Banco de Dados Relacionais e não Relacionais

Prof. Henrique Batista da Silva

Expressões aritméticas, alias (atributo e relação) e uso do asterisco

Expressões aritméticas

Expressões com dados numéricos e de data usando operadores aritméticos:

Operador	Descrição
+	Adição
-	Subtração
*	Multiplicação
/	Divisão

```
SELECT NomEmp, Salario, Salario+300
FROM EMPREGADO
```

Expressões aritméticas

- A multiplicação e a divisão têm prioridade sobre a adição e a subtração.
- Os operadores com a mesma prioridade são avaliados da esquerda para a direita.
- Parênteses são usados para forçar a avaliação priorizada e para esclarecer as instruções.
- Exemplo:

```
SELECT NomEmp, Salario, 12*Salario+100
FROM EMPREGADO

SELECT NomEmp, Salario, 12*(Salario+100)
FROM EMPREGADO
```

Aliases de Colunas

Cada coluna retornada no comando SELECT, o nome do atributo é apresentado como resultado.
Porém, **é possível atribuir rótulos próprios adicionando um alias (apelido) de coluna após cada elemento da cláusula SELECT**.
É útil para cálculos, especialmente se o nome da coluna for longo.

```
SELECT NomEmp AS NOME_EMPREGADO, Salario AS
SAL_EMPREGADO
FROM EMPREGADO EMP

SELECT NomEmp, Salario, 12*(Salario) AS
SALARIO_ANUAL
FROM EMPREGADO
```

Aliases de Colunas

Quando múltiplas tabelas são utilizadas em uma única consulta, é necessário uma maneira para identificar a qual tabela a coluna pertence.
Principalmente quando há colunas de tabelas diferentes com o mesmo nome.
Desta forma, define-se um alias a cada tabela para utilizá-lo ao longo da consulta.

```
SELECT NomEmp AS NOME_EMPREGADO, Salario AS SAL_EMPREGADO
FROM EMPREGADO AS EMP
```

Definindo Alias de Relações

Em SQL é possível dar nomes alternativos pela adição do qualificador **AS** (alias).

O alias pode ser usado tanto nos atributos quantos nas relações e em ambas as cláusulas SELECT e FROM.

O alias de relações é utilizado principalmente quando é necessário realizar junção de duas ou mais relações.

Neste caso, **o alias é importante para a explicitar a distinção dos atributos de cada uma das relações** durante toda a consulta.

Definindo Alias de Relações

Exemplo:

Recupere o nome e o endereço de todos os empregados.

```
SELECT E.NomEmp, E.EndEmp
FROM Empregado E
```

O Uso do **AS** não é obrigatório para pseudônimo de relações.

Uso do asterisco

Para recuperar **todos os valores dos atributos** da tuplas selecionadas, não é necessário especificar todos os seus nomes.
Utiliza-se apenas um asterisco (*), que significa selecionar todos os atributos.

```
SELECT *
FROM   Empregado

SELECT *
FROM   Empregado
WHERE MatEmp = 11;
```

Distinct e Cláusula Where

Removendo Linhas Duplicadas

Uma consulta pode retornar linhas de dados duplicadas. Por exemplo, ao selecionar todos os códigos do clientes que possuem conta:

```
SELECT CODCLIENTE
FROM CONTA;
```

Alguns clientes podem ter mais de uma conta, o mesmo código do cliente será exibido para cada conta de cada cliente.

Removendo Linhas Duplicadas

O objetivo é obter o conjunto específico de cliente que possuem conta (independente de quantas contas ele possui).

É necessário utilizar a cláusula DISTINCT, diretamente após a cláusula SELECT:

```
SELECT DISTINCT CODCLIENTE
FROM CONTA;
```

Cláusula WHERE

A cláusula WHERE é utilizada para restringir as linhas selecionadas.
A cláusula WHERE segue a cláusula FROM.

```
SELECT NOMEMP, CARGO
FROM EMPREGADO
WHERE BAIRRO = 'SAVASSI'
```

Cláusula WHERE

Exemplo:

```
SELECT NomEmp, CidEmp
FROM   Empregado
WHERE MatEmp = 11;
```

- Esta consulta envolve somente a relação EMPREGADO relacionada na cláusula FROM
- A consulta seleciona as tuplas de EMPREGADO que satisfazem a condição da cláusula WHERE
- Então, projeta o resultado dos atributos NomEmp e CidEmp relacionados nos cláusula SELECT.

