



Consulta com várias tabelas no PostgreSQL

Você vai aprender a projetar consultas envolvendo diversas tabelas com diferentes tipos de junção, a utilização de mecanismos de subconsultas aninhadas, correlatas e o uso de operadores de conjunto.

Prof. Sidney Venturi, Profª. Nathielly de Souza Campos

1. Itens iniciais

Preparação

Antes de iniciar este conteúdo, certifique-se de ter baixado e instalado o SGBD PostgreSQL em seu computador.

Objetivos

- Aplicar consultas envolvendo junções interior e exterior.
- Aplicar subconsultas aninhadas e correlatas.
- Aplicar consultas com o uso de operadores de conjunto.

Introdução

No vídeo a seguir, vamos introduzir os conceitos sobre a consulta com várias tabelas no PostgreSQL. Acompanhe!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Junção e produto cartesiano

Você saberia como atuar quando é preciso retornar dados de uma tabela, por exemplo, o nome do departamento armazenado na tabela **Departamento** e o nome da região armazenado na tabela **Região**? Pois é, neste caso, temos que realizar uma junção. Vamos esclarecer esses pontos!

Acompanhe neste vídeo as operações de junção e o produto cartesiano do modelo relacional de banco de dados. Você vai entender como essas operações permitem combinar dados de diferentes tabelas de maneira eficiente e estruturada.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Operação de junção de tabelas

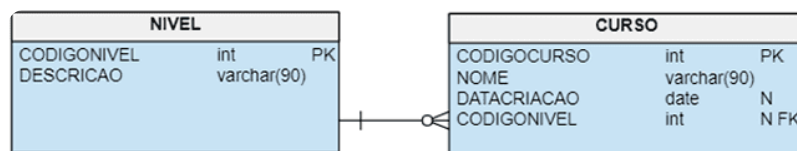
A junção de tabelas é uma das operações mais importantes criadas pelo modelo relacional de banco dados.

Na definição matemática, o resultado da junção de duas tabelas é um subconjunto do produto cartesiano entre essas tabelas. A especificação desse subconjunto é feita por uma condição de junção entre colunas das duas tabelas.

Existem dois tipos de junção:

- Interna, denominada INNER JOIN.
- Externa, denominada OUTER JOIN, que pode ser LEFT, FULL ou RIGHT OUTER JOIN.

Construiremos algumas consultas com base nas tabelas NIVEL e CURSO, conforme a imagem a seguir. Vamos lá!



Tabelas NIVEL e CURSO.

Recomendamos que você crie as tabelas e insira algumas linhas, o que pode ser feito usando o script a seguir, a partir da ferramenta de sua preferência. Para isso, tenha em mente que é necessário estar conectado ao PostgreSQL e acessando algum database criado por você.

sql

```
CREATE TABLE NIVEL (  
    CODIGONIVEL int NOT NULL,  
    DESCRICAO varchar(90) NOT NULL,  
    CONSTRAINT CHAVEPNIVEL PRIMARY KEY (CODIGONIVEL));  
CREATE TABLE CURSO (  
    CODIGOCURSO int NOT NULL,  
    NOME varchar(90) NOT NULL UNIQUE,  
    DATACRIACAO date NULL,  
    CODIGONIVEL int NULL,  
    CONSTRAINT CHAVEPCURSO PRIMARY KEY (CODIGOCURSO));  
ALTER TABLE CURSO ADD FOREIGN KEY (CODIGONIVEL) REFERENCES NIVEL;  
INSERT INTO NIVEL (CODIGONIVEL, DESCRICAO) VALUES (1,'Graduação');  
INSERT INTO NIVEL (CODIGONIVEL, DESCRICAO) VALUES (2,'Especialização');  
INSERT INTO NIVEL (CODIGONIVEL, DESCRICAO) VALUES (3,'Mestrado');  
INSERT INTO NIVEL (CODIGONIVEL, DESCRICAO) VALUES (4,'Doutorado');  
INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO, CODIGONIVEL) VALUES (1, 'Sistemas de  
Informação', '19/06/1999',1);  
INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO, CODIGONIVEL) VALUES (2, 'Medicina',  
'10/05/1990',1);  
INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO, CODIGONIVEL) VALUES (3, 'Nutrição',  
'19/02/2012',NULL);  
INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO, CODIGONIVEL) VALUES (4,  
'Pedagogia','19/06/1999',1);  
INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO, CODIGONIVEL) VALUES (5, 'Saúde da  
Família','10/09/1999',3);  
INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO, CODIGONIVEL) VALUES (6, 'Computação  
Aplicada','10/09/1999', NULL);
```

No script, foram usados três comandos: CREATE, ALTER e INSERT. A sintaxe completa desses comandos no PostgreSQL pode ser encontrada com a indicação disponibilizada no Explore + ao final do conteúdo, tudo bem?

A imagem a seguir apresenta o conteúdo da tabela NIVEL após a execução do comando SELECT * FROM NIVEL;

	codigonivel	descricao
1	1	Graduação
2	2	Especialização
3	3	Mestrado
4	4	Doutorado

Conteúdo da tabela NIVEL.

A imagem a seguir apresenta o conteúdo da tabela CURSO após a execução do comando TABLE CURSO;

	codigocurso	nome	datacriacao	codigonivel
1	1	Sistemas de Informação	1999-06-19	1
2	2	Medicina	1990-05-10	1
3	3	Nutrição	2012-02-19	[NULL]
4	4	Pedagogia	1999-06-19	1
5	5	Saúde da Família	1999-09-10	3
6	6	Computação Aplicada	1999-09-10	[NULL]

Conteúdo da tabela CURSO.

Observe que as tabelas NIVEL e CURSO estão relacionadas. A tabela NIVEL é a mais independente, visto que não possui chave estrangeira. Consequência disso é a possibilidade de um usuário cadastrar diversos níveis na tabela sem a preocupação com qualquer outra tabela do banco de dados. O relacionamento entre NIVEL e CURSO está implementado na tabela CURSO, por meio da coluna CODIGONIVEL, que exerce o papel de chave estrangeira.

Por fim, note também que o fato da coluna CODIGONIVEL ser opcional na tabela CURSO representa a possibilidade de cadastrar um curso sem, em um primeiro momento, relacioná-lo a determinado nível existente. Isso é justamente o que ocorreu após a execução dos comandos das linhas 20 e 23, nos quais o valor inserido para a coluna CODIGONIVEL é NULL.

Na sequência, acompanhe algumas observações sobre o que acontece quando declaramos mais de uma tabela em uma consulta SQL.

Operação de produto cartesiano

Em termos estruturais, a tabela NIVEL possui duas colunas e quatro registros. De forma semelhante, a tabela CURSO possui quatro colunas e seis registros. Você pode chegar a essa conclusão ao analisar o script anterior.

Agora, vamos executar a seguinte consulta SQL: `SELECT * FROM CURSO, NIVEL;` em que o resultado está expresso na imagem a seguir. Observe!

	codigocurso	nome	datacriacao	codigonivel	codigonivel	descricao
1	1	Sistemas de Informação	1999-06-19	1	1	Graduação
2	1	Sistemas de Informação	1999-06-19	1	2	Especialização
3	1	Sistemas de Informação	1999-06-19	1	3	Mestrado
4	1	Sistemas de Informação	1999-06-19	1	4	Doutorado
5	2	Medicina	1990-05-10	1	1	Graduação
6	2	Medicina	1990-05-10	1	2	Especialização
7	2	Medicina	1990-05-10	1	3	Mestrado
8	2	Medicina	1990-05-10	1	4	Doutorado
9	3	Nutrição	2012-02-19	[NULL]	1	Graduação
10	3	Nutrição	2012-02-19	[NULL]	2	Especialização
11	3	Nutrição	2012-02-19	[NULL]	3	Mestrado
12	3	Nutrição	2012-02-19	[NULL]	4	Doutorado
13	4	Pedagogia	1999-06-19	1	1	Graduação
14	4	Pedagogia	1999-06-19	1	2	Especialização
15	4	Pedagogia	1999-06-19	1	3	Mestrado
16	4	Pedagogia	1999-06-19	1	4	Doutorado
17	5	Saúde da Família	1999-09-10	3	1	Graduação
18	5	Saúde da Família	1999-09-10	3	2	Especialização
19	5	Saúde da Família	1999-09-10	3	3	Mestrado
20	5	Saúde da Família	1999-09-10	3	4	Doutorado
21	6	Computação Aplicada	1999-09-10	[NULL]	1	Graduação
22	6	Computação Aplicada	1999-09-10	[NULL]	2	Especialização
23	6	Computação Aplicada	1999-09-10	[NULL]	3	Mestrado
24	6	Computação Aplicada	1999-09-10	[NULL]	4	Doutorado

Resultado da consulta `SELECT * FROM CURSO, NIVEL;`

O resultado da consulta anterior é uma tabela (em memória principal) originada da operação de produto cartesiano. Nessa operação, o sistema gerenciador de banco de dados (SGBD) combinou cada linha da tabela CURSO com cada registro da tabela NIVEL.

Note que a quantidade de (seis) colunas na tabela resultante é a soma das colunas das tabelas envolvidas. De forma semelhante, a quantidade (vinte e quatro) de registros da tabela é igual ao produto entre o número de linhas de CURSO e NIVEL.

Perceba que, ao longo do nosso estudo, temos interpretado cada linha de uma tabela como sendo um fato registrado no banco de dados, correspondendo a uma realidade do contexto do negócio sendo modelado.



Exemplo

Ao visualizarmos o conteúdo da tabela NIVEL, afirmamos que há quatro registros ou ocorrências de nível no banco de dados. De forma semelhante, há seis cursos cadastrados na tabela CURSO.

No entanto, o mesmo raciocínio não pode ser aplicado à tabela resultante da consulta, já que temos todas as combinações possíveis entre as linhas, além de envolver todas as colunas das tabelas CURSO e NIVEL.



Comentário

Cabe aqui um alerta sobre o cuidado que se deve ter ao usar esse tipo de consulta, que resulta em produto cartesiano. Suponha uma tabela CLIENTES com 1.000.000 de linhas e uma tabela PEDIDOS com 10.000.000 de linhas. O comando `SELECT * FROM CLIENTES, PEDIDOS` retornaria 10.000.000.000.000 de linhas e, provavelmente, ocuparia longas horas de processamento no servidor, além de retornar um resultado inútil!

Há outra maneira de obtermos os mesmos resultados de `SELECT * FROM CURSO, NIVEL;`. Basta executar o código equivalente: `SELECT * FROM CURSO CROSS JOIN NIVEL;`

A seguir, vamos estudar uma forma de extrair informações úteis a partir do resultado dessa consulta.

Atividade 1

A linguagem SQL foi criada a partir da álgebra relacional, tendo como operações, entre outras, a junção e o produto cartesiano, que a partir de duas entre duas relações, gera uma nova relação. Qual é a principal diferença entre uma junção (join) e um produto cartesiano (cartesian product) em bancos de dados relacionais?

A

Uma junção combina apenas os registros que atendem a uma condição específica, enquanto um produto cartesiano combina todos os registros de ambas as tabelas, independentemente de qualquer condição.

B

Uma junção combina todos os registros de ambas as tabelas, independentemente de qualquer condição, enquanto um produto cartesiano combina apenas os registros que atendem a uma condição específica.

C

Ambos os operadores combinam os registros de duas tabelas com base em uma condição específica.

D

Uma junção combina todos os registros de ambas as tabelas, segundo uma condição específica, enquanto um produto cartesiano combina apenas os registros que atendem a uma condição específica.

E

Ambos os operadores combinam os registros de duas tabelas sem uma condição específica.



A alternativa A está correta.

Uma junção (join) em SQL é um operador utilizado para combinar registros de duas ou mais tabelas em um banco de dados, com base em uma condição de igualdade entre colunas. A junção retorna apenas os registros que atendem à condição especificada. Por outro lado, o produto cartesiano (cartesian product)

combina todos os registros de uma tabela com todos os registros de outra tabela, criando uma tabela resultante que contém todos os pares possíveis de registros entre as tabelas, sem levar em consideração qualquer condição.

Junção interna

Vamos explorar, de forma abrangente, as junções internas no SQL, destacando sua importância na manipulação eficaz de bancos de dados relacionais. Veremos técnicas avançadas de junção para extrair informações importantes e otimizar consultas complexas. Vamos lá!

Acompanhe neste vídeo a operação de junção interna em bancos de dados, que permite combinar registros de duas tabelas com base em uma condição de correspondência específica, retornando apenas os registros que têm correspondência em ambas as tabelas.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Nosso interesse agora é obter informações úteis para o nosso usuário, certo? Então, perceba que, por exemplo, o curso sistemas de informação está classificado como um curso pertencente ao nível de graduação. Chegamos a essa conclusão avaliando o conteúdo das linhas 14 e 18 do script que insere informações nas tabelas NIVEL e CURSO.

