar fisicamente tabelas e índices;
- Usar compressão de dados na tabela?
 - De página?
 - De linhas?
- Tabela em memória?
- Tabela particionada?
- Tipo de tabela x Meio de armazenamento está adequado?
- Rotina de manutenção da tabela implementada?
 - Desfragmentação;
 - Atualização de estatísticas.

5.3. Índices

A palavra índice vem do latim *índex* (“o que indica”), e em um livro se trata de uma lista ordenada dos capítulos e sessões que essa publicação contém, com o respectivo local (número da página) onde eles podem ser encontrados. Dessa forma, a função do índice é reduzir o tempo gasto para localizar determinado conteúdo.

Em SGBDs, o índice é uma estrutura auxiliar associada a uma tabela, geralmente organizada como uma árvore binária, contendo uma estrutura com ponteiros para os dados armazenados.

Figura 46 – Árvore binária de um índice.



Dessa forma, o SGBD utiliza essa estrutura (o índice) de maneira semelhante ao índice remissivo de um livro, com o objetivo de acelerar o tempo de execução de uma consulta, incrementando a performance:

- Consulta-o primeiramente para encontrar determinado assunto;
- Localiza a sua posição em uma determinada página;
- Vai até essa página e lê os dados.

Em termos de **composição**, o índice pode ser:

- **SIMPLES** (composto por apenas uma coluna);
- **COMPOSTO** (composto por duas ou mais colunas).
- **Exemplo:** cláusula WHERE utilizando-se as colunas COL1 e COL2.
 - Índice simples para a coluna COL1 e outro índice simples para a coluna COL2 → **não atende**.
 - Índice composto pelas colunas COL1 e COL2 → **será usado pela consulta**.

Em termos de **funcionalidade**, cada fornecedor disponibiliza os tipos de índices mais aderentes ao seu SGBD.

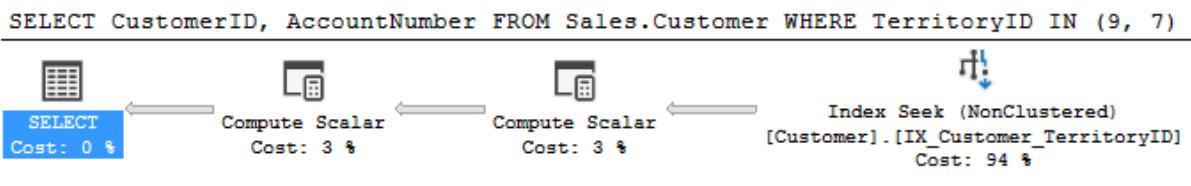
- **ÍNDICE ÚNICO**: não permite valores repetidos para a(s) coluna(s).
- **ÍNDICE NÃO ÚNICO**: permite valores repetidos para a(s) coluna(s).
- **ÍNDICE PRIMÁRIO**: sempre único, correspondendo à chave primária da tabela.
- **ÍNDICE SECUNDÁRIO**: pode ser único (chave candidata) ou não único.

Apesar de num primeiro momento se ter a impressão de que os índices são sempre benéficos, temos que ter em mente os efeitos colaterais dos mesmos:

- Índices ocupam **espaço adicional**;
- Índices podem tornar as **operações de carga, exclusão ou atualização de dados** mais lentas, devido às possíveis operações de reordenação e/ou atualização do índice;
- Índices podem interferir na **cardinalidade** dos relacionamentos existentes no modelo de dados físico;
- Índices precisam de **manutenção periódica** para remoção da fragmentação → **rebuild / reorg** e **atualização de estatísticas**.

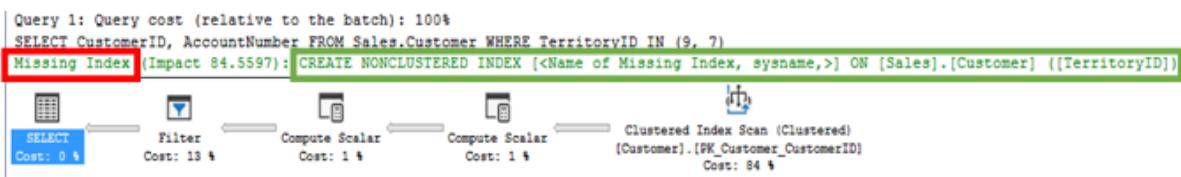
5.4 Estatísticas e Plano de Execução de Query

Estatísticas e plano de execução são usados pelo SGBD para indicar como cada query dever ser executada e contém os objetos que serão utilizados na query, os índices e mecanismos de acesso.

Figura 47 – Plano de execução de query do SQL Server.

Fonte: Gustavo (2020).

A análise do plano de execução da query é uma etapa fundamental e que reduzirá as chances de execução de queries que gerarão impactos negativos na performance do ambiente. A grande maioria dos SGBDS disponibiliza interfaces gráficas para visualizar o plano de execução e algumas até sugerem a criação de índices, inclusive já com o comando DDL para tal.

Figura 48 – Sugestões do plano de execução de query do SQL Server.

Fonte: Gustavo (2020).

Já as estatísticas, se tratando de uma amostra dos dados em uma determinada tabela/coluna de índice, são utilizadas pelo otimizador para determinar o melhor método de acesso para determinadas operações de acesso aos dados. Com isso, valores desatualizados ou distorcidos podem levar a planos de execução inefficientes, o que implica na necessidade de manutenção constante das estatísticas, que podem ser criadas/atualizadas manualmente ou automaticamente.

Figura 49 – Estatísticas do SQL Server.

The screenshot shows the SQL Server Management Studio interface. On the left, there's a tree view of database objects. In the center, there are two tabs: 'Results' and 'Messages'. The 'Results' tab contains two tables of data. The first table, titled 'IX_SalesOrderDetail_ProductID', has columns: Name, Updated, Rows, Rows Sampled, Steps, Density, Average key length, and String Index. It shows one row for the index IX_SalesOrderDetail_ProductID. The second table has columns: RANGE_HI_KEY, RANGE_ROWS, EQ_ROWS, DISTINCT_RANGE_ROWS, and AVG_RANGE_ROWS. It shows data for Product IDs 730, 732, 738, 741, and 742. To the right of these tables is a SQL query window displaying the following code:

```
SELECT ProductID, RecordCount = COUNT(*)
FROM Sales.SalesOrderDetail
WHERE ProductID >= 732 AND ProductID <= 738
GROUP BY ProductID
```

Below the query window is another 'Results' tab showing the execution results of the query, which are the same five rows from the second table above.

Name	Updated	Rows	Rows Sampled	Steps	Density	Average key length	String Index
IX_SalesOrderDetail_ProductID	Nov 7 2012 6:44PM	121317	121317	200	0.0078125	12	NO

RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
730	0	288	0	1
732	0	130	0	1
738	154	600	2	77
741	167	94	1	167
742	0	288	0	1

ProductID	RecordCount
732	130
733	44
736	110
738	600

Fonte: Gustavo (2020).

Capítulo 6. Otimização de Query no SQL Server

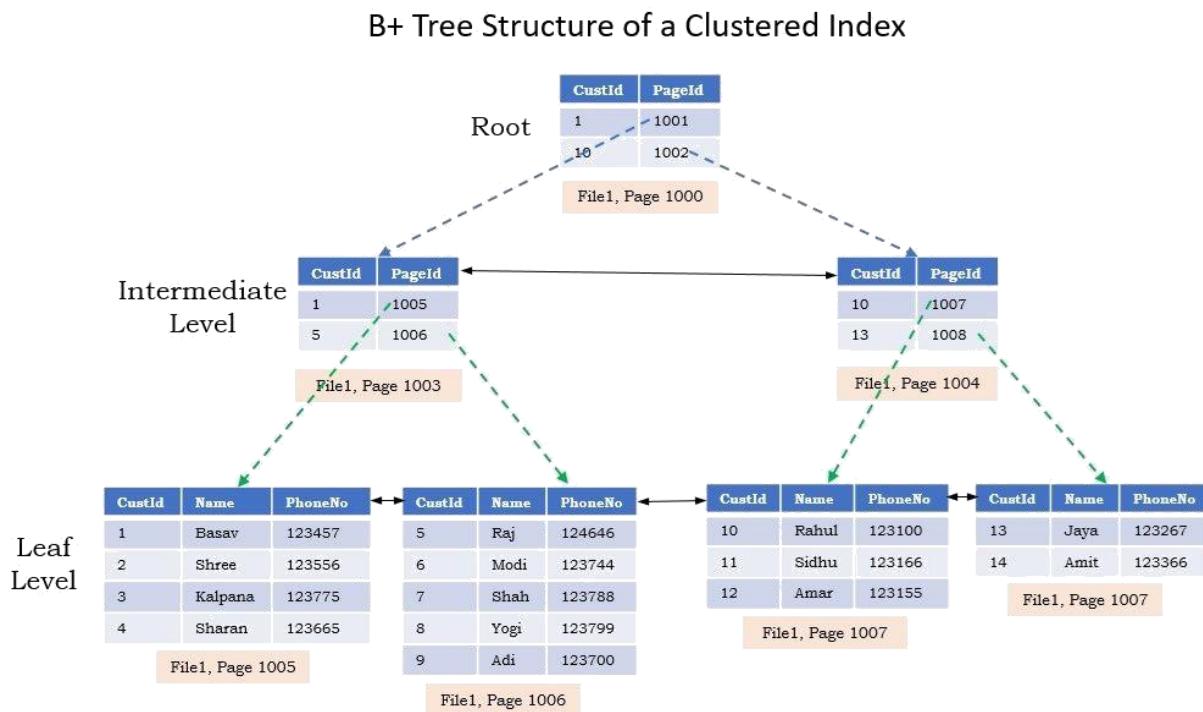
6.1. Tipos de Índices

▪ Clustered Index

Os índices clusterizados não têm uma estrutura separada das linhas de dados. Os índices clusterizados classificam e armazenam as linhas de dados na tabela ou exibição com base em seus valores-chave, podendo existir apenas um índice clusterizado por tabela, porque as próprias linhas de dados podem ser armazenadas em apenas uma ordem.

