Unidade III

5 SISTEMA DE GERENCIAMENTO DE BANCO DE DADOS

Um sistema de gerenciamento de banco de dados (SGBD) é uma aplicação informatizada que fornece a interface entre os dados que serão armazenados fisicamente no banco de dados e os usuários (pessoa ou aplicação) desses dados. Assim, o usuário deixa de saber o formato dos dados, como e onde eles serão armazenados, pesquisados, ordenados etc. É o SGBD que tem a responsabilidade de executar cada uma das tarefas.

Definição

Um SGBD é uma aplicação ou um conjunto de aplicações informatizadas utilizadas para definir, acessar e gerenciar os dados existentes em um banco de dados.

Os SGBDs foram desenvolvidos para resolver alguns problemas e limitações com que se deparavam os usuários de sistemas baseados em arquivos de dados.

Características

As aplicações informatizadas que, anteriormente, acessavam diretamente os dados passam, de acordo com esta nova arquitetura, a pedir esse serviço a uma outra aplicação informatizada projetada especialmente para esse fim – o SGBD.

O SGBD é a única entidade que manipula o banco de dados, sendo a única entidade capaz de acessar direta e fisicamente o banco de dados. Qualquer outro acesso terá de ser realizado mediante a solicitação de serviço ao SGBD.

Os SGBDs fornecem um nível intermediário entre o usuário e os dados.

As aplicações que manipulam arquivos acessam diretamente os dados.

Quando estamos diante de um SGBD, o usuário (pessoa ou aplicação) lhe solicita o serviço desejado e é o próprio SGBD sque faz todo o conjunto de acessos de modo a responder a esse pedido, verificando, previamente, se o usuário tem ou não permissão para realizar a tarefa que solicitou.

Deve-se optar por um SGBD quando:

- Informações forem armazenadas de modo permanente.

- Houver controle central dos dados.
- Desejar-se controle de redundância.
- Existir controle de consistência e integridade dos dados.
- Houver múltiplos usuários (concorrência).
- Se quiser controle de acesso e segurança.
- Houver compartilhamento de dados entre usuários.
- Existir independência dos dados e das aplicações.
- Houver *backup* e *recovery*.

Pode-se não optar por um SGBD quando:

- Dados e aplicações atuais são simples e estáveis.
- Uma simples planilha resolve o problema.
- Informações são necessárias a apenas um usuário.
- Existe a falta de recursos financeiros.
- Existe a falta de pessoal técnico qualificado.
- Existe necessidade de uma aplicação a muito curto prazo.

O objetivo de um SGBD é garantir um ambiente apropriado para acessar e armazenar informações no banco de dados de forma confiável e eficiente, fornecendo aos usuários uma visão abstrata daquilo que gerencia, ao ocultar detalhes como a localização dos dados, o formato interno dos arquivos onde os dados são armazenados.

Quadro 17 – Requisitos fundamentais de um SGBD

Eficiência	Ser capaz de acessar, processar e alterar grandes volumes de dados de forma eficiente.
Robustez	Manter os dados de forma consistente, mesmo após falhas de *hardware* ou erros de *software*.
Controle e acessos	Controlar o acesso de múltiplos usuários aos dados de forma consistente e apenas a usuários autorizados.
Persistência	Manter os dados durante longos períodos independentemente das aplicações que os acessem.

Fonte: Rob e Coronel (2011, p. 23).

Serviços prestados por um SGBD

Ao decidir por um SGBD, os usuários têm como objetivo principal utilizar um recurso informatizado capaz de acessar, manipular e processar dados de uma forma relativamente normalizada, confiável e eficiente.

Em um SGBD, em geral, além dos dados também estão armazenados os metadados, isto é, dados que contêm a definição dos próprios dados e a definição das próprias tabelas, as regras de integridade.

O SGBD é responsável por:

- **Integração com o gerente de arquivos**: todos os sistemas operacionais fornecem uma estrutura para armazenamento e manipulação de arquivos. O objetivo desse componente é minimizar os acessos a disco (I/O operations), pois o acesso a disco é muito mais lento do que operações em memórias.

- **Gerenciamento dos dados**: os dados estão centralizados e são gerenciados unicamente pelo SGBD. As relações entre os dados também devem ser gerenciadas e verificadas pelo SGBD.

- **Integridade**: verificar se as alterações do banco de dados estão de acordo com as regras de integridade e com as validações estabelecidas na sua definição.

- **Segurança**: assegurar que os usuários apenas tenham acesso às informações que lhes são permitidas.

- ***Backup* e *recovery***: capacidade de detectar falhas decorrentes de problemas de fornecimento de energia elétrica. De *hardware*, de erros de *software* etc., e ser capaz de recolocar o banco de dados no estado estável que existia imediatamente antes da ocorrência da falha.

- **Gerenciamento da concorrência**: gerenciar o acesso de múltiplos usuários aos seus dados, mantendo a consistência da informação a que cada usuário tem acesso.

Ao fornecer um meio de desenvolvimento de mais alto nível, a utilização de SGBD permite acelerar o processo de desenvolvimento de novas aplicações, reduzindo o tempo e os custos de manutenção, obtendo-se vantagens evidentes com sua utilização.

Uma das vantagens mais evidentes na adoção de um SGBD é a utilização de um único conjunto lógico e organizado de dados estruturados, autônomo em relação às aplicações que o processam.

Lembrete

Segurança física permite que apenas pessoas autorizadas tenham acesso a áreas específicas. Dependendo do tipo de implementação de banco de dados de pesquisas, criptografar os dados pode ser útil para proteger os dados de usuários não autorizados.

No *site* da Microsoft, é possível encontrar duas versões de uso de SQL Server 2019 de forma gratuita, ou seja, não comercial. A saber:

- O SQL Server 2019 Developer é uma edição gratuita completa, licenciada para uso como banco de dados de desenvolvimento e teste em um ambiente de não produção.
- O SQL Server 2019 Express é uma edição gratuita do SQL Server, ideal para desenvolvimento e produção de aplicativos de área de trabalho, web e pequenos servidores.
- O SQL Server 2019 é um sistema gerenciador de bancos de dados, relacionais, SGBDR, que funciona nos sistemas operacionais Windows e Linux.

O Microsoft SQL Server foi originalmente baseado no Sybase SQL Server X, em sua versão 4.2. Na versão 6, a Microsoft implementou modificações visando fazer uso de características multitarefas do Windows NT. Uma vez instalado o SQL Server, são criadas automaticamente quatro databases:

- Master.
- Model.
- Tempdb.
- Msdb.

Depois, podemos criar e instalar nossos próprios bancos de dados livremente, os quais serão os bancos de dados de usuário.

Embora ambos os tipos de bancos de dados (sistema e usuário) armazenem dados, o SQL Server utiliza os bancos de sistema para operar e gerenciar o sistema. O catálogo de sistema, por exemplo, consiste unicamente de tabelas armazenadas no banco de dados master.

Observação

O SQL Server mantém quatro bancos de dados do sistema, Master, Model, Tempdb e Msdb, importantes para a execução de sua instância, que não devem ser modificados.

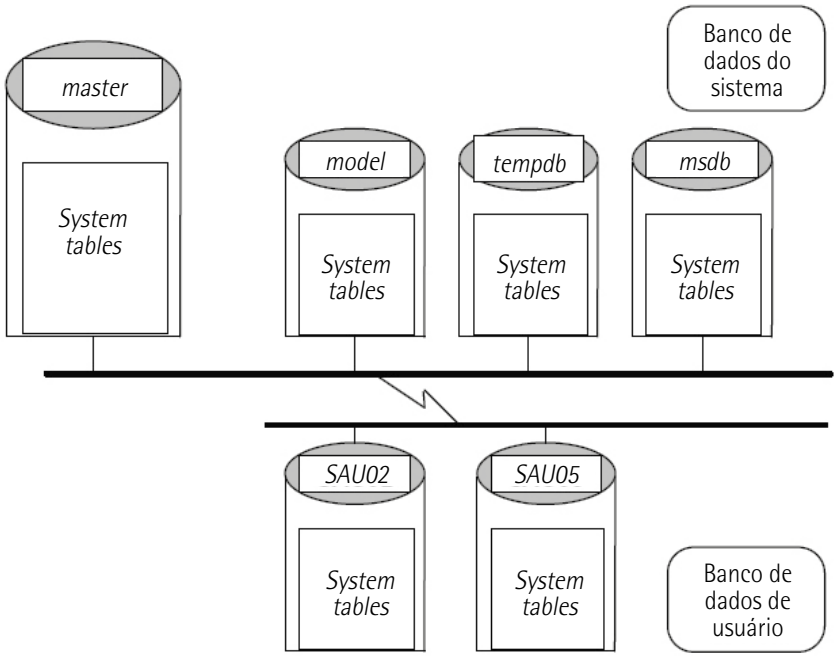

Figura 69 – Bancos de dados no SQL Server

5.1 Instalação e configuração de um SGBD

Para essa demonstração, acesse o *site* da Microsoft, através do endereço: https://bit.ly/3ouVKcE, em seguida navegue até o Centro de downloads e selecione a versão Microsoft® SQL Server® 2019 Express, disponível no endereço: https://bit.ly/342e2lQ, a qual necessita de requisito para o sistema operacional Windows 10, conforme a seguir:

Requisitos do sistema

Sistema operacional compatível

Windows 10; Windows Server 2016; Windows Server 2019

- **Processador**
  - Intel - processador compatível com uma velocidade mínima de 1 GHz ou mais rápido
- **RAM**
  - Mínimo de 512 MB
- **Espaço em disco rígido**
  - 4,2 GB de espaço em disco

Limitações: O Microsoft SQL Server Express dá suporte a um processador físico, 1 GB de memória e 10 GB de armazenamento

Figura 70 – Requisitos do sistema

Siga os procedimentos adiante, que listam o passo a passo para a instalação e mostram algumas configurações necessárias.

Após executar o arquivo baixado, será inicializada a tela que vemos na figura a seguir. Depois do *download*, inicie o instalador do SQL Server.

Instalação do Microsoft SQL Server, SQL Reporting Services e SQL Server Management Studio

Descreveremos, aqui, as configurações necessárias para a utilização do banco de dados Microsoft SQL Server. Para dar sequência à instalação, é necessário baixar a versão Express With Advanced Services, no endereço: https://bit.ly/3uQHGft.

Após acessar a página para *download*, será exibida a imagem a seguir:

Figura 71 – Versões SQL Server disponíveis para *download*

As próximas imagens mostram o passo a passo para instalar o SQL Server.

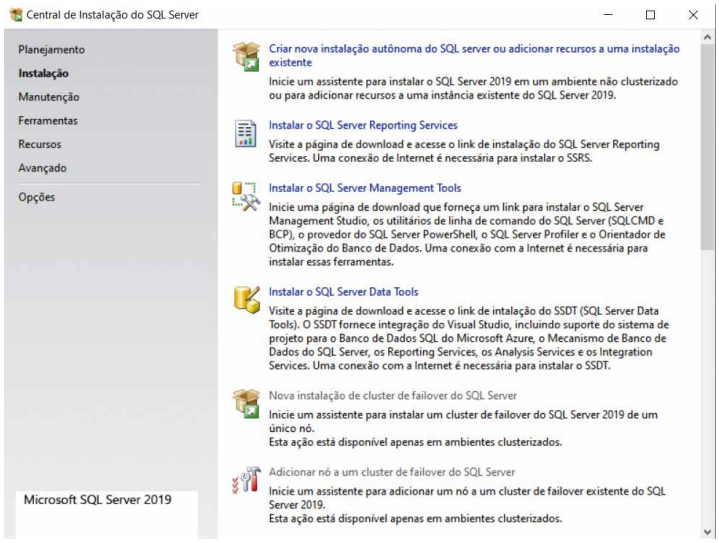

Figura 72 – Iniciando o instalador do SQL Server 2019

Saiba mais

O SQL Server oferece, através da biblioteca ADO.NET, conectores para que algumas linguagens de programação possam ser manipuladas com SGBD. Para entender melhor quais conectores podem ser utilizados em qual linguagem, acesse:

MICROSOFT. *Microsoft ADO.NET for SQL Server*. 2020. Disponível em: https://bit.ly/3dkX65Q. Acesso em: 13 abr. 2021.

