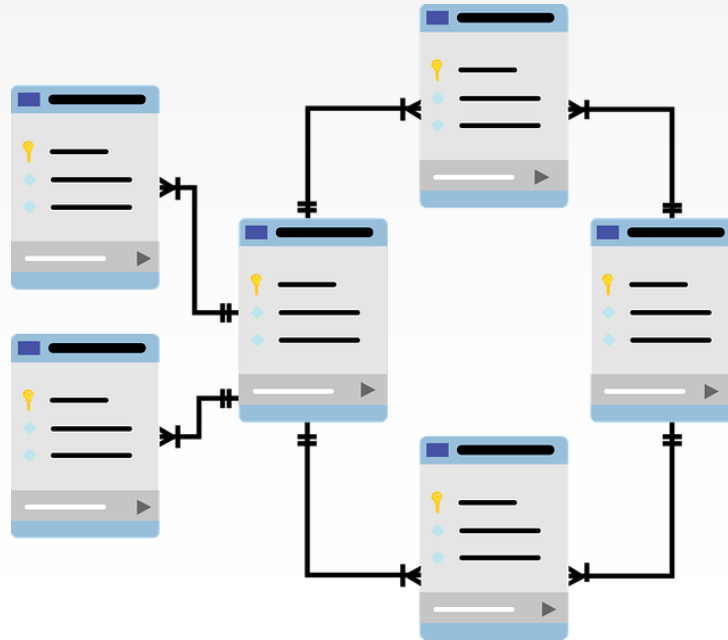


Banco de Dados

Módulo Intermediário





IMPLEMENTING A RELATIONAL MODEL

Relational Model

- **Implementing**
 - *Antes que possamos utilizar instruções DDL para criar um banco de dados no SQL Server, primeiro precisamos explorar um modelo de dados relacional que será implementado*
 - *Nosso modelo conterá algumas entidades (tabelas de dados) com definições de chave primária e alguns relacionamentos (restrições de chave estrangeira) entre as diferentes entidades*

Relational Model

- **Implementing**
 - *Uma chave primária é um valor único que identifica uma linha em uma tabela*
 - *O valor da chave pode ser um valor de coluna única ou pode ser composto de vários valores de coluna*
 - *Uma chave estrangeira é uma coluna ou várias colunas em uma tabela que fornecem um link para outra tabela*
 - *As colunas de chave estrangeira em uma tabela referiam a coluna de chave primária na outra tabela*

Relational Model

- ***Implementing***
 - *O modelo relacional terá como escopo um sistema simples de reserva de hotel*
 - *Este sistema de reservas rastreará as informações de reservas dos clientes*



Relational Model

- Modelo de BD-R simples*

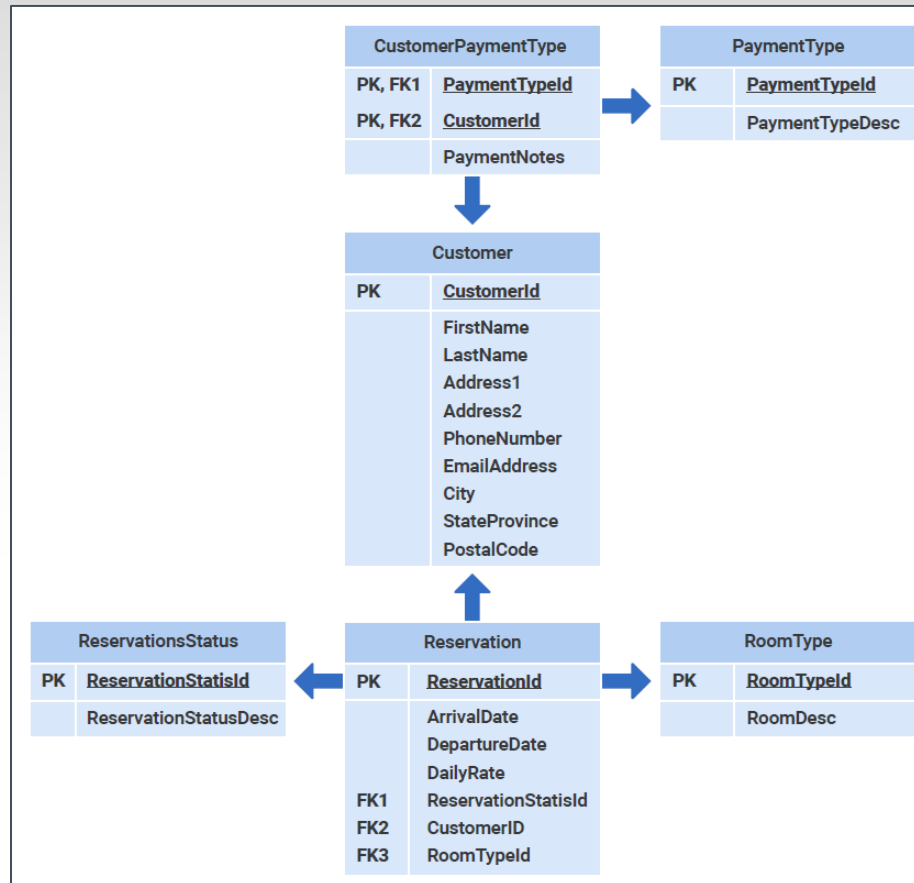


Figura 1 – Modelo Relacional Hipotético

Relational Model

- **Implementing**

- Ao *analisar este modelo*, podemos *visualizar* que ele *contém várias entidades para gerenciar informações relacionadas a reservas*
- Cada entidade é composta por alguns **atributos** (colunas) onde **um** ou **mais atributos** são *identificados* como a **chave primária** (nomes em negrito e sublinhado)
- As *setas* entre as entidades representam **relacionamentos** entre elas

Relational Model

- **Implementing**
 - *Utilizamos o modelo de entidades (**hipotético**), atributos, chaves primárias e relacionamentos para desenvolver um BD físico no SQL Server que represente o modelo relacional*
 - *Para construir um BD físico a partir desse modelo, precisamos identificar os diferentes objetos do SQL Server que serão definidos com base nesse modelo*
 - *Para cada **entidade**, criaremos uma **tabela***
 - *Para cada **atributo** de cada **entidade**, criaremos uma **coluna***

Relational Model

- *Implementing*
 - Para cada **chave primária**, criaremos um **índice clusterizado exclusivo** (observe que uma **chave primária** também pode ser criada usando um **índice não clusterizado exclusivo**)

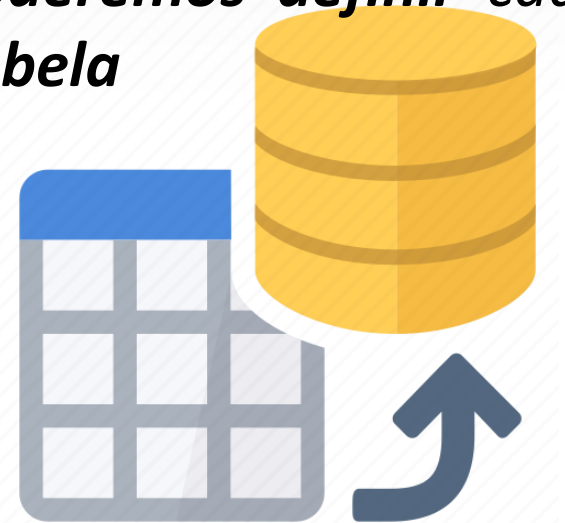


- ***Exemplo (01):***
 - ***Criando um banco de dados para contemplar todos os novos objetos do modelo relacional hipotético***
 - ***O novo BD será intitulado de **RoomReservation*****

```
USE master;
```

```
CREATE DATABASE RoomReservation;
```

