Faculdade IMPACTA TECNOLOGIA

Linguagem SQL / Banco de Dados

Aulas 11 e 12 – Recuperação de dados - Group By:

DATA QUERY LANGUAGE (DQL)

Gustavo Bianchi Maia

gustavo.maia@faculddeimpacta.com

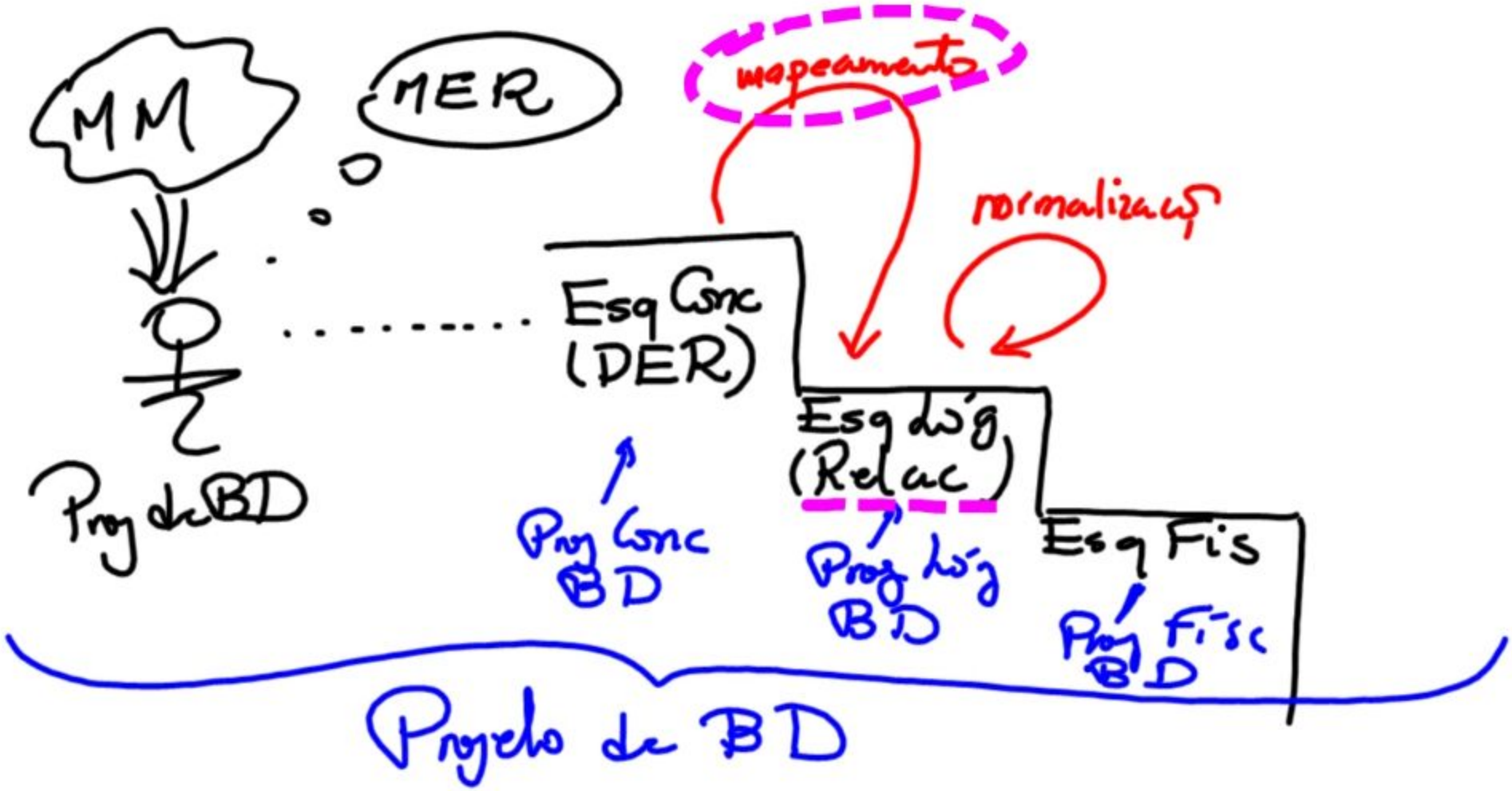
Faculdade IMPACTA TECNOLOGIA

Agenda

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

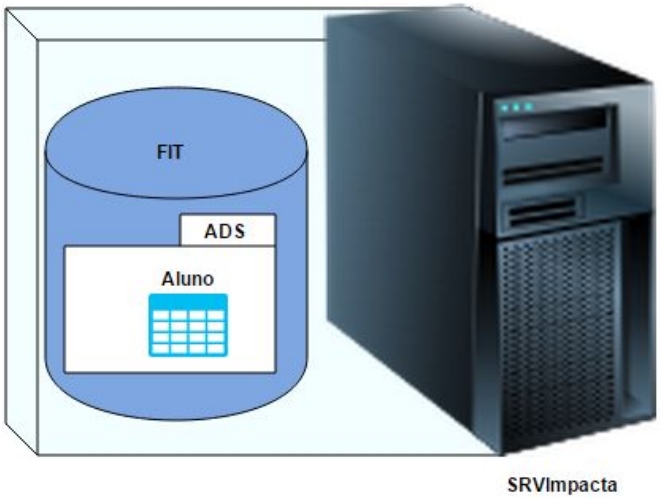
Retorna uma certa quantidade de linhas (ou percentual de linhas) definido.

```
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;
```