Agora, examine as quatro primeiras linhas da tabela resultante da consulta anterior, expressa na imagem a seguir.

	codigocurso	nome	datacriacao	codigonivel	codigonivel	descricao
1	1	Sistemas de Informação	1999-06-19	1	1	Graduação
2	1	Sistemas de Informação	1999-06-19	1	2	Especialização
3	1	Sistemas de Informação	1999-06-19	1	3	Mestrado
4	1	Sistemas de Informação	1999-06-19	1	4	Doutorado

Subconjunto da tabela resultante do produto cartesiano entre CURSO e NIVEL.

Note que, se examinarmos cada linha como um fato, vamos perceber que somente a primeira corresponde à realidade cadastrada no banco de dados. Não por coincidência, perceba que os valores das colunas CODIGONIVEL são os mesmos. Nas três últimas, diferentes.

Assim, podemos eliminar as linhas falsas se programarmos uma condição de igualdade envolvendo as colunas em questão. Devemos lembrar que as quatro primeiras colunas vêm da tabela CURSO. As duas últimas são originadas na tabela NIVEL.

Para recuperar as linhas que correspondem à realidade cadastrada no banco de dados, você pode executar o comando a seguir.

```
sql
```

```
1 SELECT *
2 FROM CURSO, NIVEL
3 WHERE NIVEL.CODIGONIVEL=CURSO.CODIGONIVEL;
```

O resultado dessa consulta está expresso na imagem. Veja!

	codigocurso	nome	datacriacao	codigonivel	codigonivel	descricao
1	1	Sistemas de Informação	1999-06-19	1	1	Graduação
2	2	Medicina	1990-05-10	1	1	Graduação
4	4	Pedagogia	1999-06-19	1	1	Graduação
4	5	Saúde da Família	1999-09-10	3	3	Mestrado

Resultado da consulta envolvendo as tabelas CURSO e NIVEL.

Perceba que foram recuperadas as linhas que de fato relacionam cursos aos seus respectivos níveis. No entanto, a forma mais usada para retornar os mesmos resultados é com o auxílio da cláusula de junção interna, a qual possui sintaxe **básica**. Vejamos!

```
SELECT *  
FROM TABELA1 [INNER] JOIN TABELA2 ON (CONDIÇÃOJUNÇÃO) [USING (COLUNA_DE_JUNÇÃO)]
```

Na sintaxe apresentada, declaramos a cláusula de junção entre as tabelas. Em seguida, a preposição ON e a condição da junção. Na maioria das vezes, essa condição corresponderá a uma igualdade envolvendo a chave primária de uma tabela e a chave estrangeira correspondente. A sintaxe completa do comando SELECT no PostgreSQL pode ser encontrada com a indicação disponibilizada no Explore +, ao final do material.

Veja a seguir o código SQL correspondente ao nosso exemplo.

```
sql  
  
1 SELECT *  
2 FROM CURSO INNER JOIN NIVEL ON(NIVEL.CODIGONIVEL=CURSO.CODIGONIVEL);
```

Note também que é possível declarar a cláusula USING especificando a coluna-alvo da junção. Em nosso exemplo, a coluna CODIGONIVEL. A consulta pode ser reescrita da seguinte forma:

```
sql  
  
1 SELECT *  
2 FROM CURSO INNER JOIN NIVEL USING(CODIGONIVEL);
```

Se desejarmos exibir o código e o nome do curso, além do código e o nome do nível, podemos, então, executar o código a seguir:

```
sql  
  
1 SELECT CURSO.CODIGOCURSO, CURSO.NOME,  
2 FROM CURSO INNER JOIN NIVEL USING(CODIGONIVEL);
```

Perceba que, no comando SELECT, usamos uma referência mais completa: NOMETABELA.NOMECOLUNA.

Essa referência só é obrigatória para a coluna CODIGONIVEL, uma vez que é necessário especificar de qual tabela o SGBD irá buscar os valores. No entanto, em termos de organização de código, é interessante usar esse tipo de referência para cada coluna.

Finalmente, observe que poderíamos também, no contexto da consulta, renomear as tabelas envolvidas, deixando o código mais elegante e legível, conforme a seguir.

```
sql  
  
1 SELECT C.CODIGOCURSO, C.NOME,  
2 N.CODIGONIVEL, N.DESCRICAO  
3 FROM CURSO C INNER JOIN NIVEL N ON  
4 (N.CODIGONIVEL=C.CODIGONIVEL) ;
```

O resultado da consulta está expresso na imagem. Veja!

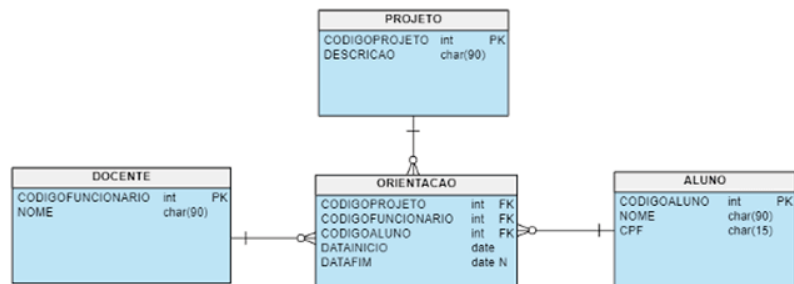
	123	codigocurso	abc	nome	123	codigonivel	abc	descricao
1		1		Sistemas de Informação		1		Graduação
2		2		Medicina		1		Graduação
3		4		Pedagogia		1		Graduação
4		5		Saúde da Família		3		Mestrado

Resultado da consulta envolvendo as tabelas CURSO e NIVEL.

O resultado de uma junção interna corresponde somente aos registros que atendem à condição da junção, ou seja, os registros que de fato estão relacionados no contexto das tabelas envolvidas.

Atividade 2

Considere as tabelas e o código SQL a seguir.



```

1 SELECT PR.DESCRICAO,D.NOME, A.NOME
2 FROM ALUNO A JOIN ORIENTACAO O ON (A.CODIGOALUNO=O.CODIGOALUNO)
3     JOIN DOCENTE D ON (D.CODIGOFUNCIONARIO=O.CODIGOFUNCIONARIO)
4     JOIN PROJETO PR ON (PR.CODIGOPROJETO=O.CODIGOPROJETO)
5 ORDER BY PR.DESCRICAO,D.NOME, A.NOME;

```

A consulta retorna:

A

a descrição dos projetos e nomes de professores e alunos envolvidos em orientação.

B

a descrição dos projetos e nomes de professores e alunos não envolvidos em orientação.

C

a descrição dos projetos e nomes de professores e alunos envolvidos e não envolvidos em orientação.

D

a descrição dos projetos sem orientação definida.

E

a descrição dos projetos e nomes de professores não envolvidos em orientação.



A alternativa A está correta.

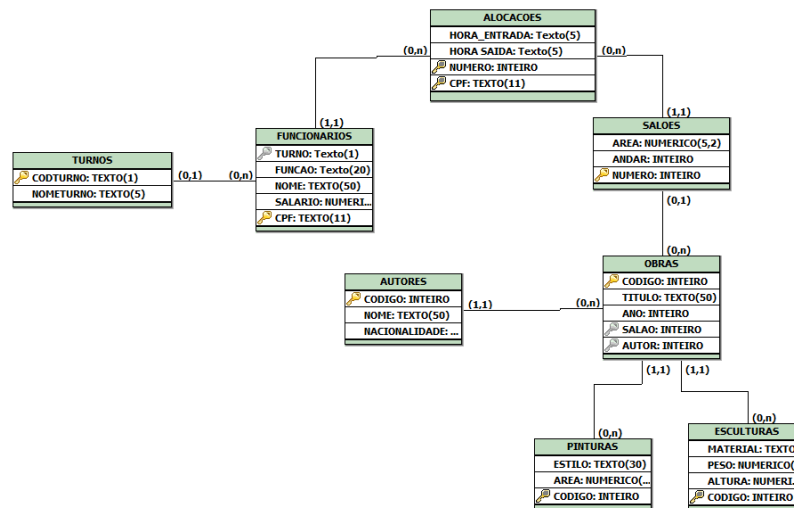
A consulta envolve somente junções internas. Retorna informações sobre todas as orientações cadastradas no banco de dados.

Praticando as junções internas

Vamos criar o banco de dados do museu que utilizaremos nas nossas demonstrações e, em seguida, iremos executar alguns comandos com junção interna.

Você pode baixar o [script da criação](#) das tabelas e inserção das linhas do banco de dados do museu.

Vejam o modelo lógico de banco de dados do minimundo MUSEU!



Modelo lógico de banco de dados do minimundo MUSEU.

Praticando as junções internas

Neste vídeo, vamos praticar as junções internas utilizando a linguagem SQL.



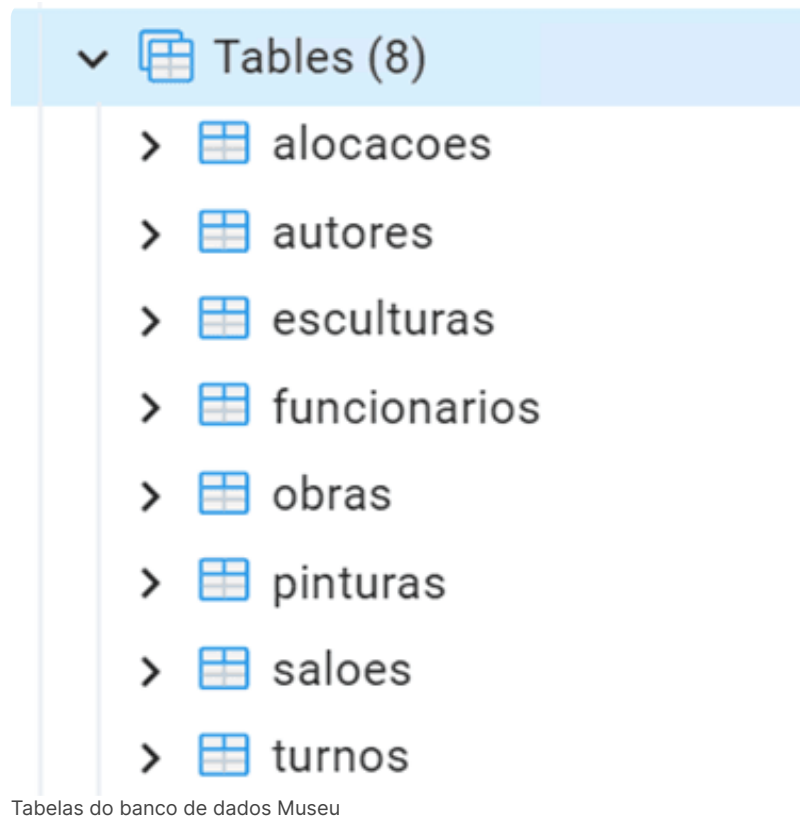
Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Roteiro de prática

Crie o banco do museu:

1. Abra o PGAdmin4 e faça conexão no servidor.
2. Crie um banco de dados chamado MUSEU.
3. Abra uma janela de consulta.
4. Carregue o script do museu na janela de consulta.
5. Execute o script.
6. Valide a criação das tabelas na aba tabelas do banco do museu.



Escreva os seguintes comandos de consulta:

- Listar todos os dados dos autores e de suas obras.
- Listar o código, o título e o nome do autor das obras de 1965 a 1975 que estão no salão 36.
- Listar o nome e a nacionalidade dos autores que possuem obras expostas. Resolver por junção.
- Listar o código e o título das obras do autor Pablo Picasso que se encontram no terceiro andar do museu.
- Listar o nome e a nacionalidade dos autores, o título da obra e o estilo de pintura.
- Listar o título da obra e o estilo da pintura utilizando junção USING.
- Listar o título da obra e o estilo da pintura utilizando junção tradicional.

Se desejar, você pode encontrar no arquivo [sqlmod1.3](#) os comandos utilizados no vídeo.

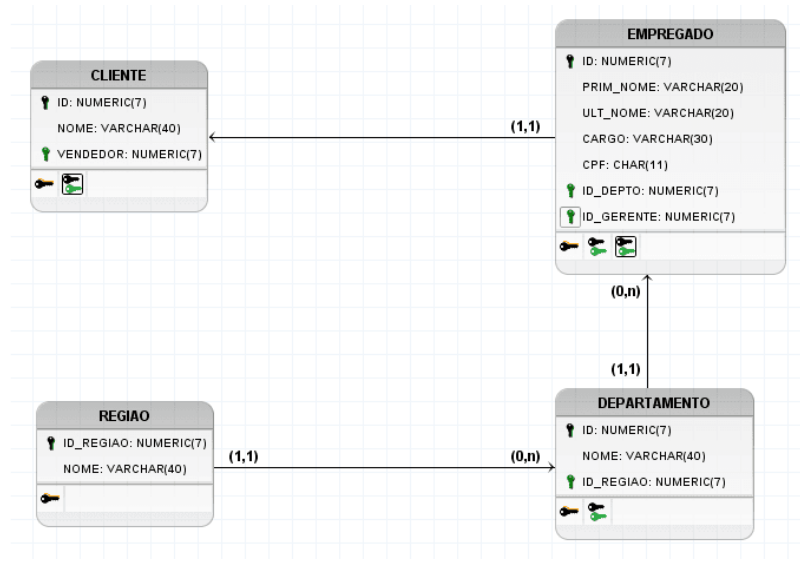
Atividade 3

Vamos agora criar o banco de dados da empresa que utilizaremos nas nossas demonstrações e, em seguida, iremos executar alguns comandos com junção interna.