- **Table**
 - *Para começar a construir objetos do BD **RoomReservation** a partir do modelo, criaremos os **objetos (tables)***
 - *Para criar uma **tabela** no **SQL Server**, precisamos **utilizar** a instrução **CREATE TABLE***
 - *Com a instrução **CREATE TABLE**, poderemos definir cada **tabela** e **todas** as **colunas** em cada **tabela***



- ***Sintaxe***

```
CREATE TABLE <table_name> (  
<column_definition> [...N]  
);
```

– *Onde:*

- <table_name> = *Name of table*
- <column_definition> = *column_name data_type, [NULL | NOT NULL]*

- **Table**

- **Exemplo (02):**

- **Criando a tabela *Customer***

```
USE RoomReservation;
```

```
CREATE TABLE dbo.Customer (  
  CustomerId      INT CONSTRAINT nnCustomerCustomerId NOT NULL,  
  FirstName       NVARCHAR(50) CONSTRAINT nnCustomerFirstName NOT NULL,  
  LastName        NVARCHAR(50) CONSTRAINT nnCustomerLastName NOT NULL,  
  Address1        NVARCHAR(100) CONSTRAINT nnCustomerAddress1 NOT NULL,  
  Address2        NVARCHAR(100) NULL,  
  PhoneNumber     NVARCHAR(22) CONSTRAINT nnCustomerPhoneNumber NOT NULL,  
  EmailAddress    VARCHAR(100) NULL,  
  City            VARCHAR(100) CONSTRAINT nnCustomerCity NOT NULL,  
  StateProvince  NVARCHAR(100) CONSTRAINT nnCustomerStateProvince NOT NULL,  
  PostalCode      NVARCHAR(100) CONSTRAINT nnCustomerPostalCode NOT NULL  
);
```

- **Primary Key**

- **Exemplo (03):**

- *Criando a restrição de chave primária, que garante que dois registros na tabela não tenham o mesmo **CustomerId***

```
USE RoomReservation;
```

```
GO
```

```
ALTER TABLE dbo.Customer  
    ADD CONSTRAINT pkCustomerCustomerId  
    PRIMARY KEY CLUSTERED (CustomerId);
```

- **Table**
 - **Exemplo (04):**
 - **Criando a tabela *Reservation***

```
USE RoomReservation;
```

```
CREATE TABLE dbo.Reservation (  
  ReservationId INT CONSTRAINT nnReservationReservationId NOT NULL,  
  ArrivalDate   DATETIME CONSTRAINT nnReservationArrivalDate NOT NULL,  
  DepartureDate DATETIME CONSTRAINT nnReservationDepartureDate NOT NULL,  
  DailyRate     SMALLMONEY CONSTRAINT nnReservationDailyRate NOT NULL,  
  ReservationStatusID INT  
                                CONSTRAINT nnReservationReservationStatusID NOT NULL,  
  CustomerId    INT CONSTRAINT nnReservationCustomerId NOT NULL,  
  RoomTypeID    INT CONSTRAINT nnReservationRoomTypeID NOT NULL  
);
```

- **Primary Key**
 - **Exemplo (05):**
 - **Criando a restrição de chave primária da tabela *Reservation***

```
USE RoomReservation;
```

```
GO
```

```
ALTER TABLE dbo.Reservation  
    ADD CONSTRAINT pkReservationReservationId  
        PRIMARY KEY CLUSTERED (ReservationId);
```


- **Table**

- **Exemplo (06):**

- **Criando a tabela *RoomType* e a restrição de chave primária da tabela**

```
USE RoomReservation;
```

```
CREATE TABLE dbo.RoomType (  
    RoomTypeId      INT CONSTRAINT nnRoomTypeRoomTypeId NOT NULL,  
    RoomDesc        NVARCHAR(1000) CONSTRAINT nnRoomTypeRoomDesc NOT NULL  
);
```

```
ALTER TABLE dbo.RoomType  
    ADD CONSTRAINT pkRoomTypeRoomTypeId  
        PRIMARY KEY CLUSTERED(RoomTypeId);
```

- **Table**

- **Exemplo (07):**

- **Criando a tabela *ReservationStatus* e a restrição de chave primária da tabela**

```
CREATE TABLE dbo.ReservationStatus (  
    ReservationStatusId INT  
        CONSTRAINT nnReservationStatusReservationStatusId NOT NULL,  
    ReservationStatusDesc NVARCHAR(50)  
        CONSTRAINT nnReservationStatusReservationStatusDesc NOT NULL  
);
```

```
ALTER TABLE dbo.ReservationStatus  
    ADD CONSTRAINT pkReservationStatusReservationStatusId  
        PRIMARY KEY CLUSTERED (ReservationStatusId);
```

- **Table**

- **Exemplo (08):**

- **Criando a tabela *PaymentType* e a restrição de chave primária da tabela**

```
CREATE TABLE dbo.PaymentType (  
    PaymentTypeId          INT CONSTRAINT nnPaymentTypePaymentTypeId NOT NULL,  
    PaymentTypeDesc        NVARCHAR(50)  
                           CONSTRAINT nnPaymentTypePaymentTypeDesc NOT NULL  
);
```

```
ALTER TABLE dbo.PaymentType  
    ADD CONSTRAINT pkPaymentTypePaymentTypeId  
        PRIMARY KEY CLUSTERED (PaymentTypeId);
```

- **Table**

- **Exemplo (09):**

- **Criando a tabela *CustomerPaymentType* e a restrição de chave primária da tabela**

```
CREATE TABLE dbo.CustomerPaymentType (  
PaymentTypeId INT CONSTRAINT nnCustomerPaymentTypePaymentTypeId NOT NULL,  
CustomerId INT CONSTRAINT nnCustomerPaymentTypeCustomerId NOT NULL,  
PaymentNotes NVARCHAR(2000) NULL  
);
```

```
ALTER TABLE dbo.CustomerPaymentType  
ADD CONSTRAINT pkCustomerPaymentTypePaymentTypeIdCustomerId  
PRIMARY KEY CLUSTERED (PaymentTypeId, CustomerId);
```

- **Foreign Key**

- **Exemplo (10):**

- *Criando uma restrição **FOREIGN KEY** na tabela **Reservation** referenciando a tabela **Customer***

```
USE RoomReservation;
```

```
GO
```

```
ALTER TABLE dbo.Reservation
```

```
    ADD CONSTRAINT fkReservationCustomerId
```

```
        FOREIGN KEY(CustomerId)
```

```
            REFERENCES dbo.Customer (CustomerId);
```

- ***Foreign Key***

- ***Exemplo (11):***

- ***Criando restrições de FOREIGN KEY adicionais***

```
ALTER TABLE dbo.Reservation
    ADD CONSTRAINT fkReservationRoomTypeId
    FOREIGN KEY (RoomTypeId)
    REFERENCES dbo.RoomType (RoomTypeId);
```

```
ALTER TABLE dbo.Reservation
    ADD CONSTRAINT fkReservationReservationStatusId
    FOREIGN KEY (ReservationStatusId)
    REFERENCES dbo.ReservationStatus (ReservationStatusId);
```

- ***Foreign Key***

- ***Exemplo (12):***

- ***Criando restrições de FOREIGN KEY adicionais***

```
ALTER TABLE dbo.CustomerPaymentType
    ADD CONSTRAINT fkCustomerPaymentTypePaymentTypeId
    FOREIGN KEY (PaymentTypeId)
    REFERENCES dbo.PaymentType (PaymentTypeId);
```

```
ALTER TABLE dbo.CustomerPaymentType
    ADD CONSTRAINT fkCustomerPaymentTypeCustomerId
    FOREIGN KEY (CustomerId)
    REFERENCES dbo.Customer (CustomerId);
```

