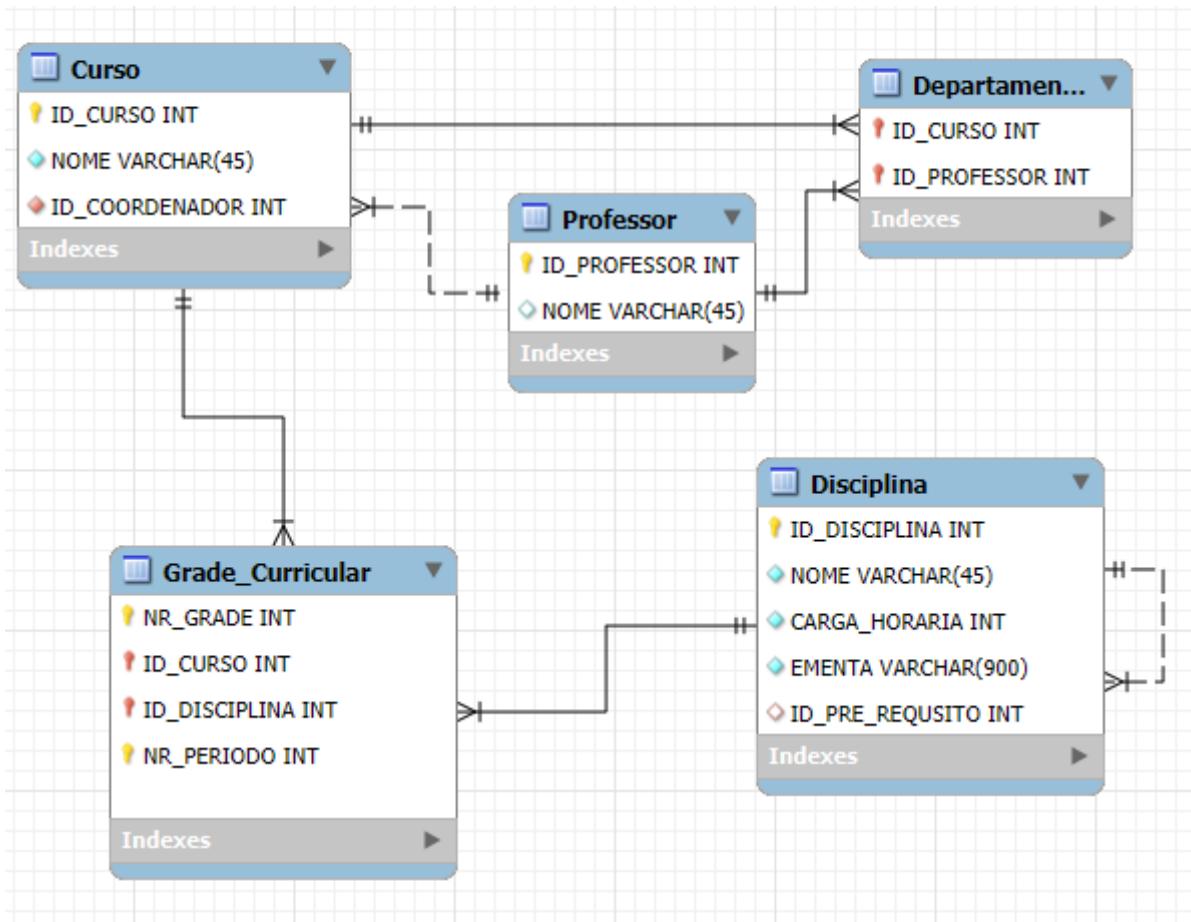


Banco de Dados 2

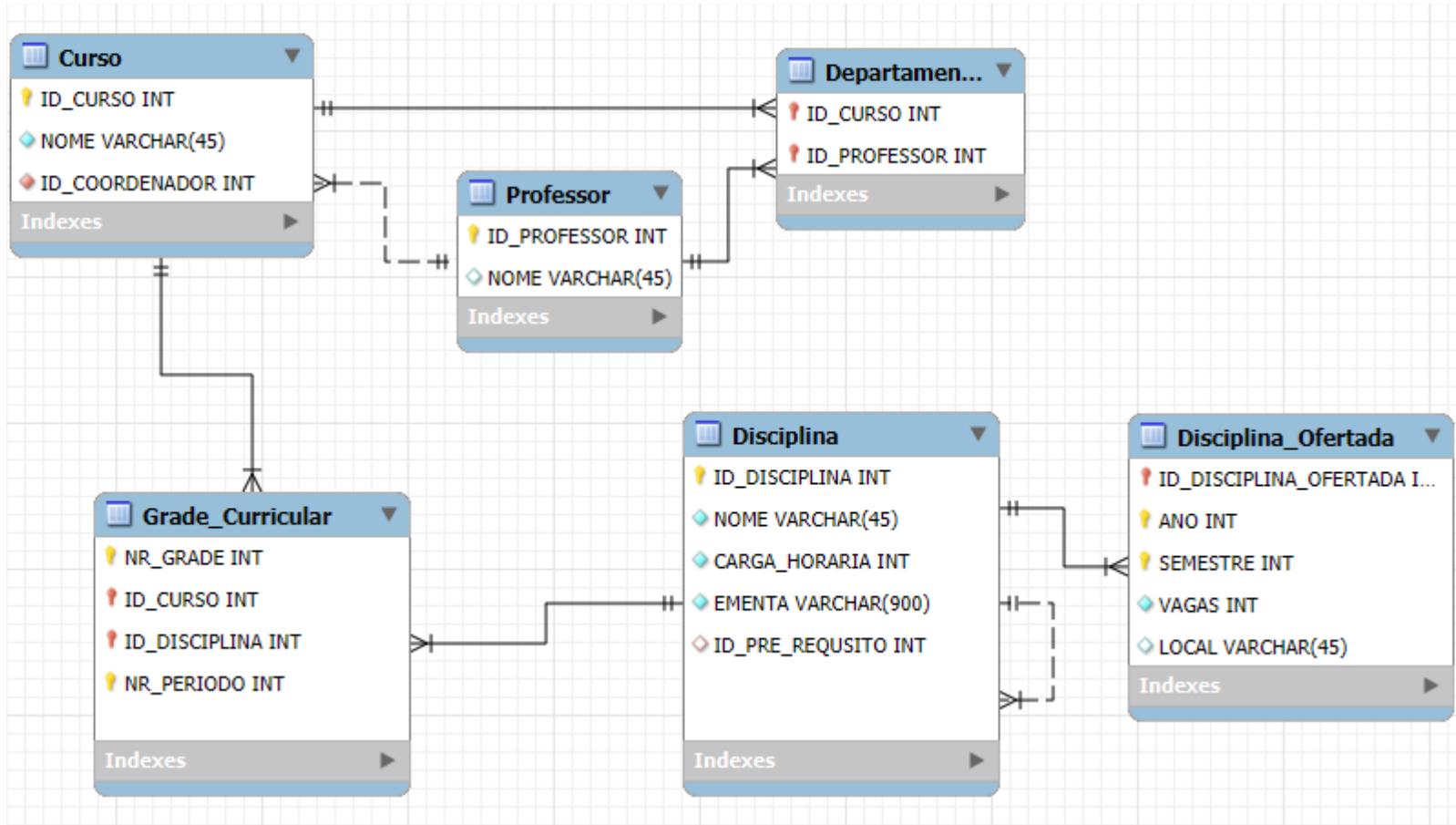
03 - Funções PLPGSQL (Parte 2) e Cursor

Modelagem



- Retornemos ao Banco de Dados **IES** (Instituição de Ensino Superior).
- Ele foi apresentado no **SLIDE 01** desta **disciplina** (**Banco de Dados 2**).
- Chegou o momento de **ampliarmos** o **modelo** que utilizaremos ao tratar de **Funções PLPGSQL**.

Ampliação do BD IES



- Adicionamos a **tabela DISCIPLINA_OFERTADA** (não confundir com a tabela **DISCIPLINA**).
- Ela descreve as disciplinas que são ofertadas em semestres específicos (ANO-SEMESTRE) como 2025-2, por exemplo.
- Ela também informa quantas VAGAS são ofertadas para os alunos se matricularem e o LOCAL onde as aulas ocorrerão.