Quando uma tabela tem um índice clusterizado, a tabela é chamada de tabela clusterizada. Se uma tabela não tiver um índice clusterizado, suas linhas de dados serão armazenadas em uma estrutura não ordenada chamada heap.

Figura 50 – Estrutura de índice cluster do SQL Server.



Criado com o comando:

CREATE [UNIQUE] CLUSTERED INDEX NOME_DO_INDICE

ON NOME_DA_TABELA (COLUNA1, [COLUNA2],...)

Ou criado automaticamente quando se define uma constraint primary key:

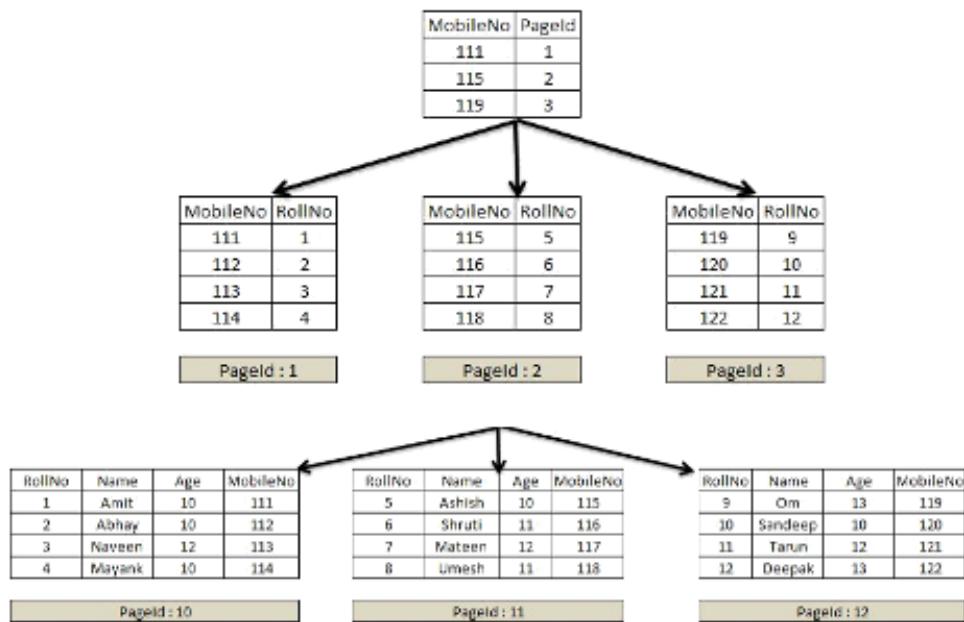
CREATE TABLE NOME_DA_TABELA

```
(    COLUNA1,  
    COLUNA2....  
    PRIMARY           KEY           CLUSTERED  
    (COLUNA1)  
    ON                [PRIMARY]  
 ) ON [PRIMARY]
```

- **Non Clustered Index**

Os índices não clusterizados têm uma estrutura separada das linhas de dados, contendo os valores-chave do índice não clusterizado, e cada entrada de valor-chave tem um ponteiro para a linha de dados que contém o valor-chave.

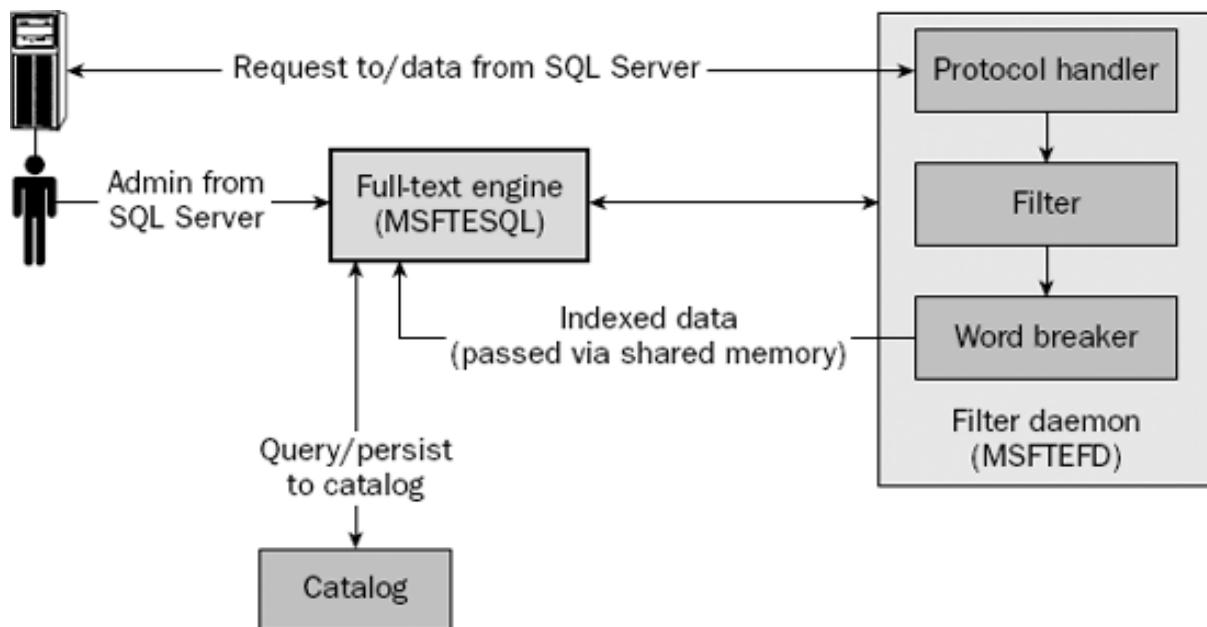
O ponteiro de uma linha de índice em um índice não clusterizado para uma linha de dados é chamado de localizador de linha. A estrutura do localizador de linha depende se as páginas de dados são armazenadas em uma heap ou em uma tabela clusterizada. Para uma heap, um localizador de linha é um ponteiro para a linha. Para uma tabela clusterizada, o localizador de linha é a chave do índice clusterizado.

Figura 51 – Estrutura de índice não cluster do SQL Server.

- **Full Text Index**

- Usado para filtros complexos em string de dados.
 - Permite consultas semânticas nos dados;
 - Pesquisas em campos binários (PDF), etc.
- Um full text index por tabela;
- Requer uma coluna não nula única na tabela;
- Requer a instalação da engine (serviço) Full-Text;
- Necessário criar e manter os catálogos;
- Comandos T-SQL específicos Specialized Transact-SQL.
 - (ex.: WHERE CONTAINS(Name, 'formsof(freetext, nut)').

Figura 52 – Full Text Index do SQL Server.



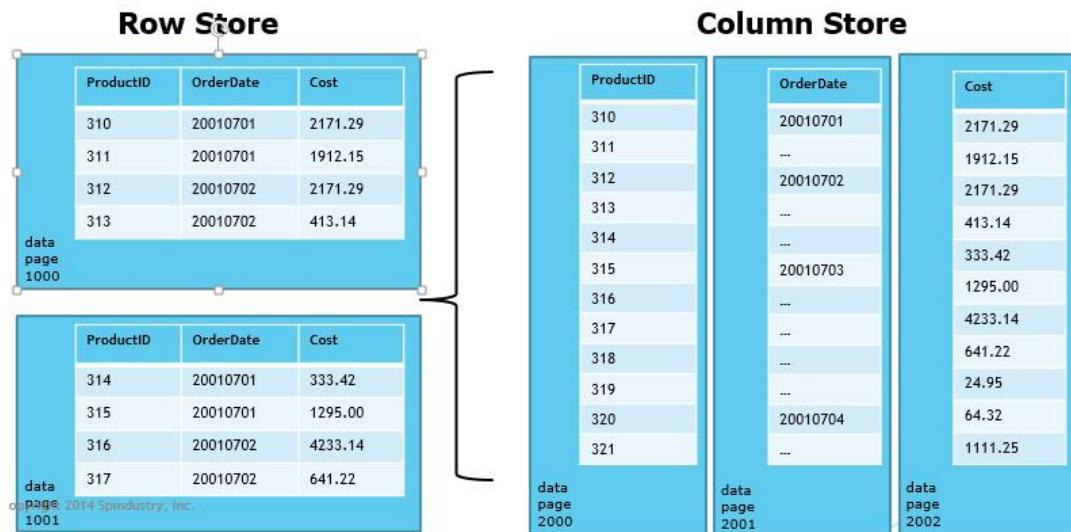
- **Columnstore Index**

- Recurso para armazenar, recuperar e gerenciar dados usando um formato de dados **colunar**, chamado **columnstore**;
- Um columnstore são dados logicamente organizados como uma tabela com linhas e colunas e, fisicamente, armazenados em formato colunar de dados;
- As colunas armazenam valores do mesmo domínio e geralmente têm valores semelhantes, o que resulta em altas taxas de compactação.
 - Altas taxas de compactação melhoram o desempenho da consulta usando um espaço menor na memória.