Validating

- ***Validando o BD***

- Assim que ***finalizamos a criação de banco de dados a partir de um modelo de dados***, devemos ***valida-lo*** para ter ***certeza de que está correto***
- ***Esse processo de validação*** permite ***garantir que todas as regras de integridade de dados*** que construímos no banco de dados ***físico*** sejam atendidas ***corretamente***



- *Validando o BD*

- *Será necessário validar:*

- *Todas as linhas inseridas ou atualizadas devem ter um valor específico definido para qualquer coluna definida como NOT NULL*
 - *As colunas que são PRIMARY KEYS **não** permitem valores duplicados*
 - *As colunas que têm restrições de chave estrangeira **não** permitem dados que **não** tenham um registro correspondente na tabela referenciada*

- *Inserindo Dados Iniciais*

- *Exemplo (13):*

```
USE RoomReservation;
```

```
GO
```

```
SET NOCOUNT ON;
```

```
-- Inserindo registros na PaymentType
```

```
INSERT INTO PaymentType
```

```
VALUES
```

```
(1, 'Visa'),
```

```
(2, 'MasterCard'),
```

```
(3, 'American Express');
```

- *Inserindo Dados Iniciais*

- *Exemplo (14):*

-- Inserindo registro na Customer

INSERT INTO Customer

VALUES

(1, 'Greg', 'Larsen', '123 Some Place', NULL,
'123-456-7890', Null, 'MyCity', 'MA', '12345');

-- Inserindo registros na ReservationStatus

INSERT INTO ReservationStatus

VALUES

(1, 'Booked'),
(2, 'Cancelled');

- *Inserindo Dados Iniciais*

- *Exemplo (15):*

-- Inserindo registros na RoomType

```
INSERT INTO RoomType
```

```
VALUES
```

```
(1, 'Kingsize'),
```

```
(2, 'Queen'),
```

```
(3, 'Double');
```

- *Testando várias restrições com instruções INSERT*
 - *Exemplo (16):*

```
USE RoomReservation;
```

```
GO
```

```
-- Violando a restrição NOT NULL
```

```
INSERT INTO Reservation
```

```
VALUES
```

```
(1, '2011-8-1 5:00 PM', '2011-8-2 9:00 AM', 150.99, NULL, 1, 1);
```

Cannot insert the value NULL into column 'ReservationStatusID', table 'RoomReservation.dbo.Reservation'; column does not allow nulls. INSERT fails.

- *Testando várias restrições com instruções INSERT*
 - *Exemplo (17):*

```
-- Violando a restrição PRIMARY KEY  
INSERT INTO RoomType  
VALUES  
(3, 'Suite');
```

Violation of PRIMARY KEY constraint 'pkRoomTypeRoomTypeId'. Cannot insert duplicate key in object 'dbo.RoomType'. The duplicate key value is (3).

- ***Testando várias restrições com instruções INSERT***

- ***Exemplo (18):***

```
-- Violando a restrição FOREIGN KEY
```

```
INSERT INTO CustomerPaymentType
```

```
VALUES
```

```
(1, 2, 'Will need an internet connection');
```

The INSERT statement conflicted with the FOREIGN KEY constraint "fkCustomerPaymentTypeCustomerId". The conflict occurred in database "RoomReservation", table "dbo.Customer", column 'CustomerId'.

- *Instruções INSERT finais (êxito)*

- *Exemplo (19):*

-- NOT NULL constraint

```
INSERT INTO Reservation  
VALUES
```

```
(1, '2011-8-1 5:00 PM', '2011-8-2 9:00 AM', 150.99, 1, 1, 1);
```

-- Primary Key Constraint

```
INSERT INTO RoomType  
VALUES
```

```
(4, 'Suite');
```

-- Foreign Key Constraint

```
INSERT INTO CustomerPaymentType  
VALUES
```

```
(1, 1, 'Will need an internet connection');
```




INSTRUÇÃO SELECT

SELECT Statement

- **Instrução SELECT**

- *Embora a instrução **SELECT** ofereça muitos argumentos que podem torna-la muito **complexa**, em sua forma mais simples ela consiste em **duas palavras-chave**: uma lista de colunas e um nome de tabela*



SELECT Statement

- **Instrução SELECT**

- Uma instrução **SELECT básica** que retorna dados de uma única tabela consiste em três partes: a **lista** de **colunas**, a cláusula **FROM** e a cláusula **WHERE**

- **Sintaxe:**

```
SELECT <Column List>  
FROM <table name>  
WHERE <where criteria>;
```

SELECT Statement

- ***Instrução SELECT***

- ***Exemplo (01):***

- *Selecionar as colunas **DepartmentID**, **Name**, **GroupName** e **ModifiedDate** da tabela **Department**, pertinente ao esquema **HumanResources***

```
USE MyAdventureWorks;
```

```
SELECT DepartmentID, Name, GroupName, ModifiedDate  
FROM HumanResources.Department;
```

SELECT Statement

- **Instrução SELECT**

- **Exemplo (02):**

- **Selecionar duas colunas distintas, ProductCategoryID e Name, da tabela *Production.ProductCategory***
 - **Como esta instrução SELECT tem uma cláusula WHERE, ela limita as linhas retornadas da tabela apenas para aquelas linhas que têm um valor ProductCategoryID menor do que 2**

```
USE MyAdventureWorks;
```

```
SELECT ProductCategoryID,  
       Name  
FROM Production.ProductCategory  
WHERE ProductCategoryID < 2;
```

Resultados		Mensagens
	ProductCategoryID	Name
1	1	Bikes

SELECT Statement

- **Instrução SELECT**

- **Exemplo (03):**

- *Selecionar todas as colunas da tabela **Departament**, simplesmente substitua a lista de colunas por um **asterisco** (*)*

```
USE MyAdventureWorks;
```

```
SELECT *  
FROM HumanResources.Department;
```

WHERE Clause

- **Cláusula WHERE**

- *Na maioria das vezes, será necessário retornar somente subconjuntos dos dados*
- *Por exemplo, imagine que desejamos elaborar um query que procure um departamento específico ou todos os departamentos que começam com a letra “P”*
- *Dessa forma, seria necessário incluir a cláusula **WHERE** como parte da instrução **SELECT***
- *A cláusula **WHERE** sempre vem após a instrução **FROM** e precede a cláusula **ORDER BY***

WHERE Clause

- **Cláusula WHERE**

- A cláusula **WHERE** de uma instrução **SELECT** é **opcional**
- A cláusula **WHERE** é usada para restringir as linhas que são retornadas de uma instrução **SELECT**
- O mecanismo de banco de dados avalia **cada linha** em relação à cláusula **WHERE** e só retorna as linhas se elas atenderem à **condição** ou **condições** de pesquisa identificadas na cláusula **WHERE**
- À medida que escrevemos mais instruções **SELECT**, descobriremos que a maioria de suas instruções **SELECT** provavelmente conterá uma cláusula **WHERE**

WHERE Clause

- **Cláusula WHERE**

- Uma cláusula **WHERE** simples conterá uma **única condição** de **pesquisa**, enquanto uma cláusula **WHERE** **mais complexa** pode conter **muitas condições**
- Quando **várias condições** são usadas em uma cláusula **WHERE**, elas serão **combinadas logicamente** usando os **operadores lógicos AND** e **OR**
- **Não** há limite para o **número de condições diferentes** que podem ser **incluídas** em uma **instrução SELECT**

WHERE Clause

- **Operadores de Comparação**

- **Exemplo (04):**

- 1ª condição verifica se a linha tem o valor **Blue** na coluna **Color**
 - 2ª condição verifica se o valor na coluna **ProductId** é maior que **900**
 - Como o operador **AND** é usado, **ambas as condições** devem ser **verdadeiras** para que **uma linha** seja retornada dessa consulta

```
USE MyAdventureWorks;
```

```
SELECT *  
FROM Production.Product  
WHERE Color = 'Blue'  
AND ProductID > 900;
```

