INSTITUTO DE GESTÃO E TECNOLOGIA DA INFORMAÇÃO

Performance e Otimização

Bootcamp Online: Analista de Banco de Dados

Gustavo Aguilar

2020

Performance e Otimização

Gustavo Aguilar

Sumário

Capítulo 1. Bancos de Dados Distribuídos

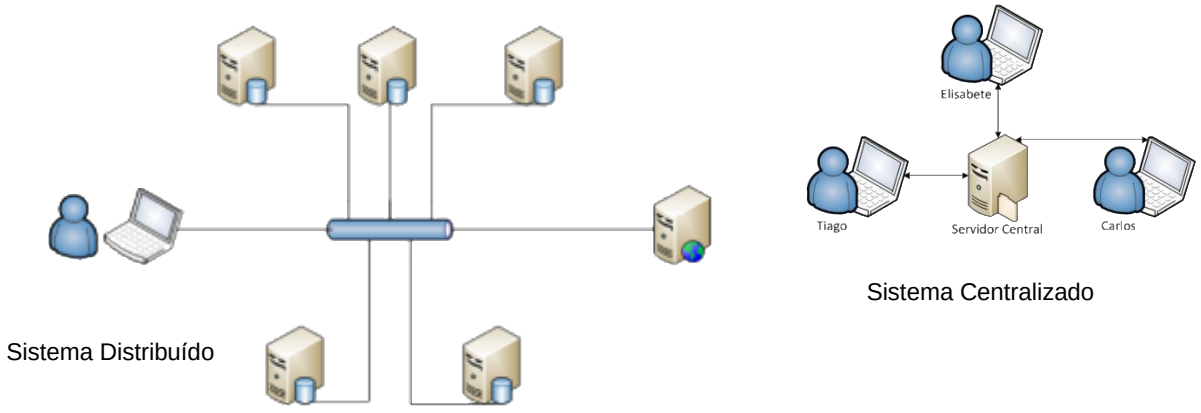
Quando falamos de performance e otimização de banco de dados, não podemos deixar de falar de banco de dados distribuídos, pois suas características de escalabilidade, replicação e particionamento de dados contribuem diretamente para maximizar a performance em um sistema de banco de dados.

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.

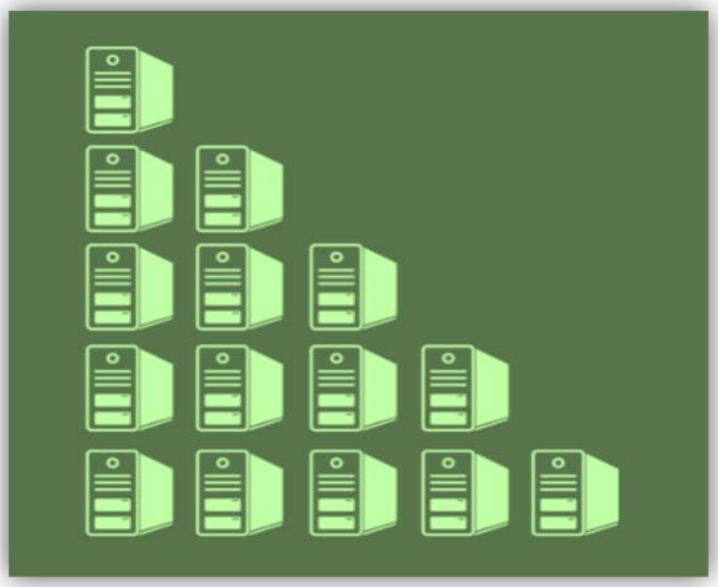
- **ESCALABILIDADE:** característica que permite aumentar o poder de processamento de um sistema. Em sistemas distribuídos, indica o aumento da quantidade de hardware ou recurso (CPU, RAM, disco, rede, etc.) necessário que possibilite aumentar o poder de processamento, sem interferir na aplicação e nos usuários. Difere-se da **elasticidade,** que é capacidade de um ambiente escalar (aumentar) e depois "desescalar" (reduzir), mas estas duas características estão intrinsicamente 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.

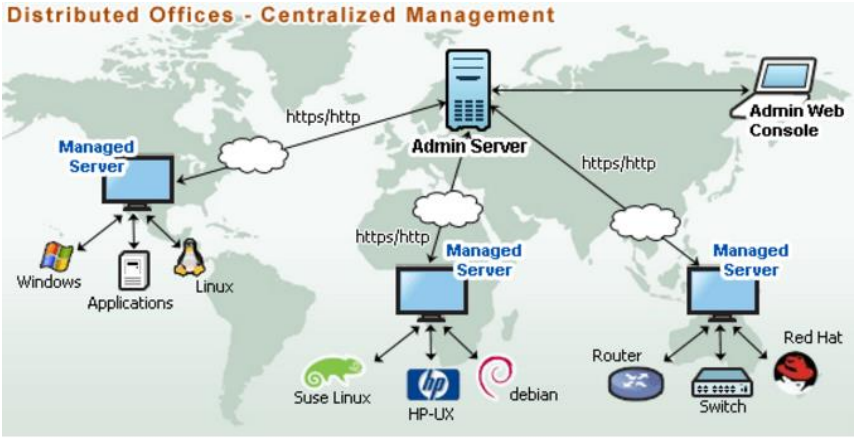


- **TRANSPARÊNCIA:** com essa característica, o objetivo é ocultar do usuário final ou do desenvolvedor, a separação dos componentes em um sistema distribuído, de modo que o sistema seja percebido como um todo, em vez de como uma coleção de componentes independentes e isolados.

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.