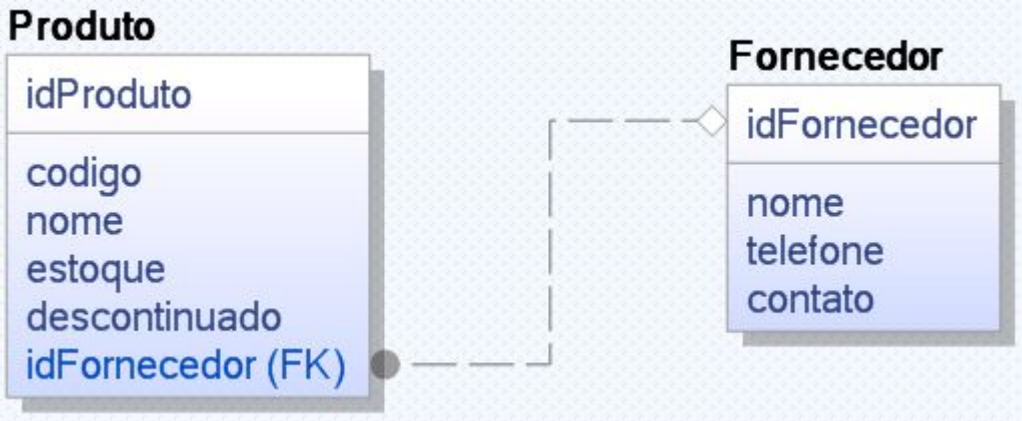
- ORDER BY with descending order:

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

Join

Em Bancos de Dados Relacionais, as entidades são projetadas para serem relacionadas umas com as outras. Tarefas de unir informações entre diversos objetos (*tables, views, functions,* ...) são muito comuns e necessárias. Estas junções precisam ser organizadas de forma a obtermos os dados necessários para apresentar aos usuários.

As junções entre as entidades são feitas através da relação de um ou mais atributos entre elas. Por exemplo, no DER abaixo a tabela Produto se relaciona com a tabela Fornecedor através das colunas idFornecedor.



Join

No relacionamento mostrado, dado um determinado produto, através do relacionamento entre a coluna idFornecedor, conseguimos mapear as informações do fornecedor daquele produto.

Para chegarmos a estas informações, escrevemos o SELECT da seguinte forma:

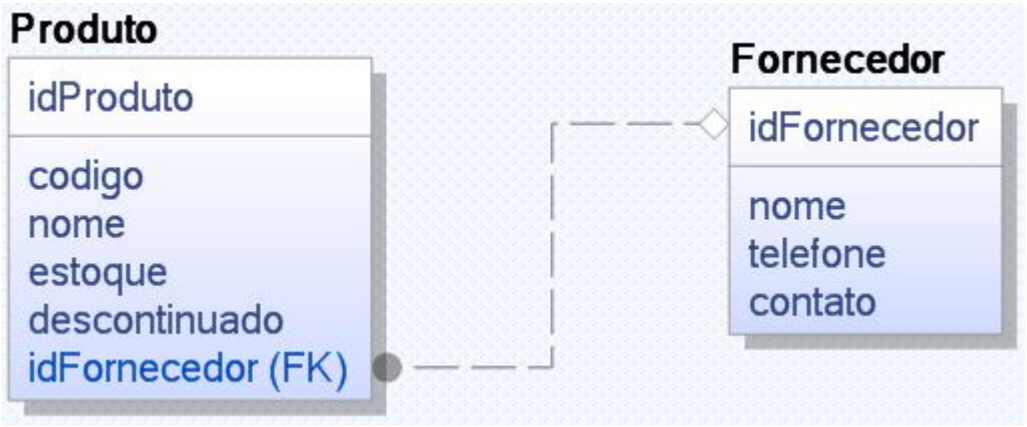
```
SELECT   <tabela 1>.<coluna 1>, ..., <tabela 1>.<coluna n>
        , <tabela 2>.<coluna 1>, ..., <tabela 2>.<coluna n>
FROM     <tabela 1>  JOIN   <tabela 2>
                    ON     <tabela 1>.<coluna chave> = <tabela 2>.<coluna chave>
```

A cláusula JOIN faz com que o banco de dados retorne informações das tabelas envolvidas, onde a expressão na cláusula ON for atendida, no caso acima, onde a coluna chave da tabela 1 seja igual a coluna chave da tabela 2.