Cláusula WHERE

- A cláusula SELECT, em SQL, especifica atributos para projeção.

- A cláusula WHERE especifica a condição de seleção.
  - Somente aquelas tuplas que satisfazem a condição serão selecionadas.

Cláusula WHERE

A ausência da cláusula Where significa que não há nenhuma condição para seleção de tuplas.

Assim, todas as tuplas da relação da cláusula FROM serão retornadas no resultado da consulta.

```
SELECT NomEmp
FROM   Empregado;
```

Operadores de Comparação

Operadores de Comparação

Em SQL, os operadores lógicos básicos de comparação usado são:

=, <, <=, >, >=, e <>

Operadores de: igualdade, menor, menor ou igual, maior, maior ou igual e diferente.

Operadores de Comparação

- Operadores lógicos de comparação:

AND, OR e NOT

  - Operado AND retorna verdadeiro (*true*), se todas as condições forem verdadeiras.
  - Operador OR retorna verdadeiro (*true*), se pelo menos umas das condições forem verdadeiras.
  - Operador NOT retorna verdadeiro (*true*), se condição for falsa.

LIKE

- Em SQL é possível criar condição para comparação de partes de uma cadeia de caracteres por meio do operador de comparação LIKE.
- Esse operador pode ser usado para comparação de padrões de cadeia.
- As partes das cadeias podem ser especificadas usando dois caracteres:
  - % : substitui um número arbitrário de caracteres.
  - Underscore (_): substitui um único caracter.

LIKE

Exemplo:

Recupere todos os empregados cujos endereços sejam de Belo Horizonte:

```
SELECT NomEmp
FROM   Empregado
WHERE CidEmp LIKE '%Belo Horizonte%';
```

Recupere todos os empregados cujo a segunda letra do nome seja ‘a’:

```
SELECT NomEmp
FROM   Empregado
WHERE NomEmp LIKE '_a%';
```

BETWEEN

- Em SQL também é possível comparar valores dentro de um determinado intervalo:
- Exemplo:
  - Recupere todos os empregados do departamento 1 que ganham entre 30 mil e 40 mil.

```
SELECT *
FROM   Empregado
WHERE (Salario BETWEEN 30000 AND 40000) AND CodDepto = 1;
```

NOT IN

- Também é possível realizar comparações com uma lista de valores ao invés de apenas um único valor.
- O operador IN (lista), pode ser utilizados para comparação com uma lista de valores.
- Exemplo:
  - Recupere os empregados cujo o cargo não seja nem de vendedor e nem de presidente.

```
SELECT NomEmp
FROM   EMPREGADO
WHERE  CARGO NOT IN ('Vendedor', 'Presidente')
```

IS NULL

- Utilizando o operador IS NULL é possível comparar e verificar se o valor do atributo é nulo.
- Exemplo:
  - Recupere os empregados que trabalham no departamento 10 e que não receberam nenhuma comissão.

```
SELECT NomEmp
FROM   EMPREGADO
WHERE  COMISSAO IS NULL AND CodDepto = 1

SELECT NomEmp
FROM   EMPREGADO
WHERE  COMISSAO IS NOT NULL AND CodDepto = 1
```

Funções Agregadas

Funções Agregadas em SQL

O SQL tem funcionalidades que incorporam conceitos de funções agregadas.
Há diversas funções para este fim:

- Max()
  - Retorna o valor máximo dentro de um conjunto.
- Min()
  - Retorna o valor mínimo dentro de um conjunto.
- Avg()
  - Retorna o valor médio de um conjunto.
- Sum()
  - Retorna a soma dos valores de um conjunto.
- Count()
  - Retorna a quantidade de valores de um conjunto.

Funções Agregadas em SQL

Exemplo 1:

Encontre a soma dos salários, o maior salário, o menor salário, e a média salarial de todos os empregados.

```
SELECT  SUM (salario), MAX (salario), MIN (salario), AVG (salario)
FROM    Empregado;
```

Funções Agregadas em SQL

Exemplo 2

Recupera o número total de empregados da empresa:

```
SELECT  COUNT (*)
FROM    Empregado;
```

Neste exemplo, * refere-se às linhas (tuplas), logo COUNT devolverá o número de linhas do resultado da consulta.

