

Implementação de banco de dados

Aula 10: Linguagem SQL – Outros objetos de banco de dados

Apresentação

Até agora, em nossa disciplina, abordamos a álgebra relacional, criamos tabelas, fizemos consultas e gerenciamos transações. Nesta última aula, vamos ver o que são visões, índices e sequencias, como são criados, utilizados e eliminados.

Bons estudos!

Objetivo

- Identificar como criar, utilizar e eliminar visões, índices e sequencias.

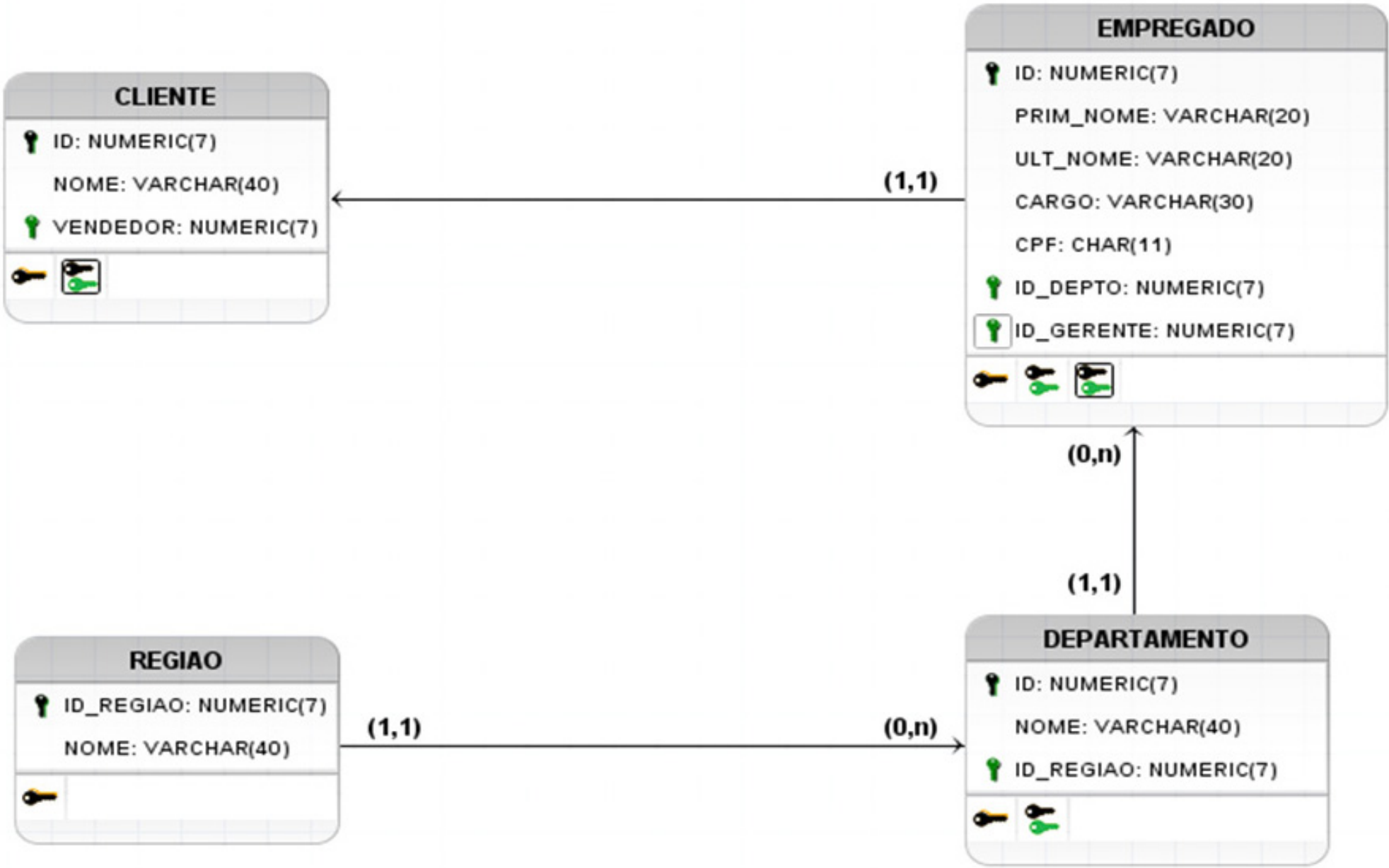


(Fonte: Wright Studio / Shutterstock).

Banco de dados de exemplo

Nesta aula, continuaremos utilizando o banco de dados da empresa para os exemplos.

Modelo lógico:



As tabelas contêm os seguintes dados:

id_regiao	nome
numeric (7)	character varying (40)
1	1 Norte
2	2 Sul

id	nome	id_regiao
numeric (7)	character varying (40)	numeric (7)
1	10 Administrativo	1
2	20 Vendas	1
3	30 Compras	2

id	nome	id_regiao	id_gerente
numeric (7)	character varying (40)	numeric (7)	numeric (7)
1	1 Nelson	1	10
2	2 Leticia	1	20
3	3 Joana	1	30
4	4 Carlos	2	40
5	5 Maria	2	50

id	nome	vendedor
numeric (7)	character varying (40)	numeric (7)
1	110 Ponto Quente	5
2	120 Casa Supimpa	6
3	130 Coisas e Trilhas	5
4	140 Casa Desconto	[null]

Região Departamento Empregado Cliente

Tendo este banco em mente, é extremante recomendável que você execute os comandos de exemplo no PostgreSQL.

Comentário

Foi escolhido como base o PostgreSQL, por ser um SGBD mais leve e fácil de instalar, porém você pode usar o SQLServer ou o Oracle; quando houver diferença entre os SGBDs, você receberá um alerta.