Ampliação do BD IES

Query Query History

```
1 CREATE TABLE IF NOT EXISTS Disciplina_Oferada (
2     ID_DISCIPLINA_OFERTADA INT NOT NULL,
3     ANO INT NOT NULL,
4     SEMESTRE INT NOT NULL CHECK (SEMESTRE > 0 AND SEMESTRE < 3),
5     VAGAS INT NOT NULL,
6     LOCAL VARCHAR(45) NULL,
7
8     PRIMARY KEY (ID_DISCIPLINA_OFERTADA, ANO, SEMESTRE),
9
10    CONSTRAINT ID_DISCIPLINA_OFERTADA
11        FOREIGN KEY (ID_DISCIPLINA_OFERTADA) REFERENCES Disciplina (ID_DISCIPLINA)
12        ON DELETE RESTRICT
13        ON UPDATE NO ACTION
14 );
15
```

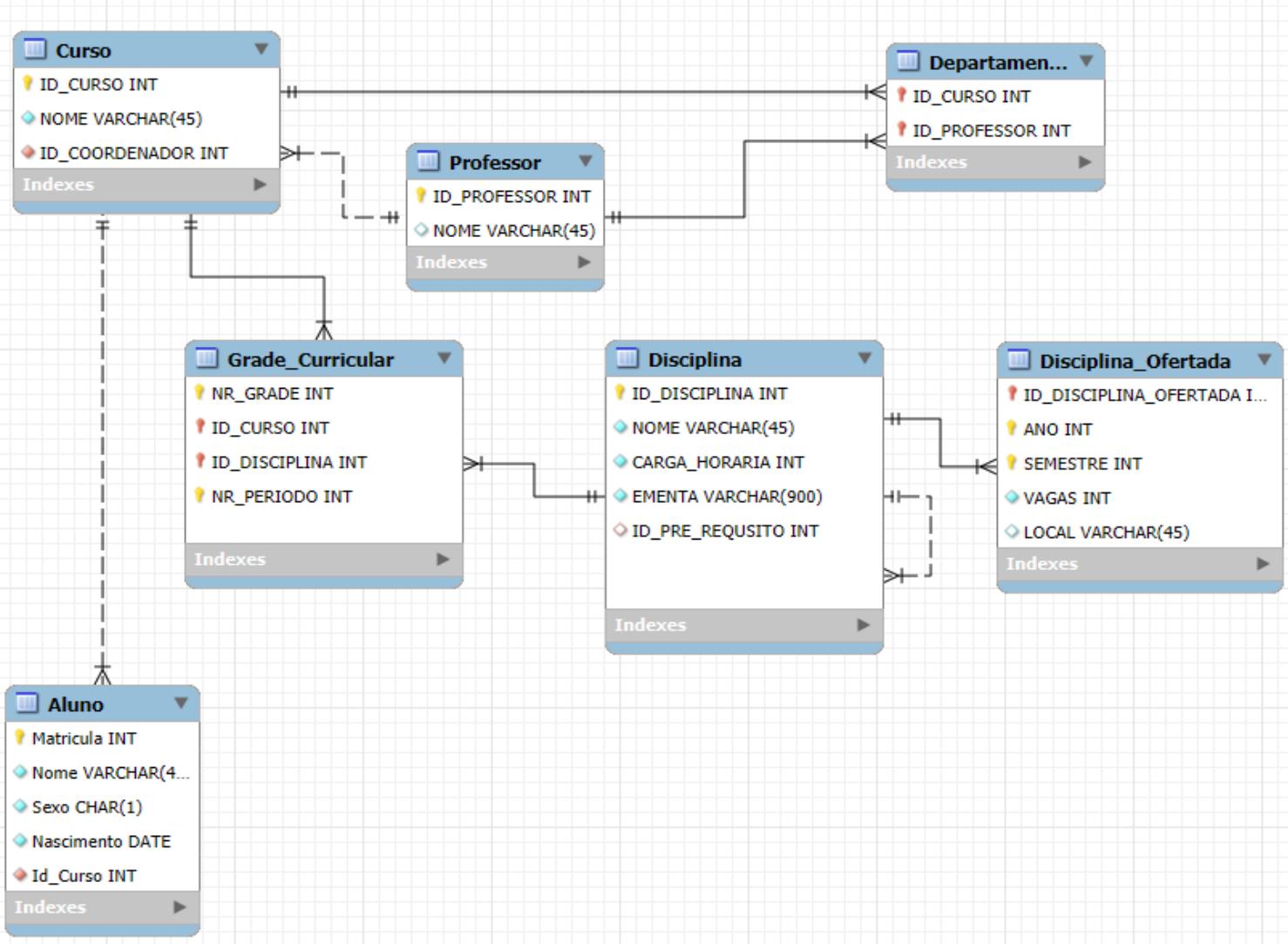
- Reparem que, para efeito de simplificação, NÃO permitimos que uma mesma DISCIPLINA seja ofertada mais de uma única vez por semestre letivo.

