Material de Banco de Dados

Release 2018

Rodrigo Dornel

out 30, 2018

Sumário

> Bad programmers worry about the code. Good programmers worry about data structures and their relationships.
>
> —Linus Torvalds

CAPÍTULO 1

Informações

1.1 Licença

CAPÍTULO 2

Introdução

Este material será usado dentro da disciplina de Banco de Dados do professor Rodrigo Dornel.

Todo material será desenvolvido durante o ano letivo com a colaboração dos alunos.

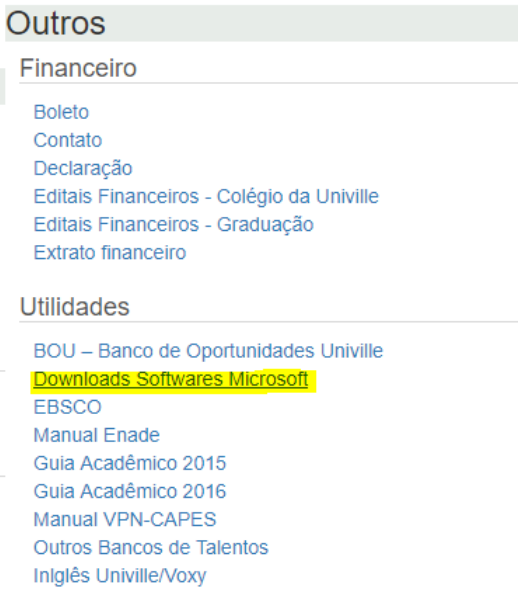
Todo e qualquer conteúdo incluído dentro deste material será avaliado antes de ser publicado.

Todo e qualquer texto, imagem, vídeo ou ainda qualquer conteúdo externo deverá ser referenciado, citando o autor ou proprietário do conteúdo.

CAPÍTULO 3

Download SQL Server 2017 e SSMS

Estando logado com o seu usuário no site da UNIVILLE, selecione a opção “espaço Aluno”, e então “Downloads Softwares Microsoft” conforme imagem abaixo:



Em seguida, logue com as suas credenciais da Univille:

Ao entrar no site da Microsoft, Selecione o “SQL Server 2017 Developer” conforme marcado na imagem a seguir.

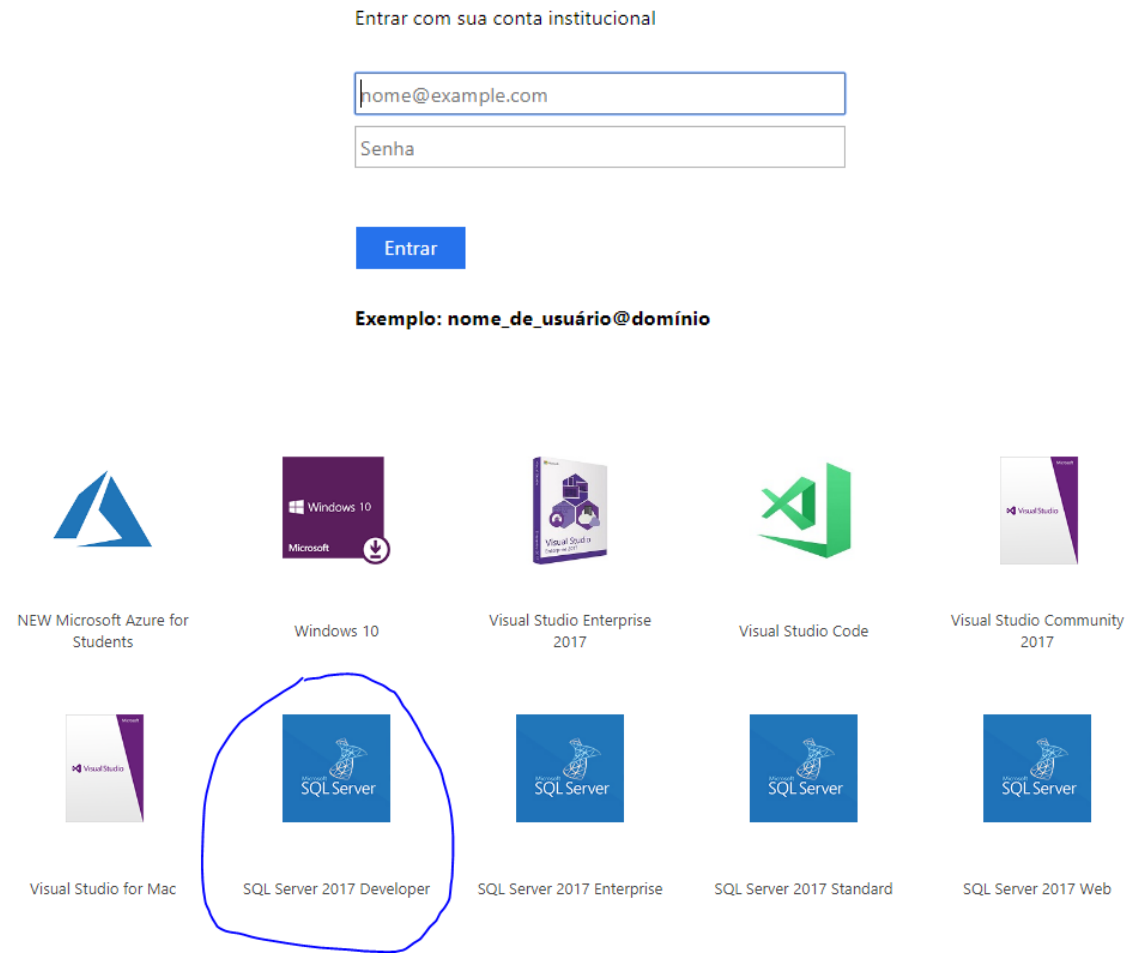
Depois disso, selecione o seu idioma de preferência e prossiga selecionando “Adicionar ao carrinho”.

Após isso, insira seu nome, sobrenome e seu e-mail da Univille nos campos solicitados e clique em “Prosseguir com o pedido”.

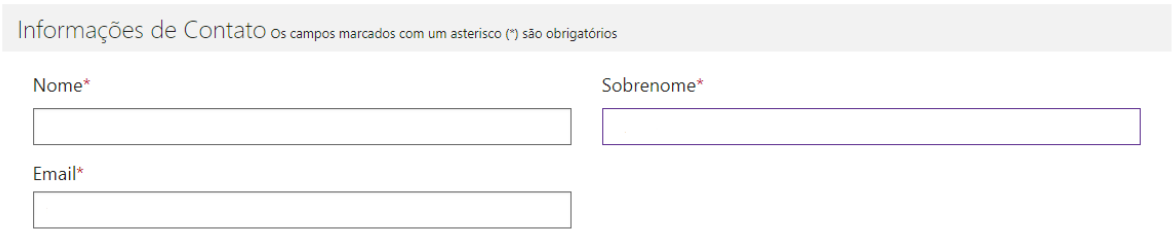
Na tela seguinte, Clique em download.

E novamente, clique em download abaixo do SQL Server 2017 Developer.

Salve o arquivo e espere o término do mesmo. Pode demorar um pouco dependendo da velocidade da sua conexão de internet. O tamanho aproximado é de 1.6GB.


Entrar com sua conta institucional
nome@example.com
Senha
Entrar
Exemplo: nome_de_usuário@domínio
NEW Microsoft Azure for Students
Windows 10
Visual Studio Enterprise 2017
Visual Studio Code
Visual Studio Community 2017
Visual Studio for Mac
SQL Server 2017 Developer
SQL Server 2017 Enterprise
SQL Server 2017 Standard
SQL Server 2017 Web


SQL Server 2017 Developer
SQL Server Developer edition lets developers build any kind of application on top of SQL Server. It includes all the functionality of Enterprise edition, but is licensed for use as a development and test system, not as a production server. SQL Server Developer is an ideal choice for people who build and test applications.
Selecionar um idioma:
Português (Brasil)
SQL Server 2017 Developer 64-bit (Portuguese-Brazil) - Microsoft Imagine
Disponível para: Usuários acadêmicos
Grátis
Adicionar ao carrinho


Informações de Contato Os campos marcados com um asterisco (*) são obrigatórios
Nome*
Sobrenome*
Email*

Itens

		Quantidade
	SQL Server 2017 Developer 64-bit (Portuguese-Brazil) - Microsoft Imagine - Download	Grátis
	Visual Studio Enterprise 2017 32/64-bit (Multilanguage) - Microsoft Imagine - Download	Grátis

Subtotal: Grátis
Tarifas: --
Total: Grátis

Prosseguir Com o Pedido

Downloads

SQL Server 2017 Developer 64-bit (Portuguese-Brazil) - Microsoft Imagine - Download

- Burning the ISO/IMG file onto a disc

Visual Studio Enterprise 2017 32/64-bit (Multilanguage) - Microsoft Imagine - Download

Chave do Produto:

Download

Resumo do Pedido

Número do Pedido:
Data do Pedido: 2018-07-18

Download de Software

Precisa de ajuda?

SQL Server 2017 Developer 64-bit (Portuguese-Brazil)

Download

Tamanho: 1 GB

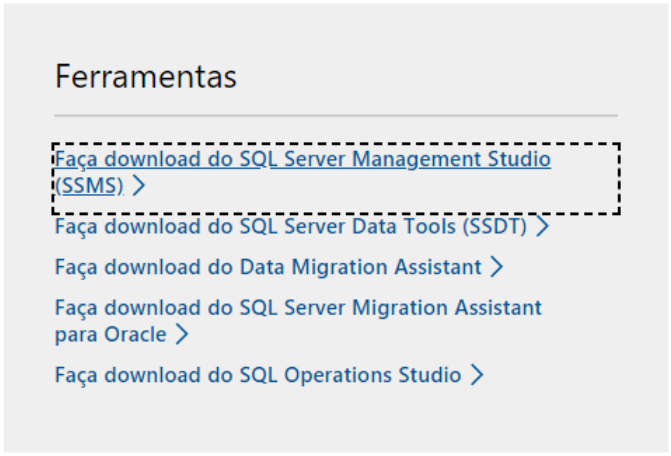
Visual Studio Enterprise 2017 32/64-bit (Multilanguage)

Download

Tamanho: 1 MB

pt_sql_server_2017....iso
0,0/1,6 GB, 18 minutos restant...

Concluindo essa etapa, vamos para o download do SQL Server Management Studio(SSMS). Entre nesse link: https://www.microsoft.com/pt-br/sql-server/sql-server-downloads E clique em "Faça download do SQL Server Management Studio (SSMS)".



Na tela seguinte, clique em "Baixar o SQL Server Management Studio 17.7".

Baixar o SQL Server Management Studio (SSMS)

13/06/2018 • 15 minutos para ler • Colaboradores

ESTE TÓPICO APLICA-SE AO: SQL Server Banco de Dados SQL do Azure SQL Data Warehouse do Azure Parallel Data Warehouse

O SSMS é um ambiente integrado para gerenciar qualquer infraestrutura de SQL, do SQL Server para o Banco de Dados SQL do Microsoft Azure. O SSMS fornece ferramentas para configurar, monitorar e administrar instâncias do SQL. Use o SSMS para implantar, monitorar e atualizar os componentes da camada de dados usados pelos seus aplicativos, além de construir consultas e scripts.

Use o SQL Server Management Studio (SSMS) para consultar, criar e gerenciar seus bancos de dados e data warehouses, independentemente de onde estiverem – no computador local ou na nuvem.

O SSMS é gratuito!

O SSMS 17.X é a última geração do *SQL Server Management Studio* e é compatível com o SQL Server 2017.

Baixar o SQL Server Management Studio 17.7

Baixar o pacote de atualização do SQL Server Management Studio 17.7 (atualiza o 17.x para o

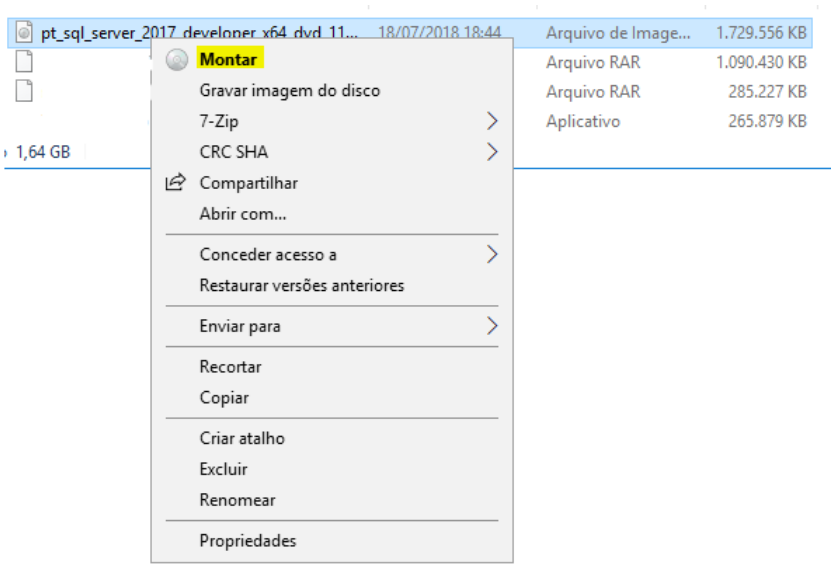
Esse arquivo tem aproximadamente 850MB.

Após o finalização do download, vamos para a instalação do SQL Server.

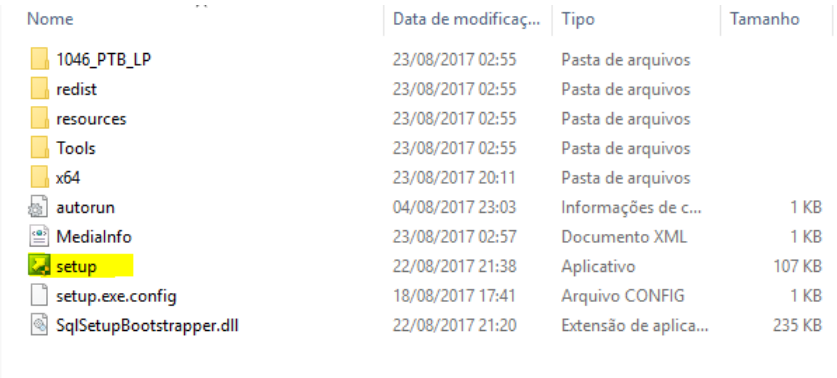
CAPÍTULO 4

Instalar SQL Server 2017

Para dar início à instalação, Clique com o botão direito no arquivo e clique em montar.



Clique em setup:



Espere até aparecer uma janela(pode demorar um pouco), após isso, clique em instalação e depois em “Nova

instalação autônoma".

Deixe a opção "Developer" marcada e clique em avançar.

Após isso, selecione a caixa "Aceito os termos de licença" e clique em avançar.

Espere a conclusão da operação e clique em avançar.

Na seguinte tela, selecione os recursos que você deseja e clique em avançar.

Agora, dê um nome a instância do SQL Server.

Nessa parte, altere as configurações de sua preferência e clique em avançar.

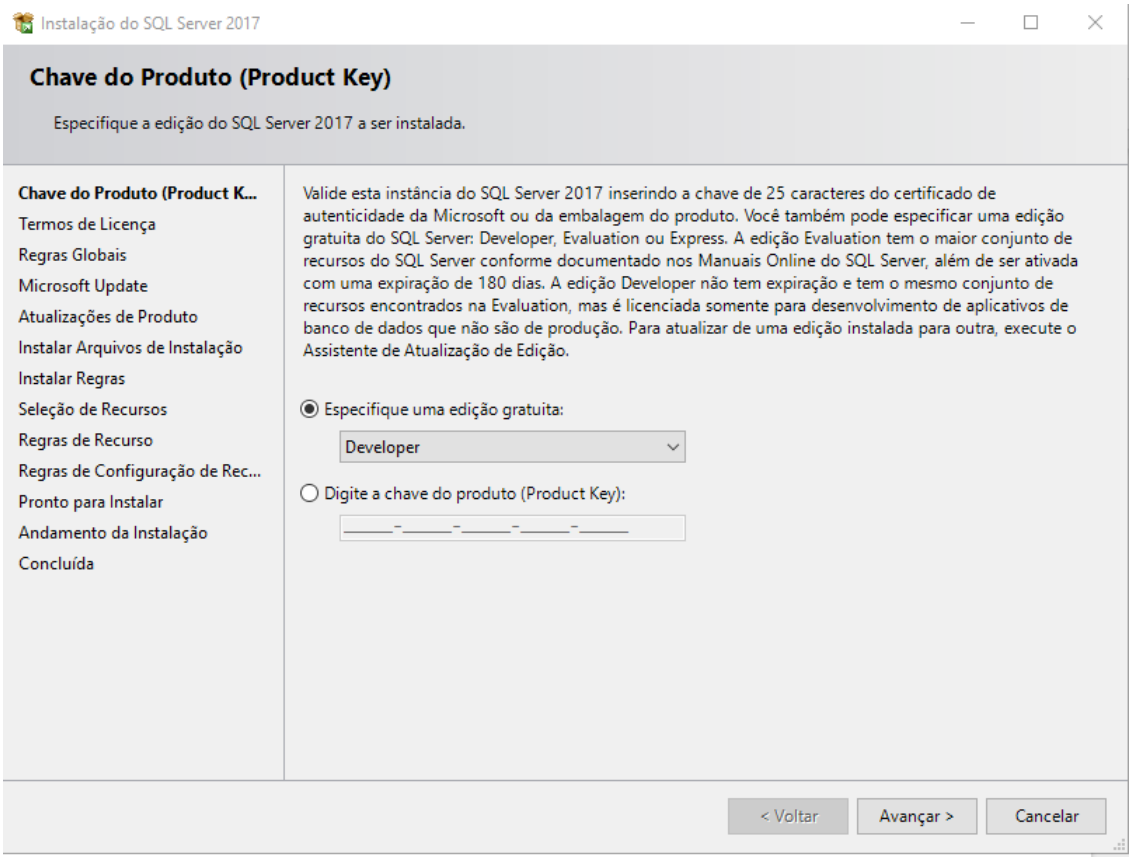
Agora, clique em Adicionar usuário atual e depois em avançar.

