Faculdade IMPACTA TECNOLOGIA

Linguagem SQL / Banco de Dados

Aula 08 – Introdução à recuperação de dados:

DATA QUERY LANGUAGE (DQL)

Gustavo Bianchi Maia

gustavo.maia@faculddeimpacta.com

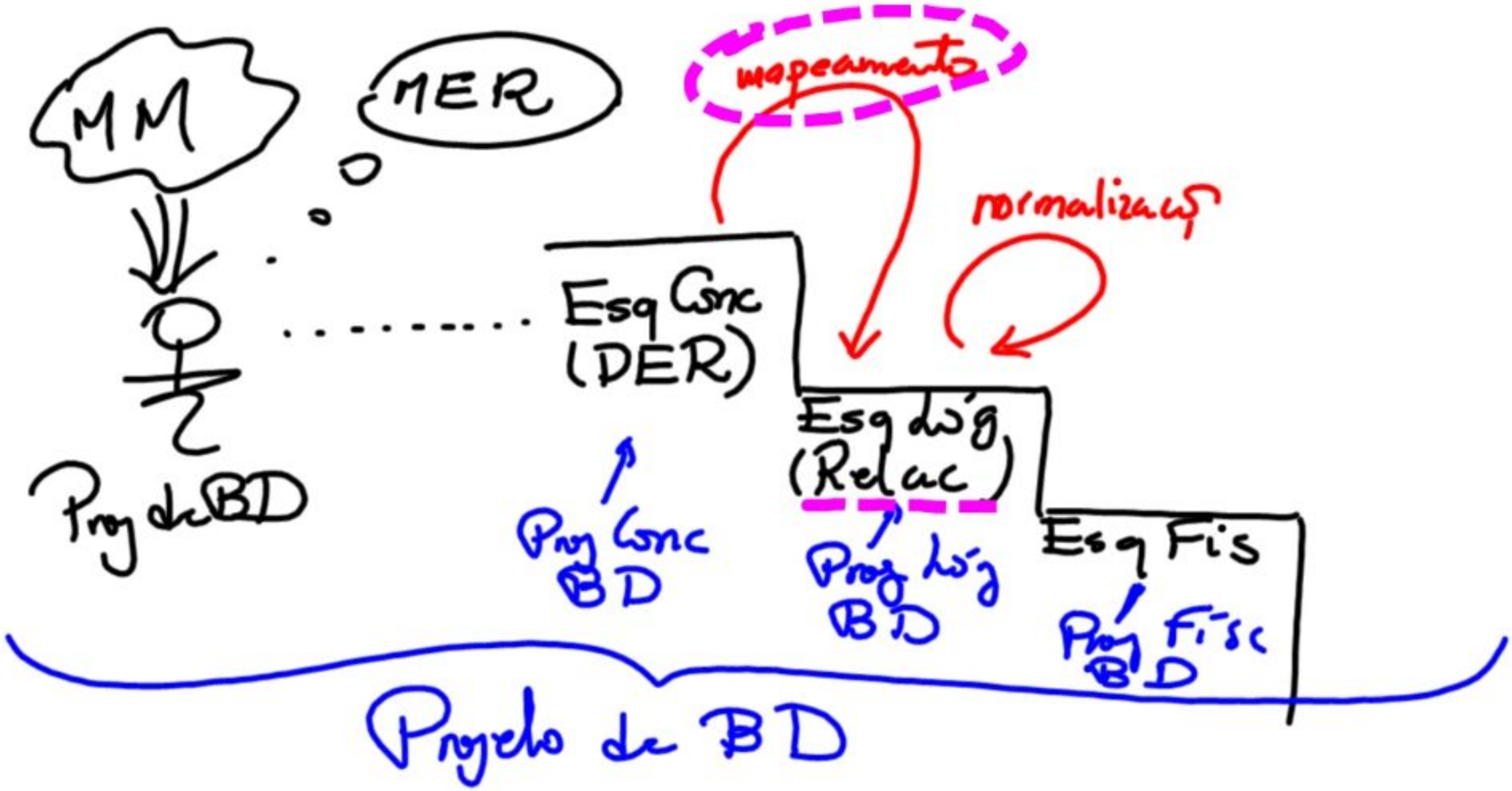
Faculdade IMPACTA TECNOLOGIA

Agenda

- Revisão DDL (modelo físico)
- Revisão DML
- Sub-Linguagem DQL
  - SELECT
  - FROM / JOINS
  - WHERE
  - GROUP BY
- Exercícios