Ampliação do BD IES

```
17  
18 INSERT INTO DISCIPLINA_OFERTADA VALUES (1079, 2025, 1, 40,'SALA 1001'),  
19  
20 (1081, 2025, 1, 45,'SALA 8902'),  
21 (1083, 2025, 1, 30,'LAB. 05'),  
22 (1080, 2025, 2, 40,'SALA 5901'),  
23 (1084, 2025, 2, 30,'LAB. 04'),  
24 (1085, 2025, 2, 30,'LAB. 03'),  
25 (1083, 2025, 2, 30,'LAB. 05');  
26
```

- No semestre letivo de 2025-1 foram ofertadas 3 disciplinas para a matrícula dos alunos.
- No semestre de 2025-2 foram ofertadas 4 disciplinas.
- A disciplina 1083 (Programação) foi ofertada nos dois semestres.

Ampliação do BD IES



- Acrescentamos uma tabela ALUNO simplificada.
- Não há a possibilidade de um mesmo aluno estar simultaneamente matriculado em mais de um CURSO.
- Caso um mesmo aluno, após se formar em um dado CURSO, resolva se matricular em um SEGUNDO CURSO, seus dados deverão ser duplicados e receber uma nova MATRÍCULA.

Ampliação do BD IES

Query Query History

```
26
27 CREATE TABLE IF NOT EXISTS ALUNO (
28     Matricula INT NOT NULL,
29     Nome VARCHAR(45) NOT NULL,
30     Sexo CHAR(1) NOT NULL CHECK (SEXO='M' OR SEXO='F'),
31     Nascimento DATE NOT NULL,
32     Id_Curso INT NOT NULL,
33
34     PRIMARY KEY (Matricula),
35
36     CONSTRAINT Id_Curso
37         FOREIGN KEY (Id_Curso)
38         REFERENCES CURSO (ID_CURSO)
39         ON DELETE RESTRICT
40         ON UPDATE NO ACTION
41 );
42
43
```