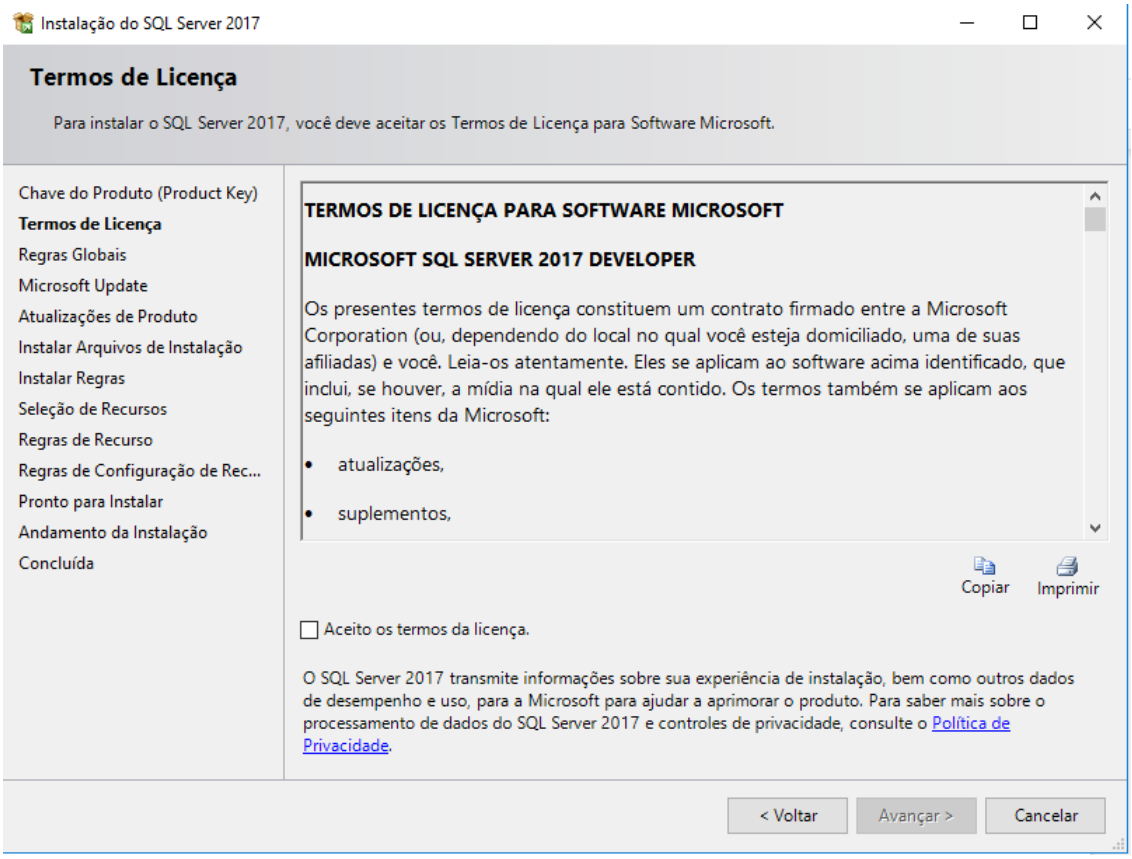
Será fornecido agora um overview de todas as suas configurações, verifique se está de acordo e clique em Instalar.

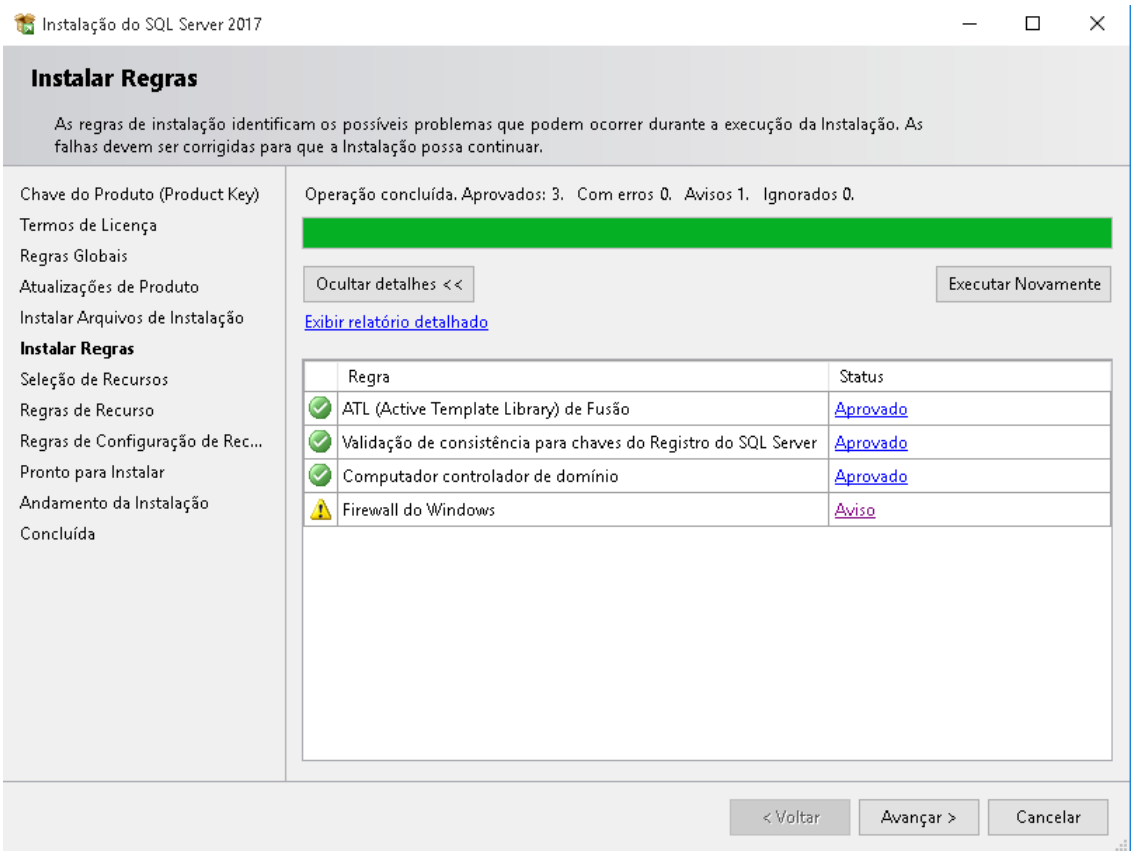
Está sendo instalado o SQL Server agora, espere até o término.

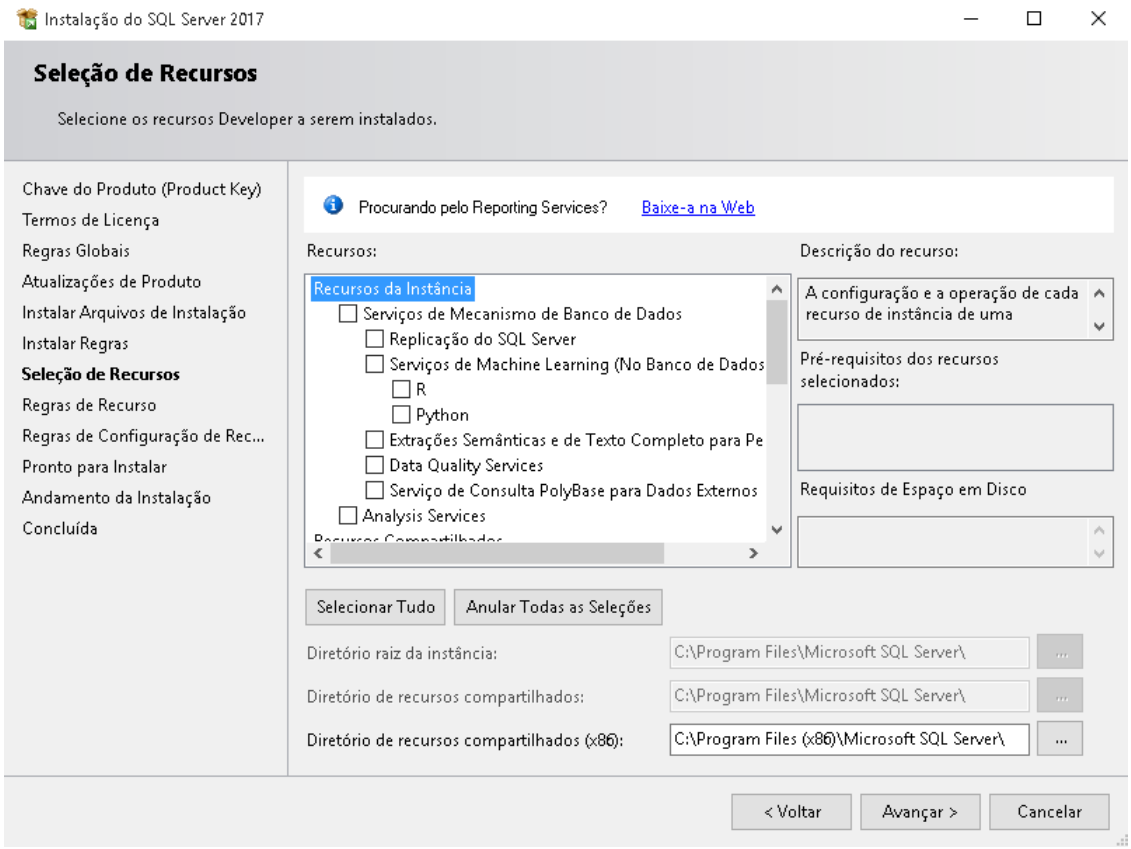
Com isso concluímos a instalação do SQL Server.

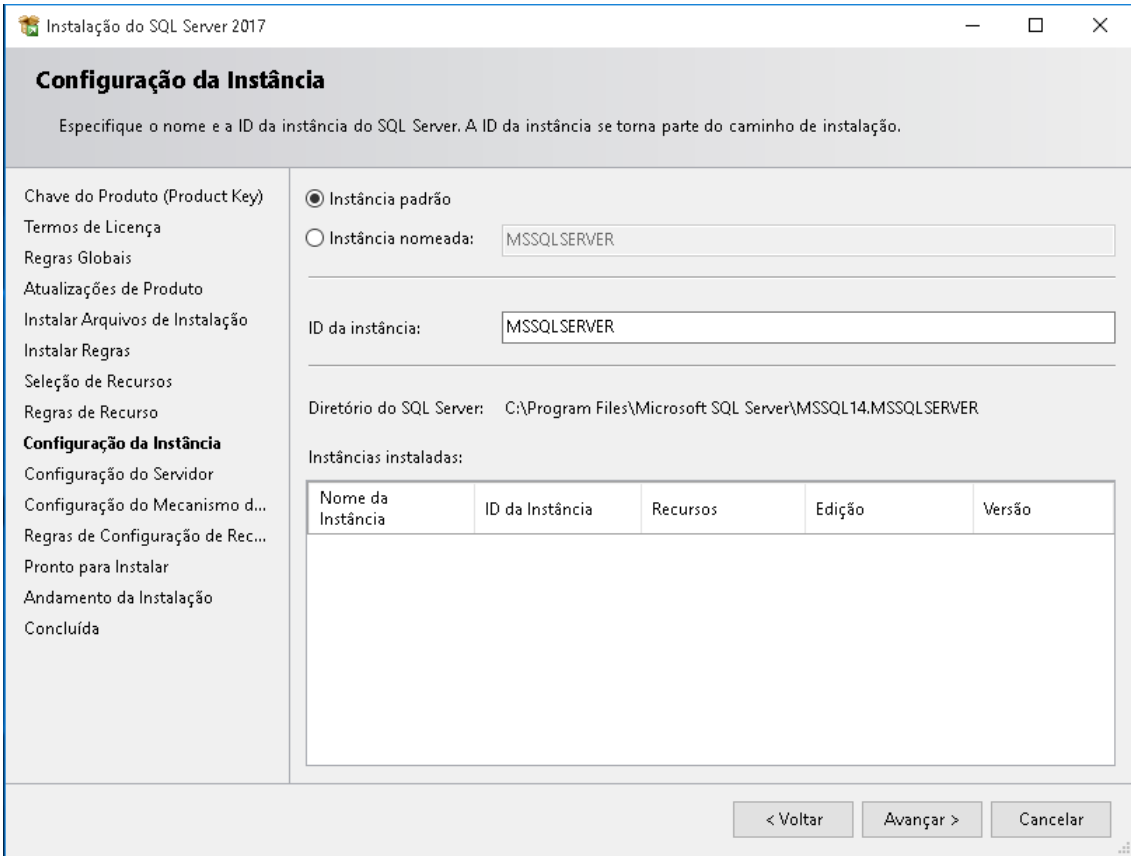
Agora tem a instalação do SSMS.

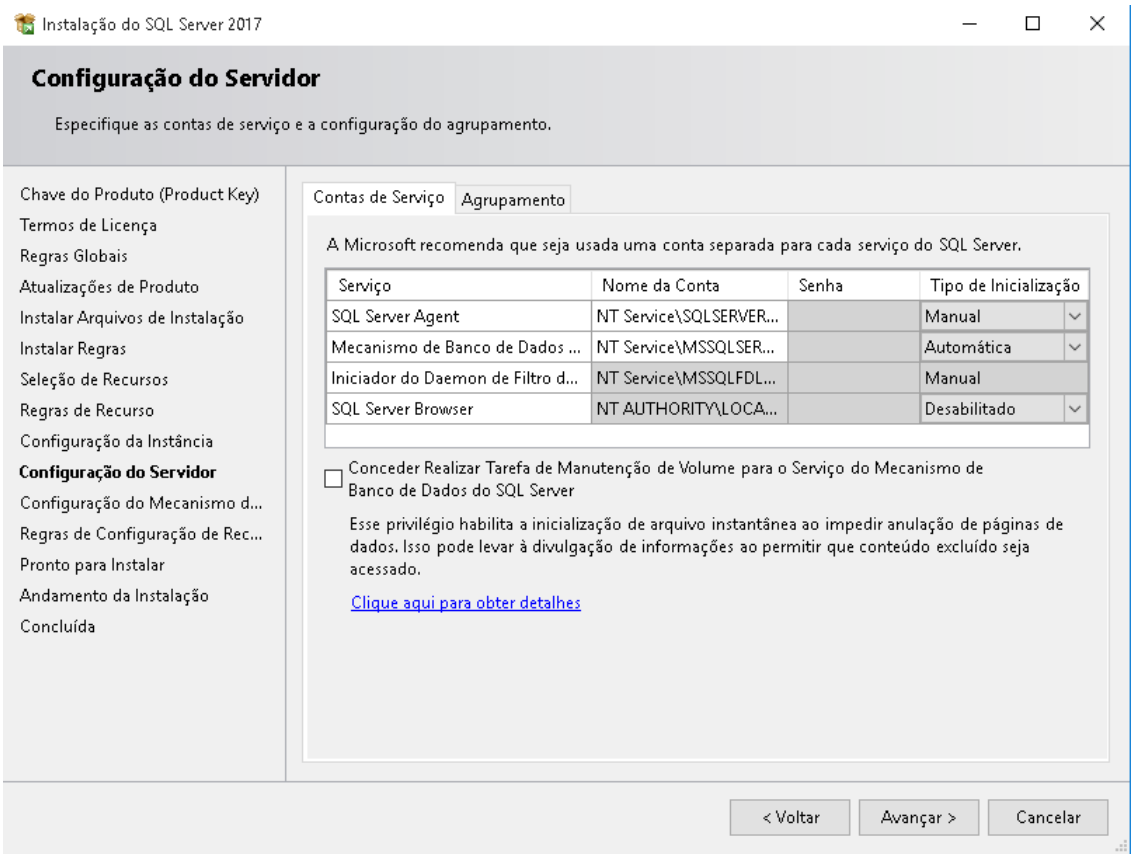

Instalação do SQL Server 2017
Chave do Produto (Product Key)
Especifique a edição do SQL Server 2017 a ser instalada.
Chave do Produto (Product K...
Termos de Licença
Regras Globais
Microsoft Update
Atualizações de Produto
Instalar Arquivos de Instalação
Instalar Regras
Seleção de Recursos
Regras de Recurso
Regras de Configuração de Rec...
Pronto para Instalar
Andamento da Instalação
Concluída
Valide esta instância do SQL Server 2017 inserindo a chave de 25 caracteres do certificado de autenticidade da Microsoft ou da embalagem do produto. Você também pode especificar uma edição gratuita do SQL Server: Developer, Evaluation ou Express. A edição Evaluation tem o maior conjunto de recursos do SQL Server conforme documentado nos Manuais Online do SQL Server, além de ser ativada com uma expiração de 180 dias. A edição Developer não tem expiração e tem o mesmo conjunto de recursos encontrados na Evaluation, mas é licenciada somente para desenvolvimento de aplicativos de banco de dados que não são de produção. Para atualizar de uma edição instalada para outra, execute o Assistente de Atualização de Edição.
Especifique uma edição gratuita:
Developer
Digite a chave do produto (Product Key):
_____-_____-_____-_____-_____
< Voltar
Avançar >
Cancelar