Order By, Group By e Having

Ordenando o resultado das consultas

SQL permite que o usuário ordene as tuplas do resultado de uma consulta pelos valores de seus atributos, utilizando a cláusula **ORDER BY**.

```
SELECT NomEmp
FROM EMPREGADO
ORDER BY NomEmp
```

A ordenação padrão é de forma ascendente. Pode-se também especificar a palavra-chave DESC (sempre depois do nome da coluna) para ordenar os valores de forma descendente.

```
SELECT NomEmp
FROM EMPREGADO
ORDER BY NomEmp DESC
```

Agrupamento

Em muitos casos, é necessário **aplicar as funções agregadas para os subgrupos de tuplas de uma relação**. No qual os subgrupos são escolhidos com base em alguns atributos.

SQL possui a cláusula **Group By** para esse fim. Esta especifica os atributos de agrupamento, que também poderiam aparecer na cláusula Select.

Agrupamento

Exemplo:

Para cada departamento, recupere seu número, número de empregados que nele trabalham e a média de seus salários.

```
SELECT CodDepto, COUNT (*) as 'qtd', AVG (Salario) as 'sal'
FROM   Empregado
GROUP BY CodDepto;
```

As tuplas de Empregado serão particionadas em grupos, **cada grupo tendo o mesmo valor para o atributo do agrupamento**.
A função COUNT() e AVG (), serão aplicada **em cada grupo de tuplas**

Agrupamento

Exemplo:

CPF	Nome	Sal	CodDepto
123	José	1000	1
223	Maria	2000	1
323	João	2000	2
123	Alberto	4000	2
121	Carla	3000	3

Agrupamento das tuplas de Empregado por meio do valor do CodDepto

Resultado da consulta

CodDepto	COUNT(*)	AVG(Sal)
1	2	1500
2	2	3000
3	1	3000

Agrupamento

Algumas vezes é necessário recuperar os valores das funções agregadas apenas para os grupos que satisfazem certas condições.

Como a cláusula GROUP BY é executada após a avaliação da cláusula WHERE, você não pode adicionar condição de filtro à cláusula WHERE para este propósito.

Por exemplo: Suponha que seja necessário recuperar, para cada projeto, o seu **numero**, seu **nome** e **numero de empregados** que nele trabalham. **Mas somente os projetos com mais de dois empregados**.

Agrupamento

Exemplo:

Suponha que seja necessário recuperar, para cada **projeto**, o seu **numero**, seu **nome** e **numero de empregados** que nele trabalham.

```
SELECT P.CodProjeto, P.ProjNome, COUNT (*)
FROM Projeto P, Trabalha_Em T
WHERE P.CodProjeto = T.CodProjeto
GROUP BY P.CodProjeto, P.ProjNome
```

Agrupamento

Exemplo:

Suponha que seja necessário recuperar, para cada projeto, o seu numero, seu nome e numero de empregados que nele trabalham. **Mas somente os projetos com mais de dois empregados**.

```
SELECT P.CodProjeto, P.ProjNome, COUNT (*)
FROM Projeto P, Trabalha_Em T
WHERE P.CodProjeto = T.CodProjeto AND COUNT (*) > 2
GROUP BY P.CodProjeto, P.ProjNome
```

Agrupamento

Exemplo:

Suponha que seja necessário recuperar, para cada projeto, o seu numero, seu nome e numero de empregados que nele trabalham. **Mas somente os projetos com mais de dois empregados**.

```
SELECT P.CodProjeto, P.ProjNome, COUNT (*)
FROM Projeto P, Trabalha_Em T
WHERE P.CodProjeto = T.CodProjeto AND COUNT (*) > 2
GROUP BY P.CodProjeto, P.ProjNome
```

Incorreto, pois os grupos ainda não foram gerados no momento em que a cláusula WHERE é avaliada.

Agrupamento

Exemplo:

Suponha que seja necessário recuperar, para cada projeto, o seu numero, seu nome e numero de empregados que nele trabalham. **Mas somente os projetos com mais de dois empregados**.

```
SELECT P.CodProjeto, P.ProjNome, COUNT (*)
FROM Projeto P, Trabalha_Em T
WHERE P.CodProjeto = T.CodProjeto
GROUP BY P.CodProjeto, P.ProjNome
HAVING COUNT (*) > 2
```

Agrupamento

Enquanto as condições de seleção da cláusula WHERE limitam as tuplas nas quais as funções serão aplicadas, a cláusula HAVING serve para escolher grupos inteiros.

Agrupamento (Exercícios)

Retorne o nome do departamento, o código e o número de empregados que nele trabalha, mas somente para os departamentos que tenham mais de 1 empregado.