Join

No exemplo Produto x Fornecedor, para trazer informações das duas tabelas podemos escrever a seguinte query:

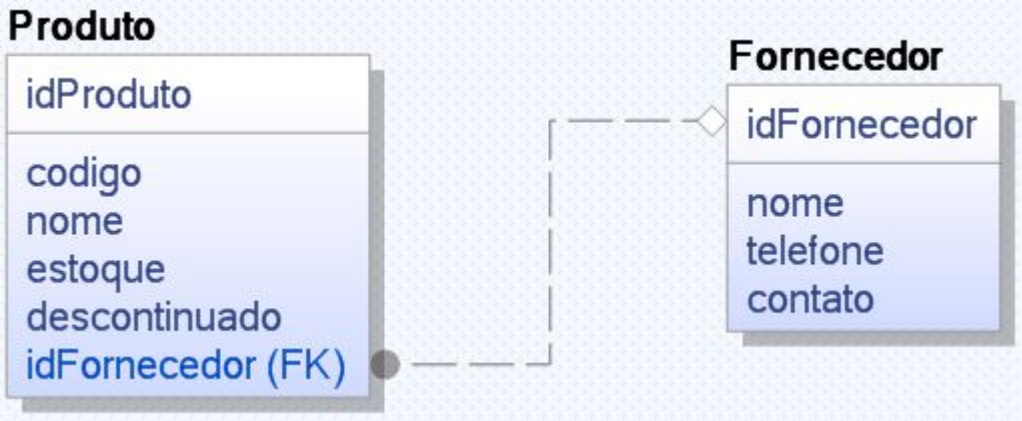
```
SELECT   Produto.Codigo, Produto.Nome, Produto.Estoque
       , Fornecedor.Nome, Fornecedor.Contato, Fornecedor.Telefone
FROM     Produto  JOIN   Fornecedor
                  ON     Produto.idFornecedor = Fornecedor.idFornecedor
```



Join

A mesma consulta poderia ser simplificada e/ou melhorada no seu entendimento adotando a utilização de apelidos para tabelas e colunas. O SELECT abaixo é exatamente o mesmo do slide anterior, utilizando *Aliases* para tabelas e colunas.

```
SELECT   P.Codigo, P.Nome AS 'Nome do Produto', P.Estoque
         , F.Nome AS 'Nome do Fornecedor', F.Contato, F.Telefone
FROM     Produto AS P JOIN   Fornecedor AS F
                     ON      P.idFornecedor = F.idFornecedor
```



Faculdade IMPACTA TECNOLOGIA

Join

Para o exemplo de dados abaixo:

Produto

idProduto	codigo	nome	estoque	descontinuado	idFornecedor
1	XT890A	Asus Zenfone	5	0	3
2	RQ765B	iPhone	0	1	9
3	WD528B	Moto X	3	0	2
4	TF897A	Xperia	7	0	1
5	RF212B	Moto Maxx	2	0	2

Fornecedor

idFornecedor	nome	telefone	contato
1	Sony	8498-8732	Allan
2	Motorola	7987-9900	Cristina
3	Asus	5476-1120	Felipe
4	Nokia	6755-5656	Fábio

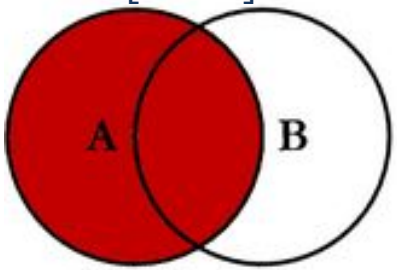
```
SELECT   P.Codigo, P.Nome AS 'Nome do Produto', P.Estoque
         , F.Nome AS 'Nome do Fornecedor', F.Contato, F.Telefone
FROM     Produto AS P JOIN   Fornecedor AS F
                ON     P.idFornecedor = F.idFornecedor
```

Codigo	Nome do Produto	Estoque	Nome do Fornecedor	Contato	Telefone
XT890A	Asus Zenfone	5	Asus	Felipe	5476-1120
WD528B	Moto X	3	Motorola	Cristina	7987-9900
TF897A	Xperia	7	Sony	Allan	8498-8732
RF212B	Moto Maxx	2	Motorola	Cristina	7987-9900

