

Banco de Dados 2

06 – Rule e Exercícios de Fixação

Sistema de Rules no PostgreSQL

- No PostgreSQL, o **sistema de RULES** (regras) é um mecanismo que **permite interceptar e reescrever comandos SQL** **antes** de eles serem realmente executados pelo executor de consultas.
- Ele faz parte do **Query Rewriter** do PostgreSQL, que é uma **etapa logo após o parser** e **antes do otimizador** no **ciclo de execução de uma query**.
- Em ciência da computação e linguística, a **análise sintática** (do inglês: **parsing**) é um processo de um **compilador** (de uma linguagem de programação), é a **segunda fase da compilação** onde se analisa uma sequência que foi dada entrada (via um arquivo de computador ou via teclado, por exemplo) para **verificar sua estrutura gramatical segundo uma determinada gramática formal**. Este processo trabalha em conjunto com a **análise lexical** (primeira etapa, onde se verifica de acordo com determinado **alfabeto**) e **análise semântica** (terceira etapa, onde verificam-se os erros semânticos).

Sistema de Rules no PostgreSQL

- **Cuidado:**
- Em muitos casos, tarefas que poderiam ser executadas por regras em **INSERT/ UPDATE/ DELETE** são melhor executadas com gatilhos (**Triggers**).
- **Gatilhos** são um pouco mais complicados em termos de notação, mas sua semântica é muito mais simples de entender.
- **Regras** tendem a ter resultados surpreendentes quando a **consulta original** contém **funções voláteis**: funções voláteis podem ser executadas mais vezes do que o esperado no processo de execução das regras.

Sistema de Rules no PostgreSQL

```
CREATE [ OR REPLACE ] RULE name AS ON event
    TO table [ WHERE condition ]
    DO [ ALSO | INSTEAD ] { NOTHING | command | ( command ; command ... ) }
```

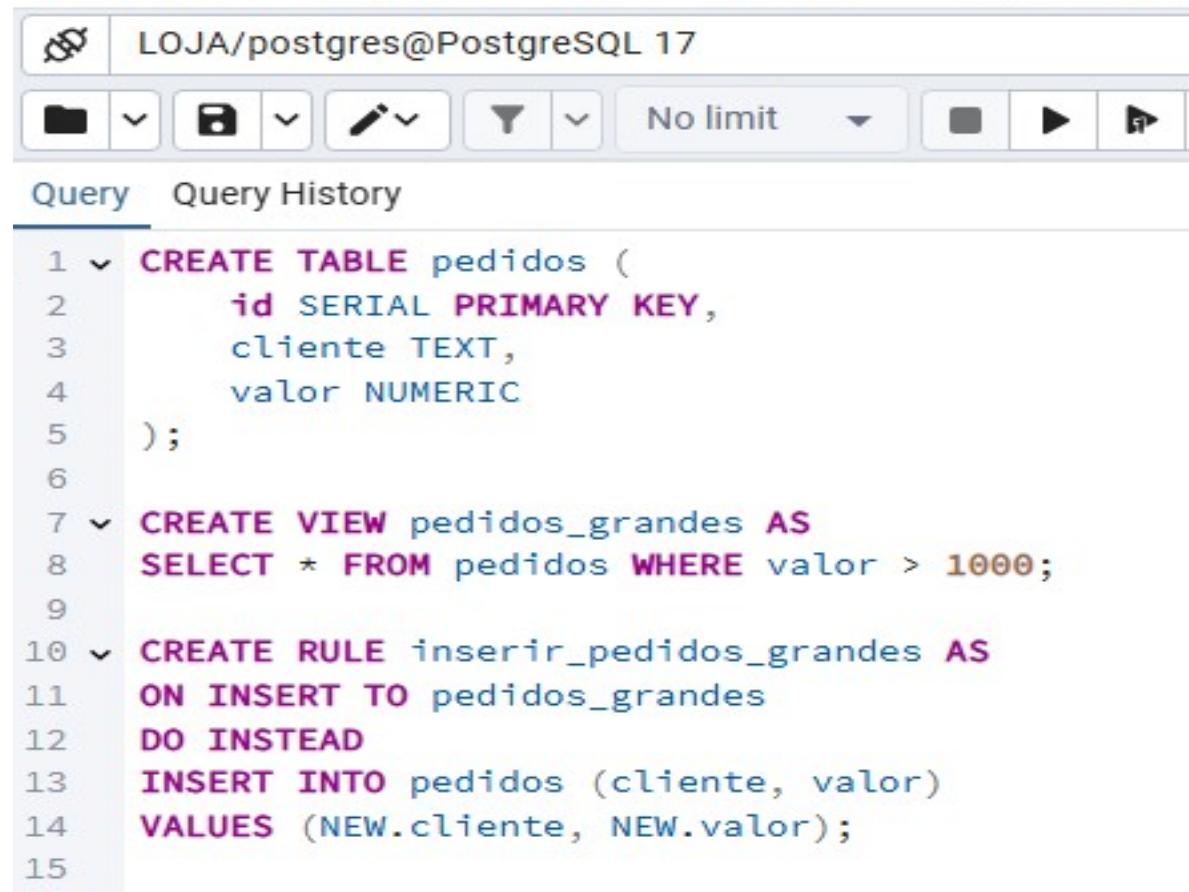
- Você define uma **regra** associada a uma tabela ou view.
- Quando uma query (por exemplo **INSERT**, **UPDATE**, **DELETE** ou até **SELECT**) é enviada para essa tabela/view, o **PostgreSQL reescreve essa query de acordo com a regra definida**.
- O comando original pode ser substituído, suplementado ou bloqueado.

Sistema de Rules no PostgreSQL

```
CREATE [ OR REPLACE ] RULE name AS ON event
    TO table [ WHERE condition ]
    DO [ ALSO | INSTEAD ] { NOTHING | command | ( command ; command ... ) }
```

- **ON** — tipo de operação interceptada.
- **TO** — tabela ou view alvo.
- **WHERE** — condição opcional para disparar a regra.
- **ALSO** — executa o comando original e o definido na regra.
- **INSTEAD** — substitui completamente o comando original pela ação da regra.
- **NOTHING** — ignora a operação (pode ser útil para views somente leitura).

Exemplo de RULE



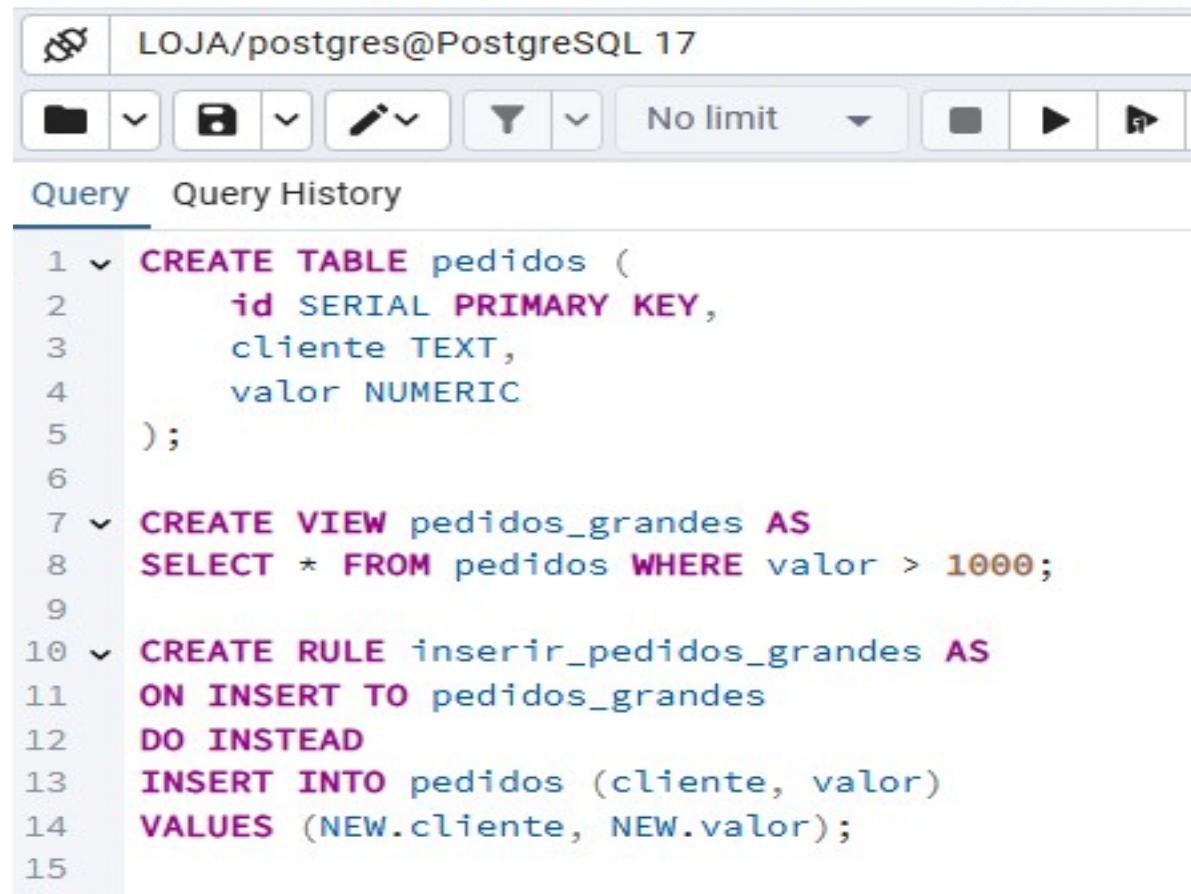
The screenshot shows a PostgreSQL client window with the following details:

- Connection: LOJA/postgres@PostgreSQL 17
- Toolbar buttons: magnifying glass, folder, clipboard, pencil, filter, play/pause, stop.
- Query History tab is selected.
- SQL code (numbered 1 to 15):

```
1 CREATE TABLE pedidos (
2     id SERIAL PRIMARY KEY,
3     cliente TEXT,
4     valor NUMERIC
5 );
6
7 CREATE VIEW pedidos_grandes AS
8 SELECT * FROM pedidos WHERE valor > 1000;
9
10 CREATE RULE inserir_pedidos_grandes AS
11 ON INSERT TO pedidos_grandes
12 DO INSTEAD
13     INSERT INTO pedidos (cliente, valor)
14     VALUES (NEW.cliente, NEW.valor);
15
```

- Criamos uma tabela de PEDIDOS.
- Também criamos uma VIEW (Visão) associada a esta tabela que recebe o nome de PEDIDOS_GRANDES.
- Estudaremos mais detalhadamente sobre VIEWS em um outro momento.

Exemplo de RULE



The screenshot shows a PostgreSQL client window with the following details:

- Connection: LOJA/postgres@PostgreSQL 17
- Toolbar: Includes icons for file, copy, paste, search, and execution.
- Query History tab: Active.
- SQL Editor:

```
1 CREATE TABLE pedidos (
2     id SERIAL PRIMARY KEY,
3     cliente TEXT,
4     valor NUMERIC
5 );
6
7 CREATE VIEW pedidos_grandes AS
8 SELECT * FROM pedidos WHERE valor > 1000;
9
10 CREATE RULE inserir_pedidos_grandes AS
11 ON INSERT TO pedidos_grandes
12 DO INSTEAD
13     INSERT INTO pedidos (cliente, valor)
14     VALUES (NEW.cliente, NEW.valor);
15
```

- Por enquanto basta saber que em **bancos de dados**, uma **view** (ou **visão**) é **uma tabela virtual**, ou seja, um **objeto que não armazena dados fisicamente, mas sim uma consulta armazenada em relação a uma ou mais tabelas base**.