Quando usar:

- Para armazenar tabelas de fatos e tabelas grandes de dimensão;
- Para realizar análises em tempo real em uma carga de trabalho OLTP.

Figura 53 – Columnstore Index do SQL Server.



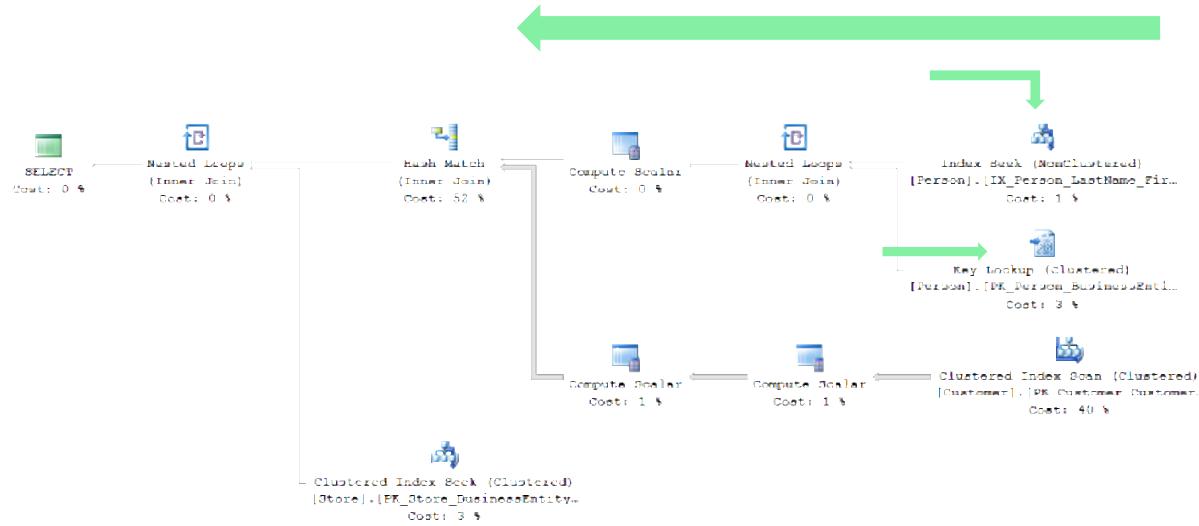
6.2. Análise do Plano de Execução de Query

Conteúdo do plano de execução da query:

- Como os dados são acessados;
- Como os dados são unidos;
- Sequência de operações;
- Uso de tabelas de trabalho temporárias e classificações;
- Contagens de linhas estimadas, iterações e custos de cada etapa;
- Contagens de linhas reais e iterações;
- Como os dados são agregados;
- Uso de paralelismo;
- Warnings de execução da query.

A análise do plano de execução da query, em termos de leitura, é feita da direita para a esquerda, de cima para baixo.

Figura 54 – Leitura do plano de query do SQL Server.



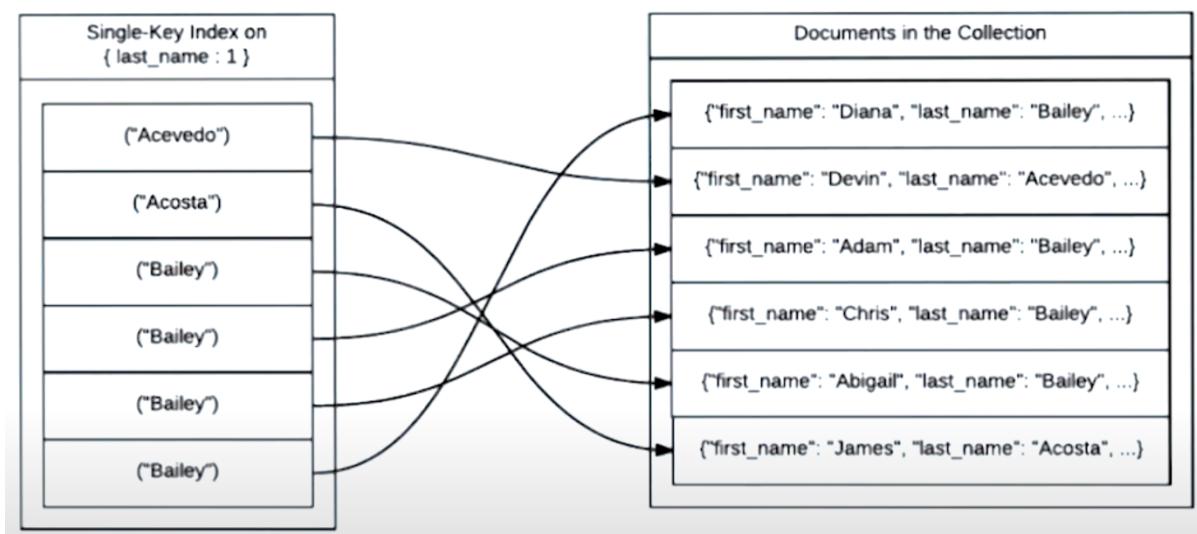
Para maiores detalhes de como analisar o plano de query, assista as aulas 6.3 (Análise do Plano de Execução de Query) e 6.4 (Demonstração: Análise do Plano de Execução de Query).

Capítulo 7. Otimização de Query no MongoDB

7.1. Tipos de Índices

Os índices são estruturas de dados especiais que armazenam uma pequena parte do conjunto de dados da collection, em uma forma fácil de se percorrer. O índice armazena o valor de um campo específico ou conjunto de campos, ordenado pelo valor do campo.

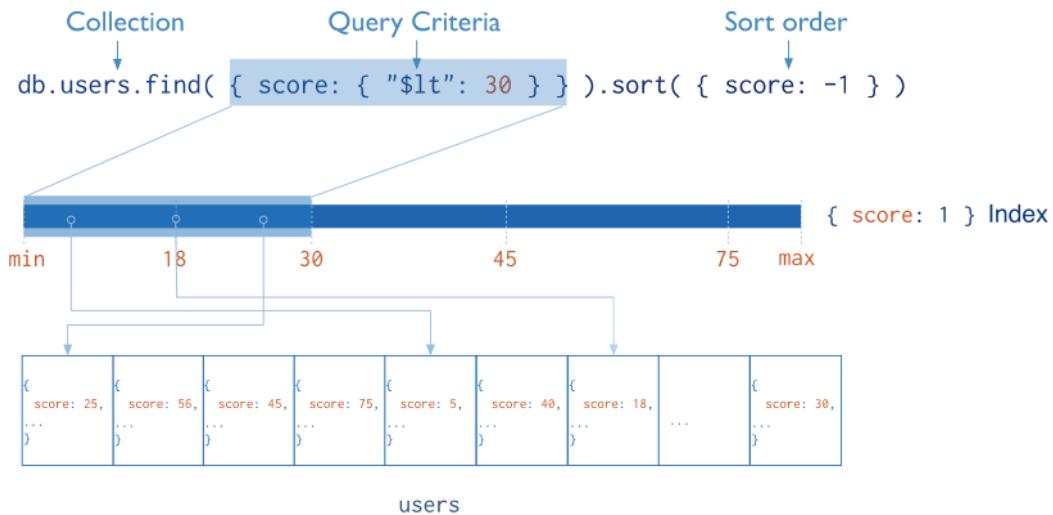
Figura 55 – Índices no MongoDB.



Se existir um índice apropriado para uma query, o MongoDB pode usar o índice para limitar o número de documentos que deve inspecionar.

O MongoDB oferece suporte a comparações de igualdade e operações de consulta baseadas em intervalo e pode retornar resultados classificados usando a ordenação no índice.

Figura 56 – Query no MongoDB usando índice.



Fonte: MongoDB (2020).

▪ Default Index

- Por default, o MongoDB cria um **índice único** no campo `_id`, durante a criação de uma collection;
- O **índice `_id`** evita que os clientes insiram dois documentos com o mesmo valor para o campo `_id`;
- Não é possível apagar esse índice.

Figura 57 – Default index do MongoDB.

```
> db.Equipamentos.insert({ "nome_equipamento": 'MAC0001', "data_insercao": new Date() });
WriteResult({ "nInserted" : 1 })
> db.Equipamentos.getIndexes();
[ { "v" : 2, "key" : { "_id" : 1 }, "name" : "_id_" } ]
```

Fonte: Gustavo (2020).