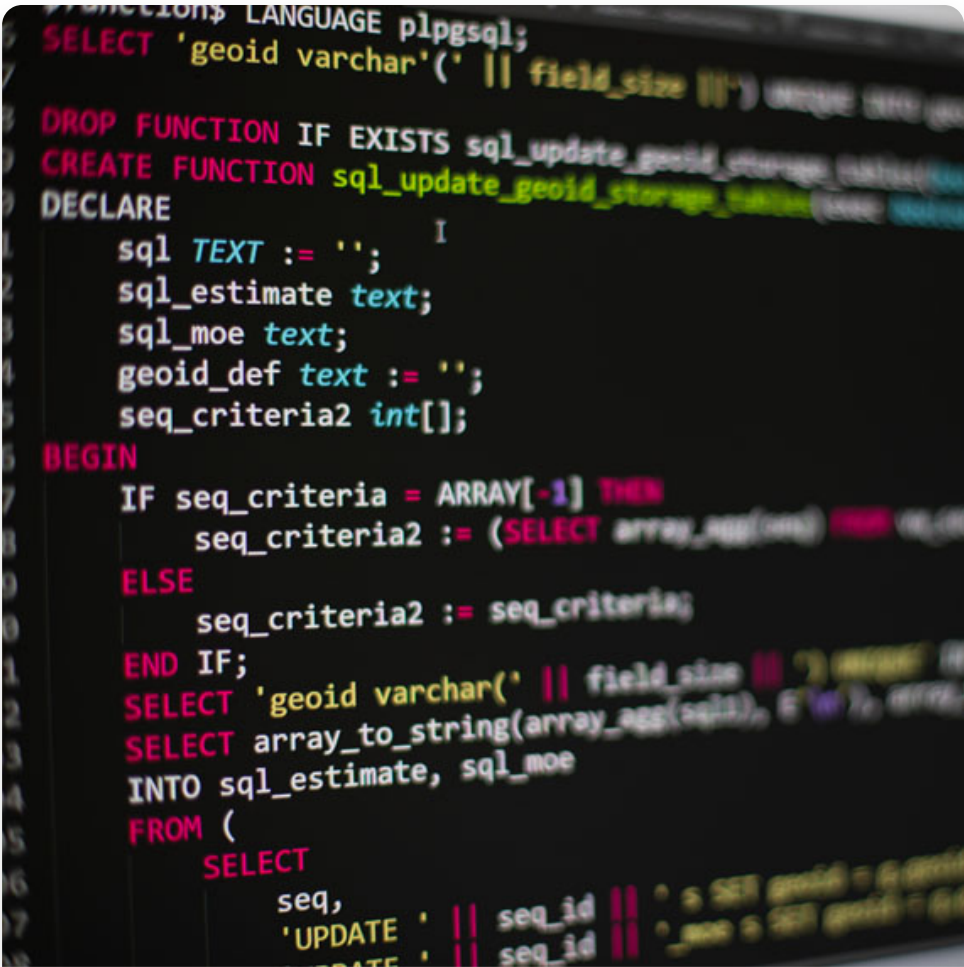
Como sempre, utilizando este banco, é altamente recomendável que você execute os comandos de exemplo no PostgreSQL.

Visões (views)

A utilização de visões nos permite materializar o esquema externo de um banco de dados.

Uma visão (view) é uma consulta previamente definida que fica armazenada no dicionário de dados, podendo ser acessada de forma similar a uma tabela. Quando a visão é referenciada em um comando de select, suas linhas e colunas são determinadas dinamicamente, ou seja, a consulta é executada, e o resultado apresentado para o usuário, atuando como um verdadeira tabela virtual.

Uma visão pode permitir, com restrições, que os dados da tabela sejam manipulados em comandos de insert, update e delete; porém, não armazena estes dados.



 (Fonte: EvalCo / Shutterstock).

Vejamos a sintaxe do comando de criação de visões:

```
CREATE VIEW nome_view
AS subquery
```

Onde:

Cláusula		Descrição
Nome_view		É o nome da view.
subquery		É o comando select que originará a view. 0

Vejamos a sintaxe do comando de criação de visões:

```
CREATE VIEW EMP_RESUMO AS
SELECT ID, ULT_NOME, CARGO
FROM EMPREGADO
```

Sendo que funciona exatamente da mesma forma nos três SGBDs que estamos estudando.

POSTGRESQL

1

2

3

4

5

CREATE VIEW EMP_RESUMO AS

SELECT ID, ULT_NOME, CARGO

FROM EMPREGADO

Data Output

Explain

Messages

Notifications

Q

CREATE VIEW

Query returned successfully in 62 msec.

ORACLE

CREATE VIEW EMP_RESUMO AS

SELECT ID, ULT_NOME, CARGO

FROM EMPREGADO

Script Output x

Task completed in 0.243 seconds

View EMP_RESUMO created.

SQLSERVER

1

2

3

CREATE VIEW EMP_RESUMO AS

SELECT ID, ULT_NOME, CARGO

FROM EMPREGADO

MESSAGES

18:19:25

Started executing query at Line 1

Commands completed successfully.

Total execution time: 00:00:00.028


Comentário

Agora, podemos utilizar EMP_RESUMO como se fosse uma tabela. A diferença é que ela obtém os dados dinamicamente através da query especificada na própria definição da view.

Consultando Visões

Para recuperarmos dados através da visão, basta executarmos um comando de select no qual o nome da visão fica na cláusula from. Por exemplo: para recuperarmos todas as linhas e colunas da visão EMP_RESUMO, basta darmos o seguintes comandos:

```
SELECT *
FROM EMP_RESUMO
```

EMPRESA on postgres@postgres

1

2

3

4

5

SELECT *

FROM EMP_RESUMO

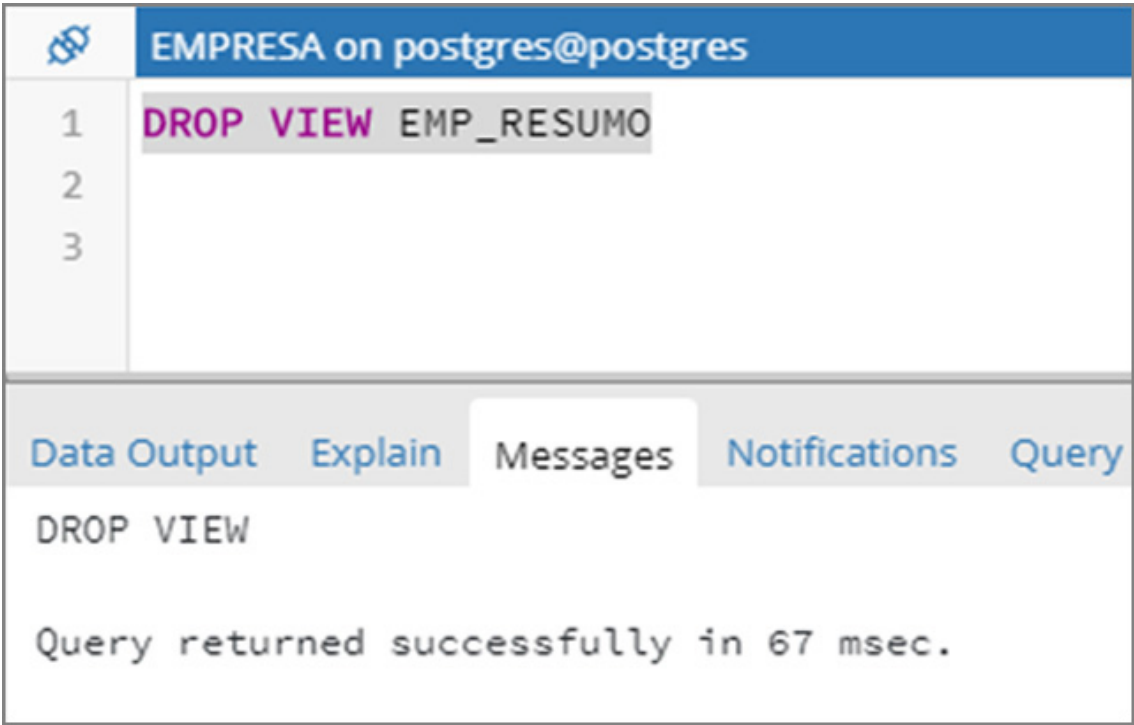
Data Output				Explain	Messages	Notifications	Query History
	id	ult_nome	cargo				
	numeric (7)	character varying (20)	character varying (30)				
1	1	Velasques	Presidente				
2	2	Neves	Diretor de Compras				
3	3	Nogueira	Diretor de Vendas				
4	4	Queiroz	Gerente de Compras				
5	5	Rodrigues	Vendedor				
6	6	Ugarte	Vendedor				

Dica

- Ao fazer uma consulta em uma view, podemos usar todos as cláusulas do select, como: where, group by, having order by;
- Podemos também utilizar os operadores relacionais (in, between, like, is null), os operadores lógicos (and, or not) e as funções de grupo;
- Podemos fazer junções de uma tabela com uma visão;
- Podemos fazer subconsultas e utilizar operadores de conjunto;
- Em resumo, podemos utilizar todos os recursos que podem ser utilizados em comandos a partir de tabelas.

