UniEVANGÉLICA
CENTRO UNIVERSITÁRIO

Banco de Dados

Bacharelados em Computação

Centro Universitário de Anápolis - UniEVANGÉLICA

Título: **Banco de Dados** / Ano – 2018

Preparo de Originais

Adalberto Dias da Luz Junior	Lucas Galvão Lima
Aline Dayany de Lemos	Luis Fernando Montes
Bruno André Pereira de Deus	Pedro Antonio Cardoso
Daniel Vaz de Oliveira	Raphael dos Santos Guedes Vieira
José André da Silva Filho	Thiago Antonio Xavier
Kelly Kianny Morais Caixeta	Yasmin Xavier dos Santos Generoso

Equipe editorial:

Revisores:
Prof. Esp. Aline Dayany de Lemos
Prof. M.e. Luciana Nishi
Prof. M.e. Francisco Edilson de Souza
Coordenadora dos Projetos Interdisciplinares: Prof. M.e Adrielle Beze Peixoto
Diretora dos cursos de Engenharia de Computação e Engenharia de Software: Prof. M.e Viviane Carla Batista Pocivi

Centro Universitário de Anápolis
Chanceler – Ernei de Oliveira Pina
Reitor – Carlos Hassel Mendes da Silva
Pró-Reitora Acadêmica – Cristiane Martins Rodrigues Bernardes
Pró-Reitor de Pós-Graduação, Pesquisa, Extensão e Ação Comunitária – Sandro Dutra e Silva
Coordenador da Pesquisa e Inovação – Bruno Junior Neves
Coordenador de Extensão e Ação Comunitária – Fábio Fernandes Rodrigues

Cursos de Engenharia de Computação e Engenharia de Software:
Diretora - Viviane Carla Batista Pocivi
Núcleo Docente Estruturante - Adrielle Beze Peixoto
Núcleo Docente Estruturante - Natasha Sophie Pereira
Núcleo Docente Estruturante - Renata Dutra Braga
Núcleo Docente Estruturante - Walquíria Fernandes Marins

Sumário

Indicação de Ícones

Saiba mais: oferece novas informações que enriquecem o assunto ou "curiosidades" e notícias recentes relacionadas ao tema estudado.

Glossário: indica a definição de um termo, palavra ou expressão utilizada no texto.

Atenção: indica pontos de maior relevância no texto.

Mídias integradas: sempre que se desejar que os estudantes desenvolvam atividades empregando diferentes mídias: vídeos, filmes, jornais e outras.

Palavra da Coordenação

A formação integral é uma preocupação constante na construção didático-pedagógica dos Bacharelados em Computação do Centro Universitário de Anápolis – UniEvangélica. Sua efetividade é garantida por meio da inserção de projetos interdisciplinares, como disciplinas integradoras na matriz curricular do curso, e objetiva que as habilidades e competências exigidas no perfil do egresso sejam fortalecidas por meio de projetos que contribuam para a integração dos conhecimentos teóricos, técnicos, práticos e interpessoais.

As disciplinas de Projeto Interdisciplinar (PI) promovem a associação entre os diferentes conteúdos, habilidades e cenários em projetos que favoreçam a construção do conhecimento científico, aliado à autoaprendizagem, proatividade, resolução conjunta de problemas, trabalho em equipe, reflexividade, entre outros. Para tanto, estas disciplinas têm início no primeiro período do curso e evolui em uma constante de maturidade pessoal, interpessoal e científica.

Esta revista apresenta o resultado dos trabalhos desenvolvidos nestas disciplinas e apresentados durante o SITES – Seminário Interdisciplinar de Tecnologia e Sociedade. Entendemos que este é um passo importante para o processo que vem sendo construído e intentamos que assim como o desenvolvimento deste material foi enriquecedor aos discentes do curso, o acesso aos mesmos, traga resultados positivos a aqueles que tiverem acesso.

Apresentação

Este material foi produzido por alunos e com a supervisão de professores do curso de Engenharia de Computação da UniEvangélica de Anápolis – Goiás. Este documento tem como intenção atender às competências e habilidades elencadas no Plano de Pedagógico do Curso e reiterado no Plano de Ensino da disciplina de Projeto Interdisciplinar IV.

A construção de um perfil profissional, interpessoal e técnico é um desafio que construímos juntos e ao longo do semestre. Despertar no aluno a capacidade de organizar e estruturar conhecimentos gerais e desafia-lo na tomada de ações além da capacidade de absolvição e reprodução de conteúdo. Proporcionar a construção de conteúdos técnicos para auxiliar colegas e a comunidade em geral, possibilitando a identificação de potencialidades e fraquezas sobre a sua comunicação verbal e escritas.

Em contraponto delíamos ao aluno do minicurso a responsabilidade quanto à sua responsabilidade de aprendizado. Em ambientes EAD, ou a responsabilidade do aluno é significativamente maior em relação à educação presencial. Esta modalidade pode ser um pouco nova para você. Estamos “conectados” em um mesmo local, porém não o estamos fisicamente. O aluno, a partir desta perspectiva, tem total responsabilidade sobre o seu aprendizado, então leia, releia e leia novamente se necessário, mas não deixe que o tempo passe e você não aprenda o que se propôs.

Sobre o Conteúdo

Neste documento vamos aprender um pouco sobre bancos de dados. Entenderemos como as informações são armazenadas em uma base, quais os modelos visuais são construídos e posteriormente aplicaremos em um sistema.

É um pouco de programação ainda, mas uma programação voltada a descobrir os dados e suas particularidades. Então mãos à obra...

Introdução

Um dos grandes benefícios de Banco de Dados ou Database é o controle de redundância de dados. É muito comum em organizações haver redundância de informações. Essa redundância muitas vezes é prejudicial, pois se em um setor “A” existe uma planilha chamada “Insumos Mensais” e num setor “B” uma cópia idêntica que precisa ser atualizada toda vez que a outra é modificada, possivelmente em um determinado momento um erro humano implicará na perda de informações, o que será prejudicial aos negócios dessa organização. Com um banco de dados essa redundância é controlada, senão eliminada. Daí, lidamos com a redundância controlada e a redundância não controlada de dados. A diferença entre elas é que a primeira se determina pelo sistema de software e a outra não. Se uma alteração é realizada em determinada categoria de dados, automaticamente, os dados replicados são atualizados e não há geração de prejuízo. Isso se deve ao compartilhamento de dados que pode ser organizado nos Sistemas Gerenciadores de Banco de Dados (Heuser, 2009).

O que é um banco de dados e para que serve?

Um Banco de Dados (BD) pode ser compreendido como um conjunto de dados ou arquivos agrupados em uma base de dados que possui um relacionamento entre si.

Podemos exemplificar isso da seguinte forma: imaginemos que exista um guarda-roupa e dentro dele vários objetos estão armazenados. O guarda-roupa assume, nesse exemplo, o papel de banco de dados e cada objeto armazenado dentro dele, os dados.

Figure 1 - Guarda Roupa

Para que os dados de um BD possam ser manuseados, é necessária a utilização de um conjunto de sistemas com as funções de inclusão, consulta (recuperação), modificação e exclusão. A esses sistemas damos o nome de Sistemas Gerenciadores de Banco de Dados ou SGBDs. No mercado existem vários tipos de SGBDs, porém o mais usado é o Relacional.

Quando precisamos construir BDs, devemos usar alguns modelos de dados, que facilitam a abstração e o desenvolvimento. Esse conjunto integra o chamado projeto de banco de dados (Heuser, 2009).

No projeto, o primeiro modelo construído é chamado de modelo conceitual, ele é totalmente abstrato e independente da implementação em um SGBD, ou seja, não precisamos nos preocupar como os dados serão arranjados no SGBD.

A partir do modelo conceitual podemos gerar o modelo lógico, este formaliza a organização dos dados do modelo anterior e tem sua visão no nível do usuário de SGBD, ou seja, possui conceitos utilizados na implementação física de um SGBD. Assim perceberemos que este tipo de modelo depende do SGBD utilizado, no nosso caso, o tipo relacional.

Por último, temos o modelo físico, nessa etapa o modelo de dados deve ter sido normalizado de acordo com as formas normais (FN) de 1 a 4[1]. No modelo físico os dados são implementados no software de SGBD e podem ser integrados a um sistema de software real.

https://www.youtube.com/watch?v=4HTOhhERyqY

.

Modelo Conceitual

É uma descrição do banco de dados, a qual é responsável por analisar os requisitos de acordo com as necessidades do usuário para assim definir os processos que estrutura o produto. Além disso, tem também como característica indicar as regras de negócio de um sistema.

Essa etapa irá ajudar no entendimento dos processos, que tem como objetivo criar um sistema coerente de objetos, propriedades e relações estruturadas para o domínio da tarefa do usuário.

Esse modelo é a descrição proposta na forma de um conjunto de ideias e conceitos a respeito de o que o sistema deve fazer, como deve se comportar e de como deve se parecer.

O desenvolvimento desse modelo consiste no entendimento dos requisitos com base nas necessidades do usuário e suposições, que nada mais são do que reflexões que poderão somar e complementar as necessidades inicialmente traçadas.

A partir daqui iremos entender um pouco mais dos componentes do diagrama de entidade-relacionamento (DER).

Entidades

Entidade é algo da realidade modelada onde se deseja manter informações no banco de dados, como por exemplo, em uma empresa na qual as entidades podem ser o chefe, a secretária, o horário, o salário, entre outros. Uma entidade pode representar tantos objetos concretos (chefe), quanto objetos abstratos (salário). Ela é representada por um retângulo que contém o nome da entidade, como representado abaixo.

Figure 2 - Entidade

Relacionamentos

Relacionamento é um conjunto de associações entre entidades, ele é representado por um losango, que fica ligado às entidades participantes do relacionamento por linhas. Vejamos os exemplos abaixo.

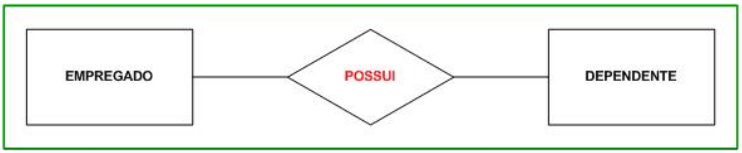


Figure 3 – Relacionamento

Pelo modelo abaixo podemos observar que um relacionamento (possui) entre as entidades EMPREGADO e DEPENDENTE. Considere as seguintes questões:

Um empregado pode não ter dependentes?

Um dependente pode ter mais de um empregado associado?

Determinado empregado pode possuir mais de um dependente?

Pode existir dependente sem algum empregado associado?

As respostas desse questionamento vão depender do problema a ser modelado. Para poder expressar essas ideias no modelo é necessário definir uma propriedade importante do relacionamento a cardinalidade.

Entidade Associativa

Existem situações onde é necessário criar uma relação entre entidades e seu relacionamento para isso criamos uma entidade associativa. Ela ocorre quando tem que se associar uma entidade dentro de um relacionamento. Segundo Alberto, [1998, p.32] "na modelagem ER não foi prevista a possibilidade de associar uma entidade com um relacionamento ou então de associar dois relacionamentos entre si. Na prática, quando está-se construindo um novo DER ou modificando um DER existente, surgem situações em que é desejável permitir a associação de uma entidade a um relacionamento"

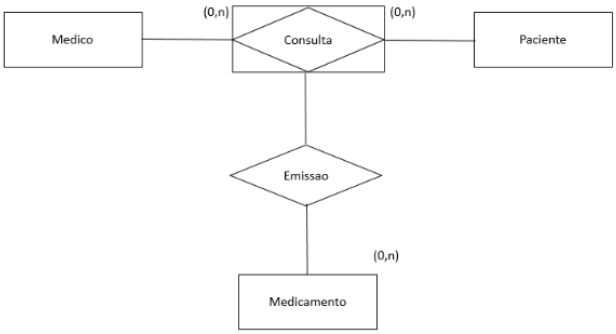


Figure 4 - Entidade Associativa

Cardinalidades

A cardinalidade é um número que expressa o comportamento (número de ocorrências) de determinada entidade associada a uma ocorrência da entidade em questão através do relacionamento.

Existem dois tipos de cardinalidade: mínima e máxima. A máxima, expressa o número máximo de ocorrências de determinada entidade associada a uma ocorrência da entidade em questão através do relacionamento. A mínima, expressa o número mínimo de ocorrências de determinada entidade associada a uma ocorrência da entidade em questão através do relacionamento.

No modelo abaixo está representado as cardinalidades mínima e máxima:

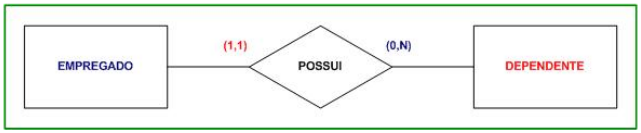


Figure 5 - Cardinalidades

Para fazer a leitura do modelo, partimos de uma entidade e a cardinalidade correspondente que é representada no lado oposto. No exemplo, a cardinalidade (0:N) faz referência a EMPREGADO, já a cardinalidade (1:1) faz referência a DEPENDENTE. Como lemos:

> ***Uma*** *ocorrência de empregado pode não estar associada a* ***uma*** *ocorrência de dependente ou pode estar associada a* ***várias*** *ocorrências dele (determinado empregado pode não possuir dependentes ou pode possuir vários) e* ***uma*** *ocorrência de dependente está associada a apenas* ***uma*** *ocorrência de empregado (determinado dependente possui apenas um empregado responsável).*

As cardinalidades máximas distinguimos dois tipos: 1(um) e N (qualquer cardinalidade maior que 1). Para as mínimas distinguimos também dois tipos: 0(zero) e 1(um).

Atributos

Atributo é uma característica associada a cada ocorrência de entidade ou relacionamento. Como no exemplo:

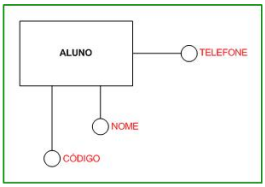


Figure 6 - Atributos

Nesse caso os atributos são: telefone, nome e código, tudo relacionado ao aluno.

Os BDs relacionais são caraterizados por possuir seus dados agrupados por categorias e por elas possuírem um relacionamento com outras. Cada categoria possui características, isto é, peculiaridades específicas. Essas categorias recebem o nome de **entidade** e suas características são chamadas de **atributos**. Quando pretendemos especificar um objeto de uma entidade, o chamamos de ocorrência ou instância da entidade (Heuser, 2009).

Os atributos podem ser de dois tipos, atributos propriamente ditos, os quais representam as características das entidades e os atributos que identificam a entidade dentre as demais. Esse atributo é único, é o **atributo identificador**. Todavia, uma entidade pode ser identificada também por um conjunto de atributos (atributo composto) ou, até mesmo, pela ligação entre as entidades (relacionamento identificador).

Generalização / Especialização

A generalização / especialização são conceitos usados para representar objetos do mundo real que possuem os mesmos atributos que podem ser categorizados e ser representados em uma hierarquia que mostra as dependências entre entidades de uma mesma categoria.

No exemplo, vamos imaginar uma empresa de seguros que vende para seus clientes, que podem ser tanto cidadãos ou como empresas. Nesse caso teríamos o cliente que pode ser pessoa jurídica ou pessoa física. No DER é representado da seguinte maneira:

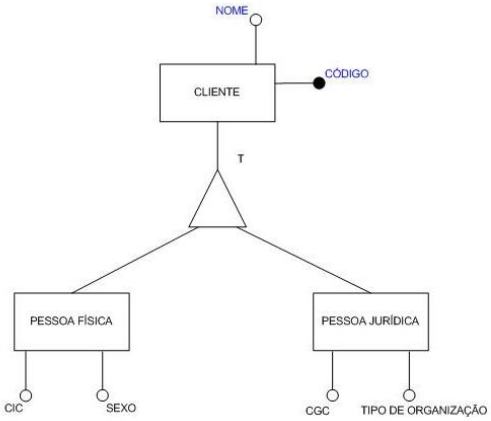


Figure 7 - Generalização / Especialização

Outro exemplo, seria pensar em uma empresa de planos de saúde na qual poderíamos ter o médico, esse que pode ser médico residente ou médico efetivo. Abaixo estão os indicadores de generalização e especialização:

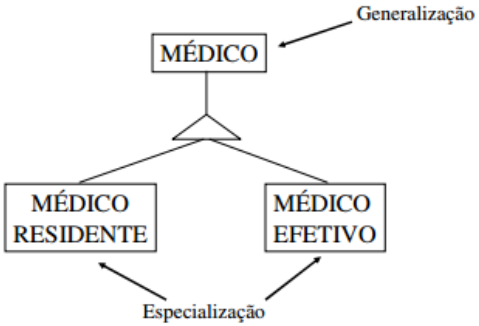


Figure 8 - Generalização / Especialização

A generalização/especificação pode ser classificada em dois tipos:

Parcial: nem toda ocorrência da entidade genérica possui uma ocorrência correspondente em uma entidade especializada.

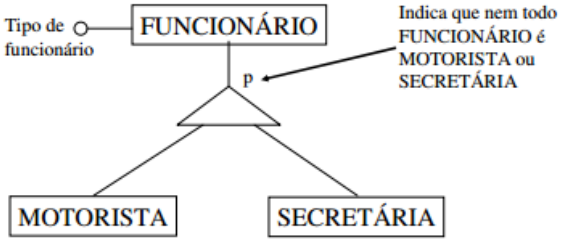


Figure 9 - Especialização Parcial

No exemplo, nem todo funcionário é motorista ou secretária.

Total: para toda ocorrência da entidade genérica existe sempre uma ocorrência em uma das entidades especializadas.

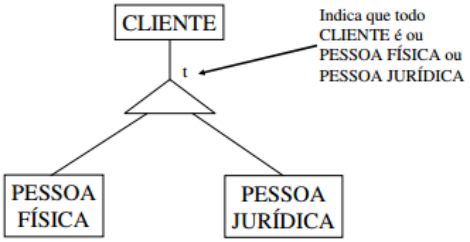


Figure 10 - Especialização Total

No exemplo acima, todo cliente ou é uma pessoa física ou uma pessoa jurídica. Não existe a possibilidade de haver um cliente que não seja pessoa física ou pessoa jurídica.

Exclusiva ou Compartilhada: uma ocorrência de entidade geral deve ou não, ser especializada uma única vez. Ou seja, uma entidade especializada pode atuar com o papel de outra entidade especializada (compartilhada) ou só pode exercer o seu papel (exclusiva) (Heuser, 2009).

No diagrama ER, as generalizações/especializações podem ser combinadas de formas diferentes. Podemos identificar isso através de uma simbologia muito simples como mostra a tabela abaixo:

	Exclusiva (x)	Compartilhada (c)
Total (t)	***xt***	***ct***
Parcial (p)	***xp***	***cp***

Figure 11 - Generalização / Especialização

O uso da Generalização é indicado quando existe algum atributo que seja aplicável a mais de uma entidade no Modelo Entidade Relacionamento.

Exercício CESGRANRIO-2016

Uma empresa que atua no ramo de entrega de encomendas precisa de um sistema de informação para controlar sua principal atividade. Durante o levantamento dos requisitos desse sistema, as seguintes informações sobre o pagamento de entregas foram fornecidas por um funcionário da empresa:

• Uma entrega é identificada internamente por um código. Além disso, é necessário registrar o peso e as dimensões do objeto a ser entregue, o endereço de entrega e o custo da operação de entrega;

• Uma entrega pode ser paga através de cartão de crédito, cartão de débito ou boleto bancário. Visando a atender às demandas de seus clientes, uma entrega pode ser paga usando-se qualquer combinação desses três meios de pagamento;

• Em relação a um pagamento com cartão de crédito, o sistema deve registrar a data de pagamento, o valor pago, o número do cartão, sua data de validade e a instituição que o emitiu;

• Em relação a um pagamento com cartão de débito, o sistema deve registrar a data de pagamento, o valor pago, o número do banco e os números da agência e da conta corrente às quais o cartão está vinculado;

• Em relação a um pagamento com boleto bancário, o sistema deve registrar a data de pagamento, o valor pago e o número do código de barras do boleto;

• Cada pagamento registrado se refere a uma única entrega;

• Não há entrega registrada no sistema que não tenha, pelo menos, um pagamento associado a ela.

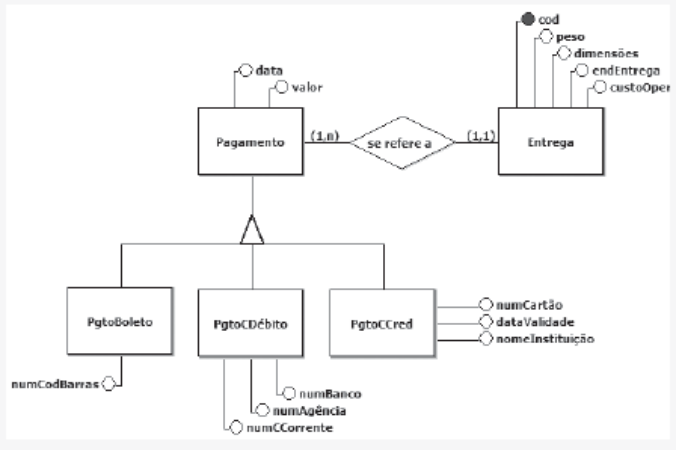


Figure 12 - Exemplo Diagrama

Modelo Lógico

Segundo (Heuser,2009) "o modelo lógico é o modelo de dados que representa a estrutura de dados de um banco de dados conforme vista pelo usuário do SGBD", ou seja, ele é a transformação do modelo conceitual em uma estrutura de dados que possibilita ao usuário de SGBD visualizar como será seu banco.

Tabela

Como estudado no modelo conceitual, as categorias das informações armazenadas no BD estão contidas em entidades. No modelo lógico, cada entidade é transformada em uma tabela. Uma **tabela** é formada por linhas e colunas, as linhas são chamadas de tuplas e armazenam os dados da ocorrência de uma entidade.

Uma tupla possui **campos,** que são a representação dos atributos. Cada campo recebe um título que, na verdade, é o título do atributo. Dizemos que o conjunto de campos semelhantes forma uma coluna (Heuser, 2009).

É muito importante evidenciar que cada campo é único dentro de uma tabela e que seus valores também são únicos. Portanto, não deve existir um campo composto nem que armazene vários valores numa mesma linha, isto é, multivalorado.