Tipos de Join

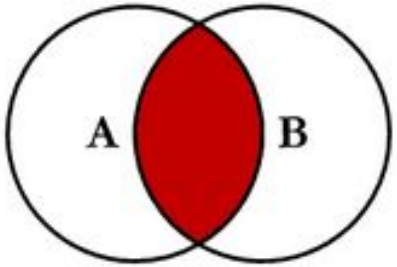
SQL JOINS

LEFT [outer] JOIN



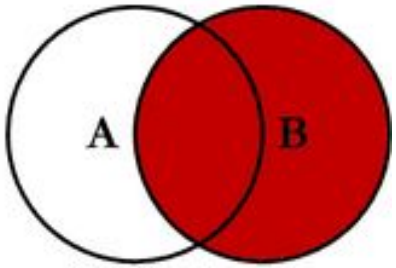

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

INNER JOIN



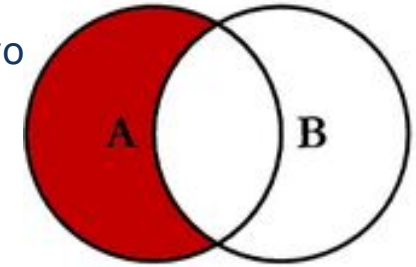

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

RIGHT [outer] JOIN



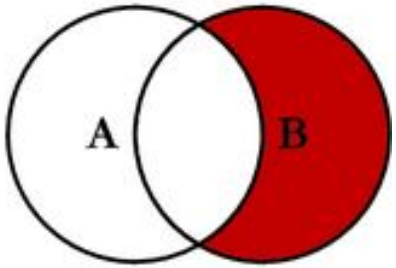

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```

LEFT Exclusivo ou LEFT + WHERE



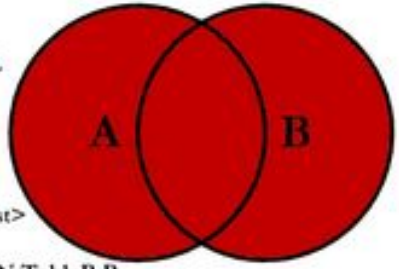

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

RIGHT Exclusivo ou RIGHT + WHERE



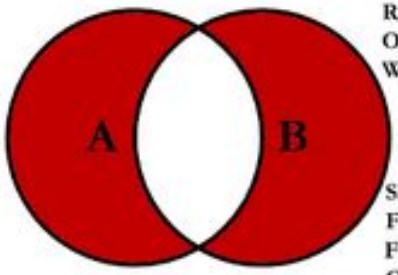

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

FULL [outer] JOIN




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

FULL Exclusivo ou FULL + WHERE




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

Left Join

Também podemos alterar as características destas junções para mostrarmos não somente os dados que são encontrados entre os relacionamentos, mas “forçar” que registros, mesmo não possuindo relacionamento, sejam apresentados mesmo assim.

Observando a mesma modelagem mostrada anteriormente, para gerar uma lista com TODOS os produtos, mesmo que não haja relação entre as duas tabelas, podemos substituir a cláusula JOIN por LEFT JOIN.

Ou seja, para gerar uma consulta que apresente OBRIGATORIAMENTE todos os produtos, mesmo aqueles onde não encontramos relação na tabela de fornecedores, poderíamos utilizar a cláusula LEFT JOIN, que faz com que todos os registros do objeto (tabela) do lado ESQUERDO da consulta seja mostrada.

Faculdade IMPACTA TECNOLOGIA

Left Join

Produto

idProduto	codigo	nome	estoque	descontinuado	idFornecedor
1	XT890A	Asus Zenfone	5	0	3
2	RQ765B	iPhone	0	1	9
3	WD528B	Moto X	3	0	2
4	TF897A	Xperia	7	0	1
5	RF212B	Moto Maxx	2	0	2

Fornecedor

idFornecedor	nome	telefone	contato
1	Sony	8498-8732	Allan
2	Motorola	7987-9900	Cristina
3	Asus	5476-1120	Felipe
4	Nokia	6755-5656	Fábio

```
SELECT   P.Codigo, P.Nome AS 'Nome do Produto', P.Estoque
         , F.Nome AS 'Nome do Fornecedor', F.Contato, F.Telefone
FROM     Produto AS P  LEFT JOIN  Fornecedor AS F
                  ON     P.idFornecedor = F.idFornecedor
```

Codigo	Nome do Produto	Estoque	Nome do Fornecedor	Contato	Telefone
XT890A	Asus Zenfone	5	Asus	Felipe	5476-1120
RQ765B	**iPhone**	**0**	**NULL**	**NULL**	**NULL**
WD528B	Moto X	3	Motorola	Cristina	7987-9900
TF897A	Xperia	7	Sony	Allan	8498-8732
RF212B	Moto Maxx	2	Motorola	Cristina	7987-9900

Right Join

Podemos fazer o inverso, ou seja, gerar uma lista com OBRIGATORIAMENTE todos os fornecedores, mesmo aqueles onde não encontramos relação na tabela de produtos, bastando alterar para cláusula RIGHT JOIN, que faz com que todos os registros do lado DIREITO da consulta seja mostrada.

O controle do modo que queremos o join é realizado apenas pelo posicionamento da tabela ao lado direito ou esquerdo do SELECT ou simplesmente trocando a cláusula de RIGHT JOIN para LEFT JOIN.

Por exemplo, sejam as tabelas TB1 e TB2, relacionadas pelas colunas C1 e C2, se quisermos gerar uma lista com TODOS os registros da tabela TB1, os SELECTs abaixo geram essa lista:

```
SELECT ... FROM  TB1  LEFT JOIN   TB2  ON  TB1.C1 = TB2.C2
```

OU

```
SELECT ... FROM TB2  RIGHT JOIN TB1 ON TB2.C2 = TB1.C1
```

Simples assim !

Right Join

Produto

idProduto	codigo	nome	estoque	descontinuado	idFornecedor
1	XT890A	Asus Zenfone	5	0	3
2	RQ765B	iPhone	0	1	9
3	WD528B	Moto X	3	0	2
4	TF897A	Xperia	7	0	1
5	RF212B	Moto Maxx	2	0	2

Fornecedor

idFornecedor	nome	telefone	contato
1	Sony	8498-8732	Allan
2	Motorola	7987-9900	Cristina
3	Asus	5476-1120	Felipe
4	Nokia	6755-5656	Fábio

```
SELECT   P.Codigo, P.Nome AS 'Nome do Produto', P.Estoque
         , F.Nome AS 'Nome do Fornecedor', F.Contato, F.Telefone
FROM     Produto AS P  RIGHT JOIN  Fornecedor AS F
                  ON     P.idFornecedor = F.idFornecedor
```

Codigo	Nome do Produto	Estoque	Nome do Fornecedor	Contato	Telefone
XT890A	Asus Zenfone	5	Asus	Felipe	5476-1120
WD528B	Moto X	3	Motorola	Cristina	7987-9900
TF897A	Xperia	7	Sony	Allan	8498-8732
RF212B	Moto Maxx	2	Motorola	Cristina	7987-9900
NULL	**NULL**	**NULL**	**Nokia**	**6755-565**	**Fábio**