WHERE Clause

- *Operadores de Comparação*

- *Exemplo (05):*

- *Retorna linhas **Production.Product** em que o valor **ProductID** é maior que **900** e o valor na coluna **Color** é **Blue** ou **Green***

```
USE MyAdventureWorks;
```

```
SELECT *
```

```
FROM Production.Product
```

```
WHERE ProductID > 900
```

```
AND (Color = 'Blue' OR Color = 'Green');
```

WHERE Clause

- *Operadores de Comparação*

- *Exemplo (06):*

- *Adicionamos o operador **NOT** logo após a operação **AND** para indicar que desejamos produtos que **não** sejam **Blue** ou **Green***

```
USE MyAdventureWorks;
```

```
SELECT *
```

```
FROM Production.Product
```

```
WHERE ProductID > 900
```

```
AND NOT (Color = 'Blue' OR Color = 'Green');
```

WHERE Clause

- **Cláusula WHERE**
 - *Existem várias implementações diferentes da cláusula, a citar:*
 - Operadores de **comparação**
 - O operador **BETWEEN**
 - Uma cláusula **WHERE** com **várias condições**
 - Uma **pesquisa** em uma **lista** de **valores**
 - Uma **pesquisa** com **curinga**

WHERE Clause

- ***Operadores de Comparação***
 - ***O SQL Server oferece vários operadores de comparação:***
 - = (*igual*)
 - < (*menor que*)
 - > (*maior que*)
 - >= (*maior ou igual a*)
 - *Dentre outros*
 - ***Mesclar esses operadores com a cláusula WHERE pode ajudar a limitar os dados de várias maneiras***

WHERE Clause

- *Operadores de Comparação*

- *Exemplo (07):*

- *Selecionar a tupla pertinente ao departamento de número 4*

USE MyAdventureWorks;

```
SELECT *  
FROM HumanResources.Department  
WHERE DepartmentID = 4;
```

Resultados		Mensagens		
	DepartmentID	Name	GroupName	ModifiedDate
1	4	Marketing	Sales and Marketing	2002-06-01 00:00:00.000

WHERE Clause

- ***Operadores de Comparação***

- ***Exemplo (08):***

- ***Selecionar as tuplas cujo DepartmentID seja superior a 4***

```
USE MyAdventureWorks;
```

```
SELECT *  
FROM HumanResources.Department  
WHERE DepartmentID > 4;
```


WHERE Clause

- *Operadores de Comparação*

- *Exemplo (09):*

- *Recuperar todas as vendas desde 01/05/2007 até 12/12/2007 com o uso do **BETWEEN***

```
USE MyAdventureWorks;
```

```
SELECT AccountNumber,  
       SalesOrderID,  
       OrderDate
```

```
FROM Sales.SalesOrderHeader
```

```
WHERE OrderDate BETWEEN '05/01/2007' AND '12/31/2007';
```

WHERE Clause

- ***Operadores de Comparação***

- ***Exemplo (10):***

- ***Gerando um relatório que implementa múltiplas condições de filtragem***

```
USE MyAdventureWorks;
```

```
SELECT SalesOrderDetailID,  
       OrderQty,  
       ProductID,  
       ModifiedDate
```

```
FROM Sales.SalesOrderDetail
```

```
WHERE ModifiedDate BETWEEN '05/01/2007' AND '12/31/2007' AND  
      ProductID = 809;
```

ORDER BY

- **Instrução SELECT**

- **Exemplo (12):**

- **Selecionando todas as colunas da tabela *Department*, cujo a tabela resultante será classificada (ordenada) pelos valores da coluna *DepartmentID* de forma *descendente***

USE MyAdventureWorks;

```
SELECT *  
FROM HumanResources.Department  
ORDER BY DepartmentID DESC;
```

DepartmentID	Name	GroupName	ModifiedDate
16	Executive	Executive General and Administration	2002-06-01 00:00:00.000
15	Shipping and Receiving	Inventory Management	2002-06-01 00:00:00.000
14	Facilities and Maintenance	Executive General and Administration	2002-06-01 00:00:00.000
13	Quality Assurance	Quality Assurance	2002-06-01 00:00:00.000
12	Document Control	Quality Assurance	2002-06-01 00:00:00.000
11	Information Services	Executive General and Administration	2002-06-01 00:00:00.000
10	Finance	Executive General and Administration	2002-06-01 00:00:00.000
9	Human Resources	Executive General and Administration	2002-06-01 00:00:00.000
8	Production Control	Manufacturing	2002-06-01 00:00:00.000
7	Production	Manufacturing	2002-06-01 00:00:00.000
6	Research and Development	Research and Development	2002-06-01 00:00:00.000
5	Purchasing	Inventory Management	2002-06-01 00:00:00.000
4	Marketing	Sales and Marketing	2002-06-01 00:00:00.000
3	Sales	Sales and Marketing	2002-06-01 00:00:00.000
2	Tool Design	Research and Development	2002-06-01 00:00:00.000
1	Engineering	Research and Development	2002-06-01 00:00:00.000

- **Instrução SELECT**

- **Exemplo (13):**

- *É possível aplicar ordenação de forma **posicional**, ou seja, ao invés de informar o nome da coluna, passamos a **posição** que a mesma se encontra para a cláusula ORDER BY*

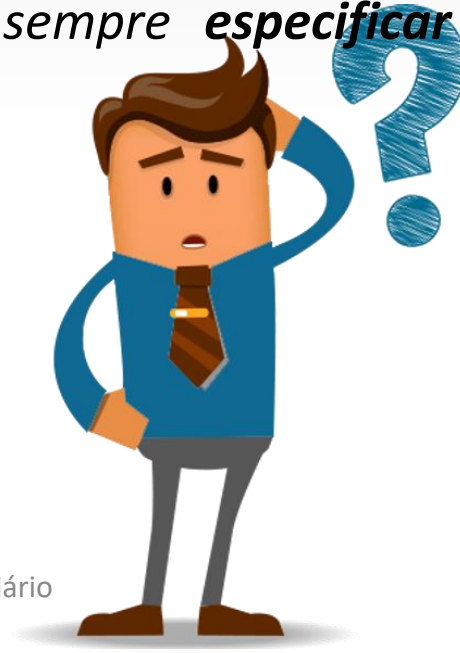
```
USE MyAdventureWorks;
```

```
SELECT DepartmentID, Name, GroupName, ModifiedDate  
FROM HumanResources.Department  
ORDER BY 3 ASC;
```

- **Instrução SELECT**

- **Observação:**

- Se existe um **índice clusterizado** na tabela e **ORDER BY não** é especificado, os resultados normalmente são retornados na ordem definida quando esse índice foi criado
 - Todavia, quando a **lógica precisa** que os dados sejam ordenados de determinada maneira, devemos sempre especificar a cláusula **ORDER BY**





CHALLENGE

Challenge #1

- **Desafio (01):**
 - *Suponha que você tenha sido solicitado pelo chefe do departamento de RH a produzir um relatório de todos os valores de **BusinessEntityID** para funcionários que têm muitas **licenças médicas** e **horas** de **férias***
 - *Para este relatório, o chefe de RH informa que a definição de "**muitas licenças**" é a seguinte:*
 - *um funcionário deve ter um valor **SickLeaveHours** maior que 68 e um valor **VacationHours** maior que 98*
 - *Você pode encontrar essas colunas junto com a coluna **BusinessEntityID** na tabela **HumanResources.Employee***

Challenge #1

- ***Desafio (01):***

```
USE MyAdventureWorks;  
  
SELECT BusinessEntityID,  
       SickLeaveHours,  
       VacationHours  
FROM HumanResources.Employee  
WHERE SickLeaveHours > 68  
       AND VacationHours > 98;
```