Faculdade IMPACTA TECNOLOGIA

Modelo de Dados Relacional



Tipos de Dados

Tipos de dados determinam quais os tipos de valores serão permitidos no armazenamento e os principais tipos são agrupados em categorias conforme mostrado abaixo:

Categorias dos Tipos de Dados	
Numéricos Exatos	Caractere Unicode
Numéricos Aproximados	Binários
Data e Hora	Outros Tipos
Strings de Caractere	

Data Definition Language

- Fator de Nulidade (NULL ou NOT NULL)
- Auto-preenchimento (valores auto-incrementais): IDENTITY (1,1)
- Criação da tabela

```
CREATE TABLE <nome da tabela>
        (
        <nome coluna 1> <tipo da coluna> (<tamanho da coluna>) [NOT NULL]  ,  ...
        );
```

- Regras:
  - Primary Key

    CONSTRAINT <nome da primary key> PRIMARY KEY (coluna1, coluna2, ...)
  - Foreign Key

```
CONSTRAINT <nome da foreign key> FOREIGN KEY (coluna1, coluna2, ...)
REFERENCES <tabela da primary key> (coluna1, coluna2, ...)
```

  - Unique

    CONSTRAINT <nome da unique key> UNIQUE (coluna1, coluna2, ...)
  - Check

    CONSTRAINT <nome da regra> CHECK (<coluna com expressão booleana>)
  - Default

```
<nome da coluna> <tipo de dados> CONSTRAINT <nome do default>  DEFAULT ( <valor, texto, data, função escalar> )
```

Data Definition Language

```
CREATE TABLE Aluno
    (
    Matricula int not null IDENTITY (500, 1)
    , Nome varchar(20)
    , CONSTRAINT pkAluno
      PRIMARY KEY (Matricula)
    );
```

```
CREATE TABLE Prova
    (
    idProva int NOT NULL  IDENTITY (1, 1)
    , Matricula int  NOT NULL
    , Nota decimal(4,2) NOT NULL
    , CONSTRAINT pkProva PRIMARY KEY (idProva)
    , CONSTRAINT fkProva FOREIGN KEY (Matricula)
      REFERENCES Aluno(Matricula)
    );
```

Matricula	Nome
500	José
501	Pedro
502	Mario

idProva	Matricula	Nota
1	500	9
2	500	8
3	502	7
4	502	3
5	502	1

Data Modification Language

- Insert

  INSERT [INTO] table_or_view [(column_list)] data_values

- Delete

  DELETE table_or_view FROM table_sources WHERE search_condition

  Truncate Table table_or_view

- Update

  UPDATE table_or_view SET column_name = expression FROM table_sources WHERE search_condition

Data Manipulation Language Exemplos

Insert into Aluno (Nome) VALUES (‘Matheus’)
Insert into Prova (Matricula, Nota) VALUES (503, 10)

Delete from prova where Matricula = 500
Delete from aluno where Matricula = 500

- Alterar a matricula da Prova, de 502 para 504

Insert into Aluno (Nome) VALUES (‘Felipe’)
Update prova set Matricula = 504 where matricula = 502

Matricula	Nome
500	José
501	Pedro
502	Mario
503	Matheus
504	Felipe

idProva	Matricula	Nota
1	500	9
2	500	8
3	504	7
4	504	3
5	504	1
6	503	10

Data Query Language - DQL

Categoria de subcomando da linguagem SQL que envolve a declaração de recuperação de dados (SELECT).

SELECT é uma declaração SQL que retorna um conjunto de resultados de registros de uma ou mais tabelas. Ela recupera zero ou mais linhas de uma ou mais tabelas-base, tabelas temporárias, funções ou visões em um banco de dados. Também retorna valores únicos de configurações do sistema de banco de dados ou de variáveis de usuários ou do sistema.

Na maioria das aplicações, SELECT é o comando mais utilizado. Como SQL é uma linguagem não procedural, consultas SELECT especificam um conjunto de resultados, mas não especificam como calculá-los, ou seja, a consulta em um "plano de consulta" é deixada para o sistema de banco de dados, mais especificamente para o otimizador de consulta, escolher a melhor maneira de retorno das informações que foram solicitadas.