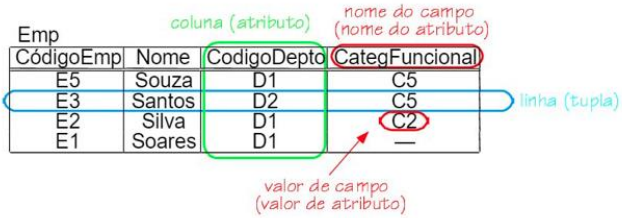


Figure 13 - Exemplo Tabela

Chave

No modelo lógico é apresentado, a você estudante, o conceito de **chave**. Elas são responsáveis, dentre outras coisas, por prover a identificação das linhas de uma tabela e possibilitar o relacionamento entre várias outras tabelas. Entre os tipos de chave existentes usaremos as seguintes no minicurso: primária, estrangeira e alternativa.

A **chave primária** tem a mesma função do atributo identificador: serve para diferenciar uma linha das demais numa tabela. Obedecendo às restrições de integridade, a chave primária deve ser única dentro de uma tabela e obrigatoriamente não pode ser nula.

As entidades são identificadas por um atributo identificador, as tabelas por sua vez, são identificadas por uma chave primária. Assim cada tabela possui uma coluna que a identifica. E cada linha dessa coluna, identifica um registro da tabela.

As **chaves estrangeiras** exercem um papel tão importante quanto as chaves primárias. Elas são responsáveis por realizar o relacionamento entres as tabelas no BD. A chave estrangeira de uma tabela, nada mais é que a representação da chave primária de outra tabela nesta. Isto é, uma chave primária de uma tabela quando presente em outra tabela a fim de criar um relacionamento, transforma-se em chave estrangeira. Esse relacionamento funciona da seguinte maneira: imaginemos que uma editora publica vários livros. Assim cada livro deve armazenar um vínculo com sua editora para que seja possível haver um relacionamento entre eles.

CodEditora	Nome	Cidade
01	Saraiva	São Paulo
02	Pearson	Belo Horizonte
03	Novatec	Goiânia

Figure 14 - Tabela Editora - Chave Primária

CodLivro	Nome	CodEditora
011	Java Descomplicado	02
026	O Pequeno Príncipe	03
032	Eterno Amanhã	01
011	SQL Essencial	02

Figure 15 - Tabela Livro - Chave Primária e Estrangeira

Chaves estrangeiras podem desempenhar o papel de chave primária de uma tabela no caso, por exemplo, de generalização/especialização, onde são criadas tabelas provenientes das generalizações. Nesse caso, uma tabela especializada tem como chave estrangeira e primária, a chave primária da tabela generalizada.

A chave estrangeira pode também relacionar a mesma tabela, ou seja, uma tabela pode ter como chave estrangeira sua própria chave primária, esse cenário se sucede quando temos auto relacionamentos: uma ocorrência de uma entidade se relaciona com ela mesma.

Por último temos as **chaves alternativas**. Essas chaves são todas as outras colunas da tabela que não são chaves primárias ou estrangeiras.

Integridade

Um SGBD deve prover a integridade dos dados presentes no banco. Heuser conclui que os dados possuem integridade quando representam confiavelmente a realidade do BD.

De acordo com Heuser, ainda podemos listar os seguintes tipos de integridade de dados:

Integridade de Domínio: os valores de um campo devem estar de acordo com o tipo de dados que aquele campo pode armazenar. Inserir um valor numérico em campo do tipo INTEGER está correto, mas inserir um caractere em um campo que está preparado para receber um inteiro viola a integridade dos dados e o SGBD não permite a operação.

Integridade de Vazio: determina se os campos de uma coluna podem ser vazios ou não.

Integridade de Chave: determina que os valores, especialmente da chave primária, devem ser únicos.

Integridade Referencial: os campos presentes numa chave estrangeira devem obrigatoriamente estar na coluna de chave primária da tabela a qual se deseja realizar o relacionamento.

Representação do Modelo Lógico

Quando transformamos o modelo conceitual para o lógico podemos representar este último de duas formas diferentes para enfim gerar o modelo físico. A primeira representação é textual e a segunda é visual, ou diagramática

Na representação textual as tabelas têm um nome e seus campos vêm imediatamente após, dentro de parênteses. A chave primária é identificada por um sublinhado e geralmente é o primeiro termo após a abertura dos parênteses (Heuser, 2009).

Nome_tabela (Chave_primária, Campo1, CampoN)

Livro (CodLivro, ISBN, Titulo, Edicao)

As chaves estrangeiras são representadas como os demais campos, mas após a declaração da tabela existe a referência da chave estrangeira à tabela onde ela é chave primária.

Nome_tabela (Chave_primária, Chave_estrangeira, CampoN)

Chave_estrangeira referencia Tabela_origem

Livro (CodLivro, CodEditora, ISBN, Titulo, Edicao)

CodEditora referencia Editora

Implementação de Relacionamentos

Quando transformamos o modelo conceitual para o modelo lógico não temos mais uma figura geométrica que indica os relacionamentos, a partir de agora, como já dito, eles são realizados através das chaves estrangeiras. Para exemplificar as implementações, recolhemos exemplos de domínios diferentes para uma melhor compreensão e também porque nem todas essas implementações estão presentes na situação problema trabalhada ao longo do minicurso.

- **N:N:** Um relacionamento é convertido em tabela quando é do tipo N:N, logo a nova tabela recebe como chave estrangeira, as chaves primárias das tabelas relacionadas. Essas chaves podem ou não ser chave primária dessa nova tabela. Caso sejam, a tabela terá uma chave primária composta. No entanto, podemos criar uma chave primária para esta nova tabela e as outras serão apenas chaves estrangeiras comuns (Heuser, 2009).

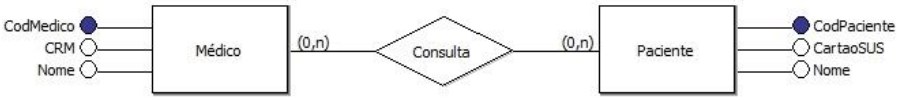


Figure 16 - Relacionamento N:N

Figure 17 - Relacionamento N:N - Chave Primária Simples

Figure 18 - Relacionamento N:N - Chave Primária Composta

- **1:N:** Quando o relacionamento é de 1:N, a tabela com cardinalidade 1 recebe como chave estrangeira, a chave primária da tabela com cardinalidade N. Assim, na tabela com cardinalidade 1 cria-se uma coluna para abrigar a chave estrangeira. Existe um macete para sabermos qual das tabelas receberá a chave estrangeira, basta observar, no diagrama do modelo conceitual, a entidade que possui ao seu lado a representação da cardinalidade N (Heuser, 2009).

Figure 19 - Relacionamento 1:N

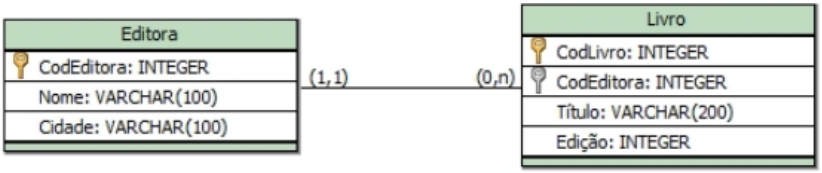


Figure 20 - Relacionamento 1:N

- **1:1:** E quando o relacionamento for 1:1? Quando isso acontecer, convenciona-se fundir as entidades formando uma tabela única. Essa tabela possuirá o nome da entidade mais forte do relacionamento (Para saber sobre a força das entidades consulte o livro: Projeto de Banco de

Dados, Carlos Alberto Heuser) e os atributos de ambas as entidades participantes do relacionamento. Caso o relacionamento possua algum atributo, a tabela fundida o receberá como um campo (Heuser, 2009).

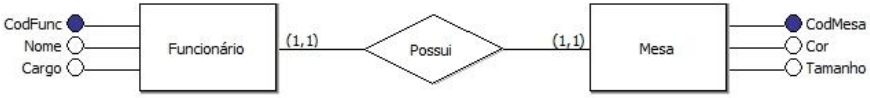


Figure 21 - Relacionamento 1:1

Figure 22 - Relacionamento 1:1

Generalização/Especialização

Para concluir temos a transformação de uma generalização/especialização do modelo conceitual para o lógico. A técnica que empregaremos será a da criação de uma tabela por entidade especializada. Existem outras, mas vimos nessa maior adequação e simplicidade. Continuando, pegamos a entidade geral e criamos uma tabela para ela com sua devida chave primária e os outros campos. Para as entidades especializadas, criamos tabelas também, mas suas chaves primárias serão estrangeiras, pois referenciarão a tabela geral.

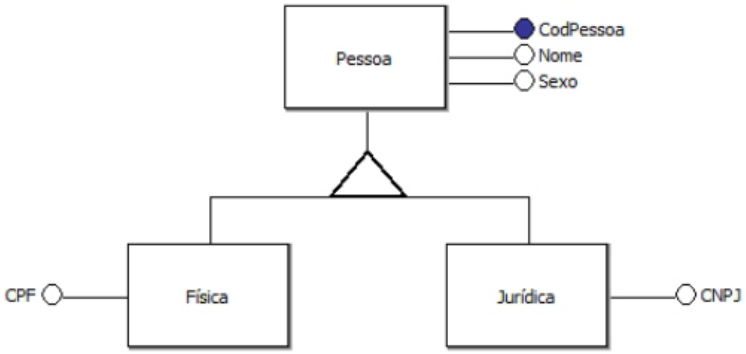


Figure 23 - Especialização / Generalização

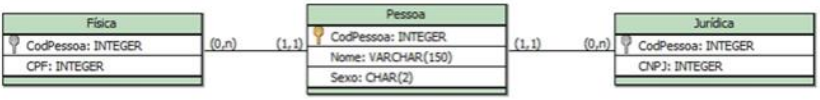


Figure 24 - Especialização / Generalização

Normalização

Tendo o esquema relacional inerente ao documento, passa-se ao processo de normalização. Esse processo é feito pelo conceito de forma normal sendo este último uma regra submetida por uma tabela que sendo feita por essa regra será considerada "bem projetada". Iremos abordar essas diversas formas normais, ou seja, diversas regras com níveis cada vez mais rígidos, para verificar tabelas relacionais. Serão relevadas quatro formas normais nomeadas simploriamente de primeira, segunda, terceira e quarta forma normal, abreviadamente 1FN, 2FN, 3FN.

Primeira forma normal - 1FN

Uma relação está na 1FN quando todos os domínios básicos não contiverem grupos repetitivos, podemos definir que a 1FN não admite repetições ou campos que tenha mais que um valor. Diz-se que a tabela está na primeira forma normal quando ela não contem tabelas aninhadas.

Primeiro iniciamos a normalização com a transformação do esquema de tabelas não-normalizadas em um esquema relacional na primeira forma normal.

Temos duas alternativas para passagem de um esquema de tabela não-normalizada em um esquema na 1FN:

- Criar uma tabela única com redundância de dados, ou seja, os dados das linhas externas à tabela aninhada são repetidos para cada linha da tabela aninhada.

- Criar uma tabela pra cada tabela aninhada, ou seja, fazer uma tabela referente a própria tabela que está sendo normalizada e uma tabela pra cada tabela aninhada.

Iremos aplicar os seguintes procedimentos padrões mais recomendados para aplicar a 1FN:

- Identificar a chave primária da entidade;
- Identificar o grupo repetitivo e removê-lo da entidade;
- Criar uma nova entidade com a chave primária da entidade anterior e o grupo repetitivo.
-

Exemplo de normalização de tabela para 1FN:

Considere a tabela cliente e suas respectivas colunas:

- Cliente
  - Código_cliente
  - Nome
  - Telefone
  - Endereço

Agora a tabela com os dados:

Código_cliente	Nome	Telefone	Endereço
C001	José	9563-6352 9847-2501	Rua Seis, 85 Morumbi 12536-965
C002	Maria	3265-8596	Rua Onze, 64 Moema 65985-963
C003	Janio	8545-8956 9598-6301	Praça ramos Liberdade 68858-633

Figure 25 - Tabela Cliente

Analisando veremos que todos os clientes possuem Rua, CEP e Bairro, e essas informações estão na mesma célula da tabela, logo ela não está na 1FN. Para normalizar, deveremos colocar cada informação em uma coluna diferente, como no exemplo a seguir:

Código_cliente	Nome	Telefone	Rua	Bairro	Cep
C001	José	9563-6352 9847-2501	Rua Seis, 85	Morumbi	12536-965
C002	Maria	3265-8596	Rua Onze, 64	Moema	65985-963
C003	Janio	8545-8956 9598-6301	Praça ramos	Liberdade	68858-633

Figure 26 - Tabela Cliente

Mesmo assim a tabela ainda não está na 1FN, pois há clientes com mais de um telefone e os valores estão em uma mesma célula. Para normalizar será necessário criar uma nova tabela para armazenar os números dos telefones e o campo-chave da tabela cliente. Veja o resultado a seguir:

Código_cliente	Nome	Rua	Bairro	Cep
C001	José	Rua Seis, 85	Morumbi	12536-965
C002	Maria	Rua Onze, 64	Moema	65985-963
C003	Janio	Praça ramos	Liberdade	68858-633

Figure 27 - Tabela Cliente

Codigo_cliente	Telefone
C001	9563-6352
C001	9847-2501
C002	3265-8596
C003	8545-8956
C003	9598-6301

Figure 28 - Tabela Telefone

No exemplo acima foi gerado uma segunda tabela para que a 1FN fosse atendida.

Segunda Forma Normal - 2FN

Para início de entendimento é importante ressaltar que para a tabela estar na 2FN é preciso já estar na 1FN. Cada coluna não chave depende da chave primária completa. Se a tabela não estiver em sua segunda forma normal provavelmente terá dependências parciais (quando uma coluna depende apenas de parte de uma chave primária composta), isto é, contém colunas não chave que dependem unicamente de uma parte da chave primária.

Obviamente, uma tabela que está na 1FN e que possui apenas uma coluna como chave primária não contém dependências parciais, já que, nesta tabela, é possível uma coluna depender de uma chave primária, visto que a chave primária não é composta por partes (por diversas colunas). Assim, toda tabela que está na 1FN e que possui apenas uma coluna como chave primária já está na 2FN. O mesmo aplica-se para tabela que contenha apenas colunas chave primária. (HEUSER, 2009, pág. 197)

Se o nome do produto já existe na tabela produtos, então não é necessário que ele exista na tabela de produtos. A segunda forma normal trata destas anomalias e evita que valores fiquem em redundância no banco de dados.

Procedimentos:

- Identificar os atributos que não são funcionalmente dependentes de toda a chave primária;
- Remover da entidade todos esses atributos identificados e criar uma nova entidade com eles.
- A chave primária da nova entidade será o atributo do qual os atributos removidos são funcionalmente dependentes.

Exemplo de normalização de tabela para 2FN:

Considere a tabela vendas e suas colunas:

- Vendas
  - N_pedido
  - Código_produto
  - Produto
  - Quant
  - Valor_unit
  - Subtotal

Agora a tabela com os dados:

N_pedido	Codigo_produto	Produto	Quant	Valor_unit	Subtotal
1005	1-934	Impressora laser	5	1.500,00	7.500,00
1006	1-956	Impressora desjet	3	350,00	1.050,00
1007	1-923	Impressora matricial	1	190,00	190,00
1008	1-908	Impressora mobile	6	980,00	5.880,00

Figure 29 - Tabela Vendas

Analisando veremos que o nome do produto depende do código do produto, porém não depende de N_pedido que é a chave primária da tabela, portanto não está na segunda forma normal. Isto gera problemas com a manutenção dos dados, pois se houver alteração no nome do produto teremos que alterar em todos os registros da tabela venda.

Para normalizar esta tabela teremos de criar a tabela Produto que ficará com os atributos Código_produto e produto e na tabela Venda manteremos somente os atributos N_pedido, código_produto, quant, valor_unit e subtotal. Veja o resultado abaixo:

Codigo_produto	Produto
1-934	Impressora laser
1-956	Impressora desjet
1-923	Impressora matricial
1-908	Impressora mobile

Figure 30 - Tabela Produto

N_pedido	Codigo_produto	Quant	Valor_unit	Subtotal
1005	1-934	5	1.500,00	7.500,00
1006	1-956	3	350,00	1.050,00
1007	1-923	1	190,00	190,00
1008	1-908	6	980,00	5.880,00

Figure 31 - Tabela Vendas

Terceira forma normal - 3FN

Uma tabela encontra-se na 3FN quando, além de estar na 2FN, toda coluna não chave depende diretamente da chave primária, isto é, quando não há dependências funcionais transitivas diretas ou indiretas. Uma dependência funcional transitiva ou indireta acontece quando uma coluna não chave primária depende funcionalmente de outra coluna ou combinação de colunas não chave primária. (HEUSER, 2009, pág. 200)

Na terceira forma normal temos de eliminar aqueles campos que podem ser obtidos pela equação de outros campos da mesma tabela.

São usados os seguintes procedimentos:

- Identificar todos os atributos que são funcionalmente dependentes de outros atributos não chave;
- Removê-los.

A chave primária da nova entidade será o atributo do qual os atributos removidos são funcionalmente dependentes.

Considere a tabela abaixo:

N_pedido	Codigo_produto	Quant	Valor_unit	Subtotal
1005	1-934	5	1.500,00	7.500,00
1006	1-956	3	350,00	1.050,00
1007	1-923	1	190,00	190,00
1008	1-908	6	980,00	5.880,00

Figure 32 - Tabela Vendas

Considerando ainda a nossa tabela Venda, veremos que a mesma não está na terceira forma normal, pois o subtotal é o resultado da multiplicação Quant X Valor_unit, desta forma a coluna subtotal depende de outras colunas não chave.

Para normalizar esta tabela na terceira forma normal teremos de eliminar a coluna subtotal, como no exemplo a seguir:

N_pedido	Codigo_produto	Quant	Valor_unit
1005	1-934	5	1.500,00
1006	1-956	3	350,00
1007	1-923	1	190,00
1008	1-908	6	980,00

Figure 33 - Tabela Vendas

Sistemas Gerenciadores de Bancos de Dados

A linguagem SQL foi projetada para Sistemas de Bancos de Dados Relacionais, geralmente considerada uma linguagem de declaração.

De certa forma, ela existe desde os anos 70 e é tão presente quanto o próprio gerenciamento de dados. O modelo foi primeiramente introduzido em um paper intitulado “A Relational Model of Data for Large Shared Data Banks” escrito por E. F. Codd. A linguagem foi intitulada na época como SEQUEL para Structured English Query Language; de forma que SQL pode ser ocasionalmente pronunciado dessa forma.

Os produtos relacionados a SQL têm sido comercializados desde 1979, quando eles foram introduzidos a um precursor do Oracle. Logo a IBM seguiu o exemplo. Atualmente o SQL é usado em muitos produtos open source. Geralmente eles incluem SQL no nome. PostgreSQL está entre um dos mais conhecidos.

Entre as operações que podemos executar ao trabalhar com SQL, podemos listar as seguintes:

- Executar estruturação em um banco de dados.
- Retornar dados de um banco de dados.

- Inserir dados em um banco de dados.
- Realizar a atualização de dados em um banco de dados.
- Deletar dados de um banco de dados.
- Criar novos bancos de dados.
- Criar novas tabelas em um banco de dados.

Embora seja um padrão ANSI, existem diferentes versões da linguagem SQL. No entanto, para ser compatível com o padrão ANSI, todas elas suportam pelos menos os principais comandos (*SELECT, UPDATE, DELETE, INSERT, WHERE*).

Iniciando o SQL Server

1. Abra o Microsoft SQL Server Management Studio.

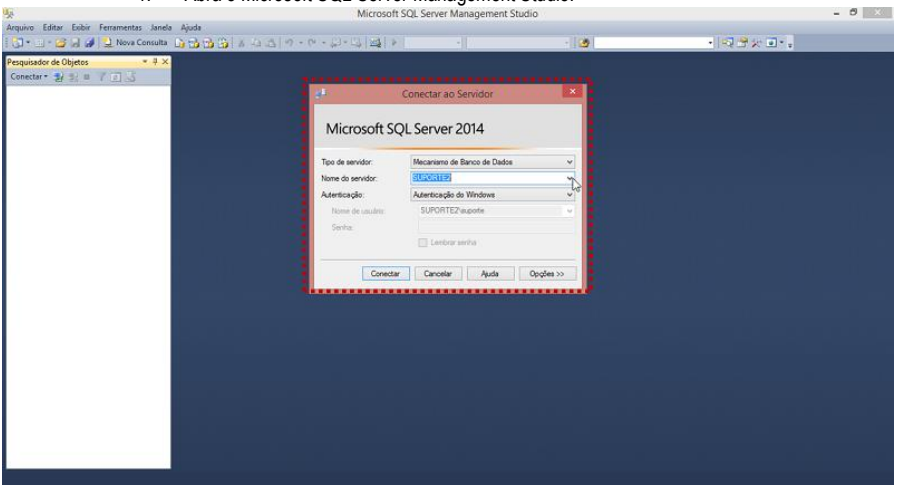


Figure 34 - SqlServer Conectar

2. Na janela **Conectar ao Servidor** selecione o nome do Usuário Local em Nome de Servidor

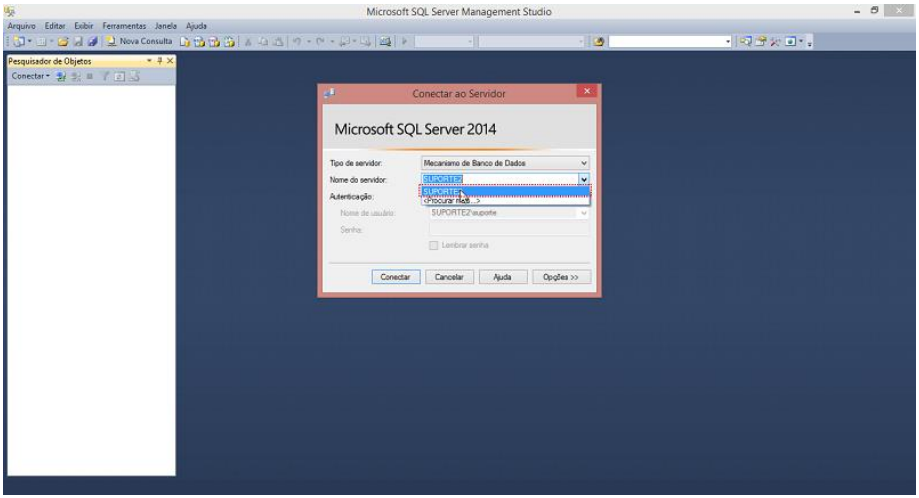


Figure 35 - SqlServer Conectar Servidor

3. Clique em *Conectar* no Botão Esquerdo do Mouse e aguarde a conexão ao banco.

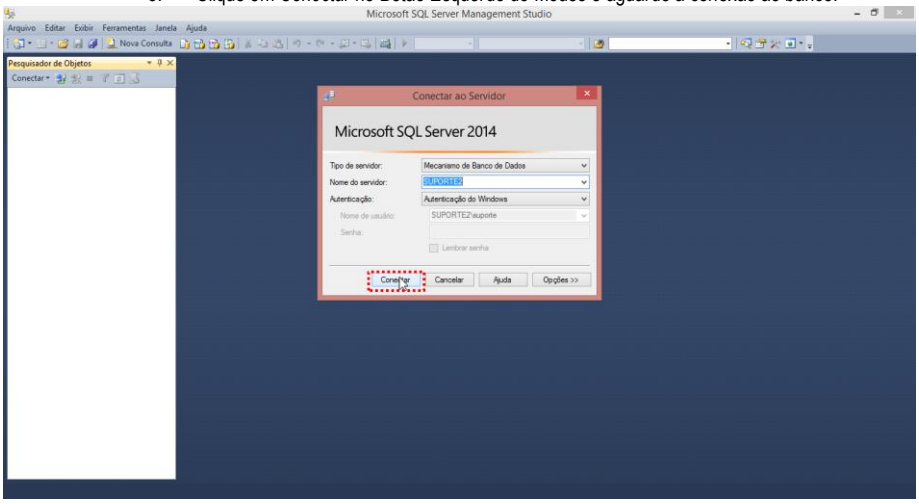


Figure 36 - SqlServer Conectar

4. Na aba esquerda do Microsoft SQL Server, onde está escrita Propriedades de Objetos, estarão apresentados os bancos de dados instanciados no computador que está servindo como servidor.