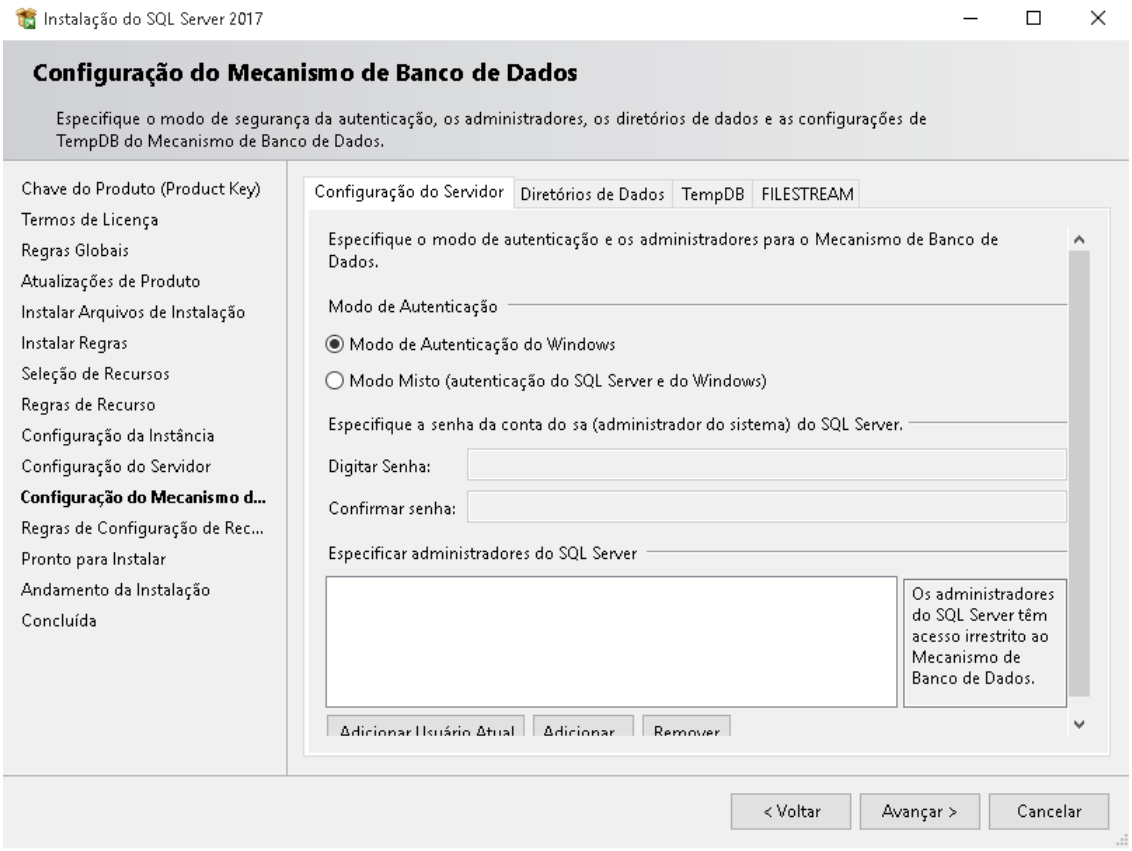

Instalação do SQL Server 2017
Termos de Licença
Para instalar o SQL Server 2017, você deve aceitar os Termos de Licença para Software Microsoft.
Chave do Produto (Product Key)
Termos de Licença
Regras Globais
Microsoft Update
Atualizações de Produto
Instalar Arquivos de Instalação
Instalar Regras
Seleção de Recursos
Regras de Recurso
Regras de Configuração de Rec...
Pronto para Instalar
Andamento da Instalação
Concluída
TERMOS DE LICENÇA PARA SOFTWARE MICROSOFT
MICROSOFT SQL SERVER 2017 DEVELOPER
Os presentes termos de licença constituem um contrato firmado entre a Microsoft Corporation (ou, dependendo do local no qual você esteja domiciliado, uma de suas afiliadas) e você. Leia-os atentamente. Eles se aplicam ao software acima identificado, que inclui, se houver, a mídia na qual ele está contido. Os termos também se aplicam aos seguintes itens da Microsoft:
atualizações,
suplementos,
Copiar
Imprimir
Aceito os termos da licença.
O SQL Server 2017 transmite informações sobre sua experiência de instalação, bem como outros dados de desempenho e uso, para a Microsoft para ajudar a aprimorar o produto. Para saber mais sobre o processamento de dados do SQL Server 2017 e controles de privacidade, consulte o Política de Privacidade.
< Voltar
Avançar >
Cancelar

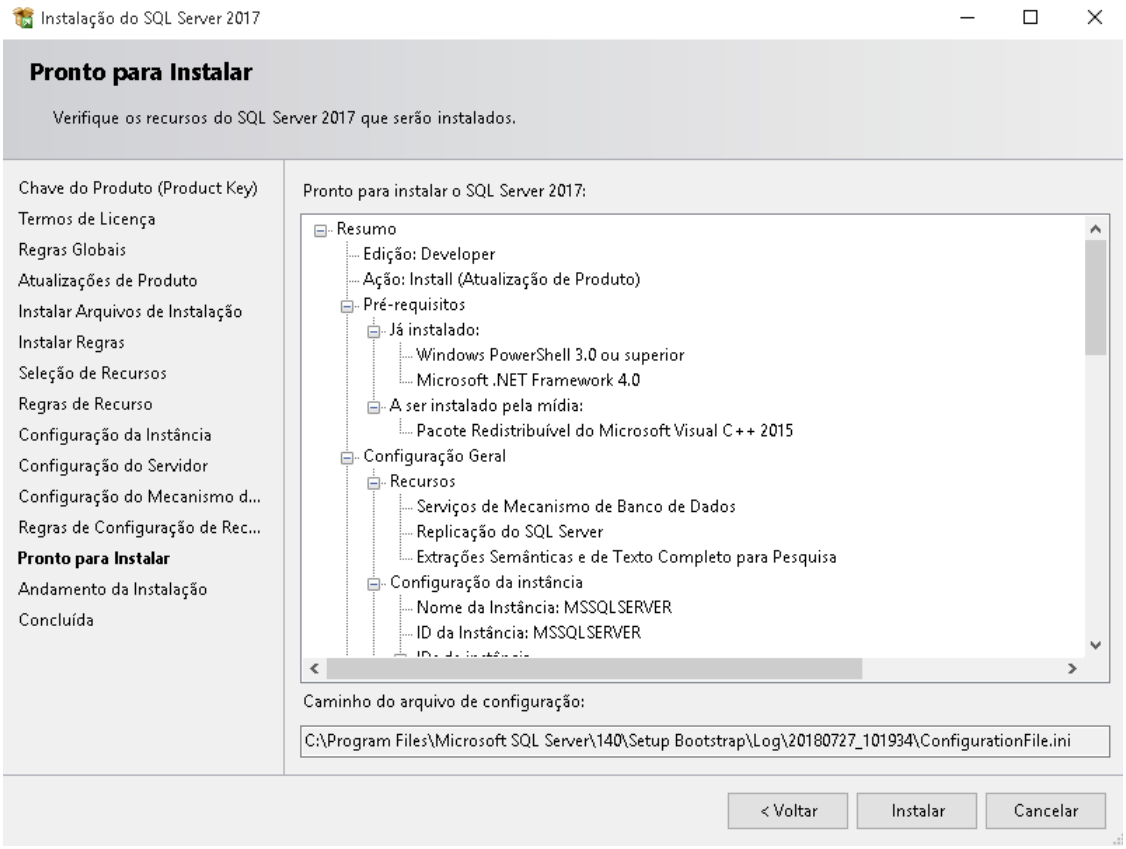

Instalação do SQL Server 2017
Instalar Regras
As regras de instalação identificam os possíveis problemas que podem ocorrer durante a execução da Instalação. As falhas devem ser corrigidas para que a Instalação possa continuar.
Chave do Produto (Product Key)
Termos de Licença
Regras Globais
Atualizações de Produto
Instalar Arquivos de Instalação
Instalar Regras
Seleção de Recursos
Regras de Recurso
Regras de Configuração de Rec...
Pronto para Instalar
Andamento da Instalação
Concluída
Operação concluída. Aprovados: 3. Com erros 0. Avisos 1. Ignorados 0.
Ocultar detalhes <<
Executar Novamente
Exibir relatório detalhado
Regra
Status
ATL (Active Template Library) de Fusão
Aprovado
Validação de consistência para chaves do Registro do SQL Server
Aprovado
Computador controlador de domínio
Aprovado
Firewall do Windows
Aviso
< Voltar
Avançar >
Cancelar

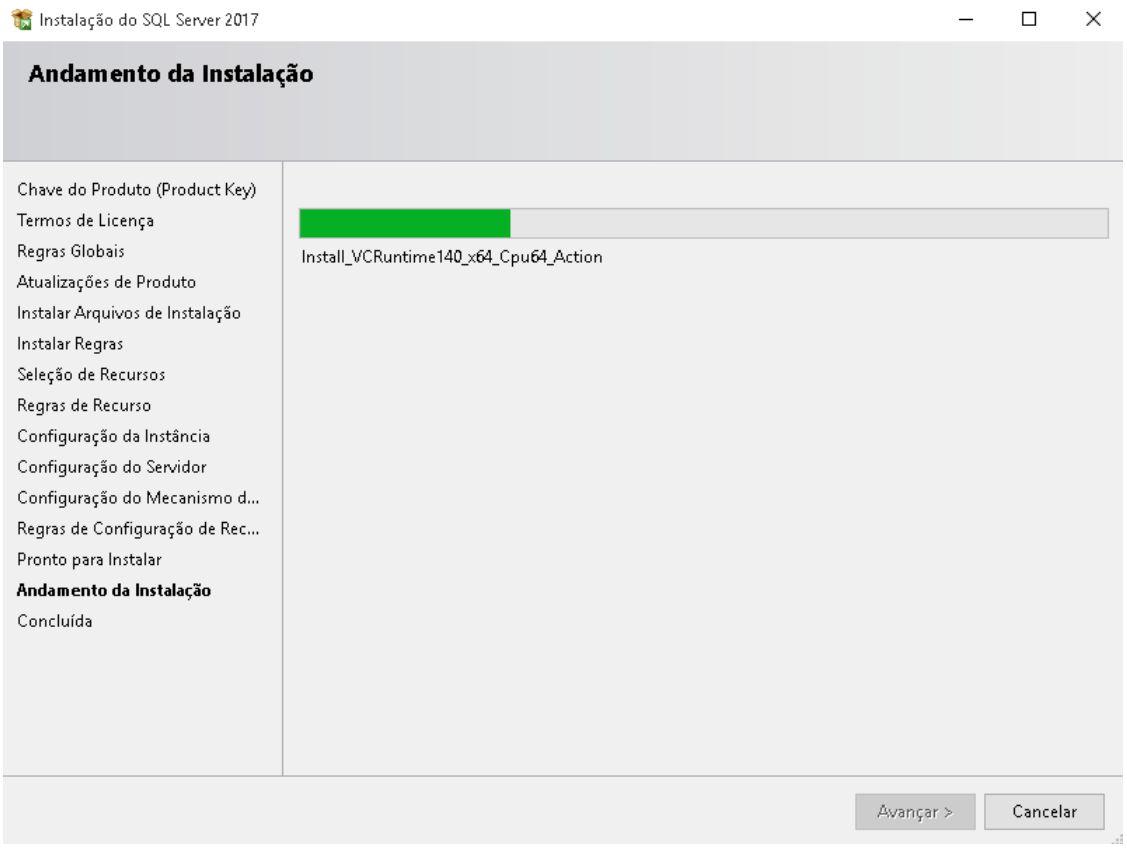

Instalação do SQL Server 2017
Seleção de Recursos
Selecione os recursos Developer a serem instalados.
Chave do Produto (Product Key)
Termos de Licença
Regras Globais
Atualizações de Produto
Instalar Arquivos de Instalação
Instalar Regras
Seleção de Recursos
Regras de Recurso
Regras de Configuração de Rec...
Pronto para Instalar
Andamento da Instalação
Concluída
Procurando pelo Reporting Services?
Baixe-a na Web
Recursos:
Descrição do recurso:
Recursos da Instância
Serviços de Mecanismo de Banco de Dados
Replicação do SQL Server
Serviços de Machine Learning (No Banco de Dados
R
Python
Extrações Semânticas e de Texto Completo para Pe
Data Quality Services
Serviço de Consulta PolyBase para Dados Externos
Analysis Services
A configuração e a operação de cada recurso de instância de uma
Pré-requisitos dos recursos selecionados:
Requisitos de Espaço em Disco
Selecionar Tudo
Anular Todas as Seleções
Diretório raiz da instância:
C:\Program Files\Microsoft SQL Server\
Diretório de recursos compartilhados:
C:\Program Files\Microsoft SQL Server\
Diretório de recursos compartilhados (x86):
C:\Program Files (x86)\Microsoft SQL Server\
< Voltar
Avançar >
Cancelar


Instalação do SQL Server 2017
Configuração da Instância
Especifique o nome e a ID da instância do SQL Server. A ID da instância se torna parte do caminho de instalação.
Chave do Produto (Product Key)
Termos de Licença
Regras Globais
Atualizações de Produto
Instalar Arquivos de Instalação
Instalar Regras
Seleção de Recursos
Regras de Recurso
Configuração da Instância
Configuração do Servidor
Configuração do Mecanismo d...
Regras de Configuração de Rec...
Pronto para Instalar
Andamento da Instalação
Concluída
Instância padrão
Instância nomeada:
MSSQLSERVER
ID da instância:
MSSQLSERVER
Diretório do SQL Server:
C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER
Instâncias instaladas:
Nome da Instância
ID da Instância
Recursos
Edição
Versão
< Voltar
Avançar >
Cancelar


Instalação do SQL Server 2017
Configuração do Servidor
Especifique as contas de serviço e a configuração do agrupamento.
Chave do Produto (Product Key)
Termos de Licença
Regras Globais
Atualizações de Produto
Instalar Arquivos de Instalação
Instalar Regras
Seleção de Recursos
Regras de Recurso
Configuração da Instância
Configuração do Servidor
Configuração do Mecanismo d...
Regras de Configuração de Rec...
Pronto para Instalar
Andamento da Instalação
Concluída
Contas de Serviço
Agrupamento
A Microsoft recomenda que seja usada uma conta separada para cada serviço do SQL Server.
Serviço
Nome da Conta
Senha
Tipo de Inicialização
SQL Server Agent
NT Service\SQLSERVER...
Manual
Mecanismo de Banco de Dados ...
NT Service\MSSQLSER...
Automática
Iniciador do Daemon de Filtro d...
NT Service\MSSQLFDL...
Manual
SQL Server Browser
NT AUTHORITY\LOCA...
Desabilitado
Conceder Realizar Tarefa de Manutenção de Volume para o Serviço do Mecanismo de Banco de Dados do SQL Server
Esse privilégio habilita a inicialização de arquivo instantânea ao impedir anulação de páginas de dados. Isso pode levar à divulgação de informações ao permitir que conteúdo excluído seja acessado.
Clique aqui para obter detalhes
< Voltar
Avançar >
Cancelar