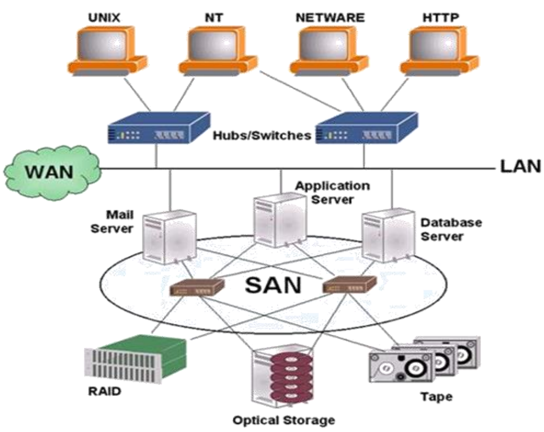
– **De Replicação:** esconde o fato de que múltiplas cópias do mesmo recurso podem estar disponíveis no sistema, permitindo que várias instâncias de recursos sejam usadas para aumentar a confiabilidade e o desempenho. Deve mascarar o conhecimento das réplicas por parte dos usuários, e implica na transparência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

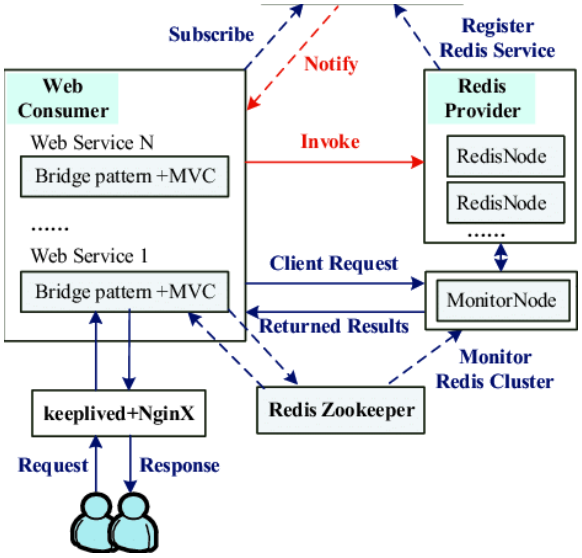
– **De Concorrência:** qualquer recurso compartilhado em um sistema distribuído deve ter garantido o correto funcionamento em um ambiente concorrente com acessos simultâneos.

– **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.

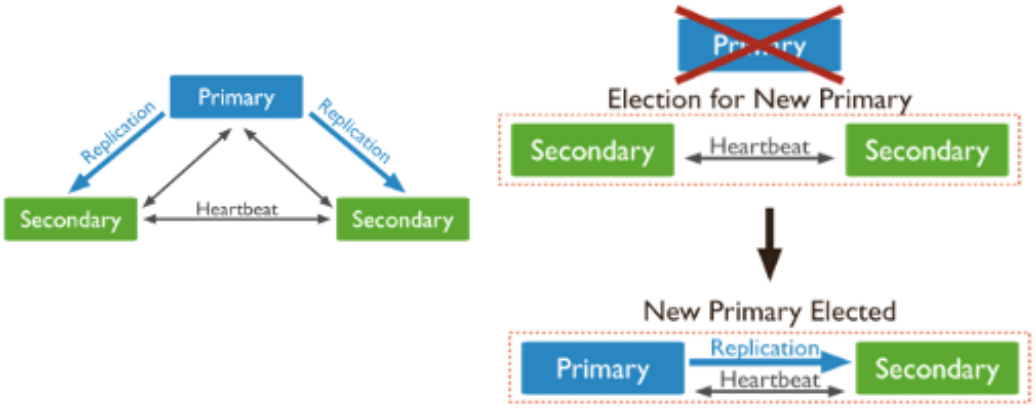
Figura 10 – Sist. Distrib. com transp. de segurança.



- **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.

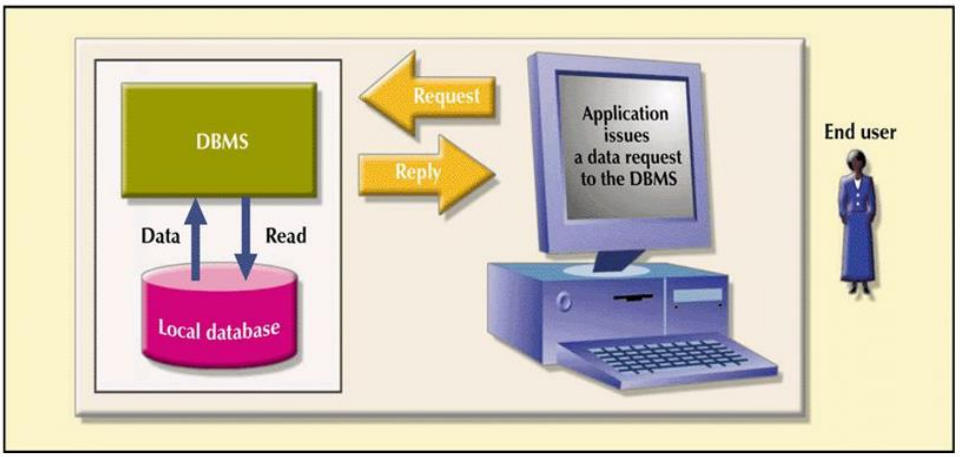
☑ **Independência de Replicação:** dados podem estar replicados em vários nós da rede, sem interferir na visão única do usuário.

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.