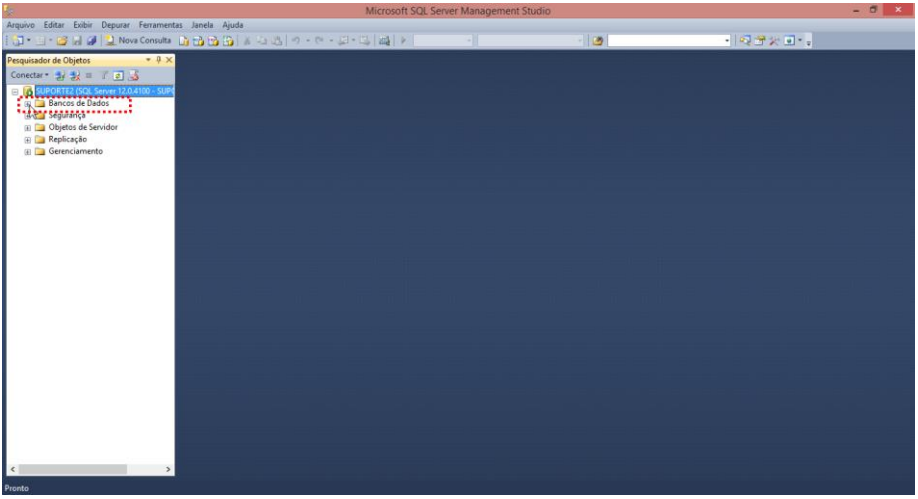


Figure 37 - SqlServer Base de Dados

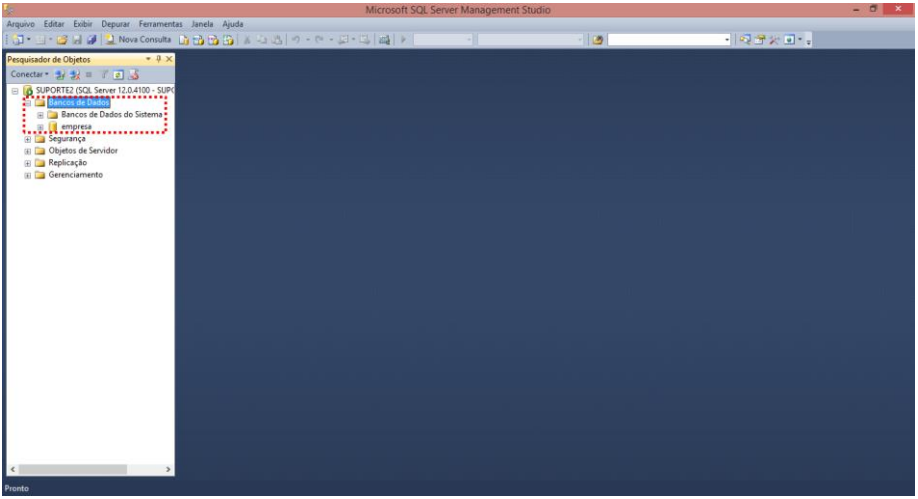


Figure 38 - SqlServer Base de Dados

5. Para realizar qualquer alteração através de comandos, primeiramente é necessário clicar no ícone *Nova Consulta* na parte superior da Janela.

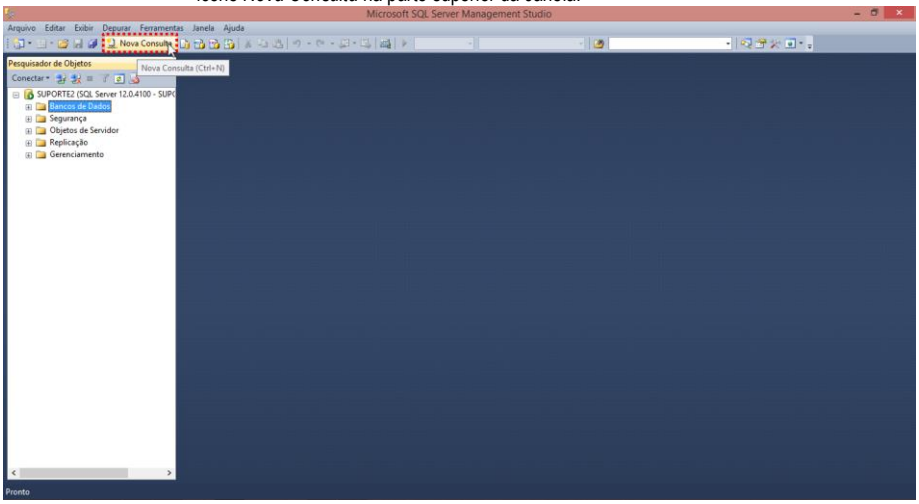


Figure 39 - SqlServer Nova Consulta

6. Será aberta uma nova consulta para que você possa começar a operar o software.

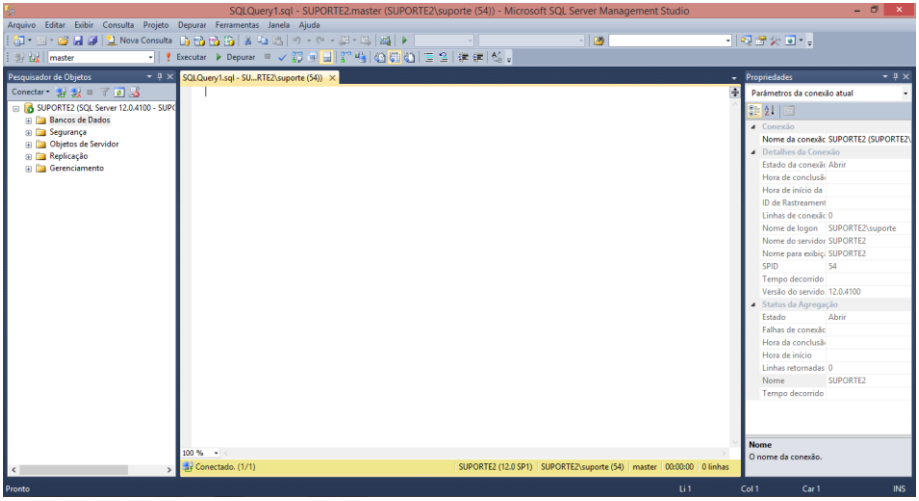


Figure 40 - SqlServer Janela Nova Consulta

Linguagem SQL

DML – **Data Manipulation Language (Linguagem de Manipulação de Dados):** É utilizado para realizar inclusões, consultas, exclusões e alterações de dados. Utiliza os comandos INSERT, SELECT, UPDATE e DELETE.

DDL - Data Definition Language (Linguagem de Definição de Dados): Permite ao desenvolvedor definir tabelas e elementos associados. Utiliza como comandos principais, o CREATE e o DROP. Alguns sistemas de banco de dados utilizam o ALTER. Comandos que utilizam CREATE DATABASE, DROP DATABASE, CREATE TABLE, DROP TABLE, ALTER TABLE.

DCL - Data Control Language (Linguagem de Controle de Dados): Controla os aspectos de autorização de dados e a utilização de licenças por usuários. Os principais comandos são GRANT e REVOKE

DTL – **Data Transaction Language (Linguagem de Transação de Dados):** Utilizado pelos desenvolvedores em transações. Os principais comandos são COMMIT e ROLLBACK. Comandos que utilizam: TRANSACTION

DQL - Data Query Language (Linguagem de Consulta de Dados): O mais importante dentre estes subconjuntos, pois consultas são realizadas a todo instante. O comando que é utilizado pelo DQL é o SELECT. Algumas literaturas constam que SELECT também é DML.

CREATE DATABASE

Para criar um banco de dados no SQL Server é necessário primeiramente abrir uma Nova Consulta como apresentado anteriormente.

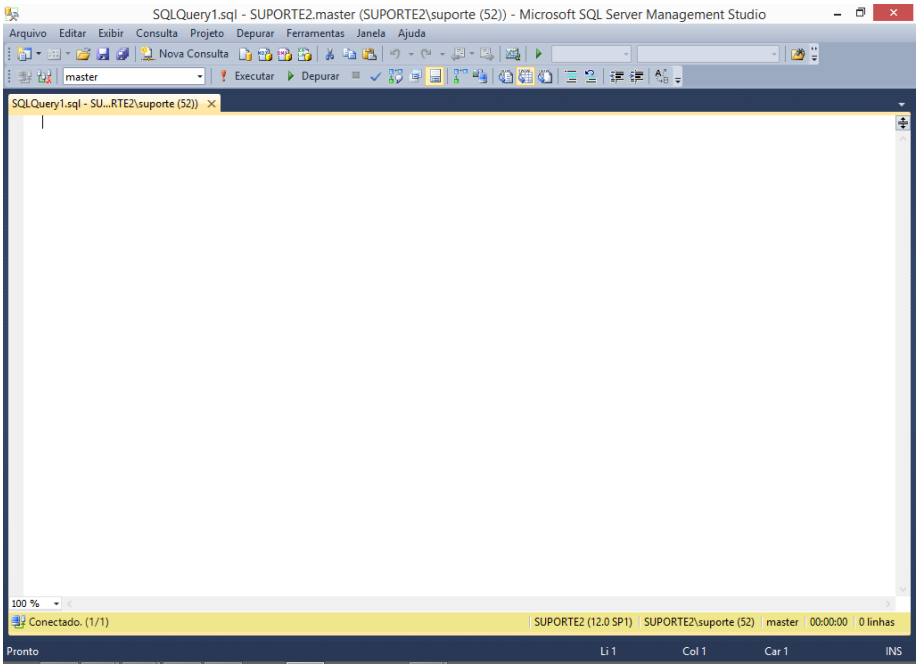


Figure 41 - SqlServer Nova Consulta

O comando padrão para criação de uma nova entidade é **CREATE**, para criar um banco de dados se utiliza **CREATE DATABASE**, a seguir é demonstrado a aplicação do comando. O trecho [nome_do_banco] deve ser substituído pelo nome que se deseja dar ao banco de dados:

Sintaxe SQL **CREATE DATABASE**

CREATE DATABASE nome_do_banco

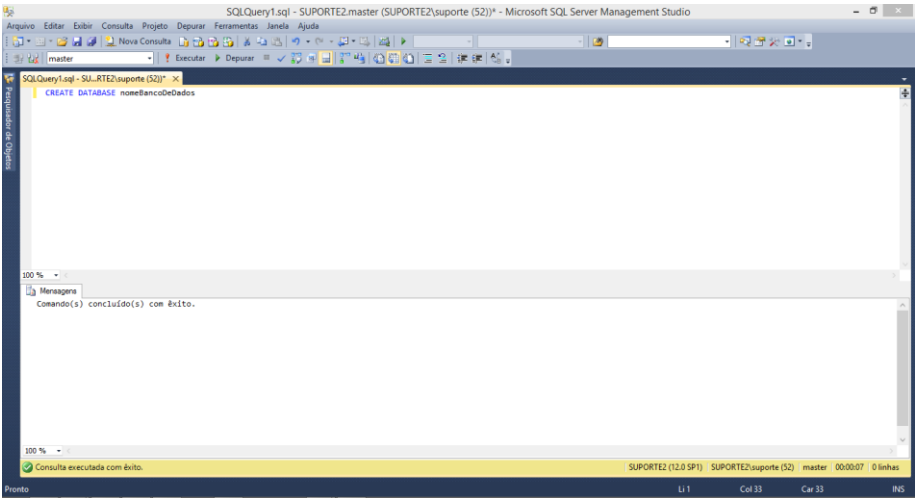


Figure 42 - SqlServer Create Database

Para exemplificar a criação de um banco de dados vamos utilizar um sistema simples de empréstimo de livros de uma biblioteca.

1. Primeiramente é necessário instanciar o banco de dados usando o comando **CREATE DATABASE**, nesse caso daremos o nome para o banco de *biblioteca*.
2. Para executar o código, é só clicar no ícone *executar* (F5) na parte superior da tela.
3. Caso o código não possua erros irá aparecer à mensagem Comando(s) concluído(s) com êxito na aba mensagens.

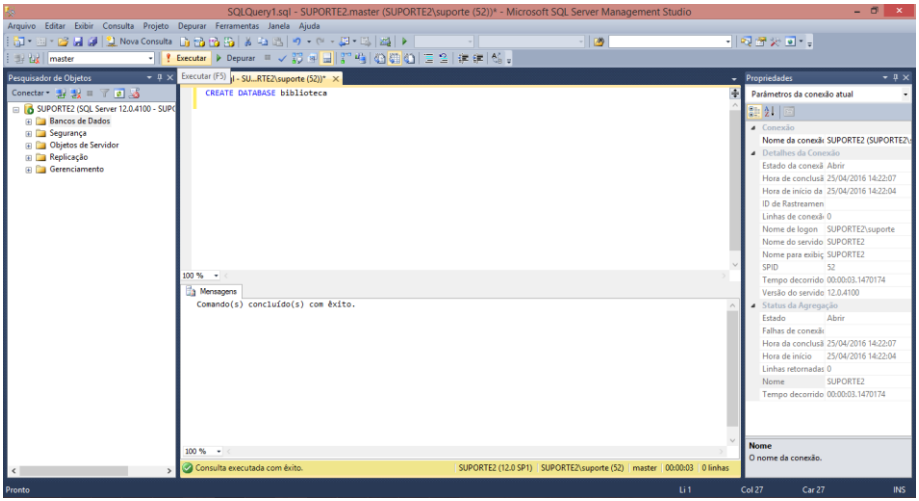


Figure 43 - SqlServer Create Database

Para verificar se o banco de dados foi devidamente criado você deve ir ao menu lateral esquerdo e clicar no ícone de atualizar (F5), como apresentado a seguir:

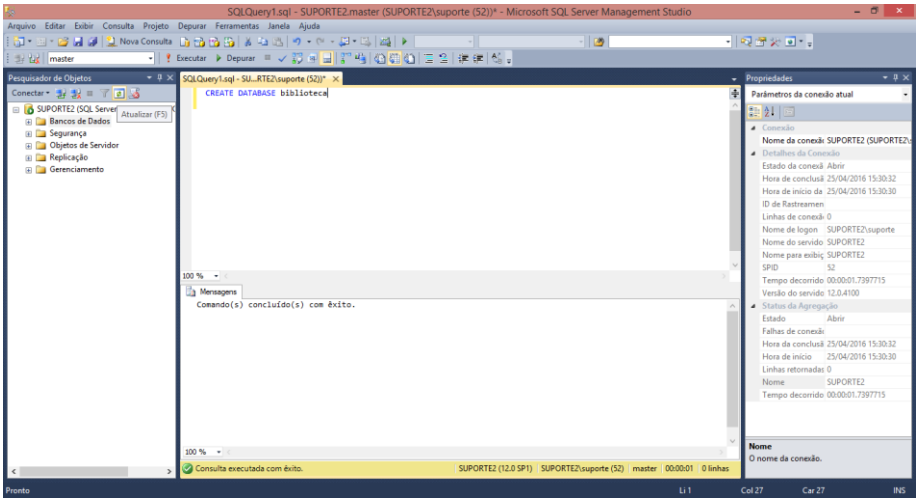


Figure 44 - SqlServer Create Database

Em seguida expanda o menu lateral através do ícone + localizado ao lado de Bancos de Dados.

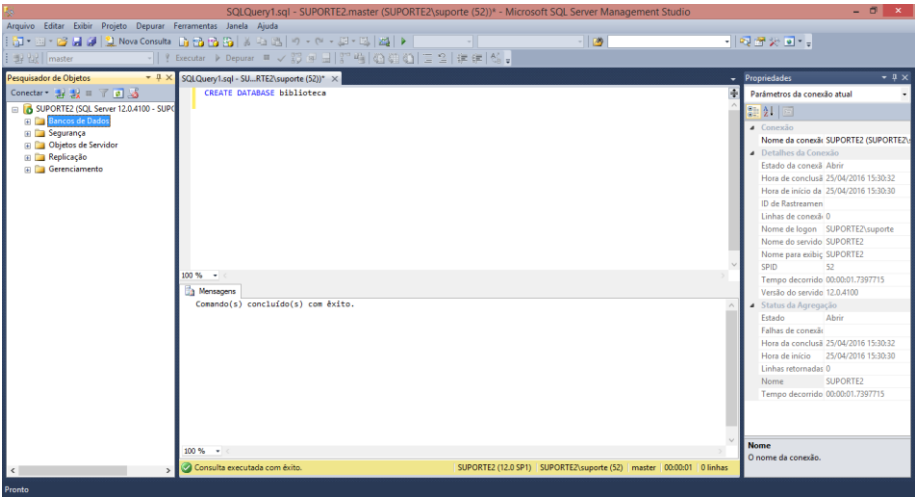


Figure 45 - SqlServer Create Database

Se tudo estiver ocorrido corretamente, você poderá localizar o banco criado no menu lateral esquerdo no menu Bancos de Dados.

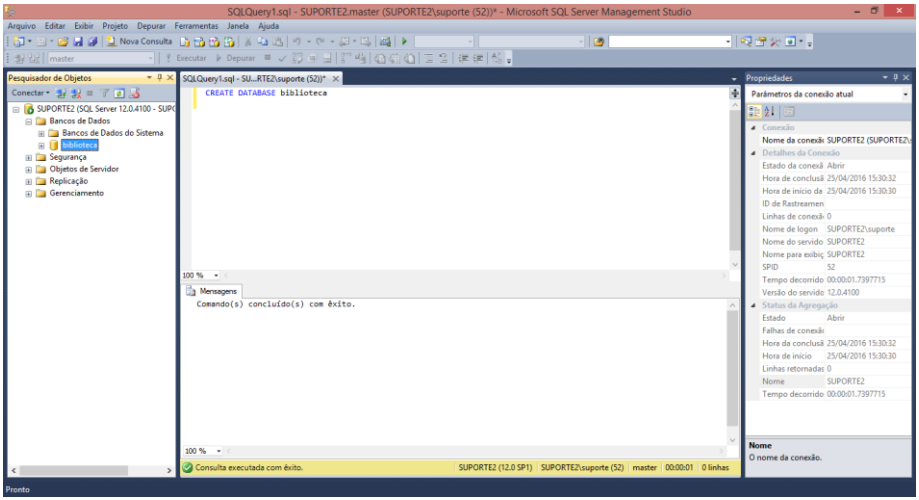


Figure 46 - SqlServer Create Database

USE DATABASE

Para que se possa utilizar o banco de dados que foi criado, é necessário o uso do comando USE, pois com ele o BD poderá ser acessado e utilizado.