Instalação do SQL Server 2017
Configuração do Mecanismo de Banco de Dados
Especifique o modo de segurança da autenticação, os administradores, os diretórios de dados e as configurações de TempDB do Mecanismo de Banco de Dados.
Chave do Produto (Product Key)
Termos de Licença
Regras Globais
Atualizações de Produto
Instalar Arquivos de Instalação
Instalar Regras
Seleção de Recursos
Regras de Recurso
Configuração da Instância
Configuração do Servidor
Configuração do Mecanismo d...
Regras de Configuração de Rec...
Pronto para Instalar
Andamento da Instalação
Concluída
Configuração do Servidor
Diretórios de Dados
TempDB
FILESTREAM
Especifique o modo de autenticação e os administradores para o Mecanismo de Banco de Dados.
Modo de Autenticação
Modo de Autenticação do Windows
Modo Misto (autenticação do SQL Server e do Windows)
Especifique a senha da conta do sa (administrador do sistema) do SQL Server.
Digitar Senha:
Confirmar senha:
Especificar administradores do SQL Server
Os administradores do SQL Server têm acesso irrestrito ao Mecanismo de Banco de Dados.
Adicionar Usuário Atual
Adicionar
Remover
< Voltar
Avançar >
Cancelar


Instalação do SQL Server 2017
Pronto para Instalar
Verifique os recursos do SQL Server 2017 que serão instalados.
Chave do Produto (Product Key)
Termos de Licença
Regras Globais
Atualizações de Produto
Instalar Arquivos de Instalação
Instalar Regras
Seleção de Recursos
Regras de Recurso
Configuração da Instância
Configuração do Servidor
Configuração do Mecanismo d...
Regras de Configuração de Rec...
Pronto para Instalar
Andamento da Instalação
Concluída
Pronto para instalar o SQL Server 2017:
Resumo
Edição: Developer
Ação: Install (Atualização de Produto)
Pré-requisitos
Já instalado:
Windows PowerShell 3.0 ou superior
Microsoft .NET Framework 4.0
A ser instalado pela mídia:
Pacote Redistribuível do Microsoft Visual C++ 2015
Configuração Geral
Recursos
Serviços de Mecanismo de Banco de Dados
Replicação do SQL Server
Extrações Semânticas e de Texto Completo para Pesquisa
Configuração da instância
Nome da Instância: MSSQLSERVER
ID da Instância: MSSQLSERVER
Caminho do arquivo de configuração:
C:\Program Files\Microsoft SQL Server\140\Setup Bootstrap\Log\20180727_101934\ConfigurationFile.ini
< Voltar
Instalar
Cancelar

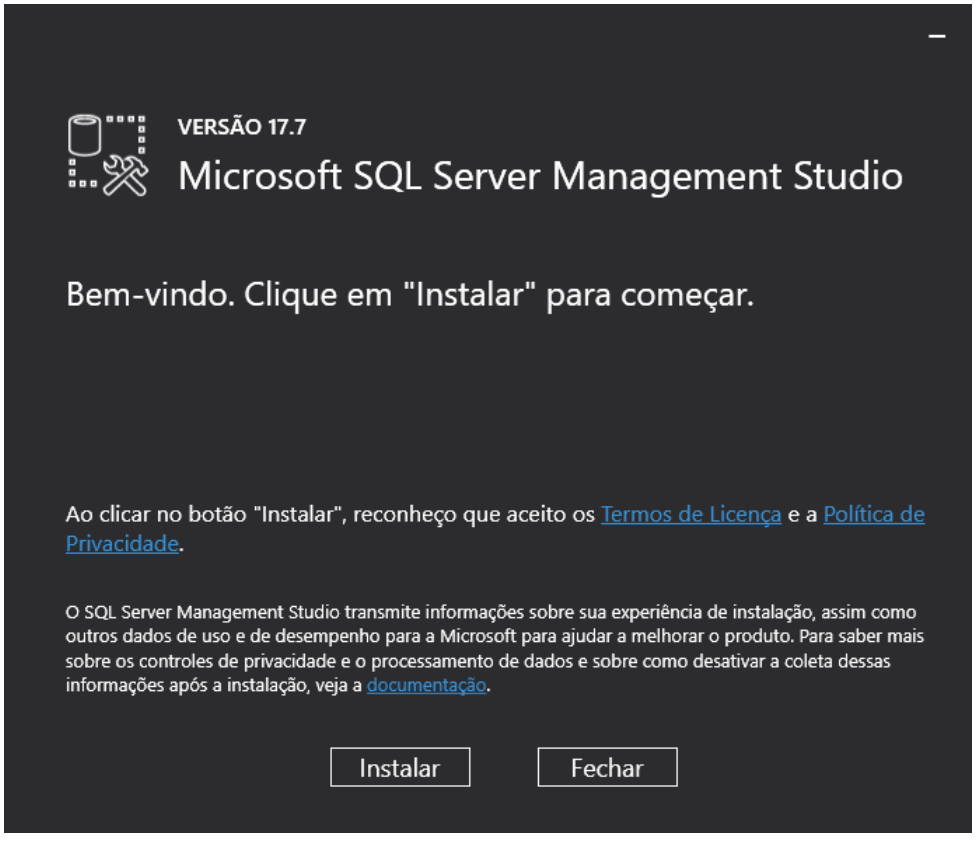

Instalação do SQL Server 2017
Andamento da Instalação
Chave do Produto (Product Key)
Termos de Licença
Regras Globais
Atualizações de Produto
Instalar Arquivos de Instalação
Instalar Regras
Seleção de Recursos
Regras de Recurso
Configuração da Instância
Configuração do Servidor
Configuração do Mecanismo d...
Regras de Configuração de Rec...
Pronto para Instalar
Andamento da Instalação
Concluída
Install_VCRuntime140_x64_Cpu64_Action
Avançar >
Cancelar

CAPÍTULO 5

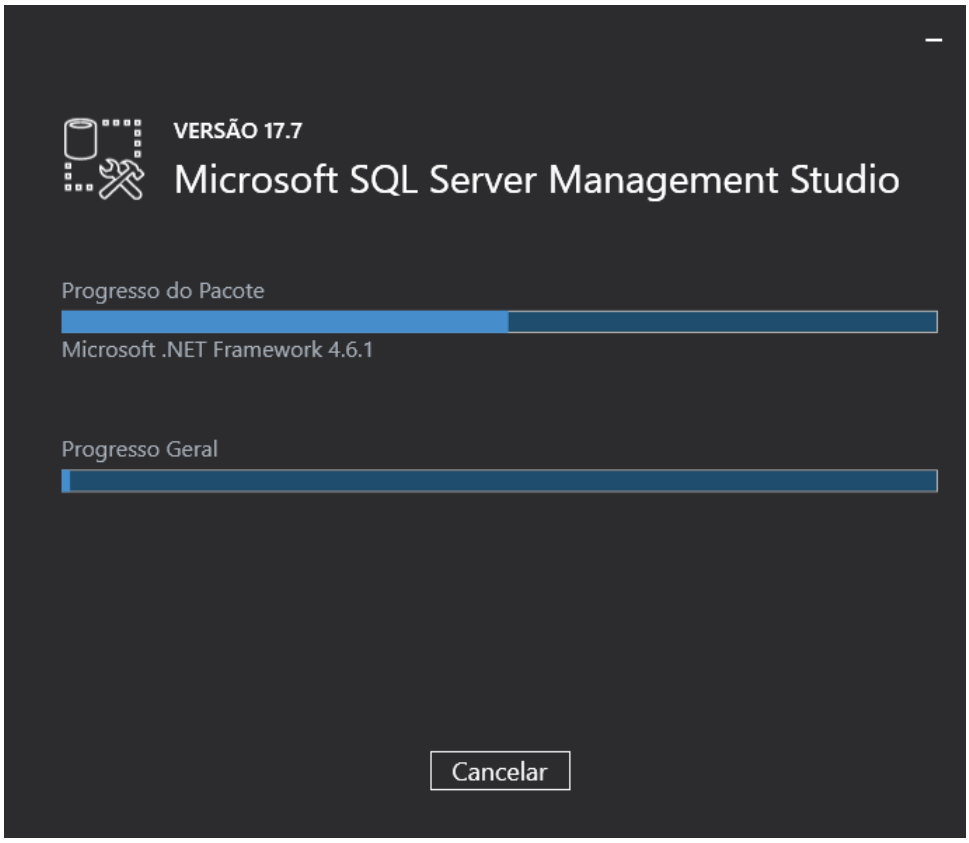
Instalar SSMS

Abra o arquivo SSMS-Setup-PTB.

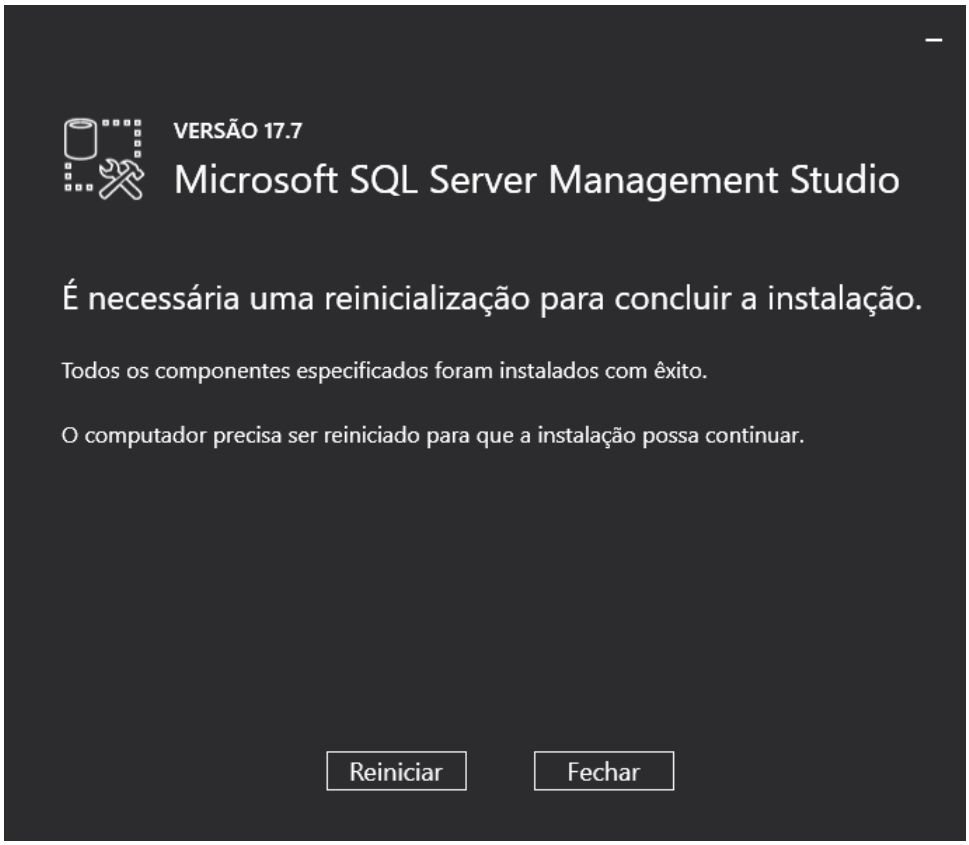
Clique em instalar



Espere.



Após o término da instalação, clique em reiniciar e estará pronto.


VERSÃO 17.7
Microsoft SQL Server Management Studio
É necessária uma reinicialização para concluir a instalação.
Todos os componentes especificados foram instalados com êxito.
O computador precisa ser reiniciado para que a instalação possa continuar.
Reiniciar
Fechar

CAPÍTULO 6

Linguagem SQL

Ótimo local para buscar refências e exemplos de comandos em diversos SGBD´s.

http://www.w3schools.com/sql/

6.1 CREATE

- Comando utilizado para criar os principais objetos em um banco de dados.

Neste tópico vamos trabalhar com as diversas variações do comando `CREATE` relacionados ao início dos trabalhos com criação das entidades no banco de dados.

O Primeiro comando é o `CREATE DATABASE`, que cria o Banco de dados e suas dependências, como arquivos e metadados dentro do sistema. Vale lembrar que alguns sistemas gerenciadores de bancos de dados podem implementar maneiras diferentes de tratar os bancos de dados ou espaços de trabalho de cada usuário ou sistema. Sugiro a leitura do link abaixo, que explica como o Oracle trabalha, ao contrário do SQL Server que vemos em sala de aula.

http://www.oracle.com/technetwork/pt/articles/database-performance/introducao-conceito-de-tablespaces-495850-ptb.html

No nosso banco de dados de Exemplo temos a criação básica de um banco de dados e a criação de uma tabela chamada `Clientes`. Depois usamos o comando use para posicionar a execução dos comandos no banco de dados `MinhaCaixa`.

```
1 CREATE DATABASE MinhaCaixa;
2
3 use MinhaCaixa;
4
5 CREATE TABLE Clientes (
6   ClienteCodigo int,
7   ClienteNome varchar(20)
8 );
```

Podemos ter variações do comando `CREATE TABLE` de acordo com a necessidade. Abaixo temos diversas implementações do comando `CREATE` e suas CONSTRAINT´s.

6.1.1 CONSTRAINT PRIMARY KEY & IDENTITY

Nesse exemplo adicionamos uma chave primária ao campo `ClienteCodigo` e configuramos a propriedade `IDENTITY` que vai gerar um número com incremento de (um) a cada inserção na tabela `Clientes`. Você pode personalizar o incremento de acordo com sua necessidade, neste exemplo temos (1,1) iniciando em um e incrementando um.

```
CREATE TABLE Clientes (
  ClienteCodigo int IDENTITY (1,1) CONSTRAINT PK_Cliente PRIMARY KEY,
  ...
);
```

Nesse exemplo adicionamos uma chave primária composta.

```
CREATE TABLE Clientes (
  ClienteCodigo int IDENTITY (1,1) ,
  ClienteCPF(11)
      CONSTRAINT PK_Cliente PRIMARY KEY (ClienteCodigo,ClienteCPF)
);
```

6.1.2 CONSTRAINT FOREIGN KEY

Neste exemplo temos a criação da `FOREIGN KEY` dentro do bloco de comando `CREATE`. Se tratando de uma chave estrangeira temos que tomar o cuidado de referenciar tabelas que já existem para evitar erros. Repare que no comando abaixo estamos criando uma tabela nova chamada `Contas` e especificando que o código de cliente deverá estar cadastrado na tabela de Cliente, portanto deve existir antes uma tabela `Cliente` que será referenciada nessa chave estrangeira `FOREIGN KEY`. Repare que sempre damos um nome para a `CONSTRAINT`, isso é uma boa prática, para evitar que o sistema dê nomes automáticos.

```
CREATE TABLE Contas
  (
  AgenciaCodigo int,
  ContaNumero VARCHAR (10) CONSTRAINT PK_CONTA PRIMARY KEY,
  ClienteCodigo int,
  ContaSaldo MONEY,
  ContaAbertura datetime
  CONSTRAINT FK_CLIENTES_CONTAS FOREIGN KEY (ClienteCodigo) REFERENCES␣
↪Clientes(ClienteCodigo)
  );
```

6.1.3 ALTER TABLE ADD COLUMN

```
ALTER TABLE Pessoas ADD PessoaSexo CHAR(2);
```

6.1.4 ALTER TABLE ADD CONSTRAINT

Também podemos adiconar CONSTRAINT´s através do comando `ALTER TABLE ... ADD CONSTRAINT`. Geralmente após criar todas as entidades podemos então criar as restrições entre elas.

```
ALTER TABLE Contas ADD CONSTRAINT FK_CLIENTES_CONTAS FOREIGN KEY (ClienteCodigo)
REFERENCES Clientes(ClienteCodigo);
```

6.1.5 CONSTRAINT´s de domínio

```
ALTER TABLE Clientes ADD CONSTRAINT chk_cliente_saldo CHECK ([ClienteNascimento] <
↪GETDATE() AND ClienteNome <> 'Sara');
```

Abaixo a mensagem de tentativa de violação da `CONSTRAINT` acima.

```
The INSERT statement conflicted with the CHECK constraint "chk_cliente_saldo". The
↪conflict occurred in database "MinhaCaixa", table "dbo.Clientes".
```

Apenas checando uma condição, data de nascimento menor que data atual. No SQL Server para pegarmos a data atual usamos `GETDATE()`:

```
ALTER TABLE Clientes ADD CONSTRAINT TESTE CHECK ([ClienteNascimento] < GETDATE());
```

6.2 INSERT

- Comando utilizando para popular as tabelas no banco.

O comando `INSERT` também possui algumas variações que devem ser respeitadas para evitar problemas. O primeiro exemplo abaixo mostra a inserção na tabela `Clientes`. Repare que logo abaixo tem um fragmento da criação da tabela `Clientes` mostando que o campo `ClienteCodigo` é `IDENTITY`, portanto não deve ser informado no momento do `INSERT`.

```
INSERT Clientes (ClienteNome) VALUES ('Nome do Cliente');

CREATE TABLE Clientes
  (
  ClienteCodigo int IDENTITY CONSTRAINT PK_CLIENTES PRIMARY KEY...
```

Quando vamos fazer o `INSERT` em uma tabela que não possui o campo `IDENTITY` passamos o valor desejado, mesmo que o campo seja `PRIMARY KEY`.

```
INSERT Clientes (ClienteCodigo, ClienteNome) VALUES (1, 'Nome do Cliente');

CREATE TABLE Clientes
  (
  ClienteCodigo int CONSTRAINT PK_CLIENTES PRIMARY KEY...

INSERT Clientes (colunas) VALUES (valores);

INSERT INTO Clientes SELECT * FROM ...
```

6.3 UPDATE

- Comando utilizado para alterar registros em um banco de dados. Antes de executar qualquer comando `UPDATE`, procure se informar sobre transações (será abordado mais pra frente).
- Sempre que for trabalhar com o comando `UPDATE` ou `DELETE`, procure executar um `SELECT` antes para validar se os registros que serão afetados, são exatamente aqueles que você deseja.

```
UPDATE CartaoCredito SET CartaoLimite = 1000 WHERE ClienteCodigo = 1;
```

6.4 DELETE

- Comando utilizado para deletes registros em um banco de dados.
- Sempre que for trabalhar com o comando `UPDATE` ou `DELETE`, procure executar um `SELECT` antes para validar se os registros que serão afetados, são exatamente aqueles que você deseja.

```
DELETE FROM CartaoCredito WHERE ClienteCodigo = 1;
```

6.5 SELECT

- Comando utilizado para recuperar as informações armazenadas em um banco de dados.