Data Query Language - DQL

Existem vários elementos na declaração SELECT, mas os principais são:

Elemento	Expressão	Descrição
SELECT	<lista de seleção>	Define quais as colunas que serão retornadas
FROM	<tabela de origem>	Define a(s) tabela(s) envolvidas na consulta
WHERE	<condição de pesquisa>	Filtra as linhas requeridas
GROUP BY	<agrupar a seleção>	Agrupa a lista requerida (utiliza colunas)
HAVING	<condição de agrupamento>	Filtra as linhas requeridas, pelo agrupamento
ORDER BY	<ordem da lista>	Ordena o retorno da lista

Data Query Language - DQL

A ordem como a consulta (query) é escrita, não significa que será a mesma ordem que o banco de dados utilizará para executar o processamento:

5: SELECT <select list>

1: FROM <table source>

2: WHERE <search condition>

3: GROUP BY <group by list>

4: HAVING <search condition>

6: ORDER BY <order by list>

Data Query Language - DQL

A forma mais simples da declaração SELECT é a utilização junto ao elemento FROM, conforme mostrado abaixo.

Note que no <select list> faz uma filtragem vertical, ou seja, retorna uma ou mais colunas de tabelas, mencionadas pela cláusula FROM.

Elemento	Expressão
SELECT	**<select list>**
FROM	**<table source>**

```
SELECT Nome, Sobrenome
FROM   Cliente;
```

Data Query Language - DQL

Outros exemplos para SELECT simples

(*) - Retorna todas as colunas da tabela *exemploSQL*

```
SELECT  * FROM exemploSQL
```

(coluna) - Retorna a coluna *texto_curto_naonulo* da tabela *exemploSQL*

```
SELECT  texto_curto_naonulo FROM exemploSQL
```

(coluna 1, coluna 2, ...) - Retorna as colunas texto_curto_naonulo e numero_check da tabela exemploSQL

```
SELECT  texto_curto_naonulo, numero_check FROM exemploSQL
```

Data Query Language - DQL

Podemos fazer utilização de diversos operadores matemáticos para cálculo de valores, abaixo mostramos os principais operadores.

Operator	Description
+	Add or concatenate
-	Subtract
*	Multiply
/	Divide
%	Modulo

```
SELECT preco, qtd, (preco * qtd)
FROM   DetalhesDoPedido;
```

OBS: Operadores possuem precedência entre si.

Data Query Language - DQL

Exemplos para SELECT simples e operadores

Retorna o resultado das operações abaixo

```
SELECT  20 + 20 / 5 FROM exemploSQL
SELECT  (20 + 20) / 5 FROM exemploSQL
SELECT  20 + (20 / 5) FROM exemploSQL

SELECT  ( (10+2) / 2 ) * 0.3 ) % 2

SELECT  Nome, Salario * 1.07 FROM Funcionario
```

Nota: O operador + se transforma em concatenador quando lidamos com string:

```
SELECT  ‘Hoje’ + ‘ ‘ + ‘é’ + ‘ terça-feira ’ + ‘ou’ + ‘ quinta-feira ’
```

Data Query Language - DQL

Pode ser necessário darmos apelidos (Aliases) a colunas para facilitar o entendimento no retorno dos dados:

- **Apelidos na coluna utilizando a cláusula AS**

```
SELECT idPedido, preco, qtd AS Quantidade
FROM   DetalhesDoPedido;
```

- **Também é possível realizar a mesma operação com =**

```
SELECT idPedido, preco, Quantidade = qtd
FROM   DetalhesDoPedido;
```

- **Ou mesmo sem a necessidade do AS**

```
SELECT idPedido, preco ValorProduto
FROM   DetalhesDoPedido;
```

Data Query Language - DQL

Também pode ser necessário darmos apelidos em tabelas, principalmente quando formos realizar joins:

- **Apelidos em tabelas com a cláusula AS**

```
SELECT  idPedido, dataPedido
FROM     Pedido AS SO;
```

- **Table aliases without AS**

```
SELECT  idPedido, dataPedido
FROM     Pedido  SO;
```

- **Usando os apelidos no SELECT**

```
SELECT  SO.idPedido, SO.dataPedido
FROM     Pedido AS SO;
```