Exemplo de RULE

- Essa consulta define como os dados devem ser apresentados ao usuário, permitindo que ele interaja com a view como se fosse uma tabela real.
- Nesse caso, a **VIEW pedidos_grandes** permite ao usuário visualizar todas as colunas da tabela PEDIDO (“**SELECT *** **FROM pedidos**”) mas somente para registros onde VALOR é maior que 1000 (“**WHERE valor > 1000;**”).

```
6  
7 ✓ CREATE VIEW pedidos_grandes AS  
8   SELECT * FROM pedidos WHERE valor > 1000;  
9
```

Exemplo de RULE

```
15
16 ✓ INSERT INTO pedidos (cliente, valor)
17   VALUES ('Maria', 1500);
18
19 ✓ INSERT INTO pedidos (cliente, valor)
20   VALUES ('José', 400);
21
22 ✓ INSERT INTO pedidos_grandes (cliente, valor)
23   VALUES ('Cassandra', 37680);
```

- Inserimos três novos registros: 2 na tabela PEDIDOS e 1 na view PEDIDOS_GRANDES.
- 2 desses registros podem ser classificados como grandes: Maria e Cassandra.
- 1 registro NÃO é grande: José.

Exemplo de RULE

```
15  
16 ✓ INSERT INTO pedidos (cliente, valor)  
17   VALUES ('Maria', 1500);  
18  
19 ✓ INSERT INTO pedidos (cliente, valor)  
20   VALUES ('José', 400);  
21  
22 ✓ INSERT INTO pedidos_grandes (cliente, valor)  
23   VALUES ('Cassandra', 37680);  
24  
25 SELECT * FROM PEDIDOS;  
26
```

Data Output Messages Notifications

The screenshot shows a database interface with a SQL editor and a results viewer. The SQL editor contains the provided code. The results viewer shows a table with three rows of data:

	id [PK] integer	cliente text	valor numeric
1	4	Maria	1500
2	5	José	400
3	6	Cassandra	37680

- Repare que todos os três registros foram inseridos na tabela PEDIDOS.
- Isso apesar de o registro de Cassandra ter sido inserido na VIEW PEDIDOS_GRANDES em vez de na TABELA PEDIDOS.
- Esse é um resultado esperado e até desejável. Afinal, PEDIDOS_GRANDES é uma visão associada à TABELA PEDIDOS.

Exemplo de RULE

```
15  
16 ✓ INSERT INTO pedidos (cliente, valor)  
VALUES ('Maria', 1500);  
17  
18 ✓ INSERT INTO pedidos (cliente, valor)  
VALUES ('José', 400);  
19  
20 ✓ INSERT INTO pedidos_grandes (cliente, valor)  
VALUES ('Cassandra', 37680);  
21  
22  
23  
24  
25 SELECT * FROM PEDIDOS;  
26  
27 SELECT * FROM PEDIDOS_GRANDES;  
28
```

Data Output Messages Notifications

The screenshot shows a database interface with a command-line area and a data output area. The command-line area contains the provided SQL code. The data output area shows two tables: 'pedidos' and 'pedidos_grandes'. The 'pedidos' table has two rows: Maria (id 4, valor 1500) and Cassandra (id 6, valor 37680). The 'pedidos_grandes' table has one row: Cassandra (id 6, valor 37680). Below the tables is a toolbar with various icons.

	id integer	cliente text	valor numeric
1	4	Maria	1500
2	6	Cassandra	37680

- Todavia, note que ao fazermos a **query** (consulta) sobre a **VIEW PEDIDOS_GRANDES**, apenas dois registros são apresentados: Maria e Cassandra.
- São os registros que possuem valor maior que 1000.
- Maria tem 1500.
- Cassandra tem 37680.

Exemplo de RULE

```
9  
10 ✓ CREATE RULE inserir_pedidos_grandes AS  
11   ON INSERT TO pedidos_grandes  
12   DO INSTEAD  
13     INSERT INTO pedidos (cliente, valor)  
14     VALUES (NEW.cliente, NEW.valor);  
15
```

- Criamos uma **regra (RULE)** de nome **`inserir_pedidos_grandes`** que será acionada sempre fizermos uma **inserção (INSERT)** na **VIEW `pedidos_grandes`** (“**ON INSERT TO `pedidos_grandes`**”).
- Essa **regra deve ser executada em vez (“DO INSTEAD”)** da inserção original.
- E o que exatamente essa regra faz?

Exemplo de RULE

```
9  
10 ✓ CREATE RULE inserir_pedidos_grandes AS  
11   ON INSERT TO pedidos_grandes  
12   DO INSTEAD  
13     INSERT INTO pedidos (cliente, valor)  
14     VALUES (NEW.cliente, NEW.valor);  
15  
16 ✓ INSERT INTO pedidos_grandes (cliente, valor)  
17   VALUES ('Mariana', 10);  
18
```

- Ora, essa regra impede, por exemplo, que esse INSERT para a cliente MARIANA resulte em um ERRO.
- Observe que a inserção foi feita sobre a VIEW e não sobre a TABELA diretamente.
- Ocorre que a VIEW trabalha apenas com valores maiores que 1000.
- E o valor do pedido de Mariana é somente 10.
- O registro NÃO seria inserido!

Exemplo de RULE

```
9  
10 ✓ CREATE RULE inserir_pedidos_grandes AS  
11   ON INSERT TO pedidos_grandes  
12   DO INSTEAD  
13     INSERT INTO pedidos (cliente, valor)  
14     VALUES (NEW.cliente, NEW.valor);  
15  
16 ✓ INSERT INTO pedidos_grandes (cliente, valor)  
17   VALUES ('Mariana', 10);  
18
```

- Nossa regra contorna esse problema ao interromper a Query original:
- “**INSERT INTO pedidos_grandes (cliente, valor) VALUES ('Mariana', 10);**”
- E a substitui (INSTEAD) pelo comando da RULE:
- “**INSERT INTO pedidos (cliente, valor) VALUES (NEW.cliente, NEW.valor);**”

Exemplo de RULE

```
9  
10 ✓ CREATE RULE inserir_pedidos_grandes AS  
11   ON INSERT TO pedidos_grandes  
12   DO INSTEAD  
13     INSERT INTO pedidos (cliente, valor)  
14     VALUES (NEW.cliente, NEW.valor);  
15  
16 ✓ INSERT INTO pedidos_grandes (cliente, valor)  
17   VALUES ('Mariana', 10);  
18
```

- NEW.cliente corresponde ao valor que a Query original tentava inserir para cliente ('Mariana').
- NEW.valor é o valor (10) que a instrução INSERT original tentava inserir.

Exemplo de RULE

```
VALUES ('Cassandra', 37680),  
24  
25   SELECT * FROM PEDIDOS;  
26  
27   SELECT * FROM PEDIDOS_GRANDES;
```

Data Output Messages Notifications

SQL

	id [PK] integer	cliente text	valor numeric
1	4	Maria	1500
2	5	José	400
3	6	Cassandra	37680
4	7	Mariana	10

```
24  
25   SELECT * FROM PEDIDOS;  
26  
27   SELECT * FROM PEDIDOS_GRANDES;
```

Data Output Messages Notifications

SQL

	id integer	cliente text	valor numeric
1	4	Maria	1500
2	6	Cassandra	37680

As diferenças entre RULES e TRIGGERS

- **Rules** atuam na reescrita da query antes da execução.
 - **Triggers** atuam durante ou após a execução.
-
- **Rules** podem gerar efeitos colaterais inesperados se mal usadas, especialmente com múltiplas regras sobre a mesma operação.
 - Hoje em dia, para a maioria dos casos de views atualizáveis, o **PostgreSQL** recomenda **INSTEAD OF TRIGGERS** em vez de **rules**, por serem mais previsíveis.
 - Ainda assim, **Rules** continuam úteis para lógica de roteamento de queries e restrições de acesso indireto.

INSTEAD OF TRIGGER

- O **INSTEAD OF TRIGGER** é um tipo especial de trigger no PostgreSQL que serve justamente para o que as RULES faziam antes: permitir que você intercepte e substitua a execução de operações (**INSERT**, **UPDATE**, **DELETE**) em uma **VIEW**.
- Normalmente, **triggers** funcionam em **tabelas reais** e disparam antes (**BEFORE**) ou depois (**AFTER**) de uma operação já estar em execução.
- Mas, em **views**, não há dados armazenados diretamente, então não há como executar essas operações “de verdade”.

INSTEAD OF TRIGGER

- O **INSTEAD OF TRIGGER** resolve isso, dizendo ao **PostgreSQL**:
- “**Quando alguém tentar executar essa operação na view, em vez (instead) de fazer o que faria normalmente, execute este código.**”

```
CREATE TRIGGER nome_do_trigger
  INSTEAD OF { INSERT | UPDATE | DELETE }
  ON nome_da_view
  FOR EACH ROW
  EXECUTE FUNCTION nome_da_funcao();
```

INSTEAD OF TRIGGER

```
41  
42 -- Função que tratará o INSERT  
43 CREATE OR REPLACE FUNCTION inserir_pedidos_grandes()  
44 RETURNS trigger AS $$  
45 BEGIN  
46     INSERT INTO pedidos (cliente, valor)  
47         VALUES (NEW.cliente, NEW.valor);  
48     RETURN NULL; -- não insere nada na view diretamente  
49 END;  
50 $$ LANGUAGE plpgsql;  
51
```

```
51  
52 -- Trigger INSTEAD OF na view  
53 CREATE TRIGGER trg_inserir_pedidos_grandes  
54 INSTEAD OF INSERT ON pedidos_grandes  
55 FOR EACH ROW  
56 EXECUTE FUNCTION inserir_pedidos_grandes();  
57  
58  
59  
60
```

Data Output Messages Notifications

CREATE TRIGGER

Query returned successfully in 82 msec.