▪ Single Field Index

- Índice com apenas um campo;

- Pode ser ascendente ou descendente;
- Criado com o comando: **`db.nome_collection.createIndex ({ campo : N })`**
 - Onde **N** é a ordem de classificação do índice:
 - **1 → ascendente**
 - **-1 → descendente**

Figura 58 – Single Field Index do MongoDB.



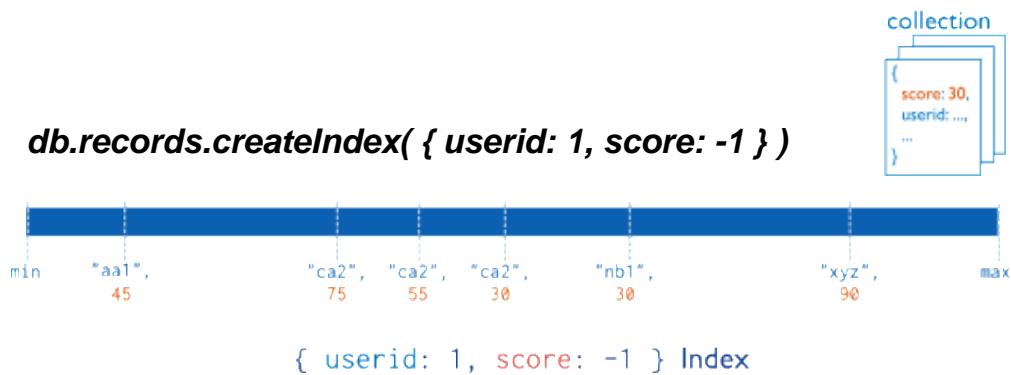
Fonte: Gustavo (2020).

- **Compound Index**

- Índice com vários campos;
- Cada campo pode ter sua ordem específica (ascendente ou descendente);
- Criado com o comando:
`db.nome_collection.createIndex ({ campo1 : N, campo2 : N... })`
- Onde **N** é a ordem de classificação do índice:
 - **1 → ascendente**

- -1 → descendente

Figura 59 – Compound Index do MongoDB.



Fonte: Gustavo (2020).

▪ Multikey Index

- Para indexar o conteúdo armazenado em campos do tipo **arrays**;
- Faz a comparação com base em um ou mais elementos do array;
- O MongoDB cria entradas de índice separadas para cada elemento do array;
- Criado com o comando:

`db.nome_collection.createIndex ({ campoarray : N})`

- Onde **N** é a ordem de classificação do índice:
 - 1 → ascendente
 - -1 → descendente

Figura 60 – Multikey Index do MongoDB.



Fonte: Gustavo (2020).

▪ Text Index

- Suporta a pesquisa de conteúdo de string em uma collection;
- Esses índices de texto não armazenam “stop words” específicas de um idioma (por exemplo, “o”, “a”, “ou”);
- Armazena apenas palavras raiz → **“Curso de Bootcamp do IGTI”**.
- Criado com o comando:

`db.nome_collection.createIndex ({ campoarray : "text" });`

- Pode ser simples ou composto.

▪ Partial Index

- Os índices parciais indexam apenas os documentos em uma collection que **atendem a uma expressão de filtro especificada**;
- Mais performático, pois realiza a busca em conjunto menor de dados (estrutura menor do índice);
- Gasta menos espaço de armazenamento;

- Menor custo na criação e manutenção.

Figura 61 – Partial Index do MongoDB.

```
db.restaurants.createIndex(  
  { cuisine: 1, name: 1 },  
  { partialFilterExpression: { rating: { $gt: 5 } } }  
)
```

Fonte: Gustavo (2020).

7.2. Plano de Execução de Query

Assim como no SQL Server, o MongoDB utiliza o plano de execução de query para indicar como cada query dever ser executada, quais os objetos que serão utilizados na query, os índices e mecanismos de acesso.

Para exibir o plano de execução de uma query no MongoDB, usa-se o método **explain()**. Olhando para todas as informações retornadas por este método, podemos descobrir coisas como:

- Quantos documentos foram “escaneados” (analisados);
- Quantos documentos foram retornados;
- Qual índice foi usado’
- Quanto tempo a consulta demorou para ser executada’
- Quais planos de execução alternativos foram avaliados.

Figura 62 – Plano de execução do MongoDB.

```
> db.people.find({ "ssn" : "720-38-5636" }).explain("executionStats")
{
    "queryPlanner" : {
        "plannerVersion" : 1,
        "namespace" : "test.people",
        "indexFilterSet" : false,
        "parsedQuery" : {
            "ssn" : {
                "$eq" : "720-38-5636"
            }
        },
        "winningPlan" : {
            "stage" : "COLLSCAN",
            "filter" : {
                "ssn" : {
                    "$eq" : "720-38-5636"
                }
            },
            "direction" : "forward"
        },
        "rejectedPlans" : [ ]
    },
    "executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 1,
        "executionTimeMillis" : 753,
        "totalKeysExamined" : 0,
        "totalDocsExamined" : 50474,
        "executionStages" : {
            "stage" : "COLLSCAN",
            "filter" : {
                "ssn" : {
                    "$eq" : "720-38-5636"
                }
            },
            "nReturned" : 1,
            "executionTimeMillisEstimate" : 725,
            "works" : 50476,
            "advanced" : 1,
            "needTime" : 50474,
            "needYield" : 0,
            "saveState" : 404,
            "restoreState" : 404,
            "isEOF" : 1,
            "invalidates" : 0,
            "direction" : "forward",
            "docsExamined" : 50474
        }
    },
    "serverInfo" : {
        "host" : "MacBook-Air-de-Gustavo.local",
        "port" : 27017,
        "version" : "4.0.10",
        "gitVersion" : "c389e7f69f637f7a1ac3cc9fae843b635f20b766"
    },
    "ok" : 1
}
```

Fonte: Gustavo (2020).

Para maiores detalhes de como analisar o plano de query, assista as aulas 6.3 (Análise do Plano de Execução de Query) e 6.4 (Demonstração: Análise do Plano de Execução de Query).

Capítulo 8. Operações Massivas de Dados

8.1. Extração e Carga Massiva de Dados no SQL Server

Para fazer extração e carga massiva de dados no SQL Server, podemos usar várias ferramentas e técnicas. Dentre as mais usadas, temos:

- **Utilitário BCP**

- Ferramenta de linha de comando (**bcp.exe**), para exportar e importar dados entre um banco de dados SQL Server e um arquivo de dados:
 - Exportar de uma tabela do SQL Server para um arquivo de dados;
 - Exportar para um arquivo de dados com base em uma consulta;
 - Importar dados de um arquivo de dados para uma tabela do SQL Server.
- Exportar dados de uma tabela para um arquivo, usando autenticação integrada:

```
bcp AdventureWorks2012.Sales.Currency
out "Currency Types.dat" -T -c
```

- Exportar dados de uma query para um arquivo:

```
bcp "SELECT FullName, PreferredName
FROM WideWorldImporters.dbo.People
ORDER BY FullName" queryout D:\BCP\People.txt -t, -c -T
```

- Importar dados de um arquivo para uma tabela:

```
bcp BDTeste.dbo.Tabela_A_Ser_Carregada IN
D:\BCP\StockItemTransactions_character.bcp -S
Nome_Servidor -c -T
```

- Bulk Insert

- Operação de carga de dados, possível de ser minimamente logada*:
 - Menos espaço gasto e contenção no transaction log;
 - Mais otimizada por não registrar cada insert como uma transação*.

*OBS.: necessários atender alguns pré-requisitos, que podem ser encontrados na documentação disponibilizada pelo fornecedor em <https://docs.microsoft.com/pt-br/sql/relational-databases/import-export/prerequisites-for-minimal-logging-in-bulk-import?view=sql-server-ver15>.

- SQL Server Integration Services (SSIS)

- O SQL Server Integration Services é uma plataforma para construir soluções de integração e transformação de dados, podendo ser usado para copiar e extrair dados;

Figura 63 – Integration Services.