Data Query Language - DQL

Veja os resultados com linhas repetidas na consulta abaixo:

```
SELECT pais
FROM   Cliente;
```

```
pais
--------------
Argentina
Argentina
Austria
Austria
Belgium
Belgium
```

Data Query Language - DQL

Podemos eliminar as linhas repetidas aplicando a cláusula DISTINCT:

```
SELECT DISTINCT <column list>

FROM <table or view>
```

```
SELECT DISTINCT pais
FROM   Cliente;
```

```
pais
-------------
Argentina
Austria
Belgium
```

Data Query Language - DQL

A cláusula DISTINCT, retira repetições de linhas para todas as colunas descritas na declaração SELECT:

```
SELECT DISTINCT empresa, pais
FROM Cliente;
```

```
Empresa       pais
------------- -------
Empresa  AHPOP UK
Empresa  AHXHT Mexico
Empresa  AZJED Germany
Empresa  BSVAR France
Empresa  CCFIZ Poland
```

Data Query Language - DQL

Muitas vezes queremos visualizar apenas o retorno de algumas linhas e não necessariamente todos os registros de uma tabela. Podemos utilizar a cláusula TOP para isso:

TOP (N) / TOP (N) PERCENT

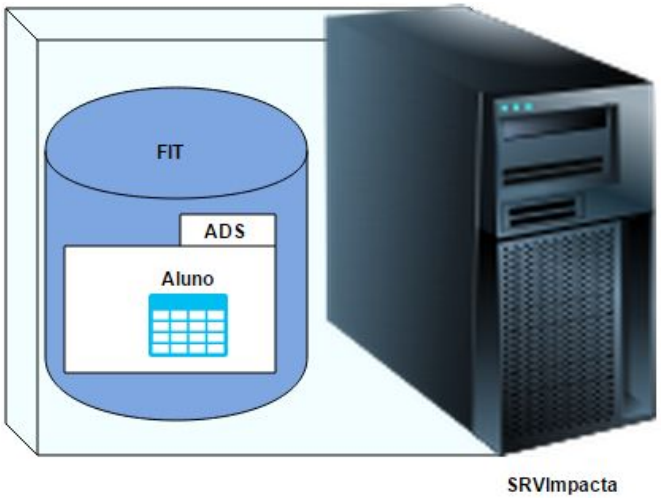
Retorna uma certa quantidade de linhas (ou percentual de linhas) definido.

```
SELECT top 10 * FROM exemploSQL
SELECT top 10 percent * FROM exemploSQL
```

DQL – Four Part Naming

Objetos (tables, views, functions, ...) no banco de dados possuem seu nome formado por 4 partes:

SELECT * FROM Server . Database . Schema . Object



SELECT * FROM SRVImpacta . FIT . ADS. Aluno

DQL – Schema

O Schema é como se fosse um repositório onde colocamos objetos como tabela, visão, função, procedimento, ... Todo objeto possui um schema e quando não escolhemos algum, ele se encarrega de colocar o schema DBO como padrão (*default*).

Quando não escrevemos explicitamente os 4 nomes, o banco tenta preenche-los automaticamente:

```
SELECT * FROM ALUNO
```

Na consulta acima o banco de dados tenta encontrar o objeto Aluno. Como não foi fornecido o SERVER, ele usará o servidor a que está conectado no momento. O mesmo procedimento é feito para o DATABASE, como não foi fornecido, tenta utilizar a base que está conectada. Para o SCHEMA, se não foi mencionado, tentará o DBO.

O SELECT só irá funcionar se com todos os *defaults* tentados pelo banco, contenham o objeto ALUNO.

DQL – Cláusula WHERE

A cláusula WHERE faz o filtro horizontal em uma consulta, ou seja, permite uma redução do número de linhas que retornarão na consulta.

Elemento	Expressão	Descrição
SELECT	<lista de seleção>	Define quais as colunas que serão retornadas
FROM	<tabela de origem>	Define a(s) tabela(s) envolvidas na consulta
WHERE	**<condição de pesquisa>**	**Filtra as linhas requeridas**
GROUP BY	<agrupar a seleção>	Agrupa a lista requerida (utiliza colunas)
HAVING	<condição de agrupamento>	Filtra as linhas requeridas, pelo agrupamento
ORDER BY	<ordem da lista>	Ordena o retorno da lista

DQL – Cláusula WHERE

Operadores são utilizados para avaliar uma ou mais expressões que retornam os valores possíveis: TRUE, FALSE ou UNKNOWN.