O vídeo mostrará a criação passo a passo, bem como a construção dos comandos.

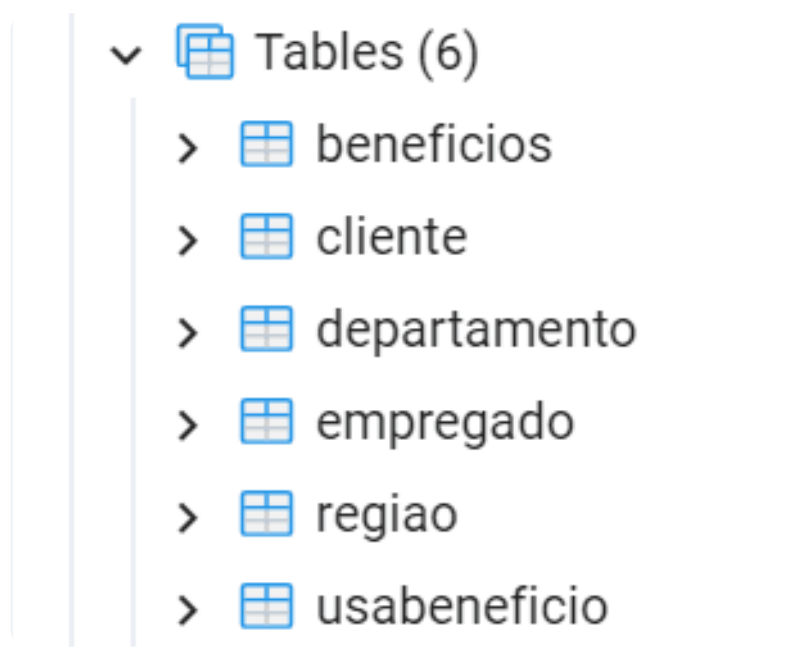
Você pode baixar o [script da criação](#) das tabelas e inserção das linhas do banco de dados da empresa.

O modelo de dados da empresa é o seguinte:



Para criar o banco de dados da empresa, siga estes passos:

1. Abra o PGAdmin4 e faça conexão no servidor.
2. Crie um banco de dados chamado EMPRESA.
3. Abra uma janela de consulta.
4. Carregue o script da empresa na janela de consulta.
5. Execute o script.
6. Valide a criação das tabelas na aba tabelas do banco da empresa.



Escreva agora os seguintes comandos:

- Listar todos os dados dos departamentos da região em que estão localizados.
- Listar todos os dados dos departamentos da região em que estão na sintaxe tradicional.
- Listar o ID do empregado, o sobrenome do empregado, o ID do departamento, o nome do departamento e o nome da região onde está localizado o departamento.
- Listar todos os dados dos departamentos da região utilizando USING.

Chave de resposta

Você pode baixar os comandos que resolvem as consultas no arquivo [exercmod1.3](#).

Junção externa

Vamos agora entender as junções externas no SQL, revelando como ampliar a análise de dados além dos limites convencionais. Aprenderemos a incorporar dados de várias tabelas e a lidar com informações ausentes com confiança e precisão.

Neste vídeo, vamos compreender a junção externa em bancos de dados, operação que combina registros de duas tabelas, retornando não apenas os registros que têm correspondência, mas os registros que não têm correspondência em uma ou ambas as tabelas.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

A imagem do resultado da consulta envolvendo as tabelas CURSO e NIVEL na junção interna exibe somente os cursos para os quais há registro de informação sobre o nível associado a eles.

E se quiséssemos incluir na listagem todos os registros da tabela CURSO?

Para incluir no resultado da consulta todas as ocorrências da tabela CURSO, podemos usar a cláusula LEFT JOIN (junção à esquerda). Nesse tipo de junção, o resultado contém todos os registros da tabela declarada à esquerda da cláusula JOIN, mesmo que não haja registros correspondentes na tabela da direita. Em especial, quando não há correspondência, os resultados são retornados com o valor NULL.

Veja a seguir exemplo de uso de **junção à esquerda**.

sql

```
1 SELECT C.CODIGOCURSO, C.NOME,  
2 N.CODIGONIVEL, N.DESCRICAO  
3 FROM CURSO C LEFT JOIN NIVEL N ON (N.CODIGONIVEL=C.CODIGONIVEL) ;
```

A imagem a seguir apresenta o resultado da consulta anterior. Confira!

	codigocurso	nome	codigonivel	descricao
1	1	Sistemas de Informação	1	Graduação
2	2	Medicina	1	Graduação
3	3	Nutrição	[NULL]	[NULL]
4	4	Pedagogia	1	Graduação
5	5	Saúde da Família	3	Mestrado
6	6	Computação Aplicada	[NULL]	[NULL]

Cursos e níveis, exibindo todos os registros da tabela CURSO.

Observe que o número de linhas do resultado da consulta coincide com o número de linhas da tabela CURSO, visto que todos os registros dessa tabela fazem parte do resultado e a chave estrangeira que implementa o relacionamento está localizada na tabela CURSO. Em especial, as linhas 3 e 6 correspondem a cursos em que não há informação sobre o nível associado a eles.

Perceba também que, de forma semelhante, poderíamos ter interesse em exibir todos os registros da tabela à direita da cláusula JOIN. Em nosso exemplo, a tabela NIVEL. A cláusula RIGHT JOIN (junção à direita) é usada para essa finalidade. Nesse tipo de junção, o resultado contém todos os registros da tabela declarada à direita da cláusula JOIN, mesmo que não haja registros correspondentes na tabela da esquerda. Em especial, quando não há correspondência, os resultados são retornados com o valor NULL.

Veja a seguir exemplo de uso de **junção à direita**.

sql

```
1 SELECT C.CODIGOCURSO, C.NOME,  
2 N.CODIGONIVEL, N.DESCRICAO  
3 FROM CURSO C RIGHT JOIN NIVEL N ON  
4 (N.CODIGONIVEL=C.CODIGONIVEL);
```

A imagem a seguir apresenta o resultado da consulta anterior. Veja!

	codigocurso	nome	codigonivel	descricao
1	1	Sistemas de Informação	1	Graduação
2	2	Medicina	1	Graduação
3	3	Nutrição	[NULL]	[NULL]
4	4	Pedagogia	1	Graduação
5	5	Saúde da Família	3	Mestrado
6	6	Computação Aplicada	[NULL]	[NULL]
7	[NULL]	[NULL]	2	Especialização
8	[NULL]	[NULL]	4	Doutorado

Cursos e níveis, exibindo todos os registros da tabela NIVEL.

Note que todos os registros da tabela NIVEL aparecem no resultado. As quatro primeiras linhas do resultado da consulta correspondem aos registros que efetivamente estão relacionados à tabela CURSO. As duas últimas linhas são registros que não estão relacionados a qualquer curso existente no banco de dados.

Perceba que o mesmo resultado pode ser obtido se usarmos junção à esquerda e junção à direita, alternando a posição das tabelas envolvidas. Com isso, queremos dizer que TABELA1 LEFT JOIN TABELA2 é equivalente a TABELA2 RIGHT JOIN TABELA1.

Veja o exemplo a seguir!

sql

```
1 SELECT C.CODIGOCURSO, C.NOME,  
2 N.CODIGONIVEL, N.DESCRICAO  
3 FROM CURSO C LEFT JOIN NIVEL N ON (N.CODIGONIVEL=C.CODIGONIVEL);
```

Vejamos a mesma consulta, mas agora utilizando a cláusula RIGHT JOIN.

sql

```
SELECT C.CODIGOCURSO, C.NOME,  
N.CODIGONIVEL, N.DESCRICAO  
FROM NIVEL N RIGHT JOIN CURSO C ON (N.CODIGONIVEL=C.CODIGONIVEL);
```

Outro tipo de junção externa, denominada FULL OUTER JOIN (junção completa), apresenta todos os registros das tabelas à esquerda e à direita, mesmo os registros não relacionados. Em outras palavras, a tabela resultante exibirá todos os registros de ambas as tabelas, além de valores NULL no caso dos registros sem correspondência. Veja o exemplo!

sql

```
1 SELECT C.CODIGOCURSO, C.NOME,  
2 N.CODIGONIVEL, N.DESCRICAO  
3 FROM CURSO C FULL OUTER JOIN NIVEL N ON  
4 (N.CODIGONIVEL=C.CODIGONIVEL);
```

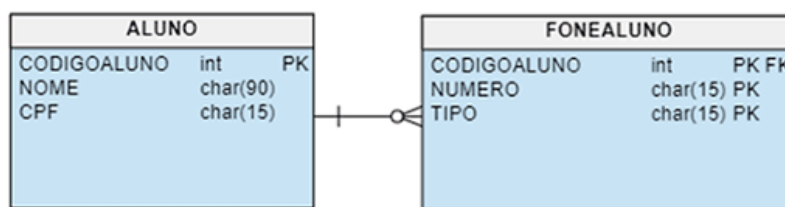
A imagem a seguir apresenta o resultado da consulta anterior. Confira!

Note que, no resultado, aparecem os registros de cada tabela. Além disso, valores NULL são exibidos nos casos em que não há correspondência entre as tabelas (linhas 3, 6, 7 e 8).

Vimos que, quando há necessidade de extrair informações de mais de uma tabela, utilizamos a cláusula de junção. Os diversos tipos de junção são utilizados de acordo com a necessidade do usuário, visto que cada um possui uma especificidade.

Atividade 4

Considere as tabelas e o código SQL a seguir.



```
1 SELECT NOME, NUMERO, TIPO  
2 FROM ALUNO A LEFT JOIN FONEALUNO F  
3 ON (A.CODIGOALUNO=F.CODIGOALUNO);
```

A consulta retorna:

A

somente os registros de ALUNO relacionados à tabela FONEALUNO.

B

registros tanto dos alunos com telefone quanto dos alunos sem telefone.

C

erro.

D

somente registros de alunos que têm mais de um telefone cadastrado.

E

somente registros de alunos que não têm um telefone cadastrado.

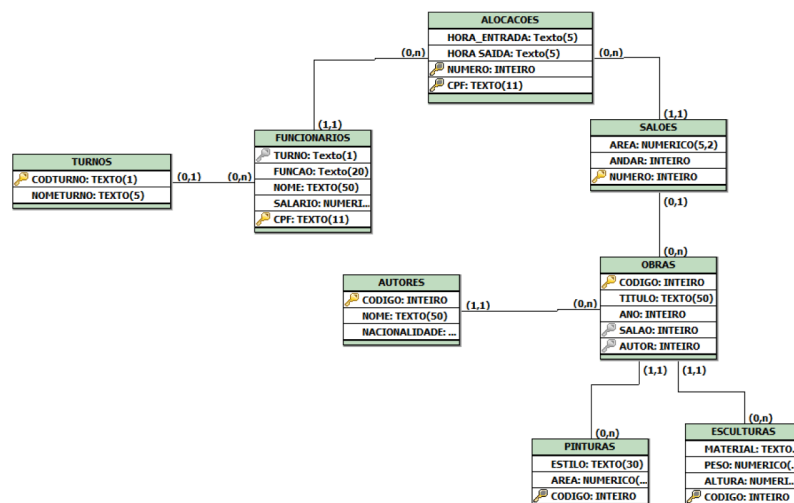


A alternativa B está correta.

A consulta usa junção à esquerda. Logo, todos os registros de ALUNO farão parte do resultado: alunos com e sem telefone cadastrados no banco de dados.

Praticando as junções externas

Vamos analisar exemplos de uso dos comandos vistos até aqui. Para isso, iremos utilizar o banco de dados do museu. Seu modelo lógico é o seguinte:



Modelo lógico de banco de dados do minimundo MUSEU.