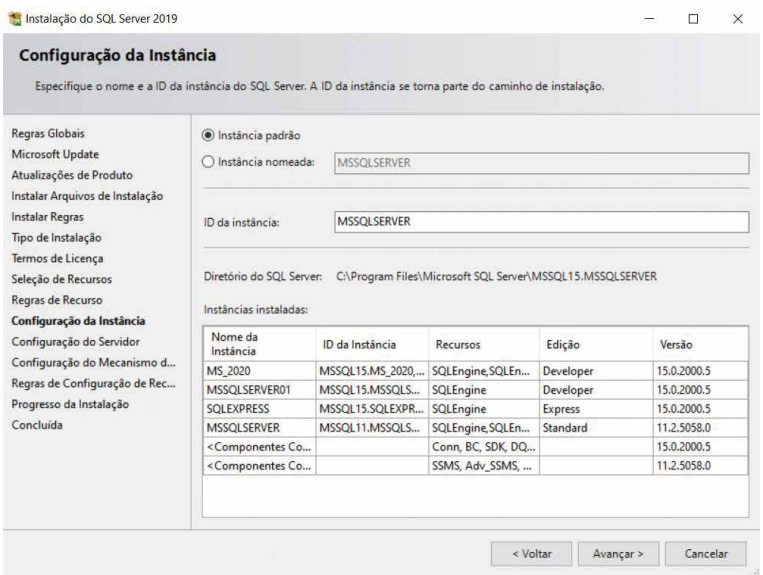

Figura 73 – Configuração da instância do SQL Server 2019

Nesta tela, podemos definir a forma de inicialização dos serviços e quais contas serão utilizadas para cada acesso. Vamos alterar apenas o modo de inicialização para manual.

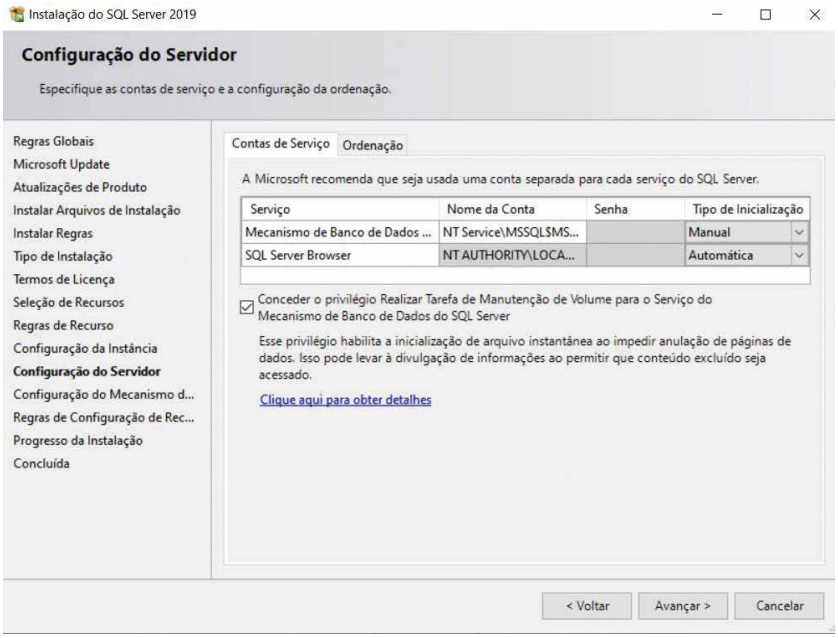

Figura 74 – Configuração do servidor do SQL Server 2019

Na aba Ordenação, podemos definir o idioma de caracteres do banco de dados e do Analysis Services. É aqui que definimos se o banco terá suporte para a acentuação, se será *case-sensitive* (letras minúsculas são diferentes de letras maiúsculas. Essa é a configuração padrão) ou *case-insensitive* (letras maiúsculas e minúsculas são interpretadas pelo banco como a mesma coisa).

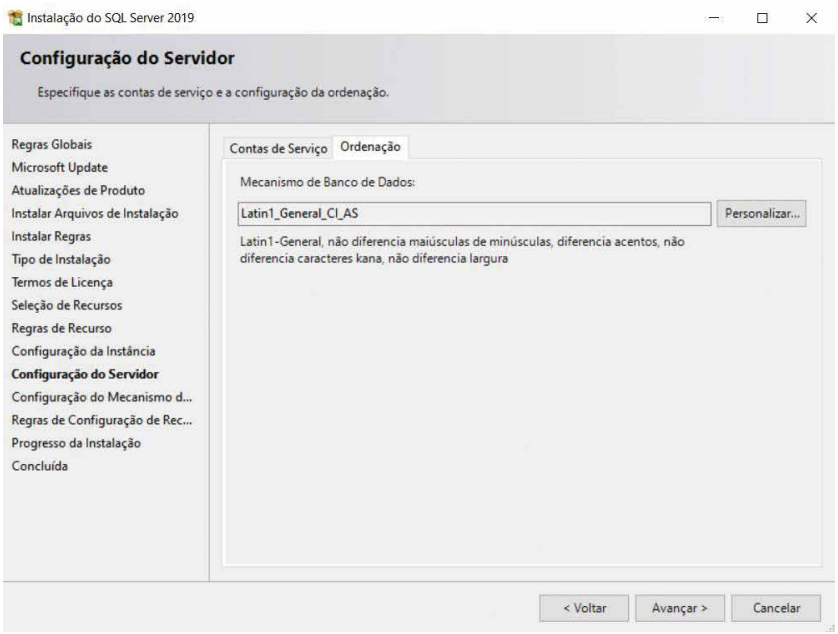

Figura 75 – Configuração do servidor com padrão *case-sensitive* do SQL Server 2019

Caso necessite executar alguma configuração adicional, clique em Personalizar e adicione as configurações necessárias. Em seguida, será gerada a tela de configuração de usuário. Anote a senha do usuário "sa":

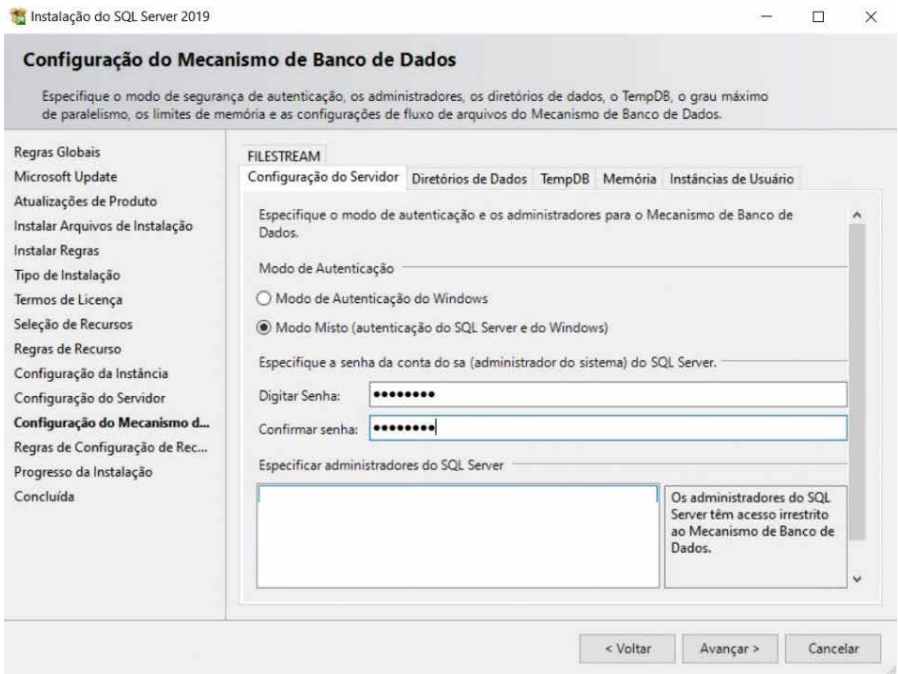

Figura 76 – Configuração do mecanismo de banco de dados

Aguarde a instalação ser finalizada, a tela a seguir confirmará que a instalação ocorreu com sucesso.

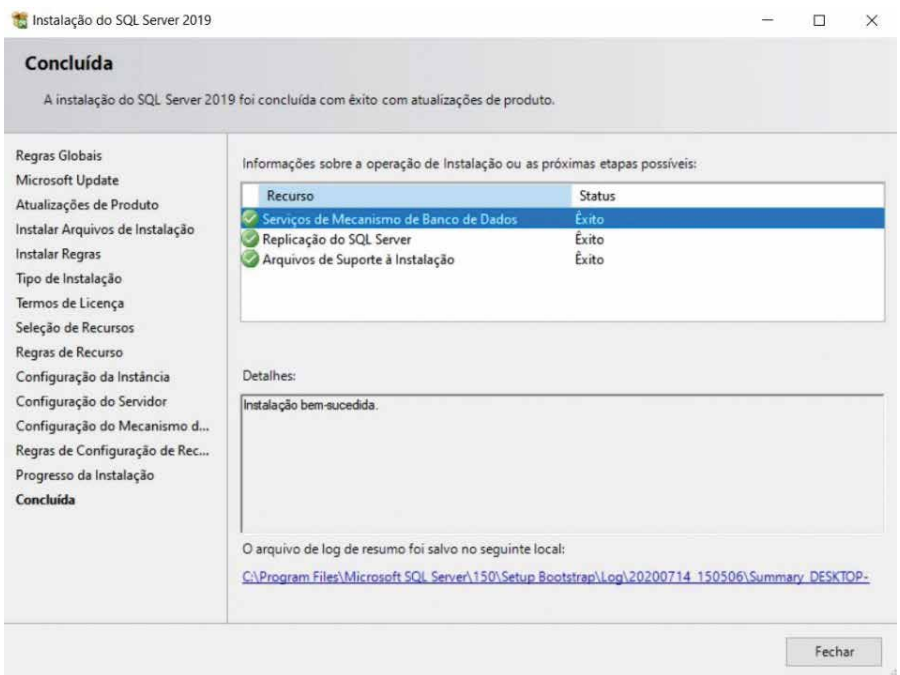

Figura 77 – Finalização

Na tela da figura anterior, clique no menu "Instalar arquivos de instalação". Isso irá redirecionar para a página de *download* da versão mais atual do SQL Server Management Studio (SSMS). Selecione a opção Baixar o SSMS (SQL Server Management Studio).