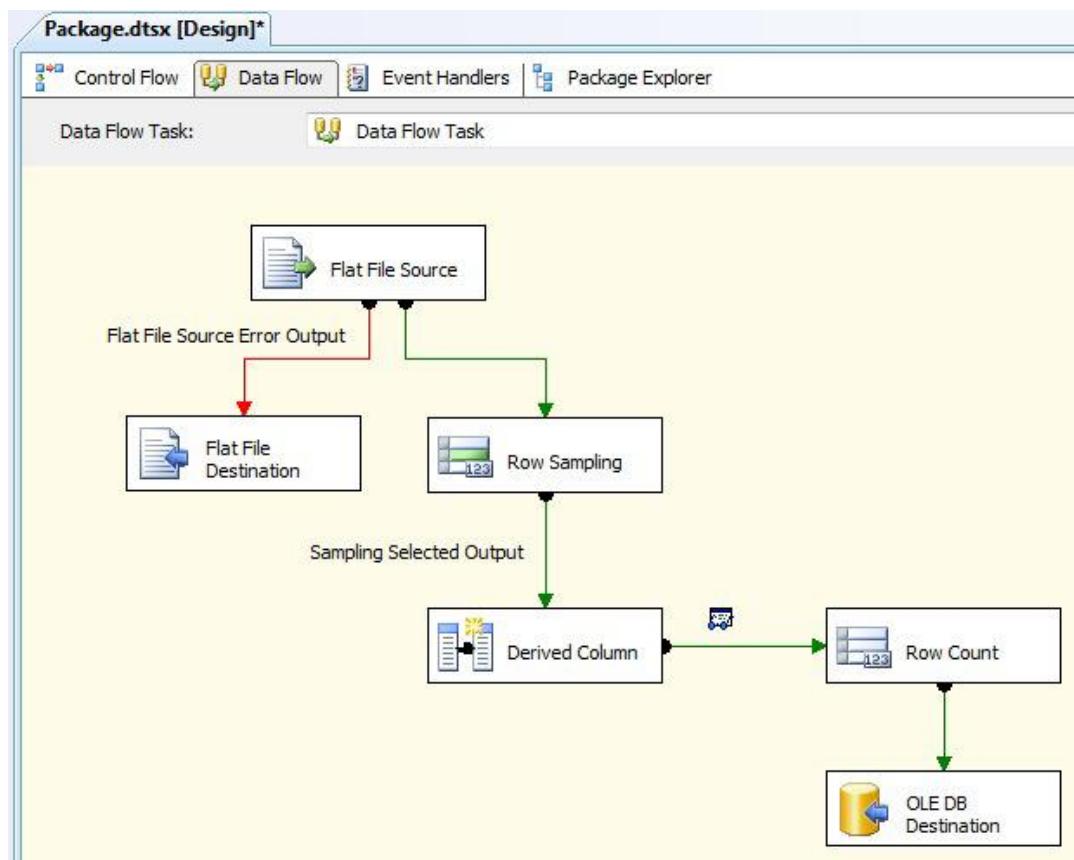


- O Integration Services pode extrair e transformar dados de uma ampla variedade de fontes, como arquivos de dados XML, arquivos simples e

fontes de dados relacionais e, em seguida, carregar os dados em um ou mais destinos.

- Trabalha com o conceito de pacotes, sendo que cada pacote contém um conjunto de componentes (tarefas).

Figura 64 – Pacote do Integration Services.



Fonte: Gustavo (2020).

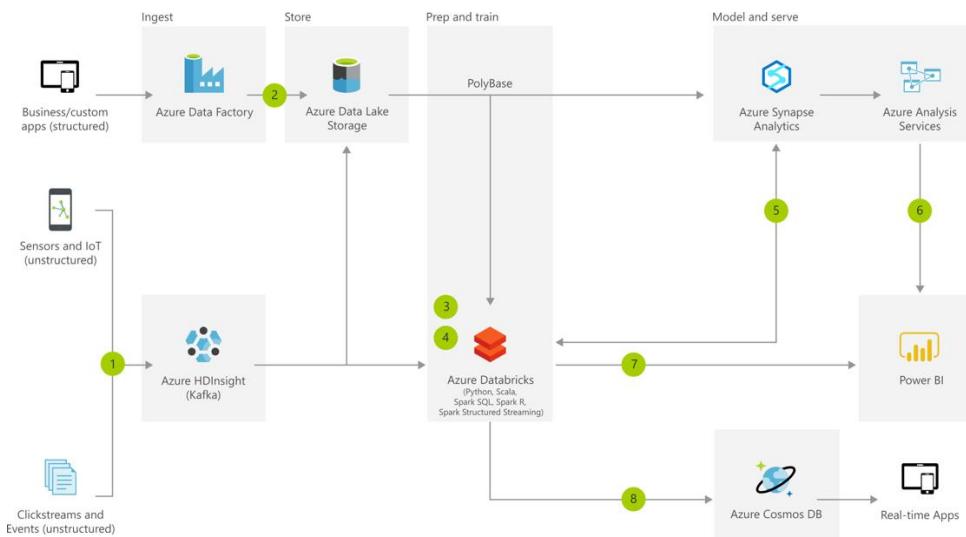
▪ **Azure Data Factory**

Azure Data Factory é o serviço de integração de dados e ETL (Extração, Transformação e Carga), baseado em nuvem, que permite criar fluxos de trabalho orientados a dados para orquestrar a movimentação de dados e transformá-los em escala. Usando o Azure Data Factory, é possível criar e agendar fluxos de trabalho baseados em dados (chamados de pipelines), que podem ingerir dados de diversas fontes de armazenamentos.

Figura 65 – Pipeline simples com o Data Factory.

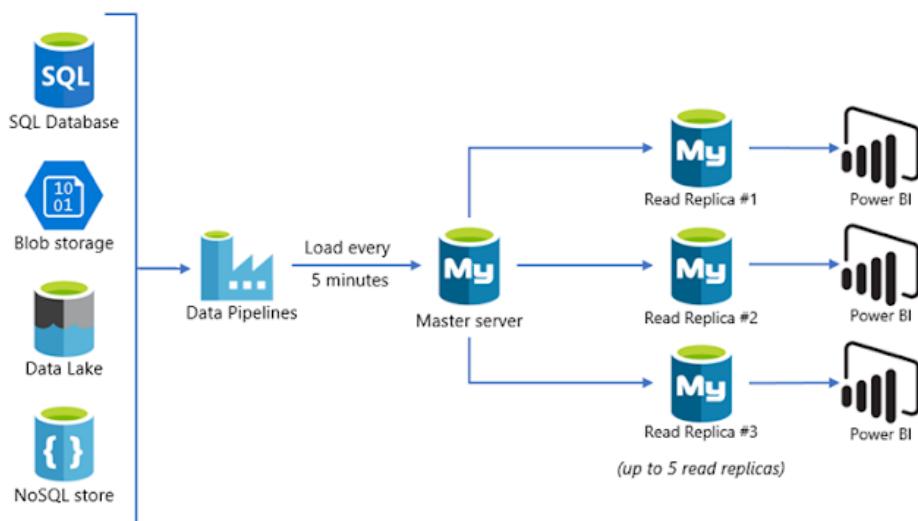
Fonte: Microsoft (2019).

Com o Data Factory, é possível também criar processos ETL complexos que transformam dados com fluxos de dados ou usando serviços de computação, como Azure HDInsight Hadoop, Azure Databricks e Azure Synapse Analytics.

Figura 66 – Pipeline do Data Factory com HDInsight, Databricks e SQL DW.

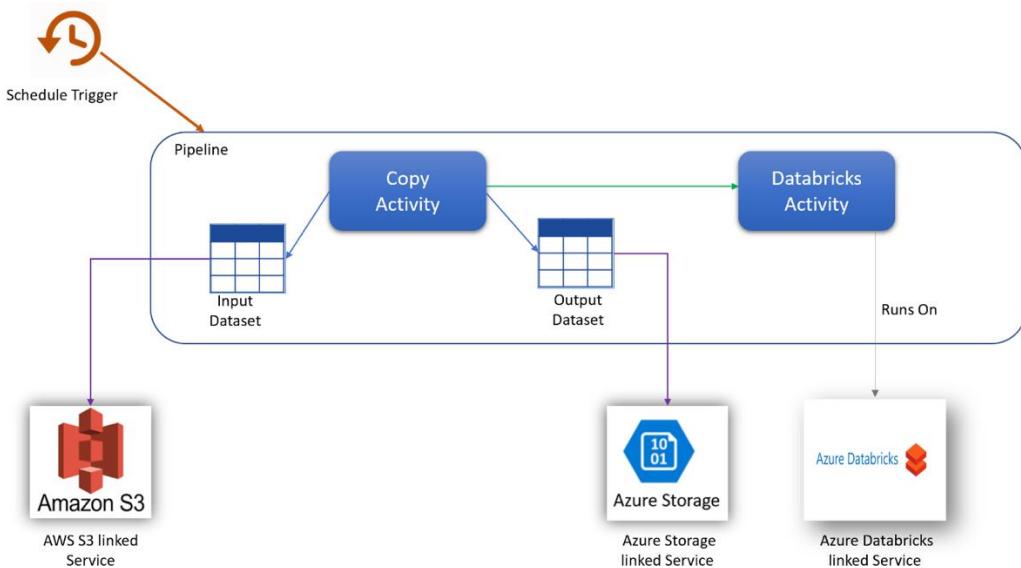
Fonte: Microsoft (2019).

Além dos pipelines executados manualmente, é possível também criar e agendar fluxos de trabalho orientados a dados para orquestrar a movimentação de dados e transformá-los.

Figura 67 – Pipeline Agendado (Triggered Pipeline).

Fonte: Microsoft (2019).