Resultados		Mensagens	
	BusinessEntityID	SickLeaveHours	VacationHours
1	1	69	99
2	88	69	99
3	117	69	99

Challenge #2

- **Desafio (02):**
 - ***Seu gerente quer que você produza um relatório que inclua todas as colunas associadas a alguns pedidos específicos para que ele possa revisar qual vendedor está associado a cada pedido***
 - ***Seu gerente especifica que ele está interessado apenas em um relatório que contenha linhas `SalesOrderHeader` para pedidos que tenham um `SalesOrderId` entre 43702 e 43712***

Challenge #2

- ***Desafio (02):***

```
USE MyAdventureWorks;
```

```
SELECT *
```

```
FROM Sales.SalesOrderHeader
```

```
WHERE SalesOrderID BETWEEN 43702 AND 43712;
```

WHERE Clause

- **Operador Like**

- *Pode haver situações em que desejamos localizar todos os registros em uma tabela onde uma coluna contém uma **string específica** em algum lugar dentro do valor da coluna*
- *Imagine que desejamos encontrar todos os **sobrenomes** que terminam em **“sen”***
- *Se tivermos esse **tipo** de **critério** de **seleção**, o **operador LIKE** poderá ser utilizado para essa finalidade*

WHERE Clause

- **Operador Like**

- **Sintaxe:**

match_expression [NOT] LIKE pattern [ESCAPE escape_character]

- **match_expression:**

- *Qualquer expressão válida do tipo string*

- **pattern:**

- *Cadeia de caracteres específica a ser pesquisada em **match_expression** e pode incluir os caracteres curinga válidos*
 - *Pattern pode ter no máximo de 8000 bytes*

WHERE Clause

- **Operador Like**

- **%**: qualquer string de zero ou mais caracteres

- **Exemplo:**

- **WHERE** title **LIKE** '%computer%'
 - **Encontra todos os títulos de livros com a palavra 'computer' esteja em qualquer lugar do título do livro**

- **_ (underscore)**: um único caractere

- **Exemplo:**

- **WHERE** au_fname **LIKE** '_ean'
 - **Encontrar todos os primeiros nomes de quatro letras que terminam com ean (Dean, Sean e assim por diante)**

WHERE Clause

- **Operador Like**

- **[]**: qualquer caractere único dentro do **intervalo** especificado **[a-f]** ou conjunto **[abcdef]**

- **Exemplo:**

- **WHERE** au_lname **LIKE** '**[C-P]**arsen'
 - **Encontra** os **sobrenomes** dos **autores** que **terminam** com **arsen** e **começam** com **qualquer caractere** entre **C** e **P**, por exemplo, **Carsen**, **Larsen**, **Karsen** e assim por diante
 - **Em pesquisas de intervalo**, os **caracteres incluídos no intervalo** podem **variar dependendo das regras de classificação do agrupamento**

WHERE Clause

- *Operador Like*

- **[^]**: qualquer caractere único que **não** esteja dentro do intervalo especificado **[^a-f]** ou conjunto **[^abcdef]**

- *Exemplo:*

- *WHERE au_lname LIKE 'de[^l]%'*
 - *Encontrar todos os sobrenomes dos autores começando com **de** e onde a letra seguinte **não** é **l***

WHERE Clause

- **Operador Like**
 - **escape_character**: um caractere colocado na frente de um caractere curinga para indicar que o caractere curinga é interpretado como um caractere regular e **não** como um curinga
 - É uma expressão de caractere que **não** tem padrão e deve avaliar apenas um caractere
 - **Exemplo:**
 - **WHERE** comment **LIKE** '%30\%%' **ESCAPE** '\'

WHERE Clause

- **Operador Like**

- **Exemplo (14):**

- **Recuperar todos os departamentos cujos nomes começam com *Pr* e qualquer caractere subsequente**

```
USE MyAdventureWorks;
```

```
SELECT *  
FROM HumanResources.Department  
WHERE Name LIKE 'Pr%';
```

Resultados		Mensagens		
	DepartmentID	Name	GroupName	ModifiedDate
1	7	Production	Manufacturing	2002-06-01 00:00:00.000
2	8	Production Control	Manufacturing	2002-06-01 00:00:00.000

WHERE Clause

- **Operador Like**

- **Exemplo (15):**

- *Pesquisar na coluna **LastName** da tabela **Person.Person**, qualquer **LastName** que **comece** com “C” ou “H” e **termine** com “sen”*

USE MyAdventureWorks;

```
SELECT FirstName, MiddleName, LastName  
FROM Person.Person  
WHERE LastName LIKE '[CH]%sen';
```

	FirstName	MiddleName	LastName
1	Charles	M.	Christensen
2	Ryan	NULL	Comelsen
3	Ryan	L	Comelsen
4	Jay	NULL	Henningsen

WHERE Clause

- **Operador IN**

- **Sintaxe:**

- ```
test_expression [NOT] IN (subquery | expression [,...n])
```

- **test\_expression:**

- *Qualquer expressão válida*

- **subquery:**

- *Uma subconsulta que possui um conjunto de resultados de uma coluna*
    - *Esta coluna deve ter o mesmo tipo de dados que **test\_expression***

# WHERE Clause

- **Operador IN**
  - *expression[ ,... n ]:*
    - Uma lista de expressões para testar uma correspondência
    - Todas as expressões devem ser do mesmo tipo que *test\_expression*
  - O **operador IN** aceita uma ou mais subconsultas ou uma expressão para identificar os valores que desejamos encontrar ou **não**

# WHERE Clause

- **Operador IN**

- **Exemplo (16):**

- *Retornar todas as vendas de uma lista de produtos em especial*
    - *Usando o operador **IN** para determinar se os itens de uma lista específica correspondem ao valor dado*

```
USE MyAdventureWorks;
```

```
SELECT SalesOrderDetailID,
 OrderQty,
 ProductID,
 ModifiedDate
FROM Sales.SalesOrderDetail
WHERE ProductID IN (776, 778, 747, 809);
```

# WHERE Clause

- **Operador IN**

- **Exemplo (17):**

- *Procurar três valores diferentes possíveis na coluna **Name** da tabela **Production.Product***

```
USE MyAdventureWorks;
```

```
SELECT Name, ListPrice
FROM Production.Product
```

```
WHERE Name IN('Road Tire Tube', 'Touring Pedal', 'Minipump');
```

| Resultados |                | Mensagens |
|------------|----------------|-----------|
|            | Name           | ListPrice |
| 1          | Minipump       | 19,99     |
| 2          | Road Tire Tube | 3,99      |
| 3          | Touring Pedal  | 80,99     |

# WHERE Clause

- **Operador IN**

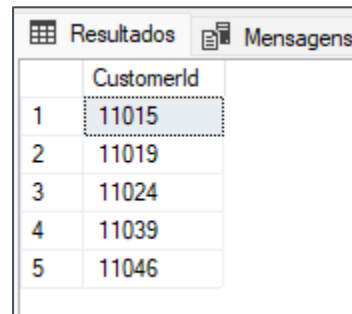
- **Exemplo (18):**

- *Outra maneira comum de identificar os valores com o operador IN é usando uma subconsulta*
    - *Usando uma subconsulta, os valores podem ser selecionados com base em código T-SQL simples ou complexo*
    - *Suponha que o chefe solicite uma consulta para uma promoção especial que a empresa está realizando*
    - *O critério para esta consulta é produzir uma lista de todos os clientes que já compraram um produto que tivesse os 4 caracteres “bike” no nome do produto*

# WHERE Clause

- **Operador IN**
  - **Exemplo (18): - continuação**

```
SELECT DISTINCT TOP 5 H.CustomerId
FROM Sales.SalesOrderHeader AS H
INNER JOIN Sales.SalesOrderDetail AS D ON (H.SalesOrderID = D.SalesOrderID)
WHERE D.ProductID IN (SELECT ProductID
 FROM Production.Product
 WHERE Name LIKE '%bike%')
ORDER BY H.CustomerID;
```