Agrupamento (Exercícios)

Retorne o nome do departamento, o código e o número de empregados que nele trabalha, mas somente para os departamentos que tenham mais de 1 empregado.

```
SELECT  D.DeptoNome as 'Nome Departamento', D.CodDepto as 'Código
Departamento', COUNT(*) as 'Número de Empregados'
FROM   Empregado E JOIN Departamento D ON D.CodDepto = E.CodDepto
Group by D.CodDepto, D.DeptoNome
Having COUNT(*) > 1
```

Agrupamento (Exercícios)

Retorne o nome do departamento, o código e a média dos salários dos empregados que nele trabalha, mas somente se a média for menor que 3000.

Agrupamento (Exercícios)

Retorne o nome do departamento, o código e a média dos salários dos empregados que nele trabalha, mas somente se a média for menor que 3000.

```
SELECT  D.DeptoNome as 'Nome Departamento', D.CodDepto as 'Código
Departamento', AVG(E.Salario) as 'Número de Empregados'
FROM   Empregado E JOIN Departamento D ON D.CodDepto = E.CodDepto
Group by D.CodDepto, D.DeptoNome
Having AVG(E.Salario) < 3000
```

Consultas Aninhadas

Consultas Aninhadas

Algumas consultas dependem da busca de valores presentes no banco de dados para, então, usá-lo na condição de comparação.

Essas consultas podem ser formuladas por meio das consultas aninhadas.

Um bloco completo de *select-from-where* dentro da cláusula WHERE da consulta externa.

Consultas Aninhadas

Exemplo:

Recupere o Nome do Funcionário que tem o maior salário Bruto da empresa:

```
SELECT NomEmp
FROM EMPREGADO
WHERE SALARIOBRUTO = (SELECT MAX(SALARIOBRUTO)
                      FROM EMPREGADO)
```

Recupera o maior salário da tabela funcionário

Pesquisa todos os salários da tabela funcionário e compara com o valor MAX. Quando encontrar, projeta o nome do funcionário (campo NOME da tupla com SALARIO = MAX)

Consultas Aninhadas

Exemplo:

Recupere o CPF de todos os empregados que trabalham a mesma quantidade de horas em algum dos projetos em que o empregado 'John' (CPF = '123') trabalhe.

```
SELECT   DISTINCT MatEmp
FROM     Trabalha_Em
WHERE    Horas IN (SELECT Horas
          FROM Trabalha_Em
          WHERE MatEmp = '123');
```

Recupera todas as horas trabalhadas nos projetos do CPF 123.
O resultado é um conjunto

Consultas Aninhadas

```
SELECT    DISTINCT MatEmp
FROM      Trabalha_Em
WHERE     Horas IN (SELECT Horas
           FROM Trabalha_Em
          WHERE MatEmp = '123');
```

Trabalha_Em

CPF	CodProj	Horas
123	1	20
223	1	10
323	2	10
123	2	15
121	3	20
112	3	15
332	3	20
333	4	20

Referência: Navathe. **Sistemas de Banco de Dados** – 2011, 6ª Ed.

Consultas Aninhadas

Trabalha_Em

CPF	CodProj	Horas
123	1	**20**
223	1	10
323	2	10
123	2	**15**
121	3	20
112	3	15
332	3	20
333	4	20

```
SELECT   DISTINCT MatEmp
FROM     Trabalha_Em
WHERE    Horas IN (SELECT Horas
          FROM Trabalha_Em
          WHERE MatEmp = '123');
```

Referência: Navathe. **Sistemas de Banco de Dados** – 2011, 6ª Ed.

Consultas Aninhadas

Trabalha_Em

CPF	CodProj	Horas
123	1	**20**
223	1	10
323	2	10
123	2	**15**
121	3	20
112	3	15
332	3	20
333	4	20

```
SELECT   DISTINCT MatEmp
FROM     Trabalha_Em
WHERE    Horas IN (SELECT Horas
          FROM Trabalha_Em
          WHERE MatEmp = '123');
```

Resultado

Conj_Resultado

Horas
20
15

Referência: Navathe. **Sistemas de Banco de Dados** – 2011, 6ª Ed.

Consultas Aninhadas

```
SELECT   DISTINCT MatEmp
FROM     Trabalha_Em
WHERE    Horas IN (SELECT Horas
          FROM Trabalha_Em
          WHERE MatEmp = '123');
```

Trabalha_Em

CPF	CodProj	Horas
123	1	20
223	1	10
323	2	10
123	2	15
121	3	20
112	3	15
332	3	20
333	4	20

Referência: Navathe. **Sistemas de Banco de Dados** – 2011, 6ª Ed.