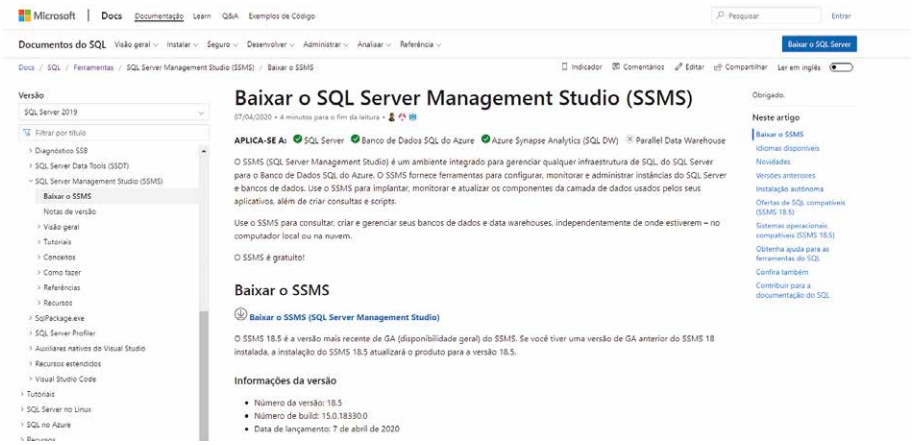

Figura 78 – *Site* da Microsoft para *download* do SSMS

Feche o instalador do SQL Server Express e execute o instalador do SQL Server Management Studio, que foi baixado no passo anterior. Quando o instalador for aberto, selecione o local da instalação que deseja e pressione o botão Instalar.

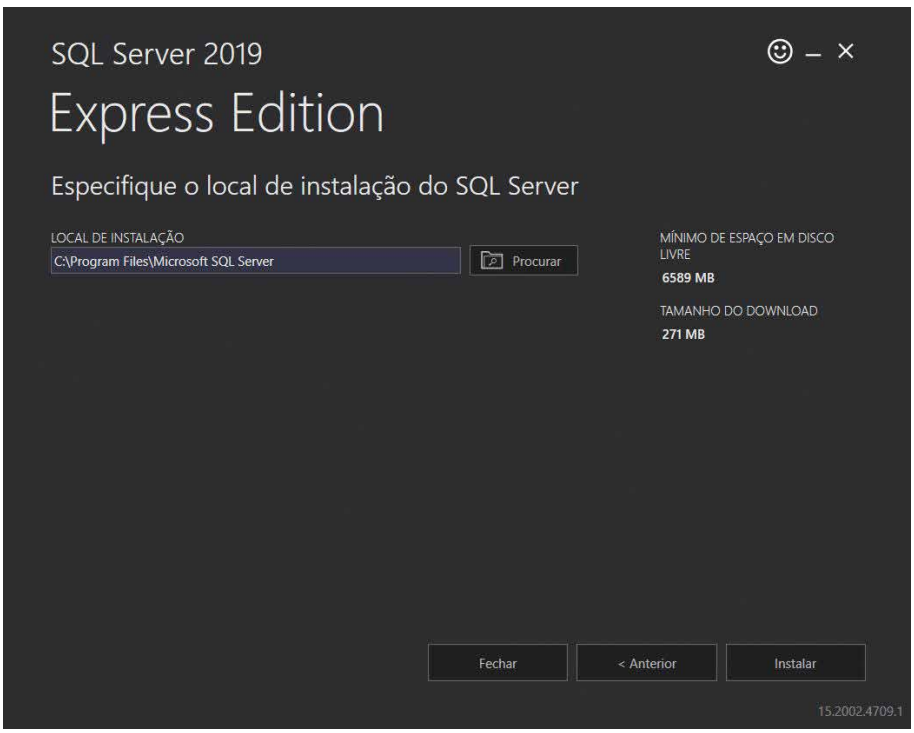

Figura 79 – Instalando o Express Edition

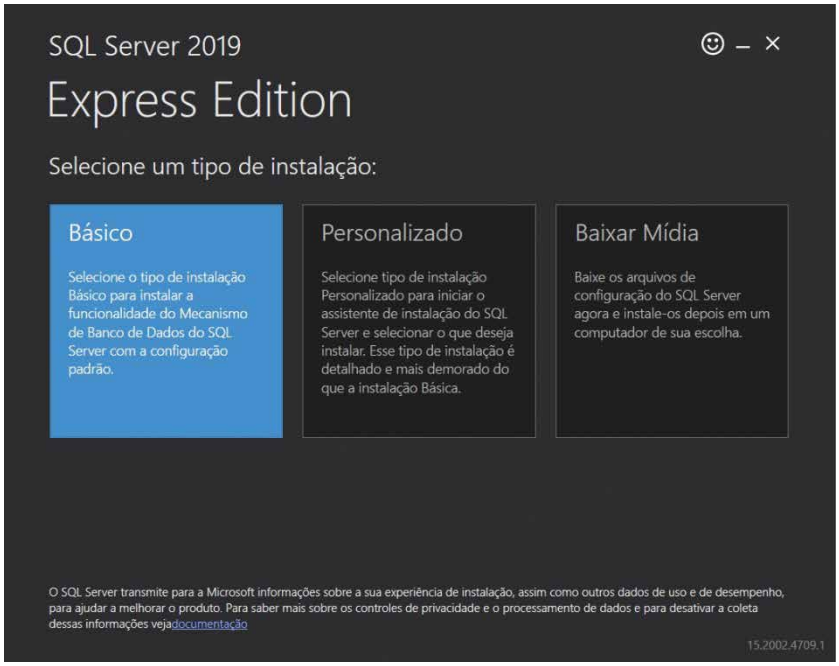

Figura 80 – Seleção do modelo básico Express Edition

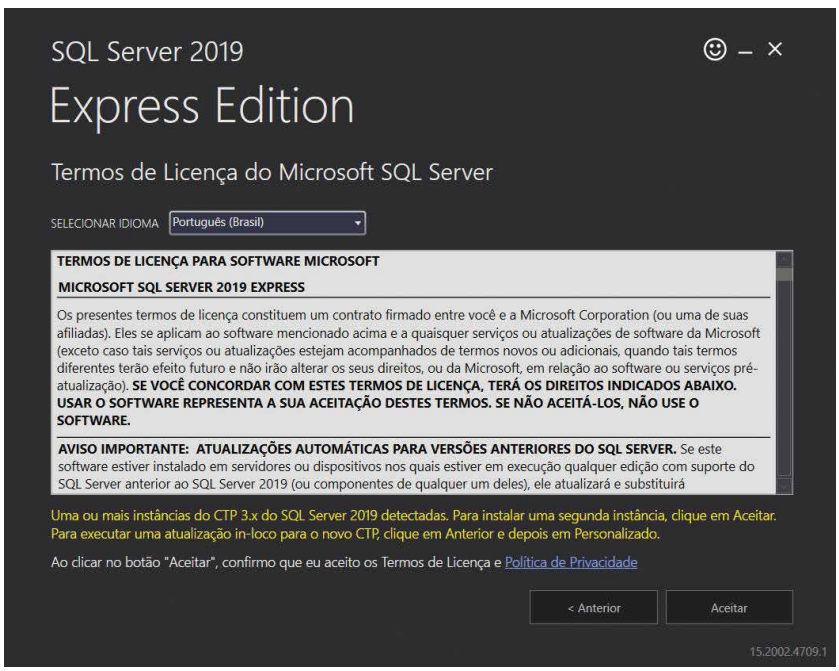

Figura 81 – Termo de licença e idioma do Express Edition

Aguarde todos os procedimentos para instalação. Logo será gerada a tela de instalação bem-sucedida, conforme vemos a seguir.

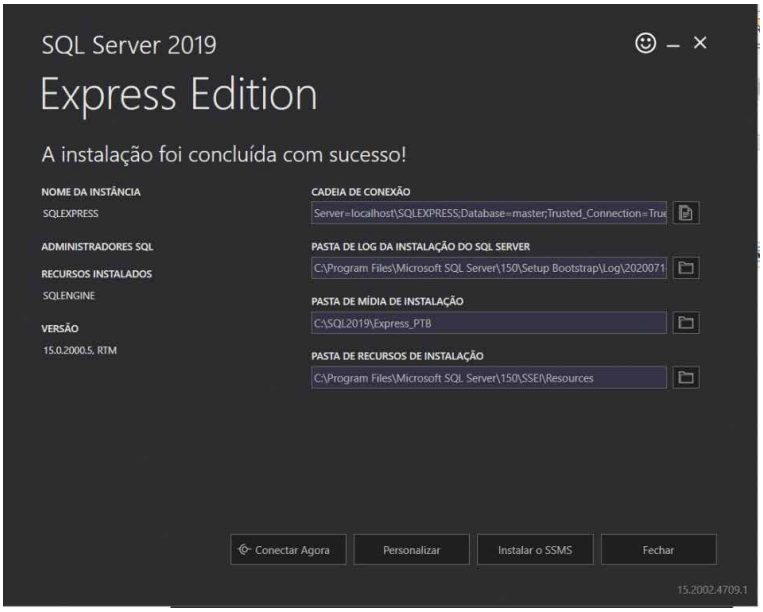

Figura 82 – Instalação finalizada do Express Edition

5.2 Inicializando o SGBD – SQL Server

Esta seção discorre sobre os passos necessários para a configuração correta da aplicação no ambiente de desenvolvimento, suportado pelas ferramentas instaladas nos passos anteriores.

Figura 83 – Iniciando o Express Edition

5.3 Administração de SGBD

Segurança (controle de acesso)

Gerenciar dados é uma tarefa que requer cuidados redobrados, visto que gerenciadores de recuperação de um SGBD também precisam possuir recursos mais rigorosos para lidar com falhas mais catastróficas,

como as falhas de disco. A principal técnica utilizada para lidar com essas falhas é se preocupar e criar *backup* do banco de dados, para que todas as informações do banco de dados, juntamente com o *log* sejam periodicamente copiados para um meio de armazenamento, que pode ser fita magnética ou outros dispositivos de armazenamento externos de alta capacidade, como os armazenamentos em nuvem Azure, AWS (grande plataforma de armazenamento de dados *on-line*). Caso ocorra uma falha catastrófica do sistema, a cópia de *backup* mais recente pode ser recarregada de qualquer recurso onde os dados estejam armazenados e o sistema poderá ser reiniciado.

Os dados de aplicações críticas, como bancos, seguros, mercado de ações e outros bancos de dados, são copiados de tempos em tempos em sua totalidade e movidos para locais seguros e fisicamente separados. Câmaras de armazenamento subterrâneas têm sido usadas para proteção contra danos ocasionados por inundação, tempestade, terremoto ou incêndio.

Eventos como o ataque terrorista de 11 de setembro de 2011, em Nova York, e o desastre do furacão Katrina, ocorrido no ano de 2005 em Nova Orleans, criaram uma maior conscientização da recuperação de desastres dos bancos de dados críticos aos negócios (ELMASRI; NAVATHE, 2011).

Para evitar perder todos os efeitos das transações que foram executadas desde o último *backup*, é comum fazer o *backup* do *log* do sistema em intervalos mais frequentes do que o do banco de dados inteiro, copiando-o periodicamente para fita magnética. O *log* do sistema costuma ser muito menor do que o próprio banco de dados, e, portanto, pode ser copiado com mais frequência. Portanto, os usuários não perdem todas as transações que realizaram. Todas as transações confirmadas e registradas na parte do *log* do sistema que foi copiada para fita podem ter efeito sobre o banco de dados refeito. Um novo *log* é iniciado após cada *backup* do banco de dados.

Assim, para recuperar-se da falha do disco, o banco de dados é primeiro recriado no disco com base em sua cópia de *backup* mais recente em fita. Depois disso, os efeitos de todas as transações confirmadas, cujas operações foram registradas nas cópias do *log* do sistema, são refeitos.