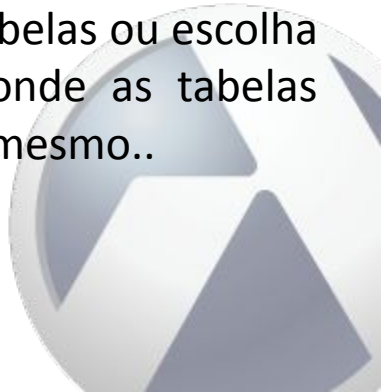
Faculdade IMPACTA TECNOLOGIA

Full Join

Existe um outro tipo de join que podemos utilizar quando queremos retornar todas as linhas das tabelas envolvidas, retornando todas as relações (como o JOIN) e todos os registros não relacionados (LEFT JOIN e RIGHT JOIN) numa mesma extração. Trata-se da cláusula FULL JOIN que em geral não é utilizada sistemicamente, mas sim para encontrarmos possíveis problemas nos e relacionamentos.

```
SELECT   <tabela 1>.<coluna 1>, ..., <tabela 1>.<coluna n>
         , <tabela 2>.<coluna 1>, ..., <tabela 2>.<coluna n>
FROM     <tabela 1>  FULL JOIN   <tabela 2>
                  ON     <tabela 1>.<coluna chave> = <tabela 2>.<coluna chave>
```

Diferentemente do LEFT JOIN ou RIGHT JOIN, onde o posicionamento das tabelas ou escolha da cláusula gera efeito na listagem, na cláusula FULL JOIN, a ordem onde as tabelas estiverem não fará diferença, ou seja, o resultado na geração da lista será o mesmo..



Faculdade IMPACTA TECNOLOGIA

Full Join

Produto

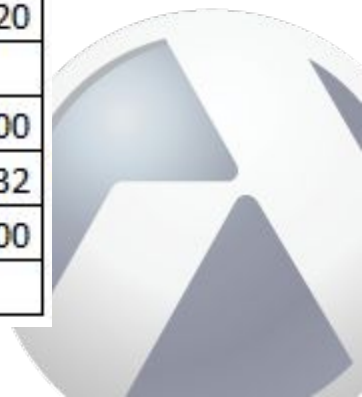
idProduto	codigo	nome	estoque	descontinuado	idFornecedor
1	XT890A	Asus Zenfone	5	0	3
2	RQ765B	iPhone	0	1	9
3	WD528B	Moto X	3	0	2
4	TF897A	Xperia	7	0	1
5	RF212B	Moto Maxx	2	0	2

Fornecedor

idFornecedor	nome	telefone	contato
1	Sony	8498-8732	Allan
2	Motorola	7987-9900	Cristina
3	Asus	5476-1120	Felipe
4	Nokia	6755-5656	Fábio

```
SELECT   P.Codigo, P.Nome AS 'Nome do Produto', P.Estoque
         , F.Nome AS 'Nome do Fornecedor', F.Contato, F.Telefone
FROM     Produto AS P  FULL JOIN  Fornecedor AS F
                  ON     P.idFornecedor = F.idFornecedor
```

Codigo	Nome do Produto	Estoque	Nome do Fornecedor	Contato	Telefone
XT890A	Asus Zenfone	5	Asus	Felipe	5476-1120
RQ765B	**iPhone**	**0**	**NULL**	**NULL**	**NULL**
WD528B	Moto X	3	Motorola	Cristina	7987-9900
TF897A	Xperia	7	Sony	Allan	8498-8732
RF212B	Moto Maxx	2	Motorola	Cristina	7987-9900
NULL	**NULL**	**NULL**	**Nokia**	**6755-565**	**Fábio**



Join com mais de duas tabelas

Conforme visto, junções entre tabelas são operações triviais em bancos de dados relacionais. Estender o mesmo conceito a mais de duas tabelas não é diferente. No dia-a-dia faremos junções entre três, quatro, cinco ou mais objetos.

Apesar da complexidade aumentar, se mantermos uma lógica em mente, não teremos problemas em estender o mesmo conceito para junções com dez ou mais objetos.

A ordem que colocarmos as tabelas ou as cláusulas JOIN, LEFT JOIN, RIGHT JOIN, ocasionam diferenças na geração dos resultados e conforme já visto, devemos estar atentos a isso.

Mas se tratarmos a consulta, envolvendo três ou mais tabelas como conjunto de resultados, não teremos maiores problemas na construção das queries.

Join com mais de duas tabelas

Mantenha em mente cada junção entre uma tabela e outra, como um conjunto de resultados e faça as junções através deles. No script abaixo perceba que, independente das junções que tivermos, se tratarmos a cada duas tabelas como um conjunto e a próxima junção como um novo conjunto o entendimento será mais fácil.