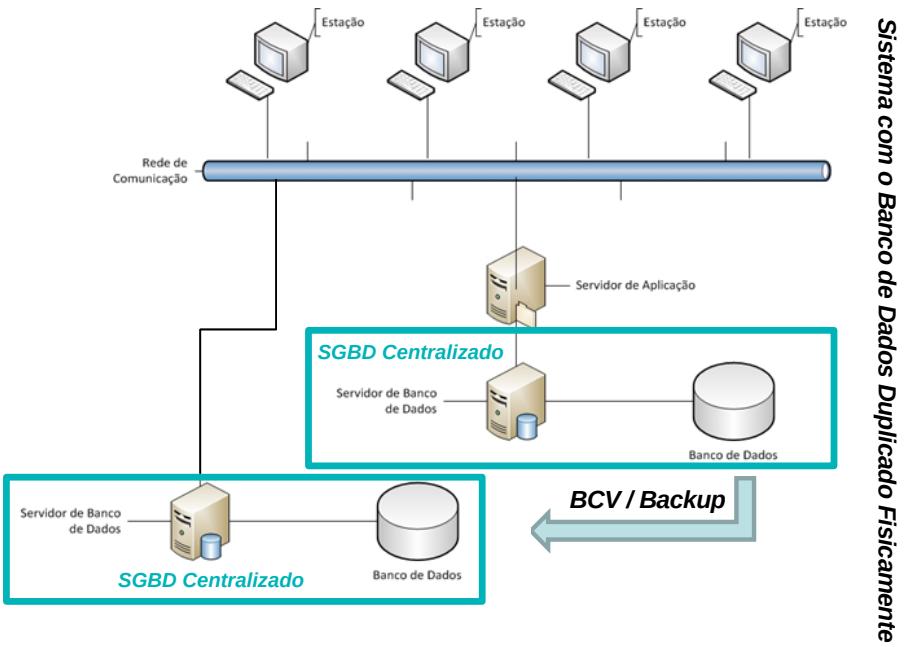
Diferentemente dessas características, um SGBD centralizado possui um front-end único, com gerenciamento de dados centralizado e processamento centralizado de consultas e transações, o que acarreta em ponto único de falha. Pode ser formado por componentes físicos únicos ou compartilhados (Cluster), mas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

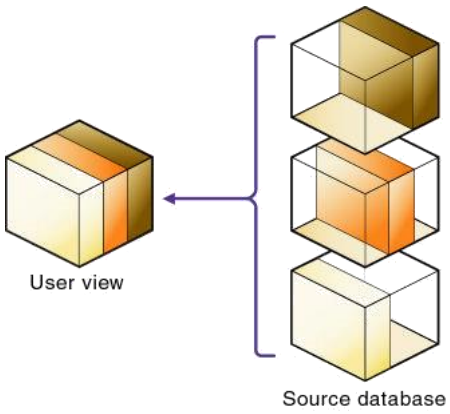
- ✓ **Replicação de Dados.**
- ✓ **Particionamento de Dados.**

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 usa 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.



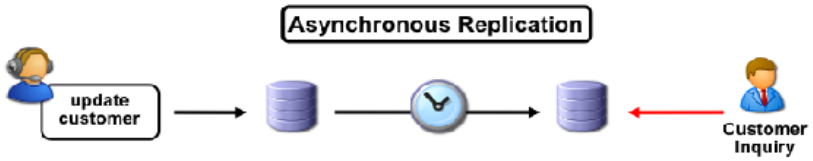
1.4. Tipos de Replicação de Dados

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.

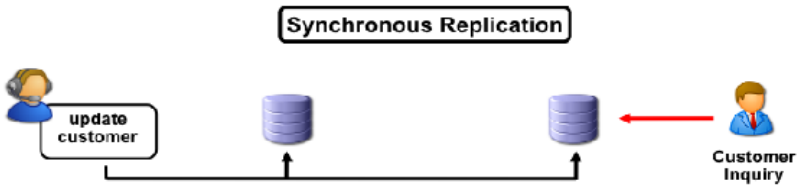
Figura 18 – Replicação Assíncrona.



- **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.

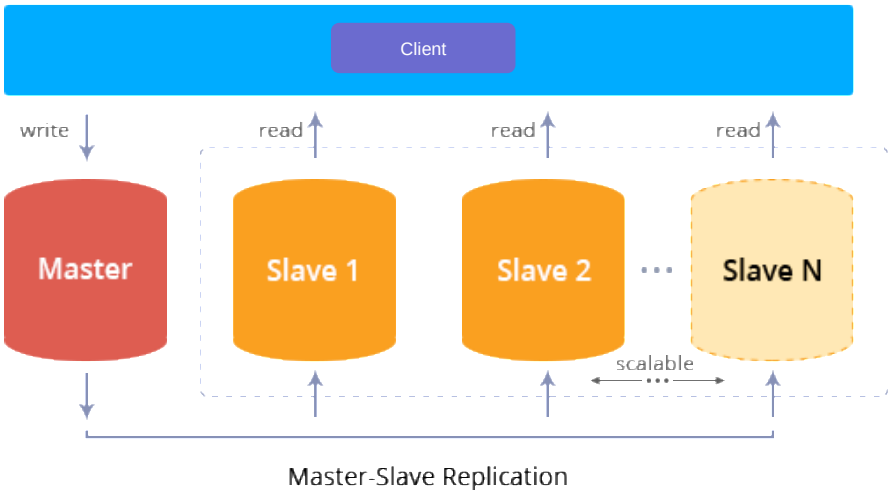
Apesar de ser menos performática para o usuário, ela oferece consistência total dos dados, em todos os pontos do SGBDD, ou seja, em qualquer momento, o usuário enxerga o mesmo conjunto de dados em qualquer um dos nós.

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.

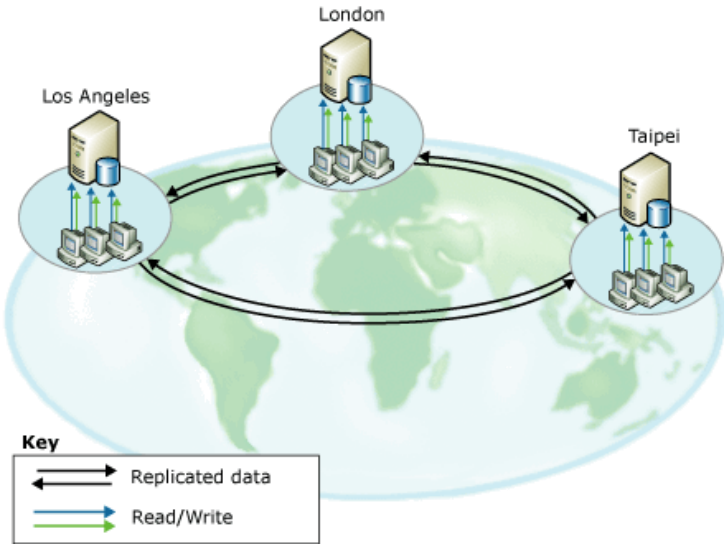
Figura 20 – Exemplo de replicação hierárquica.