Os principais objetivos da realização do *backup* de arquivos de banco de dados são:

- Evitar a perda, potencialmente catastrófica, de dados.
- Recuperar dados que foram apagados ou atualizados de forma incorreta (tabelas, registros, outros objetos do banco de dados etc.).
- Recuperar dados quando há uma falha de *hardware* (falha de mídia, uma unidade de disco danificada ou perda permanente de um servidor).
- Facilitar a criação de ambientes de testes idênticos ao de produção.

Para a realização de um *backup*, é preciso ter os seguintes itens:

- ***Software* para a realização do *backup***: utilizar um programa que faça o *backup* dos seus dados de forma satisfatória (Backup Exec, por exemplo).

- **Local físico para a gravação do *backup***: discos rígidos, fita DAT, NAS (*network attached storage*), nuvem etc.

- **Planejamento da rotina de *backup***: definir, documentar e implementar a rotina de *backup* adequada.

5.4 Tipos de *backups*

Todo o serviço de *backup* de banco de dados só é possível através de uma ferramenta de SGBD. O SQL Server oferece a possibilidade de criar algumas formas de *backup*, o que permite gerar uma maior proteção aos dados essenciais e críticos. Para isso, deve-se criar o *backup* e a restauração dos dados, os quais devem ser personalizados em um ambiente específico e devem funcionar com os recursos disponíveis. Portanto, um uso confiável de *backup* e restauração para recuperação requer uma estratégia que, se for bem projetada e equilibrada, garante os requisitos de negócio para máxima disponibilidade de dados e mínima perda de dados, levando em consideração o custo de manutenção e armazenamento de *backup*.

Desenhar o procedimento de uma estratégia de *backup* e restauração eficaz requer planejamento, implementação e testes cuidadosos. Testar é necessário, pois a estratégia só terá garantia após testar o restauro dos dados com êxito em todas as combinações incluídas, garantindo, assim, a consistência física do banco de dados restaurado. É necessário considerar uma variedade de fatores, incluindo:

- Todas as metas da organização para os bancos de dados de produção, especialmente os requisitos para disponibilidade e proteção contra perda ou danificação de dados.

- A natureza de cada banco de dados: o tamanho, os padrões de uso, a natureza de seu conteúdo, os requisitos dos dados etc.

- Restrições de recursos, como *hardware*, pessoal, espaço para armazenagem de mídia de *backup*, a segurança física da mídia armazenada etc.

O modelo de recuperação apropriado

As estratégias de operações de *backup* e restauração devem ocorrer dentro da realidade de cada organização em um modelo de recuperação. O modelo de recuperação é uma propriedade de banco de dados que controla a forma de gerenciamento do *log* de transações. Assim, o modelo de recuperação de um banco de dados determina quais tipos de *backup* e cenários de restauração são compatíveis com o banco de dados e qual será o tamanho dos *backups* de *log* de transações. Em geral, um banco de dados usa o modelo de recuperação simples ou o modelo de recuperação completa. O modelo de recuperação completa pode ser aumentado alternando para o modelo de recuperação *bulk-logged* antes das operações em massa.

A escolha do modelo de recuperação para o banco de dados depende de seus requisitos empresariais. Para evitar gerenciamento de *log* de transações e simplificar o *backup* e a restauração, use o modelo de recuperação simples. Para minimizar exposição à perda de trabalho, às custas de uma sobrecarga administrativa, use o modelo de recuperação completa. Para minimizar o impacto sobre o tamanho do *log* durante operações *bulk-logged* e, ao mesmo tempo, permitir a recuperação dessas operações, use o modelo de recuperação *bulk-logged*.

Backup somente de cópia

Backup somente de cópia é um *backup* de uso especial que é independente da sequência regular dos *backups* do SQL Server.

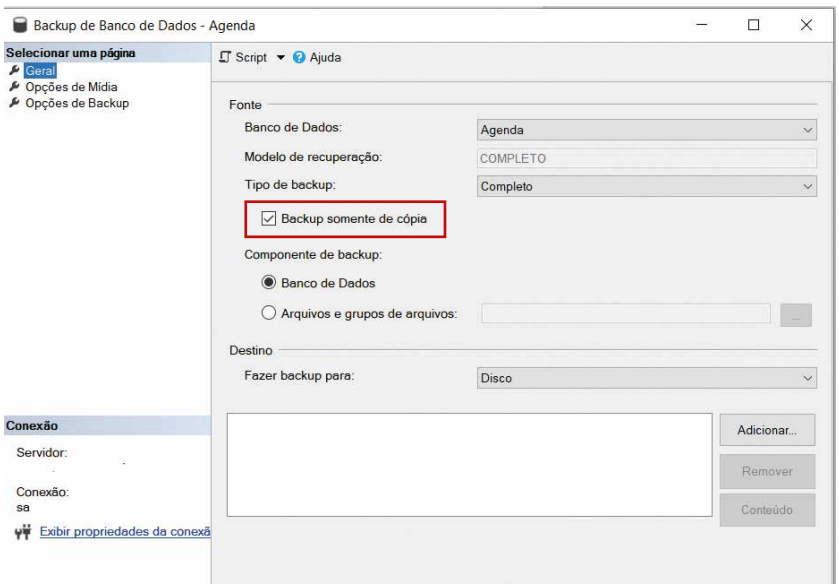

Figura 84 – Tela de *backup* somente de cópia – SQL Server

Backup de dados

Backup de dados é um *backup* de dados em um banco de dados completo (um *backup* de banco de dados), um banco de dados parcial (um *backup* parcial) ou um conjunto de arquivos de dados ou grupos de arquivos (um *backup* de arquivo).

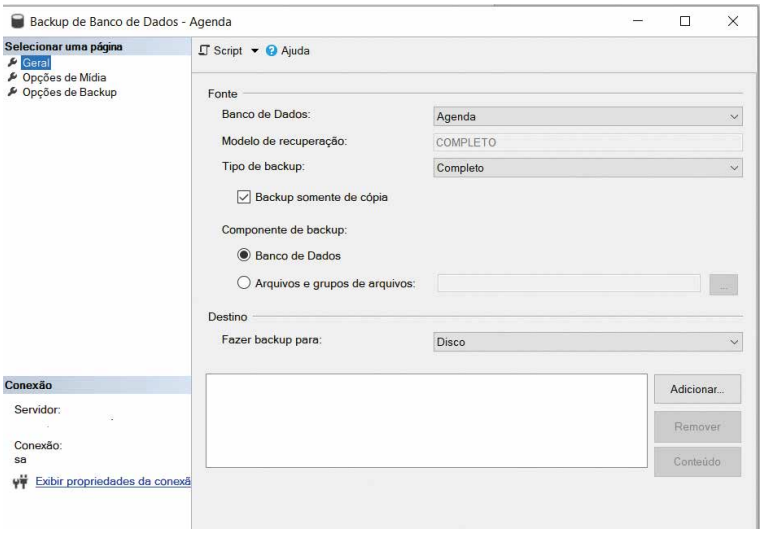

Figura 85 – Tela de *backup* completo – SQL Server

Os *backups* completos de banco de dados representam todo o banco de dados no momento em que o *backup* é concluído, enquanto os *backups* de banco de dados diferenciais contêm somente alterações feitas no banco de dados desde seu *backup* completo mais recente.

Backup diferencial

Backup diferencial é um *backup* de dados que se baseia no *backup* completo mais recente, completo ou parcial, ou um conjunto de arquivos de dados ou grupos de arquivos (a base diferencial) que contém somente as extensões de dados alterados desde a base diferencial.

Um *backup* diferencial parcial registra apenas as extensões de dados que foram alteradas nos grupos de arquivos desde o *backup* parcial anterior, conhecido como a base para o diferencial.

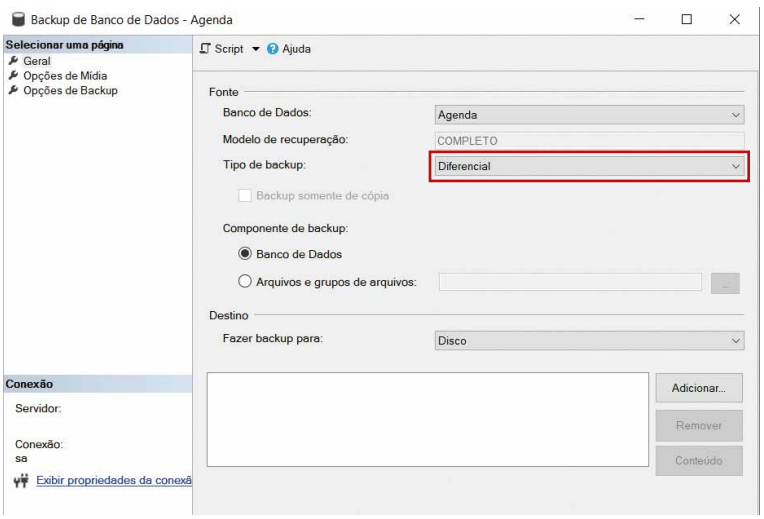

Figura 86 – Tela de *backup* diferencial – SQL Server

Backup completo

Backup completo é um *backup* de dados que contém todos os dados em um banco de dados ou em um conjunto de arquivos ou grupos de arquivos, além de *log* suficiente para permitir a recuperação desses dados. Esse tipo de *backup* faz as cópias de todos os dados para outro conjunto de mídia, independentemente de terem sido modificados ou não. A principal vantagem é que uma cópia completa de todos os dados está disponível em um único conjunto de mídia. Isso resulta em uma possibilidade maior de recuperar os dados íntegros, menor complexidade da operação de recuperação e menor tempo para recuperar os dados. As principais desvantagens são que leva mais tempo para executar um *backup* completo do que outros tipos (por vezes, por um fator de 10 ou mais) e requer mais espaço de armazenamento, já que todos os dados são armazenados a cada *backup* realizado.

Backup de *log*

Trata-se de *backup* de *logs* de transações que inclui todos os registros de *log* dos quais não foi feito *backup* em um *backup* de *log* anterior (modelo de recuperação completa).

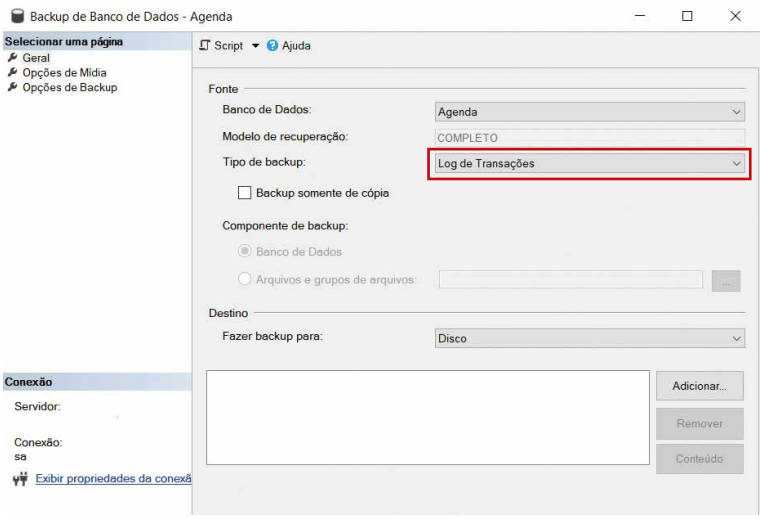

Figura 87 – Tela de *backup* de *log* de transações – SQL Server

Estratégias de *backup*

Depois de selecionar um modelo de recuperação que satisfaça os requisitos empresariais para um banco de dados específico, é preciso planejar e implementar uma estratégia de *backup* correspondente. A melhor estratégia de *backup* depende de uma série de fatores, dos quais os seguintes são especialmente significativos:

- Quantas horas ao dia os aplicativos precisam acessar o banco de dados?
- Se houver um período de pouca atividade previsível, recomendamos que você agende *backups* de banco de dados completos para aquele período.
- Com que frequência as alterações e atualizações deverão ocorrer? Se as alterações forem frequentes, considere o seguinte:

  - No modelo de recuperação simples, agende *backups* diferenciais entre os *backups* de banco de dados completos. Um *backup* diferencial captura só as alterações desde o último *backup* completo do banco de dados.

