INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO	CAMPUS São Paulo
Banco de Dados	**Aula 3**
Professor: Francisco Veríssimo Luciano **E-mail: fvluciano@hotmail.com**	

3 - Aula 10 - Modelo Físico de Banco de Dados - Parte III

3.1 – Criando mais algumas tabelas e relembrando alguns comandos

3.1.1 – Comando CREATE

Crie inicialmente um banco de dados com o nome de EMPRESA

```
-- CRIAÇÃO DE TABELAS
CREATE TABLE EMPREGADO (
NOME       VARCHAR(30)     NOT NULL,
NSS        INTEGER         NOT NULL,
CPF        NUMERIC(11)     NOT NULL,
ENDERECO   VARCHAR(50),
CARGO      VARCHAR(20),
SALARIO    DECIMAL(10,2),
COD_DEPTO  INTEGER         NOT NULL,
-- AS CHAVES PK E FK, PODEM SER DECLARADAS NO FINAL DO BLOCO
PRIMARY KEY (NSS),
FOREIGN KEY (COD_DEPTO) REFERENCES DEPARTAMENTO(CODIGO)
)
```

A criação da tabela acima funcionou ???

E se criarmos a tabela Departamento antes ???

```
CREATE TABLE DEPARTAMENTO (
NOME            VARCHAR(30)     NOT NULL,
CODIGO          INTEGER         NOT NULL,
GNSS            INTEGER,
GDATAINICIO     DATE,
-- AS CHAVES PK E FK, PODEM SER DECLARADAS NO FINAL DO BLOCO
PRIMARY KEY (CODIGO),
FOREIGN KEY (GNSS) REFERENCES EMPREGADO(NSS)
)
```

Também deu erro ???

Solução: Crie a tabela DEPARTAMENTO (com ou sem FK??), e vamos adicionar a foreign key depois de criarmos a tabela EMPREGADO.

```
CREATE TABLE EMPREGADO (
NOME       VARCHAR(30)     NOT NULL,
NSS        INTEGER         NOT NULL,
CPF        NUMERIC(11)     NOT NULL,
ENDERECO   VARCHAR(50),
CARGO      VARCHAR(20),
SALARIO    DECIMAL(10,2),
COD_DEPTO  INTEGER         NOT NULL,
PRIMARY KEY (NSS)
)
```

Após a execução do comando SQL acima, use o comando `EXEC sp_columns EMPREGADO` ou então `SELECT * FROM EMPREGADO`, para visualizar a estrutura da tabela criada.

3.1.2 – Comando ALTER

Agora vamos usar uma instrução para alterar a tabela e incluir a chave estrangeira (FK), o ALTER TABLE - inclui/remove definições de colunas e restrições:

ALTER TABLE tabela <ação>;
<ação>:
ADD novo_atrib tipo [<restrições de coluna>]
ADD [CONSTRAIN nome] <restrição de tabela>
DROP atributo
DROP CONSTRAINT nome

Vamos alterar a tabela EMPREGADO, adicionando a foreign key que referencia DEPARTAMENTO

```
ALTER TABLE EMPREGADO ADD CONSTRAINT FK_Gerente
FOREIGN KEY (COD_DEPTO) REFERENCES DEPARTAMENTO(CODIGO)
```

3.1.3 – Comando DROP

DROP TABLE - exclui uma tabela do banco de dados

Sintaxe:
DROP TABLE nome_tabela [CASCADE | RESTRICT];
CASCADE: todas as visões e restrições que referenciam a tabela são removidas automaticamente
RESTRICT: a tabela é removida somente se não for referenciada em nenhuma restrição ou visão

Exemplificando:

- ✓ Tente apagar a tabela DEPARTAMENTO usando: DROP TABLE DEPARTAMENTO
- ✓ Agora use o comando DROP com CASCADE e verifique que a tabela EMPREGADO perdeu a referência para o campo COD_DEPTO

3.1.4 – Comando SELECT

O comando SELECT recupera dados de uma ou mais tabelas. A sua sintaxe mais simples pode ser resumida da seguinte forma:

```
SELECT lista_de_colunas
FROM lista_de_tabelas
WHERE condições
```

A lista_de_colunas especifica quais colunas serão retornadas como resultado, separadas por vírgulas ou um asterisco (*) que indica todas as colunas da tabela. A cláusula FROM, com uma lista_de_tabelas, especifica quais tabelas serão consultadas. A cláusula WHERE especifica condições que devem ser satisfeitas pelas linhas das tabelas.

A partir de agora é necessária a instalação de um Banco de Dados para exemplos. Será utilizado o banco de dados de exemplo Pubs, que é um BD bem complete, utilizado para treinamento. Assim, solicite ao professor o script do banco de PUBS – está disponível no Moodle da disciplina ou pode ser baixado gratuitamente na internet, para que sejam executados os comandos SELECT a seguir:

-- consultando todas as colunas da tabela

```
Select * from authors
```

-- Consultando colunas específicas de uma Tabela

```
Select au_id, au_fname, au_lname
from authors
```

-- Consultando colunas com alias

```
Select au_id as Identif, au_fname as Nome, au_lname as Sobrenome
from authors
```

-- Consultando com Condições

```
Select au_fname, au_lname, city, state
from authors
where state='CA'
```

-- Manipulando expressões

Um comando SELECT pode retornar nas colunas de resultado uma coluna da tabela, ou um valor calculado. Por exemplo, a tabela titles contém os títulos de livro (title) e os preços de cada um (price). Se quisermos ver como fica o preço de cada um após um aumento de 10%, pode ser feito o seguinte:

```
select price Preço , (price * 1.1) "Preço com 10% de aumento", title
from titles
```

-- Funções matemáticas

Além de operadores, você pode usar funções matemáticas do SQL Server, por exemplo:

- ✓ ABS(valor) retorna o valor absoluto (sem sinal) de um item.
- ✓ POWER(valor,p)retorna o valor elevado à potência p.