O comando `SELECT` é composto dos atributos que desejamos, a ou as tabela(s) que possuem esses atributos e as condições que podem ajudar a filtrar os resultados desejados. Não é uma boa prática usar o $\star$ ou *star* para trazer os registros de uma tabela. Procure especificar somente os campos necessários. Isso ajuda o motor de execação de consultas a construir bons planos de execução. Se você conhecer a estrutura da tabela e seus índices, procure tirar proveito disso usando campos chaves, ou buscando e filtrando por atributos que fazem parte de chaves e índices no banco de dados.

```
SELECT * FROM Clientes;
```

- O Comando `FROM` indica a origem dos dados que queremos.

Na consulta acima indicamos que queremos todas as informações de clientes. É possível especificar mais de uma tabela no comando `FROM`, porém, se você indicar mais de uma tabela no comando `FROM`, lembre-se de indicar os campos que fazem o relacionamento entre as tabelas mencionadas na cláusula `FROM`.

- O comando `WHERE` indica quais as consições necessárias e que devem ser obedecidadas para aquela consulta.

Procure usar campos restritivos ou indexados para otimizar sua consulta. Na tabela `Clientes` temos o código do cliente como chave, isso mostra que ele é um bom campo para ser usado como filto.

```
SELECT ClienteNome FROM Clientes WHERE ClienteCodigo=1;
```

- Um comando que pode auxiliar na obtenção de metadados da tabela que você deseja consultar é o comando `sp_help`. Esse comando mostrar a estrutura da tabela, seus atributos, relacionamentos e o mais importante, se ela possui índice ou não.

```
sp_help clientes
```

- Repare que a tabela Clientes possui uma chave no `ClienteCodigo`, portanto se você fizer alguma busca ou solicitar o campo `ClienteCodigo` a busca será muito mais rápida. Caso você faça alguma busca por algum campo que não seja chave ou não esteja "indexado" (Veremos índice mais pra frente) a busca vai resultar em uma varredura da tabela, o que não é um bom negócio para o banco de dados.
- Para escrever um comando `SELECT` procuramos mostrar ou buscar apenas os atributos que vamos trabalhar, evitando assim carregar dados desnecessários e que serão descartados na hora da montagem do formulário da aplicação. Também recomendamos o uso do nome da Tabela antes dos campos para evitar erros de ambíguidade que geralmente aparecem quando usamos mais de uma tabela.

```
SELECT Clientes.ClienteNome FROM Clientes;
```

- Você pode usar o comando `AS` para dar apelidos aos campos e tabelas para melhorar a visualiação e compreensão.

```
SELECT Clientes.ClienteNome AS Nome FROM Clientes;

SELECT C.ClienteNome FROM Clientes AS C;
```

- Você pode usar o operador `ORDER BY` para ordenar os registros da tabela.

Procure identificar os campos da ordenação e verificar se eles possuem alguma ordenação na tabela através de algum índice. As operações de ordenação são muito custosas para o banco de dados. A primeira opção traz os campos ordenados em ordem ascendente `ASC`, não precisando informar o operador. Caso você deseje uma ordenação descendente você deverá informar o `DESC`.

```
SELECT Clientes.ClienteNome FROM Clientes
  ORDER BY Clientes.ClienteNome;

SELECT Clientes.ClienteNome FROM Clientes
  ORDER BY Clientes.ClienteNome DESC;
```

- Outro operador que é muito utilizado em parceria com o `ORDER BY` é o `TOP`, que permite limitar o conjunto de linhas retornado. Caso ele não esteja associado com o `ORDER BY` ele trará um determinado conjunto de dados baseado na ordem em que estão armazenados. Caso você use um operador `ORDER BY` ele mostrará os `TOP` maiores ou menores. O Primeiro exemplo mostra as duas maiores contas em relação ao seu saldo. A segunda, as duas menores.

```
SELECT TOP 2 ContaNumero, ContaSaldo FROM Contas
  ORDER BY ContaSaldo DESC;

SELECT TOP 2 ContaNumero, ContaSaldo FROM Contas
  ORDER BY ContaSaldo;
```

- Podemos usar mais de uma tabela no comando `FROM` como falamos anteriormente, porém devemos respeitar seus relacionamentos para evitar situações como o exemplo abaixo. Execute o comando e veja o que acontece.

```
SELECT * FROM Clientes, Contas;
```

- A maneira correta deve levar em consideração que as tabelas que serão usadas tem relação entre si "chaves", caso não tenham, poderá ser necessário passar por um outra tabela antes. Lembre-se das tabelas associativas.

```
SELECT CLientes.ClienteNome, Contas.ContaSaldo
  FROM Clientes, Contas
  WHERE Clientes.ClienteCodigo=Contas.ClienteCodigo;
```

- O comando `LIKE` é usado para encontrar registros usando parte do que sabemos sobre ele. Por exemplo podemos buscar todas as pessoas que tenham nome começado com `R`, usando um coringa `%` (Percentual). Podemos fazer diversas combinação com o `%`.

Documentação do comando LIKE

```
SELECT ClienteRua FROM dbo.Clientes WHERE ClienteRua LIKE 'a%' AND ClienteRua NOT␣
↪LIKE 'E%';

SELECT ClienteRua FROM dbo.Clientes WHERE ClienteRua LIKE '%a%';

SELECT ClienteRua FROM dbo.Clientes WHERE ClienteRua LIKE '%a';

SELECT ClienteRua FROM dbo.Clientes WHERE ClienteRua NOT LIKE 'a%';
```

- O Comando `CASE` é utilizado quando queremos fazer validações e até gerar novar colunas durante a execução da consulta. No exemplo abaixo fazemos uma classificação de um cliente com base no seu saldo, gerando assim uma nova coluna `Curva Cliente`.

```
SELECT ContaNumero,
  CASE WHEN ContaSaldo < 200 THEN 'Cliente C' WHEN ContaSaldo < 500 THEN 'Cliente B
↪'
  ELSE 'Cliente A' END AS 'Curva Cliente'
  FROM dbo.Contas;
```

- Podemos incluir em nossas consultas diversos operadores condicionais: = (igual), <> (diferente), > (maior), < (menor), <= (menor ou igual), >= (maior ou igual), `OR` (ou), `AND` (e) e `BETWEEN` (entre).

```
SELECT Nome_agencia, Numero_conta, saldo
  FROM Conta
  WHERE saldo > 500 AND Nome_agencia = 'Joinville';

SELECT AgenciaCodigo FROM dbo.Agencias
  WHERE AgenciaCodigo BETWEEN 1 AND 3;
```

- O `ALIAS` ou apelido ajuda na exibição de consultas e tabelas. Dessa forma podemos dar nomes amigáveis para campos e tabelas durante a execução de consultas. Use sempre o `AS` antes de cada `ALIAS`, mesmo sabendo que não é obrigatório.

```
SELECT Nome_agencia,C.Numero_conta,saldo AS [Total em Conta],
    Nome_cliente,D.Numero_conta AS 'Conta do Cliente'
  FROM Conta AS C, Depositante AS D
  WHERE C.Numero_conta=D.Numero_conta AND Nome_cliente IN ('Rodrigo','Laura')
  ORDER BY saldo DESC
```

- O comando `DISTINCT` serve para retirar do retorno da consulta registros repetidos.

```
SELECT DISTINCT Cidade_agencia FROM Agencia;
```

- A SUB CONSULTA, `IN` e `NOT IN` são poderosos recursos para auxiliar em buscas e filtragem de registros. Podemos criar subconjuntos de registros e usar operadores como `IN` para validar se os registros estão dentro daquele subconjunto.

```
SELECT AgenciaCodigo FROM dbo.Agencias
  WHERE AgenciaCodigo NOT IN ('1','4');

SELECT Contas.ContaNumero, Contas.ContaSaldo, Contas.AgenciaCodigo
  FROM Contas INNER JOIN
    (
    SELECT AgenciaCodigo, MAX(ContaSaldo) AS VALOR
    FROM Contas
    GROUP BY AgenciaCodigo
    ) AS TB2
  ON
  TB2.AgenciaCodigo=Contas.AgenciaCodigo AND TB2.VALOR=Contas.ContaSaldo;
```

- Os operadores `UNION` e `UNION ALL` ajudam a consolidar conjuntos de registros que são retornados por consultas distintas. O operador `ALL` faz a junção das consultas sem eliminar itens duplicados. Precisamos obedecer o mesmo número de colunas e tipos de dados entre as consultas.

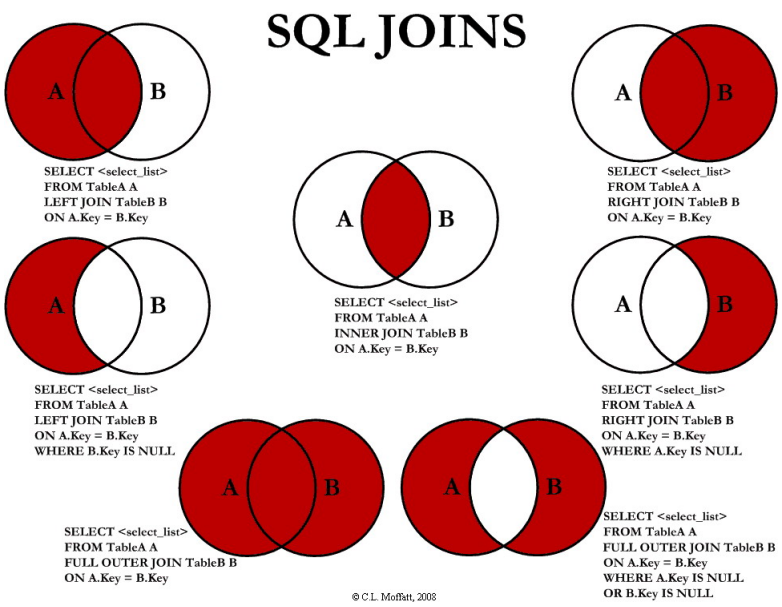
```
SELECT ClienteNome FROM dbo.Clientes WHERE ClienteCodigo = 1
UNION
SELECT ClienteNome FROM dbo.Clientes WHERE ClienteCodigo = 2;

SELECT ClienteNome FROM dbo.Clientes WHERE ClienteCodigo = 1
UNION ALL
SELECT ClienteNome FROM dbo.Clientes WHERE ClienteCodigo = 1;
```

- Existem diversos tipos de `JOINS`. O mais tradicional e restritivo é o `JOIN` ou `INNER JOIN` que requer que o registros usado na comparação exista em ambas as tabelas.

No exemplo abaixo, o `ClienteCodigo` não poderá ser vazio em nenhuma das tabelas envolvidas, caso isso ocorra, aquela linha não será retornada no resultado.

Fonte da imagem: Representação Visual das Joins




```
SELECT * FROM Clientes
  JOIN Contas
    ON Clientes.ClienteCodigo=Contas.ClienteCodigo;

SELECT * FROM CLIENTES
  INNER JOIN Contas
    ON Clientes.ClienteCodigo=Contas.ClienteCodigo;
```

- LEFT JOIN

O comando LEFT indica que todos os registros existentes na tabela da sua esquerda serão retornados e os registros da outra tabela da direita irão ser retornados ou então virão em branco.

```
SELECT ClienteNome, ContaSaldo,
  CASE WHEN CartaoCodigo IS NULL THEN 'LIGAR' ELSE 'NÃO INCOMODAR' END AS
↪'NN'
  FROM Clientes
  INNER JOIN Contas
  ON (Contas.ClienteCodigo = Clientes.ClienteCodigo)
  LEFT JOIN CartaoCredito
  ON (CartaoCredito.ClienteCodigo = Clientes.ClienteCodigo);
```

- RIGHT

Já o comando RIGHT traz todos os registros da tabela da direita e os registos da tabela da esquerda, mostrando em branco aqueles que não tem relação.

```
SELECT * FROM CartaoCredito RIGHT JOIN Clientes ON CartaoCredito.
↪ClienteCodigo=Clientes.ClienteCodigo;
```

- FULL

O comando FULL retorna todos os registros das tabelas relacionadas, mesmo que não exista um correspondente entre elas.

```
SELECT * FROM CartaoCredito FULL OUTER JOIN Clientes ON CartaoCredito.
↪ClienteCodigo=Clientes.ClienteCodigo;
```

- CROSS

Efetua um operação de produto cartesiano, para cada registro de uma tabela ele efetua um relacionamento com os registros das outras tabelas.

```
SELECT * FROM CLIENTES CROSS JOIN Contas;
```

- As FUNÇÕES DE AGREGAÇÃO, `SUM` (soma), `MIN` (mínimo), `MAX` (máximo), `COUNT` (contagem), `AVG` (média), permitem um nível mais robusto de informação, criando conjuntos de dados agrupados, médias entre outros, permitindo o resumo e a totalização de conjuntos de resultados. Sempre que usarmos a função de agregação em conjunto com um campo agregador, devemos usar a função `GROUP BY` para indicar qual o campo será o responsável pelo agrupamento das informações.

Caso você deseje comparar conjuntos de informações contidos na função de agragação você deve compará-los usando o `HAVING`.

```
SELECT TOP 2 AgenciaNome, SUM(ContaSaldo) AS TOTAL
  FROM Contas,  Agencias
  WHERE Agencias.AgenciaCodigo=Contas.AgenciaCodigo
  GROUP BY AgenciaNome
  HAVING SUM(ContaSaldo) > (SELECT MAX(ContaSaldo) AS VALORMETA FROM Contas AS
↪META)
  ORDER BY 2 DESC;

SELECT SUM( Contas.ContaSaldo),
  AgenciaCodigo, ContaNumero
  FROM Contas
  GROUP BY AgenciaCodigo,ContaNumero
  --WHERE COM AVG ???
  --WHERE COM SUBCONSULTA ???
  HAVING SUM( Contas.ContaSaldo) > (SELECT AVG( Contas.ContaSaldo) FROM  Contas); -
↪-667,0833

SELECT MAX(ContaSaldo) FROM  Contas;
SELECT MIN(ContaSaldo) FROM  Contas;
SELECT AVG(ContaSaldo) FROM  Contas;
SELECT COUNT(*), COUNT(CONTAS.ClienteCodigo), COUNT(DISTINCT CONTAS.ClienteCodigo)
↪FROM  Contas;
```

- EXISTS

O comando EXISTS é parecido com o comando `IN`, quando queremos comparar mais de um campo contra uma subconsulta.

```
SELECT * FROM  Contas C
    WHERE EXISTS
                    (SELECT * FROM  CartaoCredito CC
                            WHERE C.ClienteCodigo=CC.ClienteCodigo
                            AND C.AgenciaCodigo=CC.AgenciaCodigo
                    )
```

- FUNÇÕES DE Data e Hora

```
SET DATEFORMAT YDM

SET LANGUAGE PORTUGUESE

SELECT YEAR(getdate()) -YEAR( Clientes.ClienteNascimento),
  DATEDIFF(YEAR,ClienteNascimento,GETDATE()),
  DATEPART(yy,ClienteNascimento),
  dateadd(yy,1,ClienteNascimento),
  EOMONTH(GETDATE()),
  DATENAME(MONTH,(GETDATE()))
FROM  Clientes;
```

```
SELECT * FROM  Contas
  WHERE YEAR(ContaAbertura) = '2011'
  ORDER BY ContaAbertura;
```

- Variáveis

Muitas vezes necessitamos armazenar determinados valores para uso posterior. Um exemplo é guardar um valor total em uma variável para que ele seja usado em cálculo de percentual por exemplo

```
declare @numero int
set @numero = 1

declare @dia int
set @dia = (select day(getdate()))
```

- SELECT INTO

```
      SELECT Clientes.ClienteNome,
      DATEDIFF(YEAR,Clientes.ClienteNascimento,GETDATE()) AS IDADE
      INTO ClientesIdade -- O comando INTO vem depois do campos listados no SELECT␣
↪e antes do FROM.
      FROM Clientes

      SELECT * FROM ClientesIdade
```

- CAST, CONVERT e concatenação

Comandos utilizados para converter tipos de dados e concatenar Strings.

```
SELECT Clientes.ClienteNome + Clientes.ClienteCidade FROM Clientes;

SELECT Clientes.ClienteNome + ' ' + Clientes.ClienteCidade FROM Clientes;

SELECT Clientes.ClienteNome + ' de ' + Clientes.ClienteCidade FROM␣
↪Clientes;

SELECT Clientes.ClienteNome + ' - R$ ' + CAST (Contas.ContaSaldo AS␣
↪VARCHAR(10) )FROM Clientes INNER JOIN Contas ON Contas.ClienteCodigo =␣
↪Clientes.ClienteCodigo;

SELECT Clientes.ClienteNome + ' - R$ ' + CONVERT  (VARCHAR(10), Contas.
↪ContaSaldo )FROM Clientes INNER JOIN Contas ON Contas.ClienteCodigo =␣
↪Clientes.ClienteCodigo;
```

6.6 VIEW

- Comando utilizado para alterar registros em um banco de dados. Antes de executar qualquer comando `UPDATE`, procure se informar sobre transações (será abordado mais pra frente).
- Sempre que for trabalhar com o comando `UPDATE` ou `DELETE`, procure executar um `SELECT` antes para validar se os registros que serão afetados, são exatamente aqueles que você deseja.

```
CREATE VIEW ClientesIdade
AS
SELECT ClienteNome,DATEDIFF(YEAR,ClienteNascimento,GETDATE()) AS Idade         FROM␣
↪dbo.Clientes;
```

6.7 FUNÇÕES

- Uma função é uma sequência de comandos que executa alguma tarefa e que tem um nome. A sua principal finalidade é nos ajudar a organizar programas em pedaços que correspondam a como imaginamos uma solução do problema.