  - No modelo de recuperação completa, é importante agendar *backups* de *log* frequentes. O agendamento de *backups* diferenciais entre *backups* completos pode reduzir o tempo de restauração, reduzindo também o número de *backups* de *log* a serem restaurados após a restauração dos dados.

- As alterações ocorrem geralmente em uma pequena parte do banco de dados ou em uma grande parte do banco de dados? Para um banco de dados grande, no qual as mudanças estão concentradas em uma parte dos arquivos ou grupos de arquivos, *backups* parciais e *backups* de arquivo podem ser úteis.

- Quanto espaço em disco é necessário para um *backup* completo de banco de dados?

- Há quanto tempo sua empresa exige a manutenção de *backups*?

Verifique se tem uma agenda de *backup* adequada, estabelecida de acordo com as necessidades dos aplicativos e dos requisitos de negócios. Conforme os *backups* envelhecem, o risco de perda de dados é maior, a menos que você tenha uma maneira de regenerar todos os dados até o ponto de falha. Antes de optar por descartar os *backups* antigos, devido a limitações de recursos de armazenamento, considere se é necessária uma capacidade de recuperação distante no passado.

Estimar o tamanho de um *backup* de banco de dados completo

Antes de implementar uma estratégia de *backup* e restauração, calcule quanto espaço em disco um *backup* de banco de dados completo usará. A operação de *backup* copia os dados no banco de dados para o arquivo de *backup*. O *backup* contém só os dados reais no banco de dados e não qualquer espaço não utilizado. Portanto, o *backup* é geralmente menor do que o próprio banco de dados.

Agendar *backups*

A execução do *backup* tem um efeito mínimo sobre as transações em andamento, portanto as operações de *backup* podem ser realizadas durante a operação regular. É possível executar um *backup* do SQL Server com um efeito mínimo sobre as cargas de trabalho de produção.

Depois de decidir os tipos de *backup* necessários e a frequência de execução de cada tipo, recomendamos agendar *backups* regulares como parte de um plano de manutenção de banco de dados.

Teste seus *backups*

Não existirá uma estratégia de restauração até que os *backups* tenham sido testados. É muito importante testar a estratégia de *backup* completamente para cada um dos bancos de dados, restaurando uma cópia do banco de dados em um sistema de teste. Também é recomendável que, depois de restaurar

o *backup*, sejam executadas verificações de consistência do banco de dados usando DBCC CHECKDB do banco de dados para validar se a mídia de *backup* não foi danificada.

Documentar estratégia de *backup*/restauração

Recomenda-se que todos os procedimentos de *backup* e restauração sejam documentados e que se mantenha uma cópia da documentação em seu livro de execuções. Recomenda-se também manter um manual de operações para cada banco de dados. Esse manual operacional deve documentar o local dos *backups*, os nomes do dispositivo de *backup* (se houver) e o tempo necessário para restaurar os *backups* de teste.

5.5 Monitoramento do banco de dados

Além de todos os cuidados necessários para criação do *backup* e do *restore*, o Microsoft SQL Server e o sistema operacional Microsoft Windows possuem utilitários que permitem exibir a condição atual do banco de dados e acompanhar o desempenho conforme as condições mudam. Há uma variedade de ferramentas e técnicas que podem ser usadas para monitorar o Microsoft SQL Server. O monitoramento do SQL Server ajuda a:

- Determinar se o desempenho pode ser melhorado. Por exemplo, ao monitorar os tempos de resposta a consultas utilizadas com frequência, é possível determinar se são necessárias alterações na consulta ou nos índices das tabelas.

- Avaliar a atividade de usuário. Por exemplo, monitorando os usuários que tentam se conectar a uma instância do SQL Server, é possível determinar se a segurança está configurada adequadamente e testar aplicativos ou sistemas de desenvolvimento. Monitorando consultas SQL à medida que são executadas, é possível determinar se estão escritas corretamente e produzindo os resultados esperados.

- Solucionar problemas ou depurar componentes de aplicativos, como procedimentos armazenados.

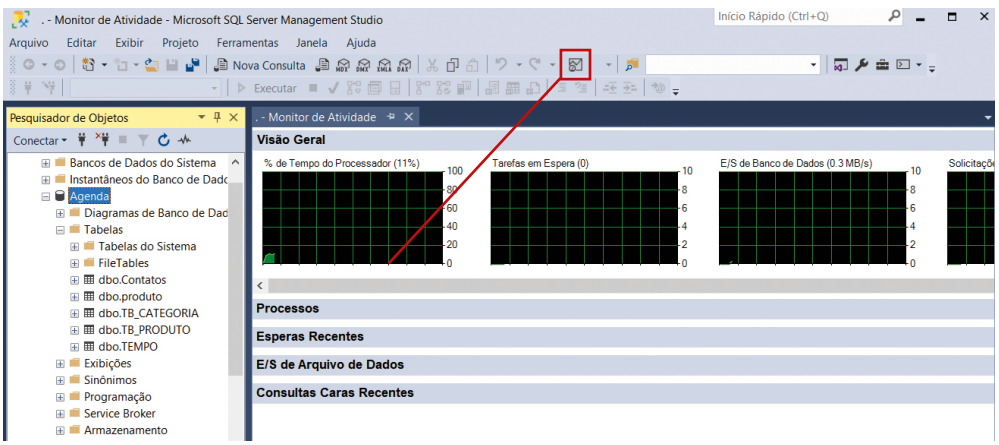

Figura 88 – Tela de monitoramento – SQL Server

Monitorando um ambiente dinâmico

Qualquer alteração nas condições da empresa pode resultar em alterações no desempenho. As alterações no desempenho podem ser observadas à medida que o número de usuários aumenta, o acesso de usuário e os métodos de conexões mudam, o conteúdo do banco de dados cresce, os aplicativos cliente se modificam, os dados nos aplicativos se alteram, as consultas se tornam mais complexas e o tráfego de rede aumenta. Com as ferramentas para monitorar o desempenho, é possível associar algumas alterações no desempenho às mudanças de condições e consultas complexas. Por exemplo:

- Monitorando os tempos de resposta a consultas utilizadas com frequência, é possível determinar se são necessárias alterações na consulta ou nos índices das tabelas em que as consultas são executadas.

- Monitorando consultas Transact-SQL à medida que são executadas, é possível determinar se elas estão escritas corretamente e produzindo os resultados esperados.

- Monitorando os usuários que tentam se conectar a uma instância do SQL Server, é possível determinar se a segurança está configurada adequadamente e testar aplicativos ou sistemas de desenvolvimento.

O tempo de resposta é o tempo necessário para que a primeira linha do conjunto de resultados seja retornada para o usuário, na forma de uma confirmação visual de que uma consulta está sendo processada. A taxa de transferência é o número total de consultas manipuladas pelo servidor durante um período de tempo especificado.

Quanto mais usuários conectados ao banco de dados, maior o aumento e a competição de recursos do servidor, o que gera diminuição do processamento global e do aumento do tempo de resposta.

5.6 Replicação de dados

A replicação de dados pode ser considerada como um processo de copiar dados de um local para outro. Algumas ferramentas de SGBD permitem criar cópias de dados atualizadas em caso de desastres. A replicação pode ocorrer em várias áreas, como rede de área de armazenamento, rede de área local ou rede de área ampla, e também os serviços de armazenamento em nuvens. Existem quatro lugares onde a replicação de dados pode ocorrer conforme vemos a seguir:

- **Baseada em *host***: a cópia acontece a partir de servidores onde a cópia de dados é feita de um *site* para outro, usando *software* de aplicativos.

- **Baseada em *hypervisor***: é uma replicação baseada em *host* para máquinas virtuais inteiras de um servidor *host* ou de *cluster* de *host* para outro. Para esse tipo de replicação deve se projetar as VMs, as quais usam os recursos da CPU, podendo afetar o seu desempenho.

- **Baseada em matrizes**: uma vez que foi utilizado um *software* compatível à replicação em matrizes, é possível copiar dados automaticamente entre matrizes. Essa replicação é mais resiliente e requer pouca coordenação quando implantada, possui um ambiente de armazenamento homogêneo mais limitado, pois exige que as matrizes possuam o mesmo destino.

- **Baseada em rede**: esse tipo de replicação ocorre em ambientes de armazenamento heterogêneo, funciona com qualquer *array* e suporta qualquer plataforma *host*. Existe limitação de recursos nesse tipo de replicação se comparada ao tipo *array* e em *host*.

A replicação potencializa o uso de informações em tempo real para DataOps, aprimorando sistemas de *big data* e aplicativos móveis, capturando até mesmo dados em constante mudança. O espelhamento do banco de dados pode ser usado para aprimorar a disponibilidade ao banco de dados de publicação. O espelhamento compreende duas cópias de um único banco de dados que, geralmente, reside em computadores diferentes. Em determinado momento, apenas uma cópia do banco de dados está atualmente disponível aos clientes. Essa cópia é conhecida como o banco de dados principal. As atualizações realizadas pelos clientes no banco de dados principal são aplicadas à outra cópia do banco de dados, conhecida como banco de dados espelho. O espelhamento envolve a aplicação do *log* de transações de cada inserção, atualização ou exclusão efetuada no banco de dados principal, para o banco de dados espelho.

Observação

DataOps é um método de gerenciamento de dados que enfatiza a comunicação, a colaboração, a integração, a automação e a medição de cooperação entre engenheiros de dados, cientistas e outros profissionais de dados.

Exigências e considerações no uso de replicação com espelhamento do banco de dados

Segundo a Microsoft, antes de executar uma replicação com espelhamento do banco de dados devemos ficar atentos às considerações a seguir:

- O principal e o espelho devem compartilhar um distribuidor. Recomendamos que seja um distribuidor remoto que ofereça tolerância maior a falhas, caso o publicador tenha um *failover* não programado.

- A replicação dá suporte ao espelhamento do banco de dados de publicação para a replicação de mesclagem e para a replicação transacional com assinantes somente leitura ou assinantes de atualização em fila. Não há suporte para assinantes de atualização imediata, publicadores Oracle, publicadores em uma topologia ponto a ponto e republicação.

- Os metadados e os objetos que existem fora do banco de dados não são copiados para o espelho, inclusive *logons*, trabalhos, servidores vinculados etc. Se precisar dos metadados e dos objetos no espelho, será preciso copiá-los manualmente.

Observação

Failover, em computação, significa tolerância a falhas. Quando um sistema, servidor ou outro componente de *hardware* ou *software* fica indisponível, um componente secundário assume operações sem que haja interrupção nos serviços. Sistemas de alta disponibilidade (HA) são tolerantes a falhas, ou seja, são construídos com equipamentos/aplicações redundantes e, caso um dos componentes fique indisponível por falha ou manutenção, nenhum serviço é interrompido.