INSTEAD OF TRIGGER



Diferença chave RULE vs INSTEAD OF TRIGGER

Característica	RULES	INSTEAD OF TRIGGERS
Nível de atuação	Reescrevem a query antes do otimizador	Executam código no momento da operação
Controle de lógica	Mais limitado, apenas redirecionamento	PL/pgSQL completo, condições e loops
Previsibilidade	Pode gerar execuções duplicadas	Mais previsível e controlável
Recomendação atual	Uso apenas em casos específicos	Uso preferencial para views atualizáveis

Outro Exemplo de RULE

```
63  
64 ✓ CREATE TABLE log_pedidos (  
65     id SERIAL PRIMARY KEY,  
66     id_pedido INT,  
67     cliente TEXT,  
68     valor NUMERIC,  
69     data_log TIMESTAMP DEFAULT now()  
70 );  
71  
72
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 106 msec.

- Criamos uma tabela de LOG simplificada para PEDIDOS.

Outro Exemplo de RULE

```
1  
2  
3  CREATE RULE log_insercao_pedidos AS  
4  ON INSERT TO pedidos  
5  DO ALSO  
6  INSERT INTO log_pedidos (id_pedido, cliente, valor)  
7  VALUES (NEW.id, NEW.cliente, NEW.valor);  
8  
9
```

Data Output Messages Notifications

CREATE RULE

Query returned successfully in 84 msec.

- Agora criamos uma RULE para quando ocorrer uma nova inserção na TABELA PEDIDOS.
- Tudo que essa RULE faz é **também (“DO ALSO”)** **executar esse novo código** que duplica o registro na tabela de LOG.
- Note que a RULE não substitui a QUERY ORIGINAL.
- A RULE mantem a execução da QUERY ORIGINAL e complementa com uma nova QUERY.

Outro Exemplo de RULE

```
62  
63  CREATE TABLE log_pedidos (  
64    id SERIAL PRIMARY KEY,  
65    id_pedido INT,  
66    cliente TEXT,  
67    valor NUMERIC,  
68    data_log TIMESTAMP DEFAULT now()  
69 );  
70  
71  
72  
73  CREATE RULE log_insercao_pedidos AS  
74    ON INSERT TO pedidos  
75    DO ALSO  
76      INSERT INTO log_pedidos (id_pedido, cliente, valor)  
77      VALUES (NEW.id, NEW.cliente, NEW.valor);  
78  
79      INSERT INTO PEDIDOS (CLIENTE, VALOR) VALUES ('RONALDO', 679);  
80
```

Data Output Messages Notifications

INSERT 0 1

Query returned successfully in 83 msec.

Outro Exemplo de RULE

```
78  
79   INSERT INTO PEDIDOS (CLIENTE, VALOR) VALUES ('RONALDO', 679);  
80  
81   SELECT * FROM PEDIDOS;
```

Data Output Messages Notifications

SQL

	id [PK] integer	cliente text	valor numeric
1	4	Maria	1500
2	5	José	400
3	6	Cassandra	37680
4	7	Mariana	10
5	8	RONALDO	679

```
78  
79   INSERT INTO PEDIDOS (CLIENTE, VALOR) VALUES ('RONALDO', 679);  
80  
81   SELECT * FROM PEDIDOS;  
82  
83   SELECT * FROM LOG_PEDIDOS;
```

Data Output Messages Notifications

SQL

	id [PK] integer	id_pedido integer	cliente text	valor numeric	data_log timestamp without time zone
1	1	9	RONALDO	679	2025-08-09 13:56:28.2341

Outro Exemplo de RULE

- Quando fazemos o INSERT em pedidos:
 - A inserção original ainda é feita (porque usamos DO ALSO, não DO INSTEAD).
 - Automaticamente o PostgreSQL executa um segundo comando para inserir na log_pedidos.
 - Isso é feito no nível de reescrita da query, antes mesmo da execução começar.
- Como as RULES são processadas antes do otimizador, se um único INSERT afetar várias linhas, a regra será aplicada a cada linha individualmente, o que pode gerar comportamento inesperado se não planejado com cuidado.

Um Terceiro Exemplo de RULE

```
79  
80  
81  
82  
83 CREATE RULE bloquear_update_pedidos AS  
84   ON UPDATE TO pedidos  
85   DO INSTEAD NOTHING;  
86  
87
```

Data Output Messages Notifications

CREATE RULE

Query returned successfully in 96 msec.

```
82  
83 CREATE RULE bloquear_update_pedidos AS  
84   ON UPDATE TO pedidos  
85   DO INSTEAD NOTHING;  
86  
87 SELECT * FROM PEDIDOS;  
88
```

Data Output Messages Notifications

SQL

	id [PK] integer	cliente text	valor numeric
1	4	Maria	1500
2	5	José	400
3	6	Cassandra	37680
4	7	Mariana	10
5	8	RONALDO	679

Um Terceiro Exemplo de RULE

```
81
82
83 ✓ CREATE RULE bloquear_update_pedidos AS
84   ON UPDATE TO pedidos
85   DO INSTEAD NOTHING;
86
87 SELECT * FROM PEDIDOS;
88
89 UPDATE PEDIDOS SET VALOR = 2099.55 WHERE CLIENTE = 'CASSANDRA';
90
```

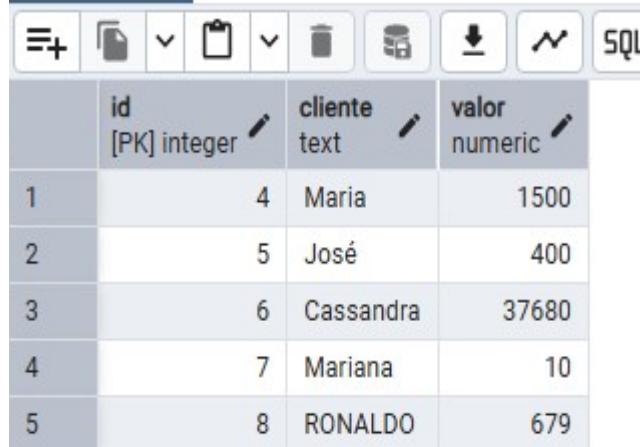
Data Output Messages Notifications

UPDATE 0

Query returned successfully in 175 msec.

```
82
83 ✓ CREATE RULE bloquear_update_pedidos AS
84   ON UPDATE TO pedidos
85   DO INSTEAD NOTHING;
86
87 SELECT * FROM PEDIDOS;
88
89 UPDATE PEDIDOS SET VALOR = 2099.55 WHERE CLIENTE = 'CASSANDRA';
90
```

Data Output Messages Notifications



	id [PK] integer	cliente text	valor numeric
1	4	Maria	1500
2	5	José	400
3	6	Cassandra	37680
4	7	Mariana	10
5	8	RONALDO	679

Um Terceiro Exemplo de RULE

- Qualquer UPDATE será **interceptado** pelo *query rewriter*.
- O comando original é **substituído** por “nada” (“**DO NOTHING**”).

Um Quarto Exemplo de RULES

The screenshot shows a database query editor interface with the following details:

- Query History Tab:** Contains the SQL code for creating two tables:

```
1 v CREATE TABLE genero (
2     id_genero SERIAL PRIMARY KEY,
3     nome TEXT NOT NULL
4 );
5
6 v CREATE TABLE filme (
7     id_filme SERIAL PRIMARY KEY,
8     titulo TEXT NOT NULL,
9     id_genero INT REFERENCES genero(id_genero)
10 );
11
```
- Data Output Tab:** Shows the message "CREATE TABLE".
- Messages Tab:** Shows the message "Query returned successfully in 88 msec."

Um Quarto Exemplo de RULES

Query Query History

```
11
12 CREATE OR REPLACE FUNCTION excluir_genero_seguro(p_id_genero INT)
13 RETURNS VOID AS $$ 
14 BEGIN
15     -- Verifica se o gênero está sendo usado em algum filme
16     IF EXISTS (SELECT 1 FROM filme WHERE id_genero = p_id_genero) THEN
17         RAISE EXCEPTION 'Não é possível excluir: o gênero % está vinculado a pelo menos um filme.', p_id_genero;
18     END IF;
19
20     -- Se não houver filmes, exclui o gênero
21     DELETE FROM genero WHERE id_genero = p_id_genero;
22 END;
23 $$ LANGUAGE plpgsql;
24
25
```

Data Output Messages Notifications

CREATE FUNCTION

Query returned successfully in 76 msec.

Um Quarto Exemplo de RULES

```
24  
25 INSERT INTO GENERO (NOME) VALUES ('EPICO'),('COMEDIA'),('FICCAO CIENTÍFICA');  
26
```

Data Output Messages Notifications

INSERT 0 3

Query returned successfully in 69 msec.

The screenshot shows a PostgreSQL query editor interface. At the top, there are tabs for 'Query' and 'Query History'. On the right, there is a button labeled 'Execute script' with the F5 key indicator. The main area contains the following SQL code:

```
22 END;  
23 $$ LANGUAGE plpgsql;  
24  
25 INSERT INTO GENERO (NOME) VALUES ('EPICO'),('COMEDIA'),('FICCAO CIENTÍFICA');  
26  
27 SELECT * FROM GENERO;
```

Below the code, there are three tabs: 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is selected. It displays a table with the following data:

	id_genero [PK] integer	nome text
1	1	EPICO
2	2	COMEDIA
3	3	FICCAO CIENTÍFICA

Um Quarto Exemplo de RULES

The screenshot shows a database query interface with the following details:

- Query History:** A tab at the top left.
- Query:** The main area contains the following SQL code:

```
24
25 INSERT INTO GENERO (NOME) VALUES ('EPICO'),('COMEDIA'),('FICCAO CIENTÍFICA');
26
27 SELECT * FROM GENERO;
28
29 INSERT INTO FILME (TITULO, ID_GERERO) VALUES ('THE KID', 2);
```

The last line, `29` (which should be `28`), is highlighted with a light blue background.
- Data Output:** A tab at the bottom left.
- Messages:** A tab at the bottom center, currently selected.
- Notifications:** A tab at the bottom right.
- Messages:** The message area displays:

```
INSERT 0 1
```
- Notifications:** The message area also displays:

```
Query returned successfully in 98 msec.
```

Um Quarto Exemplo de RULES

The screenshot shows a PostgreSQL query editor interface. At the top, there are tabs for 'Query' and 'Query History'. Below the tabs, the query text is displayed in a code editor-like area:

```
30
31 CREATE RULE validar_exclusao_genero AS
32   ON DELETE TO genero
33   DO INSTEAD
34   SELECT excluir_genero_seguro(OLD.id_genero);
35
36
37
```

Below the code editor, there are three tabs: 'Data Output', 'Messages', and 'Notifications'. The 'Messages' tab is currently selected, showing the message 'CREATE RULE'. At the bottom of the interface, a status message reads 'Query returned successfully in 73 msec.'

- Nesse exemplo, a **RULE** interrompe qualquer comando **DELETE** executado sobre a tabela **GENERO** e o substitui pela execução da Função PLPGSQL **excluir_gênero_seguro()**.

Um Quarto Exemplo de RULES

The screenshot shows a PostgreSQL query editor interface. The top navigation bar has tabs for 'Query' (which is selected) and 'Query History'. The main area contains the following SQL code:

```
30
31 ✓ CREATE RULE validar_exclusao_genero AS
32   ON DELETE TO genero
33   DO INSTEAD
34     SELECT excluir_genero_seguro(OLD.id_genero);
35
36 DELETE FROM GENERO WHERE ID_GENERO = 2;
```

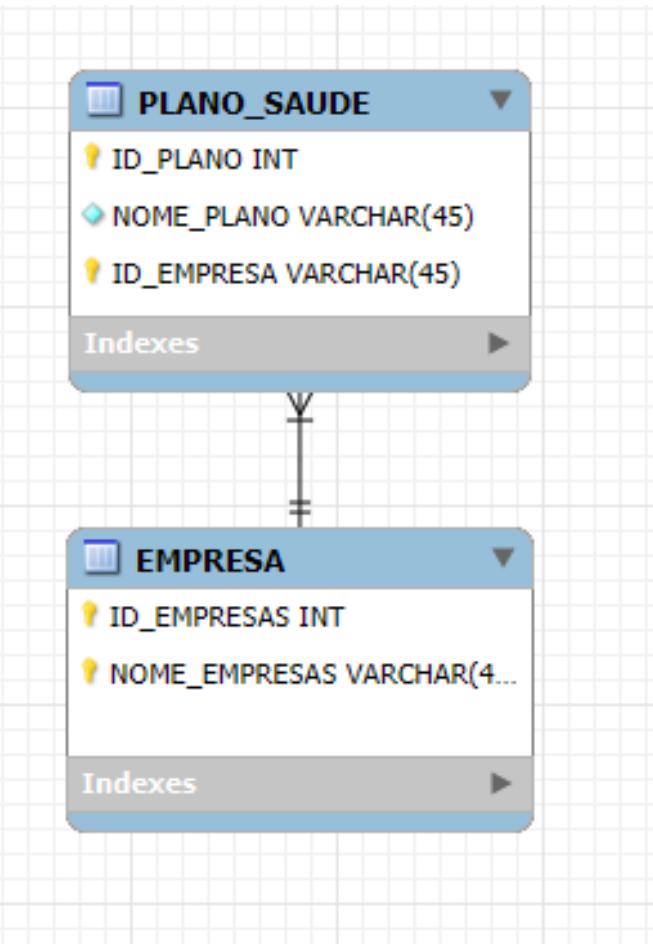
Below the code, there are three tabs: 'Data Output', 'Messages' (which is selected), and 'Notifications'. The 'Messages' tab displays the following error message:

ERROR: Não é possível excluir: o gênero 2 está vinculado a pelo menos um filme.
CONTEXT: função PL/pgSQL excluir_genero_seguro(integer) linha 5 em RAISE

ERRO: Não é possível excluir: o gênero 2 está vinculado a pelo menos um filme.
SQL state: P0001

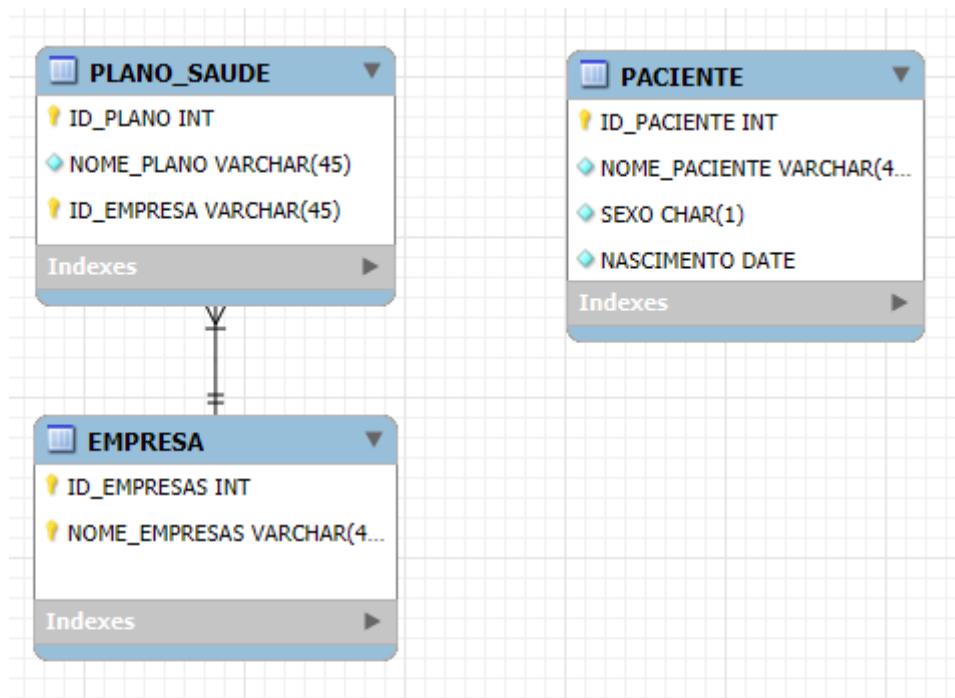
- **Testando nossa RULE.**
- Aqui fizemos uma RULE que invoca uma Função PLPGSQL.
- Todavia, não seria possível ter uma RULE que invocasse um STORED PROCEDURE.

Exercício Resolvido



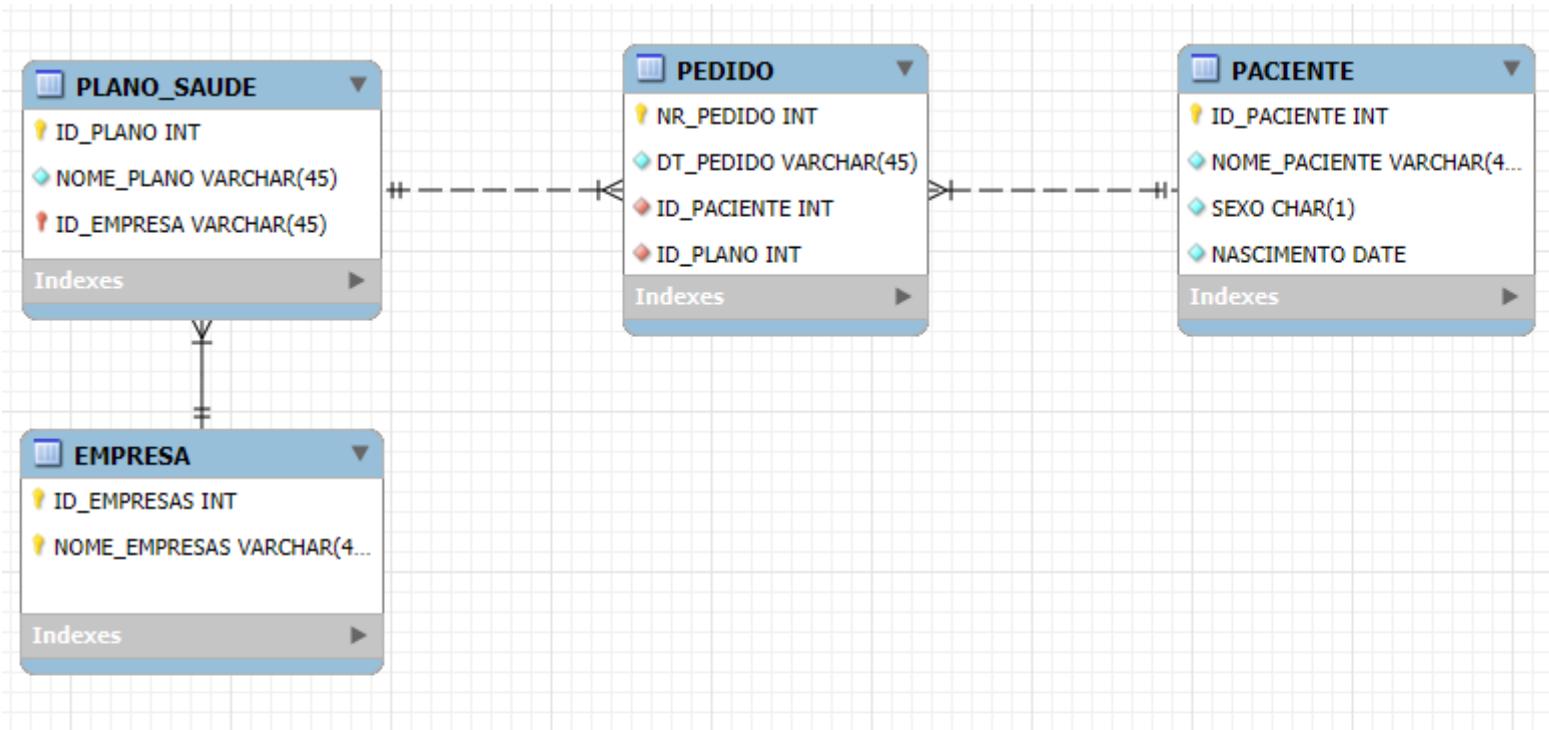
- Considere um Sistema de Análise Laboratorial.
- O referido sistema pode realizar exames laboratoriais (de sangue, fezes, urina etc.) para clientes com e sem convênios de saúde.
- A entidade **EMPRESA** é responsável pela manutenção dos referidos convênios de saúde. Estas empresas poderiam ser: “UNIMED”, “São Bernardo Saúde”, “Golden Cross” etc.
- Tais **EMPRESAS** oferecem Planos de Saúde para seus clientes que podem realizar exames laboratoriais de saúde.

Exercício Resolvido



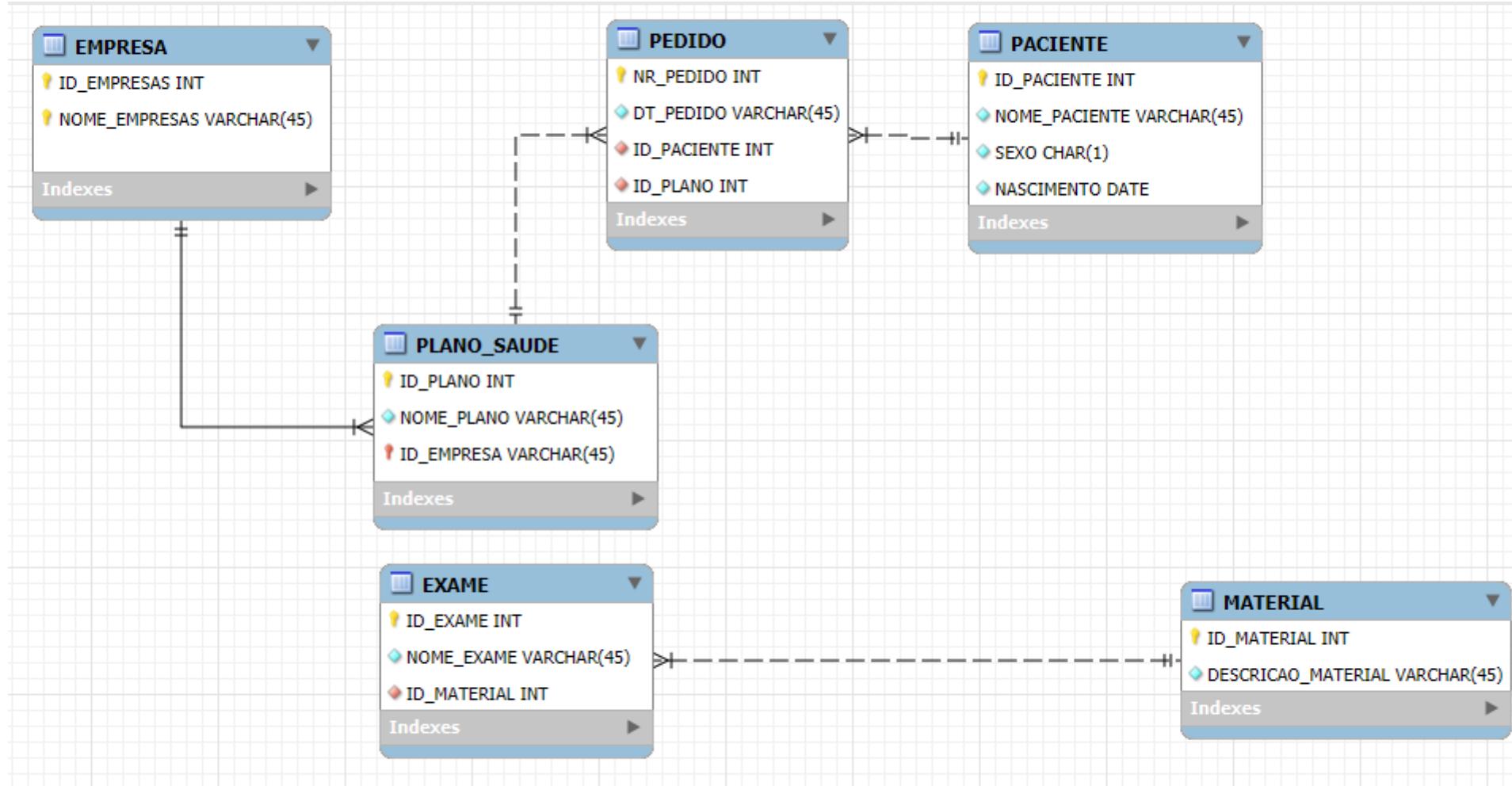
- A Tabela PACIENTE armazena os dados de todos os pacientes que realizaram exames na empresa.
- Para efeito de simplificação seus atributos serão apenas: `ID_PACIENTE`, `NOME_PACIENTE`, `SEXO` e `NASCIMENTO`.
- Na vida real teríamos CPF, Identidade, Endereço, Telefone, Celular etc.
- Note que NÃO há qualquer RELACIONAMENTO entre PACIENTE e PLANO_SAUDE.

Exercício Resolvido



- Um PEDIDO corresponde a um conjunto de exames solicitados em uma data específica (DT_PEDIDO), por um dado PACIENTE (ID_PACIENTE) e por meio de um PLANO_SAÚDE (ID_PLANO).
- Quando o PACIENTE não possuir um plano de saúde ele fará uso de um registro de PLANO_SAÚDE “Particular”.

Exercício Resolvido

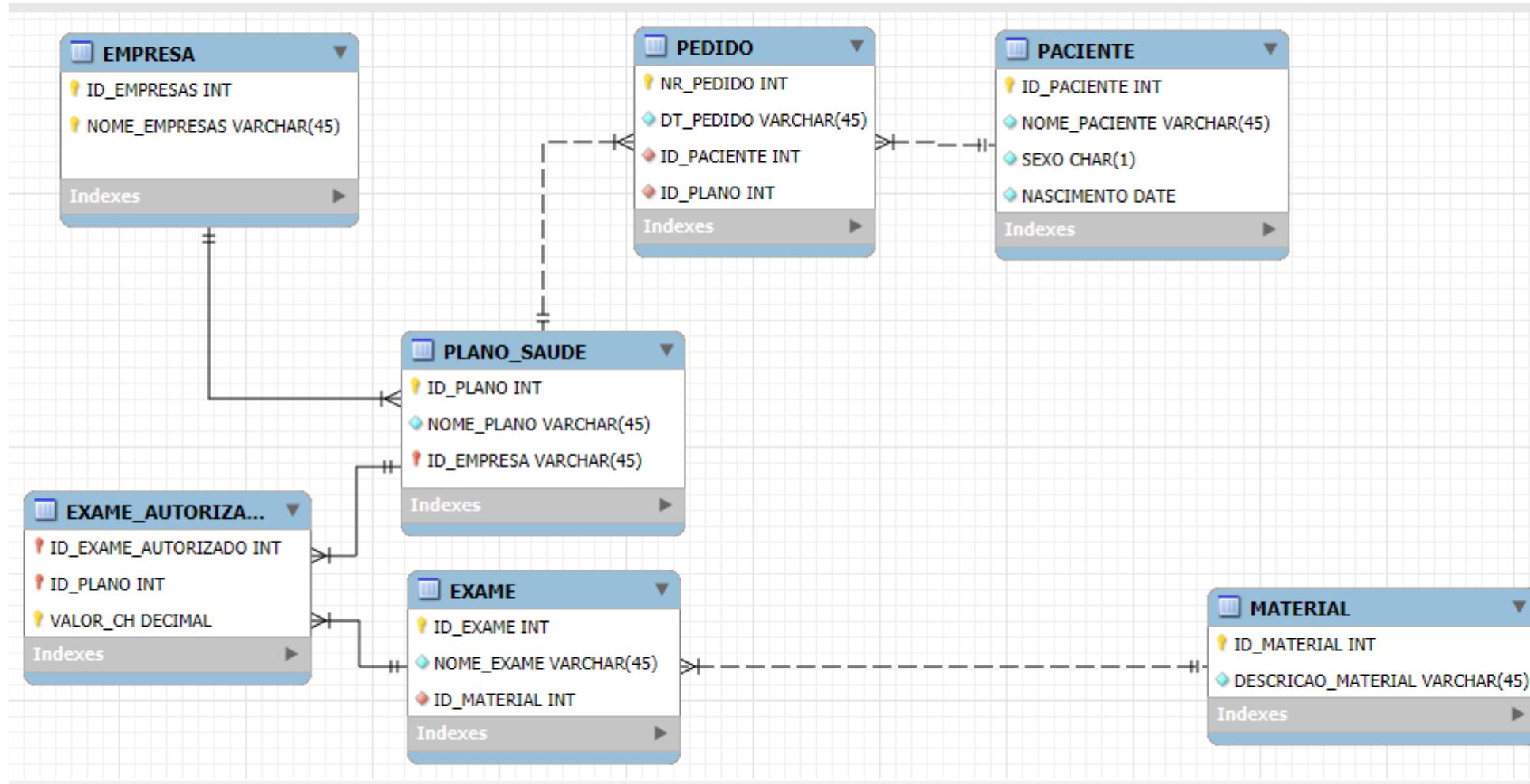


Exercício Resolvido

- EXAME armazena dados sobre todos os exames que o Laboratório está habilitado a fazer.
- EXAME está ligado a MATERIAL que classifica o material usado pelo exame em questão (“Sangue”, “Urina”, “Fezes” etc).