- ✓ ROUND(valor,n) arredonda o valor para n casas decimais.
- ✓ SQRT (valor) retorna a raiz quadrada do valor especificado.
- ✓ PI valor constante 3.141592563589793

Veja exemplos:

```
Select ABS(-13)

Select POWER(2,3)

Select SQRT(144)
```

Para arredondar o valor do preço de cada livro para duas casas decimais, pode ser feito o seguinte:

```
Select price Preço, ROUND (price, 2) "Preço com 2 casas decimais", title Título
from titles
```

Resultados | Mensagens

	Preço	Preço com 1 casa decimal	Título
1	19,99	20,00	The Busy Executive's Database Guide
2	11,95	12,00	Cooking with Computers: Surreptitious Balance Sheets
3	2,99	3,00	You Can Combat Computer Stress!
4	19,99	20,00	Straight Talk About Computers
5	19,99	20,00	Silicon Valley Gastronomic Treats
6	2,99	3,00	The Gourmet Microwave
7	NULL	NULL	The Psychology of Computer Cooking
8	22,95	23,00	But Is It User Friendly?
9	20,00	20,00	Secrets of Silicon Valley
10	NULL	NULL	Net Etiquette
11	21,59	21,60	Computer Phobic AND Non-Phobic Individuals: Beha...
12	10,95	11,00	Is Anger the Enemy?
13	7,00	7,00	Life Without Fear
14	19,99	20,00	Prolonged Data Deprivation: Four Case Studies
15	7,99	8,00	Emotional Security: A New Algorithm
16	20,95	21,00	Onions, Leeks, and Garlic: Cooking Secrets of the M...
17	11,95	12,00	Fifty Years in Buckingham Palace Kitchens
18	14,99	15,00	Sushi, Anyone?

-- Funções de caracteres

Você pode usar funções para manipular dados do tipo caracter (char ou varchar), por exemplo, para pegar uma substring de uma seqüência de caracteres. E você pode usar o operador + para concatenar dois valores de tipo caracter. Por exemplo, digite o seguinte comando:

```
select au_fname + ' ' + au_lname 'Nome completo', city + ', ' + state 'Cidade'
from authors
```

O que fizemos foi concatenar o primeiro nome e segundo nome do autor, inserindo um espaço no meio (au_fname + ' ' + au_lname), para gerar a coluna Nome completo e depois juntar o nome da cidade com o nome do estado, inserindo uma vírgula (city + ', '+state).

Existem várias funções de manipulação de strings que podem ser usadas para outras tarefas, por exemplo:

- ✓ ASCII(caractere) retorna o código ASCII de um caractere.
- ✓ CHAR(inteiro) retorna o caractere, dado o seu código ASCII
- ✓ LOWER(expr) converte para minúsculas
- ✓ UPPER(expr) converte para maiúsculas
- ✓ LTRIM(expr) retira espaços à esquerda
- ✓ RTRIM(expr) retira espaços à direita
- ✓ REPLICATE(expr, n) repete uma expressão n vezes
- ✓ SUBSTRING(expr,início,tamanho) extrai uma parte de uma string desde início e com tamanho caracteres

Veja um exemplo:

```
Select substring(title, 1, 30) Título , str(price, 5, 1) Preço
from titles
```

O que fizemos foi mostrar até 30 caracteres da coluna title e mostramos a coluna price com no máximo 5 números antes da vírgula e 1 casa decimal depois da vírgula. Então o resultado será:

Resultados | Mensagens

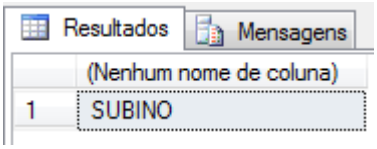
	Título	Preço
1	The Busy Executive's Database	20.0
2	Cooking with Computers: Surrep	11.9
3	You Can Combat Computer Stress	3.0
4	Straight Talk About Computers	20.0
5	Silicon Valley Gastronomic Tre	20.0
6	The Gourmet Microwave	3.0
7	The Psychology of Computer Coo	NULL
8	But Is It User Friendly?	22.9
9	Secrets of Silicon Valley	20.0
10	Net Etiquette	NULL
11	Computer Phobic AND Non-Phobic	21.6
12	Is Anger the Enemy?	10.9
13	Life Without Fear	7.0
14	Prolonged Data Deprivation: Fo	20.0
15	Emotional Security: A New Algo	8.0
16	Onions, Leeks, and Garlic: Coo	20.9
17	Fifty Years in Buckingham Pala	11.9
18	Sushi, Anyone?	15.0

-- Outras funções de caracteres

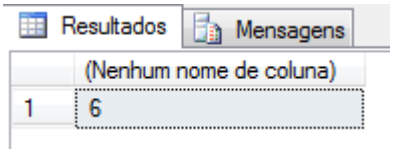
- ✓ RIGHT(*expr,n*) retorna *n* caracteres à direita da string
- ✓ REVERSE(*expr*) inverte uma string
- ✓ CHARINDEX(*'caractere', expr*) retorna a posição de um caractere dentro da string
- ✓ SPACE(*n*) retorna uma string com *n* espaços
- ✓ STR(*número,n,d*) converte um valor numérico para string, formatado
- ✓ com *n* caracteres na parte inteira (antes da vírgula)
- ✓ e *d* casas decimais depois da vírgula.
- ✓ STUFF(*expr1,início,tamanho,expr2*) substitui em *expr1*, os caracteres desde *início* até
- ✓ *tamanho* por *expr2*

✓ DATALENGTH(*expr*) retorna a quantidade de caracteres em *expr*
Veja exemplos:

```
select REVERSE('ONIBUS')
```




```
select DATALENGTH('ESCOLA')
```



-- Condições de Pesquisa

Como vimos, a cláusula WHERE permite selecionar quais as linhas da tabela a serem incluídas no resultado. Existem várias formas de montar uma cláusula WHERE, usando um dos seguintes elementos - Operadores de comparação:

= igual a
> maior que
< menor que
>= ou !< maior ou igual (não menor)
<= ou !> menor ou igual (não maior)
<> ou != diferente

Faixas: BETWEEN <valor inicial> AND <valor final>
Listas: IN (<lista>)
Casamento de padrões: LIKE <padrão>
Valores nulos: IS NULL, IS NOT NULL
Combinações de condições: AND, OR, NOT

Veja exemplos:

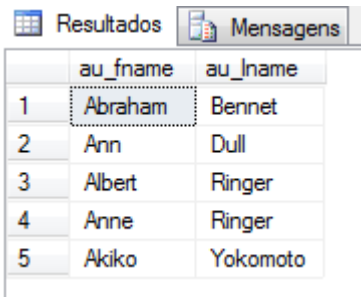
```
select au_lname, city
from authors
where state = 'CA'

select au_lname, city
from authors
where state <> 'CA'


select pubdate, title
from titles
where pubdate between '1/1/91' and '2/2/91'
```

```
select au_lname, city, state
from authors
where state in ('UT','CA')

select au_fname, au_lname from authors
where au_fname like 'A%'
```