  Exemplo de um Função:

```
CREATE FUNCTION fnRetornaAno (@data DATETIME)
RETURNS int
AS
  BEGIN
  DECLARE @ano int
  SET @ano = YEAR(@data)

      RETURN @ano

END
```

- Chamada ou execução da função

```
SELECT dbo.fnRetornaAno(GETDATE())

SELECT dbo.fnRetornaAno(Clientes.ClienteNascimento) FROM dbo.Clientes
```

6.8 PROCEDURES

- Uma procedure é um bloco de comandos ou instruções SQL organizados para executar uma ou mais tarefas. Ela pode ser utilizada para ser acionada através de uma chamada simples que executa uma série de outros comandos.

```
CREATE PROCEDURE uspRetornaIdade
@CodigoCliente int
AS
SELECT Clientes.ClienteNome, YEAR(GETDATE())-YEAR(ClienteNascimento) AS IDADE
FROM Clientes
INNER JOIN Contas ON Clientes.ClienteCodigo=Contas.ClienteCodigo
WHERE Clientes.ClienteCodigo = @CodigoCliente;
```

- Execução da procedure, opção 1

```
exec uspRetornaIdade 1;
```

- Execução da procedure, opção 2

```
declare @parametro int
set @parametro = 1 --Código do Cliente desejado
exec uspRetornaIdade @parametro;
```

6.8.1 IF

- Comando utilizado para checar condições.

```
CREATE PROCEDURE uspRetornaSeTemCartao
@CodigoCliente int
AS
BEGIN
```

(continues on next page)

(continuação da página anterior)

```
5
6  DECLARE @CodigoClienteCartao INT
7
8  SET @CodigoClienteCartao = (SELECT CartaoCredito.ClienteCodigo FROM Clientes LEFT␣
   ↪JOIN CartaoCredito
9  ON CartaoCredito.ClienteCodigo = Clientes.ClienteCodigo WHERE CartaoCredito.
   ↪ClienteCodigo = @CodigoCliente)
10
11         IF @CodigoClienteCartao IS NULL
12                 BEGIN
13                 SELECT * FROM CartaoCredito WHERE ClienteCodigo = @CodigoCliente;
14                 END
15         ELSE
16                 BEGIN
17                 SELECT 'LIGAR', * FROM Clientes WHERE ClienteCodigo =␣
   ↪@CodigoCliente
18                 END
19
20 END;
21
22 EXEC uspRetornaSeTemCartao @CodigoCliente = 25; -- TEM CARTÃO
23
24 EXEC uspRetornaSeTemCartao @CodigoCliente = 1; --NÃO TEM CARTÃO
```

6.8.2 WHILE

- Comando utilizado para realizar laços de repetição.

```
1 DECLARE @contador INT
2 SET @contador = 1
3 WHILE @contador <= 5
4 BEGIN
5   SELECT @contador
6   SET @contador = @contador + 1
7 END
```

6.9 CURSORES

- Cursor.

  Exemplo de um Cursor:

```
1   DECLARE @ClienteNome VARCHAR(50), @ClienteSexo CHAR(1), @contador INT=0;
2
3     DECLARE [cursorListaCliente] CURSOR FOR
4     SELECT Clientes.ClienteNome , ClienteSexo
5     FROM Clientes
6
7     OPEN [cursorListaCliente]
8     FETCH NEXT FROM [cursorListaCliente] INTO @ClienteNome, @ClienteSexo;
9
10    WHILE @@FETCH_STATUS = 0
11    BEGIN
12         SET @contador=@contador+1;
13
14    SELECT @ClienteNome as Nome, @ClienteSexo AS Sexo, @contador;
15    FETCH NEXT FROM [cursorListaCliente] INTO @ClienteNome, @ClienteSexo
```

(continues on next page)

(continuação da página anterior)

```
    END
    CLOSE [cursorListaCliente];
      DEALLOCATE [cursorListaCliente];
```

6.10 TRANSAÇÕES

6.10.1 Transações

- Comando utilizado para alterar registros em um banco de dados. Antes de executar qualquer comando `UPDATE`, procure se informar sobre transações (será abordado mais pra frente).
- Sempre que for trabalhar com o comando `UPDATE` ou `DELETE`, procure executar um `SELECT` antes para validar se os registros que serão afetados, são exatamente aqueles que você deseja.

```
BEGIN TRAN --> Inicia a transação

UPDATE dbo.CartaoCredito SET CartaoLimite = CartaoLimite * 1.1

COMMIT --> Finaliza a transação

--OR

ROLLBACK --> Desfaz a transação
```

Execute primeiro sem o WHERE e verifique que nenhuma linha será alterada. Depois remova o comentário e verá que apenas uma linha foi alterada.

```
BEGIN TRAN

UPDATE dbo.CartaoCredito SET CartaoLimite = CartaoLimite * 1.1
--WHERE ClienteCodigo = '12'

IF (@@ROWCOUNT > 1 OR @@ERROR > 0)

  ROLLBACK

ELSE

  COMMIT
```

6.10.2 Try Catch

```
BEGIN TRY

  SELECT 1/0

END TRY

BEGIN CATCH
  SELECT
      ERROR_NUMBER() AS ErrorNumber,
      ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
```

6.11 TRIGGERS

- Comando vinculado a uma tabela que executa um ação assim que algum comando de UPDATE, INSERT ou DELETE é executado na tabela onde a trigger está vinculada.

6.11.1 Trigger para INSERT

```
CREATE TRIGGER trgINSERT_CLIENTE
ON Clientes
FOR INSERT
AS
BEGIN
INSERT clientes_audit
SELECT *,[TRG_OPERACAO] = 'INSERT', [TRG_DATA]=GETDATE(), [TRG_FLAG]='NEW' FROM␣
↪Inserted
END;
```

6.11.2 Trigger para DELETE

```
CREATE TRIGGER trgDELETE_CLIENTE
ON dbo.Clientes
FOR DELETE
AS
BEGIN
INSERT dbo.clientes_audit SELECT *,[TRG_OPERACAO] = 'DELETE',[TRG_DATA]=GETDATE(),
↪[TRG_FLAG]='OLD' FROM Deleted
END;
```

6.11.3 Trigger para UPDATE

```
CREATE TRIGGER trgUPDATE_CLIENTE
ON dbo.Clientes
FOR UPDATE
AS
BEGIN
INSERT dbo.clientes_audit SELECT *,[TRG_OPERACAO] = 'UPDATE',[TRG_DATA]=GETDATE(),
↪[TRG_FLAG]='OLD' FROM Deleted
INSERT dbo.clientes_audit SELECT *,[TRG_OPERACAO] = 'UPDATE',[TRG_DATA]=GETDATE(),
↪[TRG_FLAG]='NEW' FROM Inserted
END;
```

6.12 INDICES

- Criação de índices e estatísitcas

Os índices garantem um bom desempenho para as consultas que serão realizadas no banco de dados. Comece verificando com a procedure `sp_help` os metadados das tabelas para verificar se não existe um índice que possa ajudar na sua consulta.

Caso precise criar um índice comece analisando os campos que estão na sua cláusula WHERE. Esses campos são conhecidos como predicados. Ainda dentro da cláusula WHERE procure filtrar primeiramente os campos com maior seletividade, que possam filtar os dados de forma que não sejam trazidos ou pesquisados dados descessários.

Em seguida olhe os campos da cláusula SELECT e adicione eles no índice.

- Atenção Leia o material complementar na biblioteca Virtual
- Exemplo

A consulta abaixo busca nome e data de nascimentos do cliente com base em uma data passada pelo usuário ou sistema. Como primeiro passo vamos olhar a cláusula WHERE e em seguida a cláusula SELECT. Dessa forma temos um índice que deverá conter ClienteNascimento e ClienteNome onde ClienteNascimento é o predicado.

Comando

```
      SELECT Clientes.ClienteNome, Clientes.ClienteNascimento
      FROM Clientes
      WHERE ClienteNascimento >= '1980-01-01'
```

Índice

```
      CREATE INDEX IX_NOME ON Clientes
      (
              ClienteNascimento,
              ClienteNome
      )
```

6.13 BACK UP

- Comando .

6.13.1 Comano para BACK UP

```
BACKUP DATABASE [MinhaCaixa]
TO  DISK = 'C:\bkp\MinhaCaixa2018.bak';


BACKUP DATABASE [MinhaCaixa]
TO  DISK = N'C:\bkp\MinhaCaixa2018_diff.bak'
WITH  DIFFERENTIAL , STATS = 10;

BACKUP LOG [MinhaCaixa] TO
DISK = N'C:\bkp\MinhaCaixa2018_log.trn' WITH NOFORMAT,  STATS = 10;


USE [master]
RESTORE DATABASE [MinhaCaixa]
FROM  DISK = N'C:\bkp\MinhaCaixa2018.bak'
WITH REPLACE,  STATS = 10;

USE [master]

RESTORE DATABASE [MinhaCaixa] FROM  DISK = N'C:\bkp\MinhaCaixa2018.bak'
WITH  FILE = 1,  NORECOVERY,  NOUNLOAD,  STATS = 5
RESTORE DATABASE [MinhaCaixa] FROM  DISK = N'C:\bkp\MinhaCaixa2018_diff.bak'
WITH  FILE = 1,  NORECOVERY,  NOUNLOAD,  STATS = 5
RESTORE LOG [MinhaCaixa] FROM  DISK = N'C:\bkp\MinhaCaixa2018_log.trn'
WITH  FILE = 1,  NOUNLOAD,  STATS = 5;
```

CAPÍTULO 7

Exercícios SQL

Exercícios

7.1 EXERCÍCIOS Parte 1

1. Crie uma tabela para armazenar o nome do feriado e data dele. Em seguida pesquise quais são os feriados nacionais (brasileiros) e insira nessa tabela. A tabela devera ter código do feriado (auto-incremento), nome do feriado e a data em que ele é comemorado.

```
CREATE TABLE FERIADOS
 (
 CODFERIADO INT IDENTITY (1,1) CONSTRAINT PK_FERIADO PRIMARY KEY,
 NOMEFERIADO VARCHAR(100),
 DATAFERIADO DATE
     );

INSERT FERIADOS (NOMEFERIADO, DATAFERIADO)
             VALUES ('INDEPENDENCIA','2018-09-07');

SELECT * FROM FERIADOS;
```

2. Escolha 5 clientes e cadastre cartões de crédito para eles.

```
INSERT CartaoCredito (AgenciaCodigo, ContaNumero, ClienteCodigo,␣
↪CartaoCodigo, CartaoLimite, CartaoExpira, CartaoCodigoSeguranca)
                            VALUES (1,'562296-2',25,'1001-2002-3003-4004
↪',3500.00,'2020-10-10',   123  );
```

7.2 EXERCÍCIOS Parte 2

1. Faça um consulta que retorne o nome e sobrenome do cliente, seu bairro, e os valores das suas movimentações, a data ordenando as movimentações pelas mais recentes.

```
SELECT ClienteNome, ClienteSobrenome, ClienteBairro, MovimentoData,␣
↪MovimentoValor
  FROM Clientes, Contas, Movimentos
  WHERE Clientes.ClienteCodigo=Contas.ClienteCodigo
  AND Contas.ContaNumero=Movimentos.ContaNumero
  ORDER BY MovimentoData desc;
```

2. Mostre o nome do cliente, sobrenome e a sua renda convertida em dolar e euro.

```
SELECT ClienteNome, ClienteSobrenome,
   (ClienteRendaAnual / 3.9) AS Dolar, (ClienteRendaAnual / 4.4) AS Euro
       FROM Clientes;
```

3. Traga o nome dos clientes, o sobrenome, o bairro, o estado civil (descrito), o sexo (descrito) e classifique o cliente de acordo com a sua renda anual, C tem renda menor que 50.000, B tem renda menor que 70.000 e A tem renda acima de 70.000.

```
SELECT ClienteNome, ClienteSobrenome, ClienteBairro, ClienteEstadoCivil,
            CASE WHEN ClienteEstadoCivil = 'S' THEN 'Solteiro' ELSE
↪'Casado' END AS ESTADOCIVILDECRITO,
            ClienteSexo,
            CASE WHEN ClienteSexo = 'M' THEN 'Masculino' ELSE 'Feminino'␣
↪END AS SEXODESCRITO,
            ClienteRendaAnual,
            CASE WHEN ClienteRendaAnual < 50000 THEN 'C'
            WHEN ClienteRendaAnual < 70000 THEN 'B'
            ELSE 'A'
            END AS 'CLASSIFICAÇÃO'
            FROM Clientes ;
```

4. Liste todos os clientes que moram no mesmo bairro das agências do banco.

```
SELECT ClienteNome, ClienteBairro, AgenciaBairro, AgenciaNome FROM␣
↪Clientes, Agencias
  WHERE ClienteBairro=AgenciaBairro;
```

5. Mostre todos os clientes que possuem número no seu e-mail.

```
SELECT Clientes.ClienteNome, Clientes.ClienteEmail
  FROM dbo.Clientes
  WHERE Clientes.ClienteEmail LIKE '%[0-9]%';
```

6. Mostre todos os clientes em que o nome da rua começa começa com R. e não com RUA.

```
SELECT ClienteRua FROM dbo.Clientes WHERE
  ClienteRua LIKE 'R.%'
  AND ClienteRua NOT LIKE 'RUA%';
```

7. Mostre o nome do cliente e a renda apenas do 5 melhores clientes com base na sua renda.

```
SELECT TOP 5 ClienteNome, ClienteRendaAnual
    FROM dbo.Clientes
    ORDER BY ClienteRendaAnual DESC;
```

8. Mostre o nome do cliente e a renda apenas do 5 piores clientes com base na sua renda.

```
SELECT TOP 5 ClienteNome, ClienteRendaAnual
    FROM dbo.Clientes
    ORDER BY ClienteRendaAnual;
```

9. Mostre o nome e a rua dos clientes que moram em residencias cujo número está entre 300 e 500.

```
SELECT ClienteNome, ClienteRua FROM dbo.Clientes
            WHERE ClienteNumero BETWEEN 300 AND 500;
```

10. Utilizando o conceito de sub consulta, mostre quais clientes não possuem cartão de crédito.

```
SELECT * FROM dbo.Clientes WHERE ClienteCodigo NOT IN
            (SELECT ClienteCodigo FROM dbo.CartaoCredito);
```

11. Mostre o nome do cliente, o nome da agência e o bairro da agência, as movimentações dos clientes e o limite do cartão de crédito deles, somente para os clientes em que a conta foi aberta a partir de 2008.

```
SELECT ClienteNome, AgenciaNome, AgenciaBairro, MovimentoValor
    FROM dbo.Clientes, dbo.Agencias, dbo.Contas, dbo.CartaoCredito, dbo.
↪Movimentos
    WHERE clientes.ClienteCodigo=Contas.ClienteCodigo
    AND agencias.AgenciaCodigo=dbo.Contas.AgenciaCodigo
    AND CartaoCredito.ClienteCodigo=Clientes.ClienteCodigo
    AND dbo.Contas.ContaNumero=dbo.Movimentos.ContaNumero
    AND ContaAbertura >= '2008-01-01';
```

12. Faça uma consulta que classifique os clientes em Regiões conforme o bairro que moram.

```
SELECT dbo.Clientes.ClienteNome, dbo.Clientes.ClienteBairro,
            CASE WHEN ClienteBairro IN ('ITINGA','FLORESTA')
            THEN 'SUL' END  AS [REGIÃO]
            FROM Clientes;
```