Praticando as junções externas

Neste vídeo, vamos praticar as junções externas utilizando a linguagem SQL.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Roteiro de prática

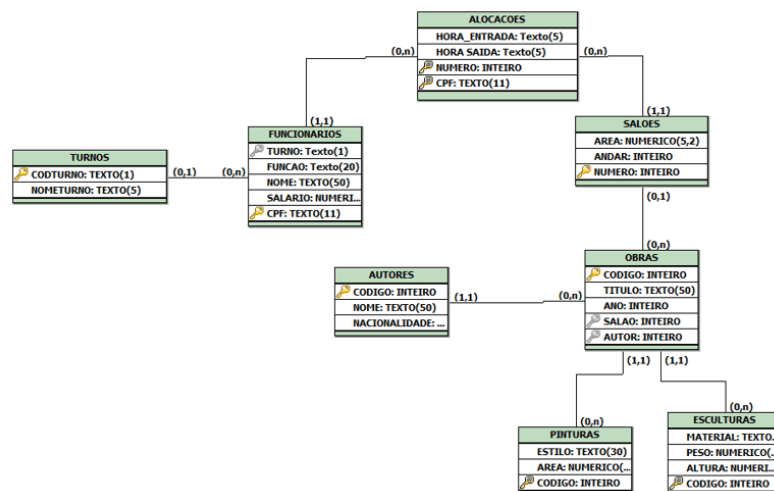
Para executar a atividade, escreva os seguintes comandos:

1. Listar o nome e a nacionalidade dos autores que não possuem obras expostas. Resolver por junção.
2. Listar todos os dados dos salões que não possuem obras expostas.
3. Listar o código e o título das obras impressionistas ou cujo material de fabricação é argila. Resolver por junção.
4. Listar os dados dos funcionários que estão alocados com os dados da alocação e os dados não alocados.

Você pode baixar os comandos que resolvem as consultas no arquivo [sqlmod1.5](#).

Atividade 5

Vamos agora executar uma atividade prática utilizando o banco de dados da empresa. Seu modelo lógico é o seguinte:



Modelo lógico de banco de dados do minimundo EMPRESA.

Para realizar o exercício, faça conexão e realize as seguintes consultas:

- Listar o ID e nome do cliente e o ID, ULT_NOME e cargo do empregado que o atende.
- Listar o ID, ULT_NOME e cargo do empregado que não atende nenhum cliente.
- Listar o ID e nome do cliente que não é atendido por nenhum empregado.
- Listar o ID e o nome do cliente e o ID, ULT_NOME e cargo do empregado para os empregados que se relacionam com cliente e para os que não se relacionam.

Chave de resposta

Você pode baixar os comandos que resolvem as consultas no arquivo [exercmod1.5](#).

Subconsultas aninhadas

Veremos as subconsultas aninhadas, explorando como estruturar consultas complexas para obter informações detalhadas dos dados. Você vai aprender a aplicar consultas eficientes e de várias camadas para realizar análises profundas e resolver problemas sofisticados de manipulação de dados. Vamos lá!

Conheça neste vídeo o processo de aplicação das subconsultas aninhadas. Descubra como essas consultas internas podem ser utilizadas para realizar operações complexas dentro do modelo relacional.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Subconsultas

O resultado de uma consulta SQL corresponde a uma tabela, mesmo que esteja temporariamente armazenada na memória principal do computador, tudo bem?

Vamos construir consultas que dependem (ou usam) dos resultados de outras consultas para recuperar informações de interesse. Essa categoria é conhecida por subconsulta, uma consulta sobre o resultado de outra consulta.

Ao longo de nosso estudo, vamos explorar dois tipos de subconsultas:

Subconsultas aninhadas

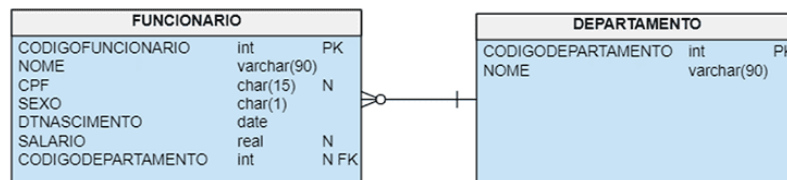
A consulta mais externa realizará algum tipo de teste junto aos dados originados da consulta mais interna.

Subconsultas correlatas

A subconsulta utiliza valores da consulta externa. Nesse tipo de consulta, a subconsulta é executada para cada linha da consulta externa.

Construiremos as consultas com base nas tabelas FUNCIONARIO E DEPARTAMENTO, conforme imagem a seguir.

Recomendamos que você crie as tabelas e insira algumas linhas, o que pode ser feito usando o script a seguir, a partir da ferramenta de sua preferência. Para isso, tenha em mente que é necessário estar conectado ao PostgreSQL e acessar algum database criado por você.



Tabelas FUNCIONARIO e DEPARTAMENTO.

A partir das tabelas FUNCIONARIO e DEPARTAMENTO, segue o script:

sql

```
CREATE TABLE DEPARTAMENTO (  
    CODIGODEPARTAMENTO int NOT NULL,  
    NOME varchar(90) NOT NULL,  
    CONSTRAINT DEPARTAMENTO_pk PRIMARY KEY (CODIGODEPARTAMENTO));  
CREATE TABLE FUNCIONARIO (  
    CODIGOFUNCIONARIO int NOT NULL,  
    NOME varchar(90) NOT NULL,  
    CPF char(15) NULL,  
    SEXO char(1) NOT NULL,  
    DTNASCIMENTO date NOT NULL,  
    SALARIO real NULL,  
    CODIGODEPARTAMENTO int NULL,  
    CONSTRAINT FUNCIONARIO_pk PRIMARY KEY (CODIGOFUNCIONARIO));  
ALTER TABLE FUNCIONARIO ADD CONSTRAINT FUNCIONARIO_DEPARTAMENTO  
    FOREIGN KEY (CODIGODEPARTAMENTO) REFERENCES DEPARTAMENTO (CODIGODEPARTAMENTO);  
INSERT INTO DEPARTAMENTO(CODIGODEPARTAMENTO,NOME) VALUES (1,'Tecnologia da Informação');  
INSERT INTO DEPARTAMENTO(CODIGODEPARTAMENTO,NOME) VALUES (2,'Contabilidade');  
INSERT INTO DEPARTAMENTO(CODIGODEPARTAMENTO,NOME) VALUES (3,'Marketing');  
INSERT INTO FUNCIONARIO (CODIGOFUNCIONARIO, NOME, CPF, SEXO, DTNASCIMENTO, SALARIO,  
CODIGODEPARTAMENTO)  
VALUES (1, 'ROBERTA SILVA BRASIL',NULL, 'F', '20/02/1980',7000,1);  
INSERT INTO FUNCIONARIO (CODIGOFUNCIONARIO, NOME, CPF, SEXO, DTNASCIMENTO, SALARIO,  
CODIGODEPARTAMENTO)  
VALUES (2, 'MARIA SILVA BRASIL',NULL, 'F', '20/09/1988',9500,2);  
INSERT INTO FUNCIONARIO (CODIGOFUNCIONARIO, NOME, CPF, SEXO, DTNASCIMENTO, SALARIO,  
CODIGODEPARTAMENTO)  
VALUES (3, 'GABRIELLA PEREIRA LIMA',NULL, 'F', '20/02/1990',6000,1);  
INSERT INTO FUNCIONARIO (CODIGOFUNCIONARIO, NOME, CPF, SEXO, DTNASCIMENTO, SALARIO,  
CODIGODEPARTAMENTO)  
VALUES (4, 'MARCOS PEREIRA BRASIL',NULL, 'M', '20/02/1999',6000,2);  
INSERT INTO FUNCIONARIO (CODIGOFUNCIONARIO, NOME, CPF, SEXO, DTNASCIMENTO, SALARIO,  
CODIGODEPARTAMENTO)  
VALUES (5, 'HEMERSON SILVA BRASIL', NULL, 'M', '20/12/1992',4000,NULL);
```

A imagem a seguir apresenta o conteúdo da tabela DEPARTAMENTO após a execução do comando: SELECT * FROM DEPARTAMENTO;

	codigodepartamento	nome
1	1	Tecnologia da Informação
2	2	Contabilidade
3	3	Marketing

Conteúdo da tabela DEPARTAMENTO.

A próxima imagem apresenta o conteúdo da tabela FUNCIONARIO após a execução do comando:

TABLE FUNCIONARIO; (equivalente a SELECT * FROM FUNCIONARIO;)

	codigofuncionario	nome	cpf	sexo	dtnascimento	salario	codigodepartamento
1	1	ROBERTA SILVA BRASIL	[NULL]	F	1980-02-20	7.000	1
2	2	MARIA SILVA BRASIL	[NULL]	F	1988-09-20	9.500	2
3	3	GABRIELLA PEREIRA LIMA	[NULL]	F	1990-02-20	6.000	1
4	4	MARCOS PEREIRA BRASIL	[NULL]	M	1999-02-20	6.000	2
5	5	HEMERSON SILVA BRASIL	[NULL]	M	1992-12-20	4.000	[NULL]

Conteúdo da tabela FUNCIONARIO.

Perceba que, originalmente, existe um relacionamento do tipo 1:N entre DEPARTAMENTO e FUNCIONARIO. A implementação desse relacionamento ocorre por meio da chave estrangeira CODIGODEPARTAMENTO da tabela FUNCIONARIO.

Subconsultas aninhadas

Uma subconsulta aninhada ocorre quando é necessário obter dados que dependem do resultado de uma - ou mais - consulta(s) mais interna(s). Para isso, cria-se uma condição na cláusula WHERE de forma a envolver o resultado da subconsulta em algum tipo de teste. Vejamos alguns exemplos a seguir:

Consulta 01: Retornar o código e o nome do(s) funcionário(s) que ganha(m) o maior salário.

```
sql

SELECT CODIGOFUNCIONARIO, NOME
FROM FUNCIONARIO
WHERE SALARIO=
    (SELECT MAX(SALARIO)
     FROM FUNCIONARIO);
```

O resultado da consulta 01 está expresso na imagem a seguir:

	123 codigofuncionario	nome
1	2	MARIA SILVA BRASIL

Resultado da consulta 01.

Inicialmente, o SGBD processa a subconsulta, a qual retorna o valor do maior salário registrado na tabela FUNCIONARIO. Em seguida, esse resultado é utilizado na avaliação da cláusula WHERE. Finalmente, são exibidos os registros dos funcionários cujo valor de SALARIO satisfaz à condição da cláusula WHERE.

Consulta 02: retornar o código, o nome e o salário do(s) funcionário(s) que ganha(m) mais que a média salarial dos colaboradores.

```
sql

SELECT CODIGOFUNCIONARIO, NOME, SALARIO
FROM FUNCIONARIO
WHERE SALARIO>
    (SELECT AVG(SALARIO)
     FROM FUNCIONARIO);
```

O resultado da consulta 02 está expresso na imagem a seguir.

	123 codigofuncionario	nome	123 salario
1	1	ROBERTA SILVA BRASIL	7.000
2	2	MARIA SILVA BRASIL	9.500

Resultado da consulta 02.

Inicialmente, o SGBD processa a subconsulta, a qual retorna a média salarial a partir da tabela FUNCIONARIO. Em seguida, esse resultado é utilizado na avaliação da cláusula WHERE. Finalmente, são exibidas as linhas da tabela FUNCIONARIO com as colunas CODIGOFUNCIONARIO, NOME e SALARIO, cujo valor de SALARIO satisfaz à condição da cláusula WHERE.

Consulta 03: Retornar o código, o nome e o salário do(s) funcionário(s) que ganha(m) menos que a média salarial dos colaboradores do departamento de tecnologia da informação (TI).

sql

```
SELECT CODIGOFUNCIONARIO, NOME, SALARIO
FROM FUNCIONARIO
WHERE SALARIO<
  (SELECT AVG(SALARIO)
   FROM FUNCIONARIO
   WHERE CODIGODEPARTAMENTO IN (SELECT CODIGODEPARTAMENTO
                                FROM DEPARTAMENTO
                                WHERE NOME='Tecnologia da Informação'));
```

O resultado da consulta 03 está expresso na imagem a seguir.

	123 codigofuncionario	nome	123 salario
1	3	GABRIELLA PEREIRA LIMA	6.000
2	4	MARCOS PEREIRA BRASIL	6.000
3	5	HEMERSON SILVA BRASIL	4.000

Resultado da consulta 03.

Perceba que, para retornar os resultados de interesse, o SGBD precisa calcular a média salarial dos funcionários do departamento de TI. Para isso, a subconsulta da linha 4 – que calcula essa média – utiliza o resultado da subconsulta da linha 6, a qual recupera o código do departamento.

Há outra maneira de resolver a consulta 03, pois você pode trocar uma subconsulta por uma junção. O código a seguir produz os mesmos resultados.

sql




```
SELECT CODIGOFUNCIONARIO, NOME, SALARIO
FROM FUNCIONARIO
WHERE SALARIO<
  (SELECT AVG(SALARIO)
   FROM FUNCIONARIO F JOIN DEPARTAMENTO D ON
     (F.CODIGODEPARTAMENTO=D.CODIGODEPARTAMENTO)
   WHERE D.NOME='Tecnologia da Informação');
```

Consulta 04: quantos funcionários recebem menos que a funcionária que possui o maior salário entre as colaboradoras de sexo feminino?

sql

```
SELECT COUNT(*) AS QUANTIDADE
FROM FUNCIONARIO
WHERE SALARIO<
  (SELECT MAX(SALARIO)
   FROM FUNCIONARIO
   WHERE SEXO='F');
```

O resultado da consulta 04 está expresso na imagem a seguir.

	123  quantidade  
1	4

Resultado da consulta 04.