Consultas Aninhadas

Trabalha_Em

CPF	CodProj	Horas
123	1	20
223	1	10
323	2	10
123	2	15
121	3	20
112	3	15
332	3	20
333	4	20

```
SELECT   DISTINCT MatEmp
FROM     Trabalha_Em
WHERE    Horas IN (SELECT Horas
          FROM Trabalha_Em
          WHERE MatEmp = '123');
```

IN

Conj_Resultado

Horas
20
15

Referência: Navathe. **Sistemas de Banco de Dados** – 2011, 6ª Ed.

Consultas Aninhadas

```
SELECT   DISTINCT MatEmp
FROM     Trabalha_Em
WHERE    Horas IN (SELECT Horas
          FROM Trabalha_Em
          WHERE MatEmp = '123');
```

Trabalha_Em

CPF	CodProj	Horas
123	1	20
223	1	10
323	2	10
123	2	15
121	3	20
112	3	15
332	3	20
333	4	20

IN

Conj_Resultado

Horas
20
15

Referência: Navathe. **Sistemas de Banco de Dados** – 2011, 6ª Ed.

Consultas Aninhadas

```
SELECT   DISTINCT MatEmp
FROM     Trabalha_Em
WHERE    Horas IN (SELECT Horas
          FROM Trabalha_Em
          WHERE MatEmp = '123');
```

Trabalha_Em

CPF	CodProj	Horas
123	1	**20**
223	1	10
323	2	10
123	2	**15**
121	3	**20**
112	3	**15**
332	3	**20**
333	4	**20**

IN

Conj_Resultado

Horas
20
15

Resultado

MatEmp
123
123
121
112
332
333

Referência: Navathe. **Sistemas de Banco de Dados** – 2011, 6ª Ed.

Consultas Aninhadas

```
SELECT   DISTINCT MatEmp
FROM     Trabalha_Em
WHERE    Horas IN (SELECT Horas
          FROM Trabalha_Em
          WHERE MatEmp = '123');
```

Trabalha_Em

CPF	CodProj	Horas
123	1	**20**
223	1	10
323	2	10
123	2	**15**
121	3	**20**
112	3	**15**
332	3	**20**
333	4	**20**

IN →

Conj_Resultado

Horas
20
15

DISTINCT →

MatEmp
123
121
112
332
333

Referência: Navathe. **Sistemas de Banco de Dados** – 2011, 6ª Ed.

Consultas Aninhadas

A palavras ALL pode ser combinada com algum dos operadores de comparação.

Exemplo:

Recupere o nome dos empregados cujo salários são maiores que os salários de todos os empregados do departamento 2.

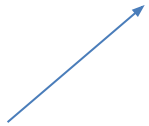
```
SELECT NomEmp
FROM Empregado
WHERESalario > ALL (SELECT Salario
                    FROM Empregado
                    WHERE CodDepto = 2);
```

Consultas Aninhadas

Exemplo (Alias):

Recupere o nome de cada um dos empregados que tenham dependentes cujo o sexo seja o mesmo do empregado em questão:

```
SELECT NomEmp
FROM Empregado AS E
WHERE E.MatEmp IN (SELECT MatEmp
                   FROM Dependente
                   WHERE E.Sexo = Sexo);
```



Atributo Sexo da tabela Empregado

Função EXISTS

É usada para verificar se o resultado de uma consulta aninhada é vazio (não contém nenhuma tupla) ou não.

Exemplo:

Recupere o nome dos empregados que possuem dependentes:

```
SELECT NomEmp
FROM Empregado E
WHERE Exists ( SELECT *
        FROM Dependente D
        WHERE D.MatEmp = E.MatEmp)
```

Se existir qualquer tupla no conjunto resultado da consulta interna, a tupla Empregado será selecionada.

Função EXISTS

Pode-se também utilizar NOT EXISTS:
Exemplo:

Recupere o nome dos empregados que **não** possuem dependentes:

```
SELECT NomEmp
FROM Empregado E
WHERE NOT Exists (SELECT *
        FROM Dependente D
        WHERE D.MatEmp = E.MatEmp)
```

Se **não** existir nenhuma tupla no conjunto resultado da consulta interna, a tupla Empregado será selecionada.

JOINS

Vínculos de tabelas

Em SQL é possível realizar vínculos entre duas ou mais relações do banco de dados.

Para tanto, **todas as relações envolvidas devem estar presentes na cláusula FROM**.