Resultados | Mensagens

	au_fname	au_lname
1	Abraham	Bennet
2	Ann	Dull
3	Albert	Ringer
4	Anne	Ringer
5	Akiko	Yokomoto

```
select au_fname, au_lname from authors
          where au_fname like '%en%'
```

	au_fname	au_lname
1	Charlene	Locksley

-- Juntando várias condições

Você pode fazer condições compostas com AND, OR ou NOT. O operador AND (E) liga duas condições e retorna verdadeiro apenas se ambas são verdadeiras e falso se pelo menos uma delas é falsa. Já o Operador OR (OU) retorna verdadeiro se pelo menos uma delas for verdadeira e falso se ambas forem falsas. O operador NOT (NÃO) inverte uma condição.

```
select title, pub_id, price
from titles
where (title like 'T%' OR pub_id = '0877') AND (price > $16.00)
```

Isso retorna os livros quando ambas as condições (1 e 2) são verdadeiras:

Condição 1: uma das seguintes afirmações é verdadeira o título ('title') começa com T,OU o código da editora ('pub_id') é '0877'
E
Condição 2: o preço ('price') é maior que 16.00

Os parênteses indicam a precedência das condições. Na falta de parênteses, o operador NOT, se presente, é aplicado primeiro, depois as condições com AND são agrupadas, depois as condições com OR são agrupadas. Você pode usar parênteses, mesmo se não necessários, para tornar a expressão mais legível.

-- Eliminando valores duplicados

Se você quiser saber quais as cidades e estados nas quais mora algum autor, pode usar a seguinte consulta:

```
select city, state from authors
```

Mas note que algumas cidades, como Oakland, CA, aparecem várias vezes. Para eliminar duplicações, use a cláusula DISTINCT. O SQL leva em conta as duas colunas em conjunto, para remover duplicatas

```
select DISTINCT city, state from authors
```

-- Ordenando resultados

Para ver o resultado numa ordem particular, use a cláusula ORDER BY. Se estiver presente, deve ser a última cláusula do comando SELECT. Por exemplo, para ver os livros em ordem de preço:

```
select title, type, price
from titles
order by price
```

-- União de conjuntos

O comando SELECT retorna um conjunto de linhas, e permite também operações com a noção matemática de conjuntos. Por exemplo, o resultado de dois comandos SELECT pode ser combinado com o operador UNION. Os dois comandos podem até mesmo trazer dados de tabelas diferentes, desde que com o mesmo nº de colunas e tipos de dados compatíveis para cada coluna correspondente de um com o outro.

Por exemplo, no banco de dados pubs, a tabela authors contém informação sobre cada autor, o que inclui a cidade e estado onde ele mora (colunas city e state). A tabela publishers contém informação sobre as editoras e suas cidades e estados. Para sabermos o conjunto de todas as cidades onde existem autores ou editoras, pode ser feita uma união dos dois conjuntos, com:

```
select city, state from authors
union
select city, state from publishers
```

	city	state
1	Ann Arbor	MI
2	Berkeley	CA
3	Boston	MA
4	Chicago	IL
5	Corvallis	OR
6	Covelo	CA
7	Dallas	TX
8	Gary	IN
9	Lawrence	KS
10	Menlo Park	CA
11	Mnchen	NULL
12	Nashville	TN
13	New York	NY
14	Oakland	CA
15	Palo Alto	CA
16	Paris	NULL

Note que na união de dois conjuntos, os elementos repetidos são eliminados, como quando se usa o DISTINCT. O resultado também aparece em ordem crescente, pois o SQL Server ordena os resultados antes de eliminar repetições. Se você quer ordenar de modo diferente os resultados da união, usando ORDER BY, essa cláusula só pode aparecer no segundo comando SELECT, ou seja, no final dos comandos, por exemplo:

```
select city, state from authors
union
select city, state from publishers
order by state
```

	city	state
1	Mnchen	NULL
2	Paris	NULL
3	Berkeley	CA
4	Covelo	CA
5	Menlo Park	CA
6	Oakland	CA
7	Palo Alto	CA
8	San Francisco	CA
9	San Jose	CA
10	Vacaville	CA
11	Walnut Creek	CA
12	Washington	DC
13	Chicago	IL
14	Gary	IN
15	Lawrence	KS
16	Boston	MA

-- Um pouco mais sobre SQL

Para saber quantas linhas existem na tabela 'publishers' pode-se usar a função agregada COUNT(*):

```
select count(*) from publishers
```

	(Nenhum nome de coluna)
1	8

Você pode também acrescentar uma condição:

```
select count(*) from publishers where state is not null
```

	(Nenhum nome de coluna)
1	6

Mais recursos e aplicações SQL:

Antes dos próximos assuntos, vamos criar um Banco de dados TESTE e as tabelas DEPARTAMENTO e FUNCIONARIO

```
CREATE DATABASE TESTE
USE TESTE

CREATE TABLE DEPARTAMENTO (
CODDEPARTAMENTO INT PRIMARY KEY,
NOMEDEPTO VARCHAR(50)
)
CREATE table FUNCIONARIO(
CODFUNCIONARIO int PRIMARY KEY,
NOME VARCHAR(50),
SALARIO DECIMAL(10,2),
CODDEPARTAMENTO INT,
FOREIGN KEY (coddepartamento) references departamento (coddepartamento)
)
INSERT INTO DEPARTAMENTO VALUES (1,'VENDAS')
INSERT INTO DEPARTAMENTO VALUES (2,'RECURSOS HUMANOS')
INSERT INTO FUNCIONARIO VALUES (1,'FUNCIONARIO 1',2500,2)
INSERT INTO FUNCIONARIO VALUES (2,'FUNCIONARIO 2',4850,1)
INSERT INTO FUNCIONARIO VALUES (3,'FUNCIONARIO 3',5400,2)
INSERT INTO FUNCIONARIO VALUES (4,'FUNCIONARIO 4',3950,2)
```

--Funções agregadas

Além da função COUNT, existem outras funções agregadas que podem ser usadas para fazer operações sobre os elementos do grupo:

- **AVG(expr)**
Calcula o valor médio da expressão expr dentro do grupo. A expressão pode ser um nome de coluna ou calculada a partir de colunas e/ou constantes. (Por exemplo, AVG (salario*1.1))

- **COUNT(expr)**
Conta quantos valores existem da expressão dada dentro do grupo (se expr for NULL para uma linha, a linha não é incluída na contagem).