No exemplo, o SGBD recupera o maior salário entre as funcionárias. Em seguida, conta os registros cujo valor da coluna SALARIO é menor que o valor do salário em questão.

Atividade 1

Considere a tabela e o código SQL a seguir.

EMPREGADO		
CODIGOFUNCIONARIO	int	PK
NOME	char(90)	
CPF	char(15)	
SEXO	char(1)	
DTNASCIMENTO	date	
SALARIO	real	

```

1 SELECT COUNT(*) AS QUANTIDADE
2 FROM EMPREGADO
3 WHERE SALARIO <
4     (SELECT MAX(SALARIO)
5      FROM EMPREGADO );

```

A consulta retorna:

A

o número de empregados que ganham menos que o empregado de maior salário.

B

o número de empregados que ganham menos que a média salarial das colaboradoras.

C

o maior salário entre as colaboradoras.

D

o número de funcionárias.

E

o número de empregados que ganham igual à média salarial das colaboradoras.

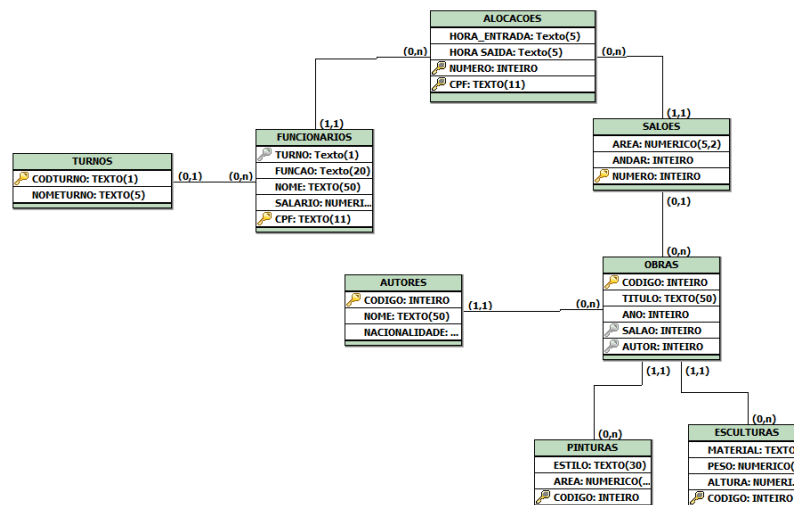


A alternativa A está correta.

A cláusula WHERE compara o salário de cada funcionário junto ao do colaborador que tem maior renda. Caso o salário do funcionário seja menor que o maior salário, esse funcionário é considerado na contagem.

Praticando as subconsultas aninhadas

Vamos analisar exemplos de uso dos comandos vistos até aqui. Para isso, iremos utilizar o banco de dados do museu. Seu modelo lógico é o seguinte:



Modelo lógico de banco de dados do minimundo MUSEU.

Praticando as subconsultas aninhadas

Neste vídeo, utilizando a linguagem SQL, pratique as subconsultas aninhadas. Mãos à obra!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Roteiro de prática

Para executar a atividade, faça conexão no banco do museu e escreva os seguintes comandos utilizando subconsulta:

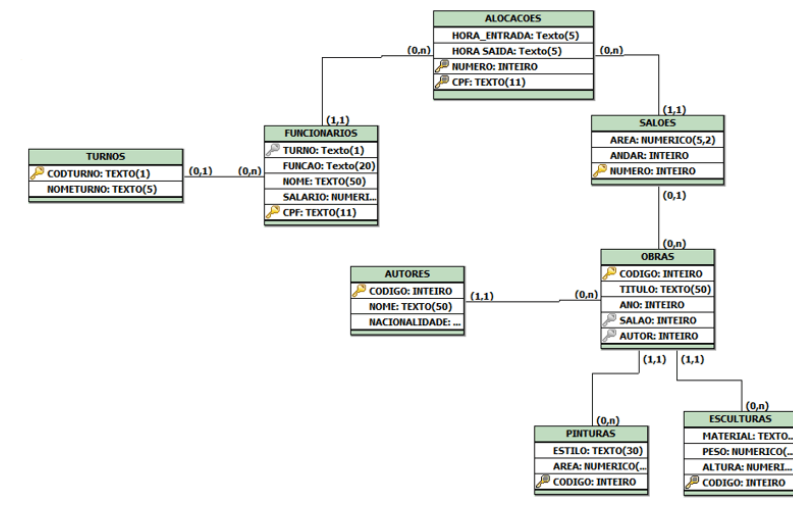
- Listar todos os dados de funcionários cujo salário seja maior que o salário médio de todos os funcionários.

- Listar todos os dados dos salões que possuem obras expostas.
- Listar todos os dados dos salões que não possuem obras expostas.
- Listar todos os dados dos autores que não possuem obras expostas no museu.
- Listar todos os dados dos funcionários que possuem alguma alocação.
- Listar todos os dados dos funcionários que não possuem alguma alocação.

Você pode baixar os comandos que resolvem as consultas no arquivo [sqlmod2.2](#).

Atividade 2

Vamos agora executar uma atividade prática utilizando o banco de dados da empresa. Seu modelo lógico é o seguinte:



Para realizar o exercício, faça conexão no banco da empresa e realize as seguintes consultas utilizando subconsultas:

- Listar todos os dados dos empregados que ganham mais que o salário médio da empresa.
- Listar todos os dados dos empregados que estão alocados em departamentos da região 1.
- Listar todos os dados dos empregados que atendem a algum cliente.

- Listar todos os dados dos empregados que não atendem a algum cliente.

Chave de resposta

Você pode baixar os comandos que resolvem as consultas no arquivo [exercmod2.2](#).

Subconsultas correlatas

Agora, vamos explorar as subconsultas correlatas, revelando como relacionar dados entre consultas para informações mais precisas e explorar técnicas avançadas para vincular informações de forma dinâmica. Você vai aprender a resolver problemas complexos de análise de dados com eficiência e clareza. Vamos lá!

Conheça neste vídeo o processo de aplicação das subconsultas correlatas. Descubra como essas consultas internas podem ser utilizadas para realizar operações complexas dentro do modelo relacional.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Uma subconsulta correlata – ou correlacionada – é um tipo especial de consulta aninhada.

A consulta correlata ocorre quando é necessário obter dados que dependem do resultado de uma (ou mais) consulta(s) mais interna(s). Para isso, cria-se uma condição na cláusula WHERE, de forma a envolver o resultado da subconsulta em algum tipo de teste.

Vejamos alguns exemplos!

Consulta 05: Retornar o código, o nome e o salário do(s) funcionário(s) que ganha(m) mais que a média salarial dos colaboradores do departamento ao qual pertencem.

sql

```
SELECT CODIGOFUNCIONARIO, NOME, SALARIO
FROM FUNCIONARIO F
WHERE SALARIO >
  (SELECT AVG(SALARIO)
   FROM FUNCIONARIO
   WHERE CODIGODEPARTAMENTO=F.CODIGODEPARTAMENTO);
```

O resultado da consulta 05 está expresso na imagem a seguir.

	codigofuncionario	nome	salario
1	1	ROBERTA SILVA BRASIL	7.000
2	2	MARIA SILVA BRASIL	9.500

Resultado da consulta 05.

Trata-se de uma consulta com lógica de construção semelhante ao que foi desenvolvido na consulta 02. No entanto, a média salarial é calculada levando em consideração somente os funcionários de cada departamento. Isso ocorre em função da condição WHERE declarada na linha 6.

Há outra maneira de resolver a consulta 05, pois você pode trocar uma subconsulta por uma junção. O código a seguir produz os mesmos resultados. Veja!

```
sql

SELECT CODIGOFUNCIONARIO, NOME, SALARIO
FROM FUNCIONARIO F
JOIN
    (SELECT CODIGODEPARTAMENTO,AVG(SALARIO) AS MEDIA
    FROM FUNCIONARIO
    GROUP BY CODIGODEPARTAMENTO) TESTE
ON F.CODIGODEPARTAMENTO=TESTE.CODIGODEPARTAMENTO
WHERE SALARIO>MEDIA;
```

O exemplo a seguir mostra o uso de uma consulta correlacionada em uma operação de atualização (UPDATE) nos dados.

Consulta 06: suponha que surgiu a necessidade de equiparar os salários dos funcionários que atuam no mesmo departamento. Os funcionários de cada departamento terão salário atualizado em função do maior salário dos seus setores.

Observe na imagem a seguir os salários “antes” da atualização.

	codigofuncionario	nome	cpf	sexo	dnascimento	salario	codigodepartamento
1	1	ROBERTA SILVA BRASIL	[NULL]	F	1980-02-20	7.000	1
2	3	GABRIELLA PEREIRA LIMA	[NULL]	F	1990-02-20	6.000	1
3	2	MARIA SILVA BRASIL	[NULL]	F	1988-09-20	9.500	2
4	4	MARCOS PEREIRA BRASIL	[NULL]	M	1999-02-20	6.000	2
5	5	HEMERSON SILVA BRASIL	[NULL]	M	1992-12-20	4.000	[NULL]

Tabela FUNCIONARIO antes da atualização dos salários.

Perceba que a listagem está ordenada pela coluna CODIGODEPARTAMENTO. O maior salário de funcionário pertencente ao departamento 1 é R\$ 7.000; em relação ao departamento 2, R\$ 9.500. Note também que há um funcionário sem a informação sobre departamento.

```
sql

UPDATE FUNCIONARIO F
SET SALARIO=
    (SELECT MAX(SALARIO)
    FROM FUNCIONARIO
    WHERE CODIGODEPARTAMENTO=F.CODIGODEPARTAMENTO)
WHERE F.CODIGODEPARTAMENTO IS NOT NULL;
```

Observe na imagem a seguir os salários após a atualização salarial.

	codigofuncionario	nome	cpf	sexo	dnascimento	salario	codigodepartamento
1	1	ROBERTA SILVA BRASIL	[NULL]	F	1980-02-20	7.000	1
2	3	GABRIELLA PEREIRA LIMA	[NULL]	F	1990-02-20	7.000	1
3	2	MARIA SILVA BRASIL	[NULL]	F	1988-09-20	9.500	2
4	4	MARCOS PEREIRA BRASIL	[NULL]	M	1999-02-20	9.500	2
5	5	HEMERSON SILVA BRASIL	[NULL]	M	1992-12-20	4.000	[NULL]

Tabela FUNCIONARIO depois da atualização dos salários.

Trata-se de uma consulta com objetivo de recuperar o maior salário dentro do contexto de cada departamento e, em seguida, usar esse valor para atualização salarial na tabela FUNCIONARIO, de acordo com o departamento de cada colaborador. Perceba também que a atualização ocorre somente para os funcionários para os quais existe alocação a departamento, ou seja, a cláusula WHERE da linha 6 inibe atualização de salário caso não haja departamento associado a algum colaborador.

A sintaxe completa do comando UPDATE no PostgreSQL pode ser encontrada com a indicação disponibilizada no Explore + ao final do material.

Atividade 3

Considerando que uma subconsulta correlata é executada repetidamente para cada linha processada pela consulta externa, usando valores da linha corrente, permitindo filtragem dinâmica e comparação contextualizada em SQL, qual das seguintes consultas SQL exemplifica o uso de uma subconsulta correlacionada?

A

```
SELECT * FROM tabela1 WHERE c1 IN (SELECT c1 FROM tabela2);
```

B

```
SELECT * FROM tabela1 JOIN tabela2 ON tabela1.id = tabela2.id;
```

C

```
SELECT MAX(coluna) FROM tabela1;
```

D

```
SELECT * FROM tabela1 t1 WHERE coluna >= (SELECT AVG(coluna) FROM tabela2 t2 where t1.c1 = t2.c1);
```

E

```
SELECT * FROM tabela1 where id = (SELECT * FROM tabela2 WHERE tabela1.id = tabela2.id);
```



A alternativa D está correta.

A consulta D utiliza uma subconsulta correlacionada ao comparar a coluna da tabela 1 com o resultado da média (AVG) da coluna da tabela 2. A subconsulta depende dos resultados da consulta externa para sua execução, tornando-a correlacionada. As outras opções não demonstram o uso de subconsultas correlacionadas.

Consulta correlacionada com uso de [NOT] EXISTS

Vamos compreender o teste de existência no SQL, destacando como verificar a presença de registros em consultas complexas. Você vai aprender a utilizar operadores de teste para garantir resultados precisos em suas análises de dados.

Acompanhe neste vídeo a aplicação da consulta correlacionada com uso do operador [NOT] EXISTS, e aprenda como escrever consultas correlacionadas eficientes, utilizando exemplos práticos em cenários reais.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Podemos utilizar o operador EXISTS em uma consulta correlacionada. Tal operador testa a existência de linha(s) retornada(s) por alguma subconsulta. Veja um exemplo!