Quando mais de um tabela aparece na cláusula FROM, as condições para vincular as tabelas devem ser incluídas (condição de junção).

Vínculos de tabelas

A **operação de junção é usada para combinar as tuplas relacionadas em duas relações dentro de uma tupla única**.

Importante pois permite processar os relacionamentos entre as relações.

Exemplo:

Recuperar o nome do gerente de cada departamento.
É necessário avaliar cada tupla de departamento com a tupla empregado na qual o valor do CPF case com o valor do CPFGerente da tupla departamento.

Vínculos de tabelas

O resultado da junção é uma relação Q, em que há uma tupla (em Q) para cada combinação de tuplas (uma de R e uma de S), quando a combinação satisfazer a condição de junção.

Na junção, **apenas as combinações de tuplas que satisfazem a condição de junção aparecerão no resultado**.

Vínculos de tabelas

O conceito de junção de tabelas foi incorporado à SQL para **especificar uma tabela que fosse resultado da junção das tabelas na cláusula FROM**.

Exemplo:

Recupere o nome e endereço de todos os empregados que trabalham no departamento ‘Pesquisa’:

Vínculos de tabelas

Resposta 1:

```
SELECT E.NomEmp, E.EndEmp
FROM Empregado E, Departamento D
WHERE D.deptoNome = 'Pesquisa' AND D.codDepto = E.CodDepto;
```

O mecanismo de vinculação das duas relações é a afiliação do funcionário e um departamento armazenada na tabela Empregado (pelo atributo FK).

Condição de junção junto com a condição de seleção.

Vínculos de tabelas

O SGBD é instruído a usar o valor da coluna “CodDepto” na relação Empregado para procurar o nome do departamento associado na relação departamento.

A condição de junção é realizada na cláusula Where (D. codDepto = E. CodDepto).

Vínculos de tabelas

```
SELECT  E.NomEmp, E.EndEmp
FROM    Empregado E, Departamento D
WHERE D.CodDepto = E.CodDepto
        AND D.deptoNome = 'Pesquisa'
```

Empregado

CPF	Nome	End	CodDepto
123	José	Rua 3	1
223	Maria	Rua 10	2
323	João	Rua 10	2
123	Alberto	Rua 15	3
121	Carla	Rua 20	1

Departamento

Codigo	Nome
1	Pesquisa
2	Vendas
3	Administrativo

Vínculos de tabelas

```
SELECT E.NomEmp, E.EndEmp
FROM Empregado E, Departamento D
WHERE D.CodDepto = E.CodDepto
      AND D.deptoNome = 'Pesquisa'
```

Empregado

CPF	Nome	End	CodDepto
123	José	Rua 3	1
223	Maria	Rua 10	2
323	João	Rua 10	2
123	Alberto	Rua 15	3
121	Carla	Rua 20	1

Departamento

Codigo	Nome
1	Pesquisa
2	Vendas
3	Administrativo

Vínculos de tabelas

```
SELECT E.NomEmp, E.EndEmp
FROM Empregado E, Departamento D
WHERE D.CodDepto = E.CodDepto
AND D.deptoNome = 'Pesquisa'
```

FK para a tabela Departamento

Empregado

CPF	Nome	End	CodDepto
123	José	Rua 3	1
223	Maria	Rua 10	2
323	João	Rua 10	2
125	Alberto	Rua 15	3
121	Carla	Rua 20	1

Departamento

Codigo	Nome
1	Pesquisa
2	Vendas
3	Administrativo

Vínculos de tabelas

```
SELECT  E.NomEmp, E.EndEmp
FROM    Empregado E, Departamento D
WHERE D.CodDepto = E.CodDepto
        AND D.deptoNome = 'Pesquisa'
```

Empregado

CPF	Nome	End	CodDepto
123	José	Rua 3	1
223	Maria	Rua 10	2
323	João	Rua 10	2
125	Alberto	Rua 15	3
121	Carla	Rua 20	1

Departamento

Codigo	Nome
1	Pesquisa
2	Vendas
3	Administrativo

Vínculos de tabelas

```
SELECT  E.NomEmp, E.EndEmp
FROM    Empregado E, Departamento D
WHERE D.CodDepto = E.CodDepto
        AND D.deptoNome = 'Pesquisa'
```

Empregado

CPF	Nome	End	CodDepto
123	José	Rua 3	1
223	Maria	Rua 10	2
323	João	Rua 10	2
125	Alberto	Rua 15	3
121	Carla	Rua 20	1

Departamento

Codigo	Nome
1	Pesquisa
2	Vendas
3	Administrativo

Conj. Resultado

Nome	End
José	Rua 3
Carla	Rua 20

JOIN

Resposta 2:

```
SELECT E.NomEmp, E.EndEmp
FROM Empregado E JOIN Departamento D ON E.CodDepto = D.CodDepto
WHERE D.DeptoNome = 'Pesquisa';
```

Condição de seleção separada da condição de junção.
A cláusula FROM contém uma única *tabela juntada* (*joined table*).