Eliminando Visões

Para eliminar uma visão, basta dar o comando **drop view**.



Tipos de views

Existem basicamente dois tipos de views: simples e complexos. O tipo simples é composto por apenas um select e utiliza apenas uma tabela; suas colunas são formadas por colunas da tabela original, sem cálculos ou funções. Já a view complexa é aquela em que há um join entre tabelas na subquery.

Comentário

- Com uma view simples, será possível executarmos os comandos insert, update e delete (além do select);
- A manipulação dos dados através de uma view não desabilita as constraints das tabelas às quais os mesmos se referem;
- Cada coluna definida para views deve ter um nome de coluna válido;
- Caso seja uma fórmula, deve possuir um alias.

Veja um exemplo:

Crie a visão CLI_RESUMO com as colunas ID e nome da tabela cliente.

1
2
3
4
5

CREATE VIEW CLI_RESUMO
AS SELECT ID,NOME
FROM CLIENTE|

Data Output

Explain

Messages

Notifications

Query

CREATE VIEW

Query returned successfully in 63 msec.

Agora consulte a visão.

1
2
3
4
5

SELECT *
FROM CLI_RESUMO
|

Data Output

Explain

Messages

Notifications

Query

	Id numeric (7)	nome character varying (40)
1	110	Ponto Quente
2	120	Casa Supimpa
3	130	Coisas e Tralhas
4	140	Casa Desconto

Note que a nossa visão é simples, então podemos fazer comandos de insert, update e delete através dela. Podemos então inserir um cliente. Veja o comando:

```
INSERT INTO CLI_RESUMO VALUES (200,'PINGO DE LEITE')
```

1
2
3
4

INSERT INTO CLI_RESUMO VALUES (200,'PINGO DE LEITE')
|

Data Output

Explain

Messages

Notifications

Query History

INSERT 0 1

Query returned successfully in 58 msec.

Se consultarmos a visão, o novo cliente vai aparecer.

1	SELECT *
2	FROM CLI_RESUMO
3	
4	
5	

Data Output	Explain	Messages	Notifications	Q
	id numeric (7)	nome character varying (40)		
1	110	Ponto Quente		
2	120	Casa Supimpa		
3	130	Coisas e Tralhas		
4	140	Casa Desconto		
5	200	PINGO DE LEITE		

E, claro, se consultarmos diretamente a tabela, também aparecerá.

1	SELECT *
2	FROM CLIENTE
3	
4	
5	

Data Output	Explain	Messages	Notifications	Query Histo
	id numeric (7)	nome character varying (40)	vendedor numeric (7)	
1	110	Ponto Quente	5	
2	120	Casa Supimpa	6	
3	130	Coisas e Tralhas	5	
4	140	Casa Desconto	[null]	
5	200	PINGO DE LEITE	[null]	

Note que a coluna Vendedor do novo cliente é nula; isto ocorre devido ao fato de na visão esta coluna não existir.

Um cuidado que você tem de tomar é que, se for inserir através de uma visão, todas as colunas obrigatórias têm que existir na visão, senão ocorrerá um erro, pois a visão não desabilita as restrições da tabela.



 (Fonte: Waruj Wongsangiem / Shutterstock).

Indexando tabelas

Um índice tem o propósito de acelerar o acesso aos dados de tabelas muito extensas, ou que são frequentemente acessadas via join.

No Oracle, ao criarmos uma primary key ou unique key, automaticamente é definido um índice único de acesso àquelas chaves. Porém, outras colunas que acessamos constantemente poderão ser indexadas, e para isso, temos que conhecer os comandos de criação e eliminação de índices.

Tipos de índices

1

Único

Garante a unicidade do valor. O índice criado na primary key ou unique key é único; porém, podemos criar restrições de unicidade em outras colunas da tabela;

2

Não único

Índices criados apenas com o propósito de acelerar a pesquisa, como em chaves estrangeiras (foreign key), nas quais a unicidade não é requerida;

3

Uma coluna

Apenas uma coluna será indexada;

4

Colunas compostas ou concatenadas

Até 255 colunas podem ser concatenadas para formar apenas um índice, e não necessitam ser adjacentes.

Utilizando índices

Devemos utilizar índices sempre que:

O meio de acesso tradicional mostrar-se ineficiente, provavelmente pelo tamanho da tabela e um número alto de consultas feitas a ela;

A coluna tiver limites extensos de valores;

A coluna tiver muitos valores nulos;

Duas ou mais colunas são frequentemente acessadas em conjunto numa cláusula where ou condição;

A tabela for muito grande e, na maior parte das queries, for esperada a recuperação de até 4% das linhas.

Não devemos utilizar índices sempre que:

A tabela for pequena (pode ser armazenada em poucos blocos Oracle);

As colunas não são frequentemente usadas em condições;

A maior parte das queries recupera mais que 4% das linhas da tabela;

A tabela sofre alta taxa de atualização.

Cuidados com índices:

É importante observar que os índices são objetos atualizados pelo SGBD, em todas as operações de insert, update e delete na tabela indexada; Portanto, se por um lado aceleram a pesquisa aos dados através do comando select, por outro lado, quanto mais índice tiver a tabela, maior é o seu tempo de atualização.

Criando um índice

Vejamos a sintaxe do comando a seguir:

```
CREATE INDEX [schema.]nome_indice ON tabela (coluna1 [, coluna2 [,...]] )
```

Onde:


schema	É o schema no qual será gerado o índice. O default é o schema do usuário que está criando o índice.
nome_indice	Nome dado ao objeto índice que será criado.
tabela	Nome da tabela que será indexada.
coluna n	A(s) coluna(s) que irão compor o índice.