Consulta 07: exibir o código e o nome do departamento em que há pelo menos um funcionário alocado.

sql

```
SELECT D.CODIGODEPARTAMENTO, D.NOME
FROM DEPARTAMENTO D
WHERE EXISTS
  (SELECT F.CODIGODEPARTAMENTO
   FROM FUNCIONARIO F
   WHERE D.CODIGODEPARTAMENTO=F.CODIGODEPARTAMENTO);
```

Observe na imagem a seguir os resultados da consulta.

	codigodepartamento	nome
1	1	Tecnologia da Informação
2	2	Contabilidade

Departamentos em que há pelo menos um funcionário alocado.

A subconsulta correlacionada (linhas 4 a 6) é executada. Caso haja pelo menos uma linha em seu retorno, a avaliação da cláusula WHERE retorna verdadeiro e as colunas especificadas na linha 1 são exibidas.

Finalmente, se estivéssemos interessados em saber se há departamento sem ocorrência de colaborador alocado, bastaria usar a negação (NOT), conforme a seguir.

sql

```
SELECT D.CODIGODEPARTAMENTO, D.NOME
FROM DEPARTAMENTO D
WHERE NOT EXISTS
  (SELECT F.CODIGODEPARTAMENTO
   FROM FUNCIONARIO F
   WHERE D.CODIGODEPARTAMENTO=F.CODIGODEPARTAMENTO);
```

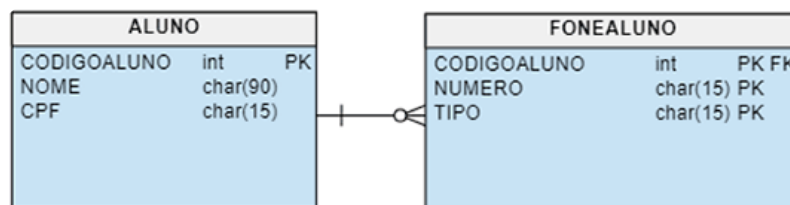
Observe na imagem seguinte os resultados da consulta.

	codigodepartamento	nome
1	3	Marketing

Departamento em que não há funcionário alocado.

Atividade 4

Considere as tabelas e o código SQL a seguir.



```
1 SELECT CODIGOALUNO, NOME
2 FROM ALUNO
3 WHERE EXISTS
4   (SELECT CODIGOALUNO
5    FROM FONEALUNO
6    WHERE FONEALUNO.CODIGOALUNO=ALUNO.CODIGOALUNO);
```

A consulta retorna:

A

somente o(s) registro(s) de aluno(s) sem telefone.

B

o(s) registro(s) de aluno(s) com pelo menos um telefone.

C

somente o(s) registro(s) de aluno(s) com pelo menos dois telefones.

D

somente o(s) registro(s) de aluno(s) com pelo menos três telefones.

E

somente o(s) registro(s) de aluno(s) com pelo menos quatro telefones.

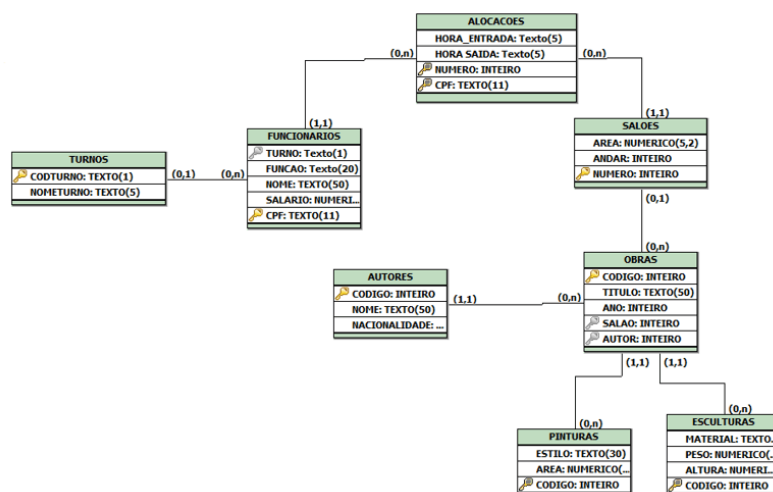


A alternativa B está correta.

Há o uso de uma subconsulta correlata, junto à cláusula EXISTS, que testa a existência de algum retorno da subconsulta. Assim, caso haja retorno da consulta, significa que o aluno em questão possui pelo menos um telefone.

Praticando as subconsultas correlatas

Vamos analisar exemplos de uso dos comandos vistos até aqui. Para isso, iremos utilizar o banco de dados do museu. Seu modelo lógico é o seguinte:



Modelo lógico de banco de dados do minimundo MUSEU.

Praticando as subconsultas correlatas

Neste vídeo, você vai explorar as chamadas subconsultas correlatas, e entender como essas consultas avançadas são utilizadas para resolver problemas complexos que envolvem relações entre tabelas.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Roteiro de prática

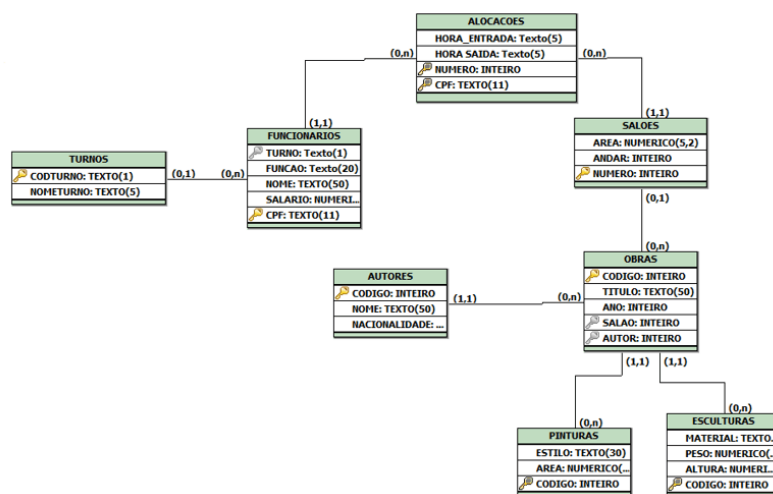
Para executar a atividade, faça conexão no banco do museu e escreva os seguintes comandos utilizando subconsulta:

- Listar todos os dados dos salões cuja área seja maior que a área média dos salões do mesmo andar.
- Listar todos os dados dos autores que possuem obras utilizando teste de existência.
- Listar todos os dados dos autores que não possuem obras utilizando teste de existência.
- Listar todos os dados dos autores que possuem mais pinturas que esculturas.

Você pode baixar os comandos que resolvem as consultas no arquivo [sqlmod2.5](#).

Atividade 5

Vamos agora executar uma atividade prática utilizando o banco de dados da empresa. Seu modelo lógico é o seguinte:



Modelo lógico de banco de dados do minimundo EMPRESA.

Para realizar o exercício, faça conexão no banco da empresa e realize as seguintes consultas utilizando subconsultas:

- Listar os empregados que ganham pelo menos o salário médio de seu departamento.

- Listar o ID, ULT_NOME e cargo dos empregados que atendem a algum cliente, utilizando teste de existência.
- Listar o ID, ULT_NOME e cargo dos empregados que não atendem a algum cliente, utilizando teste de existência.

Chave de resposta

Você pode baixar os comandos que resolvem as consultas no arquivo [exercmod2.5](#) .

3. Consultas envolvendo operadores de conjunto

Operadores de conjunto

Vamos explorar os operadores de conjunto no SQL, apresentando como combinar, interseccionar e diferenciar conjuntos de dados de maneira eficaz.

Acompanhe neste vídeo a aplicação dos operadores de conjunto e aprenda como utilizá-los para realizar uniões de conjuntos, encontrar interseções e realizar diferenças entre conjuntos de dados.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Vamos aprender que os resultados de diversas consultas podem ser combinados em um único conjunto de dados, caso sigam regras específicas dos operadores utilizados para essa finalidade. Estamos falando dos operadores de conjunto, que incluem UNION, INTERSECT e EXCEPT.

Construiremos as consultas com base nas tabelas FUNCIONARIO, ALUNO e CLIENTE, conforme imagem a seguir. Acompanhe!

FUNCIONARIO			
CODIGOFUNCIONARIO	int	PK	
NOME	varchar(90)		
CPF	char(15)	N	
SEXO	char(1)		
DTNASCIMENTO	date		
SALARIO	real	N	

ALUNO			
CODIGOALUNO	int	PK	
NOME	varchar(90)		
CPF	char(15)		
SEXO	char(1)		
DTNASCIMENTO	date		

CLIENTE			
CODIGOCLIENTE	int	PK	
NOME	varchar(90)		
CPF	char(15)		
SEXO	char(1)		

Tabelas FUNCIONARIO, ALUNO e CLIENTE.

Recomendamos que você crie as tabelas e insira algumas linhas, o que pode ser feito usando o script a seguir, a partir da ferramenta de sua preferência. Para isso, tenha em mente que é necessário estar conectado ao PostgreSQL e acessar algum database criado por você.

```
sql

CREATE TABLE FUNCIONARIO (
    CODIGOFUNCIONARIO int NOT NULL,
    NOME varchar(90) NOT NULL,
    CPF char(15) NULL,
    SEXO char(1) NOT NULL,
    DTNASCIMENTO date NOT NULL,
    SALARIO real NULL,
    CONSTRAINT FUNCIONARIO_pk PRIMARY KEY (CODIGOFUNCIONARIO));
CREATE TABLE ALUNO (
    CODIGOALUNO int NOT NULL,
    NOME varchar(90) NOT NULL,
    CPF char(15) NOT NULL,
    SEXO char(1) NOT NULL,
    DTNASCIMENTO date NOT NULL,
    CONSTRAINT ALUNO_pk PRIMARY KEY (CODIGOALUNO));
CREATE TABLE CLIENTE (
    CODIGOCLIENTE int NOT NULL,
    NOME varchar(90) NOT NULL,
    CPF char(15) NOT NULL,
    SEXO char(1) NOT NULL,
    CONSTRAINT CLIENTE_pk PRIMARY KEY (CODIGOCLIENTE));
```

O script a seguir pode ser utilizado para inserção de registros nas tabelas.

sql

```
INSERT INTO FUNCIOILARIO (COOIGOFUNCIONARIO, NOME, CPF, SEXO, DTMASCIMENTO, SALARIO)
VALUES (1, 'ROBERTA SILVA BRASIL', '82998', 'F', '20/02/1980',70e0);
INSERT INTO FUNCIOILARIO (COOIGOFUNCIONARIO, NOME, CPF, SEXO, DTMASCIMENTO, SALARIO)
VALUES (2, 'MARIA SILVA BRASIL', '9876', 'F', '20/09/1988',9500);
INSERT INTO FUNCIOIARIO (COOIGOFUNCIONARIO, NOME, CPF, SEXO, DTNASCIMENTO, SALARIO)
VALUES (3, 'GABRIELLA PEREIRA LIMA', '32998', 'F', '20/02/1990',6000);
INSERT INTO FUNCIOHLARIO (COOIGOFUNCIONARIO, NOME, CPF, SEXO, DTMASCIMEMTO, SALARIO)
VALUES (4, 'MARCOS PEREIRA BRASIL', '9999', 'M', '20/02/1999',60e0);
INSERT INTO FUNCIOHLARIO (COOIGOFUNCIONARIO, NOME, CPF, SEXO, DTMASCIMENTO, SALARIO)
VALUES (5, 'HEMERSCUI SILVA BRASIL', '9111', 'H', '20/12/1992',40e0);
INSERT INTO ALUNO (CODIGOALUNO, NOME, CPF, SEXO, DTNASCIMENTO) VALUES (1, 'JOSÉ FRANCISCO
TERRA', '82988', 'M', '28/10/1989');
INSERT INTO ALUNO (CODIGOALUNO, NOME, CPF, SEXO, DTNASCIMENTO) VALUES (2, 'ANDREY COSTA
FILHO', '0024', 'M', '20/10/1999');
INSERT INTO ALUNO (CODIGOALUNO, NONE, CPF, SEXO, DTNASCIMENTO) VALUES (3, 'ROBERTA SILVA
BRASIL', '82998', 'F', '20/02/1980');
INSERT INTO ALUNO (CODIGOALUNO, NOME, CPF, SEXO, DTNASCIMENTO) VALUES (4, 'CARLA MARIA
MACIEL', '0044', 'F', '20/11/1996');
INSERT INTO ALUNO (CODIGOALUNO, NOME, CPF, SEXO, DTNASCIMENTO) VALUES (5, 'MARCOS PEREIRA
BRASIL', '9999', 'M', '20/E2/1999');
INSERT INTO CLIENTE (COOIGOCLIENTE, NONE, CPF, SEXO) VALUES (1, 'ROBERTA SILVA BRASIL',
'82998', 'F');
INSERT INTO CLIENTE (COOIGOCLIENTE, NOHE, CPF, SEXO) VALUES (2, 'MARCOS PEREIRA BRASIL',
'9999', 'M');
20 INSERT INTO CLIENTE (COOIGOCLIENTE, NOHE, CPF, SEXO) VALUES (3, 'HEMERSON SILVA
BRASIL', '9111', 'M');
```

A imagem a seguir apresenta o conteúdo da tabela FUNCIONARIO após a execução do comando: SELECT * FROM FUNCIONARIO;

	codigofuncionario	nome	cpf	sexo	dtnascimento	salario
1	1	ROBERTA SILVA BRASIL	82998	F	1980-02-20	7.000
2	2	MARIA SILVA BRASIL	9876	F	1988-09-20	9.500
3	3	GABRIELLA PEREIRA LIMA	32998	F	1990-02-20	6.000
4	4	MARCOS PEREIRA BRASIL	9999	M	1999-02-20	6.000
5	5	HEMERSON SILVA BRASIL	9111	M	1992-12-20	4.000

Conteúdo da tabela FUNCIONARIO.