Sintaxe USE DATABASE

USE DATABASE nome_do_banco

No caso, como queremos utilizar o Banco de Dados que acabamos de criar (biblioteca) devemos utilizar USE biblioteca, como é demonstrado na figura abaixo

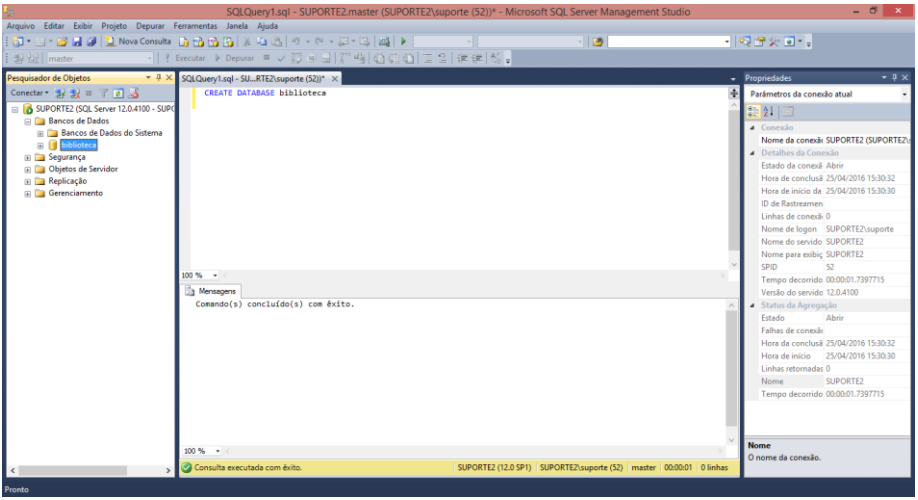


Figure 47 - SqlServer Use Database

Se estiver tudo correto o nome que aparecerá na barra superior será a do banco de dados escolhido, conforme demonstrado na figura a seguir:

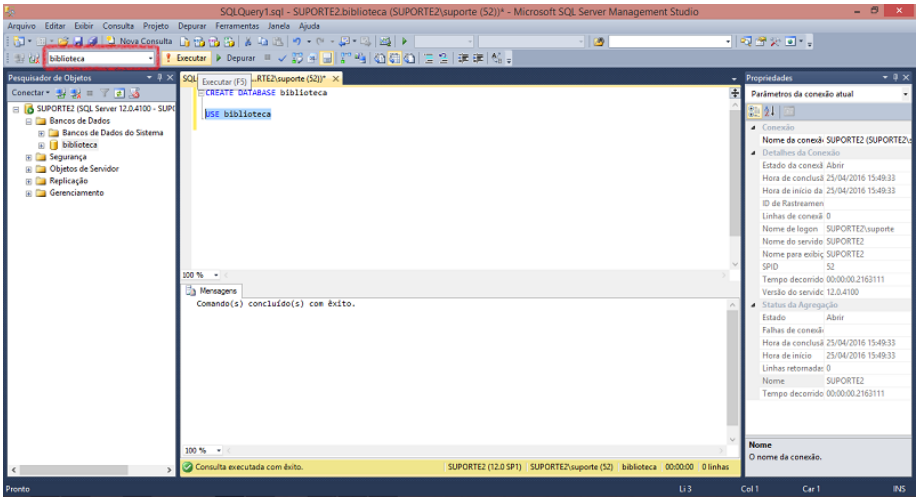


Figure 48 - SqlServer Use Database

DROP DATABASE

Para apagar um banco de dados, é preciso que se selecione outro banco na aba superior para então executar o comando de DROP DATABASE. Caso contrário, o programa irá emitir uma mensagem de erro, informando que o BD está em uso. Isso é apresentado na imagem a seguir:

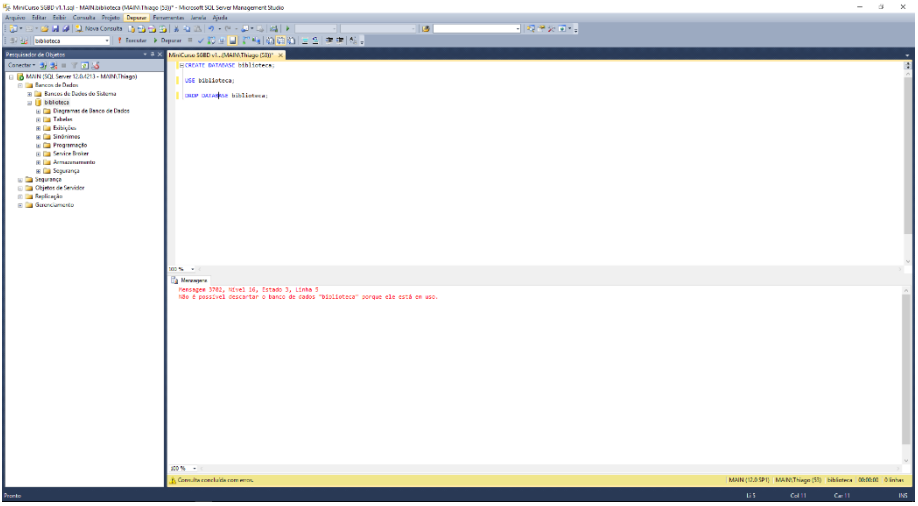


Figure 49 - SqlServer Drop Database

Para que esse erro não ocorra utilize o comando USE DATABASE master, onde master é o banco de dados padrão do sistema. Ou selecione o banco na barra caso esteja correto o nome do banco aparecerá na barra superior como demonstrado anteriormente.

Sintaxe DROP DATABASE

DROP DATABASE nome_do_banco

Exemplo:

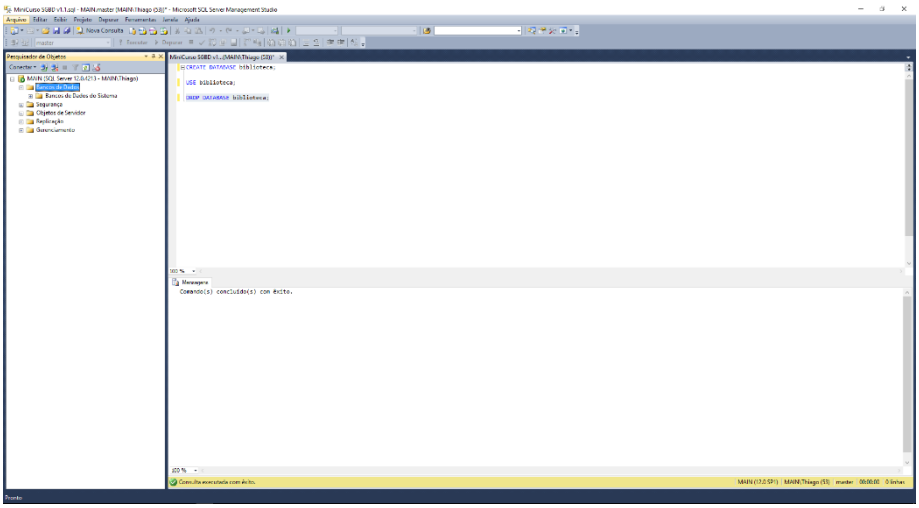


Figure 50 - SqlServer Drop Database

CREATE TABLE

O comando CREATE TABLE é usado para criar uma nova tabela dentro do banco de dados.

Sintaxe CREATE TABLE

```
CREATE TABLE nomeTabela
 (
        nome_coluna1 tipo_dado(tamanho),
        nome_coluna2 tipo_dado(tamanho),
        nome_coluna3 tipo_dado(tamanho),
        ....
 );
```

nomeTabela: É o nome que identifica a tabela dentro do BD.

nome_coluna1: É o nome da primeira coluna da tabela dentro do banco de dados, sendo seu identificador.

tipo_dado(tamanho): É o tipo de dados que vai ser inserido naquela coluna, a seguir serão apresentados os dados mais comuns e que serão utilizados nesse minicurso.

Alguns dados requerem que seja instanciado o tamanho máximo na hora da declaração do tipo de dado, como CHAR (n), VARCHAR (n) e etc.

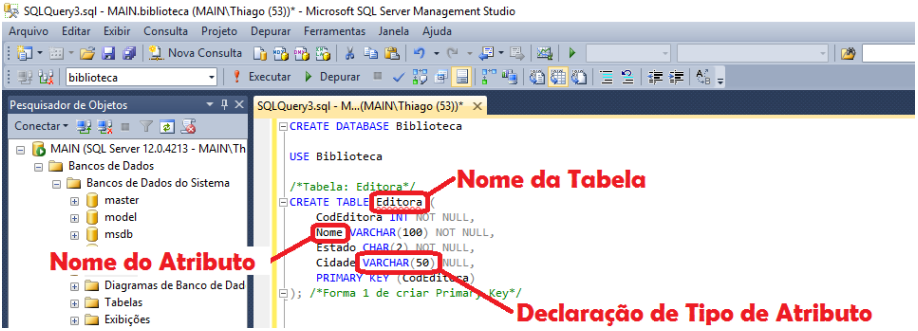


Figure 51 - SqlServer Create Table

Tipos Comuns de Dados

- **CHAR (n)**: cadeia de caracteres com exatamente n caracteres;
- **BIT (n):** cadeia com exatamente n bits;
- **VARBIT (n):** cadeia com até n bits;
- **INT**: inteiro com sinal (precisão definida pela implementação, usualmente 32 bits);
- **SMALLINT**: inteiro com sinal (precisão definida pela implementação);
- **NUMERIC (p, q)**: número decimal com 'p' dígitos mais sinal e pondo decimal implícito com 'q' dígitos a partir da direita;
- **DECIMAL (p, q)**: semelhante a NUMERIC, mas a precisão pode ser maior que p dígitos;
- **FLOAT**: número em ponto flutuante (precisão definida pela implementação);
- **VARCHAR (n)**: Aceita valores do tipo texto. É necessário especificar o tamanho desejado onde n é o tamanho do campo.

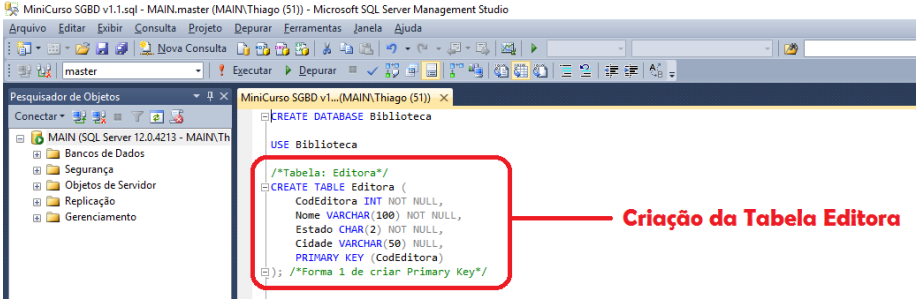


Figure 52 - SqlServer Create Table

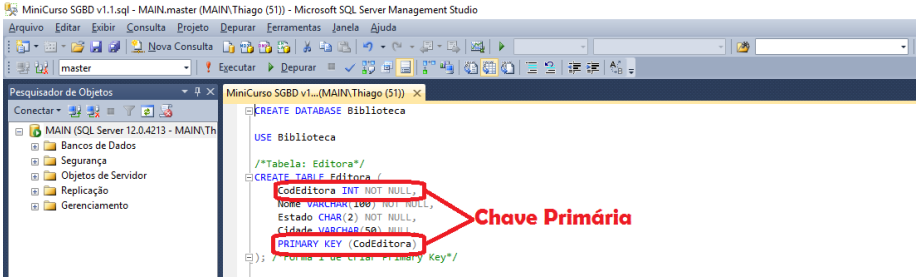


Figure 53 - SqlServer Create Table

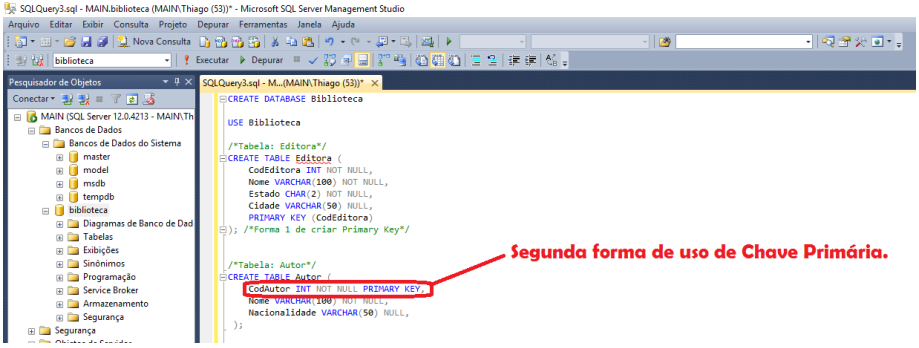


Figure 54 - SqlServer Create Table

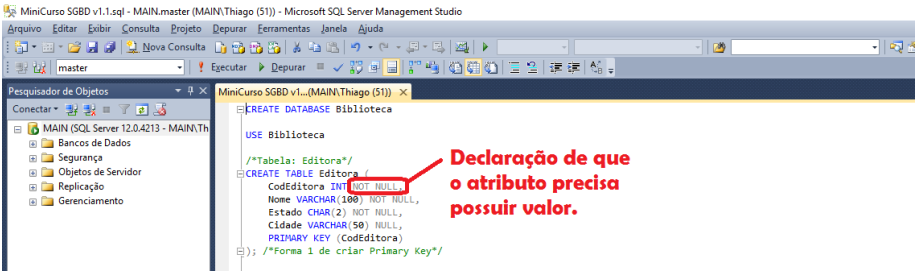


Figure 55 - SqlServer Create Table

ALTER TABLE

O comando ALTER TABLE é usado para adicionar, deletar ou modificar colunas em uma tabela existente. Você também poderá utilizar o comando ALTER TABLE para adicionar e deletar várias restrições em uma tabela.

Sintaxe ALTER TABLE ADD

ALTER TABLE nome_tabela

ADD nome_coluna tipodados

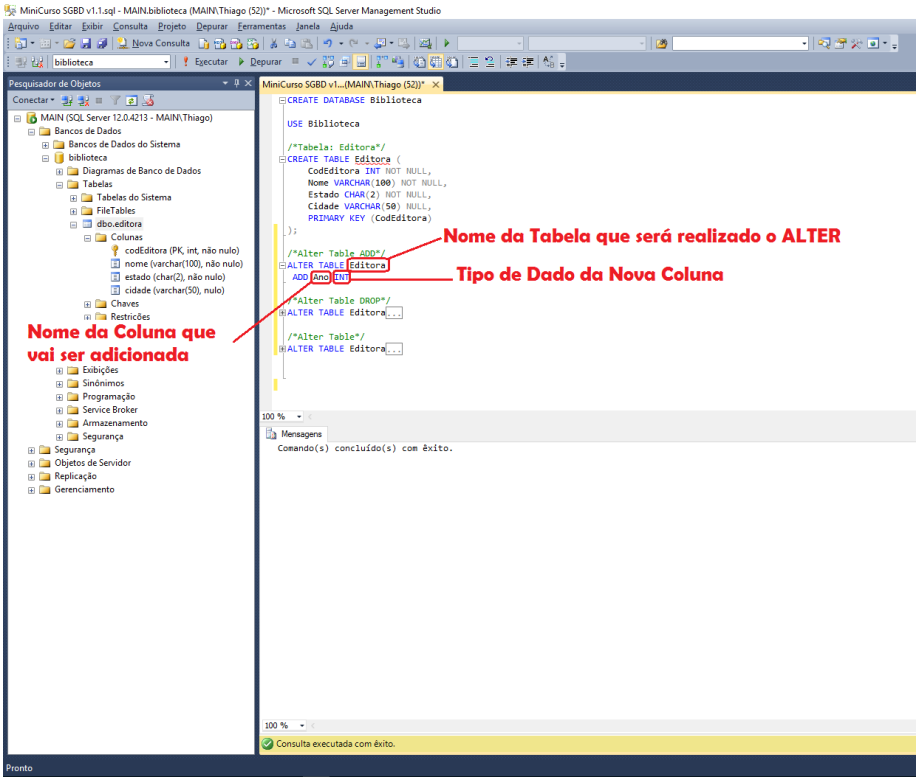


Figure 56 - SqlServer Alter Table

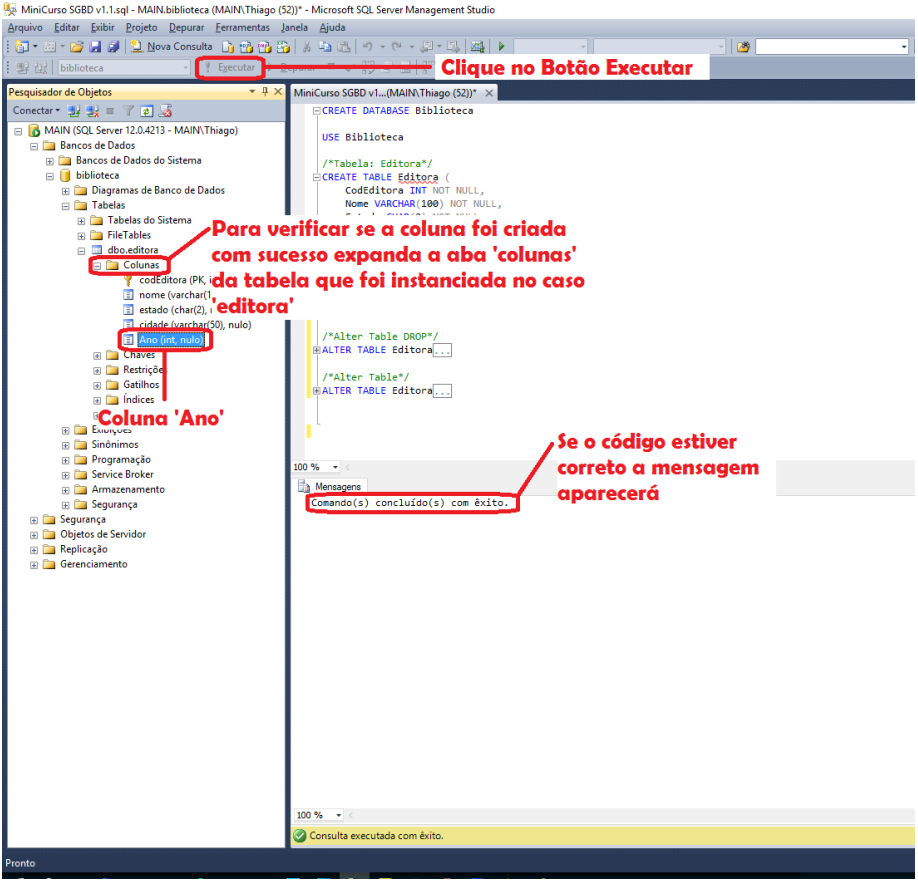


Figure 57 - SqlServer Alter Table

Para deletar uma coluna em uma tabela, use a seguinte sintaxe:

Sintaxe ALTER TABLE DROP COLUMN

```
ALTER TABLE nome_tabela
  DROP COLUMN nome_coluna
```

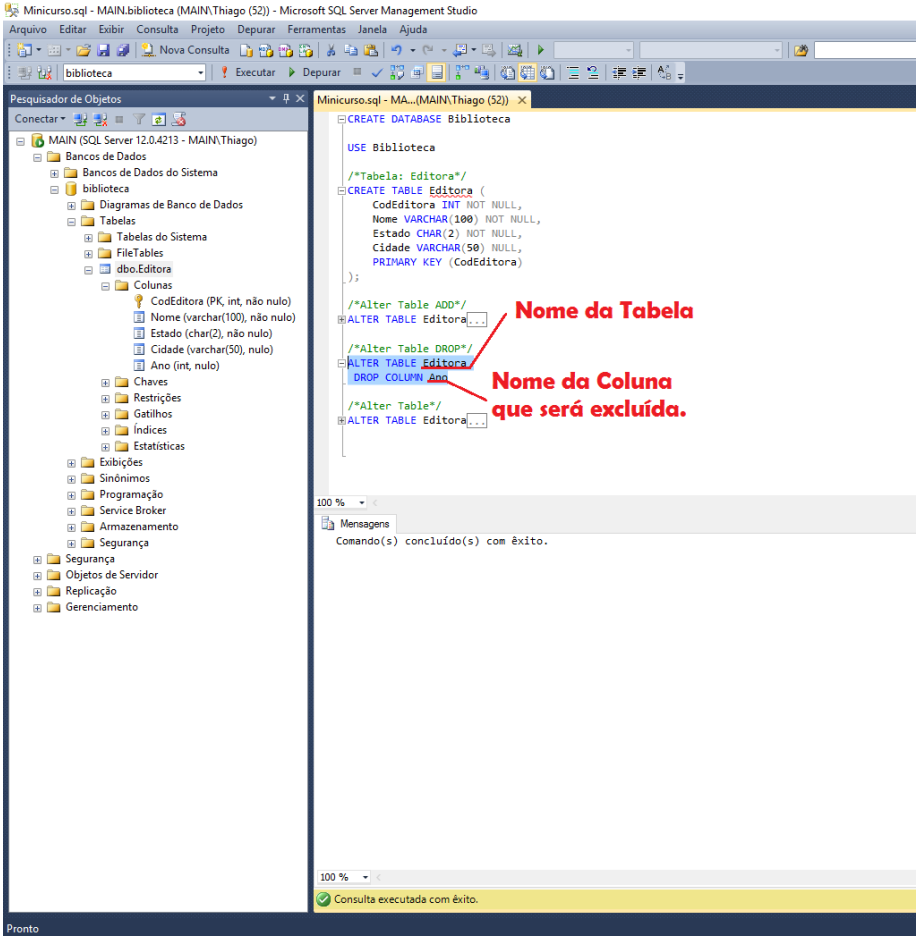


Figure 58 - SqlServer Alter Table

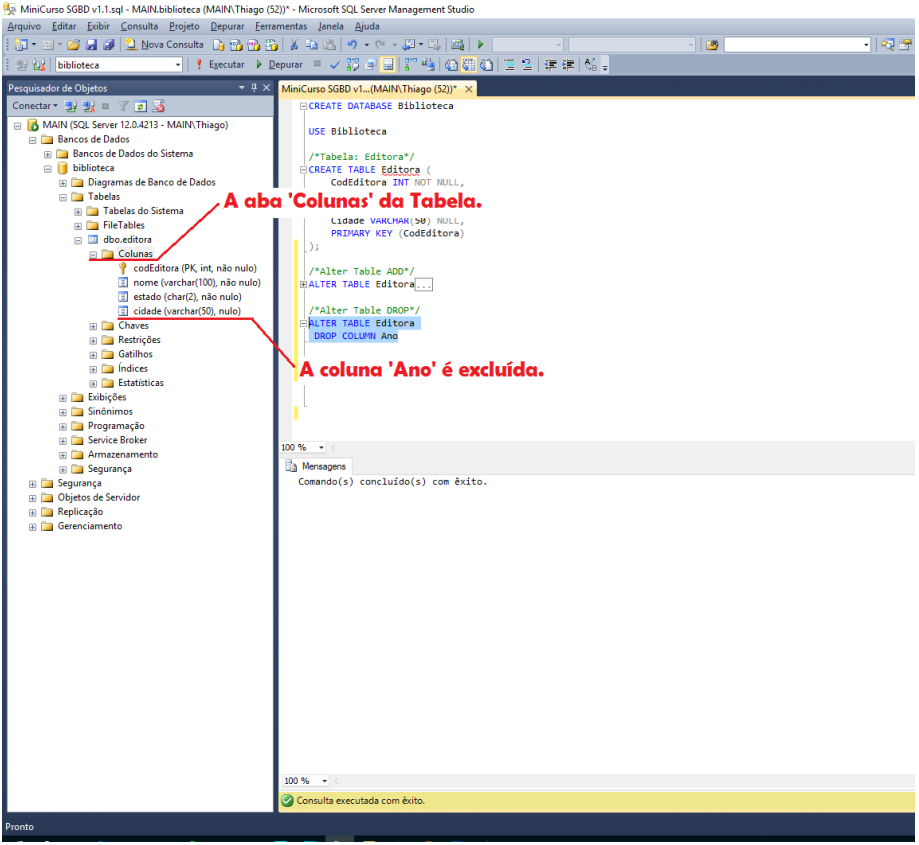


Figure 59 - SqlServer Alter Table

Para alterar uma coluna em uma tabela, use a seguinte sintaxe:

Sintaxe ALTER TABLE ALTER COLUMN

ALTER TABLE nome_tabela

ALTER COLUMN nome_coluna tipodados

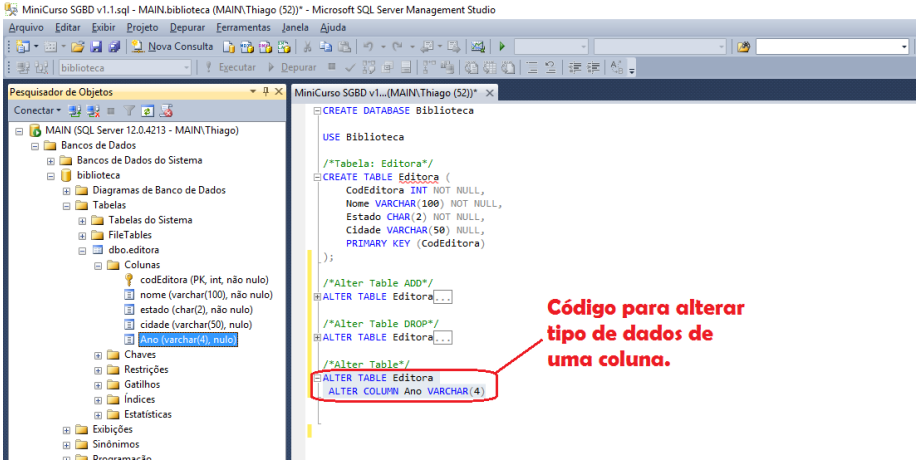


Figure 60 - SqlServer Alter Table

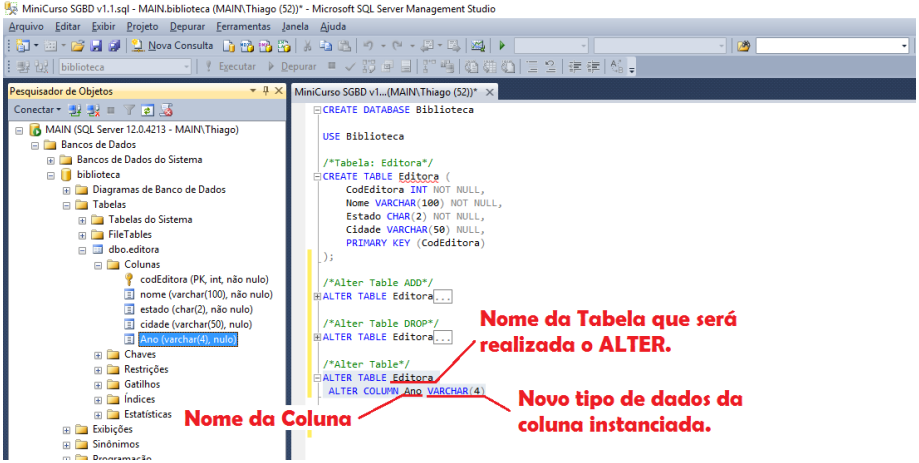


Figure 61 - SqlServer Alter Table

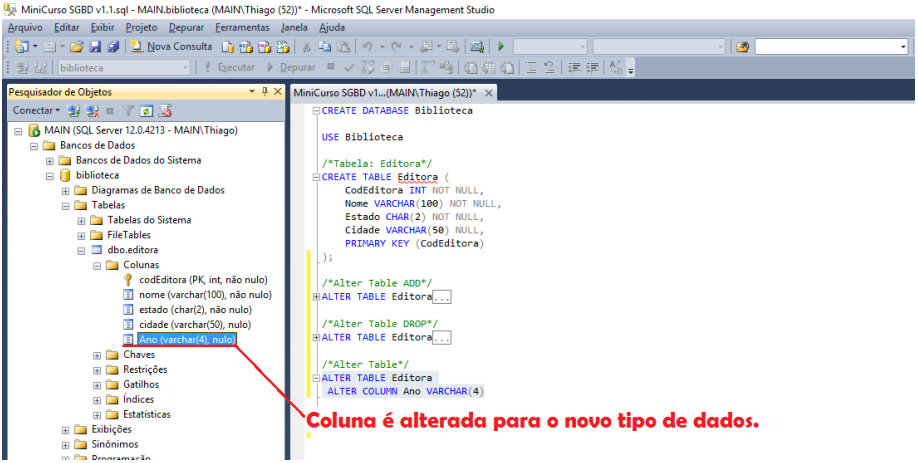


Figure 62 - SqlServer Alter Table

DROP TABLE

O comando DROP TABLE é utilizado para excluir uma tabela e todos os seus dados, índices, triggers, restrições e permissões específicas do banco de dados.

Sintaxe DROP TABLE

DROP TABLE nomeTabela

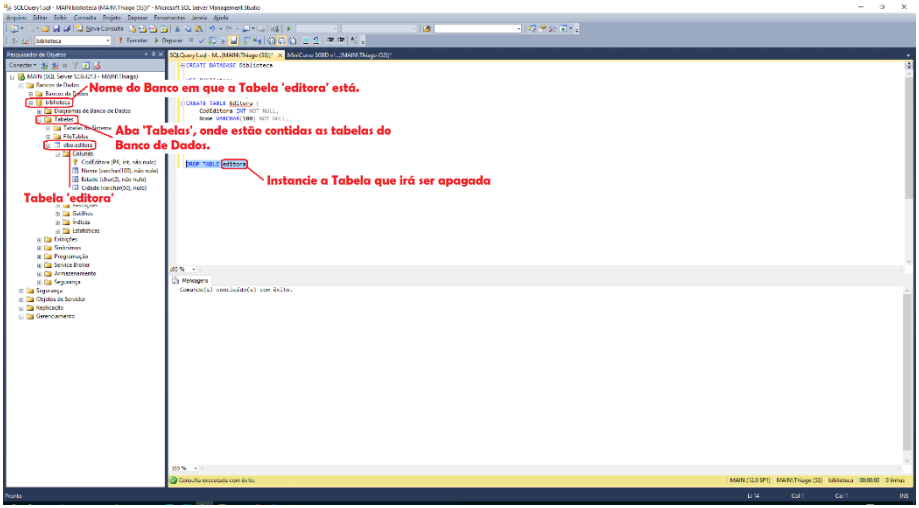


Figure 63 - SqlServer Drop Table

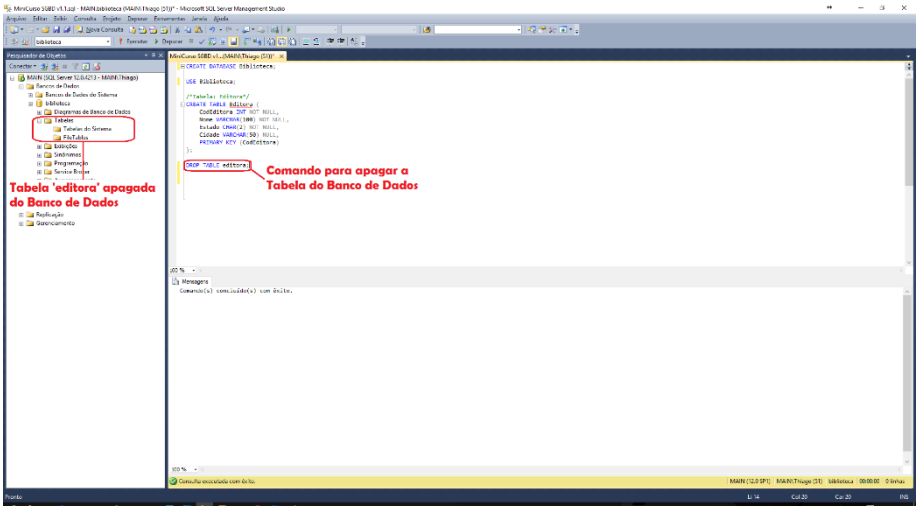


Figure 64 - SqlServer Drop Table

INSERT

A PRIMEIRA FORMA não especifica os nomes das colunas em que os dados serão inseridos, apenas seus valores:

Sintaxe INSERT INTO

INSERT INTO nome_tabela VALUES (valor1, valor2, valor3, ...);

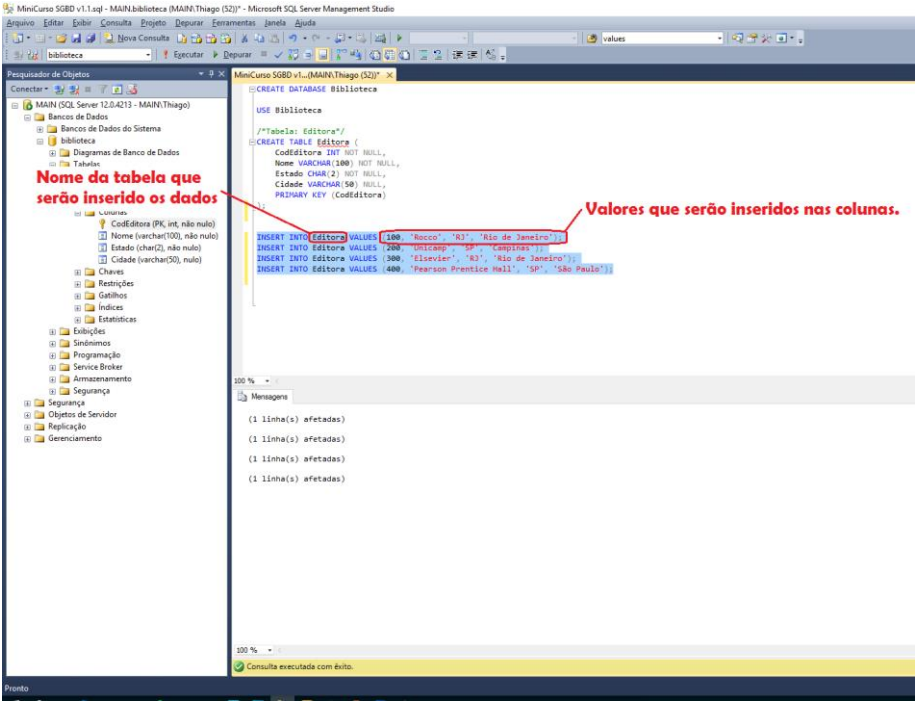


Figure 65 - SqlServer Insert

A SEGUNDA FORMA específica tanto o nome da coluna quanto o valor a ser inserido:

Sintaxe INSERT INTO

INSERT INTO nome_tabela (coluna1,coluna2,coluna3,...) VALUES (valor1, valor2, valor3, ...);

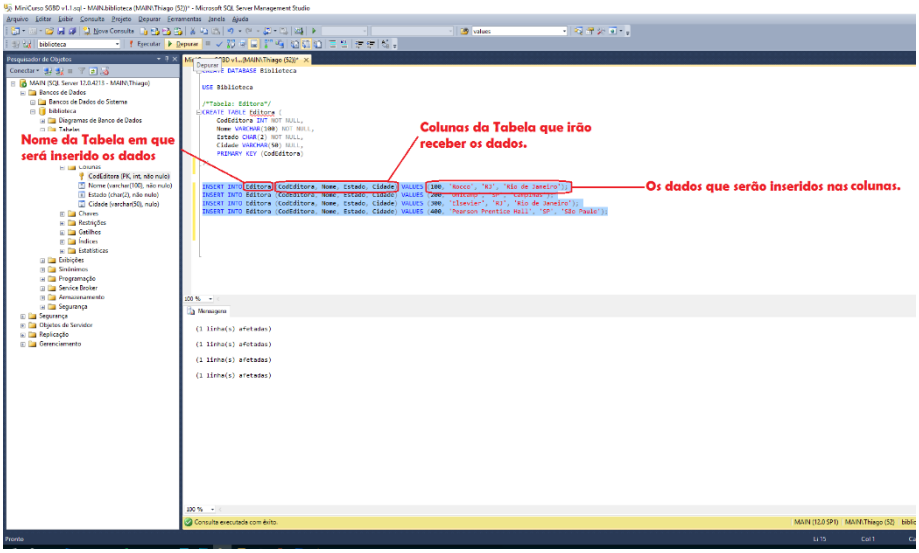


Figure 66 - SqlServer Insert

DELETE

Caso você deseja excluir dados de uma tabela o comando que se deve utilizar é DELETE, ele é sempre acompanhado de WHERE, que irá instanciar quais linhas da tabela que se deseja apagar, se WHERE não for inserido junto, todos os dados daquele tabela serão apagados.

Sintaxe DELETE

DELETE FROM nome_tabela WHERE nome_coluna=valor;

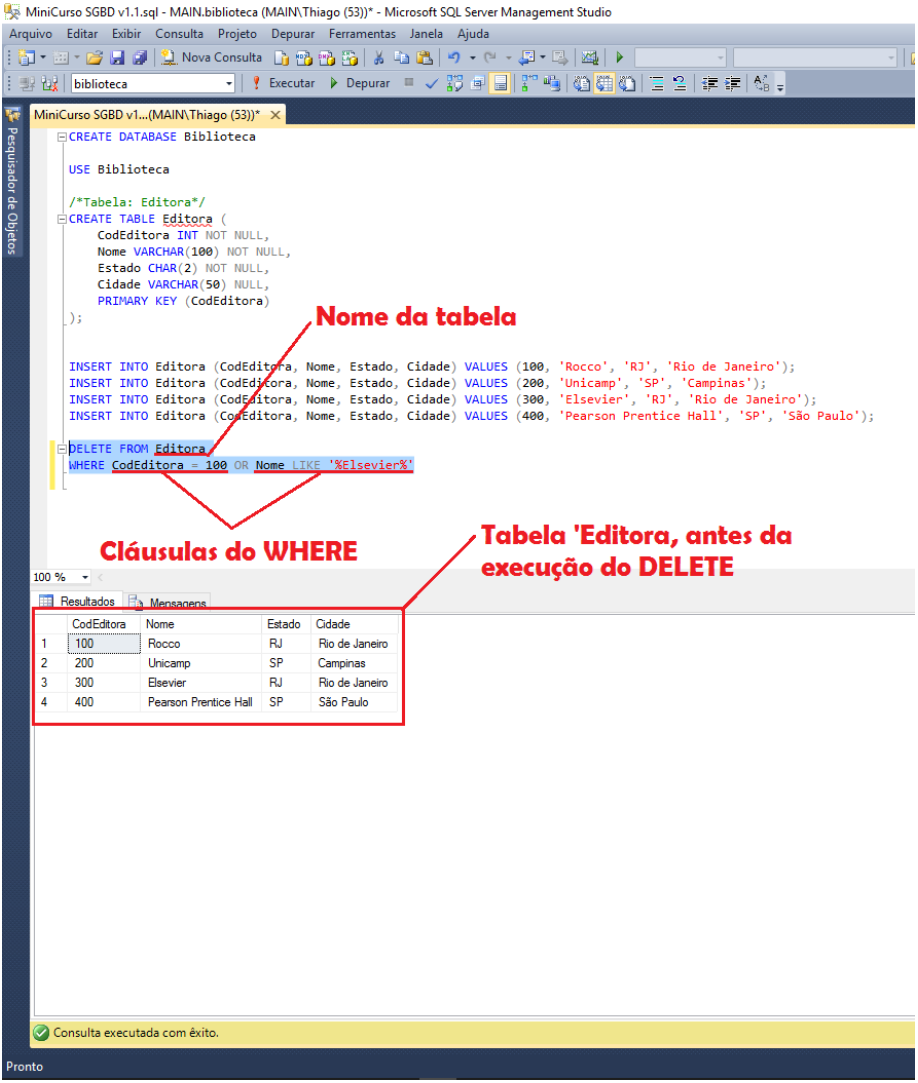


Figure 67 - SqlServer Delete

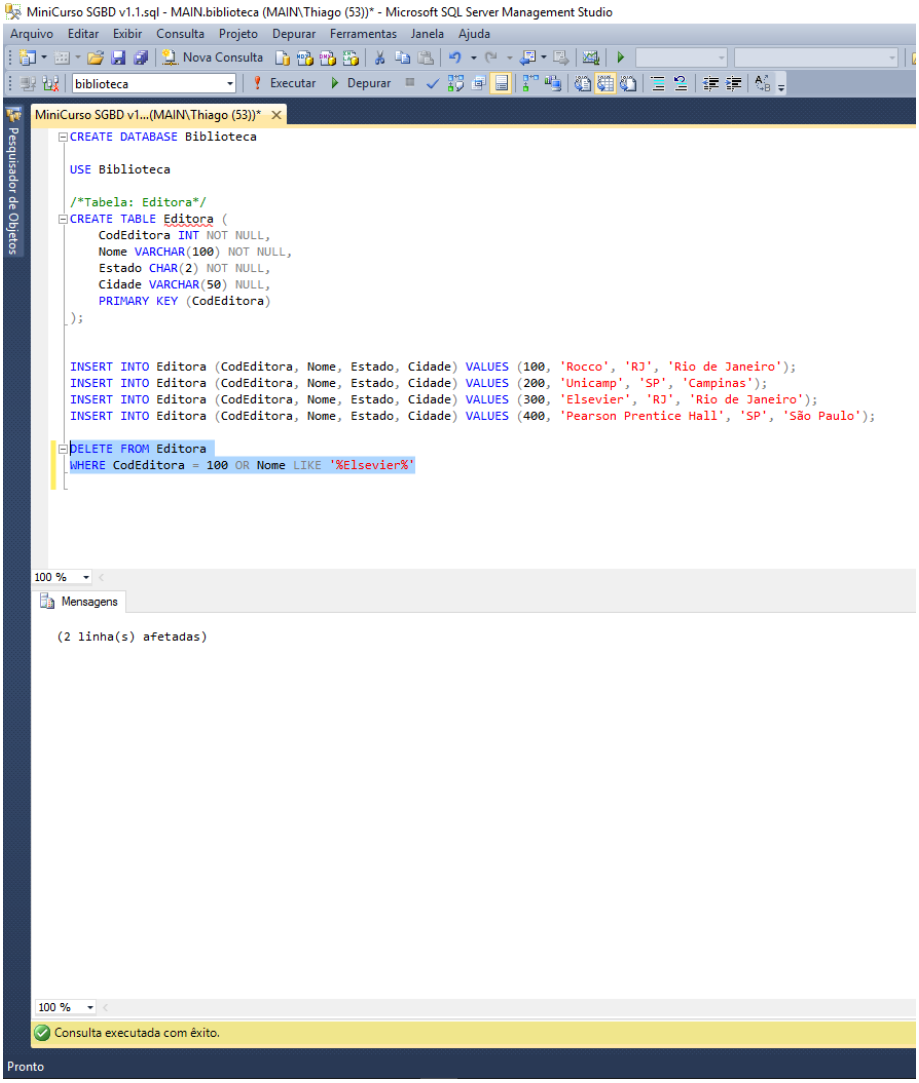


Figure 68 - SqlServer Delete

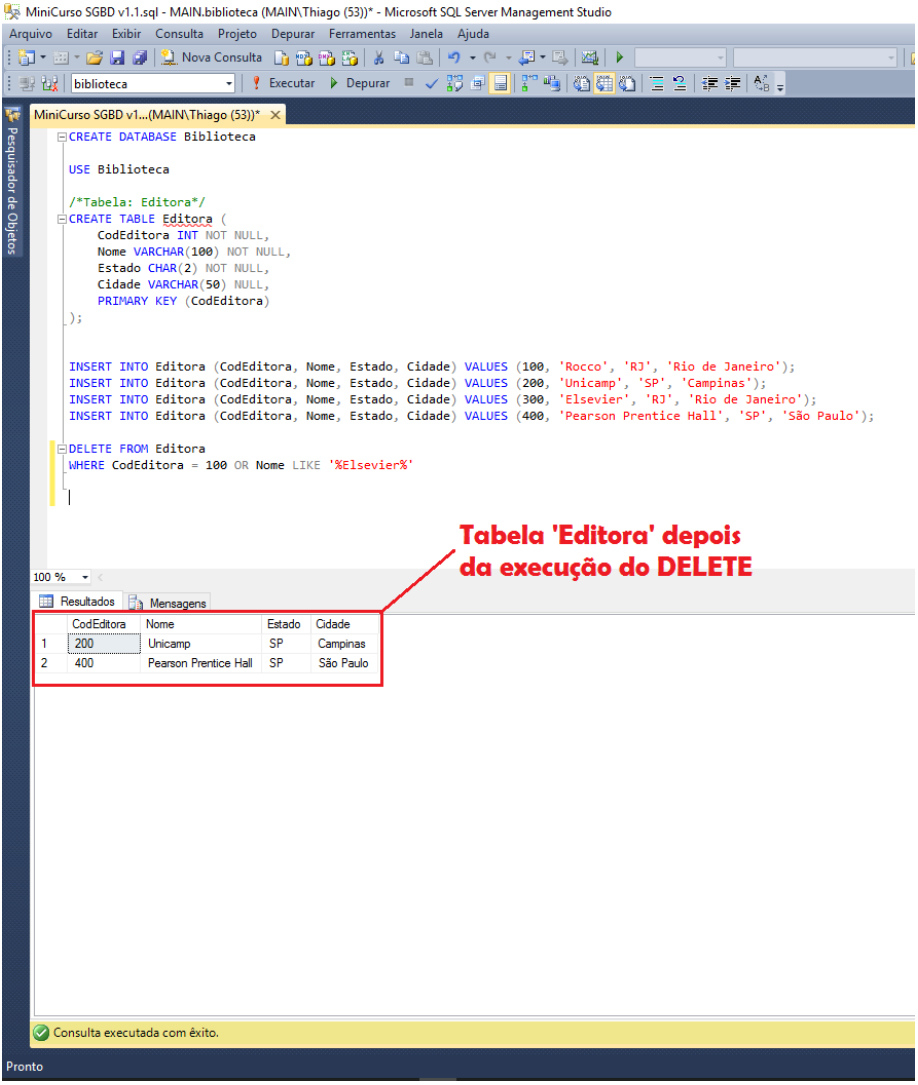


Figure 69 - SqlServer Delete

UPDATE

O comando UPDATE é utilizado para modificar dados existentes em uma tabela. Você também pode utilizar a cláusula WHERE juntamente com o UPDATE para selecionar quais linhas serão alteradas. Ou se não todas elas serão.

Sintaxe UPDATE

```
UPDATE nome_tabela
  SET coluna1=valor1,coluna2=valor2,...
  WHERE nome_coluna=valor;
```

- SET irá determinar o novo valor para os dados que irão ser substituídos.
- WHERE irá determinar quais as colunas que dados que serão atualizados.