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.



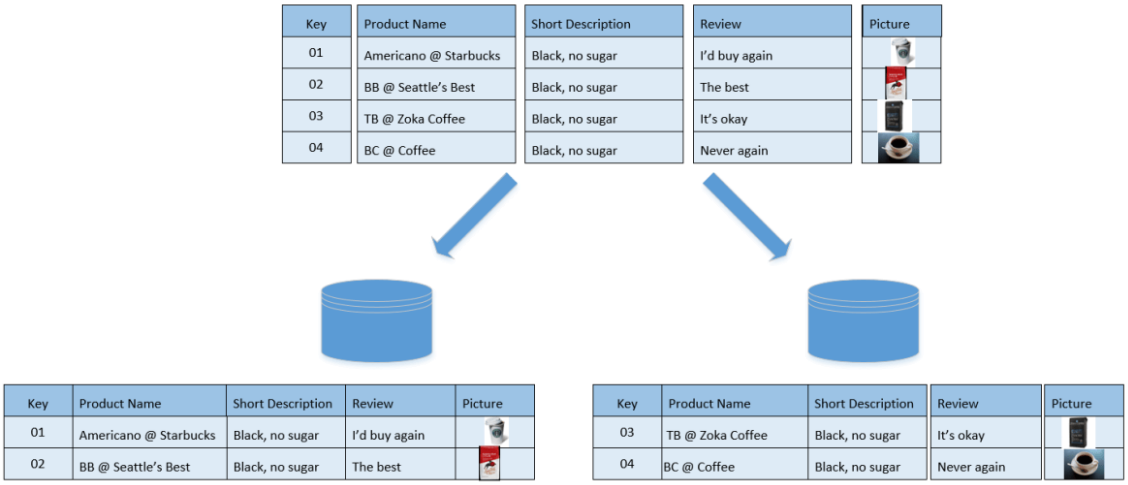
1.5. Tipos de Particionamento de Dados

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.

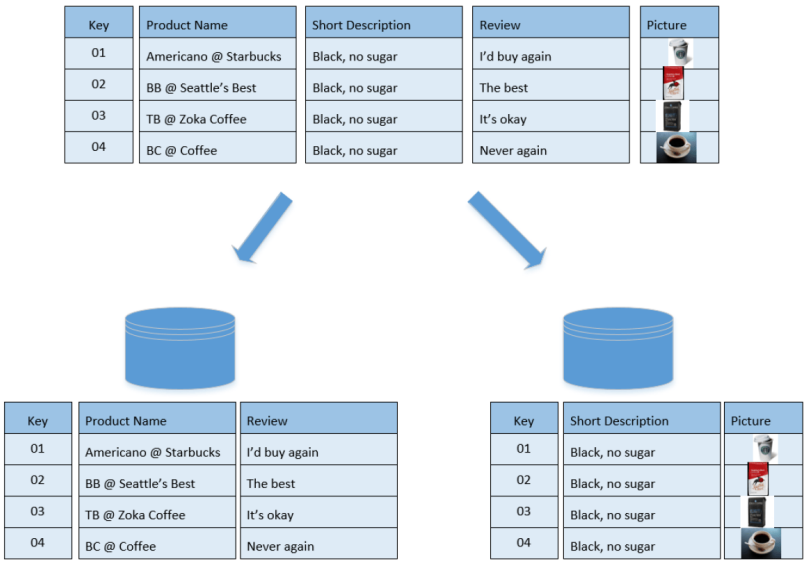
Muito utilizado em conjunto com a distribuição geográfica dos dados.

Figura 22 – Particionamento horizontal de dados.



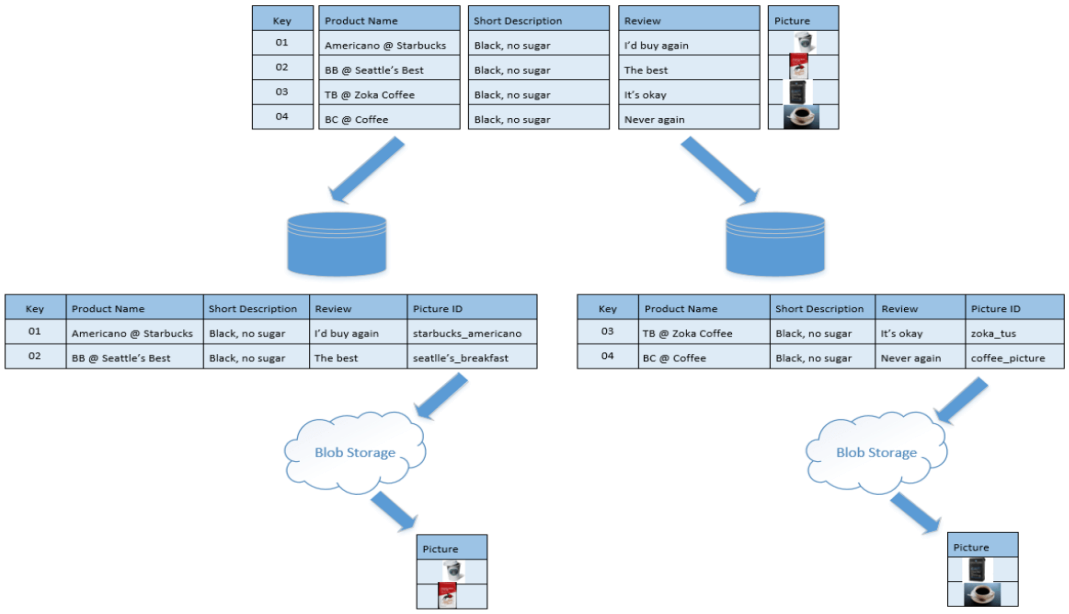
- **PARTICIONAMENTO VERTICAL:** cada fragmento (shard) consiste em um subconjunto de atributos (colunas), definidos pela operação de projeção (select), com base em na afinidade dos atributos, confidencialidade dos dados, armazenamento ou localização.