The screenshot shows a SQL Server query results window with two tabs: 'Resultados' (Results) and 'Mensagens' (Messages). The 'Resultados' tab is active, displaying a table with two columns: an implicit index column and 'CustomerId'. The table contains 5 rows of data, with the first row (CustomerId 11015) highlighted. The results are ordered by CustomerId in ascending order.

|   | CustomerId |
|---|------------|
| 1 | 11015      |
| 2 | 11019      |
| 3 | 11024      |
| 4 | 11039      |
| 5 | 11046      |



# WHERE Clause

- **Operador IN**

- **Exemplo (19):**

- O operador IN tem algumas **restrições** quando várias subconsultas são usadas
    - Existem algumas outras armadilhas que vale a pena mencionar
    - A primeira está relacionada a manipulação de valores **NULL**
    - Um valor **NULL** em uma coluna significa que a coluna **não** tem um valor
    - Portanto, ao usar operadores lógicos (**NOT**), concomitante ao operador IN, para pesquisar colunas que contenham NULL tenha cuidado para **não** obter resultados inconsistentes

# WHERE Clause

- **Operador IN**

- **Exemplo (19): - continuação**

- No exemplo a seguir, a **consulta não** retornará nenhum produto que **tenha** um **valor NULL** para a **coluna Color**

```
USE MyAdventureWorks;
```

```
GO
```

```
SELECT Name, Color
```

```
FROM Production.Product
```

```
WHERE Color NOT IN ('White', 'Grey', NULL);
```

# WHERE Clause

- **Operador IN**

- **Exemplo (20):**

- Se desejarmos **encontrar** os **Production.Products** que são “**White**”, “**Grey**” ou **NULL**, devemos **converter** esses **valores NULL** em um **valor não nulo** antes de **compará-lo** com o operador **NOT IN**

```
USE MyAdventureWorks;
```

```
GO
```

```
SELECT Name, Color
```

```
FROM Production.Product
```

```
WHERE COALESCE(Color, '') NOT IN ('White', 'Grey', COALESCE(NULL, ''));
```

- **Utilizando nomes alternativos**
  - Um **alias** pode ser um **nome alternativo** mais curto ou mais **compreensível**
  - Criado para **nomes de tabela e coluna** a fim de **facilitar o trabalho** com **agregações**, **expressões** e **query** que **envolvem várias tabelas**
  - Eventualmente, o **BD** pode **conter nomes de coluna enigmáticos**, e talvez **desejamos fornecer nomes mais significativos** para **aplicativos e usuários finais**
  - O **uso de alias** permite a **alteração ou encurtamento dos nomes de tabelas e colunas**

- ***Utilizando nomes alternativos***

- ***Exemplo (21):***

- ***Criando nomes alternativos aplicáveis para tabela e colunas***

```
USE MyAdventureWorks;
```

```
SELECT
```

```
 DepartmentID,
```

```
 Name AS "Nome do Departamento",
```

```
 GroupName AS "Nome do Grupo de Depto"
```

```
FROM HumanResources.Department AS d;
```

- ***Utilizando nomes alternativos***
  - ***Exemplo (21): - continuação***
    - ***Criando nomes alternativos aplicáveis para tabela e colunas***

| Resultados |              | Mensagens                  |                                      |
|------------|--------------|----------------------------|--------------------------------------|
|            | DepartmentID | Nome do Departamento       | Nome do Grupo de Depto               |
| 1          | 1            | Engineering                | Research and Development             |
| 2          | 2            | Tool Design                | Research and Development             |
| 3          | 3            | Sales                      | Sales and Marketing                  |
| 4          | 4            | Marketing                  | Sales and Marketing                  |
| 5          | 5            | Purchasing                 | Inventory Management                 |
| 6          | 6            | Research and Development   | Research and Development             |
| 7          | 7            | Production                 | Manufacturing                        |
| 8          | 8            | Production Control         | Manufacturing                        |
| 9          | 9            | Human Resources            | Executive General and Administration |
| 10         | 10           | Finance                    | Executive General and Administration |
| 11         | 11           | Information Services       | Executive General and Administration |
| 12         | 12           | Document Control           | Quality Assurance                    |
| 13         | 13           | Quality Assurance          | Quality Assurance                    |
| 14         | 14           | Facilities and Maintenance | Executive General and Administration |
| 15         | 15           | Shipping and Receiving     | Inventory Management                 |
| 16         | 16           | Executive                  | Executive General and Administration |

- **Utilizando nomes alternativos**
  - **Exemplo (21): - continuação**
    - **Criando nomes alternativos aplicáveis para tabela e colunas**

| Resultados |              | Mensagens            |                          |
|------------|--------------|----------------------|--------------------------|
|            | DepartmentID | Nome do Departamento | Nome do Grupo de Depto   |
| 1          | 1            | Engineering          | Research and Development |
| 2          | 2            | Tool Design          | Research and Development |
| 3          | 3            | Sales                | Sales and Marketing      |
| 4          | 4            | Marketing            | Sales and Marketing      |

A palavra-chave **AS** utilizada na query anterior é opcional ao se gerar alias para itens dentro de uma query do SQL Server

|    |    |                            |                                      |
|----|----|----------------------------|--------------------------------------|
| 10 | 10 | Finance                    | Executive General and Administration |
| 11 | 11 | Information Services       | Executive General and Administration |
| 12 | 12 | Document Control           | Quality Assurance                    |
| 13 | 13 | Quality Assurance          | Quality Assurance                    |
| 14 | 14 | Facilities and Maintenance | Executive General and Administration |
| 15 | 15 | Shipping and Receiving     | Inventory Management                 |
| 16 | 16 | Executive                  | Executive General and Administration |

- ***Manipulando Múltiplas Tabelas***
  - ***Até agora, nos concentramos principalmente manipulação de tuplas de uma única tabela***
  - ***Na prática, é muito improvável que suas queries referenciem apenas uma tabela***
  - ***Na maioria das vezes, será necessário retornar dados de várias tabelas***
  - ***Para isso, utilizamos o operador de JOIN (INNER, LEFT OUTER e RIGHT OUTER)***

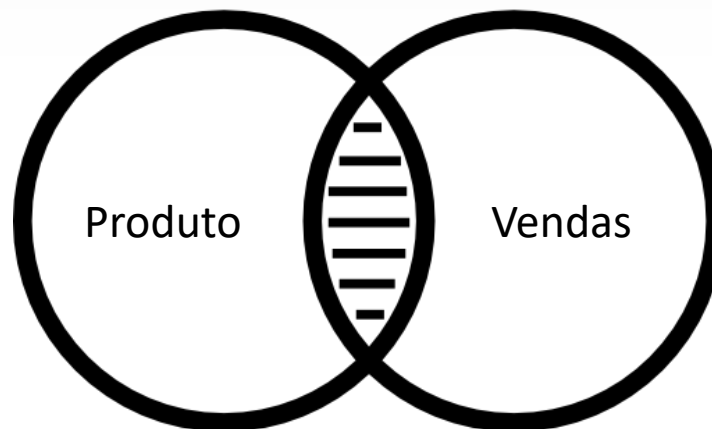


- **INNER JOIN**

- *Configura uma correspondência de igualdade entre duas ou mais tabelas*

- **Exemplo:**

- *Temos uma tabela contendo **produtos** e outra contendo **vendas**, e desejamos localizar somente os produtos que foram vendidos (comercializados)*