6 NOÇÕES SOBRE TECNOLOGIAS DE ACESSO AO BANCO DE DADOS

Atualmente, os profissionais de tecnologia possuem muitas ferramentas comerciais de mineração de dados disponíveis para utilização. Essas ferramentas são usadas para diversas técnicas comuns para extrair conhecimento. Entre elas existem regras de associação, agrupamento, redes neurais, sequenciação e análise estatística. Também são usadas árvores de decisão, que são uma representação das regras utilizadas na classificação ou agrupamento, e análises estatísticas, que podem incluir regressão e muitas outras técnicas. Outros produtos comerciais utilizam técnicas avançadas, como algoritmos genéticos, lógica baseada em caso, redes bayesianas, regressão não linear, otimização combinatória, combinação de padrão e lógica Fuzzy. Discutiremos sobre alguns deles adiante.

6.1 ODBC

A grande maioria das ferramentas de mineração de dados utilizam a interface ODBC (*open database connectivity*). ODBC é um padrão da indústria que funciona com bancos de dados, pois ele permite o acesso aos dados na maioria dos programas de banco de dados populares, como Access, dBASE, MySQL, Oracle e SQL Server. Alguns desses pacotes de *software* oferecem interfaces para programas específicos de banco de dados, sendo que os mais comuns são Oracle, Access e SQL Server. A maior parte das ferramentas funciona no ambiente Microsoft Windows e algumas no sistema operacional Unix. A tendência é que todos os produtos operem no ambiente Microsoft Windows.

Em geral, esses programas realizam processamento sequencial em uma única máquina. Muitos desses produtos atuam no modo cliente-servidor. Alguns deles incorporam o processamento paralelo em arquiteturas de computador paralelas e atuam como uma parte das ferramentas de processamento analítico *on-line* (OLAP).

Lembrete

O *driver* ODBC é fornecido com ferramentas como sqlcmd (utilitário do sqlcmd, o qual permite a inserção de instruções Transact-SQL) e bcp (utilitário bcp – *bulk copy program*), que copiam dados em massa entre uma instância do Microsoft SQL Server. O utilitário bcp copia dados em massa entre uma instância do Microsoft SQL Server e um arquivo de dados no formato especificado pelo usuário. O utilitário bcp pode ser usado para importar um grande número de linhas novas em tabelas do SQL Server ou para exportar dados de tabelas para arquivos de dados.

Interface com o usuário

A maioria das ferramentas é executada em um ambiente de interface gráfica com o usuário (GUI, do inglês *graphical user interface*). Alguns produtos incluem técnicas de visualização sofisticadas para exibir dados e regras (por exemplo, MineSet da SGI) e são até capazes de manipular dados assim interativamente. As interfaces de texto são raras e mais comuns em ferramentas disponíveis para Unix, como o Intelligent Miner da IBM.

Interface de programação de aplicações

Normalmente, a interface de programação de aplicações (API) é uma ferramenta opcional. A maioria dos produtos não permite o uso de suas funções internas. Porém, alguns deles permitem que o programador de aplicação reutilize seu código. As interfaces mais comuns são bibliotecas C e dynamic link libraries (DLLs). Algumas ferramentas incluem linguagens próprias de comando de banco de dados. No quadro adiante, listamos 11 ferramentas de mineração de dados representativas.

Direções futuras

As ferramentas de mineração de dados estão continuamente evoluindo, com base nas ideias da pesquisa científica mais recente. Muitas dessas ferramentas incorporam os algoritmos mais recentes tomados da inteligência artificial (IA), estatística e otimização. Atualmente, o processamento rápido é feito usando técnicas modernas de banco de dados — como o processamento distribuído — em arquiteturas cliente-servidor, em bancos de dados paralelos e em *data warehouse*. Para o futuro, a tendência é em direção ao desenvolvimento de capacidades de internet mais completas. Além disso, abordagens híbridas se tornarão comuns e o processamento será feito usando todos os recursos disponíveis. Existe uma necessidade definitiva de incluir dados fora do padrão, inserindo imagens e outros dados de multimídia, como dados de origem para mineração de dados ambientes de computação paralelo e distribuído. Essa mudança é especialmente importante porque os bancos de dados modernos contêm uma quantidade de informação muito grande. Não apenas os bancos de dados de multimídia estão crescendo, mas também o armazenamento e a recuperação de imagens são operações lentas. Além do mais, o custo do armazenamento secundário está diminuindo, de modo que o armazenamento maciço de informações

será viável até mesmo para pequenas empresas. Assim, os programas de mineração de dados terão de lidar com conjuntos de dados maiores de outras empresas. A maioria dos *softwares* de mineração de dados usará o padrão ODBC para extrair dados de bancos de dados comerciais.

Quadro 18 – Ferramentas de mineração de dados representativas

Empresa	Produto	Técnica	Plataforma	Interface
AcknoSoft	Kate	Árvores de decisão, raciocínio baseado em caso	Windows UNIX	Microsoft Access
Angoss	Knowledge SEEKER	Árvores de decisão, estatística	Windows	ODBC
Business Objects	Business Miner	Redes neurais, aprendizado de máquina	Windows	ODBC
CrossZ	QueryObject	Análise estatística, algoritmo de otimização	Windows MVS UNIX	ODBC
Data Distilleries	Data Surveyor	Abrangente, pode misturar diferentes tipos de mineração de dados	UNIX	ODBC compatível com ODMG
DBMiner Technology Inc.	DBMiner	Análise OLAP, associações, classificação, algoritmos de agrupamento	Windows	Microsoft 7.0 OLAP
IBM	Intelligent Miner	Classificação, regras de associação, modelos de previsão	UNIX (AIX)	IBM DB2
Megaputer Intelligence	PolyAnalyst	Aquisição de conhecimento simbólico, programação evolucionária	Windows OS/2	ODBC Oracle DB2
NCR	Management Discovery Tool (MDT)	Regras de associação	Windows	ODBC
Purple Insight	MinseSet	Árvores de decisão, regras de associação	UNIX (Irix)	Oracle Sybase Informix
SAS	Enterprise Miner	Árvores de decisão, redes neurais, regressão, agrupamento	UNIX (Solaris) Windows Macintosh	ODBC Oracle AS/400

6.2 JDBC

Em uma SQL embutida, a integração da SQL com uma linguagem de programação de propósito geral é uma prática muito convencional, um pré-processador específico do SGBD transforma os comandos da SQL embutida em chamadas de funções na linguagem hospedeira (C++, C#, Java, Python etc). Os detalhes dessa tradução variam com os SGBDs e, assim, mesmo que o código-fonte possa ser compilado para funcionar com SGBDs diferentes, o executável final funciona apenas com um SGBD específico. ODBC e JDBC – abreviações para *open database connectivity* (conectividade aberta a banco de dados) e *Java database connectivity* (conectividade Java a banco de dados) – também permitem a integração da

SQL com uma linguagem de programação de propósito geral. Ambos, ODBC e JDBC, expõem os recursos de banco de dados de uma forma padronizada ao programador de aplicativo através de uma interface de programação de aplicativo (API – *application programming interface*).

Em contraste à SQL embutida, o ODBC e o JDBC possibilitam que um único executável acesse diferentes SGBDs sem recompilação. Assim, enquanto a SQL embutida é independente do SGBD somente no nível do código, os aplicativos usando ODBC ou JDBC são independentes do SGBD no nível do código-fonte e no nível do executável. Além disso, usando o ODBC e o JDBC um aplicativo pode acessar não apenas um SGBD, mas diversos SGBDs simultaneamente.

O ODBC e o JDBC adquirem portabilidade no nível do executável pela introdução de um nível extra de vias indiretas. Toda interação direta com um SGBD específico acontece através de um *driver* específico do SGBD. Um *driver* é um programa de *software* que traduz as chamadas ODBC ou JDBC em chamadas específicas do SGBD. Os *drivers* são carregados dinamicamente sob demanda, uma vez que os SGBDs que os aplicativos acessarão são conhecidos apenas em tempo de execução. Os *drivers* disponíveis são registrados através de um gerenciador de *drivers*. Um ponto interessante a se observar é que um *driver* não necessariamente precisa interagir com um SGBD que entende SQL. É suficiente que o *driver* traduza os comandos SQL do aplicativo em comandos equivalentes que o SGBD entenda. Assim, no restante desta seção, vamos nos referir a um subsistema de armazenamento de dados com o qual um *driver* interage como uma fonte de dados.

Um aplicativo que interage com uma fonte de dados através de ODBC ou JDBC seleciona uma fonte de dados, carrega dinamicamente o *driver* correspondente e estabelece uma conexão com a fonte de dados. Não há limites para o número de conexões abertas e um aplicativo pode ter diversas conexões abertas para diferentes fontes de dados. Cada conexão tem semântica de transação, ou seja, as alterações de uma conexão são visíveis a outras conexões apenas após a conexão ter consolidado suas alterações. Enquanto uma conexão estiver aberta, as transações são executadas submetendo comandos SQL, obtendo resultados, processando erros e, finalmente, consolidando ou abortando. O aplicativo se desconecta da fonte de dados para terminar a interação.

Saiba mais

JDBC é uma API da Sun que permite criar arquitetura de duas e três camadas. Para entender como funciona, leia o artigo a seguir:

OLIVEIRA, M. S. *Desenvolvimento de aplicações de bancos de dados*. [s.d.]. Disponível em: https://bit.ly/3ggJkD8. Acesso em: 13 abr. 2021.

6.2.1 Arquitetura

Quando se trata da arquitetura do JDBC, este pode ser acessado por quatro componentes principais: o aplicativo, o gerenciador de *drivers*, *drivers* específicos para diversas fontes de dados e as fontes de dados correspondentes. O aplicativo inicia e termina a conexão com uma fonte de dados. Ele configura os limites da transação, submete os comandos SQL e obtém os resultados – tudo através de uma interface bem definida conforme especificada pela API do JDBC. O principal objetivo do gerenciador de *drivers* é carregar os *drivers* JDBC e passar as chamadas de funções JDBC do aplicativo para o *driver* correto. O gerenciador de *drivers* também trata as chamadas de inicialização e informação JDBC e as chamadas de informação dos aplicativos, e pode registrar todas as chamadas das funções. Além disso, o gerenciador de *drivers* executa uma verificação de erro bem rudimentar. O *driver* estabelece a conexão com a fonte de dados. A seguir, serão listados os quatro principais componentes:

- **Tipo I – Pontes**: transforma as chamadas de função JDBC em chamadas de funções de outra API que não é nativa do SGBD. Um exemplo é uma ponte JDBC-ODBC; um aplicativo pode usar chamadas JDBC para acessar uma fonte de dados compatível com ODBC. O aplicativo carrega apenas um *driver*, a ponte. As pontes têm a vantagem da facilidade de aproveitar os recursos do aplicativo em uma instalação existente e de nenhum *driver* novo precisar ser instalado. Mas o uso de pontes tem diversas desvantagens. O número aumentado de camadas entre a fonte de dados e o aplicativo afeta o desempenho. Além disso, o usuário é limitado à funcionalidade que o *driver* ODBC suporta.

- **Tipo II – Tradução direta à API nativa via *driver* não Java:** este tipo de *driver* traduz as chamadas de função JDBC diretamente em chamadas de métodos da API de uma fonte de dados específica. O *driver* é normalmente escrito usando uma combinação de C++ e Java, ele é ligado dinamicamente e específico da fonte de dados. Essa arquitetura tem um desempenho significativamente melhor do que uma ponte JDBC-ODBC. Uma desvantagem é a de que o *driver* do banco de dados que implementa a API necessita ser instalado em cada computador que executa o aplicativo.

