11 – Introdução

- Uma **subquery** (ou subconsulta) pode ser definida como uma consulta contida dentro de outra instrução SQL (conhecida como query contêiner).

- As consultas mais internas serão executadas primeiro, sendo que somente após o término de sua execução é que a instrução contêiner será iniciada.

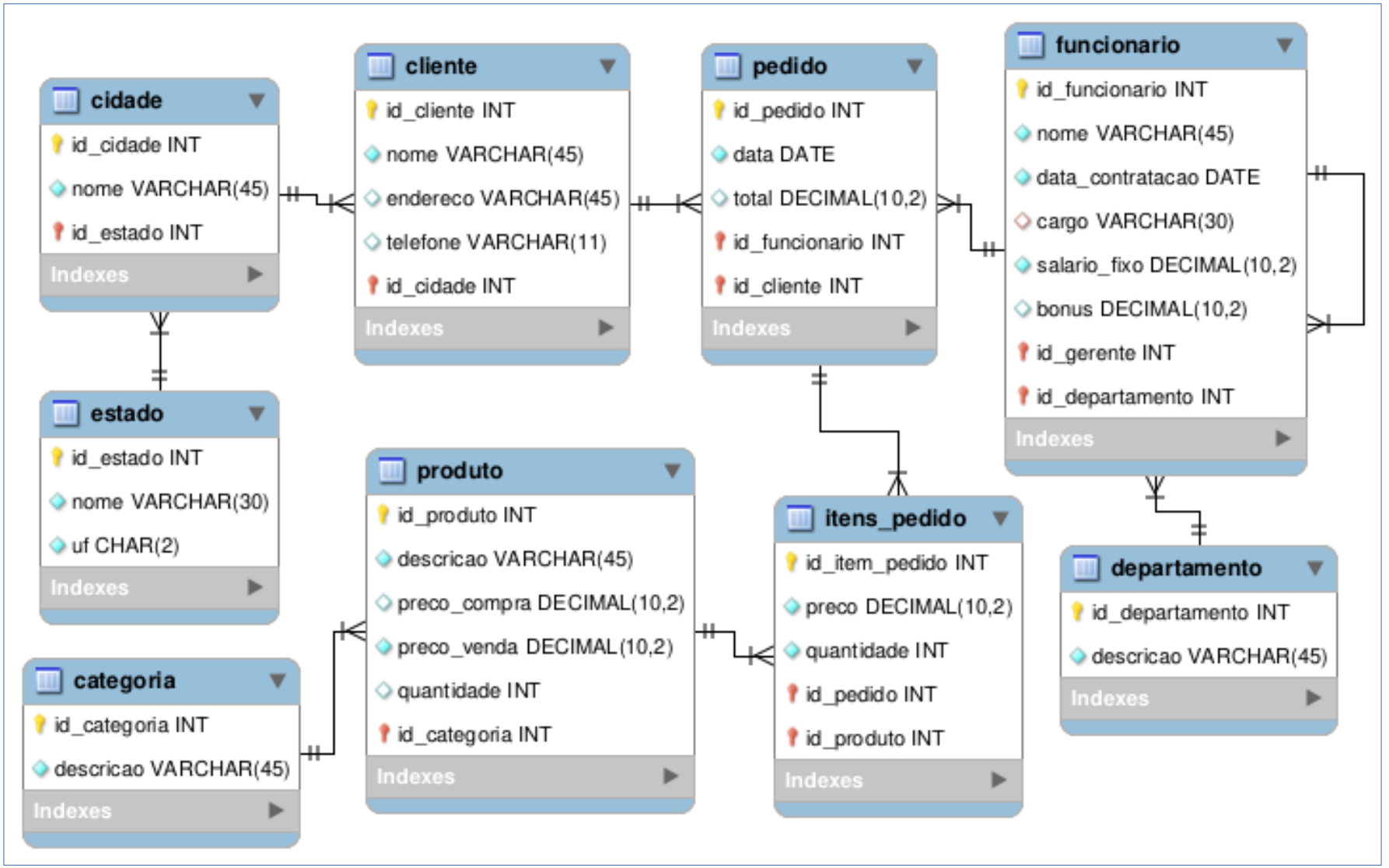
- A query contêiner receberá os valores retornados pela subquery, e utilizará os mesmos como parâmetros de entrada para então ser executada.

- As subquerys sempre devem estar entre parênteses, visto que deverão ter prioridade de execução.

- O resultado das subquerys devem ser compatíveis com as condições de comparação da instrução contêiner.

11 – Introdução

- Será usado o banco de dados loja para exemplificar o funcionamento das subquerys:



11 – Introdução

- Imaginando uma situação onde é necessário selecionar todos os funcionários que possuem o salário acima da média salarial de todos os funcionários da empresa.