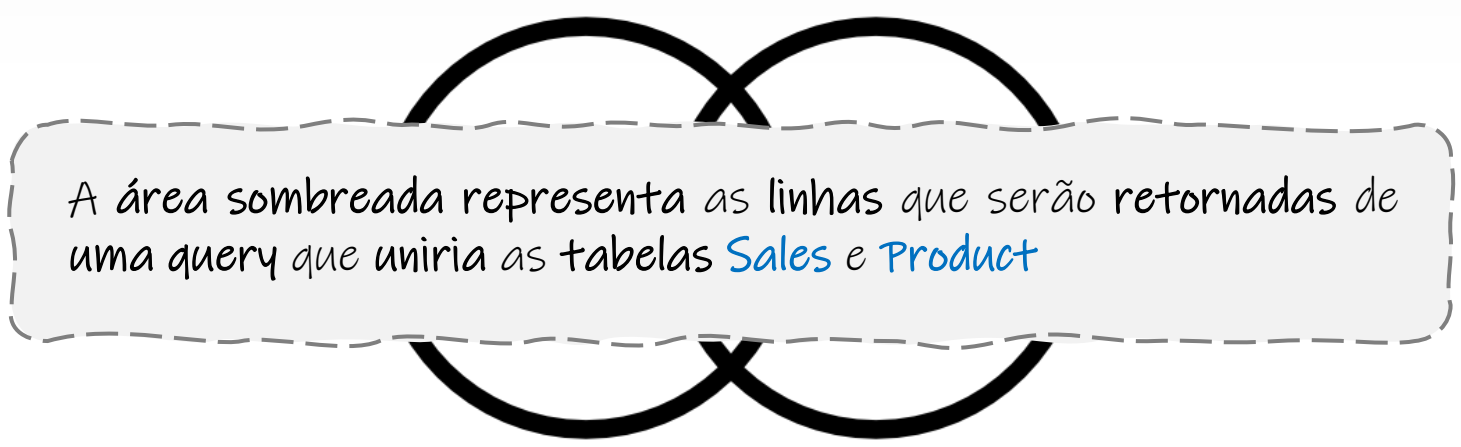


- **INNER JOIN**

- **Configura uma correspondência de igualdade entre duas ou mais tabelas**

- **Exemplo:**

- **Temos uma tabela contendo *produtos* e outra contendo *vendas*, e desejamos localizar somente os produtos que foram vendidos (comercializados)**



A área sombreada representa as linhas que serão retornadas de uma query que uniria as tabelas *Sales* e *Product*

- **INNER JOIN**

- **Exemplo (22):**

- *Independentemente de estar escrevendo uma junção INNER ou uma junção OUTER, iniciaremos com uma instrução SELECT básica*

```
USE MyAdventureWorks;
```

```
SELECT
```

```
 p.FirstName,
```

```
 p.LastName
```

```
FROM Person.Person p;
```

- **INNER JOIN**

- **Exemplo (23):**

- A **tabela** na **cláusula FROM** deve **incluir uma coluna com valores que existem na tabela que pretendemos fazer a junção**
    - Nesse caso, **queremos incluir um endereço de e-mail na tabela resultante** (*devemos referenciar uma segunda tabela na query*)

```
USE MyAdventureWorks;
```

```
SELECT
```

```
 p.FirstName,
 p.LastName,
 ea.EmailAddress
```

```
FROM Person.Person p
```

```
INNER JOIN Person.EmailAddress ea
```

```
 ON(p.BusinessEntityID = ea.BusinessEntityID);
```

- **INNER JOIN**

- **Exemplo (24):**

- Recuperar o **ProductID**, **Name** (**Product**) e **OrderQty** e **UnitPrice** (**SalesOrderDetail**)

```
USE MyAdventureWorks;
```

```
SELECT
```

```
 p.ProductID "Código do Produto",
 p.Name "Nome do Produto",
 sd.OrderQty "Quantidade do Pedido",
 sd.UnitPrice "Preço Unitário"
```

```
FROM Production.Product p
```

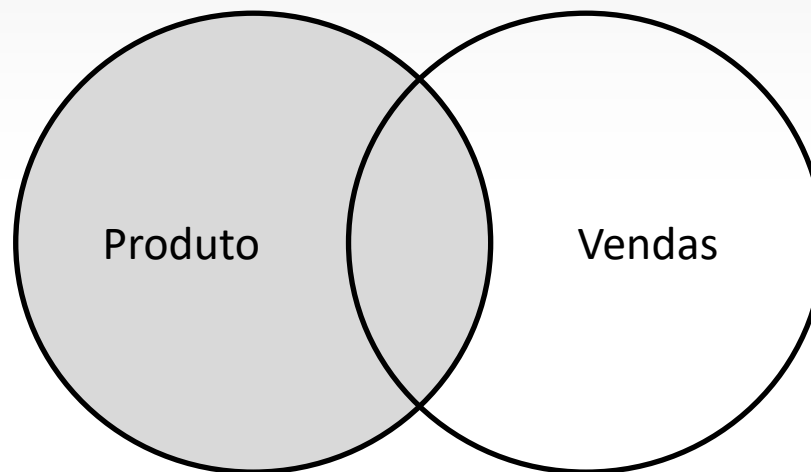
```
INNER JOIN Sales.SalesOrderDetail sd
 ON(p.ProductID = sd.ProductID);
```

- **OUTER JOINS**

- *Existem dois tipos de **OUTER JOINS** (**LEFT** e **RIGHT**)*
- *Ambos oferecem funcionalidade muito similar, mas há **uma pequena diferença** que depende da ordem das tabelas na query*
- *Usando o exemplo anterior de **vendas** de **produto**, se começarmos a ler a query da **esquerda** para a **direita**, qual tabela encontraremos primeiro?*
- *A tabela **Production.Product**, que o torna a **tabela** da **esquerda**. A segunda tabela que encontramos (**ainda lendo da esquerda para a direita**) é **Sales.SalesOrderDetail**, a **tabela da direita***

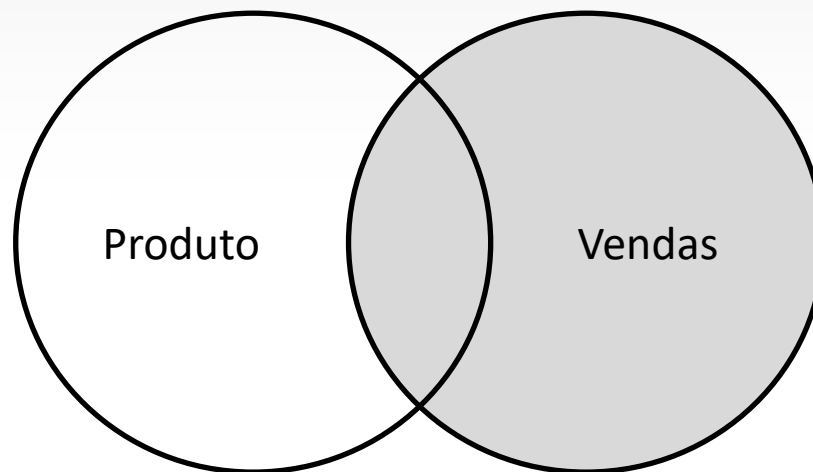
- **OUTER JOINS**

- Se desejarmos recuperar uma lista de produtos **independentemente** de sua **existência** na **tabela** **Sales.SalesDetail**, **LEFT OUTER JOIN** deverá ser nossa **escolha**



- **OUTER JOINS**

- *Se estivermos tentando recuperar todas as vendas, que estejam ou **não** associadas a um produto, devemos implementar uma **RIGHT OUTER JOIN***