13. Mostre o nome do cliente e o tipo de movimentação, apenas para as movimentações de débito.

```
SELECT ClienteNome, MovimentoValor, MovimentoTipo , TipoMovimentoDescricao
            FROM Clientes, Contas, Movimentos, TipoMovimento
            WHERE Clientes.ClienteCodigo=Contas.ClienteCodigo
            AND Contas.ContaNumero=dbo.Movimentos.ContaNumero
            AND dbo.Movimentos.MovimentoTipo=dbo.TipoMovimento.
↪TipoMovimentoCodigo
            AND TipoMovimento.TipoMovimentoCodigo=-1;
```

7.3 EXERCÍCIOS Parte 3

1. Mostre quais os clientes tem idade superior a média.

```
SELECT ClienteNome, YEAR(GETDATE()) - YEAR(ClienteNascimento) AS idade
  FROM Clientes
  WHERE YEAR(GETDATE()) - YEAR(ClienteNascimento) >
    (
      SELECT AVG(YEAR(GETDATE()) -YEAR(ClienteNascimento)) AS IDADE FROM␣
↪Clientes
    );
```

2. Mostre qual agência tem quantidade de clientes acima da média.

```
SELECT AgenciaNome, COUNT(ClienteCodigo) AS QDTE
FROM Contas INNER JOIN Agencias
  ON Agencias.AgenciaCodigo = Contas.AgenciaCodigo
GROUP BY AgenciaNome
HAVING COUNT(ClienteCodigo) >
  (SELECT COUNT(DISTINCT ClienteCodigo)/
  COUNT(DISTINCT AgenciaCodigo) FROM Contas);
```

3. Mostre o nome da agência o saldo total, o mínimo, o máximo e a quantidade de clientes de cada agência.

```
SELECT AgenciaNome, SUM(ContaSaldo) AS TOTAL ,MIN(ContaSaldo) AS MINIMO,␣
↪MAX(ContaSaldo) AS MAXIMO,
COUNT(Contas.ClienteCodigo) AS QTDE_CLIENTES
FROM Contas INNER JOIN dbo.Agencias ON Agencias.AgenciaCodigo = Contas.
↪AgenciaCodigo
GROUP BY dbo.Agencias.AgenciaNome;
--ATENCAO AQUI PARA COUNT(*) E COUNT(DISTINT)
```

4. Mostre o percentual que cada agencia representa no saldo total do banco.

```
SELECT AgenciaNome, SUM(ContaSaldo) / (SELECT SUM(ContaSaldo) FROM dbo.
↪Contas) * 100 AS PERCENTUAL
FROM Contas INNER JOIN dbo.Agencias ON Agencias.AgenciaCodigo = Contas.
↪AgenciaCodigo
GROUP BY dbo.Agencias.AgenciaNome;
```

5. Mostre as duas cidades que tem o maior saldo total.

```
SELECT TOP 2 AgenciaCidade, SUM(ContaSaldo) AS SALDO_TOTAL
FROM Contas INNER JOIN Agencias ON Agencias.AgenciaCodigo = Contas.
↪AgenciaCodigo
GROUP BY AgenciaCidade
ORDER BY 2 DESC;
```

6. Mostre qual a agência tem o maior montante de emprestimo.

```
SELECT TOP 1 AgenciaCidade, Emprestimos.EmprestimoTotal
FROM dbo.Emprestimos INNER JOIN Agencias ON Agencias.AgenciaCodigo =␣
↪Emprestimos.AgenciaCodigo
ORDER BY 2 DESC;
```

7. Mostre qual o menor valor devido, o maior e o total devido da tabela devedor.

```
SELECT MIN(DevedorSaldo) AS MINIMO, MAX(DevedorSaldo) AS MAXIMO,␣
↪SUM(DevedorSaldo) AS TOTAL
FROM dbo.Devedores;
```

8. Mostre o nome do cliente, se ele tem cartão de crédito, apenas do cliente que é o maior devedor.

```
SELECT TOP 1 --Experimente remover o TOP 1 para conferir o resultado
ClienteNome
,CASE WHEN dbo.CartaoCredito.ClienteCodigo IS NULL THEN 'NÃO TEM CARTÃO␣
↪CRÉDITO' ELSE 'TEM CARTÃO CRÉDITO' END AS 'CARTAO'
,DevedorSaldo FROM dbo.Clientes
INNER JOIN dbo.Devedores ON Devedores.ClienteCodigo = Clientes.
↪ClienteCodigo
LEFT JOIN dbo.CartaoCredito ON CartaoCredito.ClienteCodigo = Clientes.
↪ClienteCodigo
ORDER BY 3 DESC;
```

9. Mostre o nome do cliente, a idade, o saldo total em conta, seu total devido, seu total emprestado e se tem cartão de crédito ou não. Os valores nulos devem aparecer como 0.00. A ordenação dever ser sempre pelo maioir devedor.

```
SELECT Clientes.ClienteNome, DATEDIFF(YEAR,Clientes.ClienteNascimento,␣
↪GETDATE()) AS IDADE,
ISNULL(Devedores.DevedorSaldo,0) AS DevedorSaldo, ISNULL(Emprestimos.
↪EmprestimoTotal,0) AS EmprestimoTotal,
CASE WHEN CartaoCredito.CartaoCodigo IS NULL THEN 'NÃO TEM' ELSE 'TEM'␣
↪END AS CARTAOCREDITO
FROM Clientes
```

(continues on next page)

(continuação da página anterior)

```
LEFT JOIN Devedores ON Devedores.ClienteCodigo = Clientes.ClienteCodigo
LEFT JOIN Emprestimos ON Emprestimos.ClienteCodigo = Clientes.
↪ClienteCodigo
LEFT JOIN CartaoCredito ON CartaoCredito.ClienteCodigo = Clientes.
↪ClienteCodigo
ORDER BY 3 DESC;
```

10. Utilizando a questão anterior, inclua o sexo do cliente e mostre também a diferença entre o que ele emprestou e o que ele está devendo.

```
SELECT Clientes.ClienteNome, DATEDIFF(YEAR,Clientes.ClienteNascimento,␣
↪GETDATE()) AS IDADE,
ISNULL(Devedores.DevedorSaldo,0) AS DevedorSaldo, ISNULL(Emprestimos.
↪EmprestimoTotal,0) AS EmprestimoTotal,
CASE WHEN .CartaoCredito.CartaoCodigo IS NULL THEN 'NÃO TEM' ELSE 'TEM'␣
↪END AS CARTAOCREDITO,
CASE WHEN ClienteNome LIKE '%a' THEN 'FEMININO' ELSE 'MASCULINO' END AS␣
↪SEXO,
ISNULL((Emprestimos.EmprestimoTotal-DevedorSaldo),0) AS DIFERENÇA
FROM Clientes
LEFT JOIN Devedores ON Devedores.ClienteCodigo = Clientes.ClienteCodigo
LEFT JOIN Emprestimos ON Emprestimos.ClienteCodigo = Clientes.
↪ClienteCodigo
LEFT JOIN CartaoCredito ON CartaoCredito.ClienteCodigo = Clientes.
↪ClienteCodigo
ORDER BY 3 DESC;
```

11. Insira um novo cliente chamado Silvio Santos, crie uma conta para ele com saldo de R$ 500,00 na agência Beira Mar. Cadastre um cartão de crédito com limite de 5000,00.

```
INSERT Clientes (ClienteNome, ClienteRua, ClienteCidade,␣
↪ClienteNascimento) VALUES  ('Silvio Santos', 'Rua João Colin, 1234',
↪'Joinville','1980-01-01' );

SELECT @@IDENTITY --RETORNA O CÓDIGO DO CLIENTE GERADO PELO AUTO␣
↪INCREMENTO --> IDENTITY

INSERT Contas (AgenciaCodigo ,ContaNumero , ClienteCodigo , ContaSaldo ,␣
↪ContaAbertura)
OUTPUT INSERTED.* --RETORNA OS REGISTROS INSERIDOS NA TABELA
VALUES (5,'C-999',14,500,'2016-01-01');

INSERT CartaoCredito ( AgenciaCodigo , ClienteCodigo , CartaoCodigo ,␣
↪CartaoLimite)
VALUES  (5,14,'1234-1234-1234-1234',5000);
```

12. Altere a rua do cliente Ana para Rua da Univille.

```
UPDATE dbo.Clientes SET ClienteRua = 'Rua da Univille' WHERE ClienteNome␣
↪= 'Ana';
```

13. Apague todos os registros da cliente Vânia.

```
DECLARE @ClienteCodigo INT = (SELECT ClienteCodigo FROM dbo.Clientes␣
↪WHERE ClienteNome = 'Vânia')

DELETE FROM dbo.Emprestimos WHERE ClienteCodigo = @ClienteCodigo
DELETE FROM dbo.Devedores WHERE ClienteCodigo = @ClienteCodigo
DELETE FROM dbo.Depositantes WHERE ClienteCodigo = @ClienteCodigo
DELETE FROM dbo.CartaoCredito WHERE ClienteCodigo = @ClienteCodigo
```

(continues on next page)

(continuação da página anterior)

```
DELETE FROM dbo.Contas WHERE ClienteCodigo = @ClienteCodigo
DELETE FROM dbo.Clientes WHERE ClienteCodigo = @ClienteCodigo
```

14. Mostre nome e sobrenome e se o cliente for homem, mostre Sr. e se for mulher Sra. na frente do nome. Use o MinhaCaixa_Beta para resolver essa questão.

```
USE MinhaCaixa_Beta
GO
SELECT
CASE WHEN ClienteSexo = 'M' THEN 'Sr. ' + dbo.Clientes.ClienteNome + ' '␣
↪+ dbo.Clientes.ClienteSobrenome
ELSE 'Sra. ' + dbo.Clientes.ClienteNome + ' ' + dbo.Clientes.
↪ClienteSobrenome END AS Cliente
FROM dbo.Clientes
```

15. Mostre os bairros que tem mais clientes.

```
USE MinhaCaixa_Beta
GO
SELECT COUNT(dbo.Clientes.ClienteCodigo) AS Quantidade,
dbo.Clientes.ClienteBairro
FROM dbo.Clientes
GROUP BY dbo.Clientes.ClienteBairro
ORDER BY 1 desc
```

16. Mostre a renda de cada cliente convertida em dólar.

```
USE MinhaCaixa_Beta
GO
SELECT ClienteNome +' ' + ClienteSobrenome AS Cliente,
CONVERT(DECIMAL(10,2),Clientes.ClienteRendaAnual / 3.25) AS RENDADOLAR
FROM dbo.Clientes
```

17. Mostre o nome do cliente, o número da conta, o saldo da conta, apenas para os 15 melhores clientes.

```
    USE MinhaCaixa_Beta
GO
SELECT TOP 15
ClienteNome +' ' + ClienteSobrenome AS Cliente, Contas.ContaNumero,
SUM(MovimentoValor*MovimentoTipo) AS Saldo
FROM dbo.Clientes INNER JOIN dbo.Contas ON Contas.ClienteCodigo =␣
↪Clientes.ClienteCodigo
INNER JOIN dbo.Movimentos ON Movimentos.ContaNumero = Contas.ContaNumero
GROUP BY ClienteNome + ' ' + ClienteSobrenome , Contas.ContaNumero
ORDER BY 3 DESC
```

18. Mostre quais são os 5 dias com maior movimento (valor) no banco.

```
USE MinhaCaixa_Beta
GO
SELECT TOP 5 DAY(Movimentos.MovimentoData) AS DIA,
SUM(dbo.Movimentos.MovimentoValor*dbo.Movimentos.MovimentoTipo) AS VALOR
FROM dbo.Movimentos
GROUP BY DAY(Movimentos.MovimentoData)
ORDER BY 2 DESC
```

19. Crie uma função que receba o código do estado civil e mostre ele por extenso.
20. Crie uma função que receba o código do sexo e mostre ele por extenso.
21. Crie um procedure que receba o número da conta e cadastre um cartão de crédito com limite de R$ 500 para o cliente caso ele não tenha (MinhaCaixa).

22. Use o script abaixo para criar uma procedure que receba a matricula, disciplina, ano e calcule o total de pontos e a média do aluno.

```
CREATE TABLE Notas
(
Matricula INT,
Materia CHAR (3),
Ano INT,
Nota1 FLOAT,
Nota2 FLOAT,
Nota3 FLOAT,
Nota4 FLOAT,
TotalPontos FLOAT,
MediaFinal FLOAT
);
INSERT Notas (Matricula, Materia, Ano, Nota1, Nota2, Nota3, Nota4) VALUES
↪ (1,'BDA',2016,7,7,7,7);
```

23. Use o script abaixo para criar duas procedures:

Uma procedure para cadastrar os alunos em duas matérias (BDA e PRG). Exemplo: exec procedure @matricula, @materia, @ano

(matricular 6 alunos)

Uma procedure que receba a matricula, disciplina, ano, bimestre, aulas dadas, notas e faltas. Quando a condição dentro da procedure identificar que é o quarto bimestre calcule o total de pontos, total de faltas, percentual de frequencia,a média do aluno e calcule o resultado final, A, E ou R.

Exemplo: exec procedure @matricula, @materia, @ano, 1, 32, 7, 0

```
CREATE TABLE Notas
(
Matricula INT,
Materia CHAR (3),
Ano INT,
    Aulas1 INT,
    Aulas2 INT,
    Aulas3 INT,
    Aulas4 INT,
Nota1 FLOAT,
Nota2 FLOAT,
Nota3 FLOAT,
Nota4 FLOAT,
    Falta1 INT,
    Falta2 INT,
    Falta3 INT,
    Falta4 INT,
TotalPontos FLOAT,
    TotalFaltas INT,
    TotalAulas INT,
MediaFinal FLOAT,
    PercentualFrequencia float,
    Resultado char(1)
);
```

7.4 EXERCÍCIOS Procedure

1. Neste exercício vamos criar um banco de dados para armazenar os dados dos alunos de uma universidade. Além de desenhar o diagrama, criar o banco de dados e seus objetos, você deverá criar os scripts de população básica. Em seguida deverá criar as procedures que irão executar as operações de manipulação das notas e faltas. Abaixo uma sugestão de parte da solução:

```
    USE MASTER
    ALTER DATABASE Universidade SET SINGLE_USER WITH ROLLBACK IMMEDIATE
    GO
    DROP DATABASE Universidade;
    GO
    USE master;
    CREATE DATABASE Universidade;
    GO
    USE Universidade;
    GO
    CREATE TABLE ALUNOS (MATRICULA INT NOT NULL IDENTITY CONSTRAINT PK_
↪ALUNO PRIMARY KEY, NOME VARCHAR(50) NOT NULL);
    GO
    CREATE TABLE CURSOS (CURSO CHAR(3) NOT NULL CONSTRAINT PK_CURSO␣
↪PRIMARY KEY, NOME VARCHAR(50) NOT NULL);
    GO
    CREATE TABLE PROFESSOR (PROFESSOR INT IDENTITY NOT NULL CONSTRAINT PK_
↪PROFESSOR PRIMARY KEY, NOME VARCHAR(50) NOT NULL);
    GO
    CREATE TABLE MATERIAS (SIGLA CHAR(3) NOT NULL, NOME VARCHAR(50) NOT␣
↪NULL, CARGAHORARIA INT NOT NULL,
    CURSO CHAR(3) NOT NULL, PROFESSOR INT
    CONSTRAINT PK_MATERIA PRIMARY KEY (SIGLA,CURSO,PROFESSOR)
    CONSTRAINT FK_CURSO FOREIGN KEY (CURSO) REFERENCES CURSOS(CURSO),
    CONSTRAINT FK_PROFESSOR FOREIGN KEY (PROFESSOR) REFERENCES PROFESSOR␣
↪(PROFESSOR)
    );
    GO
    INSERT ALUNOS (NOME) VALUES ('Pedro')
    GO
    INSERT CURSOS (CURSO, NOME) VALUES ('SIS','SISTEMAS'),('ENG',
↪'ENGENHARIA')
    GO
    INSERT PROFESSOR (NOME ) VALUES ('DORNEL'),('WALTER')
    GO
    INSERT MATERIAS (SIGLA, NOME, CARGAHORARIA, CURSO,PROFESSOR)
    VALUES ('BDA','BANCO DE DADOS',144,'SIS',1), ('PRG','PROGRAMAÇÃO',144,
↪'SIS',2)
    GO
    INSERT MATERIAS (SIGLA, NOME, CARGAHORARIA, CURSO,PROFESSOR)
    VALUES ('BDA','BANCO DE DADOS',144,'ENG',1), ('PRG','PROGRAMAÇÃO',144,
↪'ENG',2)
    GO
    CREATE TABLE MATRICULA (MATRICULA INT, CURSO CHAR(3), MATERIA CHAR(3),
↪ PROFESSOR INT, PERLETIVO INT,
    N1 FLOAT, N2 FLOAT, N3 FLOAT, N4 FLOAT, TOTALPONTOS FLOAT, MEDIA␣
↪FLOAT,
    F1 INT, F2 INT, F3 INT, F4 INT, TOTALFALTAS INT, PERCFREQ FLOAT,␣
↪RESULTADO VARCHAR(20)

    CONSTRAINT PK_MATRICULA PRIMARY KEY (MATRICULA,CURSO,MATERIA,
↪PROFESSOR,PERLETIVO),
    CONSTRAINT FK_ALUNOS_MATRICULA FOREIGN KEY (MATRICULA) REFERENCES␣
↪ALUNOS (MATRICULA),
    CONSTRAINT FK_CURSOS_MATRICULA FOREIGN KEY (CURSO) REFERENCES CURSOS␣
↪(CURSO),
    --CONSTRAINT FK_MATERIAS FOREIGN KEY (MATERIA) REFERENCES MATERIAS␣
↪(SIGLA),
    CONSTRAINT FK_PROFESSOR_MATRICULA FOREIGN KEY (PROFESSOR) REFERENCES␣
↪PROFESSOR(PROFESSOR)
    )