Data Output Messages Notifications

CREATE TABLE

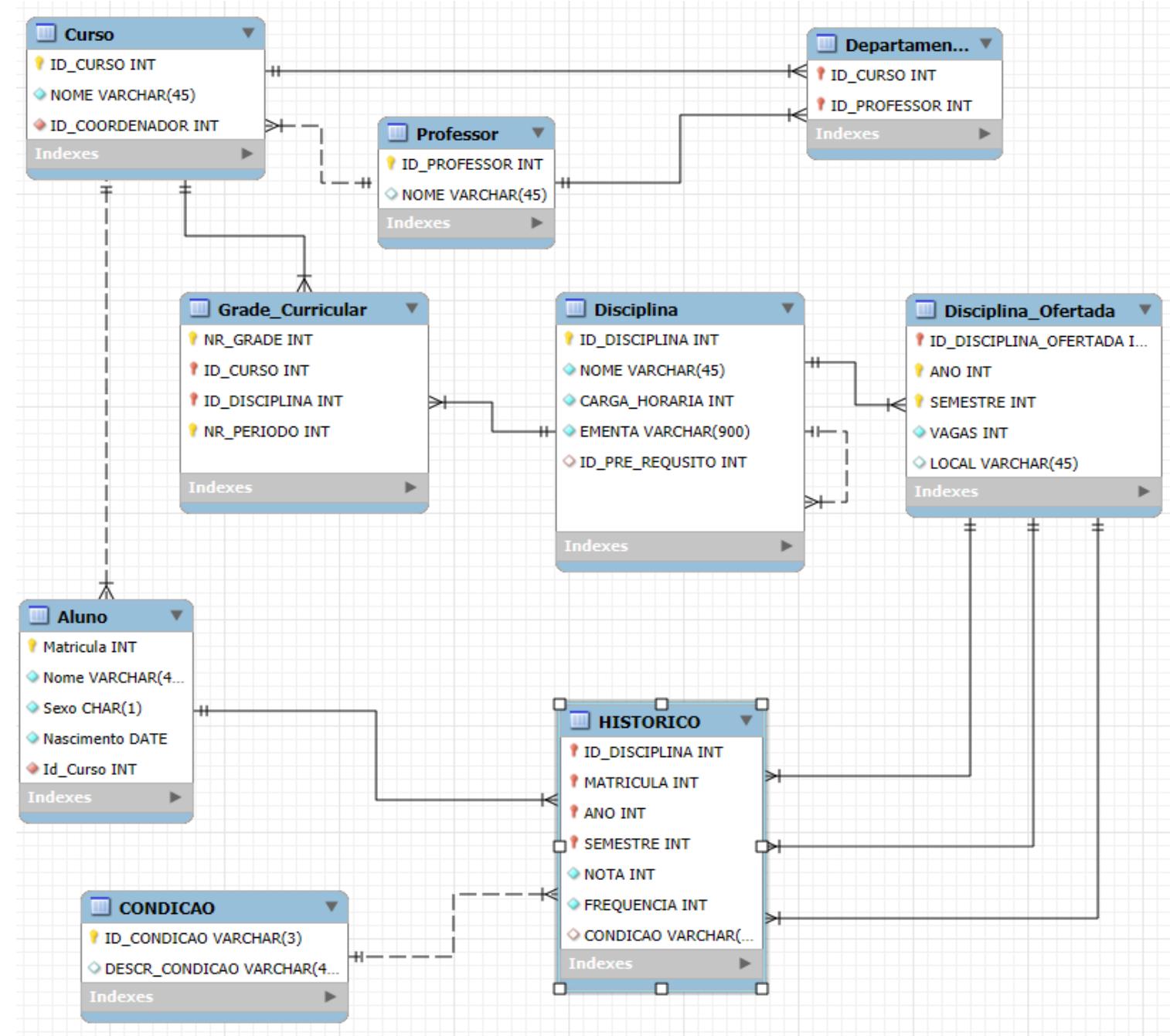
Query returned successfully in 77 msec.

```
42
43
44 INSERT INTO ALUNO VALUES
45 (2099,'FRANCISCO BUARQUE DE HOLANDA','M','1992-09-22',200),
46 (2100,'CAETANO VELOSO','M','1993-01-21',100),
47 (2101,'MILTON NASCIMENTO','M','1994-04-03',100),
48 (2102,'AGNETHA FALTSKOG','F','1995-05-01', 300),
49 (2200,'JOHN LENNON','M','1995-07-21',300),
50 (2300,'ANNY FRID LYNGSTAD','F','1996-03-25',300);
51
```

Data Output Messages Notifications

INSERT 0 6

Query returned successfully in 106 msec.



Ampliação do BD IES

51

52

```
53 CREATE TABLE IF NOT EXISTS CONDICAO (
54     ID_CONDICAO VARCHAR(3) NOT NULL,
55     DESCR_CONDICAO VARCHAR(45) NULL,
56     PRIMARY KEY (ID_CONDICAO));
```

57

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 74 msec.

57

```
58 INSERT INTO CONDICAO VALUES ('APR','APROVADO'), ('RNO','REPROVADO POR NOTA'),
59                               ('RFA','REPROVADO POR FALTA');
```

60

Data Output Messages Notifications

INSERT 0 3

Query returned successfully in 72 msec.