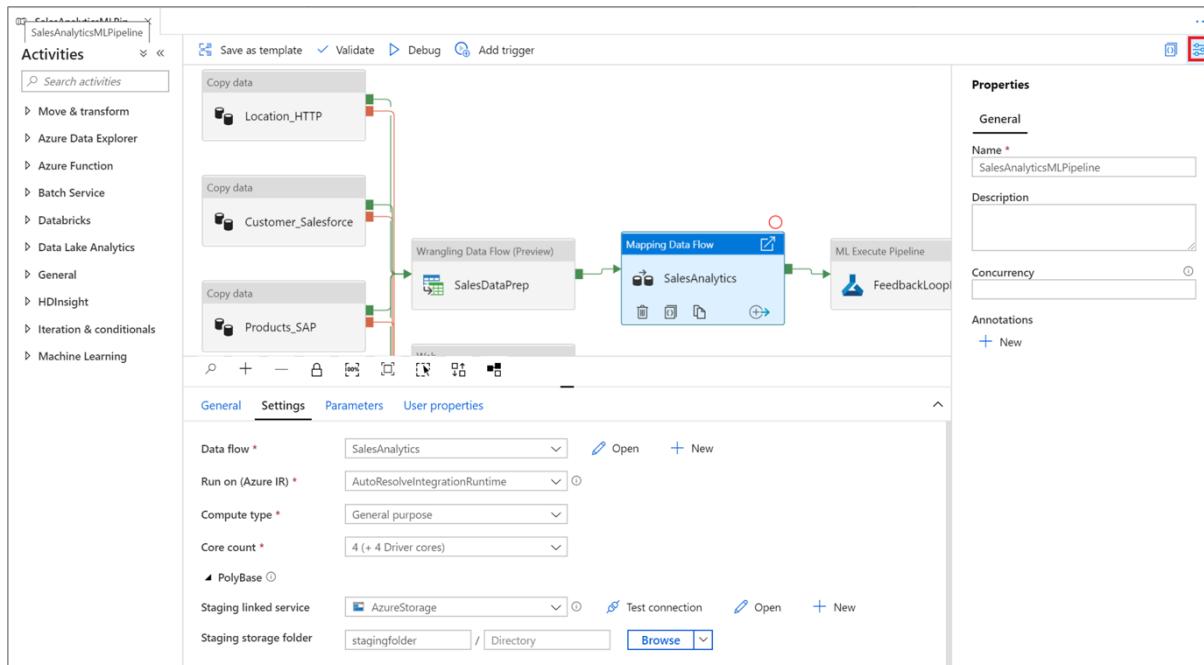
Importante frisar que as fontes de dados do Data Factory podem ser externas ao Azure, como no exemplo abaixo, onde os dados são extraídos de um bucket S3 na AWS.

Figura 68 – Fonte de dados externa ao Azure Data Factory.

Fonte: Microsoft (2019).

O ponto auge do Azure Data Factory é permitir a criação de pipelines de forma gráfica, além da forma via linha de comando (código).

Figura 69 – Interface do Data Factory para criação de pipelines graficamente.



Fonte: Microsoft (2019).

Componentes do Azure Data Factory

▪ Atividade:

- Representa uma etapa de processamento em um pipeline.
 - Atividade para copiar dados de um repositório de dados para outro;
 - Atividade que executa uma consulta de Hive em um cluster do Azure HDInsight para transformar ou analisar dados;
 - Etc.
- O Data Factory dá suporte a três tipos de atividades:

- Atividades de movimentação de dados;
- Atividades de transformação de dados;
- Atividades de controle.

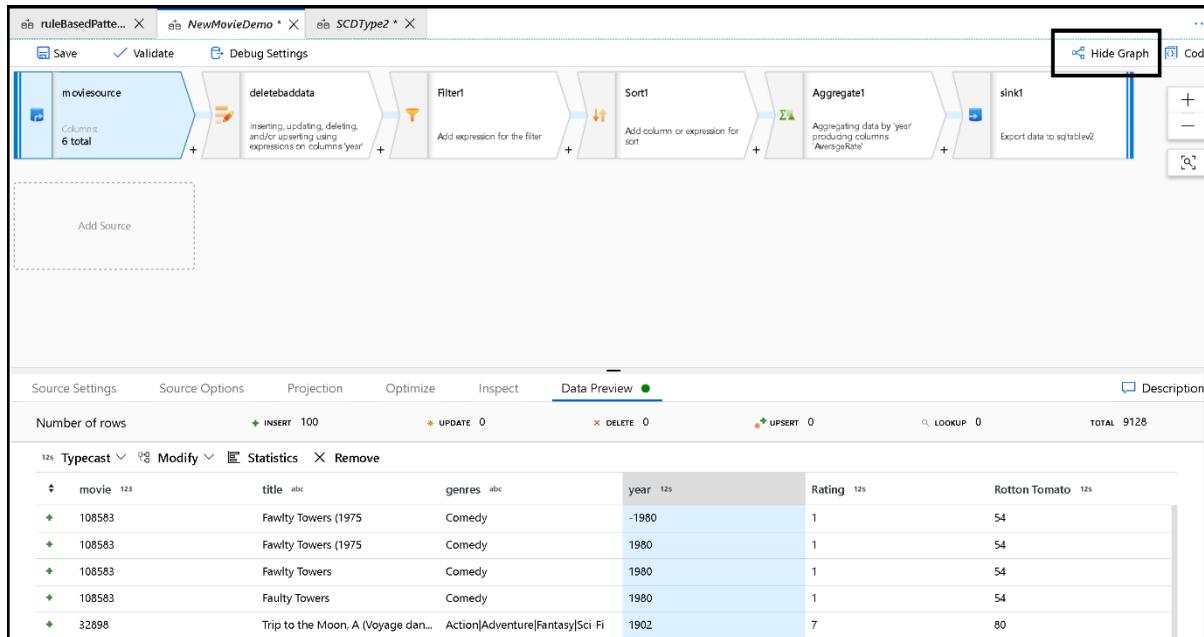
▪ **Pipeline:**

- Agrupamento lógico de atividades que realiza uma unidade de trabalho.
Juntas, as atividades em um pipeline executam uma tarefa.
- Exemplo: pipeline contém um grupo de atividades que ingere dados provenientes de um blob do Azure e, em seguida, executa uma consulta Hive em um cluster HDInsight para particionar os dados.
- Pipeline permite gerenciar atividades como um conjunto, em vez de gerenciar cada uma individualmente.
- Atividades podem operar de modo sequencial ou de forma independente, em paralelo.

▪ **Mapeamento de Fluxo de Dados (Mapping Data Flow):**

- São transformações de dados visualmente projetadas no Azure Data Factory.
- Permitem que os engenheiros de dados desenvolvam lógicas de transformação de dados sem escrever código.
- O Data Factory executará a lógica em um cluster Spark, autogerenciado pelo Azure, que será ativado e desativado quando necessário.

Figura 70 – Azure Data Factory Mapping Data Flow.



Fonte: Microsoft (2019).

▪ **Conjunto de Dados (Dataset):**

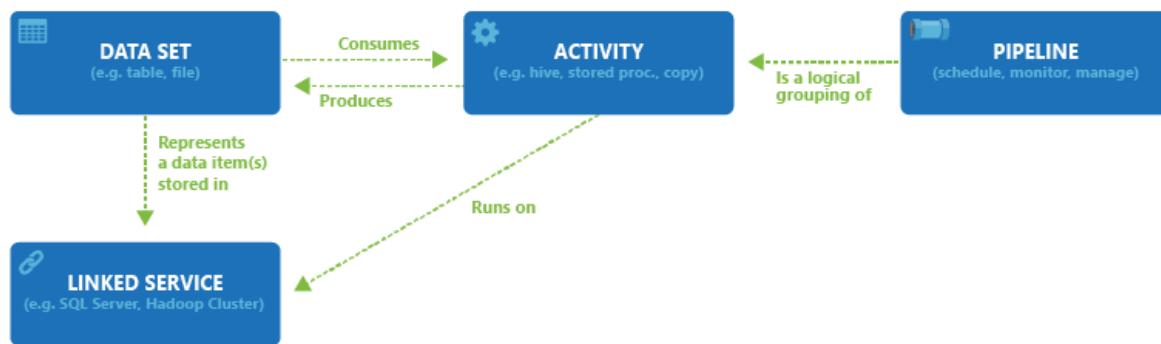
- Representam as estruturas de dados nos repositórios de dados, que simplesmente apontam para, ou fazem referência, aos dados que se deseja usar em atividades, seja como entrada ou saída.

▪ **Serviço Vinculado (Linked Service):**

- Define as informações de conexão necessárias para que o Data Factory se conecte aos recursos externos. Duas finalidades:
 - Para representar um armazenamento de dados: ex. banco SQL/Oracle;
 - Para representar um recurso de computação que pode hospedar a execução de uma atividade: ex. um cluster Hadoop do HDInsight, onde a atividade HDInsightHive é executada.

Resumindo, um serviço vinculado define a conexão à fonte de dados e um conjunto de dados representa a estrutura dos dados. Por exemplo, um serviço vinculado de armazenamento do azure especifica a string de conexão para conectar-se à conta de Armazenamento do Azure (Storage Account), e um conjunto de dados de blob do Azure especifica o contêiner de blob e a pasta que contém os dados.

Figura 71 – Componentes do Azure Data Factory.



Fonte: Microsoft (2019).