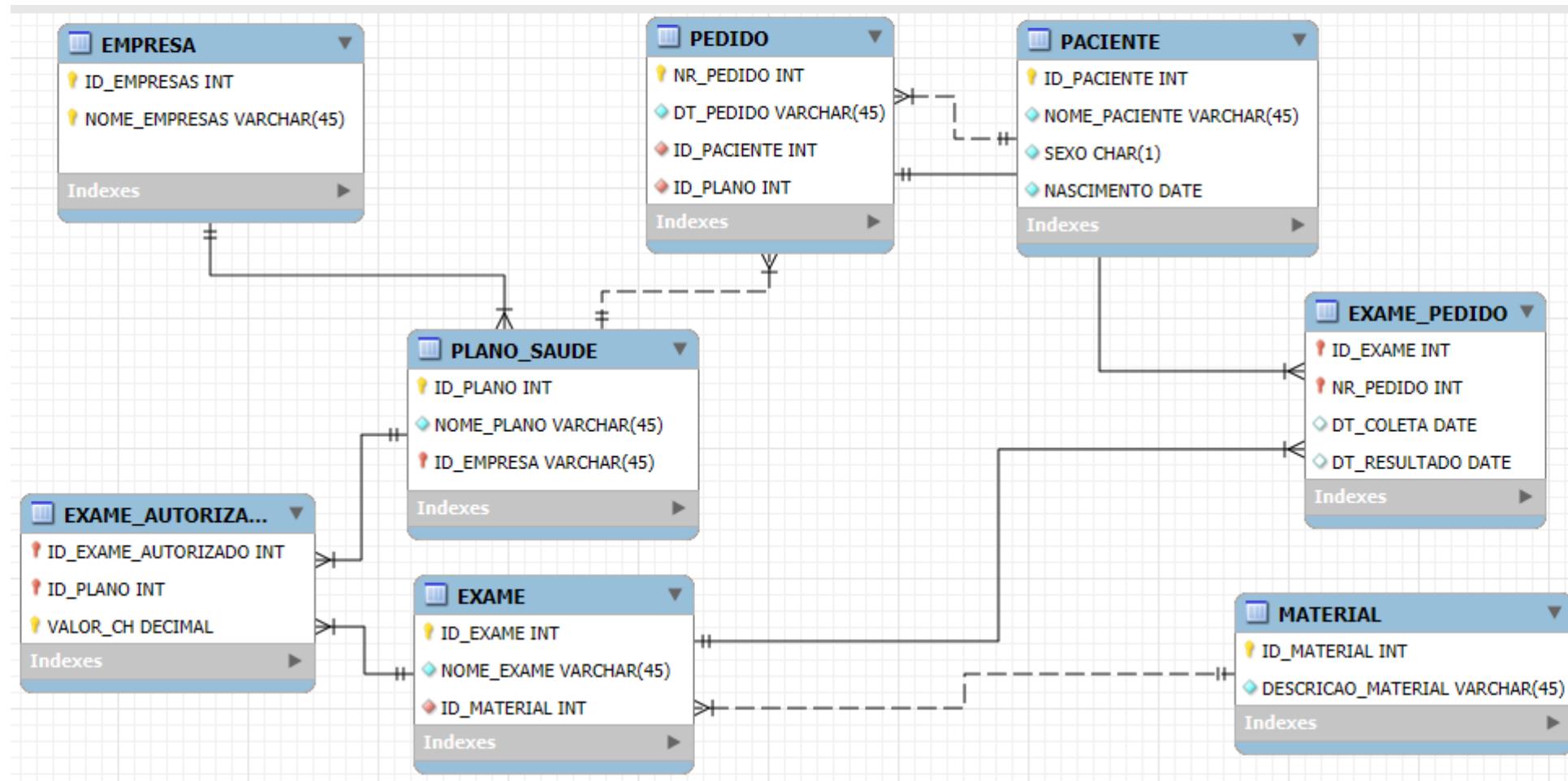
Exercício Resolvido



Exercício Resolvido

- EXAME_AUTORIZADO corresponde a um conjunto de EXAMES que são possíveis de serem solicitados por um PACIENTE que possui um dado PLANO_SAÚDE.
- Esses EXAMES_AUTORIZADOS são definidos em um contrato entre o Laboratório e a Empresa que oferece o Plano de Saúde.
- O PACIENTE não pode efetuar um EXAME pelo PLANO_SAÚDE que não esteja em EXAME_AUTORIZADO.

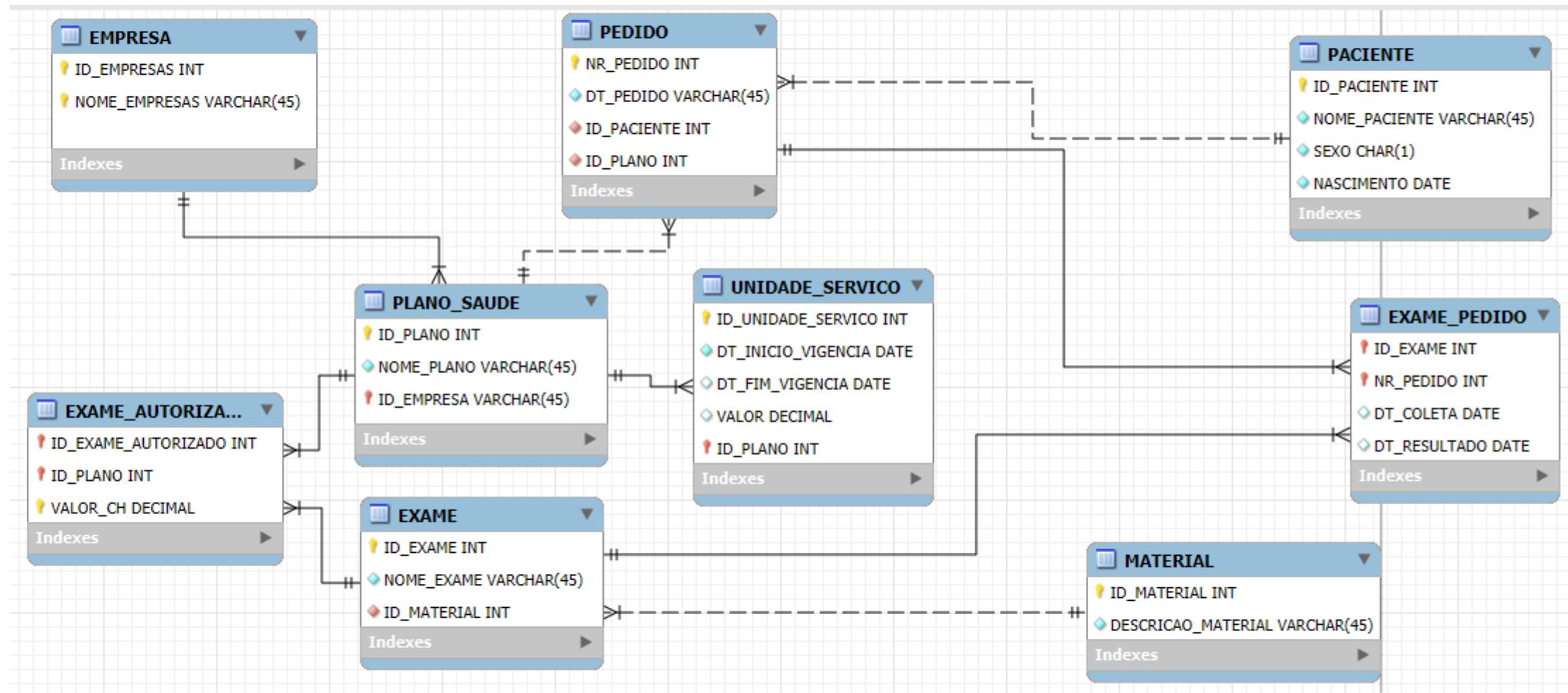
Exercício Resolvido



Exercício Resolvido

- EXAME_PEDIDO corresponde aos EXAMES solicitados em um dado PEDIDO de um PACIENTE.
- O PACIENTE chega com uma solicitação do médico.
- O sistema valida cada EXAME para saber se o mesmo é um EXAME_AUTORIZADO pelo PLANO_SAÚDE.
- Caso seja autorizado um novo registro é inserido em EXAME_PEDIDO.

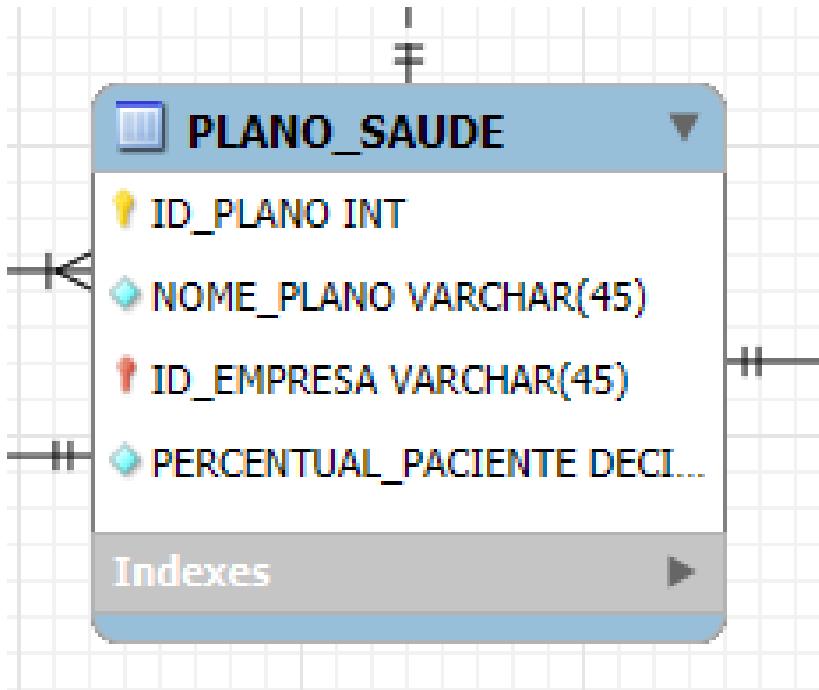
Exercício Resolvido



Exercício Resolvido

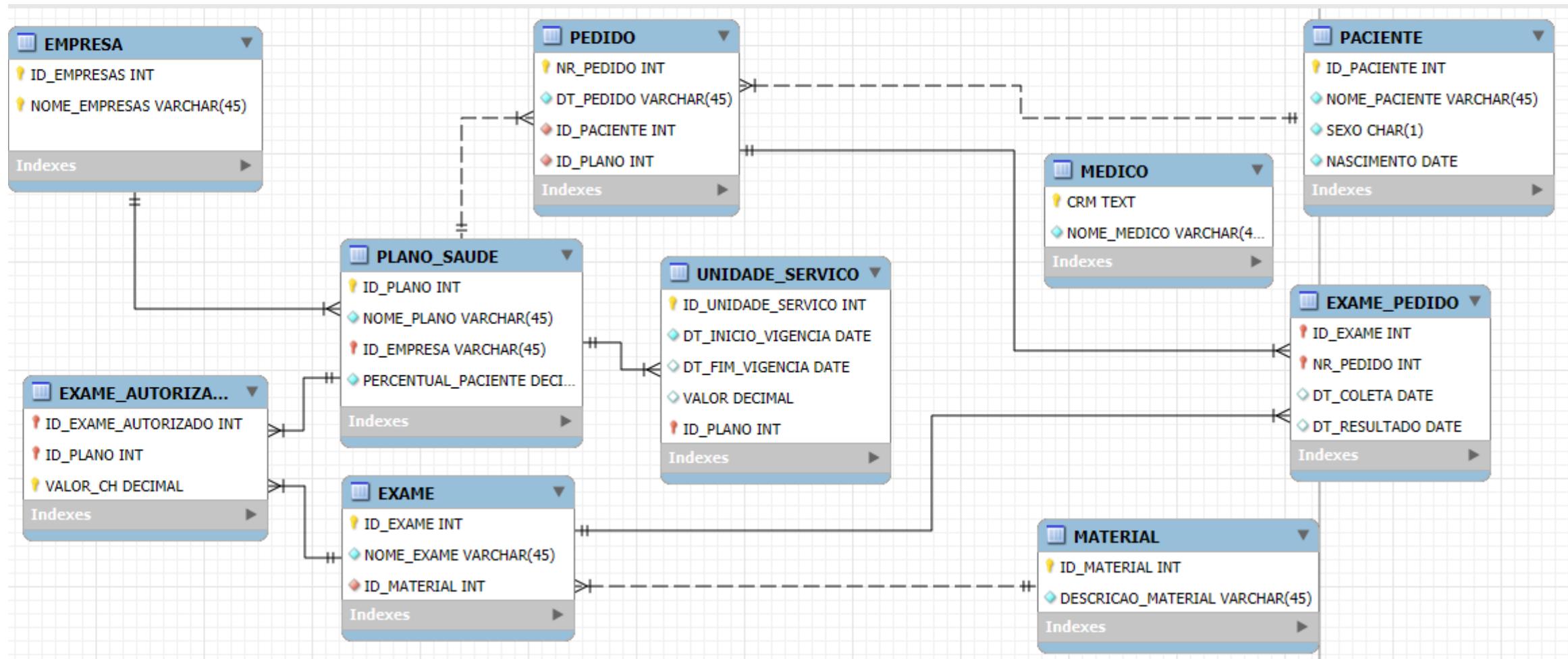
- UNIDADE_SERVICO (US) é utilizada no cálculo do valor a ser cobrado do PACIENTE ou do PLANO_SAÚDE (convênio).
- O PACIENTE informa qual é o seu PLANO_SAÚDE e, ao entregar a requisição de exames feito pelo médico, o PEDIDO é gerado.
- Para cada EXAME_PEDIDO gerado o sistema acessa o valor do CH (Coeficiente de Honorários) em EXAME_AUTORIZADO e o valor da US (unidade de serviço) em UNIDADE_SERVICO.
- O preço do exame corresponde ao produto: CH x US.

Exercício Resolvido



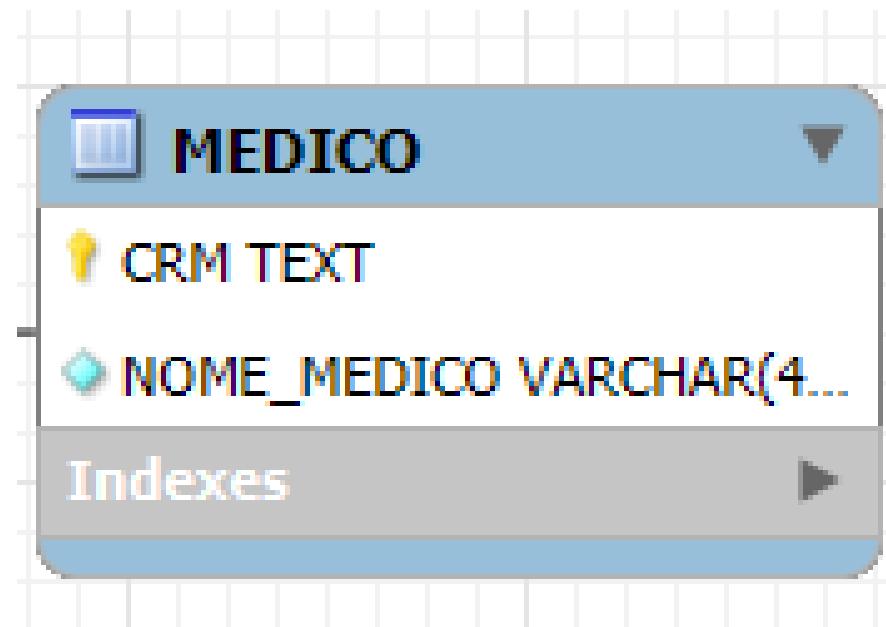
- Acrescentamos a coluna PERCENTUAL_PACIENTE na tabela PLANO_SAUDE.
- Em alguns planos quem paga o valor total dos EXAMES ao Laboratório é o Plano de Saúde (PERCENTUAL_PACIENTE = 0.00).
- Em outros casos, o PACIENTE arca contratualmente com parte do valor dos exames pedidos. Por exemplo, 30% (PERCENTUAL_PACIENTE = 0.30).

Exercício Resolvido

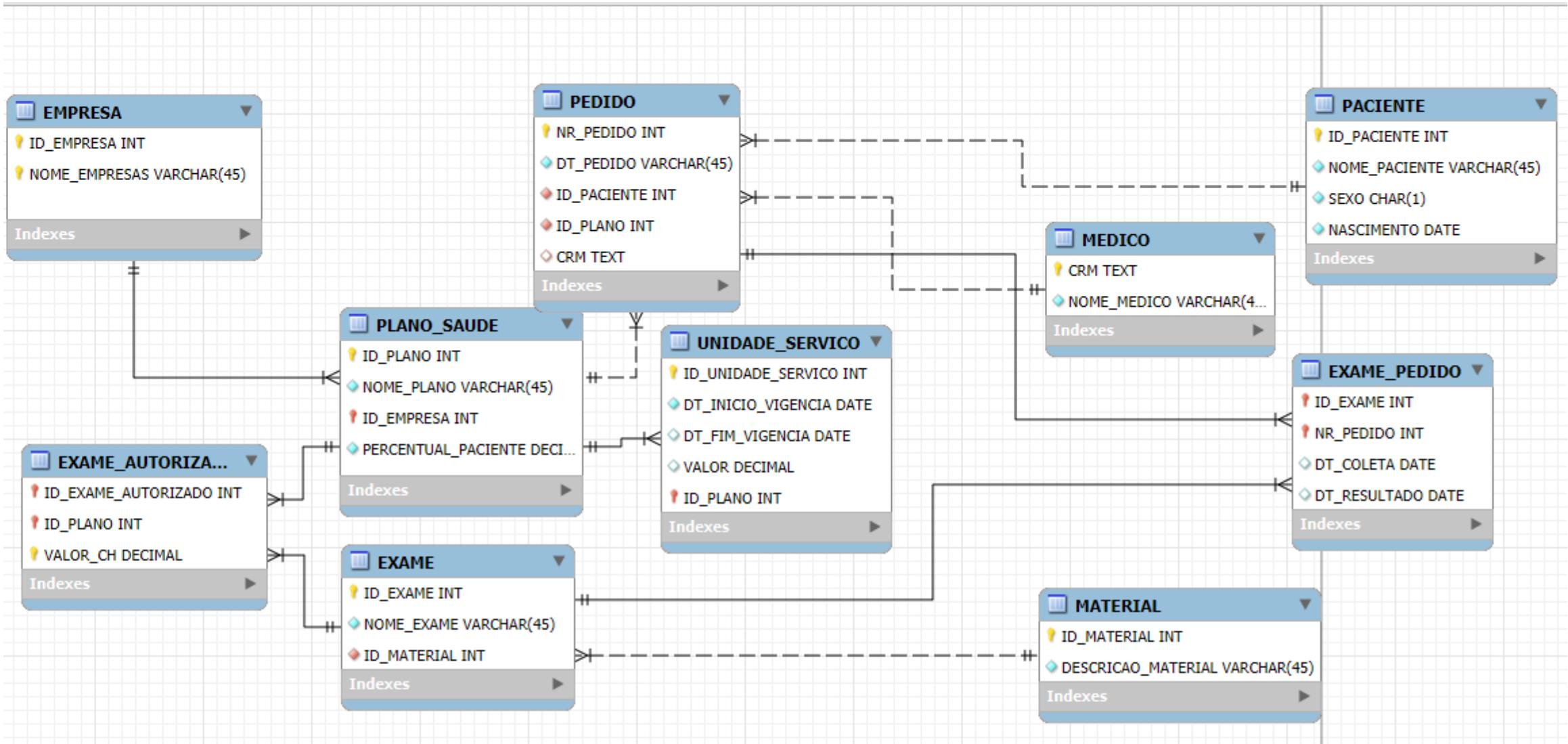


Exercício Resolvido

- A entidade MEDICO armazena os dados de médicos que são responsáveis pela solicitação do EXAME_PEDIDO.
- Ela deve se relacionar com o PEDIDO.



Exercício Resolvido



Exercício Resolvido

The screenshot shows a PostgreSQL query editor interface. The title bar indicates the connection is to 'LABORATORIO/postgres@PostgreSQL 17'. The toolbar includes various icons for file operations, search, and navigation. Below the toolbar, there are tabs for 'Query' (which is selected) and 'Query History'. The main area displays a numbered SQL script:

```
1 -----
2 -- LABORATÓRIO DE ANÁLISES CLÍNICAS --
3 ----- 14-08-2025 -----
4 -----
5
6 CREATE TABLE EMPRESA (
7     ID_EMPRESA SERIAL NOT NULL PRIMARY KEY,
8     NOME_EMPRESA VARCHAR(50) NOT NULL
9 );
10
11 INSERT INTO EMPRESA (NOME_EMPRESA) VALUES ('*** INEXISTENTE ***'),
12                 ('UNIMED COOPERATIVA'), ('SAO BERNARDO SAUDE'),
13                 ('FUNSSEST'), ('GOLDEN CROSS ENTERPRISES');
```

The code creates a table 'EMPRESA' with two columns: 'ID_EMPRESA' (primary key, serial type) and 'NOME_EMPRESA' (varchar(50), not null). It then inserts four rows into the table. The first row is a placeholder with three asterisks. The subsequent three rows list company names: 'UNIMED COOPERATIVA', 'SAO BERNARDO SAUDE', and 'FUNSSEST', followed by 'GOLDEN CROSS ENTERPRISES'. The bottom of the editor shows tabs for 'Data Output', 'Messages' (which is selected), and 'Notifications', along with the message 'INSERT 0 5'.

Query returned successfully in 79 msec.

Exercício Resolv

The screenshot shows a PostgreSQL query editor interface with the following details:

- Connection:** LABORATORIO/postgres@PostgreSQL 17
- Toolbar:** Includes icons for file operations, search, and various database functions.
- Query Tab:** Active tab, showing the SQL code.
- Code:** The code creates a table `PLANO_SAUDE` with columns `ID_PLANO`, `NOME_PLANO`, and `ID_EMPRESA`, and includes a foreign key constraint referencing `EMPRESA`. It then inserts four rows into the table.