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 (*C*onsistency, *A*vailability, *P*artition 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** (***B****asically* ***A****vailable*, ***S****oft state,* ***E****ventual consistency*).

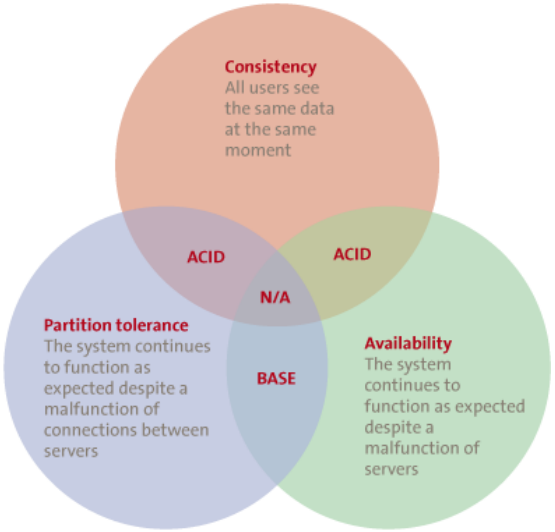
- **Basicamente Disponível** (*Basically Available*): essa restrição afirma que o sistema garante a disponibilidade dos dados no que diz respeito ao Teorema de CAP, ou seja, haverá uma resposta a qualquer solicitação. Entretanto, pode retornar uma "falha" para obter os dados solicitados ou os dados podem 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

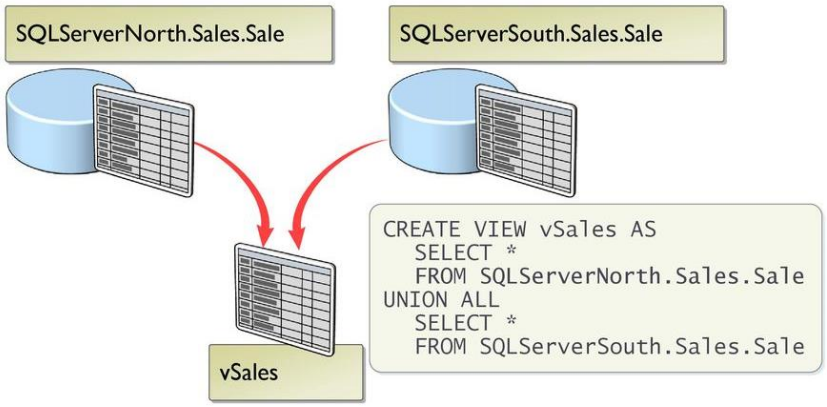
de paradigmas para se aceitar certa consistência eventual dos dados, em detrimento dos outros benefícios.

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

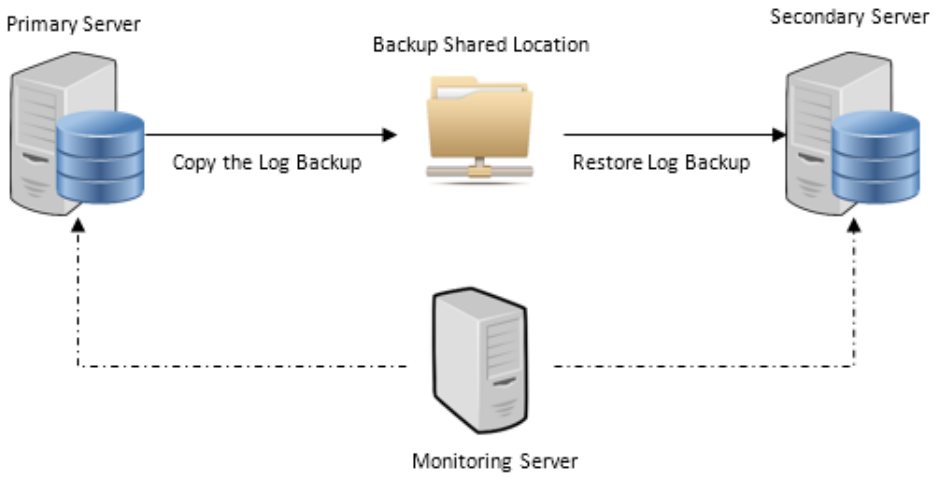
- **LOG SHIPPING:**
  - Escrita apenas no servidor primário;
  - Assíncrono;
    - Faz a replicação através de backup dos logs com o restore destes logs na réplica secundária, requerendo um local compartilhado para armazenar esses backups de log.

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 failback.

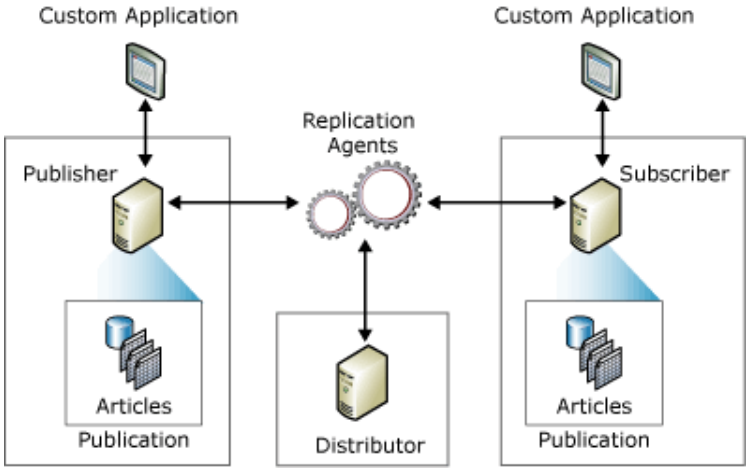
Figura 27 – SQL Server Log Shipping.