Veja um exemplo:

Para criar um índice na coluna Vendedor da tabela Cliente, o comando seria:

```
CREATE INDEX IND_VEND ON CLIENTE(VENDEDOR)
```

POSTGRESQL

EMPRESA on postgres@postgres

1

2

3

CREATE INDEX IND_VEND ON CLIENTE(VENDEDOR)

Data Output

Explain

Messages

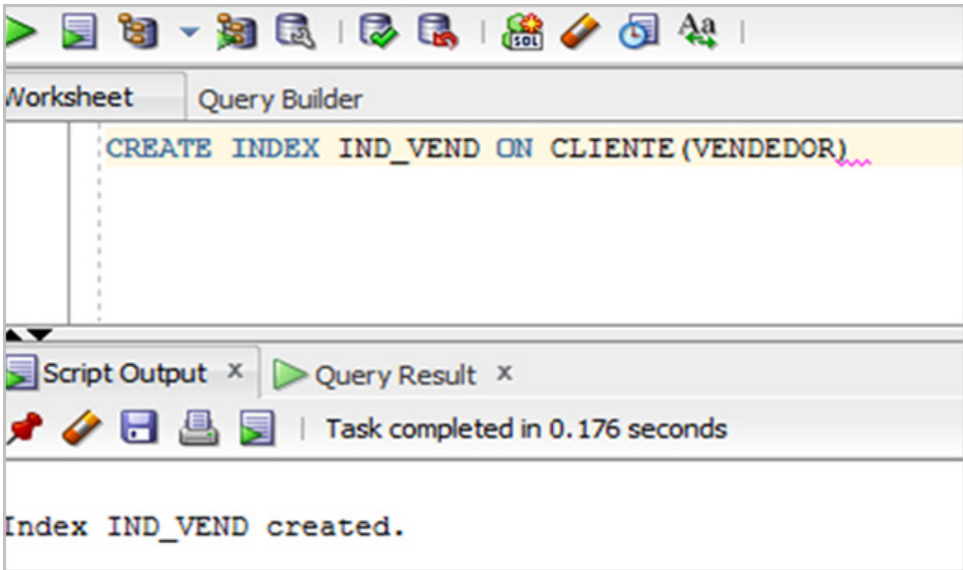
Notifications

Query History

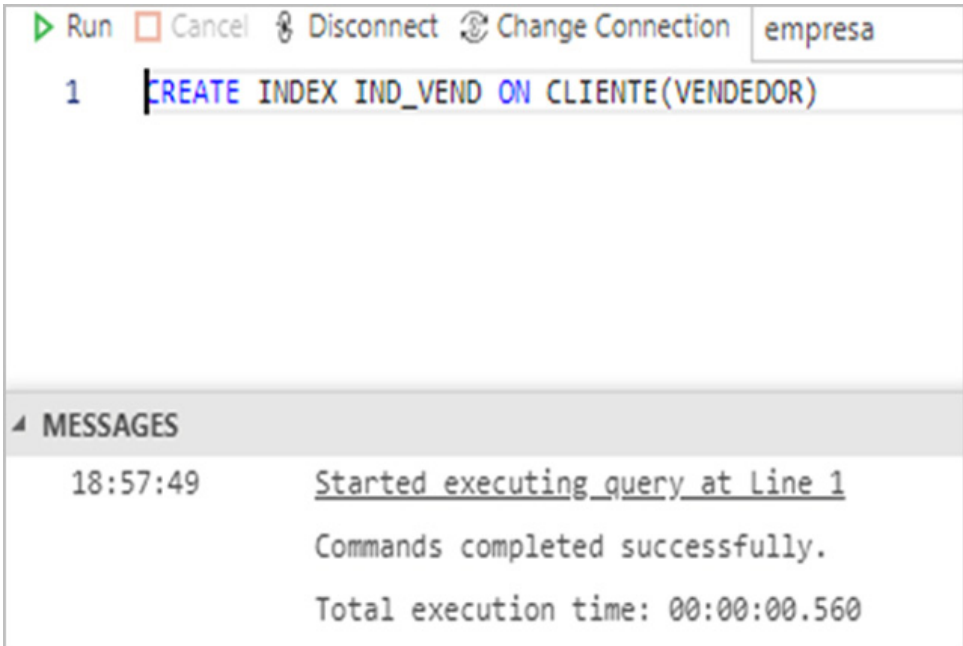
CREATE INDEX

Query returned successfully in 98 msec.

ORACLE



SQLSERVER



Eliminando um índice

Os índices podem ser eliminados, mas nunca alterados. Para alterá-los, devemos efetuar a remoção e depois recriá-los. O comando de eliminação de um índice é simples e exige apenas o privilégio drop index.

Vejam os a sintaxe do comando a seguir:

DROP INDEX [schema.]nome_índice

Onde:

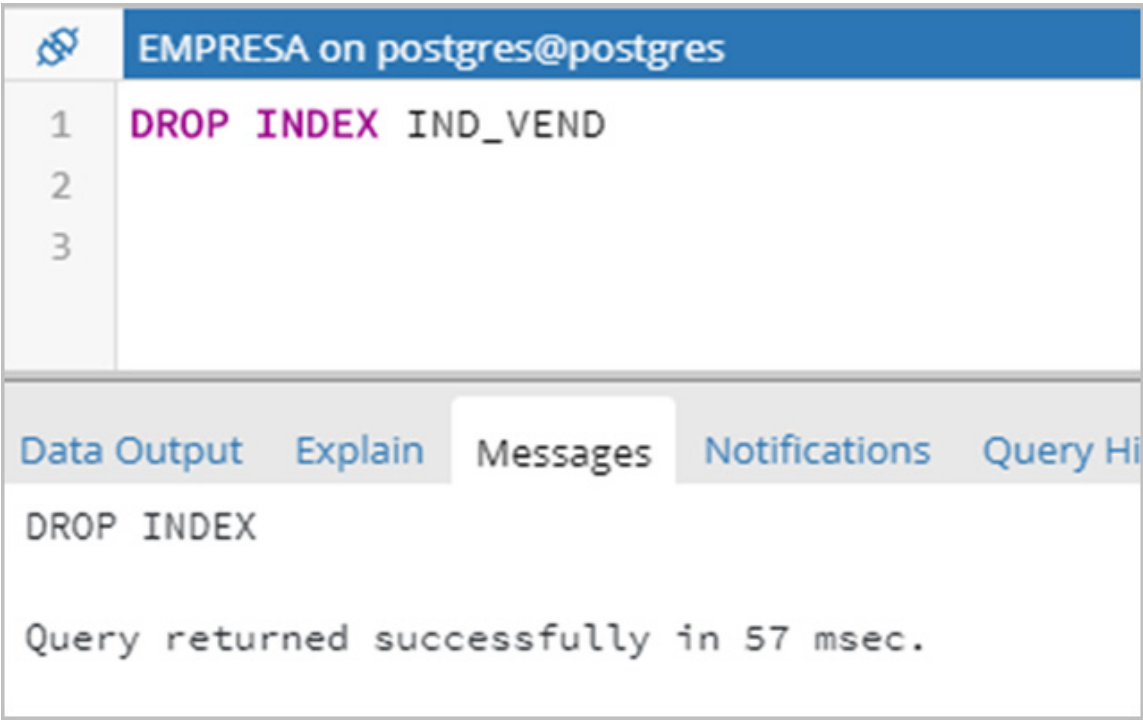
Cláusula	Descrição
schema	É o schema a quem pertence o índice. O default é o schema do usuário que está removendo o índice.
nome_índice	Nome do índice que será removido.