JOIN

Execute as duas consultas e veja o resultado:

```
select E.MatEmp as 'Matrícula Empregado', E.NomEmp as 'Nome
Empregado', D.NomDep as 'Nome Dependente'
from Empregado E, Dependente D
WHERE D.MatEmp = E.MatEmp
```

```
select E.MatEmp as 'Matrícula Empregado', E.NomEmp as 'Nome
Empregado', D.NomDep as 'Nome Dependente'
from Empregado E Join Dependente D ON D.MatEmp = E.MatEmp
```

Tips de JOINS (conteúdo extra)

Tipos de JOINS

O tipo de junção padrão entre duas tabelas é a interseção entre as linhas das duas tabelas:

```
SELECT  *
FROM Empregado E JOIN Departamento D ON
E.CodDepto = D.CodDepto
```

Execute a consulta e veja o resultado

Vamos conhecer todos os tipos de Joins entre múltiplas tabelas

Tipos de JOINS

- Inner Join
  - Tipo de junção que retorna somente as linhas que satisfazem a condição de junção entre ambas as tabelas (se omitir o a palavra INNER, por default o resultado será o mesmo)

```
SELECT  *
FROM Empregado E INNER JOIN Departamento D ON E.CodDepto = D.CodDepto
```

Tipos de JOINS

- Outer Join
  - Left Outer Join: retorna todas as linhas da tabela da esquerda e somente as linhas da tabela da direta que satisfazem a condição de junção (preenche como NULL campos das linhas da tabela da esquerda que não satisfazem a condição de junção)

```
SELECT  *
FROM Empregado E LEFT OUTER JOIN Departamento D ON E.CodDepto = D.CodDepto
```

Tipos de JOINS

- Outer Join
  - Right Outer Join: retorna todas as linhas da tabela da direita e somente as linhas da tabela da esquerda que satisfazem a condição de junção (preenche como NULL campos das linhas da tabela da direita que não satisfazem a condição de junção)

```
SELECT  *
FROM Empregado E RIGHT OUTER JOIN Departamento D ON E.CodDepto = D.CodDepto
```

Tipos de JOINS

- Outer Join
  - Full Outer Join: retorna todas as linhas de ambas as tabelas (preenche como NULL campos das linhas que não satisfazem a condição de junção)

```
SELECT  *
FROM Empregado E FULL OUTER JOIN Departamento D ON E.CodDepto = D.CodDepto
```

Tipos de JOINS

- Cross Join
  - Retorna o produto cartesiano de todas as tabelas participantes da junção. Não requer a condição de junção. A quantidade de linhas retornadas é $n \times m$, sendo n e m a quantidade de linhas das tabelas da esquerda e da direita, respectivamente.

```
SELECT  *
FROM Empregado E CROSS JOIN Departamento D
```

Tipos de JOINS

- Self Join
  - Usado para a junção (Inner, Outer ou Cross) de uma tabela com ela mesma, quando a tabela possui um campo de chave estrangeira referenciando sua própria chave primária. Porém, não existe uma cláusula “self” no SQL.

```
SELECT  *
FROM Empregado E INNER JOIN Empregado Emp ON E.CodSupervisor = Emp.MatEmp
```

Observe que está sendo feito um join com a própria tabela

Junção (Join) em mais de duas Tabelas

É possível fazer junção entre várias tabelas: retorne o nome dos empregados que trabalham em algum projeto e o nome de todos os projetos em que eles trabalham

```
SELECT E.NomEmp, P.ProjNome
FROM Empregado E INNER JOIN Trabalha_Em T ON E.MatEmp = T.MatEmp
INNER JOIN Projeto P ON T.CodProjeto = P.CodProjeto
```

Desafios - Linguagem SQL - Realizando Consultas

Desafios - Linguagem SQL - Realizando Consultas

Utilize o Oracle Apex para executar o script fornecido no material de aula.