8.2. Extração e Carga Massiva de Dados no MongoDB

Para fazer extração e carga massiva de dados no MongoDB, podemos usar várias ferramentas e técnicas. Dentre as mais usadas, temos:

- **Bulk Write**

```
db.collection.bulkWrite(  
  [  
    { insertOne : <document> },  
    { updateOne : <document> },  
    { updateMany : <document> },  
    { replaceOne : <document> },  
    { deleteOne : <document> },  
    { deleteMany : <document> }  
  ]  
)
```

```
try {
    db.characters.bulkWrite([
        { insertOne: { "document": { "_id": 4, "char": "Dithras", "class": "barbarian", "lvl": 4 } } },
        { insertOne: { "document": { "_id": 4, "char": "Taeln", "class": "fighter", "lvl": 3 } } },
        { updateOne : {
            "filter" : { "char" : "Eldon" },
            "update" : { $set : { "status" : "Critical Injury" } }
        }},
        { replaceOne : {
            "filter" : { "char" : "Meldane" },
            "replacement" : { "char" : "Tany", "class" : "oracle", "lvl": 4 }
        }}
    ], { ordered : false } );
} catch (e) {
    print(e);
}
```

- **mongoimport**

- Ferramenta de linha de comando para importar dados de arquivos em formatos JSON, CSV ou TSV para collections em um banco de dados MongoDB.

mongoimport --db = users --collection = contacts --file = contacts.json

- **mongoexport**

- Ferramenta de linha de comando para exportar dados de collections para arquivos nos formatos JSON ou CSV.

mongoexport --collection = events --db = reporting --out = events.json

- **mongodump**

- Ferramenta de linha de comando para gerar uma exportação binária do conteúdo de um banco de dados MongoDB.

mongodump --archive = test.20150715.gz --gzip --db = test

- **mongorestore**

- Ferramenta de linha de comando para carregar dados de um dump binário de banco de dados MongoDB criado com o mongodump.

mongorestore --db=reporting dump/test/salaries.bson

8.3. Expurgo Massivo de Dados no SQL Server

Para fazer expurgo massivo de dados no SQL Server, podemos usar várias ferramentas e técnicas. Dentre as mais usadas, temos:

- **Switch Table**

- Mover dados de uma tabela para outra de forma rápida;
- Remove dados da origem, arquivando os dados no destino.

Figura 72 – Switch Table

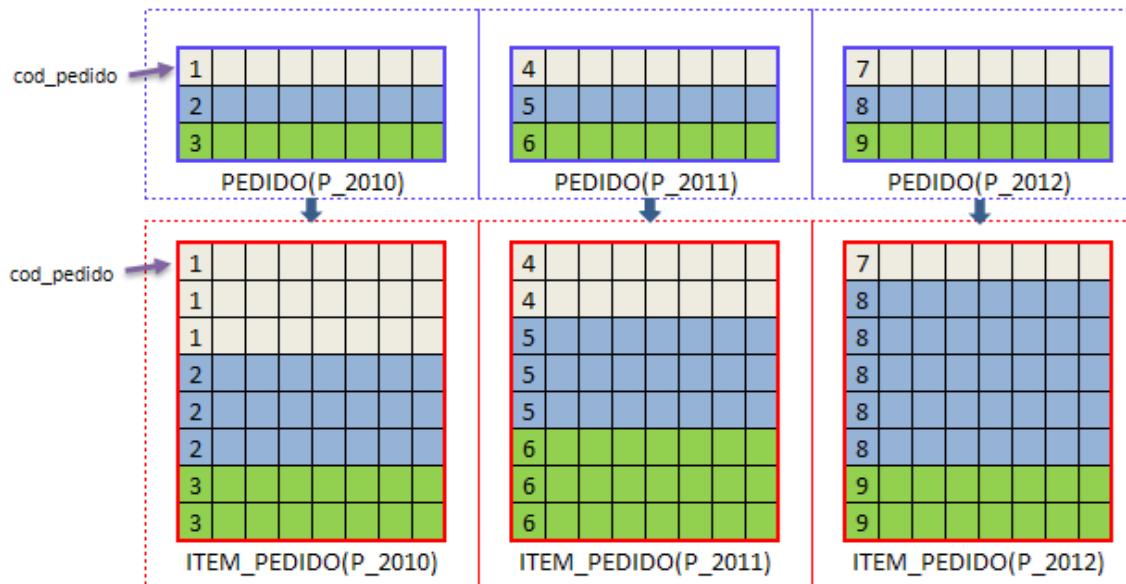


Fonte: Gustavo (2020).

- **Tabela Particionada**

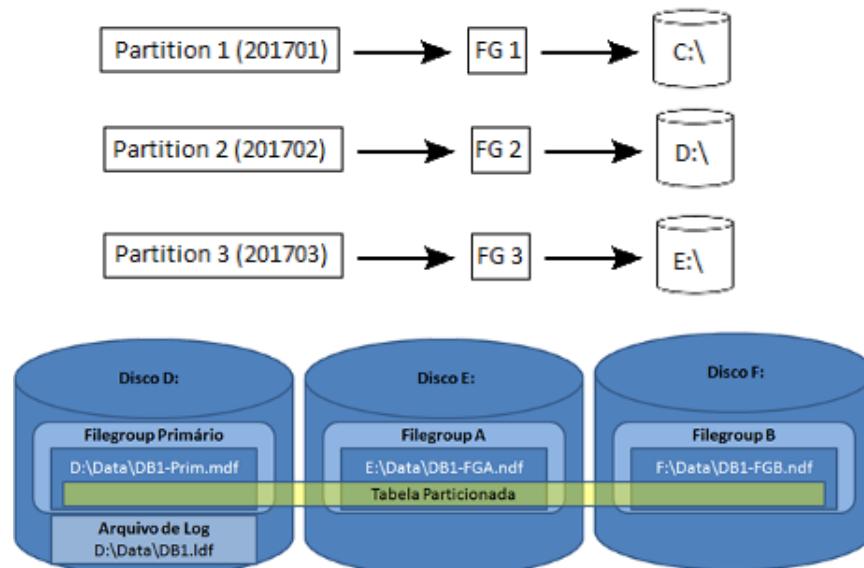
- Recurso que permite a distribuição dos dados de tabelas e índices em vários grupos, separados fisicamente.

Figura 73 – Tabela Particionada.



- Essa distribuição pode ser feita para discos físicos distintos.

Figura 74 – Tabela particionada separada em discos.



Para criar uma tabela particionada, são necessários os seguintes recursos/etapas:

- **Função de particionamento:** define como as linhas são distribuídas para um conjunto de partições, com base nos valores de certas colunas, chamadas de colunas de particionamento.

Figura 75 – Resultado de uma função de particionamento.

2012-01-01
2012-12-31
2013-01-01
2013-12-31
2014-01-01
2014-12-31
2015-01-01
2015-12-31

Ex.: criar uma função de particionamento de nome FP_Particiona_Tabela que irá particionar a tabela em 4 partições:

```
CREATE PARTITION FUNCTION FP_Particiona_Tabela (int)
AS RANGE LEFT FOR VALUES (1, 100, 1000);
```

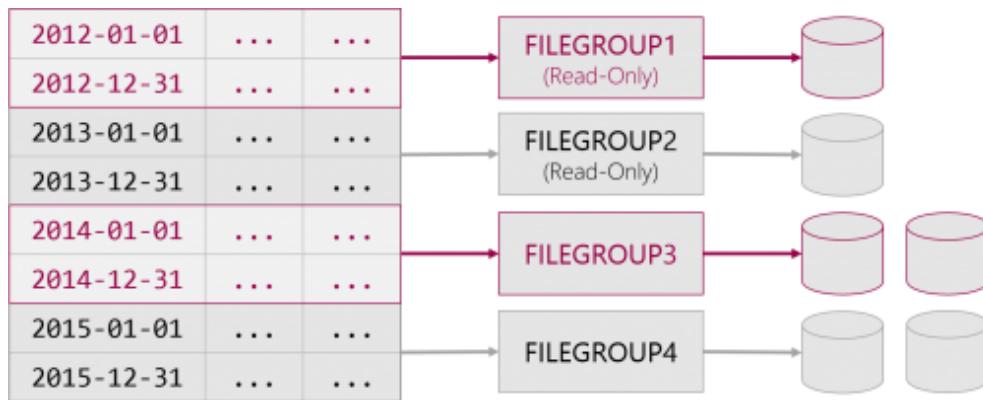
Figura 76 – Exemplo de janelas de dados criadas com função de particionamento.

Partition	1	2	3	4
Valores	col1 <= 1	col1 > 1 AND col1 <= 100	col1 > 100 AND col1 <= 1000	col1 > 1000

Fonte: Gustavo (2020).

- **Esquema de particionamento:** mapeia cada partição especificada pela função de particionamento para um filegroup.

Figura 77 – Exemplo de esquemas de particionamento.