- **COUNT(*)**
Conta quantas linhas existem dentro do grupo.

- **MAX(expr)**
Retorna o máximo valor de expr dentro do grupo.

- **MIN(expr)**
Retorna o mínimo valor de expr dentro do grupo.

- **SUM(expr)**
Retorna o somatório da expressão dentro do grupo

As funções AVG e SUM podem ser usadas apenas com dados numéricos. As outras podem ser usadas com qualquer tipo de coluna.

As funções SUM, AVG e COUNT(expr) permitem especificar também o operador DISTINCT, que indica para considerar apenas os valores distintos. Por exemplo, a tabela 'funcionarios' do banco

de dados Exemplo contém informação sobre qual departamento este funcionário trabalha. Para saber quais os departamentos que têm ao menos um funcionario, execute o seguinte comando:

```
select count(distinct coddepartamento)
from funcionario
```

-- Clausula GROUP BY

Quando se utiliza o GROUP BY com o nome de uma coluna, os resultados são agregados por essa coluna. Por exemplo, para saber quantos funcionários existem por departamento:

```
select count(*) 'Quantidade Funcionário', CodDepartamento
from funcionario
GROUP BY CodDepartamento
```

Vera outro exemplo:

```
select count(*) 'Quantidade de Livros', pub_id as Editor
from titles
GROUP BY pub_id
```

A cláusula GROUP BY agrupa valores baseando-se em uma ou mais colunas. No último SELECT, GROUP BY coddepartamento significa que todas as linhas que têm o mesmo valor da coluna 'coddepartamento' serão agrupadas em uma só.
Uma função agregada, como SUM, COUNT, AVG calcula valores sobre todos os elementos do grupo. Uma linha de resumo é gerada para o grupo, contendo o valor representante do grupo, 'coddepartamento' e o resultado de SUM(Salario).
Note que as colunas de resultado da cláusula SELECT(a lista de colunas após o SELECT) podem ser apenas:

- Uma coluna presente na lista do GROUP BY

OU

- Um valor gerado por uma função agregada Outras colunas não podem ser incluídas no resultado, porque teriam valores diferentes para cada linha do grupo.

-- Usando a cláusula HAVING

Após feito o agrupamento, pode-se usar a cláusula HAVING para selecionar quais os grupos a serem incluídos no resultado. Por exemplo, para selecionar os departamentos que pagam mais que 1000.00, pode-se fazer:

```
select coddepartamento Departamento, sum(Salario) 'Salário
Total' from funcionario
group by coddepartamento
having sum(Salario) > 1000.00
```

Note que as cláusulas WHERE e HAVING são semelhantes. Mas WHERE seleciona as linhas da tabela que irão participar da geração do resultado. Essas linhas serão agrupadas e depois HAVING é aplicado ao resultado de cada grupo, para saber quais

grupos vão aparecer no resultado. Nas condições usadas por HAVING só podem aparecer valores que sejam os mesmos em todos os elementos do grupo.

-- Junções de tabelas

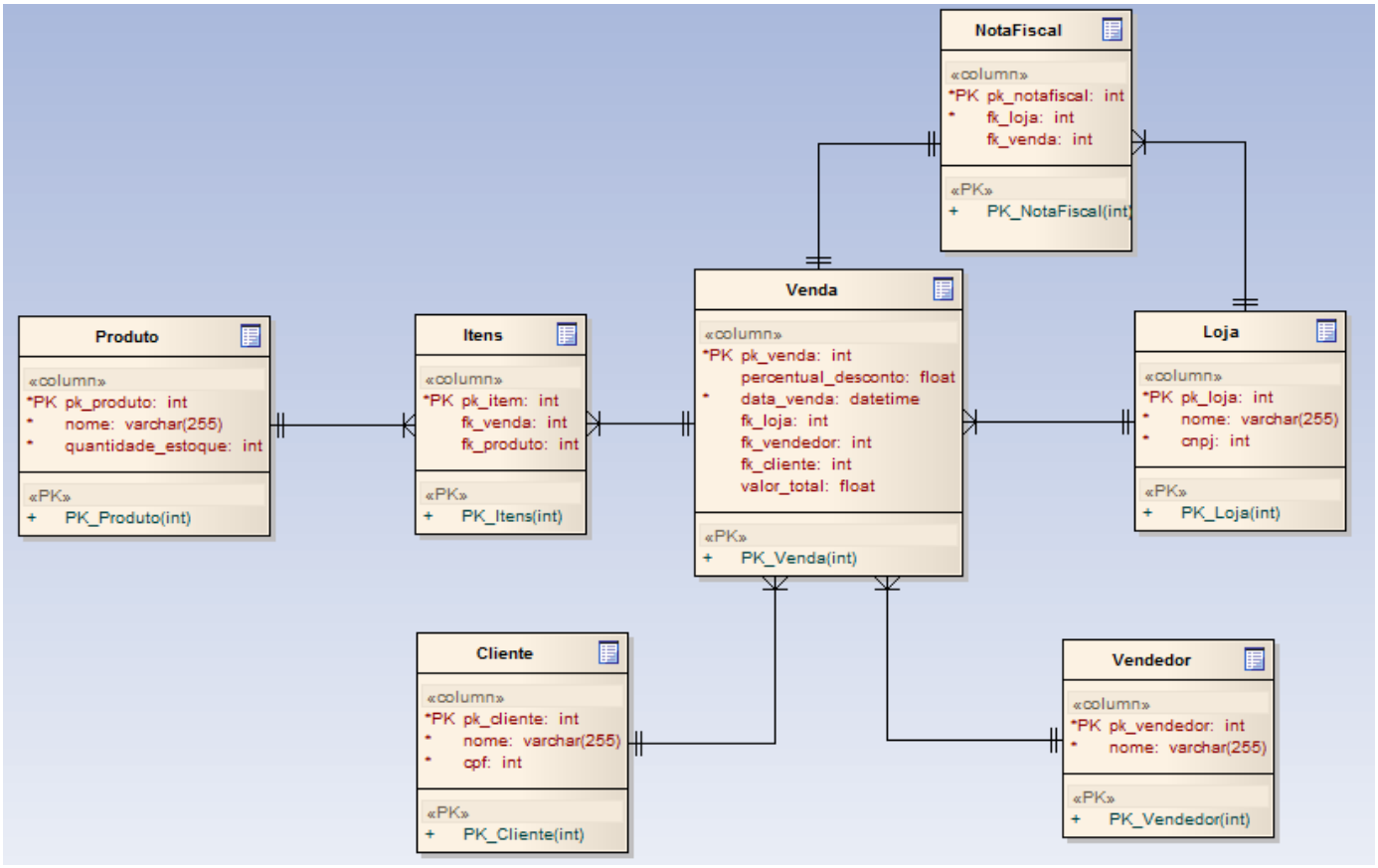
Um comando SELECT também pode fazer uma consulta que traz dados de duas ou mais tabelas. Esse é um processo chamado de junção [join]. As tabelas têm uma coluna em comum que é usado para fazer as junções.

```
select departamento.nome AS 'Departamento', funcionario.nome
from funcionario, departamento
where funcionario.coddepartamento = departamento.coddepartamento
```