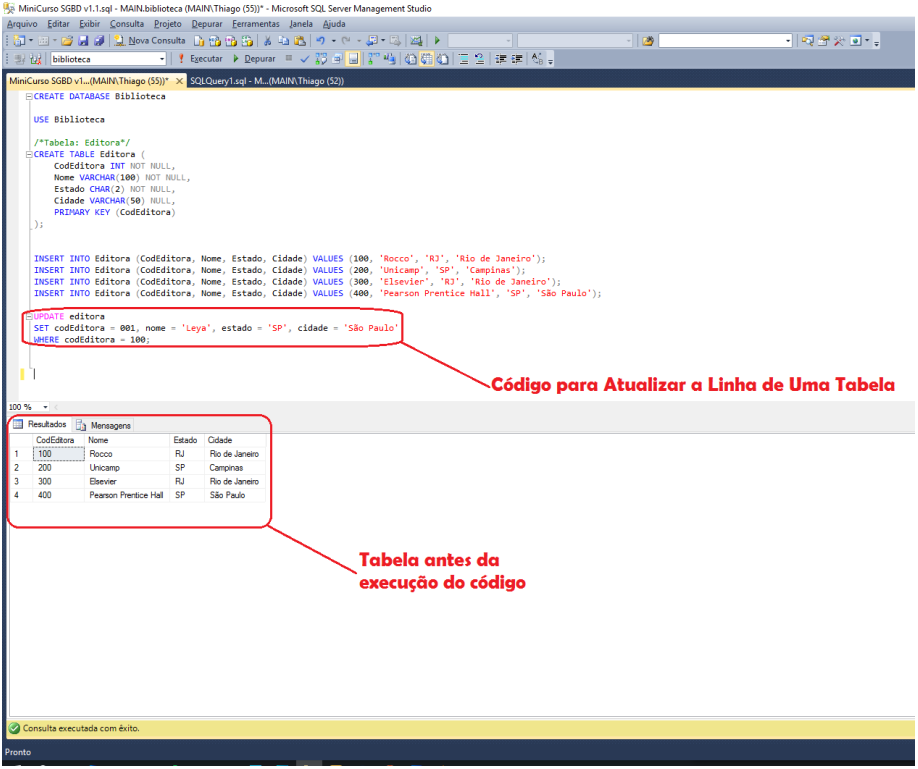


Figure 70 - SqlServer Update

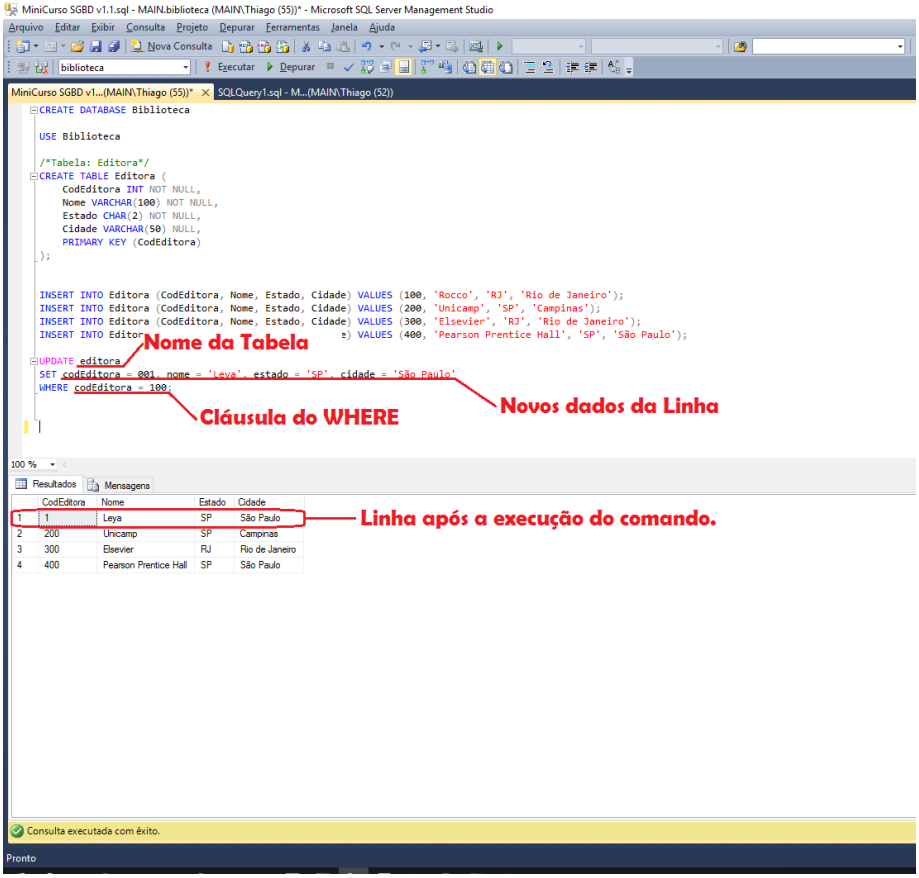


Figure 71 - SqlServer Update

TRANSACTION

Uma transação é uma unidade ou sequência de trabalhos realizados em uma ordem lógica, sendo realizado por um usuário ou automaticamente por algum tipo de programa de banco de dados. Transação é a propagação de uma ou mais mudanças no banco de dados. Por exemplo, se você está criando um registro, atualizando ou deletando um registro de uma tabela, então você está realizando uma transação em uma tabela. É importante controlar transações para garantir a integridade dos dados e lidar com erros no banco de dados.

O comando BEGIN TRANSACTION inicia as transações, qualquer tipo de transação feita e necessário o comando iniciando.

Sintaxe BEGIN TRANSACTION

```
BEGIN TRANSACTION
        commandos
        .....
        COMMIT ou ROLLBACK
```

Onde:

- **Begin Transaction:** Tag inicial para o início de uma transação.
- **Corpo de comando:** Conjunto de comando a serem executados dentro de uma transação.
- **COMMIT** ou ROLLBACK: Comandos que finalizam a transação onde o 'COMMIT' confirma o conjunto de comandos e o 'ROLLBACK' desfaz todo o processo executado pelo corpo de comandos caso tenha ocorrido algum evento contrário ao desejado.

No SQL SERVER temos uma função de sistema que faz a identificação de um erro dentro de uma transação chamada de '@@ERROR' função essa que por padrão recebe o valor 0 (zero) caso não ocorra nem um erro, no caso de algum erro ela assume o valor 1 (um).

Exemplo:

```
UPDATE FROM TbContas
      SET NuSaldo= 10.000
 WHERE NuSaldo IF @@ERROR = 0
      COMMIT
ELSE
      ROLLBACK
END
```

Exemplo do início de uma transação para a inserção de valores na tabela 'Banco' e em seguida o uso do comando SELECT para apresentar os registros inseridos.

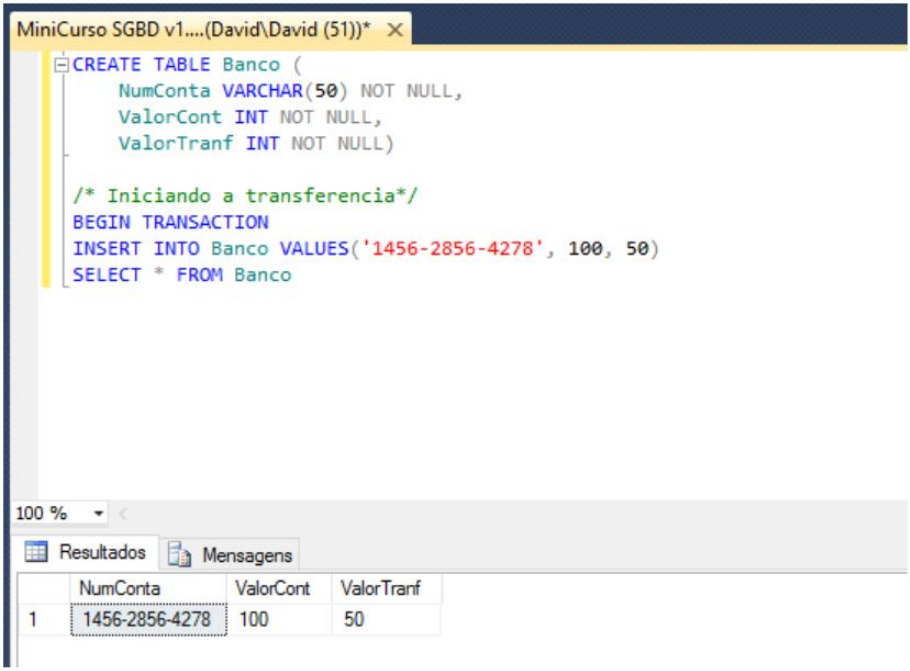


Figure 72 - SqlServer Transaction

COMMIT

O COMMIT serve para fazer a confirmação da transação (registro), feito dos dados inserido na transação.

Exemplo da aplicação do bloco de comandos do TRANSACTION, caso os comandos tenham sido executados conforme esperado se executa o comando COMMIT que aplica o bloco de comando executado.

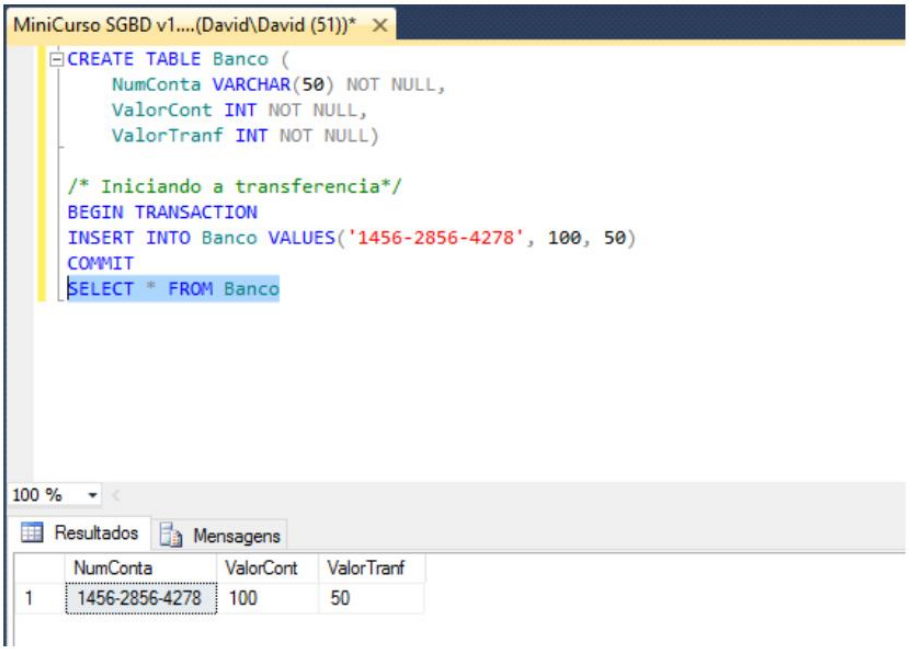


Figure 73 - SqlSerever Transaction

ROLLBACK

Junto de comando BEGIN TRANSACTION podemos usar o comando ROLLBACK para desfazer a transação (registro) feito mostrado na imagem anterior como iremos apresentar agora. Usando o ROLLBACK nos cancelamos a transação feita e usando o SELECT na tabela mostramos que todo registro foi desfeito.

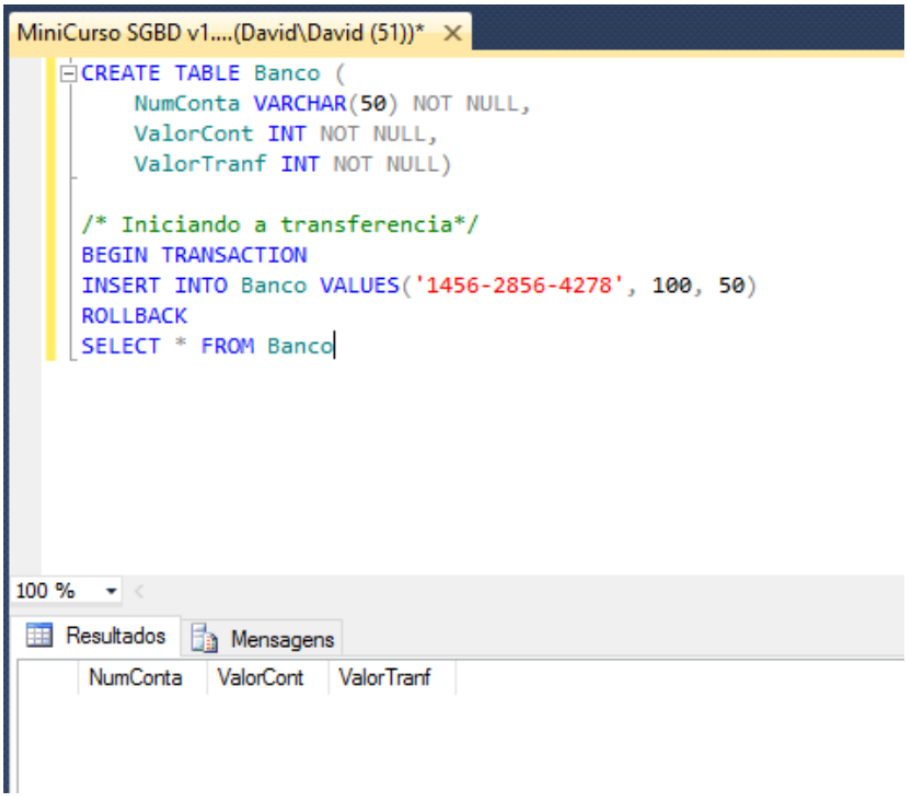


Figure 74 - SqlServer Transaction

SELECT

Usando em conjunto cláusulas você pode especificar (refinar) dentro da declaração original do SELECT como vou apresentar agora. Para fazer uma pesquisa simples usando o SELECT vamos usar esta sintaxe:

SELECT * FROM Editora

Esta sintaxe seleciona todas as colunas de uma tabela o "*" significa all por isto ele seleciona todas as colunas, no caso a tabela é chamada de "Editora".

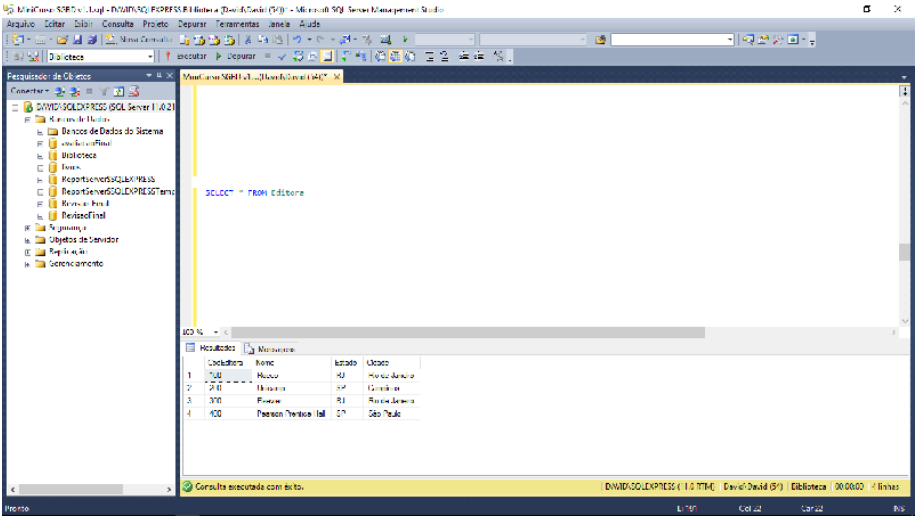


Figure 75 - SqlServer Select

Primeiramente selecione toda a sintaxe do SELECT e suas declarações.

Depois clique no ícone na parte superior da tela, lembrando que você tem quer estar dentro do banco de dados para funcionar, ou apertando a tecla F5.

Caso o comando não possua erros irá aparecer uma tabela como na Figura acima, com todas as colunas com dados que já haviam sido inseridos na tabela.

Especificando os dados na hora de executar o SELECT você irá obter pesquisas mais específicas, como ocorre nesta sintaxe:

SELECT Nome FROM Editora

Especificando a pesquisa de dados, no caso aqui ele só quer os dados da coluna “Nome” da tabela “Editora”.

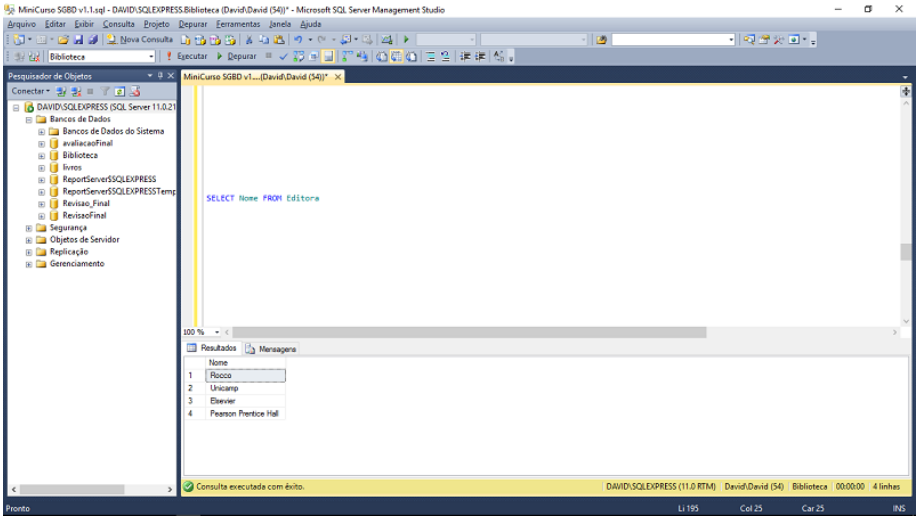


Figure 76 - SqlServer Select

Primeiramente selecione toda a sintaxe do SELECT e suas declarações.

Depois clique no ícone na parte superior da tela, lembrando que você tem quer estar dentro do banco de dados para funcionar, ou apertando a tecla F5.

Caso o comando não possua erros irá aparecer uma tabela como na Figura anterior, a coluna “Nome” com dados que já haviam sido inseridos na tabela.

Agora iremos usar a cláusula WHERE, ao usar esta cláusula você indica as condições que os registros têm que satisfazer para serem selecionados, usando WHERE em conjunto com o SELECT, iremos fazer ecrever esta sintaxe:

SELECT * FROM Usuario WHERE MatricUsuario=3

Nesta sintaxe ele está selecionando todas as colunas, mas somente com o valor especificado na condição WHERE.

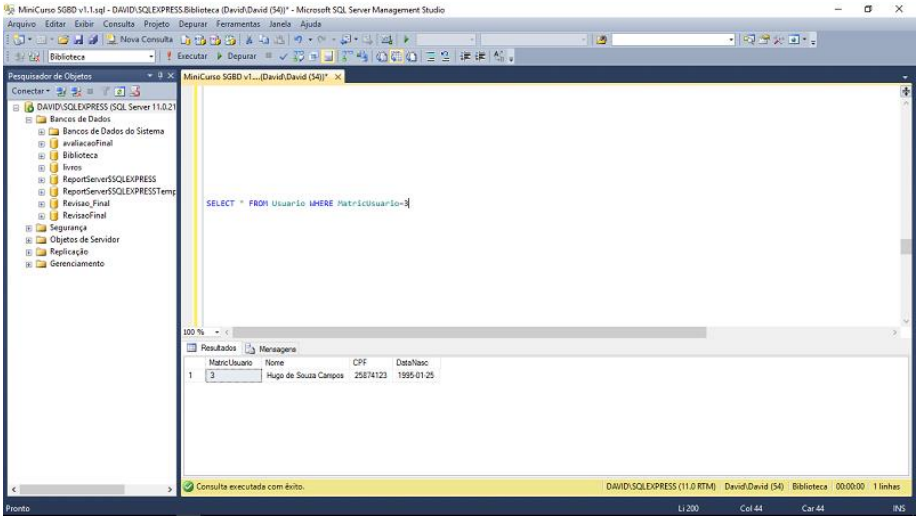


Figure 77 - SqlServer Select

Primeiramente selecione toda a sintaxe do SELECT e suas declarações.

Depois clique no ícone na parte superior da tela, lembrando que você tem quer estar dentro do banco de dados para funcionar, ou apertando a tecla F5.

Caso o comando não possua erros irá aparecer uma tabela como na Figura anterior, aparecera só os dados onde “MatricUsuario” tem valor 3.

Sintaxes com outros exemplos de WHERE:

SELECT * FROM Usuario WHERE MatricUsuario in (2,5)

Seleciona todas as colunas da tabela, mas somente com os valores especificados no IN.

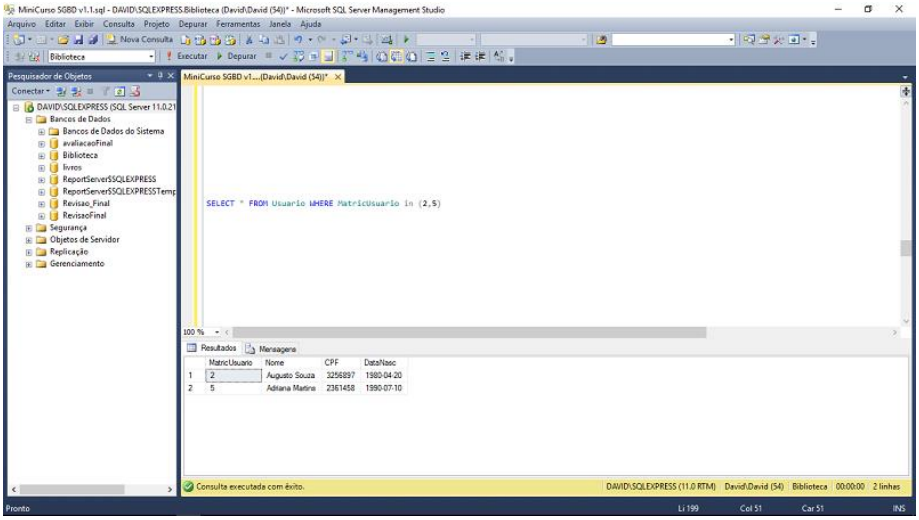


Figure 78 - SqlServer Select

SELECT * FROM Usuario WHERE MatricUsuario in (2,5)

Seleciona todas as colunas da tabela, mas somente com os valores especificados no intervalo do BETWEEN.