- **Tipo III – Pontes de rede**: o *driver* "fala" através de uma rede com um servidor intermediário que traduz as solicitações JDBC em chamadas de métodos específicos às solicitações, o *driver* traduz dados, formatos e códigos de erros de um formato que é específico à fonte de dados para o padrão JDBC. A fonte de dados processa os comandos do *driver* e retorna os resultados. Dependendo da localização relativa da fonte de dados e do aplicativo, diversos cenários arquiteturais são possíveis. Os *drivers* em JDBC são classificados em quatro tipos, dependendo do relacionamento arquitetural entre o aplicativo e a fonte de dados do SGBD. Nesse caso, o *driver* do lado do cliente (isto é, a ponte de rede) não é específico do SGBD. O *driver* JDBC carregado pelo aplicativo pode ser bem pequeno, já que a única funcionalidade que ele precisa implementar é o envio de comandos SQL ao servidor intermediário. Este, por sua vez, pode usar um *driver* JDBC tipo II para conectar-se à fonte de dados.

- **Tipo IV – Tradução direta à API nativa via *driver* Java**: em vez de chamar diretamente a API do SGBD, o *driver* comunica-se com o SGBD através de *sockets* Java. Nesse caso, o *driver* do lado do cliente é escrito em Java, mas é específico para o SGBD. Ele traduz as chamadas JDBC em API nativa do sistema de banco de dados. Essa solução não requer uma camada intermediária e, como a implementação é toda feita em Java, seu desempenho é normalmente muito bom.

Classes e interfaces JDBC

O JDBC é uma coleção de classes e interfaces Java que possibilita o acesso a banco de dados por meio de programas escritos em linguagem Java. Ele contém métodos para conectar a uma fonte de dados remota, executar comandos SQL, examinar conjuntos de resultados dos comandos SQL, gerenciar transações e tratar exceções. As classes e interfaces são parte do pacote java.sql. Assim, todos os trechos de código no restante desta seção devem incluir o comando import java.sql. * no início do código; omitiremos esse comando no restante desta seção. O JDBC 2.0 também inclui o pacote javax.sql, o Pacote Opcional Java. O pacote javax.sql acrescenta, entre outras coisas, o recurso de *pool* de conexões (gerenciamento de um conjunto de conexões estabelecidas) e a interface RowSet.

6.3 Conexões nativas

Uma sessão com uma fonte de dados é iniciada através da criação de um objeto Connection. Uma conexão identifica uma sessão lógica com uma fonte de dados; múltiplas conexões dentro do mesmo programa Java podem referenciar diferentes fontes de dados ou a mesma fonte de dados. As conexões são especificadas através de uma URL JDBC, uma URL que usa o protocolo jdbc. Tal URL tem o formato:

jdbc : <subprotocolo> : <outrosParametros>

O exemplo de código, ilustrado na figura a seguir, estabelece uma conexão com um banco de dados Oracle considerando que as *strings userId* e *password* estejam configuradas com valores válidos.

```
String url = "jdbc:oracle:www.store.com:3083";
        Connection connection;
        try {

        Connection connection =
        DriverManager.getConnection
        (url,userId, password);
        }
        catch (SQLException excpt)
        {
            System.out.println
            ( excpt.getMessage());
            return;
        }
```

Figura 89 – Estabelecendo uma conexão com JDBC

Em JDBC, as conexões podem ter diferentes propriedades. Por exemplo, uma conexão pode especificar a granularidade das transações. Se *autocommit* é configurado para uma conexão, então cada comando SQL é considerado sua própria transação. Se *autocommit* está desabilitado, então uma série de comandos que compõem uma transação pode ser consolidada usando o método *commit* () da classe Connection, ou abortada usando o método *rollback* (). A classe Connection tem métodos para

configurar o modo *autocommit* (*Connection.setAutoCommit*) e para obter o modo *autocommit* atual (*getAutoCommit*). Os seguintes métodos são parte da interface Connection e possibilitam configurar e obter outras propriedades:

public int getTransactionisolation () throws SQLException e

public void setTransactionisolation(int 1) throws SQLException .

Estas duas funções obtêm e configuram o nível atual de isolamento das transações tratadas na conexão atual.

6.4 Exemplos de conexão de aplicativos com banco de dados: criação de uma aplicação em Java para cadastro de produtos e com acesso ao SGBD — SQL Server 2019

A linguagem Java possui classes que permitem a conexão com um banco de dados, as quais fazem parte do pacote JDBC (*Java database connectivity*), uma API (*aplication program interface*) que permite a comunicação com diversos sistemas de gerenciamento de banco de dados, como Oracle, MySQL, SQL Server, PostgreSQL, entre outros. Independentemente do banco de dados usado, a linguagem padrão para a manipulação de dados é a SQL (*structured query language*).

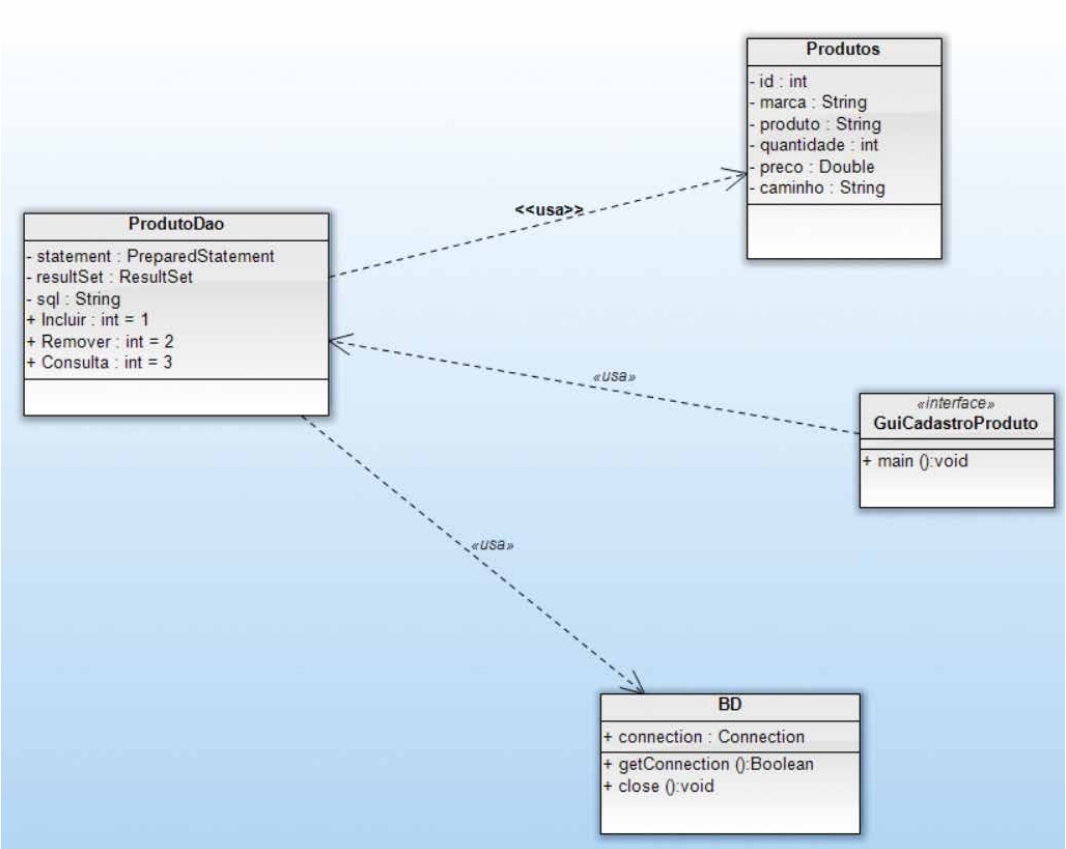

Figura 90 – Diagrama de classe utilizada para representar uma aplicação de cadastro de produtos

Seja qual for o SGBD usado, os passos básicos para a manipulação dos dados por meio de uma aplicação são os seguintes:

1) A criação do banco de dados.

2) Inclusão do *drive* a ser usado para a conexão ao banco de dados.

3) A definição do endereço (URL) da localização do banco de dados que será incluído na aplicação.

4) A criação da aplicação Java propriamente dita para acessar os dados.

Para ilustrar todo o processo será criada uma pequena aplicação que realiza o cadastro de um produto, consulta, alteração e remoção dos dados.

6.4.1 Criação do banco de dados

Os procedimentos a seguir consideram as etapas básicas para a criação do banco de dados por meio do SQL Server.

Abra o SQL Server a partir do botão Iniciar: Iniciar/Todos os programas/Microsoft SQL Server Management Studio 18. Forneça o usuário e senha cadastrados no momento da instalação (veja a figura a seguir que ilustra o acesso ao SQL server), o qual necessita de usuário e senha válida para acessar a conexão.

Figura 91 – Inicializando o SGBD – SQL Server

Abra uma nova consulta e digite as informações conforme vemos a seguir:

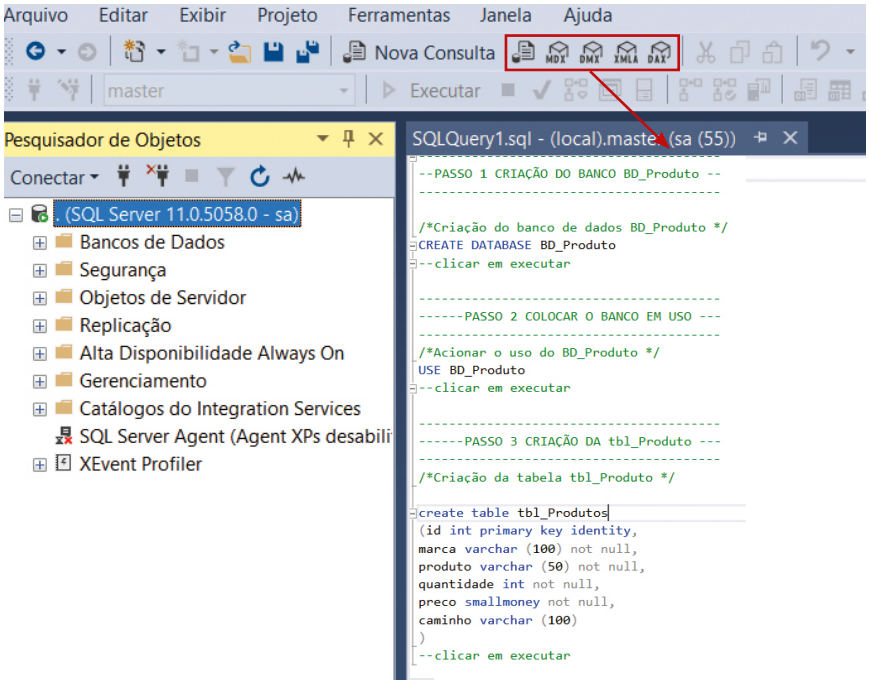

Figura 92 – *Script* com criação do banco de dados BD_Produto

Crie a tabela tbl_Produtos, usando a sintaxe a seguir:

```
create table tbl_Produtos (id int primary key identity, marca varchar (100) not null, produto varchar (50) not null, quantidade int not null, preco smallmoney not null, caminho varchar (100));
```

Ao ser executado o *script* no SQL Server, deve ser apresentada a estrutura que vemos aqui:

	Nome da Coluna	Tipo de Dados	Permitir Nulos
🔑	id	int	☐
	marca	varchar(100)	☐
	produto	varchar(50)	☐
	quantidade	int	☐
	preco	smallmoney	☐
▶	caminho	varchar(100)	☑

Figura 93 – Tabela tbl_Produtos

6.4.2 Definição do *drive* para conexão

Para acessar um banco de dados por meio da plataforma Java, é necessário carregar um *drive* específico do banco de dados (no nosso exemplo foi utilizado o sqljdbc41.jar). Existem diversas versões para os *drives*, que devem ser associadas ao fabricante. Na maioria dos casos, é necessário baixar o *driver* através de *download* (por exemplo: https://bit.ly/3bEDg4c). Após o driver ser carregado ao programa Java, deve-se ainda ser inserido o Class.forName ("pacote.nome-do-drive").