A imagem a seguir apresenta o conteúdo da tabela ALUNO após a execução do comando: TABLE ALUNO;

	codigoaluno	nome	cpf	sexo	dtnascimento
1	1	JOSÉ FRANCISCO TERRA	82988	M	1989-10-28
2	2	ANDREY COSTA FILHO	0024	M	1999-10-20
3	3	ROBERTA SILVA BRASIL	82998	F	1980-02-20
4	4	CARLA MARIA MACIEL	0044	F	1996-11-20
5	5	MARCOS PEREIRA BRASIL	9999	M	1999-02-20

Conteúdo da tabela ALUNO.

A imagem a seguir apresenta o conteúdo da tabela CLIENTE após a execução do comando: TABLE CLIENTE;

	codigocliente	nome	cpf	sexo
1	1	ROBERTA SILVA BRASIL	82998	F
2	2	MARCOS PEREIRA BRASIL	9999	M
3	3	HEMERSON SILVA BRASIL	9111	M

Conteúdo da tabela CLIENTE.

Convém esclarecer que, em todos os registros, os dados são puramente fictícios. Além disso, os dados da coluna CPF estão com somente quatro caracteres e não representam um CPF válido.

No entanto, vamos considerar, para efeitos de nosso estudo, que registros com mesmo valor de CPF representam a informação de um mesmo cidadão. De forma complementar, estamos considerando que cada tabela pertence a um banco de dados – e um domínio de aplicação – diferente.

Atividade 1

A operação UNION em banco de dados combina resultados de duas ou mais consultas SELECT em um único conjunto de resultados, eliminando duplicatas, e unindo dados de forma consolidada. Qual das seguintes opções descreve corretamente a compatibilidade na união de conjuntos em SQL?

A

A compatibilidade na união de conjuntos significa que os tipos de dados dos atributos dos conjuntos devem ser os mesmos.

B

A compatibilidade na união de conjuntos refere-se à necessidade de os conjuntos terem o mesmo número de elementos.

C

A compatibilidade na união de conjuntos requer que os conjuntos estejam armazenados na mesma tabela do banco de dados.

D

A compatibilidade na união de conjuntos exige que os conjuntos tenham chaves primárias distintas.

E

A compatibilidade na união de conjuntos significa que os conjuntos devem ter o mesmo número de colunas e os tipos de dados das colunas correspondentes devem ser compatíveis.



A alternativa E está correta.

Na união de conjuntos em SQL, é essencial que os conjuntos tenham o mesmo número de colunas, e as colunas correspondentes devem ser do mesmo tipo de dados ou de tipos de dados compatíveis. Dessa forma, a operação de união pode ser realizada sem erros devido a diferenças estruturais entre os conjuntos

Consultas com o operador UNION

Vamos explorar agora o operador UNION, revelando como combinar e consolidar conjuntos de dados de diferentes fontes em uma única consulta. Você vai aprender a utilizar o UNION a partir de diversos exemplos. Vamos lá!

Confira neste vídeo a importância da aplicação de consultas com o operador UNION, que é essencial para combinar resultados de consultas de maneira eficiente e flexível, permitindo a criação de conjuntos de dados completos a partir de múltiplas fontes.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

O operador de união serve para consolidar linhas resultantes de consultas. Para isso, todas as consultas envolvidas devem possuir a mesma quantidade de colunas e deve haver compatibilidade de tipo de dados. Além disso, linhas repetidas são eliminadas do resultado, uma vez que o resultado é uma tabela que não permite duplicata de linhas. O operador de união possui a seguinte forma geral:

CONSULTASQL UNION [ALL|DISTINCT] CONSULTASQL

A sintaxe completa do comando SELECT no PostgreSQL pode ser encontrada com a indicação disponibilizada no Explore + ao final do material.

Consulta 01: retornar o nome e o CPF de todos os funcionários e clientes.

sql

```
1 SELECT NOME, CPF
2 FROM FUNCIONARIO
3 UNION
4 SELECT NOME, CPF
5 FROM CLIENTE;
```

O resultado da consulta 01 está expresso na imagem a seguir.

	nome	cpf
1	ROBERTA SILVA BRASIL	82998
2	GABRIELLA PEREIRA LIMA	32998
3	MARCOS PEREIRA BRASIL	9999
4	MARIA SILVA BRASIL	9876
5	HEMERSON SILVA BRASIL	9111

Resultado da consulta 01.

Perceba que há cinco funcionários cadastrados (linhas 1 a 5 do script de inserção) e, de forma semelhante, três clientes (linhas 18 a 20 do script de inserção). Note também que todos os clientes cadastrados também são funcionários. Após o processamento da operação de união, somente cinco registros foram exibidos, uma vez que as repetições por padrão são eliminadas.

E se quiséssemos que todos os registros aparecessem no resultado? Bastaria usar o UNION ALL, conforme a seguir.

sql

```
SELECT NOME, CPF
FROM FUNCIONARIO
UNION ALL
SELECT NOME, CPF
FROM CLIENTE;
```

O resultado da consulta 01 modificada está expresso na imagem a seguir.

	nome	cpf
1	ROBERTA SILVA BRASIL	82998
2	MARIA SILVA BRASIL	9876
3	GABRIELLA PEREIRA LIMA	32998
4	MARCOS PEREIRA BRASIL	9999
5	HEMERSON SILVA BRASIL	9111
6	ROBERTA SILVA BRASIL	82998
7	MARCOS PEREIRA BRASIL	9999
8	HEMERSON SILVA BRASIL	9111

Resultado da consulta 01 modificada para contemplar todos os registros das tabelas.

Finalmente, se quiséssemos especificar a “origem” de cada registro, poderíamos alterar o nosso código conforme a seguir.

```
sql
```

```
SELECT NOME, CPF, 'Dados da tabela FUNCIONARIO' AS ORIGEM
FROM FUNCIONARIO
UNION ALL
SELECT NOME, CPF, 'Dados da tabela CLIENTE' AS ORIGEM
FROM CLIENTE;
```

O resultado da consulta 01 modificada está expresso na imagem a seguir.

	nome	cpf	origem
1	ROBERTA SILVA BRASIL	82998	Dados da tabela FUNCIONARIO
2	MARIA SILVA BRASIL	9876	Dados da tabela FUNCIONARIO
3	GABRIELLA PEREIRA LIMA	32998	Dados da tabela FUNCIONARIO
4	MARCOS PEREIRA BRASIL	9999	Dados da tabela FUNCIONARIO
5	HEMERSON SILVA BRASIL	9111	Dados da tabela FUNCIONARIO
6	ROBERTA SILVA BRASIL	82998	Dados da tabela CLIENTE
7	MARCOS PEREIRA BRASIL	9999	Dados da tabela CLIENTE
8	HEMERSON SILVA BRASIL	9111	Dados da tabela CLIENTE

Resultado da consulta 01 modificada para contemplar todos os registros das tabelas.

Atividade 2

A imagem mostra o conteúdo de algumas colunas da tabela empregado em um SGBD PostgreSQL. Confira!

id [PK] numeric (7)	ult_nome character varying (20)	cargo character varying (30)	id_depto numeric (7)	salario numeric (7,2)
1	Velasques	Presidente	10	37695.00
2	Neves	Diretor de Compras	30	25595.00
3	Nogueira	Diretor de Vendas	20	23780.00
4	Queiroz	Gerente de Compras	30	11680.00
5	Rodrigues	Vendedor	20	6840.00
6	Ugarte	Vendedor	20	6235.00

Se fosse emitido o seguinte comando retornariam alguns empregados de ID.

```
SELECT ID, ULT_NOME, CARGO, ID_DEPTO ,SALARIO
FROM EMPREGADO WHERE SALARIO BETWEEN 6500 AND 25000
UNION
SELECT ID, ULT_NOME, CARGO, ID_DEPTO ,SALARIO
FROM EMPREGADO WHERE ID_DEPTO IN (10, 20)
```

Dito isso, assinale a alternativa que apresenta corretamente quais seriam esses empregados.

A

2, 4, 5 e 6.

B

2, 3, 4, 5 e 6.

C

1, 3, 5, e 6.

D

1, 3, 4, 5 e 6.

E

1, 2, 3, 4, 5 e 6.

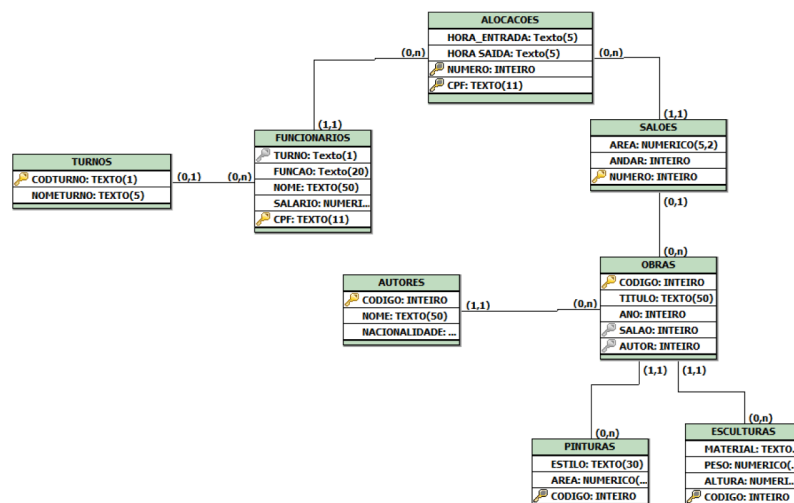


A alternativa D está correta.

No comando emitido, o primeiro conjunto é formado pelos empregados de ID 3, 4 e 5. Já o segundo conjunto é composto pelos empregados de ID 1, 3, 5 e 6. Como foi solicitada a união, o resultado é composto pelos empregados de ID 1, 3, 4, 5 e 6.

Praticando consultas com o operador UNION

Vamos agora analisar exemplos de uso dos comandos vistos até aqui. Para isso, iremos utilizar o banco de dados do museu. Seu modelo lógico é o seguinte:



Modelo lógico de banco de dados do minimundo MUSEU.

Praticando consultas com o operador UNION

Neste vídeo, vamos explorar consultas utilizando o operador UNION. Aprenda passo a passo como escrever consultas eficientes utilizando UNION para integrar e consolidar informações de diferentes origens. Prepare-se para aprimorar suas habilidades em SQL e explorar as capacidades versáteis desse operador!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Roteiro de prática

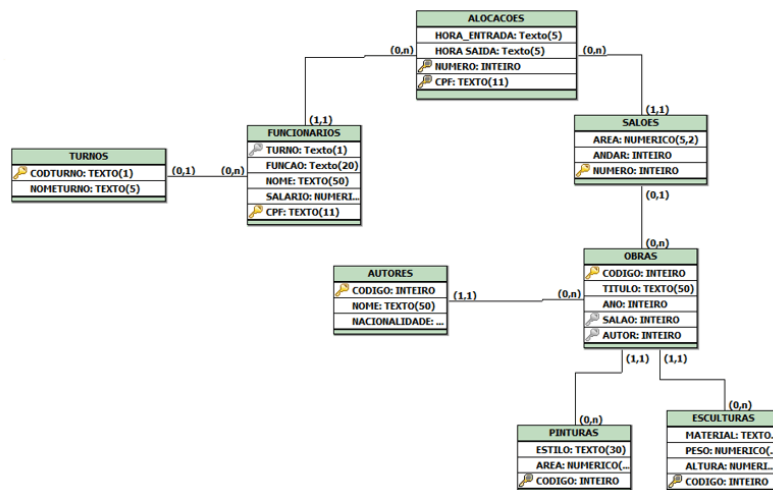
Para executar a atividade, faça conexão no banco do museu e escreva os seguintes comandos utilizando UNION:

- Listar o número dos salões que possuem obras ou que possuem funcionários alocados.
- Listar o número dos salões que possuem obras ou que possuem funcionários alocados, listando os duplicados.
- Listar o código das obras do estilo impressionista ou feitas de argila.
- Listar o código das obras do estilo impressionista ou feitas de argila, repetindo os duplicados.
- Listar o título e o nome do autor cujas obras são do estilo impressionista ou feitas de argila.

Você pode baixar os comandos que resolvem as consultas no arquivo [sqlmod3.3](#).