Fonte: Microsoft (2018).

- **LOG SHIPPING:**
  - 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.

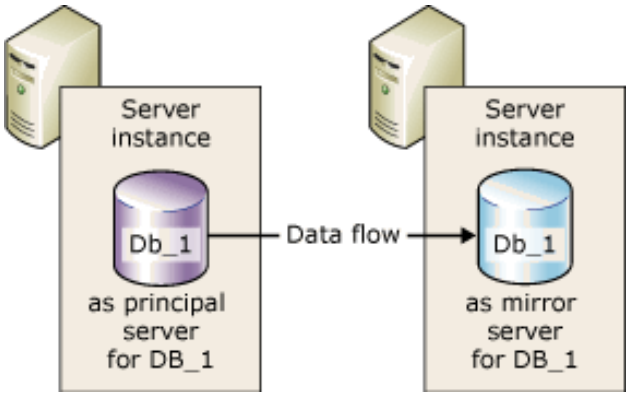


Fonte: Microsoft (2018).

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 parra leituras);
  - Replicação assíncrona ou síncrona;
    - Assíncrona: mais performática, mas posso apenas failover manual;

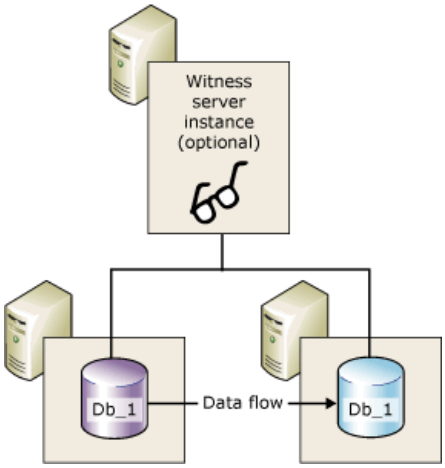
Figura 29 – Database mirroring assíncrono.



Fonte: Microsoft (2018).

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

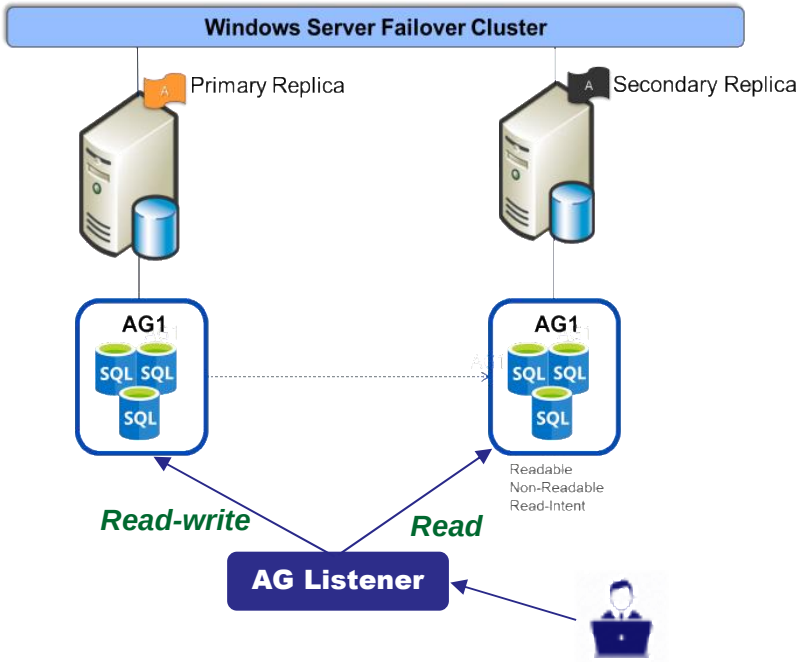
Figura 30 – Database mirroring síncrono com Witness.



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).

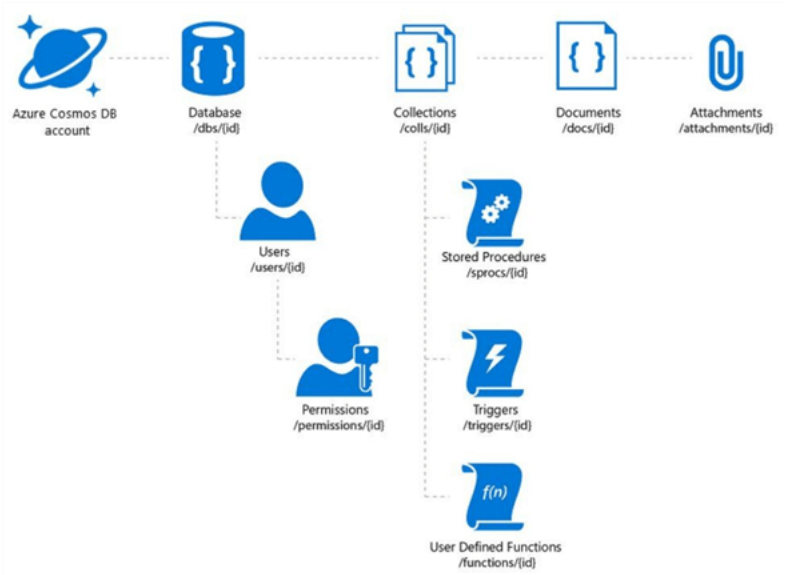
Modelos de contratação

- **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.