Por enquanto, basta entender que uma *Join* (junção) é uma maneira de se ligar as tabelas em uma instrução SQL. Este tema será retomado mais adiante, envolvendo as suas variações e diferentes aplicações.

3.2 Modelo de Caso

Para esta parte da aula, o BD abaixo será utilizado, portanto, recomenda-se executar o script para criação.



```sql
CREATE DATABASE bd_vendas;

USE bd_vendas;

CREATE TABLE tb_cliente (
	pk_cliente int PRIMARY KEY IDENTITY(0,1),
	nome varchar(255) NOT NULL,
	cpf varchar(11) NOT NULL
)
;

CREATE TABLE tb_vendedor (
	pk_vendedor int PRIMARY KEY IDENTITY(0,1),
	nome varchar(255) NOT NULL
)
;

CREATE TABLE tb_produto (
	pk_produto int PRIMARY KEY IDENTITY(0,1),
	nome varchar(255) NOT NULL,
	quantidade_estoque int NOT NULL
)
;

CREATE TABLE tb_loja (
	pk_loja int PRIMARY KEY IDENTITY(0,1),
	nome varchar(255) NOT NULL,
	cnpj varchar(14) NOT NULL
)
;

CREATE TABLE tb_venda (
	pk_venda int PRIMARY KEY IDENTITY(0,1),
	percentual_desconto float,
	data_venda datetime NOT NULL,
	fk_loja int REFERENCES tb_loja(pk_loja),
	fk_vendedor int REFERENCES tb_vendedor(pk_vendedor),
	fk_cliente int REFERENCES tb_cliente(pk_cliente),
	valor_total float,
)
;

CREATE TABLE tb_NotaFiscal (
	pk_notafiscal int PRIMARY KEY IDENTITY(0,1),
	fk_loja int NOT NULL,
	fk_venda int
)
;

CREATE TABLE tb_itens (
	pk_item int PRIMARY KEY IDENTITY(0,1),
	fk_venda int,
	fk_produto int
)
;

INSERT INTO tb_produto VALUES ('keyboard',20);
INSERT INTO tb_produto VALUES ('monitor',50);
INSERT INTO tb_produto VALUES ('mouse',100);
INSERT INTO tb_produto VALUES ('mouse pad',200);
INSERT INTO tb_produto VALUES ('mouse wireless',50);
INSERT INTO tb_produto VALUES ('pen drive 2GB',200);
INSERT INTO tb_produto VALUES ('pen drive 8GB',200);
INSERT INTO tb_produto VALUES ('pen drive 16GB',200);
```

```
INSERT INTO tb_cliente VALUES ('Joao Pedro Neves','11122233300');
INSERT INTO tb_cliente VALUES ('Joao Botelho Alves','11122233301');
INSERT INTO tb_cliente VALUES ('Maria da Silva Soares','11122233302');
INSERT INTO tb_cliente VALUES ('Ana Maria Rocha','11122233303');
INSERT INTO tb_cliente VALUES ('Cristina Maria Ana Silva','11122233304');
INSERT INTO tb_cliente VALUES ('Carlos Augusto Vieira','11122233305');
INSERT INTO tb_cliente VALUES ('Marcelo Lopes Vieira','11122233306');
INSERT INTO tb_cliente VALUES ('Joao Jose da Silva','11122233307');
INSERT INTO tb_cliente VALUES ('Carlos Magno Monteiro','11122233308');
INSERT INTO tb_cliente VALUES ('Jose Maria da Silva','11122233309');
INSERT INTO tb_cliente VALUES ('Marta Vieira Nunes','11122233310');
INSERT INTO tb_cliente VALUES ('Carla Carolina Mendes','11122233311');
INSERT INTO tb_cliente VALUES ('Daiana Moreira Silva','11122233312');

INSERT INTO tb_vendedor VALUES ('Daniel Oliveira');
INSERT INTO tb_vendedor VALUES ('Marcos Cunha');
INSERT INTO tb_vendedor VALUES ('Olivia Marta');
INSERT INTO tb_vendedor VALUES ('Carlos Dias');
INSERT INTO tb_vendedor VALUES ('Pedro Pedreira');
INSERT INTO tb_vendedor VALUES ('GUSTAVO MOTTA');
INSERT INTO tb_vendedor VALUES ('ADILSON EVANDRO');


INSERT INTO tb_loja VALUES ('Loja Unidade Minas Gerais','11111111110');
INSERT INTO tb_loja VALUES ('Loja Unidade São Paulo','11111111111');
INSERT INTO tb_loja VALUES ('Loja Unidade Rio de Janeiro','11111111112');
INSERT INTO tb_loja VALUES ('Loja Unidade Epírito Santo','11111111113');

INSERT INTO tb_venda VALUES (3.5,convert(datetime,'18-06-12 10:34:09 PM',5),1,1,0,0)
INSERT INTO tb_venda VALUES (1.5,convert(datetime,'15-06-12 10:34:09 PM',5),2,2,1,0)
INSERT INTO tb_venda VALUES (2.5,convert(datetime,'13-06-12 10:34:09 PM',5),0,3,2,0)
INSERT INTO tb_venda VALUES (1.5,convert(datetime,'12-06-12 10:34:09 PM',5),1,4,3,0)
INSERT INTO tb_venda VALUES (1.5,convert(datetime,'15-06-12 10:34:09 PM',5),1,0,4,0)
INSERT INTO tb_venda VALUES (1.5,convert(datetime,'19-06-12 10:34:09 PM',5),1,1,5,0)
INSERT INTO tb_venda VALUES (1.5,convert(datetime,'20-06-12 10:34:09 PM',5),2,2,6,0)
INSERT INTO tb_venda VALUES (2.5,convert(datetime,'18-07-12 10:34:09 PM',5),2,3,7,0)
INSERT INTO tb_venda VALUES (2.5,convert(datetime,'11-07-12 10:34:09 PM',5),2,4,8,0)
INSERT INTO tb_venda VALUES (2.5,convert(datetime,'10-07-12 10:34:09 PM',5),0,0,9,0)
INSERT INTO tb_venda VALUES (3.5,convert(datetime,'28-07-12 10:34:09 PM',5),0,1,1,0)
INSERT INTO tb_venda VALUES (3.5,convert(datetime,'1-05-13 10:34:09 PM',5),0,1,2,0)
INSERT INTO tb_venda VALUES (2.5,convert(datetime,'8-05-13 10:34:09 PM',5),1,1,2,0)
INSERT INTO tb_venda VALUES (3.5,convert(datetime,'12-05-13 10:34:09 PM',5),2,2,3,0)
INSERT INTO tb_venda VALUES (13.5,convert(datetime,'18-05-13 10:34:09 PM',5),0,2,3,0)
INSERT INTO tb_venda VALUES (3.5,convert(datetime,'28-05-13 10:34:09 PM',5),1,2,5,0)
INSERT INTO tb_venda VALUES (23.5,convert(datetime,'19-08-13 10:34:09 PM',5),2,3,5,0)
INSERT INTO tb_venda VALUES (33.5,convert(datetime,'20-08-13 10:34:09 PM',5),0,3,5,0)
INSERT INTO tb_venda VALUES (43.5,convert(datetime,'21-09-13 10:34:09 PM',5),1,4,7,0)
INSERT INTO tb_venda VALUES (53.5,convert(datetime,'18-10-13 10:34:09 PM',5),2,4,7,0)
INSERT INTO tb_venda VALUES (63.5,convert(datetime,'11-11-13 10:34:09 PM',5),0,4,7,0)

INSERT INTO tb_itens VALUES (0,1)
INSERT INTO tb_itens VALUES (1,1)