- ***OUTER JOINS***

- ***Exemplo (25):***

- ***Recuperar todos os produtos (*Product*), independentemente se os produtos possuem ou não correlação com vendas (*SalesOrderDetail*)***

```
USE MyAdventureWorks;
```

```
SELECT
```

```
 p.ProductID,
 p.Name,
 sd.OrderQty,
 sd.UnitPrice
```

```
FROM Production.Product p
```

```
LEFT OUTER JOIN Sales.SalesOrderDetail sd
 ON(p.ProductID = sd.ProductID);
```

- ***Limitando os Dados***

- ***Além de usar uma cláusula **WHERE** em nossa query, existem várias outras maneiras de limitar os dados retornados em nosso conjunto de resultados***
- ***Embora haja uma longa lista de métodos e técnicas que podemos utilizar, o SQL Server apresenta palavras-chave que oferecem uma estratégia muito simples para limitar nosso conjunto de resultados***

- **TOP**

- *Limita o número de linhas retornadas em um resultado a uma quantidade específica ou a determinada porcentagem de linhas*
- *TOP sempre deve ser usada com a cláusula ORDER BY*
- *Na maioria das vezes, estaremos procurando o maior ou o menor conjunto de valores para determinada coluna, e classificar os dados fornecerá essa informação*

- **TOP**

- **Exemplo (26):**

- *Desejamos retornar as cinco maiores vendas da tabela **Sales**, adicionamos **TOP(5)** imediatamente após a palavra-chave **SELECT***
    - *Incluimos também uma **cláusula ORDER BY** especificando como **coluna de ordenação** aquela que **continha** o **valor de venda real** para **cada linha***

```
USE MyAdventureWorks;
```

```
SELECT TOP(5)
 SalesOrderID,
 OrderDate,
 SalesOrderNumber,
 TotalDue
FROM Sales.SalesOrderHeader
ORDER BY
 TotalDue DESC;
```

|   | SalesOrderID | OrderDate               | SalesOrderNumber | TotalDue    |
|---|--------------|-------------------------|------------------|-------------|
| 1 | 51131        | 2007-07-01 00:00:00.000 | SO51131          | 187487,825  |
| 2 | 55282        | 2007-10-01 00:00:00.000 | SO55282          | 182018,6272 |
| 3 | 46616        | 2006-07-01 00:00:00.000 | SO46616          | 170512,6689 |
| 4 | 46981        | 2006-08-01 00:00:00.000 | SO46981          | 166537,0808 |
| 5 | 47395        | 2006-09-01 00:00:00.000 | SO47395          | 165028,7482 |

# DISTINCT e NULL

- ***DISTINCT***

- *Retorna uma lista de valores únicos ou distintos de cada coluna especificada em uma instrução SELECT*
- *Se houver valores duplicados na lista, serão removidos todos eles, exceto um*
- *Exemplo (27):*
  - *Retornando uma lista de produtos com alguns dos nomes de produto repetidos várias vezes*

```
USE MyAdventureWorks;
```

```
SELECT
```

```
 p.Name "Nome do Produto"
```

```
FROM Production.Product p
```

```
INNER JOIN Sales.SalesOrderDetail sd
```

```
 ON(p.ProductID = sd.ProductID);
```

# DISTINCT e NULL

- **NULL**

- *É um valor especial*
- *Na verdade, ele **não** é um valor, ou seja, é a ausência de um valor*
- *Como resultado, existem valores de comparação especiais que podem ser utilizados ao referenciar NULL em uma cláusula WHERE*
- *Se estivesse procurando valores NULL, você usaria a seguinte sintaxe*

**WHERE <nome da coluna> IS NULL**

Se tivesse pesquisando colunas NOT NULL:

**WHERE <nome da coluna> IS NOT NULL**

# DISTINCT e NULL

- **NULL**

- **Exemplo (28):**

- *Procurar todos os produtos que **não** foram despachados, ou seja, todas as linhas no conjunto de resultados que tenham o valor **NULL** para a coluna **CarrierTrackingNumber** de **SalesOrderDetail***
    - *Na sequência, adicionamos a palavra-chave **DISTINCT** entre a palavra-chave **SELECT** e o nome da coluna **Name** (**Product**)*

```
USE MyAdventureWorks;

SELECT DISTINCT
 p.Name "Nome do Produto"
FROM Production.Product p
INNER JOIN Sales.SalesOrderDetail sd
 ON(p.ProductID = sd.ProductID)
WHERE sd.CarrierTrackingNumber IS NULL
ORDER BY
 "Nome do Produto";
```

- ***Combinando Conjuntos de Resultados***
  - ***Muitas vezes, teremos duas instruções SELECT que precisam ser combinadas em um resultado para consumo de um aplicativo ou usuário final***
  - ***O operador UNION tem duas variações:***
    - ***UNION: remove qualquer linha duplicada no conjunto de resultados***
    - ***UNION ALL: inclui registros duplicado. Se registros duplicados são possíveis, use UNION ALL; ela é muito mais performática, porque **não** precisa incluir DISTINCT***



- ***Combinando Conjuntos de Resultados***

- ***Pseudocódigo ilustra o uso do UNION:***

```
SELECT coluna1, coluna2, coluna3
FROM tabela1
UNION
SELECT coluna1, coluna2, coluna3
FROM tabela2
```

- Ao se ***escrever*** uma ***query*** com **UNION**, as ***duas instruções SELECT*** devem conter o mesmo número de colunas e tipos de dados devem corresponder para cada coluna
  - Ao ***usar*** **UNION**, forneça apenas uma cláusula **ORDER BY** após a última instrução **SELECT**

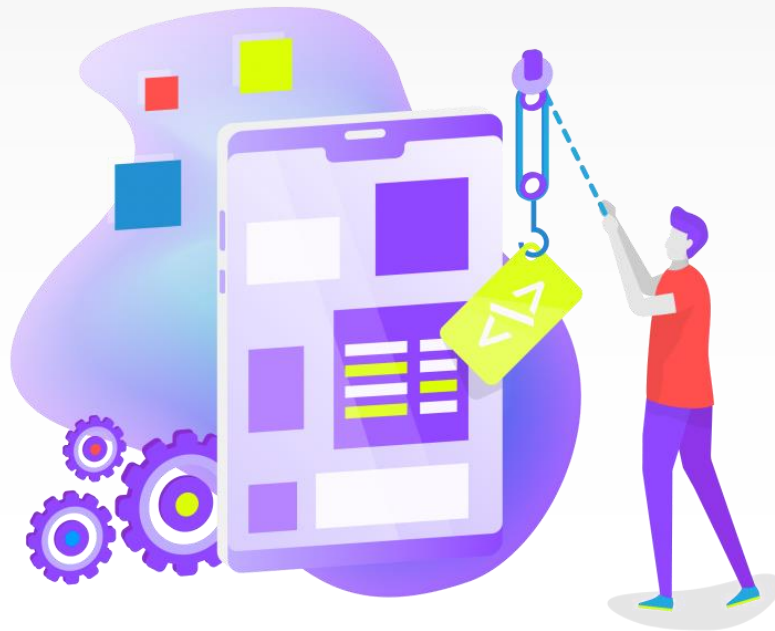
- ***Combinando Conjuntos de Resultados***

- ***Exemplo (29):***

- ***A query a seguir recupera uma lista de produtos Black e Silver***

```
USE MyAdventureWorks;

SELECT
 Name ProductName
FROM Production.Product
WHERE
 Color = 'Black'
UNION
SELECT
 Name ProductName
FROM Production.Product
WHERE
 Color = 'Silver';
```



# EXERCÍCIOS

# Referências

Noble, E.; Pro T-SQL 2019 Toward Speed, Scalability, and Standardization for SQL Server Developers. Apress, 2020.

Ben-Gan, I.; Microsoft SQL Server 2012 T-SQL Fundamentals. Pearson Education. 2012.

Lahoud, P.; Lopes, P.; T-SQL Querying: A guide to developing efficient and elegant T-SQL code. Packt Publishing. 2019.