No Microsoft SQL Server 2019, exemplo de *drive* que será utilizado no aplicativo:

```
Class.forName("net.sourceforge.jtds.jdbc.Driver");
```

6.4.3 Criação da aplicação em Java

1) Criação da classe clsConexao em Java. Com a importação das classes java.sql.Connection, java.sql.DriverManager e java.sql.SQLException:

```
package util;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class clsConexao {

    private final String nomeBanco = "BD_Produto";
    private final String usuario = "userID";
    private final String senha = "password";


    public static void main(String[] args) {

        clsConexao c = new clsConexao();
        boolean conectado = c.testarConexao();
        if(conectado)
        {
            System.out.println
            ("Sucesso : Conectou no banco");
        }
    }

    public Connection getConnection()
            throws ClassNotFoundException,
            SQLException {

        Class.forName("net.sourceforge.jtds.jdbc.Driver");
        Connection con = DriverManager.getConnection
       ("jdbc:jtds:sqlserver://localhost:1433/"+nomeBanco, usuario, senha);

        return con;
    }

    public boolean  testarConexao()
    {
        try {
            Connection con = getConnection();
            if(con != null){
                return true;
            }else
                return false;
        } catch (ClassNotFoundException | SQLException e) {
            System.out.println(e.getMessage());
            return false;
        }
    }
}
```

Figura 94 – Classe clsConexao

2) Criação da classe clsImagem em Java, a qual deverá guardar as imagens dos produtos cadastrados, conforme figura a seguir.

```
import java.awt.Image;
import java.io.File;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JFileChooser;
import javax.swing.JLabel;
import javax.swing.filechooser.FileNameExtensionFilter;


public class clsImagem {

        public clsImagem()
        {


        }
        public void loadImageToLabel(JLabel lblImagem, String pathFile)
        {
            File file = new File(pathFile);
            if(file.exists())
            {
                System.out.println("IMAGEM BUSCADA: "+pathFile);
                ImageIcon myImage = new ImageIcon(pathFile);
                Image img = myImage.getImage();
                Image newImg = img.getScaledInstance(lblImagem.getWidth(), lblImagem.getHeight(), Image.SCALE_SMOOTH);
                lblImagem.setIcon(new ImageIcon(newImg));
            }
        }

        //retorna o caminho selecionado
        public String openDialogAndLoadImageSelectedToLabel(JLabel lblImagem){
        JFileChooser jfc = new JFileChooser();
        jfc.setCurrentDirectory(new File(System.getProperty("user.dir")));//setando o diretiorio a iniciar
        jfc.setFileFilter(new FileNameExtensionFilter("Images file", ImageIO.getReaderFileSuffixes())); //bmp jpg jpeg wbmp png gif
        jfc.setAcceptAllFileFilterUsed(true);//removo o filtro de Todos Arquivos

        //se a caixa de dialogo foi confirmada
        if(jfc.showOpenDialog(new javax.swing.JPanel()) == JFileChooser.APPROVE_OPTION)
        {
            //pega o arquivo selecionado
            java.io.File f = jfc.getSelectedFile();

            loadImageToLabel(lblImagem, f.toString());
            lblImagem.setHorizontalAlignment(javax.swing.JLabel.CENTER);

            return f.toString();
        }else
        {
            return null;
        }
    }
}
```

Figura 95 – Classe clsImagem

3) Criação da classe Cadastrar, a qual deverá possuir os componentes, conforme interface da figura a seguir. Os quais estão alinhados aos dados criados no BD_Produto.

Figura 96 – Tela Cadastro de produtos

```
private void btnIncluirActionPerformed
(java.awt.event.ActionEvent evt) {

    if(txtMarca.getText().equals("") || txtProduto.getText().equals("")){
        JOptionPane.showMessageDialog(null, "Nao pode ficar campos vazio");
        return;
    }

    String strValor = ftxtPreco.getText().
            replaceAll("\\.", "").replace(",", ".");
    Double valor =
            strValor.equals("") ? 0 : Double.parseDouble(strValor);

    try {
        Connection con =  new clsConexao().getConnection();
        //adicionando os parametros
        try (PreparedStatement pstmt = con.prepareStatement
                ("insert into tbl_Produtos  VALUES (?, ?, ?, ?, ?)")
        ) {
            //adicionando os parametros
            pstmt.setString(1, txtMarca.getText());
            pstmt.setString(2, txtProduto.getText());
            pstmt.setObject(3, txtQuantidade.getText());
            pstmt.setDouble(4, valor);
            if(lblImagem.getIcon() != null)
            {
                pstmt.setString(5, caminhoImagem);
            }else
                pstmt.setNull(5, java.sql.Types.VARCHAR);
            pstmt.executeUpdate();
        }
        JOptionPane.showMessageDialog(null, "Inserindo com Sucesso");
        limparCampos();
    } catch (ClassNotFoundException | SQLException ex) {
        JOptionPane.showMessageDialog(null, ex.getMessage());
    }
}
```

Figura 97 – Evento botão Incluir

Este foi um exemplo simples de criação de uma aplicação em Java com conexão com um banco de dados SQL Server, todavia existe uma grande quantidade de outras possibilidades, onde poderíamos construir a aplicação em Java usando o modelo MVC (model view controler), entre outras possibilidades.

Resumo

Na unidade III, apresentamos a linguagem SQL, uma linguagem declarativa que permite ao usuário expressar aquilo que se pretende sem ter que entrar em grandes detalhes. A linguagem SQL foi dividida em quatro sublinguagens:

DDL – *data definition language* (CREATE, ALTER, DROP etc.).

DML – *data manipulation language* (SELECT, INSERT, UPDATE, DELETE etc.).

DCL – *data control language* (GRANT, REVOKE etc.).

DTL – *data transaction language* (COMMIT TRANSACTION e ROLLBACK TRANSACTION).

Os sistemas gerenciadores de banco de bados (SGBDs) representam conjuntos de programas que serão utilizados em diversos dispositivos (computadores, *notebooks*, servidores etc.). Seu principal objetivo está em gerenciar o acesso, a manipulação e a organização dos dados da aplicação em uso. O SGBD disponibiliza uma interface para que seus clientes possam incluir, alterar ou consultar dados, entre outras funções. Nesta unidade, mostramos o passo a passo para a instalação do SQL Server 2019 e o SQL Server Management Studio, para a manipulação dos procedimentos da linguagem SQL, bem como das funções de administração, gerenciamento, *backups*, *restore*, monitoramento dos serviços do banco de dados e replicação de dados.

Apresentamos também algumas noções sobre tecnologias de acesso ao banco de dados como a interface ODBC (open database connectivity), uma interface de linguagem de programação que possibilita que os aplicativos acessem dados de uma variedade de DBMS (sistemas de gerenciamento de banco de dados). Além da interface ODBC, vimos o *driver* JDBC (Java database connectivity), usado no SQL Server, que permite a conexão com os aplicativos desenvolvidos em Java por meio das interfaces de programa aplicativo (APIs) JDBC padrão. Além disso, foram apresentadas as principais arquiteturas, conexões nativas e um exemplo de conexão de aplicativo desenvolvido em Java com banco e conexão ao banco de dados SQL Server.

Exercícios

Questão 1. Sabemos que a linguagem SQL é dividida em subgrupos ou sublinguagens, cada uma com usos e características específicos.

Com relação a essas sublinguagens, considere as afirmativas a seguir.

I – O comando INSERT faz parte da DDL (data definition language), ou linguagem de definição de dados.

II – O comando UPDATE faz parte da DML (data manipulation language), ou linguagem de manipulação de dados.

III – O comando CREATE é utilizado para criar objetos no banco de dados, por exemplo tabelas. Ele faz parte da DDL (data definition language), ou linguagem de definição de dados.

É correto o que se afirma em:

A) I, apenas.

B) II, apenas.

C) III, apenas.

D) I e II, apenas.

E) II e III, apenas.

Resposta correta: alternativa E.

Análise das afirmativas

I – Afirmativa incorreta.

Justificativa: o comando INSERT faz parte da DML (data manipulation language), ou linguagem de manipulação de dados, e não da DDL, como diz a afirmativa. A DML é utilizada, por exemplo, para inserir e manipular os registros das tabelas em um banco de dados.

II – Afirmativa correta.

Justificativa: de forma similar ao comando INSERT, o comando UPDATE também faz parte da DML (data manipulation language), ou linguagem de manipulação de dados, uma vez que ele é utilizado para modificar registros nas tabelas de um banco de dados.

III – Afirmativa correta.

Justificativa: os comandos da DDL (data definition language), ou linguagem de definição de dados, estão ligados à criação de objetos do banco de dados em si, e não diretamente aos dados que vão ser armazenados nesses objetos (que são manipulados pela DML). Assim, por exemplo, utilizamos o comando CREATE TABLE, da DDL, para criar uma tabela no banco de dados, enquanto comandos da DML vão inserir, modificar ou apagar esses registros. Observe que o uso da palavra objeto, nesse contexto, não é o mesmo uso da palavra objeto da orientação a objetos.

Questão 2. Considere as afirmativas a seguir sobre a linguagem SQL.

I – A linguagem SQL não apresenta nenhum mecanismo que permita especificar o controle de acesso de usuários específicos às tabelas de um banco de dados relacional.

II – Podemos utilizar o comando UPDATE da DML (data manipulation language), ou linguagem de manipulação de dados, para alterar a estrutura de uma tabela no banco de dados, criando uma nova coluna em uma tabela.

III – Podemos utilizar o comando DROP da DDL (data definition language), ou linguagem de definição de dados, para remover uma tabela de um banco de dados relacional.

É correto o que se afirma em:

A) I, apenas.

B) II, apenas.

C) III, apenas.

D) I e II, apenas.

E) II e III, apenas.

Resposta correta: alternativa C.

Análise das afirmativas

I – Afirmativa incorreta.

Justificativa: podemos utilizar os comandos GRANT e REVOKE da DCL (data control language), ou linguagem de controle de dados, parte da SQL, para definir o acesso ou o bloqueio de usuários aos objetos de um banco de dados, como uma tabela.

II – Afirmativa incorreta.

Justificativa: o comando UPDATE não é utilizado para alterar a estrutura das tabelas em um banco de dados, mas sim para manipular os registros em uma ou mais tabelas. O comando ALTER, da DDL (data definition language), é o comando utilizado para alterar a estrutura de um objeto em um banco de dados, como uma tabela.

III – Afirmativa correta.

Justificativa: para remover uma tabela chamada de FUNCIONARIO, podemos utilizar o seguinte comando: DROP TABLE FUNCIONARIO.