```
15
16
17 CREATE TABLE PLANO_SAUDE (
18     ID_PLANO SERIAL NOT NULL PRIMARY KEY,
19     NOME_PLANO VARCHAR(50) NOT NULL,
20     ID_EMPRESA INTEGER NOT NULL,
21     PERCENTUAL_PACIENTE NUMERIC,
22
23     FOREIGN KEY(ID_EMPRESA) REFERENCES EMPRESA (ID_EMPRESA)
24     ON DELETE RESTRICT
25 );
26
27 INSERT INTO PLANO_SAUDE (NOME_PLANO, ID_EMPRESA, PERCENTUAL_PACIENTE)
28 VALUES ('UNIMED INTEGRAL',2, 0.00),
29         ('UNIMED PARTICIPACAO GOLDEN',2, 0.20),
30         ('UNIMED PARTICIPACAO SILVER',2, 0.30),
31         ('SAO BERNARDO INTEGRAL',3, 0.00);
```

- Data Output Tab:** Shows the result of the `INSERT` command: `INSERT 0 4`.
- Messages Tab:** Shows the message: `Query returned successfully in 83 msec.`

Exercício Resolvido

Query Query History

```
33
34
35
36 CREATE TABLE PACIENTE (
37     ID_PACIENTE SERIAL NOT NULL PRIMARY KEY,
38     NOME_PACIENTE VARCHAR(45) NOT NULL,
39     SEXO CHAR(1) NOT NULL,
40     NASCIMENTO DATE NOT NULL
41 );
42
43 INSERT INTO PACIENTE (NOME_PACIENTE, SEXO, NASCIMENTO)
44 VALUES ('LIVIA XAVIER','F','2015-10-02'), ('RENATO SOUZA JR','M','2002-03-30'),
45 ('CASSANDRA SILVA','F','1961-06-17');
```

Data Output Messages Notifications

INSERT 0 3

Query returned successfully in 75 msec.

Exercício Resolvido

Query Query History

```
47
48 CREATE TABLE MEDICO (
49     CRM TEXT NOT NULL PRIMARY KEY,
50     NOME_MEDICO VARCHAR(45) NOT NULL
51 );
52
53 INSERT INTO MEDICO VALUES ('SP 125900','JOSE MADUREIRA ALENCASTRO'),
54                         ('ES 2356','MARIANA CASTRO ALLENDE');
55
56
57
```

Data Output Messages Notifications

INSERT 0 2

Query returned successfully in 81 msec.

Exercício Resolvido

Query Query History

```
56
57 CREATE TABLE PEDIDO (
58     NR_PEDIDO SERIAL NOT NULL PRIMARY KEY,
59     DT_PEDIDO DATE NOT NULL,
60     ID_PACIENTE INTEGER NOT NULL,
61     ID_PLANO INTEGER NOT NULL,
62     CRM TEXT NOT NULL,
63
64     FOREIGN KEY (ID_PACIENTE) REFERENCES PACIENTE(ID_PACIENTE)
65     ON DELETE RESTRICT,
66
67     FOREIGN KEY (ID_PLANO) REFERENCES PLANO_SAUDE (ID_PLANO)
68     ON DELETE RESTRICT,
69
70     FOREIGN KEY (CRM) REFERENCES MEDICO (CRM)
71     ON DELETE RESTRICT
72 );
73
74 INSERT INTO PEDIDO (DT_PEDIDO, ID_PACIENTE, ID_PLANO, CRM) VALUES
75     ('2025-08-18', 1, 2, 'SP 125900'), ('2025-09-18', 2, 1, 'ES 2356'),
76     ('2025-11-03', 1, 2, 'SP 125900');
77
78 -----
```

Data Output Messages Notifications

INSERT 0 3

Query returned successfully in 85 msec.

Exercício Resolvido

Query Query History

```
78
79 CREATE TABLE MATERIAL (
80     ID_MATERIAL SERIAL NOT NULL PRIMARY KEY,
81     DESCRICAO_MATERIAL VARCHAR(45) NOT NULL
82 );
83
84 INSERT INTO MATERIAL VALUES (100,'SANGUE'),(110,'URINA'),(120,'FEZES'),(130,'ESPERMA');
```

Data Output Messages Notifications

INSERT 0 4

Query returned successfully in 95 msec.

Exercício Resolvido

```
Query Query History
85
86 CREATE TABLE EXAME (
87     ID_EXAME INTEGER NOT NULL PRIMARY KEY,
88     NOME_EXAME VARCHAR(50) NOT NULL,
89     ID_MATERIAL INTEGER NOT NULL,
90
91     FOREIGN KEY (ID_MATERIAL) REFERENCES MATERIAL (ID_MATERIAL)
92     ON DELETE RESTRICT
93 );
94
95 INSERT INTO EXAME VALUES (1099,'HEMOGRAMA',100), (1100,'GLICEMIA',100),
96                             (1101,'GLICEMIA',110), (1200,'GLICEMIA POS-PRANDIAL',100),
97                             (1300,'CREATININA',100), (1400,'PROTEINURIA',110),
98                             (1500,'UREIA',100), (1600,'BETA HCG',100),(1700,'PSA - ANTIGENO PROSTATICO ESPECIFICO',100),
99                             (1800,'PARASITOLOGICO', 120), (1900,'ACIDO FOLICO',100),(2000,'ESPERMOGRAMA',130),
100                            (2100,'DOSAGEM VITAMINA D', 100);
101
102
Data Output Messages Notifications
INSERT 0 13
Query returned successfully in 80 msec.
```

Query Query History