Atividade 3

Vamos agora executar uma atividade prática utilizando o banco de dados da empresa. Seu modelo lógico é o seguinte:



Modelo lógico de banco de dados do minimundo EMPRESA.

Para realizar o exercício, faça conexão no banco da empresa e realize as seguintes consultas utilizando UNION:

- Listar o ID dos empregados que são vendedores ou que trabalham nos departamentos 20 ou 30.
- Listar o ID dos empregados que são vendedores ou que trabalham nos departamentos 20 ou 30. Repetir os duplicados.
- Listar todos os dados dos empregados que são vendedores ou que trabalham nos departamentos 10 ou 30.
- Listar todos os dados dos departamentos que estão na região 1 ou que possuem empregados com cargo de vendedor.

Chave de resposta

Você pode baixar os comandos que resolvem as consultas no arquivo [exercmod3.3](#).

Consultas com os operadores INTERSECT e EXCEPT

Vamos explorar agora os operadores INTERSECT e EXCEPT, revelando os dados em comum, ou os diferentes, em um conjunto originário de diferentes fontes em uma única consulta.

Neste vídeo, vamos explorar a utilização dos operadores de conjunto INTERSECT e EXCEPT em SQL, que são fundamentais para comparar e manipular conjuntos de resultados de consultas de maneira precisa e eficiente.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Consultas com o operador INTERSECT

O operador de interseção serve para exibir linhas que aparecem em ambos os resultados das consultas envolvidas. Para isso, todas as consultas devem possuir a mesma quantidade de colunas e deve haver compatibilidade de tipos de dados. Além disso, linhas repetidas são eliminadas do resultado. O operador de interseção possui a seguinte forma geral:

```
CONSULTASQL INTERSECT [ALL|DISTINCT] CONSULTASQL
```

A seguir, veremos alguns exemplos. Vamos lá!

Consulta 02: retornar o nome e o CPF de todos os cidadãos que são funcionários e clientes.


```
sql
```

```
SELECT NOME, CPF  
FROM FUNCIONARIO  
INTERSECT  
SELECT NOME, CPF  
FROM CLIENTE;
```

O resultado da consulta 02 está expresso na imagem a seguir.

	nome	cpf
1	MARCOS PEREIRA BRASIL	9999
2	ROBERTA SILVA BRASIL	82998
3	HEMERSON SILVA BRASIL	9111

Resultado da consulta 02.

A consulta retorna três linhas que são fruto da interseção entre as tabelas FUNCIONARIO e CLIENTE. Como visto, todos os clientes são funcionários.

Consulta 03: retornar o nome e o CPF de todos os cidadãos que são funcionários, clientes e alunos.

```
sql
```

```
1 SELECT NOME, CPF  
2 FROM FUNCIONARIO  
3 INTERSECT  
4 SELECT NOME, CPF  
5 FROM CLIENTE  
6 INTERSECT  
8 FROM ALUNO;
```

O resultado da consulta 03 está expresso na próxima imagem.

	nome	cpf
1	MARCOS PEREIRA BRASIL	9999
2	ROBERTA SILVA BRASIL	82998

Resultado da consulta 03.

A consulta retorna duas linhas que são fruto da interseção entre as tabelas FUNCIONARIO, CLIENTE e ALUNO.

Um aspecto importante é que uma consulta sob o formato X UNION Y INTERSECT Z é interpretada sendo X UNION (Y INTERSECT Z). Veja o exemplo!

```
sql
```

```
1 SELECT NOME, CPF  
2 FROM FUNCIONARIO  
3 UNION  
4 SELECT NOME, CPF  
5 FROM CLIENTE  
6 INTERSECT  
8 FROM ALUNO;
```

O resultado da consulta envolvendo as três tabelas está expresso na imagem a seguir.

	nome	cpf
1	ROBERTA SILVA BRASIL	82998
2	GABRIELLA PEREIRA LIMA	32998
3	MARCOS PEREIRA BRASIL	9999
4	MARIA SILVA BRASIL	9876
5	HEMERSON SILVA BRASIL	9111

Resultado da consulta envolvendo três tabelas.

Consultas com o operador EXCEPT

O operador EXCEPT implementa a operação de subtração da teoria dos conjuntos e serve para exibir linhas que aparecem em uma consulta e não aparecem na outra. Para isso, todas as consultas devem possuir a mesma quantidade de colunas e deve haver compatibilidade de tipos de dados. Além disso, linhas repetidas são eliminadas do resultado. O operador de subtração possui a seguinte forma geral:

```
CONSULTASQL EXCEPT [ALL|DISTINCT] CONSULTASQL
```

Alguns SGBDs implementam a mesmo operador, usando um nome diferente. O Oracle, por exemplo, utiliza o operador MINUS, significando subtração ou diferença. Vejamos alguns exemplos!

Consulta 04: retornar o nome e o CPF dos funcionários que não são clientes.

```
sql
SELECT NOME, CPF
FROM FUNCIONARIO
EXCEPT
SELECT NOME, CPF
FROM CLIENTE;
```

O resultado da consulta 04 está expresso na imagem a seguir.

	nome	cpf
1	GABRIELLA PEREIRA LIMA	32998
2	MARIA SILVA BRASIL	9876

Resultado da consulta 04.

A consulta retorna duas linhas que são fruto da subtração entre as tabelas FUNCIONARIO e CLIENTE.

Perceba que uma operação X EXCEPT Y é diferente de Y EXCEPT X. Veja o código!

```
sql
SELECT NOME, CPF
FROM CLIENTE
EXCEPT
SELECT NOME, CPF
FROM FUNCIONARIO;
```

A consulta retorna vazio, pois todos os clientes são funcionários.

Consulta 05: retornar o nome e o CPF dos cidadãos que são somente funcionários.

```
sql
```

```
SELECT NOME, CPF  
FROM FUNCIONARIO  
EXCEPT  
SELECT NOME, CPF  
FROM CLIENTE  
EXCEPT  
SELECT NOME, CPF  
FROM ALUNO;
```

O resultado da consulta 05 está expresso na próxima imagem.

Inicialmente, o SGBD processa a operação de subtração da linha 3. Em seguida, o resultado da operação é usado na subtração da linha 6.

Atividade 4

A imagem mostra o conteúdo de algumas colunas da tabela empregado em um SGBD PostgreSQL.

id [PK] numeric (7)	ult_nome character varying (20)	cargo character varying (30)	id_depto numeric (7)	salario numeric (7,2)
1	Velasques	Presidente	10	37695.00
2	Neves	Diretor de Compras	30	25595.00
3	Nogueira	Diretor de Vendas	20	23780.00
4	Queiroz	Gerente de Compras	30	11680.00
5	Rodrigues	Vendedor	20	6840.00
6	Ugarte	Vendedor	20	6235.00

Tabela EMPREGADO.

```
SELECT ID FROM EMPREGADO WHERE ID_DEPTO IN (20,30)  
INTERSECT  
SELECT ID FROM EMPREGADO WHERE UPPER(CARGO) = 'VENDEDOR'
```

Dito isso, assinale a alternativa que apresenta corretamente quais seriam esses empregados:

A

5 e 6.

B

2, 3 e 6.

C

3, 5 e 6.

D

4, 5 e 6.

E

1, 2, 5 e 6.

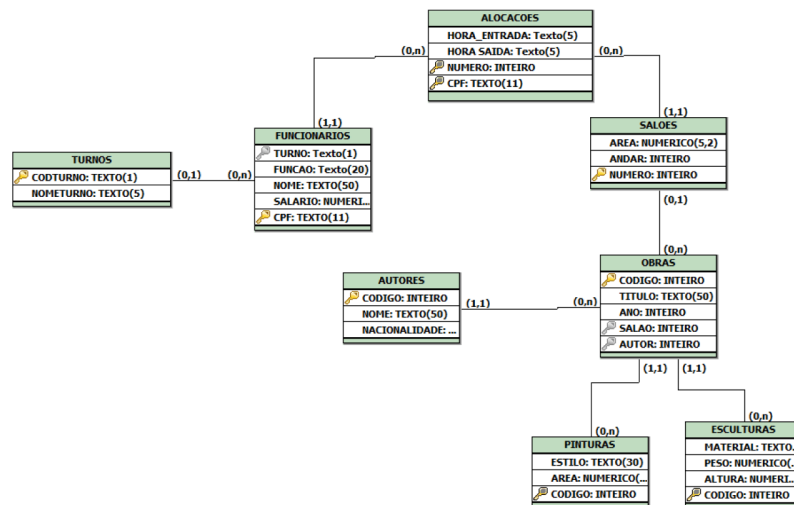


A alternativa A está correta.

No comando emitido, o primeiro conjunto é formado pelos empregados de ID 3, 4 e 5. Já o segundo conjunto é composto pelos empregados de ID 1, 3, 5 e 6. Como foi solicitada a intersecção, o resultado é composto pelos empregados de ID 5 e 6.

Praticando os operadores INTERSECT e EXCEPT

Vamos agora analisar exemplos de uso dos comandos vistos até aqui. Para isso, iremos utilizar o banco de dados do museu. Seu modelo lógico é o seguinte:



Modelo lógico de banco de dados do minimundo MUSEU.

Praticando os operadores INTERSECT e EXCEPT

Neste vídeo, vamos aprofundar nossos conhecimentos praticando consultas com os operadores INTERSECT e EXCEPT. Prepare-se para aprimorar suas habilidades em SQL e explorar o potencial desses importantes operadores.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Roteiro de prática

Para executar a atividade, faça conexão no banco do museu e escreva os seguintes comandos utilizando EXCEPT e INTERSECT:

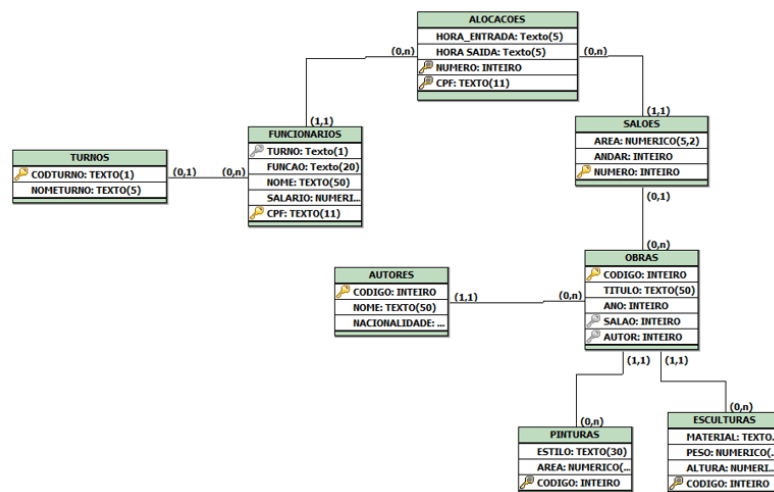
- Listar código dos autores que não possuem obras expostas no museu, utilizando operadores de conjunto.
- Listar código dos autores que possuem obras expostas no museu, utilizando operadores de conjunto.

- Listar o código dos autores que possuem apenas pinturas no museu, utilizando operadores de conjunto.
- Listar o código dos autores que possuem apenas esculturas no museu, utilizando operadores de conjunto.
- Listar o código dos artistas que possuem ou apenas esculturas ou apenas pinturas, utilizando operadores de conjunto.

Você pode baixar os comandos que resolvem as consultas no arquivo [sqlmod3.5](#).

Atividade 5

Vamos agora executar uma atividade prática utilizando o banco de dados da empresa. Seu modelo lógico é o seguinte:



Modelo lógico de banco de dados do minimundo EMPRESA.

Para realizar o exercício, faça conexão no banco da empresa e realize as seguintes consultas utilizando EXCEPT e INTERSECT:

- Listar o ID dos empregados que são vendedores e que trabalham nos departamentos 20 ou 30.
- Listar o ID dos empregados que trabalham nos departamentos 20 ou 30 e que não são vendedores.
- Listar o ID dos empregados que são vendedores e que não trabalham nos departamentos 20 ou 30.
- Listar todos os dados dos departamentos que estão na região 1 e que possuam empregados com o cargo de vendedor.

- Listar todos os dados dos departamentos que estão na região 1 e que não possuam empregados com o cargo de vendedor.
- Listar todos os dados dos departamentos que possuam empregados com o cargo de vendedor e que não estão na região 1.

Chave de resposta

Você pode baixar os comandos que resolvem as consultas no arquivo [exercmod3.5](#) .

Considerações finais

- Operações de junções internas e externas.
- Subconsultas aninhadas e correlatas.
- Operadores de conjunto: UNION, INTERSECT e EXCEPT.

Explore +

Para saber mais sobre os comandos CREATE, ALTER E INSERT, acesse:

- Postgresql em create table; altertable; sql-altertable.
- Postgresql em sql-select, e entenda o comando INSERT.
- Postgresql em sql-uptade, e entenda o comando UPDATE.

Referências

POSTGRESQL. **Manual de referência do PostgreSQL**. Consultado na internet em: 11 ago. 2020.