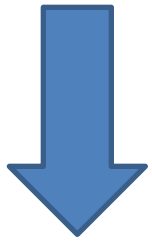
Quando aplicamos JOIN entre as tabelas, a ordem em que estão, não irão afetar os resultados, assim podemos inverter as ordens das tabelas que o efeito será o mesmo:

```
SELECT   <tabela 1>.<coluna 1>, ..., <tabela 1>.<coluna n>
       , <tabela 2>.<coluna 1>, ..., <tabela 2>.<coluna n>
       , <tabela 2>.<coluna 1>, ..., <tabela 2>.<coluna n>
FROM     <tabela 1>  JOIN   <tabela 3>
                    ON     <tabela 3>.<coluna chave> = <tabela 1>.<coluna chave>
                    JOIN   <tabela 2>
                    ON     <tabela 2>.<coluna chave> = <tabela 3>.<coluna chave>
```

Join com mais de duas tabelas

Se alternarmos os joins entre JOIN, LEFT JOIN ou RIGHT JOIN devemos ter cuidado, pois o resultado final será afetado. Veja o exemplo abaixo e perceba como há diferença entre as junções:

1º. Conjunto de Resultados



```
SELECT   <tabela 1>.<coluna 1>, ..., <tabela 1>.<coluna n>
       , <tabela 2>.<coluna 1>, ..., <tabela 2>.<coluna n>
       , <tabela 2>.<coluna 1>, ..., <tabela 2>.<coluna n>
FROM     <tabela 1>  LEFT JOIN   <tabela 2>
                     ON     <tabela 1>.<coluna chave> = <tabela 2>.<coluna chave>
                     JOIN   <tabela 3>
                     ON     <tabela 1>.<coluna chave> = <tabela 3>.<coluna chave>
```

2º. Conjunto de Resultados

Join com mais de duas tabelas

Que é totalmente diferente se invertermos os joins conforme mostrado abaixo:

1º. Conjunto de Resultados

```
SELECT   <tabela 1>.<coluna 1>, ..., <tabela 1>.<coluna n>
       , <tabela 2>.<coluna 1>, ..., <tabela 2>.<coluna n>
       , <tabela 2>.<coluna 1>, ..., <tabela 2>.<coluna n>
FROM     <tabela 1> JOIN   <tabela 2>
                    ON     <tabela 1>.<coluna chave> = <tabela 2>.<coluna chave>
                    LEFT JOIN   <tabela 3>
                    ON     <tabela 1>.<coluna chave> = <tabela 3>.<coluna chave>
```

2º. Conjunto de Resultados

Join com mais de duas tabelas

A mesma diferença será estendida se trocarmos pela cláusula RIGHT JOIN, ou seja, quando temos apenas a cláusula JOIN, não precisamos nos preocupar com a ordem em que as tabelas são posicionadas.

Mas se em nossas junções tivermos qualquer tipo de LEFT JOIN ou RIGHT JOIN, temos que ter atenção no posicionamento e ordem em que as tabelas estarão, assim como cada cláusula de junção.

A forma mais fácil de projetarmos o resultado final é combinando as junções por conjunto de resultado, ou seja, obter o primeiro conjunto de resultado através da junção entre duas tabelas. A partir deste conjunto de resultados, fazer nova junção com outra tabela, obtendo assim o segundo conjunto de resultados e assim por diante.

Exemplos de Fixação

Para ilustrarmos ainda mais o conceito de junção entre mais de duas tabelas, veja o seguinte DER:

Exemplos de Fixação

Gerando a extração com JOIN entre as tabelas:

Cor

idCor	descricao
1	Branco
2	Preto
3	Azul
4	Vermelho
5	Amarelo

Produto

idProduto	codigo	nome	estoque	descontinuado	idFornecedor	idCor
1	XT890A	Asus Zenfone	5	0	3	4
2	RQ765B	iPhone	0	1	9	4
3	WD528B	Moto X	3	0	2	7
4	TF897A	Xperia	7	0	1	1
5	RF212B	Moto Maxx	2	0	2	1

Fornecedor

idFornecedor	nome	telefone	contato
1	Sony	8498-8732	Allan
2	Motorola	7987-9900	Cristina
3	Asus	5476-1120	Felipe
4	Nokia	6755-5656	Fábio

```
SELECT   P.Codigo, P.Nome AS 'Nome do Produto', P.Estoque
         , F.Nome AS 'Nome do Fornecedor', F.Contato, F.Telefone
         , C.Descricao AS 'Cor'
FROM     Produto AS P JOIN Cor AS C          ON C.idCor = P.idCor
                      JOIN Fornecedor AS F   ON F.idFornecedor = P.idFornecedor
```

Codigo	Nome do Produto	Estoque	Nome do Fornecedor	Contato	Telefone	Cor
XT890A	Asus Zenfone	5	Asus	Felipe	5476-1120	Vermelho
TF897A	Xperia	7	Sony	Allan	8498-8732	Branco
RF212B	Moto Maxx	2	Motorola	Cristina	7987-9900	Branco

Exemplos de Fixação

Quando aplicamos as cláusulas LEFT JOIN ou RIGHT JOIN, percebam os resultados:

Cor

idCor	descricao
1	Branco
2	Preto
3	Azul
4	Vermelho
5	Amarelo

Produto

idProduto	codigo	nome	estoque	descontinuado	idFornecedor	idCor
1	XT890A	Asus Zenfone	5	0	3	4
2	RQ765B	iPhone	0	1	9	4
3	WD528B	Moto X	3	0	2	7
4	TF897A	Xperia	7	0	1	1
5	RF212B	Moto Maxx	2	0	2	1