O retorno de dados se dará em todas as tuplas onde a combinação das expressões retornarem TRUE.

Operadores de Comparação Escalar

=, <>, >, >=, <, <=, !=

```
SELECT FirstName, LastName, MiddleName
FROM Person.Person
WHERE ModifiedDate >= '20040101'
```

FirstName	LastName	MiddleName
Ken	Sánchez	J
Terri	Duffy	Lee
Roberto	Tamburello	NULL
Rob	Walters	NULL
Gail	Erickson	A

DQL – Cláusula WHERE

- **Operadores Lógicos são usados para combinar condições na declaração**

Retorna somente registros onde o primeiro nome for 'John' **E** o sobrenome for 'Smith'

```
WHERE FirstName = 'John' AND LastName = 'Smith'
```

Retorna todos as linhas onde o primeiro nome for 'John' **OU** todos onde o sobrenome for 'Smith'

```
WHERE FirstName = 'John' OR LastName = 'Smith'
```

Retorna todos as tuplas onde o primeiro nome for 'John' e o sobrenome **NÃO** for 'Smith'

```
WHERE FirstName = 'John' AND NOT LastName = 'Smith'
```

DQL – Cláusula WHERE

- Cláusula WHERE Simples

```
SELECT BusinessEntityID AS 'Employee Identification Number', HireDate,
       VacationHours, SickLeaveHours
FROM   HumanResources.Employee
WHERE BusinessEntityID <= 1000
```

```
SELECT FirstName, LastName, Phone
FROM   Person.Person
WHERE FirstName = 'John',
```

DQL – Cláusula WHERE

- Cláusula WHERE usando Predicado

Nem sempre usamos operadores de comparação. Em algumas situações podemos usar outros operadores que são chamados de predicados, simplificando a escrita do script.

Alguns exemplos de Predicado são: IN, BETWEEN, ANY, SOME, IS, ALL, OR, AND, NOT, EXISTS ...

```
SELECT FirstName, LastName, Phone
FROM Person.Person
WHERE EmailAddress IS NULL;
```

DQL – Cláusula WHERE

- **BETWEEN** – restringe dados através de uma faixa de valores possíveis.

```
SELECT OrderDate, AccountNumber, SubTotal, TaxAmt
FROM    Sales.SalesOrderHeader
WHERE OrderDate BETWEEN ’20110801’ AND ’ 20110831’
```

- **BETWEEN** – A mesma lógia do uso de >= AND <=

```
SELECT OrderDate, AccountNumber, SubTotal, TaxAmt
FROM    Sales.SalesOrderHeader
WHERE OrderDate >= ’ 20110801’
          AND OrderDate <= ’ 20110831’
```

OrderDate	AccountNumber	SubTotal	TaxAmt
2001-08-01 00:00:00.000	10-4020-000018	39677.4848	3174.1988
2001-08-01 00:00:00.000	10-4020-000353	24299.928	1943.9942
2001-08-01 00:00:00.000	10-4020-000206	10295.8366	823.6669
2001-08-01 00:00:00.000	10-4020-000318	1133.2967	90.6637
2001-08-01 00:00:00.000	10-4020-000210	1086.6152	86.9292
2001-08-01 00:00:00.000	10-4020-000164	21923.9352	1753.9148
2001-08-01 00:00:00.000	10-4020-000697	24624.706	1969.9765
2001-08-01 00:00:00.000	10-4020-000191	12286.7218	982.9377

DQL – Cláusula WHERE

- **IN** – fornece uma lista de possibilidades de valores que poderiam atender a consulta.

```
SELECT SalesOrderID, OrderQty, ProductID, UnitPrice
FROM Sales.SalesOrderDetail
WHERE ProductID IN (750, 753, 765, 770)
```

- **IN** – Usa a mesma lógica de múltiplas comparações com o predicado OR entre elas.

```
SELECT SalesOrderID, OrderQty, ProductID, UnitPrice
FROM Sales.SalesOrderDetail
WHERE ProductID = 750 OR ProductID = 753
      OR ProductID = 765 OR ProductID = 770
```