- Neste caso, seria necessário a utilização de duas consultas: Uma para retornar o salário médio dos funcionários e outra que retorne os funcionários com salário acima da média (a partir do retorno da primeira consulta):

```
#Retorna o salário médio (no caso 5500)
select avg(salario_fixo) from funcionario


#Retorna os funcionários com salário maior que a média
select nome, salario_fixo from funcionario where salario_fixo>5500
```

nome	salario_fixo
Mario Bros	15000.00
Luigi Bros	12000.00
Paulo Nobre	13000.00

11 – Introdução

- É muito mais custoso para o banco de dados resolver consultas separadas, pois cada uma delas gera tráfego de rede, sobrecarrega o gerenciador de conexões e de fila de transações do SGBD.

- Na grande maioria das vezes, é sempre melhor buscar os dados em apenas uma consulta, mesmo que ela seja mais pesada.

- A subquery abaixo retorna os funcionários com salário acima de média (de forma dinâmica):

```
select nome, salario_fixo from funcionario where salario_fixo>(select avg(salario_fixo)
from funcionario)
```

nome	salario_fixo
Mario Bros	15000.00
Luigi Bros	12000.00
Paulo Nobre	13000.00

11.1 – Subconsultas Não-Correlatas

- A maioria das subquerys são categorizadas como **não-correlatas,** onde a subquery e a instrução contêiner são totalmente independentes.

- Nesta categoria, a subquery não possui qualquer relação com a instrução contêiner, ou seja, não possuem campos em comum (a subconsulta não usa nenhum campo da instrução contêiner).

- O exemplo abaixo é uma consulta não-correlata, visto que a subquery não utiliza nenhuma coluna da instrução contêiner. Seu objetivo é retornar todos os produtos que possuem o preço de venda acima média:

```
select descricao from produto where preco_venda>=(select
avg(preco_venda) from produto)
```

descricao
Processador i7 2.8GHz
HD 2TB Samsung
HD SSD 100GB Samsung
HD SSD 200GB Seagate
Placa Mãe Intel P8
Placa Mãe ASUS XM
Monitor LG 15
Monitor Dell 14

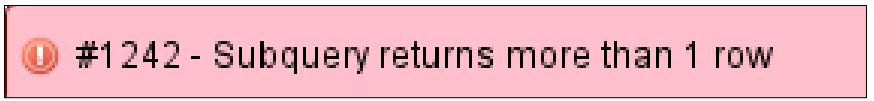
11.2 – Subconsultas de Linhas Múltiplas

- Se uma subquery retornar mais de uma linha, não será possível utilizá-la em operações de comparação utilizando os operadores básicos (<>, =,<,>,<=,>=).

- Em subquerys que retornam mais de uma linha devem ser utilizados outros operadores **(IN, NOT IN, ALL, ANY, EXISTS)**.

- A subquery abaixo retorna a coluna “id_pedido” de todos os pedidos que possuem pelo menos um produto com preço de venda acima de R$500. Entretanto, esta consulta não funcionará, visto que a subquery retorna mais de uma linha (diversos produtos que custa mais que 500):

```
select distinct id_pedido from itens_pedido where id_produto =(select id_produto from produto where preco_venda>500);
```



11.3 – Operador IN e NOT IN

- Apesar de não ser possível igualar um valor único a um conjunto de valores, pode-se verificar se um valor único pode ser encontrado dentro de um conjunto de valores. Isso é possível com o operador **IN**.

- O mesmo exemplo anterior, onde é necessário retornar todos os pedidos que possuem produtos com preço de venda acima de R$500. Mas neste caso será utilizado o operador IN ao invés do "=":

```
select distinct id_pedido from itens_pedido where id_produto IN
(select id_produto from produto where preco_venda>500);
```

id_pedido
1
2
3
4
5
6
7

11.3 – Operador IN e NOT IN

- Assim como o operador IN compara um valor único com todos os valores de um conjunto, verificando se existe algum valor igual, o operador **NOT IN** faz exatamente ao contrário.

- No exemplo abaixo, deseja-se selecionar todos os funcionários que não fizeram nenhum pedido no ano de 2017. Nesse caso, é necessário selecionar todos os funcionários que fizeram pedido em 2017, para em seguida selecionar os que não fizeram com o operador NOT IN:

```
select id_funcionario, nome from funcionario where id_funcionario is not null and
id_funcionario NOT IN(select id_funcionario from pedido where year(data)=2017)
```

Observação: Quando se uso o NOT IN é preciso garantir que o valor do parâmetro que vem da subquery não seja NULL, pois existe o risco do resultado inteiro se tornar NULL.

id_funcionario	nome
1	Mario Bros
2	Luigi Bros
3	Paulo Nobre
4	Bill Gates
5	Tony Stark
6	José da Silva
9	Joaquim Fernandez
10	Maria da Silva

11.3 – Operador IN e NOT IN

- O exemplo abaixo irá mostrar todos os funcionários que supervisionam outros empregados:

```
SELECT nome, cargo FROM funcionario WHERE id_funcionario IN (SELECT id_gerente
FROM funcionario)
```

nome	cargo
Mario Bros	Presidente
Luigi Bros	Diretor
Bill Gates	Programador
José da Silva	Gerente