Fornecedor

idFornecedor	nome	telefone	contato
1	Sony	8498-8732	Allan
2	Motorola	7987-9900	Cristina
3	Asus	5476-1120	Felipe
4	Nokia	6755-5656	Fábio

```
SELECT   P.Codigo, P.Nome AS 'Nome do Produto', P.Estoque
         , F.Nome AS 'Nome do Fornecedor', F.Contato, F.Telefone
         , C.Descricao AS 'Cor'
FROM     Produto AS P JOIN  Fornecedor AS F  ON  P.idFornecedor = F.idFornecedor
                    RIGHT JOIN  Cor AS C  ON C.idCor =  P.idCor
```

Codigo	Nome do Produto	Estoque	Nome do Fornecedor	Contato	Telefone	Cor
XT890A	Asus Zenfone	5	Asus	Felipe	5476-1120	Vermelho
TF897A	Xperia	7	Sony	Allan	8498-8732	Branco
RF212B	Moto Maxx	2	Motorola	Cristina	7987-9900	Branco
NULL	NULL	NULL	NULL	NULL	NULL	Preto
NULL	NULL	NULL	NULL	NULL	NULL	Azul
NULL	NULL	NULL	NULL	NULL	NULL	Amarelo

Exemplos de Fixação

Trocando o RIGHT pelo LEFT o resultado é alterado:

Cor

idCor	descricao
1	Branco
2	Preto
3	Azul
4	Vermelho
5	Amarelo

Produto

idProduto	codigo	nome	estoque	descontinuado	idFornecedor	idCor
1	XT890A	Asus Zenfone	5	0	3	4
2	RQ765B	iPhone	0	1	9	4
3	WD528B	Moto X	3	0	2	7
4	TF897A	Xperia	7	0	1	1
5	RF212B	Moto Maxx	2	0	2	1

Fornecedor

idFornecedor	nome	telefone	contato
1	Sony	8498-8732	Allan
2	Motorola	7987-9900	Cristina
3	Asus	5476-1120	Felipe
4	Nokia	6755-5656	Fábio

```
SELECT   P.Codigo, P.Nome AS 'Nome do Produto', P.Estoque
         , F.Nome AS 'Nome do Fornecedor', F.Contato, F.Telefone
         , C.Descricao AS 'Cor'
FROM     Produto AS P JOIN   Fornecedor AS F   ON  P.idFornecedor = F.idFornecedor
                     LEFT JOIN   Cor AS C     ON C.idCor =  P.idCor
```

Codigo	Nome do Produto	Estoque	Nome do Fornecedor	Contato	Telefone	Cor
XT890A	Asus Zenfone	5	Asus	Felipe	5476-1120	Vermelho
WD528B	Moto X	3	Motorola	Cristina	7987-9900	NULL
TF897A	Xperia	7	Sony	Allan	8498-8732	Branco
RF212B	Moto Maxx	2	Motorola	Cristina	7987-9900	Branco

Funções de Agregação

Funções de agregação são funções que estão embutidas (bulti-in) no banco de dados e utilizamos quando precisamos calcular valores, contabilizar número de registros ou retornar os maiores e menores valores dentro de uma coluna.

Possuem as seguintes características:

- ✔ Retornam valores escalares
- ✔ Retornam a coluna sem nome
- ✔ Ignoram colunas NULL (exceção a COUNT(*))
- ✔ Podem ser usadas nas cláusulas:

SELECT, HAVING e ORDER BY

Funções de Agregação

São frequentemente utilizadas com a cláusula GROUP BY, mas não são restritas ao uso sem este comando. O exemplo abaixo mostra funções de agregação sem o uso do GROUP BY.

```
SELECT    AVG(unitprice) AS avg_price,
                  MIN(qty)AS min_qty,
                  MAX(discount) AS max_discount
FROM Sales.OrderDetails;
```

```
avg_price min_qty max_discount
--------- ------- ------------
26.2185   1       0.250
```

Funções de Agregação

Iremos focar nas funções de agregação de uso comum, mas é importante saber que existem outras categorias de funções agregadas como estatísticas e outras.

Uso Comum	Estatísticas
• SUM • MIN • MAX • AVG • COUNT • COUNT_BIG	• STDEV • STDEVP • VAR • VARP

Funções de Agregação

Podemos combinar a cláusula DISTINCT com funções de agregação para sumarizar somente valores ÚNICOS.

A agregação com a claúsula DISTINCT elimina valores duplicados, não linhas (exceto se utilizarmos SELECT DISTINCT).

Compare os resultados parciais do exemplo abaixo, com e sem o uso da cláusula DISTINCT:

```
SELECT empid, YEAR(orderdate) AS orderyear,
       COUNT(custid) AS all_custs,
       COUNT(DISTINCT custid) AS unique_custs
FROM Sales.Orders
GROUP BY empid, YEAR(orderdate);
```

```
empid       orderyear   all_custs   unique_custs
----------- ----------- ----------- ------------
1           2006        26          22
1           2007        55          40
1           2008        42          32
2           2006        16          15
```

Funções de Agregação

A maioria das funções de agregação, simplesmente ignoram o NULL e não geram nenhum erro. As funções a seguir Ignoram o NULL:

AVG(<coluna>), COUNT (<coluna>), ...