INSERT INTO tb_itens VALUES (2,1)
INSERT INTO tb_itens VALUES (3,1)
INSERT INTO tb_itens VALUES (4,1)
INSERT INTO tb_itens VALUES (5,1)
INSERT INTO tb_itens VALUES (6,1)
INSERT INTO tb_itens VALUES (7,1)
INSERT INTO tb_itens VALUES (8,1)
INSERT INTO tb_itens VALUES (9,1)
INSERT INTO tb_itens VALUES (10,1)
INSERT INTO tb_itens VALUES (11,1)
INSERT INTO tb_itens VALUES (12,1)
```

```
INSERT INTO tb_itens VALUES (13,1)
INSERT INTO tb_itens VALUES (14,1)
INSERT INTO tb_itens VALUES (15,1)
INSERT INTO tb_itens VALUES (16,1)
INSERT INTO tb_itens VALUES (17,1)
INSERT INTO tb_itens VALUES (18,1)
INSERT INTO tb_itens VALUES (19,1)
INSERT INTO tb_itens VALUES (20,1)
INSERT INTO tb_itens VALUES (21,1)
INSERT INTO tb_itens VALUES (0,2)
INSERT INTO tb_itens VALUES (1,2)
INSERT INTO tb_itens VALUES (2,2)
INSERT INTO tb_itens VALUES (3,2)
INSERT INTO tb_itens VALUES (4,2)
INSERT INTO tb_itens VALUES (5,2)
INSERT INTO tb_itens VALUES (6,2)
INSERT INTO tb_itens VALUES (7,2)
INSERT INTO tb_itens VALUES (8,2)
INSERT INTO tb_itens VALUES (9,2)
INSERT INTO tb_itens VALUES (10,2)
INSERT INTO tb_itens VALUES (11,2)
INSERT INTO tb_itens VALUES (12,2)
INSERT INTO tb_itens VALUES (13,2)
INSERT INTO tb_itens VALUES (14,2)
INSERT INTO tb_itens VALUES (15,2)
INSERT INTO tb_itens VALUES (16,2)
INSERT INTO tb_itens VALUES (17,2)
INSERT INTO tb_itens VALUES (18,2)
INSERT INTO tb_itens VALUES (19,2)
INSERT INTO tb_itens VALUES (20,2)
INSERT INTO tb_itens VALUES (21,2)
INSERT INTO tb_itens VALUES (0,3)
INSERT INTO tb_itens VALUES (1,4)
INSERT INTO tb_itens VALUES (2,5)
INSERT INTO tb_itens VALUES (3,6)
INSERT INTO tb_itens VALUES (4,7)
INSERT INTO tb_itens VALUES (5,0)
INSERT INTO tb_itens VALUES (6,1)
INSERT INTO tb_itens VALUES (7,2)
INSERT INTO tb_itens VALUES (8,3)
INSERT INTO tb_itens VALUES (9,4)
INSERT INTO tb_itens VALUES (10,5)
INSERT INTO tb_itens VALUES (11,6)
INSERT INTO tb_itens VALUES (12,7)
INSERT INTO tb_itens VALUES (13,0)
INSERT INTO tb_itens VALUES (14,1)
INSERT INTO tb_itens VALUES (15,2)
INSERT INTO tb_itens VALUES (16,3)
INSERT INTO tb_itens VALUES (17,4)
INSERT INTO tb_itens VALUES (18,5)
INSERT INTO tb_itens VALUES (19,6)
INSERT INTO tb_itens VALUES (20,7)
INSERT INTO tb_itens VALUES (21,0)
INSERT INTO tb_itens VALUES (0,1)
INSERT INTO tb_itens VALUES (1,2)
INSERT INTO tb_itens VALUES (2,3)
INSERT INTO tb_itens VALUES (3,4)
INSERT INTO tb_itens VALUES (4,5)
INSERT INTO tb_itens VALUES (5,6)
INSERT INTO tb_itens VALUES (6,7)
INSERT INTO tb_itens VALUES (7,0)
INSERT INTO tb_itens VALUES (8,1)
INSERT INTO tb_itens VALUES (9,2)
INSERT INTO tb_itens VALUES (10,3)
```

```
INSERT INTO tb_itens VALUES (11,4)
INSERT INTO tb_itens VALUES (12,5)
INSERT INTO tb_itens VALUES (13,6)
INSERT INTO tb_itens VALUES (14,7)
INSERT INTO tb_itens VALUES (15,0)
INSERT INTO tb_itens VALUES (16,1)
INSERT INTO tb_itens VALUES (17,2)
INSERT INTO tb_itens VALUES (18,3)
INSERT INTO tb_itens VALUES (19,4)
INSERT INTO tb_itens VALUES (20,5)
INSERT INTO tb_itens VALUES (21,6)
INSERT INTO tb_itens VALUES (0,7)
INSERT INTO tb_itens VALUES (1,0)
INSERT INTO tb_itens VALUES (2,1)
INSERT INTO tb_itens VALUES (3,2)
INSERT INTO tb_itens VALUES (4,3)
INSERT INTO tb_itens VALUES (5,4)
INSERT INTO tb_itens VALUES (6,5)
INSERT INTO tb_itens VALUES (7,6)
INSERT INTO tb_itens VALUES (8,7)
INSERT INTO tb_itens VALUES (9,0)
INSERT INTO tb_itens VALUES (10,1)
INSERT INTO tb_itens VALUES (11,2)
INSERT INTO tb_itens VALUES (12,3)
INSERT INTO tb_itens VALUES (13,4)
INSERT INTO tb_itens VALUES (14,5)
INSERT INTO tb_itens VALUES (15,3)
INSERT INTO tb_itens VALUES (16,6)
INSERT INTO tb_itens VALUES (17,7)
INSERT INTO tb_itens VALUES (18,0)
INSERT INTO tb_itens VALUES (19,1)
INSERT INTO tb_itens VALUES (20,4)
INSERT INTO tb_itens VALUES (21,5)


INSERT INTO tb_notafiscal VALUES (0,0)
INSERT INTO tb_notafiscal VALUES (0,1)
INSERT INTO tb_notafiscal VALUES (2,2)
INSERT INTO tb_notafiscal VALUES (2,3)
INSERT INTO tb_notafiscal VALUES (0,4)
INSERT INTO tb_notafiscal VALUES (2,5)
INSERT INTO tb_notafiscal VALUES (0,6)
INSERT INTO tb_notafiscal VALUES (1,7)
INSERT INTO tb_notafiscal VALUES (2,8)
INSERT INTO tb_notafiscal VALUES (0,9)
INSERT INTO tb_notafiscal VALUES (2,10)
INSERT INTO tb_notafiscal VALUES (2,11)
INSERT INTO tb_notafiscal VALUES (0,12)
INSERT INTO tb_notafiscal VALUES (2,13)
INSERT INTO tb_notafiscal VALUES (2,14)
INSERT INTO tb_notafiscal VALUES (0,15)
INSERT INTO tb_notafiscal VALUES (1,16)
INSERT INTO tb_notafiscal VALUES (2,17)
INSERT INTO tb_notafiscal VALUES (0,18)
INSERT INTO tb_notafiscal VALUES (1,19)
INSERT INTO tb_notafiscal VALUES (2,20)
INSERT INTO tb_notafiscal VALUES (0,21)
INSERT INTO tb_notafiscal VALUES (1,22)
INSERT INTO tb_notafiscal VALUES (1,23)
INSERT INTO tb_notafiscal VALUES (0,24)
INSERT INTO tb_notafiscal VALUES (1,25)
INSERT INTO tb_notafiscal VALUES (2,26)
INSERT INTO tb_notafiscal VALUES (1,27)
INSERT INTO tb_notafiscal VALUES (1,28)
```

```
INSERT INTO tb_notafiscal VALUES (2,29)
INSERT INTO tb_notafiscal VALUES (1,30)
INSERT INTO tb_notafiscal VALUES (1,31)
INSERT INTO tb_notafiscal VALUES (1,32)
INSERT INTO tb_notafiscal VALUES (0,33)
INSERT INTO tb_notafiscal VALUES (1,34)
INSERT INTO tb_notafiscal VALUES (1,35)
INSERT INTO tb_notafiscal VALUES (0,36)
```