11.3 – Operador IN e NOT IN

- O exemplo abaixo retorna o vendedor que mais vendeu em 2017.

```
select id_funcionario, nome, cargo, sum(total) from funcionario join pedido
using(id_funcionario) where year(data)=2017 group by(id_funcionario) having sum(total)
= (SELECT sum(total) from pedido where year(data)=2017 group by(id_funcionario)
order by sum(total) desc limit 0,1);
```

id_funcionario	nome	cargo	sum(total)
8	John Rambo	Vendedor	3200.00

11.3 – Operador IN e NOT IN

- O exemplo abaixo retorna o vendedor que mais vendeu em 2017.

```
select id_funcionario, nome, cargo, sum(total) from funcionario join pedido
using(id_funcionario) where year(data)=2017 group by(id_funcionario) having sum(total)
= (SELECT sum(total) from pedido where year(data)=2017 group by(id_funcionario)
order by sum(total) desc limit 0,1);
```

id_funcionario	nome	cargo	sum(total)
8	John Rambo	Vendedor	3200.00

11.4 – Operador ALL

- O operador **ALL** sempre será usado em conjunto com os operadores de comparação tradicionais (=, >=, <=, >, <, <>). Ele faz a comparação com cada elemento individualmente.

- Se na expressão SQL tivermos um comparativo com >ALL, significa que para ser verdadeiro o elemento comparado terá que ser maior do que todos os elementos do lado direito.

- Como exemplo, na situação >ALL(3, 4, 5), significa que o elemento terá que ser maior do que o primeiro (3), segundo(4) e terceiro (5) elemento. Em outras palavras, significa maior do que o valor máximo, ou seja, maior do que 5.

- O operador ALL sempre precisará usar um dos operadores de comparação existentes na linguagem SQL (=, >=, <=, >, <, <>).

11.4 – Operador ALL

- O exemplo abaixo irá retornar todas as categorias que não tiveram nenhum produto vendido no ano de 2017 (Será utilizado o operador ALL, mas seria possível usar o NOT IN também):

```
select descricao from categoria where id_categoria <>ALL(select id_categoria from produto where id_produto IN (select id_produto from itens_pedido join pedido using(id_pedido) where year(data)=2017));
```

descricao
Memórias
Disco
Acessórios
Monitores

11.5 – Operador ANY

- O operador **ANY** permite que um valor seja comparado aos membros de um conjunto de valores, assim como o operador ALL.

- Uma condição que utilize o operador ANY é avaliada como verdadeira tão logo uma única comparação seja verdadeira. Diferentemente do operador ALL, que precisa que todas as comparações sejam verdadeiras.

- O operador ANY é totalmente similar ao IN, com a diferença de utilizar os operadores de comparação da linguagem SQL.

- Como exemplo, tendo uma comparação de um valor com a condição >ANY (3, 4, 5), basta o valor ser maior do que 3, e o resultado já será verdadeiro.

11.5 – Operador ANY

- O exemplo abaixo irá retornar todos os funcionários do departamento de “vendas” que possuem salário maior do que algum funcionário dos departamentos “Diretoria”, “TI” e “Financeiro”:

```
select funcionario.nome from funcionario join departamento using(id_departamento)
where departamento.nome='Vendas' and salario_fixo >ANY(select DISTINCT salario_fixo
from funcionario join departamento using(id_departamento) where departamento.nome in
('Diretoria','TI','Financeiro'));
```

nome
John Rambo

11.10 – Subconsultas na Inserção

- Também é possível utilizar subquery em operações de inserção, basta inserir uma consulta que retorne um valor compatível com o tipo de coluna.

- No exemplo abaixo, é feita uma inserção na tabela de cliente. Na coluna “id_cidade” é utilizada uma subquery que retorna o “id_cidade” da cidade de “Curitiba”:

```
INSERT INTO `cliente` (`id_cliente`, `nome`, `endereco`, `id_cidade`, `telefone`)
VALUES (NULL, 'Chuck Nurris', 'Rua Matador, ', (select id_cidade from cidade where
nome='Curitiba'), '44999432110');
```

12	Chuck Nurris	Rua Matador,	2	44999432110

11.11 – Exemplos Complementares

- O código abaixo mostra o funcionário que efetuou o pedido com maior valor em 2017:

```
SELECT id_funcionario, nome, total FROM pedido JOIN funcionario
USING(id_funcionario) WHERE total=(SELECT MAX(total) FROM pedido WHERE
YEAR(data)='2017')
```

id_funcionario	nome	total
8	Maria José	1800

11.11 – Exemplos Complementares

- O código abaixo mostra o nome, salário e salário reajustado em 30% dos empregados que venderam mais do que 1000 reais no ano de 2017:

```
SELECT nome, salario_fixo, (salario_fixo*1.3) AS 'Salario Atual' FROM funcionario
WHERE id_funcionario IN(SELECT id_funcionario FROM pedido WHERE YEAR(data)='2017'
GROUP BY id_funcionario HAVING SUM(total)>=1000)
```

nome	salario_fixo	Salario Atual
João Garrincha	1000	1300
Maria José	1000	1300