A função COUNT usada com *, é uma EXCEÇÃO a regra acima, contabilizando TODAS AS LINHAS:

COUNT(*)

Esse tipo de comportamento das funções agregadas perante o NULL, pode produzir resultados INCORRETOS, como é o caso abaixo, utilizando a função AVG. Se quisermos contabilizar os registros nulos, podemos ajustar os dados com a função ISNULL.

```
SELECT  AVG(c2) AS AvgWithNULLs,
        AVG(ISNULL(c2,0)) AS AvgWithNULLReplace
FROM    dbo.t2;
```

Cláusula GROUP BY

GROUP BY cria grupos no retorno das linhas de acordo com a combinação da(s) coluna(s) escritas na cláusula GROUP BY.

```
SELECT <select_list>
FROM <table_source>
WHERE <search_condition>
GROUP BY <group_by_list>;
```

O GROUP BY retira os “detalhes” das linhas, fazendo um cálculo com a função de agregação selecionada para a coluna escolhida.

```
SELECT empid, SUM(freight) AS fht
FROM Sales.Orders
GROUP BY empid;
```

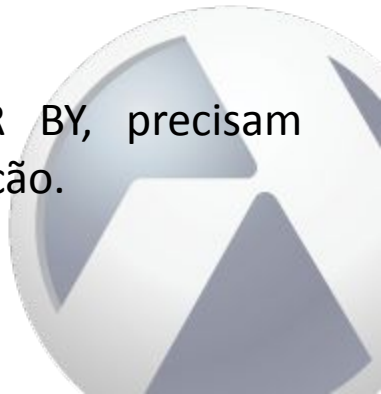
```
SELECT empid, COUNT(*) AS cnt
FROM Sales.Orders
GROUP BY empid
```

Faculdade IMPACTA TECNOLOGIA

Cláusula GROUP BY

Ordem Lógica	Fase	Comentário
5	SELECT	
1	FROM	
2	WHERE	
3	GROUP BY	Cria Grupos
4	HAVING	Opera na Filtragem dos Grupos
6	ORDER BY	

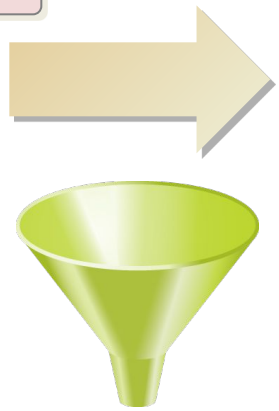
- Se a consulta (query) usa GROUP BY, todas as fases subsequentes irão operar nos grupos.
- HAVING, SELECT e ORDER BY precisam NECESSARIAMENTE que retornar apenas um valor por grupo.
- Todas as colunas que aparecerem no SELECT, HAVING e ORDER BY, precisam OBRIGATORIAMENTE estar ou no GROUP BY ou numa Função de Agregação.



Cláusula GROUP BY

```
SELECT orderid, empid, custid
FROM Sales.Orders;
```

orderid	empid	custid
10643	6	1
10692	4	1
10926	4	2
10625	3	2
10365	3	3



orderid	empid	custid
10643	6	1
10692	4	1
10926	4	2
10625	3	2

WHERE custid IN(1,2)

GROUP BY empid

SELECT output

empid	COUNT(*)
6	1
4	2
3	1

Cláusula GROUP BY

Funções de agregação são comumente usadas em cláusula SELECT, sumarizando a(s) coluna(s) colocada(s) no GROUP BY.

```
SELECT custid, COUNT(*) AS cnt
FROM Sales.Orders
GROUP BY custid;
```

Funções de agregação podem referir qualquer coluna, não apenas as que estiverem escritas no GROUP BY.

```
SELECT productid, MAX(qty) AS largest_order
FROM Sales.OrderDetails
GROUP BY productid;
```

Cláusula HAVING

- HAVING filtra os dados obtidos através do GROUP BY.
- HAVING fornece condição de pesquisa que precisa ser satisfeita para cada grupo.
- HAVING é processado após a execução do GROUP BY.

```
SELECT custid, COUNT(*) AS count_orders
FROM Sales.Orders
GROUP BY custid
HAVING COUNT(*) > 10;
```

WHERE x HAVING

WHERE

- Filtra linhas ANTES dos grupos serem criados
- Controla quais linhas serão passadas para o GROUP BY

HAVING

- Filtra GRUPOS
- Controla quais GRUPOS serão passados para próxima fase lógica

WHERE x HAVING

A utilização da expressão COUNT(*) combinado com a cláusula HAVING é muito útil para solucionar problemas comuns de negócios. Exemplos:

- Mostre apenas os clientes que fizeram mais de um pedido

```
SELECT c.custid, COUNT(*) AS cnt
FROM Sales.Customers AS c
      JOIN Sales.Orders AS o ON c.custid = o.custid
GROUP BY c.custid
HAVING COUNT(*) > 1;
```

- Retorne somente produtos que aparecem mais de 10 vezes nos pedidos

```
SELECT p.productid, COUNT(*) AS cnt
FROM Production.Products AS p JOIN Sales.OrderDetails AS od ON
p.productid = od.productid
GROUP BY p.productid
HAVING COUNT(*) >= 10;
```

Faculdade IMPACTA TECNOLOGIA

Obrigado!

Aula Gravada por:

Prof. Msc. Gustavo Bianchi Maia

gustavo.maia@faculdadeimpacta.com.br

Material criado e oferecido por :

Prof. Sand Jacques Onofre

Sand.onofre@faculdadeimpacta.com.br