3.3 - Consultas Avançadas

Apesar de a maioria das consultas feitas por quem está começando em SQL ser com SELECT externos, um recurso bastante útil que vai ajudar você a melhorar a legibilidade da sua query assim como, em alguns casos, otimizar o tempo do retorno das suas informações para o usuário, é representado pelas chamadas consultas aninhadas ou subconsultas.

Vale lembrar que uma subconsulta nada mais é do que uma instrução SELECT dentro de outro SELECT, que retorna algumas colunas específicas, que também são usadas em algumas funções como INSERT e UPDATE por exemplo.

Veja alguns exemplos de aplicações de instruções SELECT e Consultas Aninhadas com EXISTS, NOT EXISTS, IN e NOT IN, a partir do banco de dados `bd_vendas`, criado acima:

1. Verificar quais vendedores possuem vendas realizadas.

```
SELECT
	nome
FROM
	tb_vendedor
WHERE
	EXISTS (
			SELECT
				fk_vendedor
			FROM
				tb_venda
			WHERE
				fk_vendedor = pk_vendedor
				)
```

Outra forma de fazer a seleção sem usar seleção aninhada – os nomes ficarão em ordem alfabética:

```
SELECT
	distinct(tb_vendedor.nome)
FROM
tb_vendedor, tb_venda  --pode usar alias para as tabelas

WHERE
	 tb_vendedor.pk_vendedor = tb_venda.fk_vendedor
```

2. Verificar quais vendedores não possuem vendas realizadas

```
SELECT
	nome
FROM
	tb_vendedor
WHERE
	NOT EXISTS (
		SELECT
			fk_vendedor
		FROM
			tb_venda
		WHERE
			fk_vendedor = pk_vendedor
			)
```