Atenção

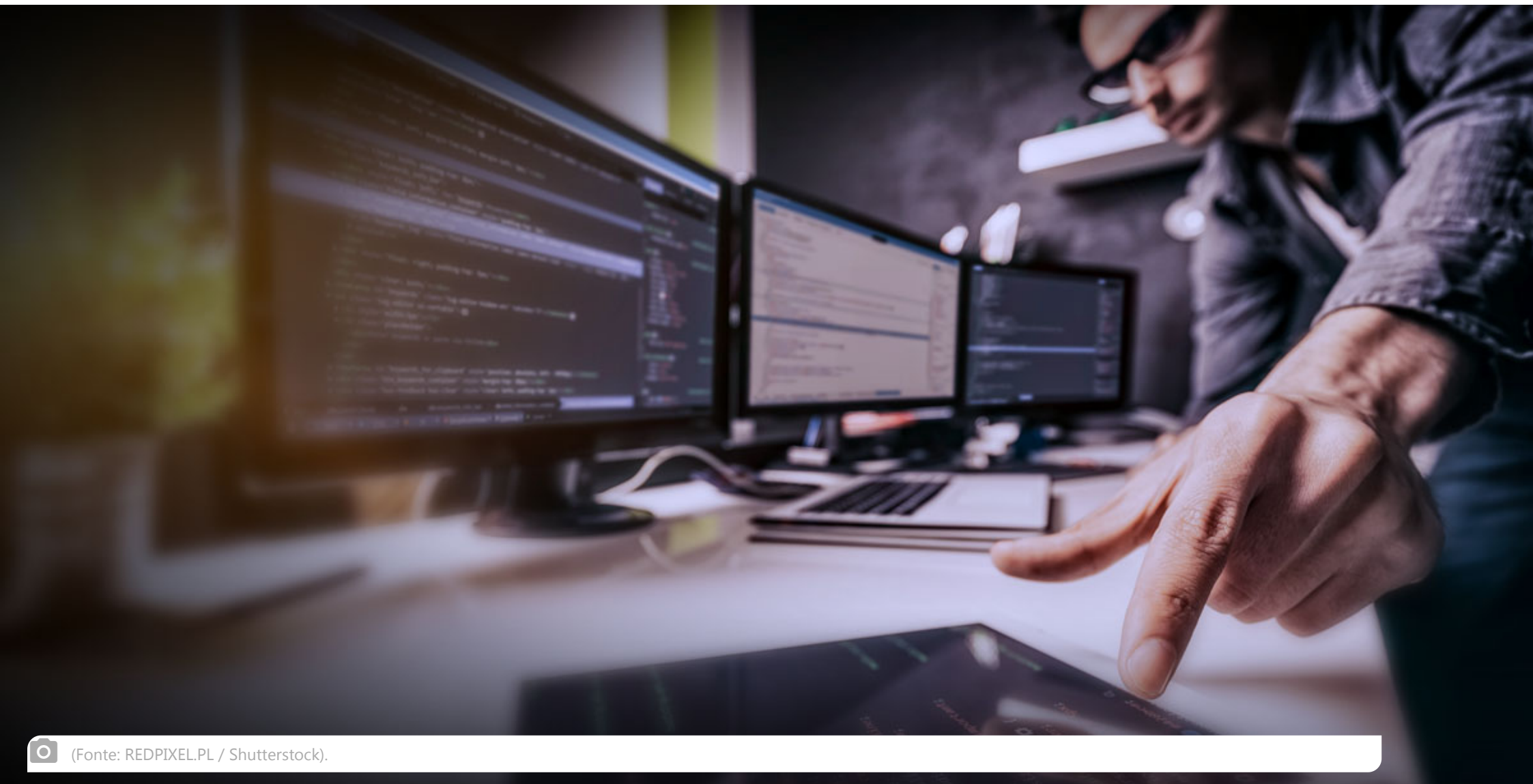
Não podemos eliminar os índices criados automaticamente pelas constraints primary key e unique key. Estes são automaticamente removidos quando eliminamos ou desabilitamos as constraints.

Por exemplo: para eliminar o índice criado na coluna Vendedor, o comando seria:

```
DROP INDEX IND_VEND
```



Um gerador de sequências (sequences)



Os SGBDs possuem internamente uma máquina geradora de números sequenciais que pode perfeitamente ser usado para produzir números únicos, consecutivos e incrementados conforme determinado.

A esses números chamamos de sequences, que podem ser utilizados para gerar valores para chaves primárias em tabelas nas quais não existe uma coluna mais apropriada, ou qualquer outra aplicação em que haja a necessidade de números únicos.

A sintaxe do comando de criação de uma sequence é:

```
CREATE SEQUENCE    sequence_name
    [INCREMENT BY n ]
    [START WITH n]
    [MAXVALUE n I NOMAXVALUE]
    [MINVALUE n I NOMINVALUE]
    [CYCLE I NOCYCLE]
    [CACHE 1 NOCACHE]
```

Onde:

Cláusula	Descrição
INCREMENT BY	Especifica o intervalo entre os números sequenciais. Pode ser positivo ou negativo, e a omissão significa incremento positivo de 1 em 1.
START WITH	É o primeiro número da sequência.
MAXVALUE	Especifica o valor máximo que a sequence pode alcançar.
NOMAXVALUE	É o default. Especifica um valor com precisão de 27 dígitos positivos ou negativos.
MINVALUE	Especifica um valor mínimo para a sequence.
NOMINVALUE	É o default. Especifica o valor mínimo de 1 para uma sequence ascendente e precisão negativa de 26 dígitos para sequences descendentes.
CYCLE	Especifica que, após atingido o valor limite, a sequence recomeçará no minvalue ou maxvalue.
NOCYCLE	É o default. Especifica que a sequence não pode gerar mais números após atingir o limite.
CACHE	Especifica quantos números a sequence pré-aloca e mantém em memória para acesso mais rápido.
NOCACHE	Valores não são pré-allocados em memória.

Vamos criar uma sequence chamada **Números**, que terá um valor inicial de 40, será incrementada de 3 em 3 e um valor máximo de 60.

O comando seria:

```
CREATE SEQUENCE NUMEROS
    INCREMENT BY 3
    START WITH 40
    MAXVALUE 60;
```

POSTGRESQL

1

CREATE SEQUENCE NUMEROS

2

INCREMENT BY 3

3

START WITH 40

4

MAXVALUE 60

5

Data Output

Explain

Messages

Notifications

Query

CREATE SEQUENCE

Query returned successfully in 66 msec.

ORACLE

CREATE SEQUENCE NUMEROS

INCREMENT BY 3

START WITH 40

MAXVALUE 60;

Script Output x

Task completed in 0.173 seconds

Sequence NUMEROS created.

SQLSERVER

Run

Cancel

Disconnect

Change Connection

empresa

1

CREATE SEQUENCE NUMEROS

2

INCREMENT BY 3

3

START WITH 40

4

MAXVALUE 60

MESSAGES

19:24:05

Started executing query at Line 1

Obtendo valores da sequence

Cada um dos SGBDs que estudamos recupera os valores da sequence de forma diferente. Vejamos cada um.

POSTGRESQL

Este SGBD utiliza uma função chamada Nextvall () , cujo retorno é o valor recuperado da sequence.

Veja primeiro no comando de Select:

EMPRESA on postgres@postgres	
1	SELECT NEXTVAL('numeros')
2	
3	
4	
Data Output Explain Messages Notificatio	
	nextval bigint
1	40

Note que o nome da sequence está entre apóstrofes e que o valor retornado é 40, que é o valor inicial que definimos. Agora vamos utilizar em um insert na tabela departamento.

1	INSERT INTO DEPARTAMENTO
2	VALUES (NEXTVAL('numeros'), 'ESPECIAL',2)
3	
4	
5	
Data Output Explain Messages Notifications Query History	
INSERT 0 1	
Query returned successfully in 253 msec.	