```

(continues on next page)

(continuação da página anterior)

```
47
48      CREATE PROCEDURE sp_MatriculaAluno
49      (
50      @NOMEALUNO VARCHAR(50),
51      @CURSOALUNO VARCHAR(50)
52      )
53      AS
54      BEGIN
55
56      DECLARE @MATRICULAALUNO INT, @CODIGOCURSO VARCHAR(3)
57
58      SET @MATRICULAALUNO = (SELECT MATRICULA FROM ALUNOS WHERE NOME =␣
    ↪@NOMEALUNO)
59
60      SET @CODIGOCURSO = (SELECT CURSO FROM CURSOS WHERE NOME = @CURSOALUNO)
61
62      INSERT MATRICULA
63              (
64                              MATRICULA,
65                              CURSO,
66                              MATERIA,
67                              PROFESSOR,
68                              PERLETIVO
69
70              )
71              SELECT @MATRICULAALUNO AS MATRICULA, CURSO, SIGLA,PROFESSOR,␣
    ↪YEAR(GETDATE()) AS PERLETIVO FROM MATERIAS WHERE CURSO ='ENG'
72
73      END
74
75      --Calculo do percentual de Frequencia (144-NrFaltas*100)/144
76
77
78      EXEC sp_MatriculaAluno @NOMEALUNO = 'Guilherme', -- varchar(50)
79                                                      @CURSOALUNO = 'Sistemas' --
    ↪ varchar(50)
```

Exemplo de INSERT com SELECT

```
1       INSERT MATRICULA
2       (
3               MATRICULA,
4               CURSO,
5               MATERIA,
6               PROFESSOR,
7               PERLETIVO
8
9       )
10      SELECT 1 AS MATRICULA, CURSO, SIGLA,PROFESSOR, YEAR(GETDATE()) FROM␣
    ↪MATERIAS WHERE CURSO ='ENG'
```

Exemplo de PROCEDURE para inserir (atualizar) as notas

```
1       CREATE PROCEDURE [dbo].[sp_CadastraNotas]
2       (
3               @MATRICULA INT,
4               @CURSO CHAR(3),
5               @MATERIA CHAR(3),
6               @PERLETIVO CHAR(4),
7               @NOTA FLOAT,
8               @FALTA INT,
```

(continues on next page)

(continuação da página anterior)

```
        @PARAMETRO INT
)
AS
BEGIN

        IF @PARAMETRO = 1
        BEGIN

                UPDATE MATRICULA
                SET N1 = @NOTA,
                        F1 = @FALTA,
                        TOTALPONTOS = @NOTA,
                        TOTALFALTAS = @FALTA,
                        MEDIA = @NOTA
                WHERE MATRICULA = @MATRICULA
                          AND CURSO = @CURSO
                          AND MATERIA = @MATERIA
                          AND PERLETIVO = @PERLETIVO;
        END;

        ELSE IF @PARAMETRO = 2
        BEGIN

                UPDATE MATRICULA
                SET N2 = @NOTA,
                        F2 = @FALTA,
                        TOTALPONTOS = @NOTA + N1,
                        TOTALFALTAS = @FALTA + F1,
                        MEDIA = (@NOTA + N1) / 2
                WHERE MATRICULA = @MATRICULA
                          AND CURSO = @CURSO
                          AND MATERIA = @MATERIA
                          AND PERLETIVO = @PERLETIVO;
        END;

        ELSE IF @PARAMETRO = 3
        BEGIN

                UPDATE MATRICULA
                SET N3 = @NOTA,
                        F3 = @FALTA,
                        TOTALPONTOS = @NOTA + N1 + N2,
                        TOTALFALTAS = @FALTA + F1 + F2,
                        MEDIA = (@NOTA + N1 + N2) / 3
                WHERE MATRICULA = @MATRICULA
                          AND CURSO = @CURSO
                          AND MATERIA = @MATERIA
                          AND PERLETIVO = @PERLETIVO;
        END;

        ELSE IF @PARAMETRO = 4
        BEGIN

                DECLARE @RESULTADO VARCHAR(50),
                                @FREQUENCIA FLOAT,
                                @MEDIAFINAL FLOAT;



                UPDATE MATRICULA
                SET N4 = @NOTA,
```

(continues on next page)

(continuação da página anterior)

```
                                F4 = @FALTA,
                                TOTALPONTOS = @NOTA + N1 + N2 + N3,
                                TOTALFALTAS = @FALTA + F1 + F2 + F3,
                                MEDIA = (@NOTA + N1 + N2 + N3) / 4,
                                @MEDIAFINAL = (@NOTA + N1 + N2 + N3) / 4,
                                MEDIAFINAL = @MEDIAFINAL,
                                @FREQUENCIA = 100 - (((@FALTA + F1 + F2 + F3) * 144) / 100),
                                PERCFREQ = @FREQUENCIA,
                                RESULTADO = CASE
                                                                        WHEN @FREQUENCIA >= 75
                                                                                AND @MEDIAFINAL >= 7 THEN
                                                                        'APROVADO'
                                                                        WHEN @FREQUENCIA >= 75
                                                                                AND @MEDIAFINAL >= 3 THEN
                                                                                'EXAME'
                                                                        ELSE
                                                                        'REPROVADO'
                                                                END
                        WHERE MATRICULA = @MATRICULA
                                  AND CURSO = @CURSO
                                  AND MATERIA = @MATERIA
                                  AND PERLETIVO = @PERLETIVO;



                END;

                ELSE IF @PARAMETRO = 5
                BEGIN

                        DECLARE @MEDIA FLOAT =
                                        (
                                                SELECT MEDIA
                                                FROM MATRICULA
                                                WHERE MATRICULA = @MATRICULA
                                                          AND CURSO = @CURSO
                                                          AND MATERIA = @MATERIA
                                                          AND PERLETIVO = @PERLETIVO
                                                          AND RESULTADO = 'EXAME'
                                        );

                        UPDATE MATRICULA
                        SET NOTAEXAME = @NOTA,
                                RESULTADO = CASE
                                                                        WHEN (@NOTA + @MEDIA) >= 10 THEN
                                                                        'APROVADO'
                                                                        ELSE
                                                                        'REPROVADO'
```

(continues on next page)

(continuação da página anterior)

```
                                                    END
                    WHERE MATRICULA = @MATRICULA
                                AND CURSO = @CURSO
                                AND MATERIA = @MATERIA
                                AND PERLETIVO = @PERLETIVO
                                AND RESULTADO = 'EXAME';


            END;

    END;
    GO
```

Exemplo de execução da PROCEDURE para inserir (atualizar) as notas

```
    --ALTER TABLE MATRICULA ADD MEDIAFINAL FLOAT

    --ALTER TABLE MATRICULA ADD NOTAEXAME FLOAT


    EXEC sp_CadastraNotas @MATRICULA = 4,       -- int
                                                @CURSO = 'ENG',       --
↪char(3)
                                                @MATERIA = 'BDA',     --
↪char(3)
                                                @PERLETIVO = '2018', --
↪char(4)
                                                @NOTA = 7.0,          --
↪float
                                                @FALTA = 2,
                                                @PARAMETRO = 4;       -- int
```

Exemplo de INSERT - SELECT

```
            CREATE TABLE pedidos
    (
    idpedido INT,
    idproduto INT,
    valorpedido float
    )

    CREATE TABLE itenspedido
    (
    idpedido INT,
    iditem int,
    idproduto int
    )

    CREATE TABLE itens
    (
    iditem INT,
    nome varchar(50)
    )
    INSERT itens
    (
            iditem,
            nome
    )
    VALUES
    (   1, -- iditem - int
            'AR CONDICIONADO' -- nome - varchar(50)
```

(continues on next page)

(continuação da página anterior)

```
            )


    CREATE TABLE subitens
    (
    idsubitem INT,
    iditem INT,
    nomesubitem VARCHAR(50)
    )
    INSERT subitens
    (
            idsubitem,
            iditem,
            nomesubitem
    )
    VALUES
    (   2, -- idsubitem - int
            1, -- iditem - int
            'MOTOR' -- nomesubitem - varchar(50)
            )



            SELECT * FROM itens
            SELECT * FROM subitens

            SELECT * FROM PEDIDOS


            INSERT pedidos
            (
                    idpedido,
                    idproduto,
                    valorpedido
            )
            VALUES
            (   1,  -- idpedido - int
                    1,  -- idproduto - int
                    1000.00 -- valorpedido - float
                    )

                    DECLARE @produto INT
                    SET @produto = (SELECT idproduto FROM pedidos WHERE
↪idpedido =1)

                    SELECT @produto AS 'AR COND'

                    INSERT itenspedido
                    (
                            idpedido,
                            iditem,
                            idproduto
                    )
                    SELECT IDPEDIDO=1, idsubitem, iditem
                    FROM subitens WHERE iditem = 1--@CURSO

                    --VALUES
                    --(   0, -- idpedido - int
                    --    0, -- iditem - int
                    --    0  -- idproduto - int
                    --    )
```

7.5 EXERCÍCIOS Trigger

1. Neste exercício vamos criar triggers...

```
CREATE TABLE FUNCIONARIOS
(
NOME VARCHAR(10),
SALARIO MONEY
)
SELECT * FROM FUNCIONARIOS
SELECT * FROM auditFuncionarios

INSERT FUNCIONARIOS (NOME,SALARIO)
VALUES ('PEDRO',1000)

DELETE FROM FUNCIONARIOS WHERE NOME = 'PEDRO'

UPDATE FUNCIONARIOS SET SALARIO = SALARIO * 10


CREATE TRIGGER TGR_SALARIO
ON FUNCIONARIOS
FOR INSERT
AS
INSERT auditFuncionarios
SELECT *, GETDATE(),'INSERT' FROM inserted



alter TRIGGER TGR_SALARIO_DEL
ON FUNCIONARIOS
FOR DELETE
AS
INSERT auditFuncionarios
SELECT *, GETDATE(),'DELETE' FROM deleted


create TRIGGER TGR_SALARIO_up
ON FUNCIONARIOS
FOR UPDATE
AS
INSERT auditFuncionarios
SELECT *, GETDATE(),'UPD_VELHO' FROM deleted

INSERT auditFuncionarios
SELECT *, GETDATE(),'UPD_NOVO' FROM INSERTED
```

2. Neste exercício vamos criar triggers...

```
SELECT * FROM dbo.ALUNOS

SELECT * FROM dbo.MATRICULA


ALTER TABLE dbo.ALUNOS ADD CURSO CHAR(3)


CREATE TRIGGER TGR_MATRICULA_ALUNO ON ALUNOS FOR
INSERT
AS
BEGIN

```

(continues on next page)

(continuação da página anterior)

```
14
15          DECLARE @MATRICULAALUNO INT, @CODIGOCURSO VARCHAR(3)
16
17          SET @MATRICULAALUNO = (SELECT MATRICULA FROM INSERTED)
18
19          SET @CODIGOCURSO = (SELECT CURSO FROM INSERTED)
20
21
22      INSERT MATRICULA
23                              (
24                                                              MATRICULA,
25                                                              CURSO,
26                                                              MATERIA,
27                                                              PROFESSOR,
28                                                              PERLETIVO
29
30                              )
31                              SELECT @MATRICULAALUNO AS MATRICULA,␣
    ↪@CODIGOCURSO, SIGLA,PROFESSOR,
32                              YEAR(GETDATE()) AS PERLETIVO FROM MATERIAS␣
    ↪WHERE CURSO =
33                              @CODIGOCURSO
34
35          END
36
37
38
39            EXEC sp_MatriculaAluno @NOMEALUNO = 'José', -- varchar(50)
40                                                                          ␣
    ↪                   @CURSOALUNO = 'SIS' -- varchar(50)
41
42
43      CREATE PROCEDURE sp_MatriculaAluno
44              (
45              @NOMEALUNO VARCHAR(50),
46              @CURSOALUNO VARCHAR(50)
47              )
48              AS
49              BEGIN
50
51              INSERT dbo.ALUNOS
52              (
53                      NOME,CURSO
54              )
55              VALUES
56              (@NOMEALUNO,@CURSOALUNO)
57              END;
```

CAPÍTULO 8

Administração de Banco de Dados

Objeito de aprender tarefas básicas do dia a dia de um administrador de banco de dados

8.1 Segurança

8.1.1 Logins

8.2 Manutenção

8.2.1 Rotinas

CAPÍTULO 9

Extra

9.1 Documentação dos SGBD

SGBD	Versão	Link
MariaDB	Inglês (Oficial)	https://mariadb.com/kb/en/mariadb/documentation/
MySQL	Inglês (Oficial)	https://dev.mysql.com/doc/
PostgreSQL	Inglês (Oficial) Português (8.0)	https://www.postgresql.org/docs/manuals/ http://pgdocptbr.sourceforge.net/pg80/index.html
SQL Server	Português (2016) Português (2016 Desenvolvimento)	https://msdn.microsoft.com/pt-br/library/ms130214.aspx https://technet.microsoft.com/pt-br/library/bb500155(v=sql.105).aspx
SQLite	Inglês (Oficial)	https://www.sqlite.org/docs.html
Berkeley DB	Inglês (Oficial)	https://docs.oracle.com/database/bdb181/index.html
MongoDB	Inglês (Oficial)	https://docs.mongodb.com/manual/
IBM DB2	Inglês (Oficial)	https://www.ibm.com/support/knowledgecenter/SSEPGG
MemSQL	Inglês (Oficial)	https://docs.memsql.com/
mSQL	Inglês (Oficial)	http://www.hughes.com.au/products/msql/msql-4.0-manual.pdf
MonetDB	Inglês (Oficial)	https://www.monetdb.org/Documentation
NuoDB	Inglês (Oficial)	http://doc.nuodb.com/Latest/Default.htm
SQream DB	Inglês (Oficial)	http://docs.sqream.com/latest/manual/

9.2 Sites Interessantes

- Database Cast: Podcast sobre banco de dados.
- Documentação do SQL: Site para a documentação do SQL.

9.3 Palestras

- A Arquitetura (Peculiar) do Stack Overflow
- PostgreSQL no Debian

9.4 Como Contribuir?

- Crie um fork do projeto no GitHub.
- Faça suas alterações no seu fork.
  - Se possível, utilize o plugin EditorConfig no seu editor de texto.
  - Escreva o conteúdo usando reStructuredText.
    - Fique atento a marcação dos títulos.
    - Utilize um bloco de código com syntax highlight para código.

      Exemplo:

```
.. code-block:: sql

  SELECT * FROM tabela;
```

  - Adicione os link nos arquivos `index.rst` caso tenha criado algum arquivo novo.
  - Adicione os arquivos modificados (`git add`) e faça o commit (`git commit`).
- Crie um pull request no GitHub.
- Espere sua contribuição ser aprovada.

9.5 Git

9.5.1 Links de material

9.5.1.1 Livros / Documentação

- Pro Git (v2 inglês) (v1 português)
- Getting Git Right

9.5.1.2 Tutoriais

- git - guia prático - sem complicação!
- Try Git
- Learn Git Version Control using Interactive Browser-Based Labs

9.5.1.3 Vídeos

- Introdução ao Git
- Git para quem gosta de Git

9.5.1.4 Ferramentas

- Learn Git Branching

9.6 Como Compilar o Material com o Sphinx

9.6.1 Instalar o Python

- Usar preferencialmente a versão 3.
- Ambientes Unix provavelmente já possuem ele instalado.
- Pode ser encontrado em https://www.python.org/downloads/.
- No Windows, durante a instalação marcar para instalar o "pip" também.

9.6.2 Instalar as Dependências

Dentro do diretório do código do material executar:

```
pip install -r requirements.txt
```

9.6.3 Compilar o Material

Para listar as opções de compilação execute:

```
make help
```

Alguns exemplos:

HTML

```
make html
```

PDF

```
make latexpdf
```

> **Nota:** É necessário que o LaTeX esteja instalado no sistema para gerar o PDF.

ePub

```
make epub
```

Nota: Após a execução do comando, o material compilado, junto com alguns outros arquivos, podem ser encontrados dentro do diretório `_build` na raíz do projeto.