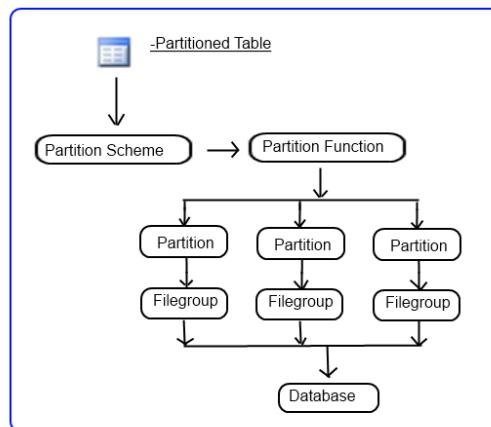
Ex.: criar o esquema de partição chamado PS_Particoes que direciona as partições para os filegroups específicos:

```
CREATE PARTITION SCHEME PS_Particoes
AS PARTITION FP_Particiona_Tabela
TO (Filegroup1, Filegroup2, Filegroup3, Filegroup4);
```

- Criar a **tabela particionada** apontando para o esquema de particionamento.

```
CREATE TABLE TabelaParticionada (col1 int PRIMARY KEY, col2
char(10))
ON PS_Particoes (col1);
```

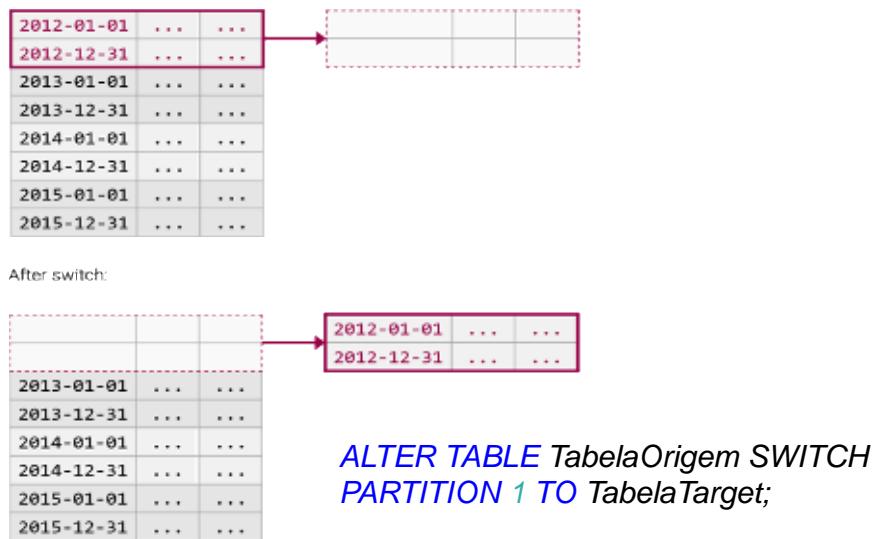
Figura 78 – Tabela particionada.



Com a tabela particionada, podemos expurgar dados da mesma, de forma muito rápida, usando-se as seguintes estratégias:

- Arquivar uma partição para uma tabela não particionada.

Figura 79 – Switch partition.



- Expurgar uma partição com truncate (*somente a partir do SQL 2016).

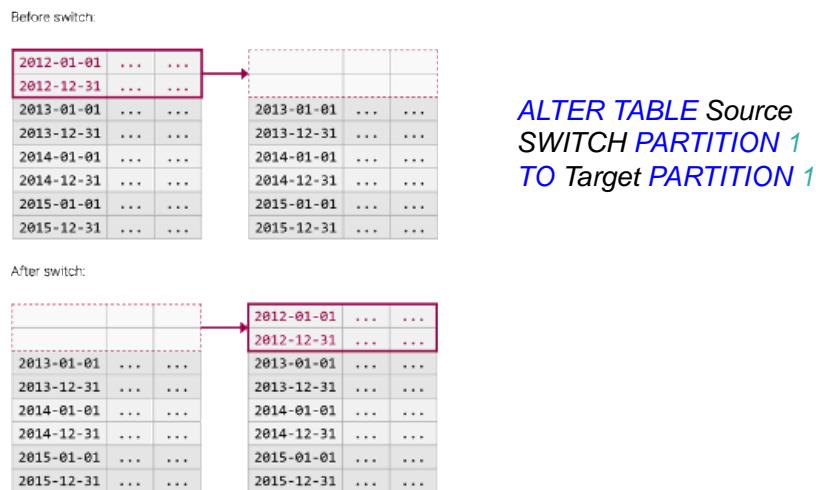
TRUNCATE TABLE TabelaParticionada

WITH (PARTITIONS (1, 2, 6 TO 8))

GO

- Arquivar uma partição para uma partição de outra tabela particionada.

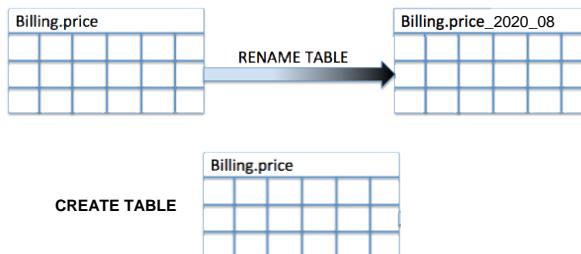
Figura 80 – Switch partition entre tabelas particionadas.



▪ **Shift Table:**

- Arquivar dados renomeando a tabela e criando a tabela novamente, em branco (chamado também de *soft delete*, onde os dados são excluídos apenas logicamente).

Figura 81 – Shift table.



8.4. Expurgo Massivo de Dados no MongoDB

Para fazer expurgo massivo de dados no SQL Server, podemos usar várias ferramentas e técnicas. Dentre as mais usadas, temos:

- Bulk Write:

```
db.collection.bulkWrite(  
  [  
    { insertOne : <document> },  
    { updateOne : <document> },  
    { updateMany : <document> },  
    { replaceOne : <document> },  
    { deleteOne : <document> },  
    { deleteMany : <document> }  
  ]  
)
```

```
try {  
  db.characters.bulkWrite([  
    { insertOne: { "document": { "_id": 4, "char": "Dithras", "class": "barbarian", "lvl": 4 } } },  
    { insertOne: { "document": { "_id": 5, "char": "Taeln", "class": "fighter", "lvl": 3 } } },  
    { updateOne : {  
      "filter" : { "char" : "Eldon" },  
      "update" : { $set : { "status" : "Critical Injury" } }  
    } },  
    { deleteOne : { "filter" : { "char" : "Brisbane" } } },  
    { deleteMany : { "filter" : { "encounter": { $lt : 0 } } } },  
    { replaceOne : {  
      "filter" : { "char" : "Meldane" },  
      "replacement" : { "char" : "Tanys", "class" : "oracle", "lvl": 4 }  
    } }  
  ]);  
} catch (e) {  
  print(e);  
}
```

- Índices TTL:

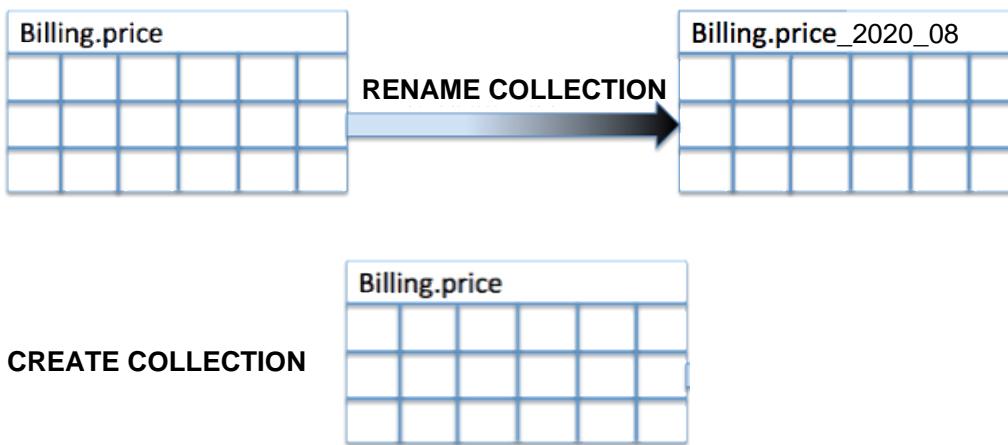
- Dados são “deletados” (expirados) logicamente;
- Deletados fisicamente pelo MongoDB, em background, sem sobrecarregar o ambiente.

```
db.eventlog.createIndex  
(  
  { "lastModifiedDate": 1 },  
  { expireAfterSeconds: 3600 }  
)
```

- Shift Collection:

- Arquivar dados renomeando a collection e criando a collection novamente, em branco (chamado também de *soft delete*, onde os dados são excluídos apenas logicamente).

Figura 82 – Shift collection.



Referências

COULOURIS, G.; DOLLIMORE, J. & KINDBERG, T. *Distributed Systems: concepts and design*. Éssex, Reino Unido: Addison Wesley, 1996.

KORTH, Henry F. *Sistema de bancos de dados*. 3. ed. São Paulo: McGraw-Hill, 1999.

MICROSOFT SQL Documentation. Disponível em <<https://docs.microsoft.com/en-us/sql>>. Acesso em: 24 fev. 2021.

MONGODB Documentation. Disponível em <<https://docs.mongodb.com/manual>>. Acesso em: 24 fev. 2021.

NAVATHE, Shamkant B.; ELSMARI, Ramez. *Sistemas de Banco de Dados*. 4. ed. São Paulo: Pearson Addison Wesley, 2005.

NOSQL Database Org. Disponível em: <<http://nosql-database.org/>>. Acesso em: 24 fev. 2021.

TANENBAUM, Andrew S.; STEEN, Maarten V. *Sistemas Distribuídos – Princípios e Paradigmas*. 2. ed. Pearson – Prentice Hall, 2007.