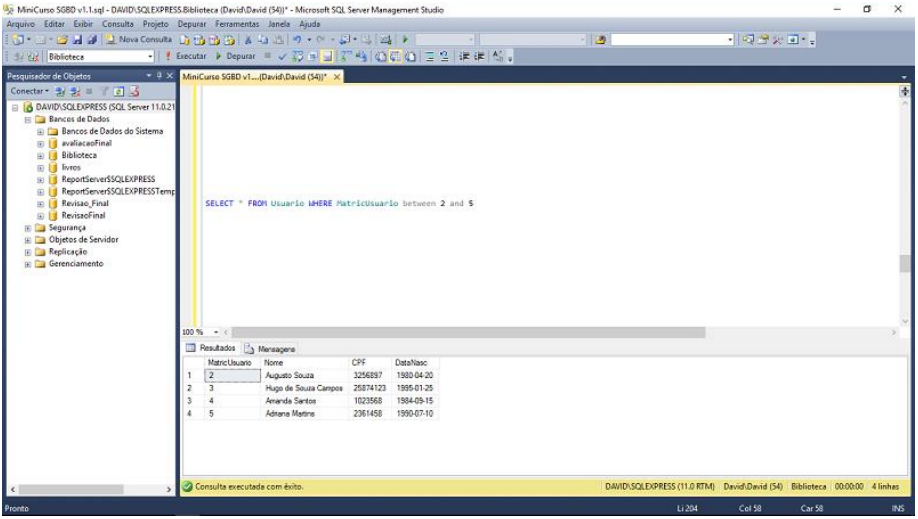


Figure 79 - SqlServer Select

SELECT * FROM Usuario u, aluno a WHERE a.FK_MatricUsuario = u.MatricUsuario

Junta as tabelas onde a chave estrangeira é igual à chave primária.

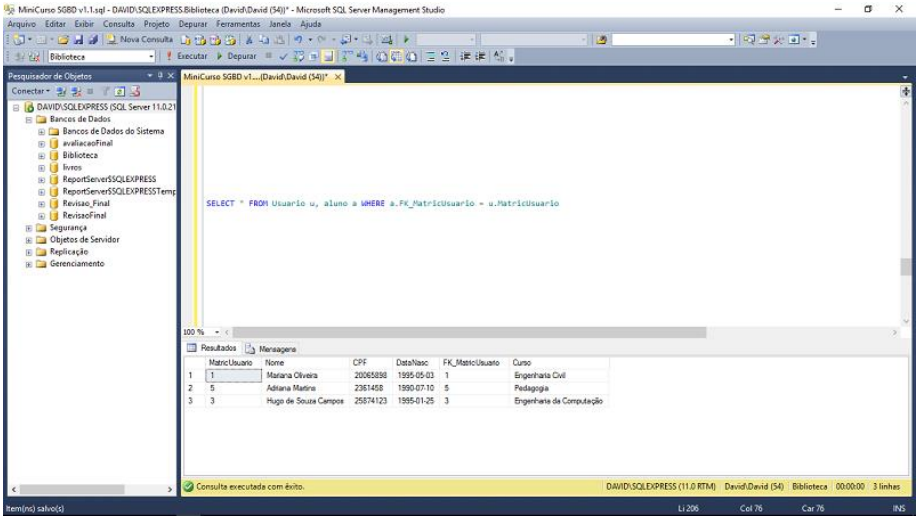


Figure 80 - SqlServer Select

Agora iremos usar em conjunto com o SELECT e o WHERE a cláusula GROUP BY, com esta cláusula ele irá agrupar de acordo com o que for definido nele, um exemplo seria este:

```
SELECT a.Nome FROM Autor a, Livro l, Autor_Livro al
WHERE al.FK_CodAutor = a.CodAutor and al.FK_CodLivro = l.CodLivro
GROUP BY a.Nome
```

Ele irá selecionar a coluna “Nome” da tabela Autor, pegar as tabelas Autor, Livro e Autor_Livro, juntar as tabelas e agrupar os dados pela coluna “Nome” da tabela Autor.

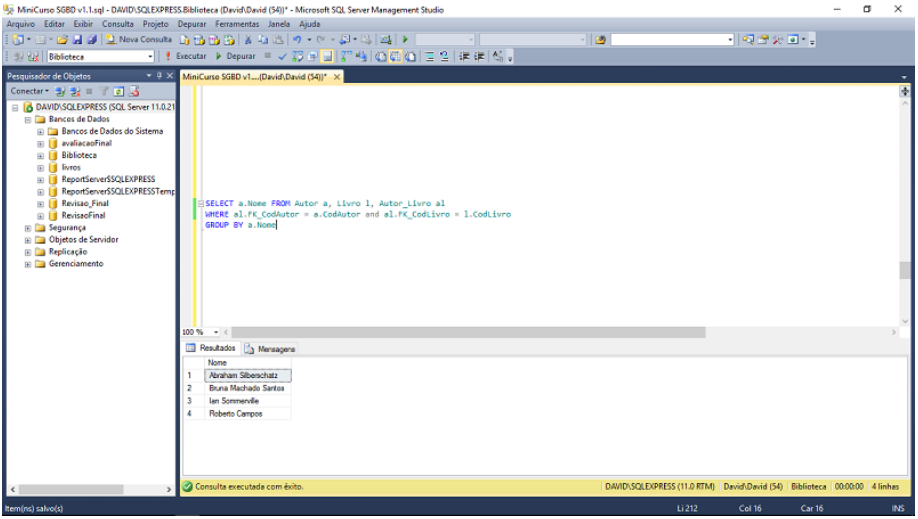


Figure 81 - SqlServer Select

Passo a passo de como fazer as pesquisas:

Primeiramente selecione toda a sintaxe do SELECT e suas declarações.

Depois clique no ícone na parte superior da tela, lembrando que você tem quer estar dentro do banco de dados para funcionar, ou apertando a tecla F5.

Caso o comando não possua erros irá aparecer uma tabela como na Figura anterior, aparecera só os dados “Nome” em ordem alfabética.

Agora iremos usar em conjunto com o SELECT e o WHERE a cláusula ORDER BY, com esta cláusula ele irá ordenar os dados de acordo com o que for definido nele, um exemplo seria este:

SELECT CodAutor, Nome, Nacionalidade

FROM Autor

ORDER BY Nome

Ele selecionaria as colunas “CodAutor”, “Nome”, “Nacionalidade” da tabela Autor e ordenaria os dados de acordo com a coluna “Nome”.

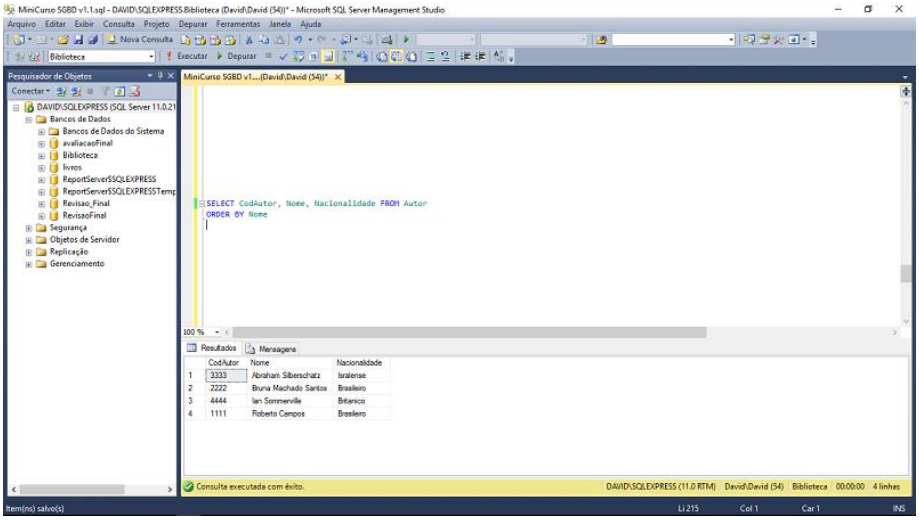


Figure 82 - SqlServer Select

Passo a passo de como fazer pesquisas:

Primeiramente selecione toda a sintaxe do SELECT e suas declarações.

Depois clique no ícone na parte superior da tela, lembrando que você tem quer estar dentro do banco de dados para funcionar, ou apertando a tecla F5.

Caso o comando não possua erros irá aparecer uma tabela como na Figura anterior, aparecera todos os dados selecionados, mas eles estarão em ordem de acordo com o "Nome" e não pela ordem registrada.

Com o comando SELECT você pode usar o SUM que irá somar todos os valores da coluna, sintaxe:

SELECT SUM(MatricUsuario) FROM Usuario

Realiza a soma dos valores inteiros da coluna MatricUsuario da tabela Usuario.

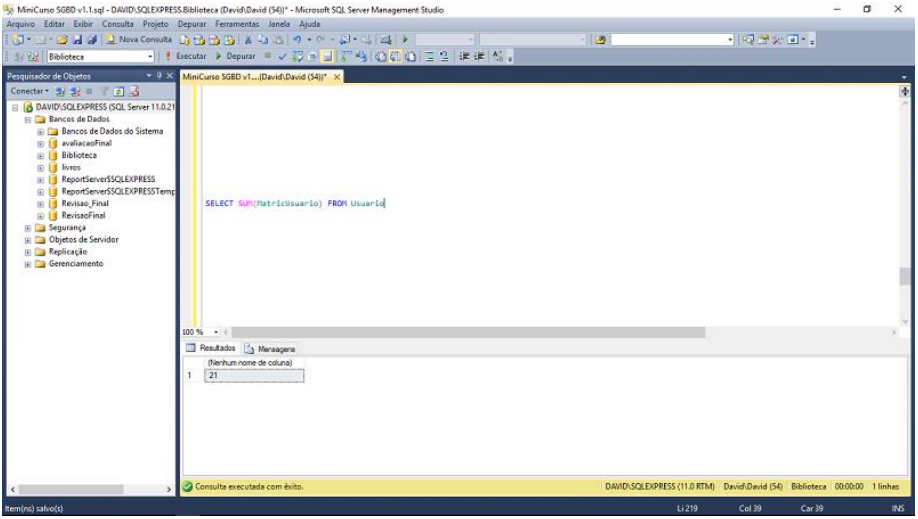


Figure 83 - SqlServer Select

Passo a passo de como fazer pesquisas:

Primeiramente selecione toda a sintaxe do SELECT e suas declarações.

Depois clique no ícone na parte superior da tela, lembrando que você tem quer estar dentro do banco de dados para funcionar, ou apertando a tecla F5.

Caso o comando não possua erros irá aparecer uma tabela como na Figura anterior, irá aparecer a soma de todos os números inteiros da coluna.

Com o comando SELECT você pode usar o COUNT para contar os registros de uma tabela, sintaxe:

SELECT COUNT(MatricUsuario) FROM Usuario

Ele irá contar o número de registro do “MatricUsuario” da tabela Usuario.

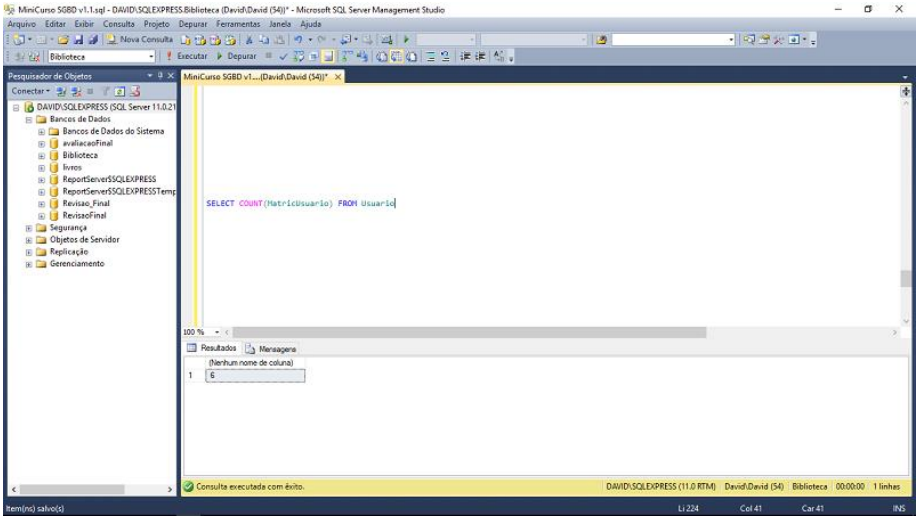


Figure 84 - SqlServer Select

Passo a passo de como fazer pesquisas:

Primeiramente selecione toda a sintaxe do SELECT e suas declarações.

Depois clique no ícone na parte superior da tela, lembrando que você tem quer estar dentro do banco de dados para funcionar, ou apertando a tecla F5.

Caso o comando não possua erros irá aparecer uma tabela como na Figura anterior, irá aparecer o número de registros da coluna “MatricUsuario”.

Com a clausula MAX ele irá selecionar o maior valor de uma determinada coluna, sintaxe:

SELECT MAX(CodEmprestimo) FROM Emprestimo

Ele irá selecionar a coluna “CodEmprestimo” e pegara o seu maior valor.

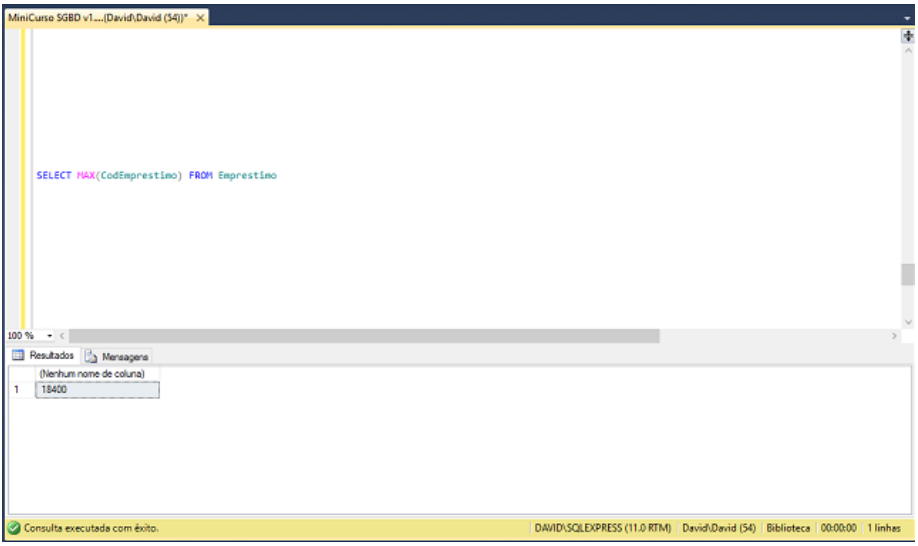


Figure 85 - SqlServer Select

Com a clausula MIN ele irá selecionar o menor valor de uma determinada coluna, sintaxe:
SELECT MIN(CodEmprestimo) FROM Emprestimo
Ele irá selecionar a coluna “CodEmprestimo” e pegara o seu menor valor.

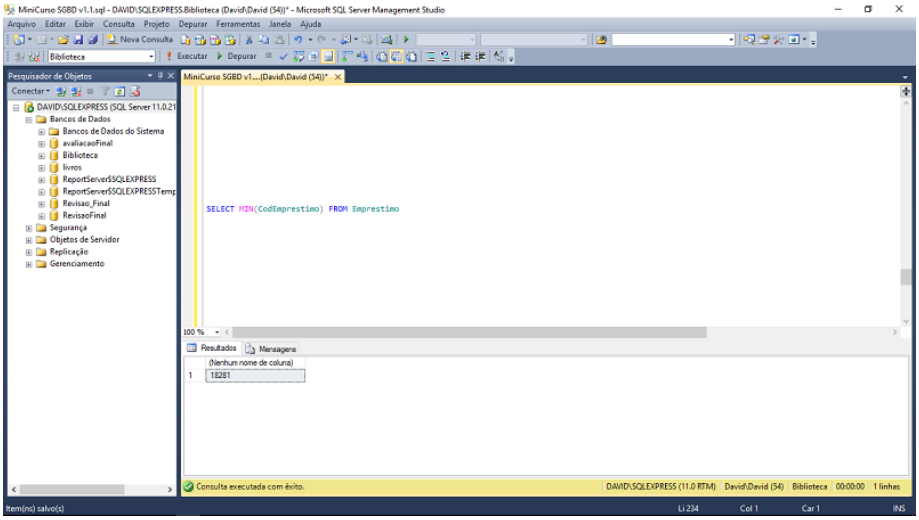


Figure 86 - SqlServer Select

Passo a passo de como fazer pesquisas:

Primeiramente selecione toda a sintaxe do SELECT e suas declarações.

Depois clique no ícone na parte superior da tela, lembrando que você tem quer estar dentro do banco de dados para funcionar, ou apertando a tecla F5.

Caso o comando não possua erros irá aparecer uma tabela como na Figura anterior se for MAX e você fizer MIN será como na Figura, irá aparecer o maior número de registros da coluna “CodEmprestimo” ser for MAX e se for MIN o menor número de registros desta coluna.

WHERE

Um exemplo de sua aplicação é demonstrado a seguir:

Sintaxe WHERE

```
SELECT column_name,column_name
  FROM table_name
  WHERE column_name operator value;
```

Operadores para a cláusula WHERE

Operador	Descrição
=	Igual
<>	Diferente **Nota:** Em algumas versões do SQL esse operador talvez seja escrito como !=.
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a
BETWEEN	Entre um valor específico
LIKE	Pesquisar um padrão
IN	Usado para verificar se um valor está contido num conjunto de constantes
AND	Retorna o valor se tanto a primeira condição quanto a segunda são verdadeiras
OR	Retorna o valor se uma das opções for verdadeira

Figure 87 - SqlServer Select - Where

IdCliente	nomeCliente	nomeContato	endereco	cidade	cep	pais
1	Pedro da Costa Nascimento	Pedro da Costa	Rua das Palmeiras N.81	Anápolis	75030-057	Brasil
2	Antonio Moreno Taqueria	Antonio Moreno	Mataderos 2312	México D.F.	05023	México
3	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlim	12209	Alemanha
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	Londres	WA1 1DP	Reino Unido
5	Christina Berglund	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Suíça

Figure 88 - SqlServer Tabela Clientes

*Exemplo da aplicação dos operadores AND e OR - **Operador AND***

```
SELECT * FROM Clientes
  WHERE pais=' Alemanha'
  AND cidade='Berlim';
```

Operador OR

```
SELECT * FROM Clientes
  WHERE cidade='Berlim'
  OR cidade='München';
```

Combinando AND e OR

```
SELECT * FROM Clientes
  WHERE pais=' Alemanha'
  AND (cidade='Berlim' OR cidade='München');
```

Exemplo da aplicação do operador LIKE

O seguinte comando seleciona todos os clientes de um país contendo o padrão 'região':

```
SELECT * FROM Clientes
  WHERE pais LIKE '%região%';
```

JOIN

INNER JOINS (as operações típicas realizadas com JOIN, que usam alguns operadores de comparação como = [igual] ou <> [diferente]). Usam operadores de comparação para unir linhas de duas tabelas baseadas nos valores comuns de cada tabela. Por exemplo, retornar todos as linhas onde o id do estudante é igual tanto na tabela estudante quanto na tabela cursos.

OUTER JOINS são especificadas em uma das seguintes categorias, onde elas são especificadas na cláusula FROM:

LEFT JOIN ou LEFT OUTER JOIN: O resultado final de uma LEFT OUTER JOIN incluí todas as linhas da tabela a esquerda especificada na cláusula do LEFT OUTER, não apenas aqueles em que as colunas do JOIN coincidem. Quando uma linha na tabela a esquerda não tem correspondência com uma linha da tabela da direita, o resultado final possui resultados 'NULL' para todas as linhas selecionadas vindas da tabela da direita.

RIGHT JOIN ou RIGHT OUTER JOIN: Um RIGHT OUTER JOIN é o reverso de um LEFT OUTER JOIN. Todas as linhas da tabela da direita são retornadas. Valores em 'NULL' são retornados da tabela da esquerda toda vez que não há correspondência com as linhas da tabela da direita.

FULL JOIN ou FULL OUTER JOIN: Um FULL OUTER JOIN retorna todas as linhas de ambas as tabelas, LEFT e RIGHT. Toda vez que uma linha não tem correspondência na outra tabela, as listas de linhas selecionadas da outra tabela vão conter valores 'NULL'. Quando tiver correspondência entre as tabelas, a tabela de resultado contém todos os valores da tabela base.

INNER JOIN

Um INNER JOIN produz um resultado que é limitado ao número de linhas que possuem correspondência em ambas as tabelas nas quais estão sendo comparadas.

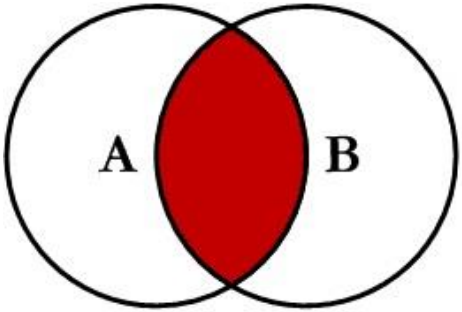


Figure 89 - Inner Join

Sintaxe INNER JOIN

```
SELECT a.Nome, b.Nome
  FROM TabelaA as A
  INNER JOIN TabelaB as B
  on a.Nome = b.Nome
```

Exemplo da execução do comando INNER JOIN:

Primeiro para a exemplificar a utilização do comando INNER JOIN será utilizado um banco de dados de uma distribuidora fictícia, criando duas tabelas, 'pedidos' e 'vendedores' e as preenchendo suas colunas respectivamente.