Consultando a tabela, obtemos:

1 SELECT *

2 FROM DEPARTAMENTO

3

4

5

Data Output

Explain

Messages

Notifications

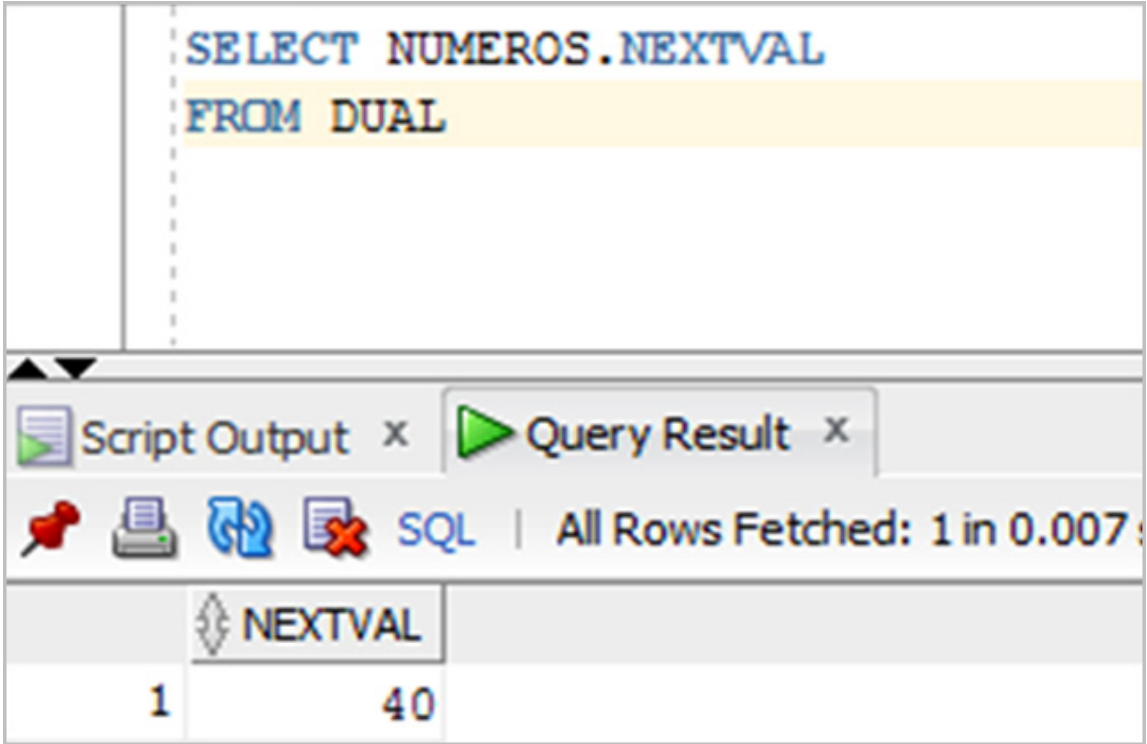
Query History

	id numeric (7)	nome character varying (40)	id_regiao numeric (7)
1	10	Administrativo	1
2	20	Vendas	1
3	30	Compras	2
4	40	TT	1
5	43	ESPECIAL	2

Observe que foi inserido o valor 43, já que a sequence incrementa de 3 em 3.

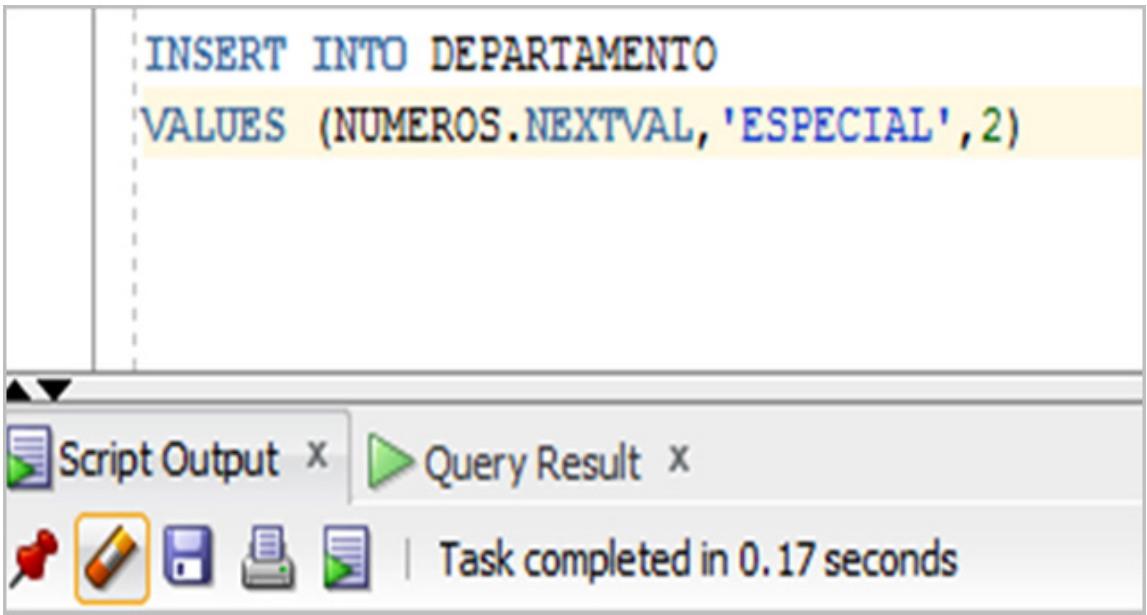
ORACLE

Existe uma pseudocoluna chamada Nextval, que é utilizada para extrair valores de uma sequence qualquer, sempre que for referenciada.