modelo_relacional.sql

Consulta 1

Consulta 1

Escreva um comando em SQL para retornar o nome dos empregados que são do departamento "Pesquisa" e a quantidade de horas trabalhadas destes empregados no projeto "Tecnologia" e ordena pela quantidade de horas trabalhadas

Consulta 1

```sql
SELECT E.NomEmp, TE.Horas
FROM Empregado E
JOIN Departamento D ON E.CodDepto = D.CodDepto
JOIN Trabalha_Em TE ON E.MatEmp = TE.MatEmp
JOIN Projeto P ON TE.CodProjeto = P.CodProjeto
WHERE D.DeptoNome = 'Pesquisa' AND P.ProjNome = 'Tecnologia'
ORDER BY TE.Horas DESC;
```

Consulta 2

Consulta 2

Escreva um consulta em SQL que retorne a quantidade de horas total trabalhada dos projeto agrupada por departamento. retornar apenas os departamentos com o total de horas trabalhadas acimas de 40

Consulta 2

```sql
SELECT D.DeptoNome, SUM(TE.Horas) AS TotalHoras
FROM Empregado E
JOIN Departamento D ON E.CodDepto = D.CodDepto
JOIN Trabalha_Em TE ON E.MatEmp = TE.MatEmp
JOIN Projeto P ON TE.CodProjeto = P.CodProjeto
GROUP BY D.DeptoNome
HAVING SUM(TE.Horas) > 40;
```

Links para saber mais:

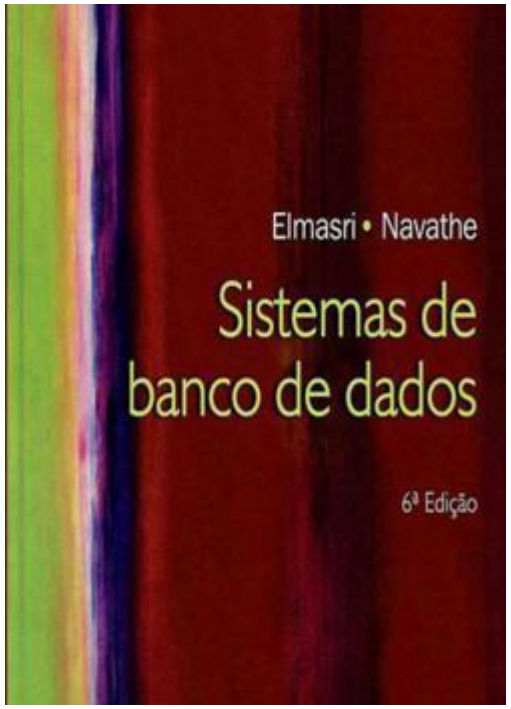
Microsoft T-SQL

https://docs.microsoft.com/pt-br/sql/t-sql/lesson-1-creating-database-objects?view=sql-server-ver15

Oracle PL/SQL

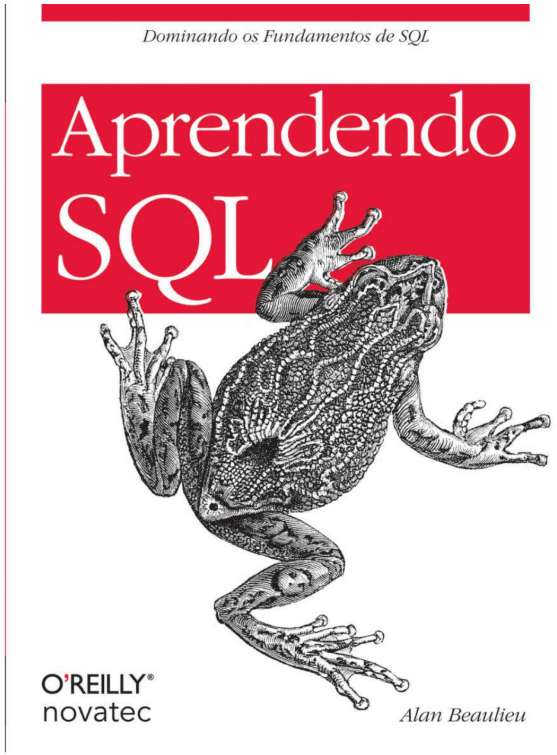
https://www.oracle.com/br/database/technologies/appdev/plsql.html

Principais Referências



Navathe. **Sistemas de Banco de Dados** – 2011, 6ª Ed.

Principais Referências



Beaulieu, Alan. **Aprendendo SQL: Dominando os Fundamentos de SQL** – 2010, 1ª Ed.