SalesOrderID	OrderQty	ProductID	UnitPrice
43662	5	770	419.4589
43662	3	765	419.4589
43662	1	753	2146.962
43666	1	753	2146.962
43668	2	753	2146.962
43668	6	765	419.4589
43668	2	770	419.4589
43671	1	753	2146.962
43673	2	770	419.4589

DQL – Cláusula WHERE

- **LIKE** – Permite consultas mais refinadas em colunas do tipo string (CHAR, VARCHAR, …).

```
WHERE LastName = 'Johnson'
```

```
WHERE LastName LIKE 'Johns%n'
```

FirstName	LastName	MiddleName
Abigail	Johnson	NULL
Alexander	Johnson	M
Alexandra	Johnson	J
Alexis	Johnson	J
Alyssa	Johnson	K
Andrew	Johnson	F
Anna	Johnson	NULL

FirstName	LastName	MiddleName
Meredith	Johnsen	NULL
Rebekah	Johnsen	J
Ross	Johnsen	NULL
Willie	Johnsen	NULL
Abigail	Johnson	NULL
Alexander	Johnson	M
Alexandra	Johnson	J

DQL – Cláusula WHERE

- **LIKE** – Este predicado é usado para verificar padrões dentro de campos strings e utiliza símbolos, chamados de coringas, para permitir a busca desses padrões.

  - Símbolos (coringas)
    - **%** (Percent) representa qualquer string e qualquer quantidade de strings
    - **_** (Underscore) representa qualquer string, mas apenas uma string
    - [<List of characters>] representa possíveis caracteres que atendam a string procurada
    - [<Character> - <character>] representa a faixa de caracteres, em ordem alfabética, para a string procurada
    - [**^**<Character list or range>] representa o caractere que não queremos na pesquisa

```
SELECT categoryid, categoryname, description
FROM Production.Categories
WHERE description LIKE 'Sweet%'
```

DQL – Utilização do NULL

~~**NULL = 0 (zero)**~~

~~**NULL = '' (branco ou vazio)**~~

~~**NULL = 'NULL' (string NULL)**~~

~~**NULL = NULL**~~

DQL – Cláusula WHERE

- **NULL** – É ausência de valor ou valor desconhecido. Nenhuma das sentenças acima é verdadeira pois o banco de dados não pode comparar um valor desconhecido com outro valor que ele também não conhece.
- Para trabalhar com valores NULL, temos que utilizar os predicados IS NULL e IS NOT NULL.

```
SELECT custid, city, region, country
FROM Sales.Customers
WHERE region IS NOT NULL;
```

- Predicados retornam UNKNOWN quando comparados com valores desconhecidos (valores faltando), ou seja, não são retornados na consulta.

DQL – Cláusula ORDER BY

Conforme mencionado anteriormente, por padrão, não há garantia de ordenação no retorno dos dados de uma consulta.

Para garantir que o retorno da consulta tenha uma ordenação, utilizamos a cláusula ORDER BY.

Elemento	Expressão	Descrição
SELECT	<lista de seleção>	Define quais as colunas que serão retornadas
FROM	<tabela de origem>	Define a(s) tabela(s) envolvidas na consulta
WHERE	<condição de pesquisa>	Filtra as linhas requeridas
GROUP BY	<agrupar a seleção>	Agrupa a lista requerida (utiliza colunas)
HAVING	<condição de agrupamento>	Filtra as linhas requeridas, pelo agrupamento
ORDER BY	**<ordem da lista>**	**Ordena o retorno da lista**

As cláusulas ASC e DESC podem ser usadas após cada campo do commando ORDER BY. A ordenação ASCendente é a padrão quando não mencionamos explicitamente.

DQL – Cláusula ORDER BY

- ORDER BY com nome de colunas:

```
SELECT orderid, custid, orderdate
FROM Sales.Orders
ORDER BY orderdate;
```

- ORDER BY com apelido:

```
SELECT orderid, custid, YEAR(orderdate) AS orderyear
FROM Sales.Orders
ORDER BY orderyear;
```

- ORDER BY with descending order:

```
SELECT orderid, custid, orderdate
FROM Sales.Orders
ORDER BY orderdate DESC;
```

Faculdade IMPACTA TECNOLOGIA

Obrigado!

Aula Gravada por:

Prof. Msc. Gustavo Bianchi Maia

gustavo.maia@faculddeimpacta.com

Material criado e oferecido por :

Prof. Sand Jacques Onofre