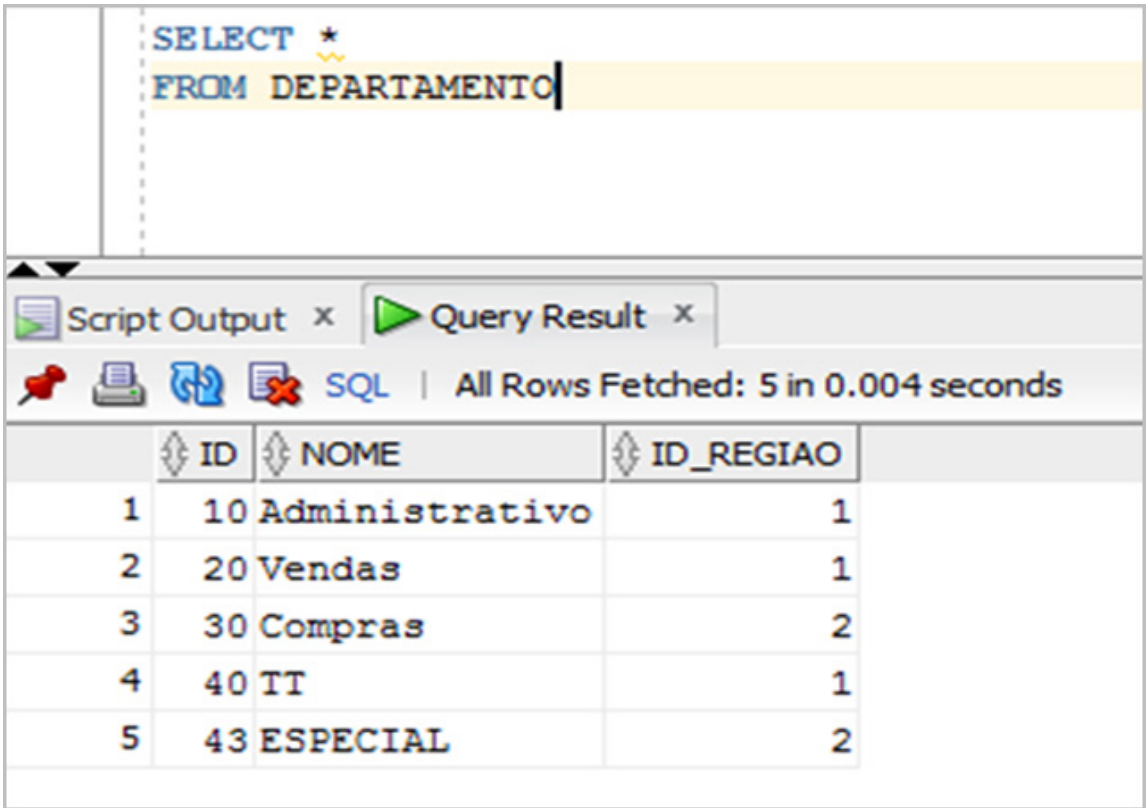


Nota: lembre que o Oracle não aceita select sem from; por isso o uso da tabela dual.

Agora o insert:



E o select:



SQLSERVER

No SQL Server, utilizamos um expressão para recuperar o valor da sequence, que é **NEXT VALUE FOR**.

Veja um exemplo de select:

RunCancelDisconnectChange Connection

empresa

1SELECT NEXT VALUE FOR numeros

RESULTS

	(No column name)
1	40

Agora o insert:

RunCancelDisconnectChange Connection

empresa

1INSERT INTO DEPARTAMENTO

2VALUES (NEXT VALUE FOR numeros, 'ESPECIAL',2)

MESSAGES

19:50:38Started executing query at Line 1

(1 row affected)

Total execution time: 00:00:00.082

E confirmando a inserção:

RunCancelDisconnectChange Connection

empresa

1SELECT *

2FROM DEPARTAMENTO

RESULTS

	ID	NOME	ID_REGIAO
1	10	Administrativo	1
2	20	Vendas	1
3	30	Compras	2
4	40	TT	1
5	43	ESPECIAL	2

MESSAGES

19:51:37Started executing query at Line 1

(5 rows affected)

Total execution time: 00:00:00.004

Particularidades de sequences:

Quando um número sequencial é gerado pela sequence, este é incrementado independentemente de a transação ter sido efetivada (commit) ou não (rollback);

Logo, valores podem ser perdidos para o sistema, ou seja, existirá sempre a garantia de unicidade (caso seja nocycle), porém poderá haver intervalos na aplicação;

Temos que ter cuidados com aplicações que manipulam dados fiscais como emissão de notas fiscais. Não devemos numerá-las com sequences;

Da mesma forma que usuários acessam a mesma sequence, estes podem receber valores intercalados, visto que num dado momento a sequence poderá estar atendendo um número para cada um;

Dois usuários jamais conseguirão gerar o mesmo número sobre uma mesma sequence;

Os valores que estão no cache (memória) são perdidos caso o banco de dados sofra uma interrupção anormal.

Alterando uma sequence

Sequences podem ser submetidas a alteração e eliminação tal qual uma tabela, através dos comandos alter sequence e drop sequence.

Veja a sintaxe do comando:

```
ALTER SEQUENCE [schema.]sequence_name  
    [INCREMENT BY n ]  
    [MAXVALUE n I NOMAXVALUE]  
    [MINVALUE n I NOMINVALUE]
```

Comentário

Controles sobre sequences alteradas:

- Serão afetados apenas os números a serem gerados;
- Algumas validações serão feitas como um novo valor maxvalue; não poderá ser definido se menor que o último número gerado;
- A cláusula start with não poderá ser alterada; neste caso, a sequence deverá ser eliminada e recriada.

Eliminando uma sequence

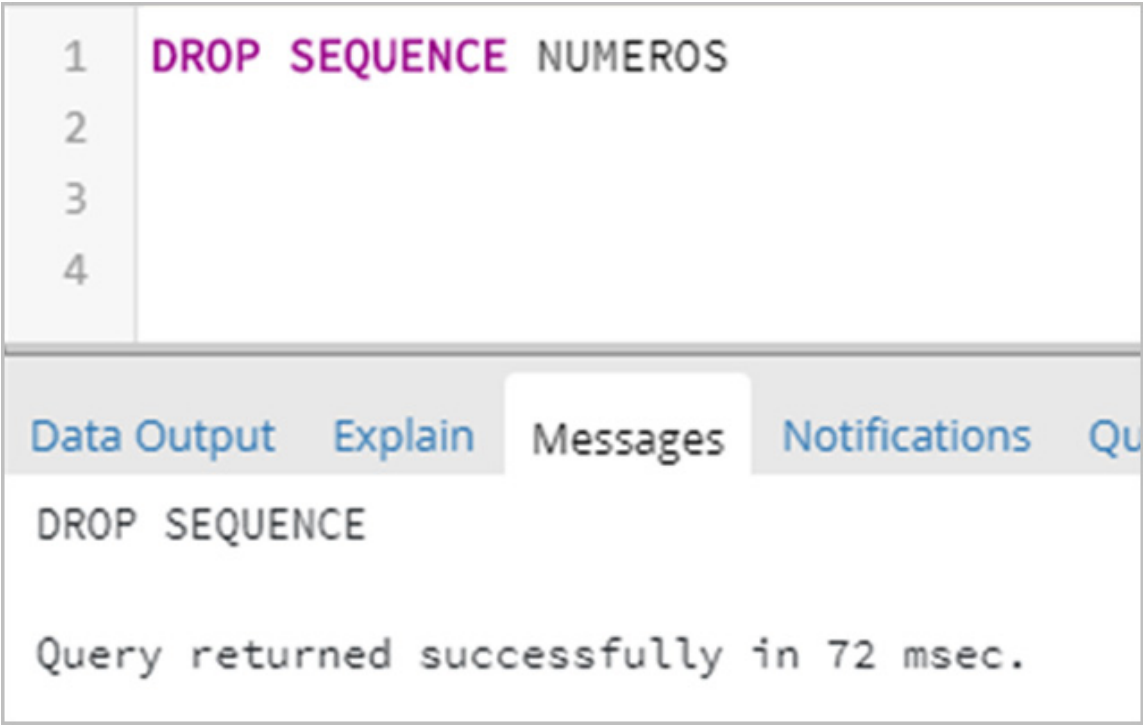
Para apagarmos uma sequence, utilizamos o comando **drop sequence**.

A sintaxe para eliminação de uma sequence:

```
DROP SEQUENCE [schema.]sequence_name;
```

Exemplo:

Drop sequence números.



Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Colunas autonumeradas

Conforme já vimos, as sequences podem ser utilizadas para gerar valores para chaves primárias autonumeradas. No Oracle, este é o único, já que este SGBD não possui o tipo de autonumeração. A seguir, vamos verificá-los no SQLServer e PostgreSQL.

POSTGRESQL:

O tipo autonumerado do PostgreSQL é o serial que deve ser utilizado na PK da tabela.

Veja o comando de criação de tabela:

12345

```
CREATE TABLE PK_AUTONUMERADA
(C1 SERIAL PRIMARY KEY,
C2 VARCHAR(10))
```

Data Output

Explain

Messages

Notifications

Query History

CREATE TABLE

Query returned successfully in 130 msec.

Nossa tabela possui duas colunas: a primeira autonumerada e a segunda caracter.

Veja a inserção de duas linhas:

12345

```
INSERT INTO PK_AUTONUMERADA (C2 ) VALUES ('LINHA1');
INSERT INTO PK_AUTONUMERADA (C2) VALUES ('LINHA2');
```

Data Output

Explain

Messages

Notifications

Query History

INSERT 0 1

Query returned successfully in 79 msec.

Note que na lista de colunas só foi colocada a coluna C2. A C1 terá seu valor preenchido automaticamente, conforme podemos ver na seguinte consulta:

1234

```
SELECT *
FROM PK_AUTONUMERADA
```

Data Output

Explain

Messages

Notifications

	c1 integer	c2 character varying (10)
1	1	LINHA1
2	2	LINHA2

SQLSERVER:

Neste SGBD, a autonumeração é obtida a partir de uma propriedade chamada identity, que você adiciona ao tipo da coluna.

A sintaxe é:

```
<nome da coluna> <tipo numérico> IDENTITY (X,Y)
```

Onde X define o valor inicial, e Y o valor do incremento.

Veja o comando de criação da tabela:

1
2
3

CREATE TABLE PK_AUTONUMERADA
(C1 INTEGER IDENTITY (1,1) PRIMARY KEY,
C2 VARCHAR(10))

4 MESSAGES

20:10:52

Started executing query at Line 1
Commands completed successfully.
Total execution time: 00:00:00.043

Agora vamos inserir as duas linhas.

Run Cancel Disconnect Change Connection empresa

1
2
3

INSERT INTO PK_AUTONUMERADA (C2) VALUES ('LINHA1');

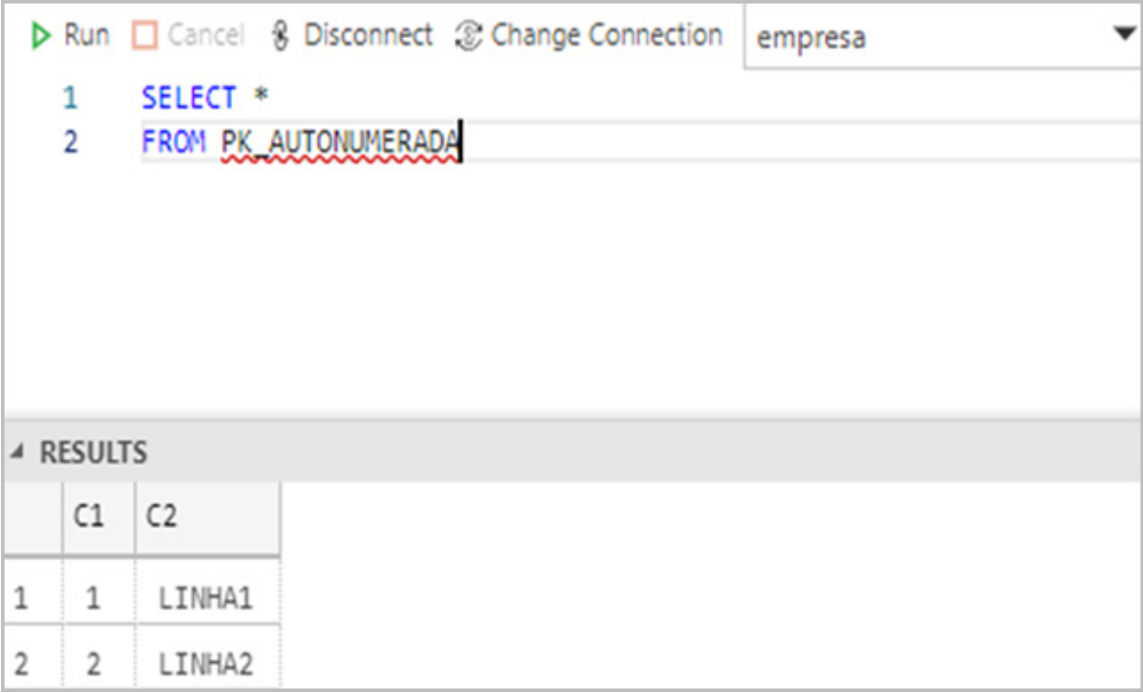
INSERT INTO PK_AUTONUMERADA (C2) VALUES ('LINHA2');

4 MESSAGES

20:11:40

Started executing query at Line 1
(1 row affected)
(1 row affected)
Total execution time: 00:00:00.043

Confirmando a inclusão:



Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Atividade

- 1. Qual são as constraints que criam automaticamente índices?
- 2. Se um sequence foi criado com valor inicial de 30 e incremento de 5, qual será o terceiro valor obtido dela?
- 3. Qual o tipo de dados do PostgreSQL que torna desnecessário o uso de uma sequence para gerar PK?
- 4. Qual o objeto de banco de dados que nos permite implementar o esquema externo da arquitetura ANSI/SPARC?

Notas

Título modal ¹

Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos. Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos. Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos.

Título modal ¹

Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos. Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos. Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos.

Referências

DATE, C. J. **ntrodução a sistemas de banco de dados**l. 7. ed. Rio de Janeiro: Campus, 2000.

ELMASRI, R.; NAVATHE, S. B. **Sistemas de banco de dados**l. 7. ed. São Paulo: Pearson Addison Wesley, 2015.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistemas de banco de dados**l. 5. ed. Rio de Janeiro: Campus, 2006.

Explore mais

Leia os seguintes artigos:

- [Utilizando Sequences no Microsoft SQL Server 2012 <http://www.linhadecodigo.com.br/artigo/3378/utilizando-sequences-no-microsoft-sql-server-2012.aspx >](http://www.linhadecodigo.com.br/artigo/3378/utilizando-sequences-no-microsoft-sql-server-2012.aspx)
- [Sequence no Oracle - Criando autoincremento <https://www.devmedia.com.br/sequence-no-oracle-criando-auto-incremento/402 >](https://www.devmedia.com.br/sequence-no-oracle-criando-auto-incremento/402)
- [Create Sequence \(Transact-SQL\) <https://docs.microsoft.com/pt-br/sql/t-sql/statements/create-sequence-transact-sql?view=sql-server-2017 >](https://docs.microsoft.com/pt-br/sql/t-sql/statements/create-sequence-transact-sql?view=sql-server-2017)
- [Trabalhando com campos autoincremento \(Identity\) no SQL Server <https://www.devmedia.com.br/trabalhando-com-campos-auto-incremento-identity-no-sql-server/17974 >](https://www.devmedia.com.br/trabalhando-com-campos-auto-incremento-identity-no-sql-server/17974)