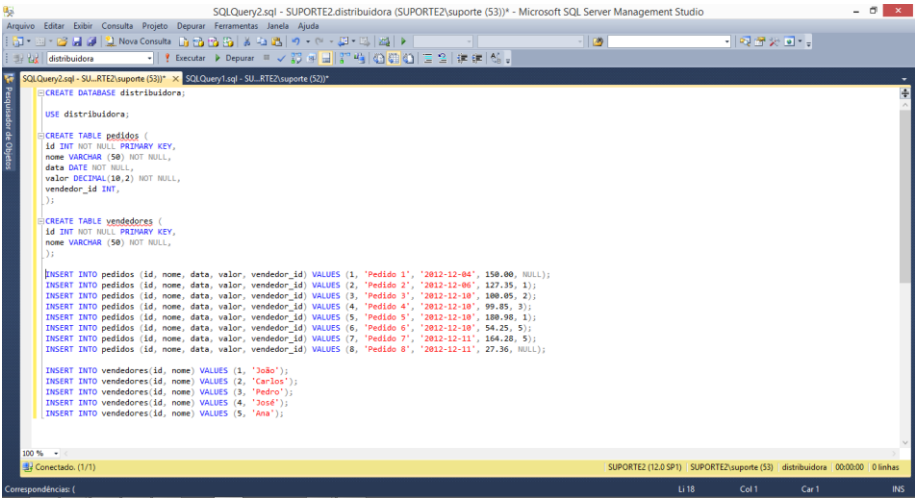


Figure 90 - Inner Join

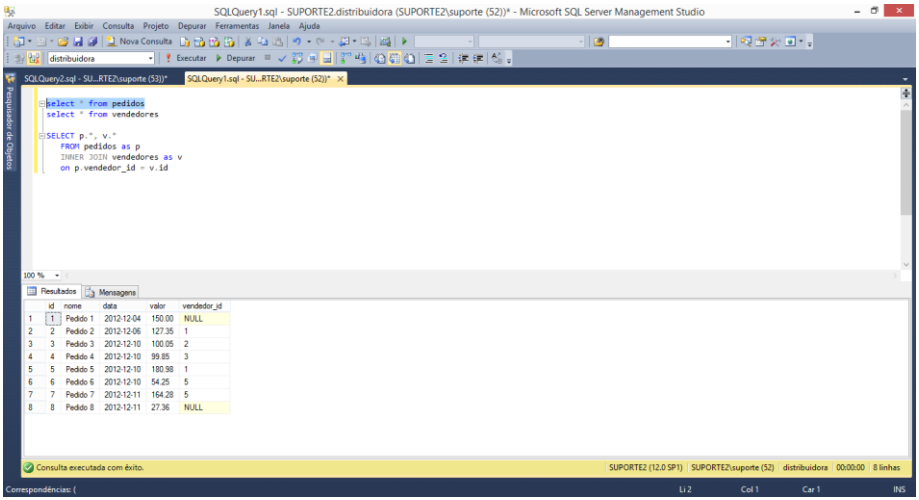


Figure 91 - Inner Join

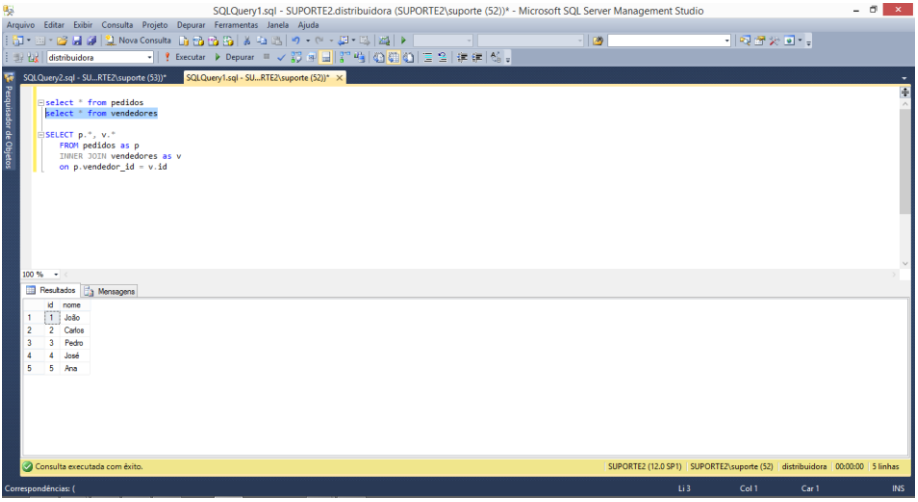


Figure 92 - Inner Join

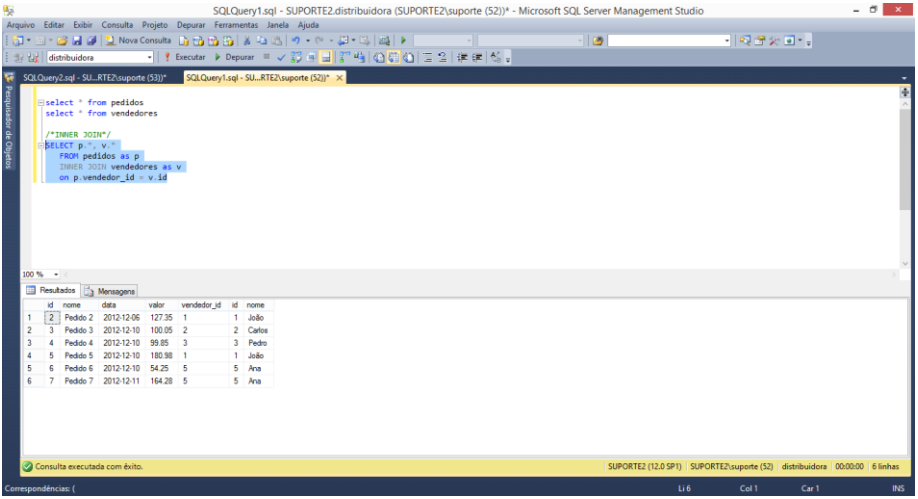


Figure 93 - Inner Join

LEFT JOIN

LEFT JOIN, é o arranjo em que todas as linhas da primeira tabela, aquela a esquerda é preservada. As linhas da segunda tabela, a da direita, apenas aparecem se possuem correspondência com as linhas da primeira tabela. Onde existem valores da tabela da esquerda, mas não da direita, a tabela lerá como 'NULL', o que significa que o valor não será setado.

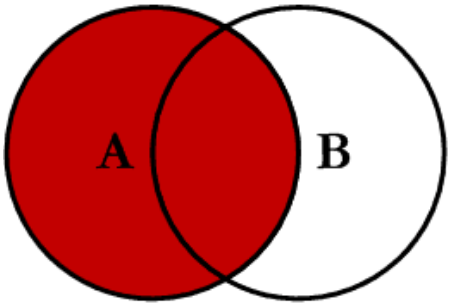


Figure 94 - Left Join

Sintaxe LEFT JOIN

```
SELECT a.Nome, b.Nome
  FROM TabelaA as A
  LEFT JOIN TabelaB as B
  on a.Nome = b.Nome
```

Exemplo da execução do comando LEFT JOIN:

O mesmo exemplo do banco de dados 'distribuidora' será utilizado para execução do código LEFT JOIN.

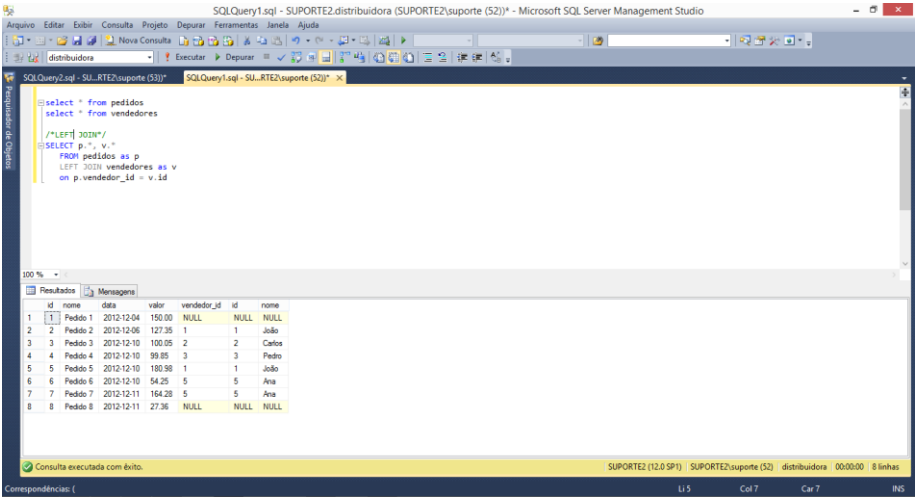


Figure 95 - Left Join

RIGHT JOIN

O RIGHT JOIN funciona de maneira similar ao LEFT JOIN só que de modo reverso a ele. Todas as linhas da tabela a direita apresentam o resultado, mas as linhas da tabela da esquerda apenas apresentam resultados se elas correspondem a tabela da direita.

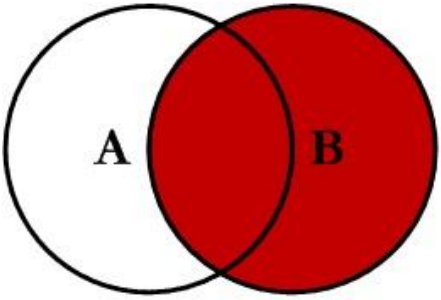


Figure 96 - Right Join

```
SELECT a.Nome, b.Nome
  FROM TabelaA as A
  RIGHT JOIN TabelaB as B
  on a.Nome = b.Nome
```

Exemplo da execução do comando RIGHT JOIN:

O mesmo exemplo do banco de dados 'distribuidora' será utilizado para execução do código RIGHT JOIN.

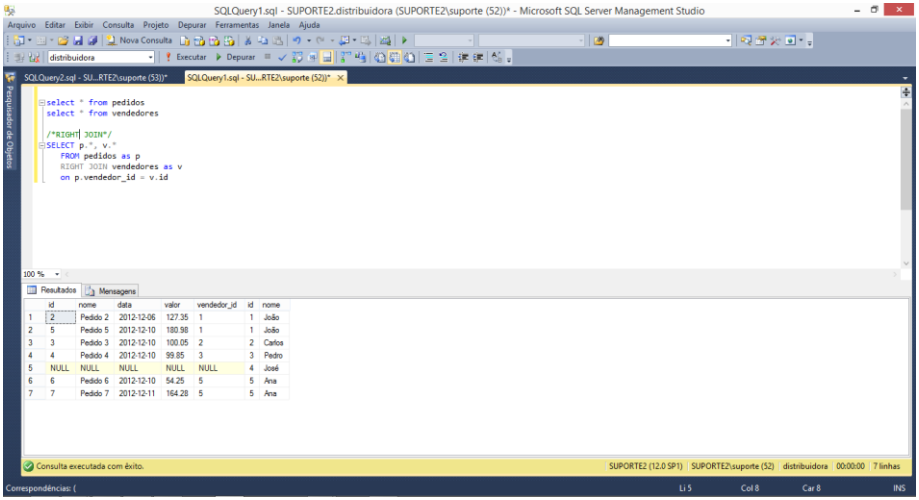


Figure 97 - Right Join

OUTER JOIN

O OUTER JOIN produz uma tabela com todas as linhas de ambas as tabelas, independente se ambas tenham ou não qualquer correspondência. Similar ao LEFT e RIGHT JOIN's, todos os espaços vazios são NULL's.

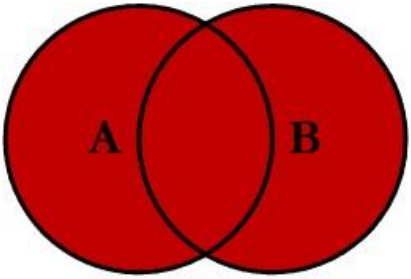


Figure 98 - Outer Join

Sintaxe OUTER JOIN

```
SELECT a.Nome, b.Nome
  FROM TabelaA as A
  FULL OUTER JOIN TabelaB as B
  on a.Nome = b.Nome

  SELECT a.Nome, b.Nome
  FROM TabelaA as A
  FULL JOIN TabelaB as B
  on a.Nome = b.Nome
```

Exemplo da execução do comando OUTER JOIN:

O mesmo exemplo do banco de dados 'distribuidora' será utilizado para execução do código OUTER JOIN.

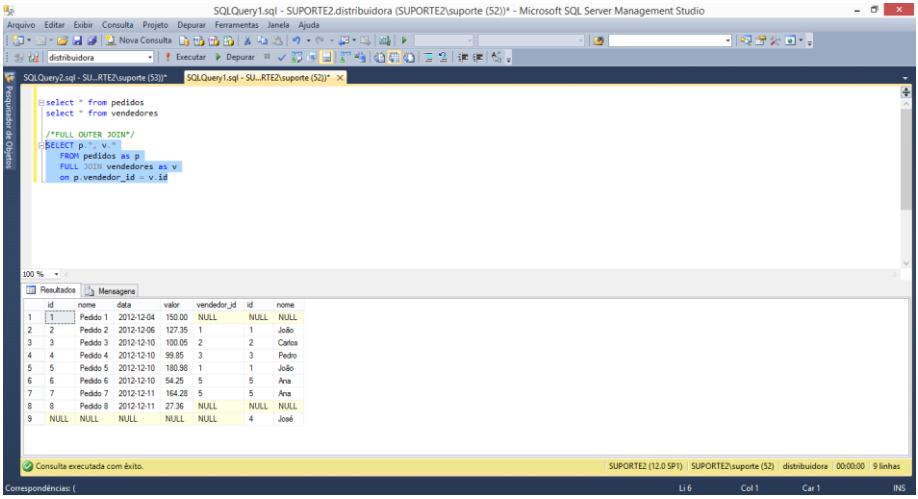


Figure 99 - Full Outer Join

VIEW

Uma VIEW é uma tabela virtual que é baseada em uma consulta realizada previamente. Uma VIEW possuí linhas e colunas, tal como uma tabela real. Os campos em uma tabela são campos de uma ou mais tabelas reais do Banco de Dados.

Você pode adicionar funções, WHERE, e JOIN a uma VIEW e apresentar os dados como se estivessem vindo de uma só tabela.

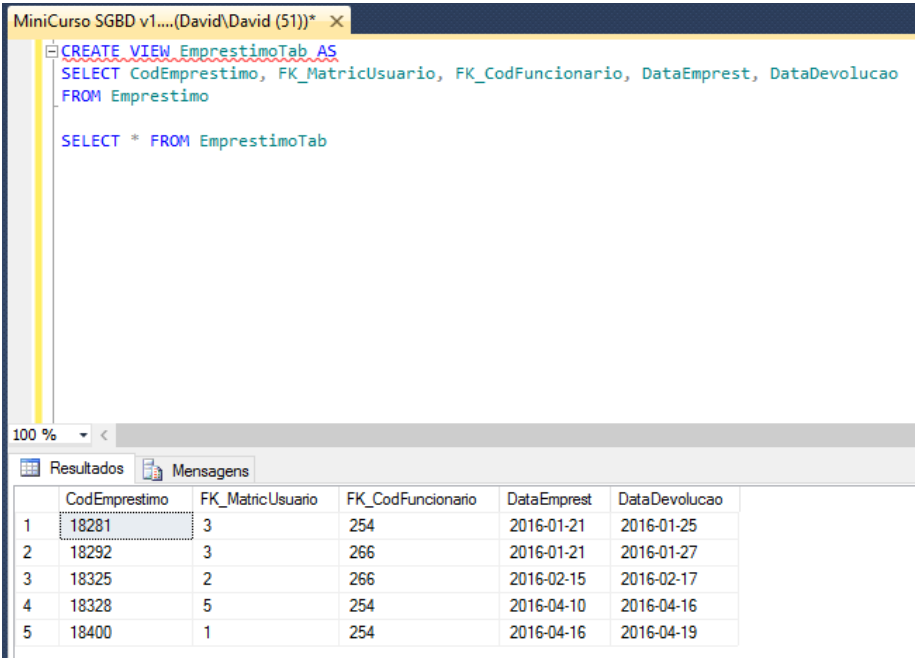
CREATE VIEW

O comando VIEW seria como uma tabela virtual, onde são pegos os dados de outras coluna e linhas de outras tabelas. Mas a VIEW não se limita a só uma tabela, ela pode pegar dados de várias tabelas e também de outras VIEW, a VIEW pode filtrar os conteúdos das tabelas e pegar só aquilo que deseja.

Sintaxe CREATE VIEW

```
CREATE VIEW nome_view AS
  SELECT nome_coluna(s)
```

FROM nome_tabela(s)
WHERE condição



	CodEmprestimo	FK_MatricUsuario	FK_CodFuncionario	DataEmprest	DataDevolucao
1	18281	3	254	2016-01-21	2016-01-25
2	18292	3	266	2016-01-21	2016-01-27
3	18325	2	266	2016-02-15	2016-02-17
4	18328	5	254	2016-04-10	2016-04-16
5	18400	1	254	2016-04-16	2016-04-19

Figure 100 - Create View

Como apresentado na imagem foi criado uma VIEW com o nome de EmprestimoTab, onde foram selecionadas algumas colunas da tabela Emprestimo. E depois disso foi usado o comando SELECT para apresentar a VIEW que acabou de ser criada.

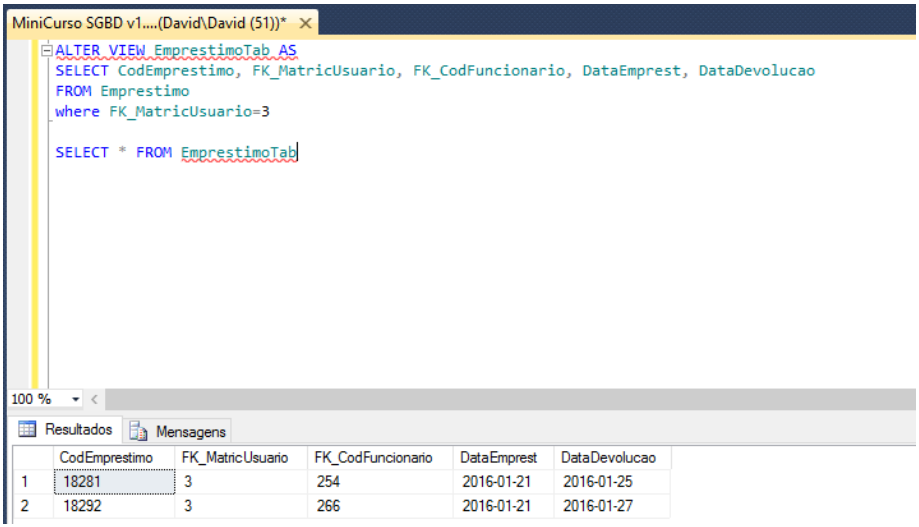
ALTER VIEW

O comando ALTER VIEW serve para fazer alterações na tabela.

Sintaxe ALTER VIEW

```
ALTER VIEW nome_view AS
  SELECT nome_coluna(s)
  FROM tabela_nome(s)
  WHERE condição
```

Na Figura abaixo é apresentado o exemplo da execução de um ALTER VIEW.



	CodEmprestimo	FK_MatricUsuario	FK_CodFuncionario	DataEmprest	DataDevolucao
1	18281	3	254	2016-01-21	2016-01-25
2	18292	3	266	2016-01-21	2016-01-27

Figure 101 - Alter View

Depois de alterado a VIEW para que ela pegue os dados com FK_MatricUsuario=3, quando feito o SELECT da VIEW os valores apresentados só tinham com o FK_MatricUsuario=3, pois esses são os únicos valores dentro da VIEW.

DROP VIEW

Para excluir uma VIEW que foi criada se utiliza o comando DROP VIEW.

Sintaxe DROP VIEW

DROP VIEW nome_view

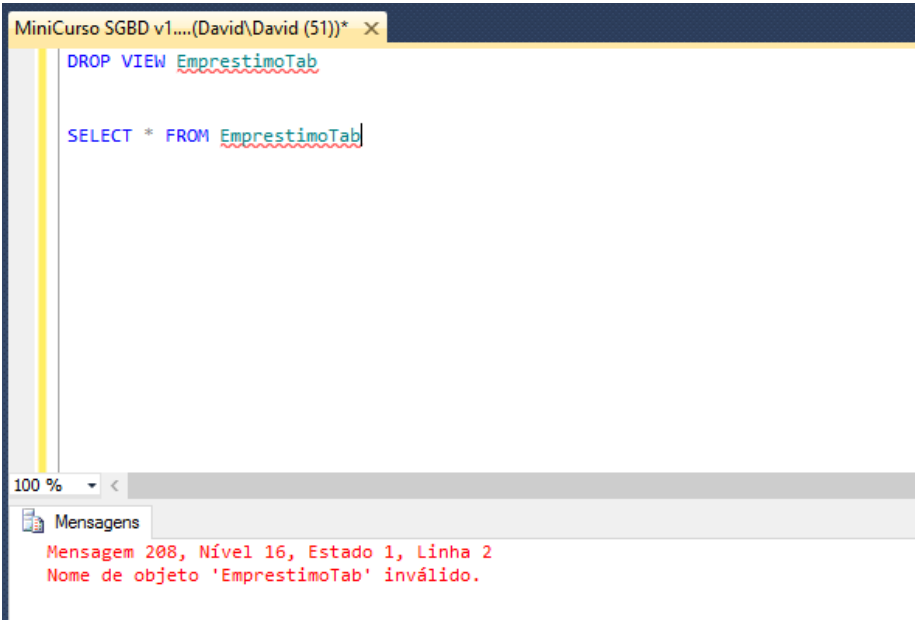


Figure 102 - Drop View

Depois que a VIEW é removida se for usado o SELECT para procurar ocorrera um erro, pois não será possível achar a VIEW, já que ela foi excluída.

Referências bibliográficas

w3schools.com - SQL Tutorial (Gratuito) – Disponível em: www.w3schools.com/sql/

SQL Magazine (Pago) – Disponível em: www.devmedia.com.br/revista-sql-magazine

ELMASRI, Ramez. NAVATHE, Shankant B. **Sistemas de Banco de Dados**. 6ª ed São Paulo: Pearson Addson Wesley, 2011.

GUIMARÃES, C. C. **Fundamentos de Bancos de Dados**. Campinas: Editora da UNICAMP, 2003.

HEUSER, C. A. **Projeto de Banco de Dados**. 6. ed. Porto Alegre: Bookman, v. 4, 2008.

SILBERSCHATZ, A. et al. **Sistema de Banco de Dados**. 6ª edição. Rio de Janeiro: Elsevier, 2012.

Alberto, Carlos. (2009). **Projeto de banco de dados.** São Paulo. Ed. Bookman.

Alberto, Carlos. (1998). **Projeto de banco de dados.** São Paulo. Ed. Sagra Luzzato.

Nery, Felipe. (2004) **Banco de dados-Projeto e implementação**. São Paulo, Ed. Erica.

Machado, F., Abreu. (2002) M. **Projeto de Banco de Dados – Uma Visão Prática.** Editora Érica, São Paulo.

MEDEIROS, João Bosco; ANDRADE, Maria Margarida. Manual de elaboração de referências bibliográficas: a nova NBR 6023:2000 da ABNT: exemplos e comentários. São Paulo: Atlas, 2001.