3. Verificar quais vendedores realizaram vendas para a Loja de São Paulo

```
SELECT
	nome
FROM
	tb_vendedor
WHERE
	pk_vendedor IN (
		SELECT
			fk_vendedor
		FROM
			tb_venda
		WHERE
			fk_loja IN (
					SELECT
						pk_loja
					FROM
						tb_loja
					WHERE
						nome = 'Loja Unidade São Paulo'
					)
			)
```

4. Verificar quais vendedores realizaram vendas mas não para a Loja de São Paulo

```
SELECT
	nome
FROM
	tb_vendedor
WHERE
	pk_vendedor IN (
		SELECT
			fk_vendedor
		FROM
			tb_venda
		WHERE
			fk_loja NOT IN (
					SELECT
						pk_loja
					FROM
						tb_loja
					WHERE
						nome = 'Loja Unidade São Paulo'
					)
			)
```

Agrupar dados

1. Listar o número de vendas por cliente

```
SELECT
	c.nome, COUNT(v.pk_venda)AS qtd_venda
FROM
	tb_cliente AS c, tb_venda AS v
WHERE
	c.pk_cliente=v.fk_cliente
GROUP BY
	c.nome;
```

2. Listar o número de vendas por cliente que tenham realizado no mínimo 4 vendas.

```
SELECT
	c.nome, COUNT(v.pk_venda)AS qtd_venda
FROM
	tb_cliente AS c, tb_venda AS v
WHERE
	c.pk_cliente=v.fk_cliente
GROUP BY
	c.nome
HAVING
	COUNT(v.pk_venda)> 4  --equivale a quantidade de vendas (alias qtd_venda)
```

- **Junção de duas tabelas para um único resultado**

1. Listar todas as notas fiscais para a Loja Unidade Minas Gerais

```
SELECT
	l.nome, nf.pk_notafiscal
FROM
	tb_loja AS l, tb_notafiscal AS nf
WHERE
	l.pk_loja = nf.fk_loja AND
		l.nome = 'Loja Unidade Minas Gerais'
```

2. Listar todas as notas fiscais que não sejam da Loja São Paulo

```
SELECT
	l.nome, nf.pk_notafiscal
FROM
	tb_loja AS l, tb_notafiscal AS nf
WHERE
	l.pk_loja = nf.fk_loja AND
		l.nome <> 'Loja Unidade São Paulo'  -- poderia ser != (não igual)
```

Associações de tabelas podem ser utilizadas para diversas finalidades, como converter em informação os dados encontrados em duas ou mais tabelas.

Esse tipo de operação pode ser feito por meio das cláusulas WHERE e JOIN. Além disso, as tabelas podem ser combinadas por meio de uma condição ou um grupo de condições de junção. Por exemplo, podemos usar as chaves estrangeiras como condição para relacionar as tabelas.

Veremos a seguir, operações com estas clausulas de junções de tabelas.

Referências

MECENAS, Ivan. OLIVEIRA, Vivianne. **Banco de Dados: Do modelo conceitual à implementação física**. Rio de Janeiro: Alta Books, 2005. 180 p.

ELMASRI, Ramez E. & NAVATHE, Shamkant B, **Sistema de Bancos de Dados – Fundamentos e Aplicações**, São Paulo: ADDISON WESLEY BRASIL, 2005.

MACEDO, Diego. **Modelagem Conceitual, Lógica e Física de Dados**. 2011. Acessado em <14/02/2015>. Disponível em <http://www.diegomacedo.com.br/modelagem-conceitual-logica-e-fisica-de-dados/>.

NASCIMENTO, Manuela Cicco do. **Br2Oracle :Geração automática de esquema relacional a partir da ferramenta BrModelo para o SGBD Oracle**. Monografia para conclusão de graduação em Ciência da Computação. Orientador: Prof. Fernando da Fonseca de Souza. Recife, 30 de Janeiro de 2008. Universidade Federal de Pernambuco - Centro de Informática.

RABELO, Ricardo J. **Banco de Dados. MaterialDAS5316**. 2009. Acessado em <14/02/2015>. Disponível em <http://www.das.ufsc.br/~rabelo/Ensino/DAS5316/MaterialDAS5316/Banco_de_Dados/bd_intro.pdf>.

MACHADO, Felipe Nery Rodrigues. **Projeto de banco de dados: uma visão prática.** -- 16 ed. rev.e atual. -- São Paulo: Érica, 2009.

ROCHA, Ricardo. **Bases de Dados 2005/2006. Parte VIII: Normalização**. DCC - Faculdade de Ciências da Universidade do Porto. Acessado em <18/02/2015>. Disponível em <http://www.dcc.fc.up.pt/~ricroc/aulas/0506/bd/apontamentos/parteVIII.pdf>.

PIVETTA, Elisa Maria. BANCO DE DADOS: Modelagem de Dados – **Normalização**. UFSM/CAFW, 2011. Acessado em <18/02/2015>. Disponível em <http://www.cafw.ufsm.br/~elisa/bd/normalizacao1.pdf>.

FROZZA, Angelo Augusto. **BANCO DE DADOS - NORMALIZAÇÃO**. Instituto Federal Catarinense. 2011. Acessado em <18/02/2015>. Disponível em <http://www.ifc-camboriu.edu.br/~frozza/2011.1/IE10/IE10-BDD-Aula003-Normalizacao-Revisao.pdf>.

IFSudesteMG. **Exercícios sobre normalização**. Acessado em <19/02/2015>. Disponível em <http://sistemas.riopomba.ifsudestemg.edu.br/dcc/materiais/1997751601_Exemplos_sobre_normalizacao%20resolvido.pdf>.

SANTOS, Ricardo. **Módulo 2: correção exercícios - normalização**. Acessado em <19/02/2015>. Disponível em <http://ricardosantos.net/arquivos/Normalizacao_resolvidos.pdf>.

LOPES, ABRAHÃO. **AULA 11-12 Modelo Conceitual, Lógico e Físico, Entidade-Relacionamento**. Instituto Federal do Rio Grande do Norte, campus Mossoró, 2014. Acessado em <19/02/2015>. Disponível em <http://docente.ifrn.edu.br/abrahaolopes/2014.1-integrado/3.2401.1v-prog-b-dados/modelo-conceitual-logico-fisico-relacionamento-cardinalidade>.

BIANCHI, Wagner. **Entendendo e usando índices - Parte 1**. 2008. Acessado em <19/02/2015>. Disponível em <http://www.devmedia.com.br/entendendo-e-usando-indices-parte-1/6567>.