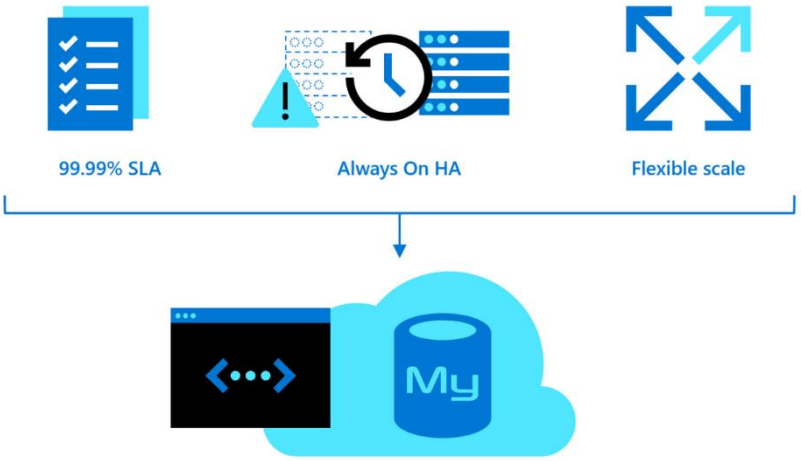
3.4. Bancos de Dados Open Source no Azure

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).

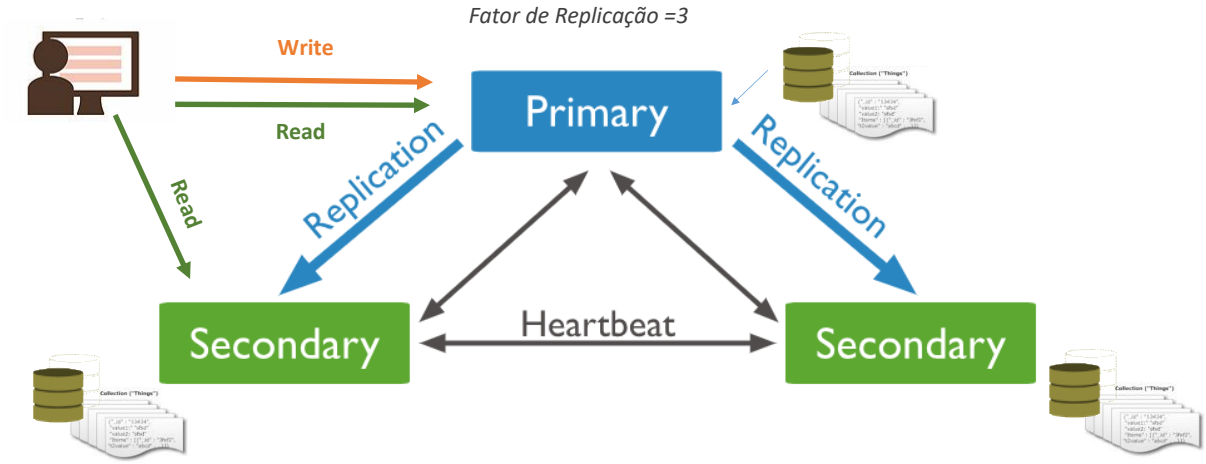
Capítulo 4. Distribuição de Dados no MongoDB

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.

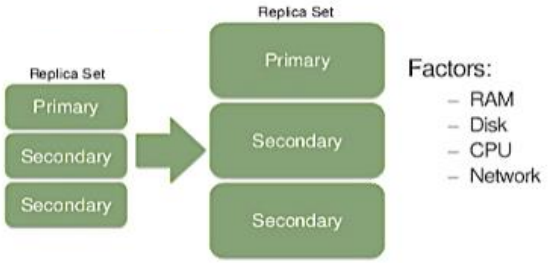


Fonte: MongoDB (2018).

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.

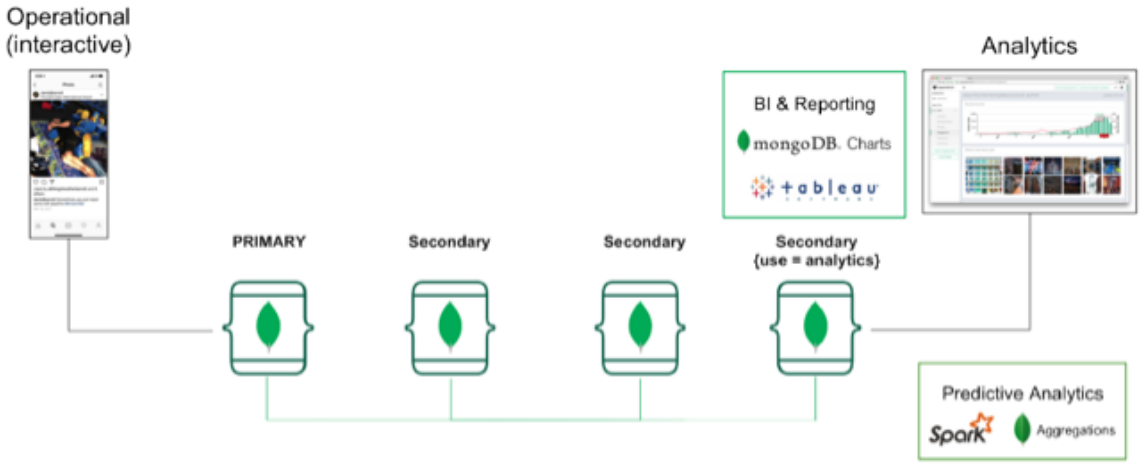
Figura 35 – Escalabilidade Vertical no MongoDB.



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.