```
101
102 CREATE TABLE EXAME_AUTORIZADO (
103     ID_EXAME_AUTORIZADO INTEGER NOT NULL,
104     ID_PLANO INTEGER NOT NULL,
105     VALOR_CH NUMERIC(9,2) NOT NULL,
106
107     PRIMARY KEY (ID_EXAME_AUTORIZADO, ID_PLANO),
108     FOREIGN KEY (ID_EXAME_AUTORIZADO) REFERENCES EXAME (ID_EXAME)
109     ON DELETE RESTRICT,
110
111     FOREIGN KEY (ID_PLANO) REFERENCES PLANO_SAUDE (ID_PLANO)
112     ON DELETE RESTRICT
113 );
114
115
116 INSERT INTO PLANO_SAUDE VALUES (0,'PARTICULAR - SEM CONVÊNIO', 1, 1.00);
117
118 INSERT INTO EXAME_AUTORIZADO VALUES (1099, 0, 5.99), (1099, 1, 3.00), (1099,2, 3.00), (1099,3, 3.00),(1099, 4, 2.50),
119 (1100, 0, 7.50), (1100, 1, 4.00), (1100,2, 5.00), (1100,3, 4.00),(1100, 4, 4.00),
120 (1101, 0, 6.00), (1101, 1, 4.00), (1101,4, 3.00),
121 (1200, 0, 6.00), (1200, 1, 5.00), (1200,2, 4.50), (1200,3, 4.00),(1200,4, 3.50),
122 (1300, 0, 4.00), (1300, 1, 4.00), (1300,2, 3.50), (1300,3, 3.00),(1300,4, 2.00),
123 (1400, 0, 9.00), (1400, 1,10.00), (1400,2,12.00), (1400,3,10.00),(1400,4,10.00),
124 (1500, 0, 8.00), (1500, 1, 8.00), (1500,2,10.00), (1500,3,12.00),(1500,4,8.00),
125 (1600,0, 7.00), (1600, 1, 7.50), (1600,2,8.00), (1600,3,9.00),(1600,4,10.00),
126 (1700,0, 8.00), (1700, 1, 8.00), (1700,2,8.00), (1700,3,8.00),(1700,4,8.00),
127 (1800,0,10.00), (1800, 4, 9.00),
128 (1900,0, 15.00), (1900,1, 12.00),
129 (2000,0, 10.00),
130 (2100,0, 20.00);
```

Data Output Messages Notifications

INSERT 0 49

Query returned successfully in 120 msec.

Exercício Resolvido

Query Query History

```
133
134 ✓ CREATE TABLE EXAME_PEDIDO (
135     NR_PEDIDO INTEGER NOT NULL,
136     ID_EXAME_PEDIDO INTEGER NOT NULL,
137     DT_COLETA DATE,
138     VALOR_PACIENTE NUMERIC(9,2) NOT NULL,
139     VALOR_PLANO NUMERIC(9,2) NOT NULL,
140
141     FOREIGN KEY (NR_PEDIDO) REFERENCES PEDIDO (NR_PEDIDO)
142     ON DELETE RESTRICT,
143
144     FOREIGN KEY (ID_EXAME_PEDIDO) REFERENCES EXAME (ID_EXAME)
145     ON DELETE RESTRICT
146 );
147
148 SELECT * FROM PEDIDO;
149
150 ✓ INSERT INTO EXAME_PEDIDO VALUES (1,1099,'2025-08-18',0, 60.00),
151             (1,1500, NULL, 0, 40.50), (1, 1100,'2025-08-20', 0, 35.00),
152             (2,1099,'2025-09-18', 0, 70.00),(2,1200,'2025-09-18', 0 ,40.00),
153             (3,1300,'2025-11-03',0, 92.00), (3,1400,'2025-11-03',0, 95.00);
```

Data Output Messages Notifications

INSERT 0 7

Query returned successfully in 78 msec.

Exercício Resolvido

```
Query  Query History
154
155 -----
156 CREATE TABLE UNIDADE_SERVICO (
157     ID_PLANO INTEGER NOT NULL,
158     DT_INICIO_VIGENCIA DATE NOT NULL,
159     DT_FIM_VIGENCIA DATE,
160     VALOR NUMERIC(9,2) NOT NULL,
161
162     PRIMARY KEY (ID_PLANO, DT_INICIO_VIGENCIA),
163
164     FOREIGN KEY (ID_PLANO) REFERENCES PLANO_SAUDE (ID_PLANO)
165     ON DELETE RESTRICT
166 );
167
168 INSERT INTO UNIDADE_SERVICO VALUES (0, '2024-01-02', '2025-01-01', 2.50), (0, '2025-01-02', NULL, 3.00),
169                                         (1, '2025-01-02', NULL, 1.80), (2, '2025-04-01', NULL, 2.00),
170                                         (3, '2025-02-10', NULL, 2.10), (4, '2025-01-02', NULL, 2.40);
171
172 -----
173
```

Data Output Messages Notifications

INSERT 0 6

Query returned successfully in 97 msec.

Query Query History

```
1 -- Visão (VIEW) que efetua o cálculo do valor dos exames
2 ✓ CREATE OR REPLACE VIEW vw_exame_pedido_valores AS
3 SELECT
4     ep.nr_pedido,
5     ep.id_exame_pedido,
6     e.nome_exame,
7     p.dt_pedido,
8     ps.nome_plano,
9     ea.valor_ch,
10    us.valor AS valor_unidade,
11    (ea.valor_ch * us.valor) AS valor_total,
12    ROUND((ea.valor_ch * us.valor) * ps.percentual_paciente, 2) AS valor_paciente,
13    ROUND((ea.valor_ch * us.valor) * (1 - ps.percentual_paciente), 2) AS valor_plano
14 FROM exame_pedido ep
15 JOIN pedido p
16     ON ep.nr_pedido = p.nr_pedido
17 JOIN plano_saude ps
18     ON p.id_plano = ps.id_plano
19 JOIN exame_autorizado ea
20     ON ep.id_exame_pedido = ea.id_exame_autorizado
21     AND p.id_plano = ea.id_plano
22 JOIN exame e
23     ON ep.id_exame_pedido = e.id_exame
24 JOIN unidade_servico us
25     ON us.id_plano = ps.id_plano
26     AND p.dt_pedido >= us.dt_inicio_vigencia
27     AND (us.dt_fim_vigencia IS NULL OR p.dt_pedido <= us.dt_fim_vigencia);
28
```

Data Output Messages Notifications

CREATE VIEW

Query returned successfully in 104 msec.

Exercício Resolvido

The screenshot shows a MySQL Workbench interface with the following details:

Query Editor:

```
19 JOIN exame_autorizado ea
20   ON ep.id_exame_pedido = ea.id_exame_autorizado
21   AND p.id_plano = ea.id_plano
22 JOIN exame e
23   ON ep.id_exame_pedido = e.id_exame
24 JOIN unidade_servico us
25   ON us.id_plano = ps.id_plano
26   AND p.dt_pedido >= us.dt_inicio_vigencia
27   AND (us.dt_fim_vigencia IS NULL OR p.dt_pedido <= us.dt_fim_vigencia);
28
29
30 SELECT * FROM vw_exame_pedido_valores;
```

Data Output:

	nr_pedido integer	id_exame_pedido integer	nome_exame character varying (50)	dt_pedido date	nome_plano character varying (50)	valor_ch numeric (9,2)	valor_unidade numeric (9,2)	valor_total numeric	valor_paciente numeric	valor_plano numeric
1	1	1099	HEMOGRAMA	2025-08-18	UNIMED PARTICIPACAO GOLDEN	3.00	2.00	6.0000	1.20	4.80
2	1	1500	UREIA	2025-08-18	UNIMED PARTICIPACAO GOLDEN	10.00	2.00	20.0000	4.00	16.00
3	1	1100	GLICEMIA	2025-08-18	UNIMED PARTICIPACAO GOLDEN	5.00	2.00	10.0000	2.00	8.00
4	2	1099	HEMOGRAMA	2025-09-18	UNIMED INTEGRAL	3.00	1.80	5.4000	0.00	5.40
5	2	1200	GLICEMIA POS-PRANDIAL	2025-09-18	UNIMED INTEGRAL	5.00	1.80	9.0000	0.00	9.00
6	3	1300	CREATININA	2025-11-03	UNIMED PARTICIPACAO GOLDEN	3.50	2.00	7.0000	1.40	5.60
7	3	1400	PROTEINURIA	2025-11-03	UNIMED PARTICIPACAO GOLDEN	12.00	2.00	24.0000	4.80	19.20

Exercício Resolvido

LABORATORIO/postgres@PostgreSQL 17

No limit

Query History

```
21 AND p.id_plano = ep.id_plano
22 JOIN exame e
23     ON ep.id_exame_pedido = e.id_exame
24 JOIN unidade_servico us
25     ON us.id_plano = ps.id_plano
26     AND p.dt_pedido >= us.dt_inicio_vigencia
27     AND (us.dt_fim_vigencia IS NULL OR p.dt_pedido <= us.dt_fim_vigencia);
28
29
30 SELECT * FROM vw_exame_pedido_valores;
31
32 SELECT * FROM vw_exame_pedido_valores WHERE nr_pedido = 1;
```

Data Output Messages Notifications

Showing rows

	nr_pedido integer	id_exame_pedido integer	nome_exame character varying (50)	dt_pedido date	nome_plano character varying (50)	valor_ch numeric (9,2)	valor_unidade numeric (9,2)	valor_total numeric	valor_paciente numeric	valor_plano numeric
1	1	1099	HEMOGRAMA	2025-08-18	UNIMED PARTICIPACAO GOLDEN	3.00	2.00	6.0000	1.20	4.80
2	1	1500	UREIA	2025-08-18	UNIMED PARTICIPACAO GOLDEN	10.00	2.00	20.0000	4.00	16.00
3	1	1100	GLICEMIA	2025-08-18	UNIMED PARTICIPACAO GOLDEN	5.00	2.00	10.0000	2.00	8.00

Exercício Resolvido

- [1] - Faça um STORED PROCEDURE para a inserção de um novo PACIENTE. Não esqueça de implementar as validações necessárias para tanto.
- [2] - Escreva um STORED PROCEDURE com as devidas validações dos dados de entrada para a geração de um novo PEDIDO.
- [3] - Produza um STORED PROCEDURE que valide os dados de entrada e insira um novo EXAME_PEDIDO.
- [4] - Elabore um TRIGGER que impeça um PACIENTE do sexo MASCULINO solicitar um EXAME_PEDIDO de "Beta HCG" (Exame de Gravidez).

Exercício Resolvido

- [5] - Implemente uma RULE que impeça uma mulher de solicitar um exame de "PSA" (Próstata).