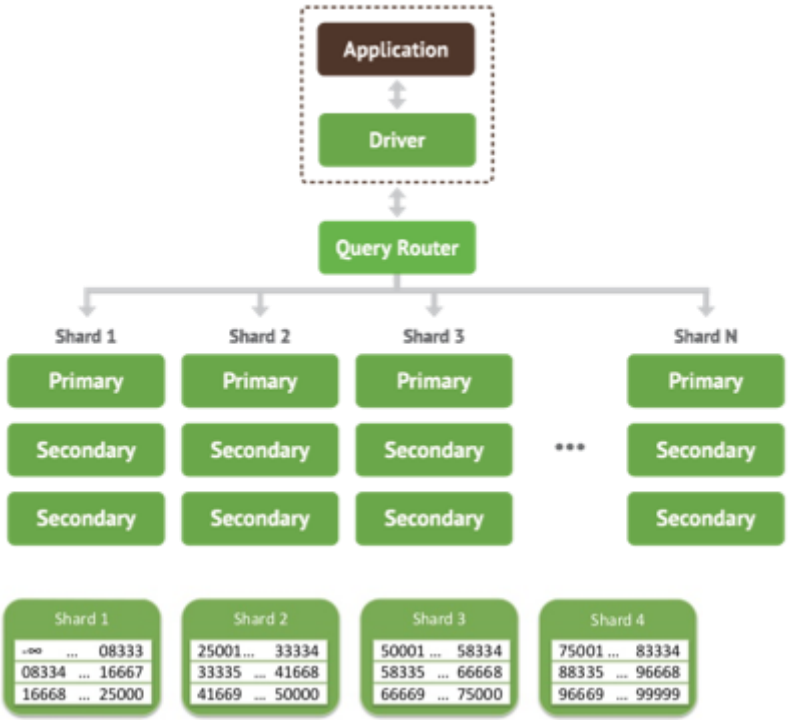
Figura 36 – Separação de workloads no MongoDB.



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.

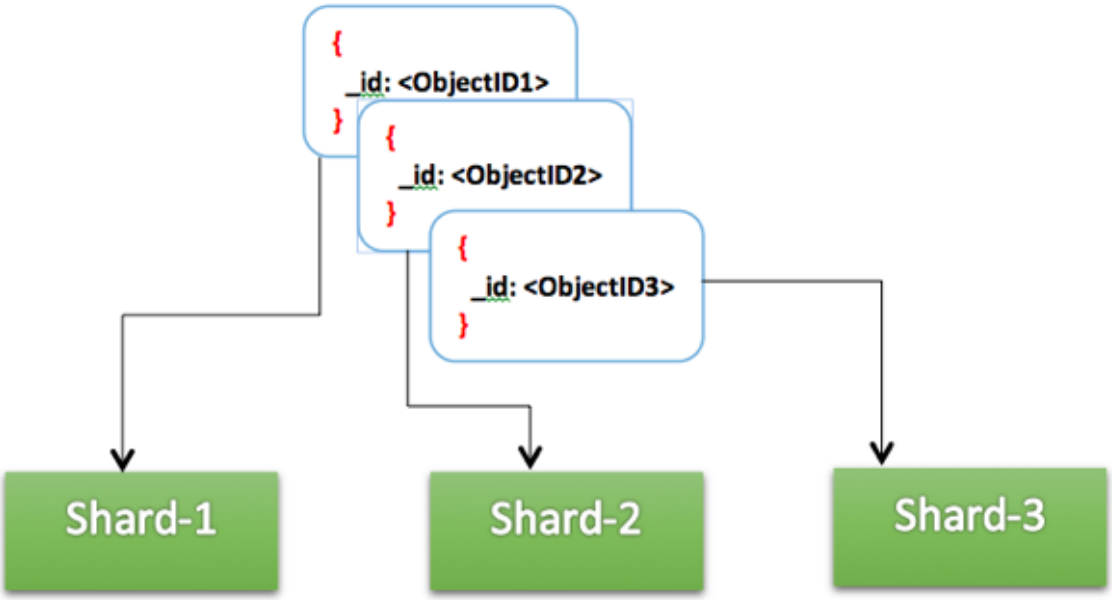
Figura 37 – MongoDB Sharding.



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.



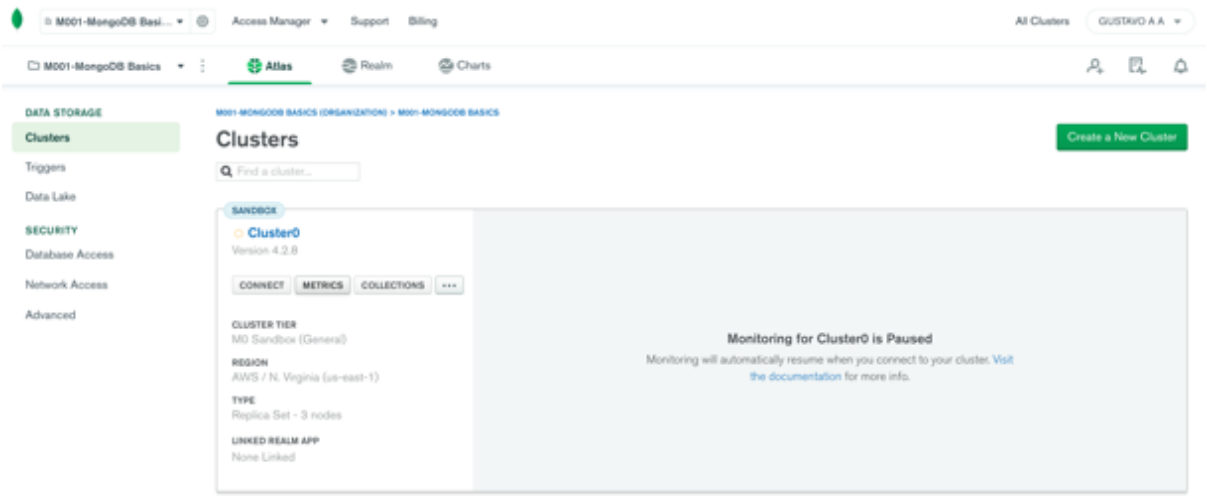
Fonte: MongoDB (2018).

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.



Fonte: Gustavo (2020).

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: 27 ago. 2020.

MONGODB Documentation. Disponível em <https://docs.mongodb.com/manual>. Acesso em: 27 ago. 2020.

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: 27 ago. 2020.

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