Query Query History

```
60
61 CREATE TABLE IF NOT EXISTS HISTORICO (
62     ID_DISCIPLINA INT NOT NULL,
63     MATRICULA INT NOT NULL,
64     ANO INT NOT NULL,
65     SEMESTRE INT NOT NULL,
66     NOTA INT NOT NULL,
67     FREQUENCIA INT NOT NULL,
68     CONDICAO VARCHAR(3) NULL,
69
70     PRIMARY KEY (ID_DISCIPLINA, MATRICULA, ANO, SEMESTRE),
71
72     CONSTRAINT ID_DISCIPLINA
73         FOREIGN KEY (ID_DISCIPLINA, ANO, SEMESTRE)
74             REFERENCES Disciplina_Oferada (ID_DISCIPLINA_OFERTADA, ANO, SEMESTRE)
75             ON DELETE RESTRICT
76             ON UPDATE NO ACTION,
77
78     CONSTRAINT MATRICULA
79         FOREIGN KEY (MATRICULA)
80             REFERENCES Aluno (Matricula)
81             ON DELETE RESTRICT
82             ON UPDATE NO ACTION,
83
84
85     CONSTRAINT CONDICAO
86         FOREIGN KEY (CONDICAO)
87             REFERENCES CONDICAO (ID_CONDICAO)
88             ON DELETE RESTRICT
89 );
90
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 148 msec.

Funções PLPGSQL

A função **PLPGSQL inserir_disciplina_ofertada()**, como o próprio nome sugere, é responsável por criar um novo registro (linha) de **DISCIPLINA_OFERTADA**.

Elá retorna um **String (TEXT)**.

Elá também efetua **consistências** para **impedir a inserção de dados equivocados**:

A coluna **SEMESTRE** somente deve aceitar **valores 1 ou 2**.

A quantidade de **VAGAS** deve ser **maior que zero**.

O **ID_DISCIPLINA** informado deve existir na tabela **DISCIPLINA**.

```
Query Query History
1 CREATE OR REPLACE FUNCTION inserir_disciplina_ofertada(
2     p_id_disciplina_ofertada INT,
3     p_ano INT,
4     p_semestre INT,
5     p_vagas INT,
6     p_local VARCHAR
7 )
8 RETURNS TEXT AS $$ 
9 DECLARE
10     v_exists INT;
11 BEGIN
12     -- Consistência 1: semestre válido
13     IF p_semestre NOT IN (1, 2) THEN
14         RETURN 'Erro: o semestre deve ser 1 ou 2.';
15     END IF;

16     -- Consistência 2: vagas deve ser positivo
17     IF p_vagas <= 0 THEN
18         RETURN 'Erro: o número de vagas deve ser positivo.';
19     END IF;

20     -- Consistência 3: verificar se a disciplina existe na tabela DISCIPLINA
21     SELECT COUNT(*) INTO v_exists
22     FROM DISCIPLINA
23     WHERE ID_DISCIPLINA = p_id_disciplina_ofertada;
24
25     IF v_exists = 0 THEN
26         RETURN 'Erro: ID_DISCIPLINA_OFERTADA não existe na tabela DISCIPLINA.';
27     END IF;
28
29
30
```

Funções PLPGSQL

- A quarta consistência procura evitar a duplicidade de registros na tabela **DISCIPLINA_OFERTADA** – algo que resultaria em uma violação de chave primária.

```
Query Query History
30
31      -- Consistência 4: checar duplicidade de oferta
32      SELECT COUNT(*) INTO v_exists
33      FROM DISCIPLINA_OFERTADA
34      WHERE ID_DISCIPLINA_OFERTADA = p_id_disciplina_ofertada
35      AND ANO = p_ano
36      AND SEMESTRE = p_semestre;
37
38      IF v_exists > 0 THEN
39          RETURN 'Erro: essa disciplina já foi ofertada nesse ano e semestre.';
40      END IF;
41
42      -- Inserção
43      INSERT INTO DISCIPLINA_OFERTADA (
44          ID_DISCIPLINA_OFERTADA,
45          ANO,
46          SEMESTRE,
47          VAGAS,
48          LOCAL
49      ) VALUES (
50          p_id_disciplina_ofertada,
51          p_ano,
52          p_semestre,
53          p_vagas,
54          p_local
55      );
56
57      RETURN 'Inserção realizada com sucesso.';
58
59  $$ LANGUAGE plpgsql;
```

Funções PLPGSQL

```
56
57     RETURN 'Inserção realizada com sucesso.';
58 END;
59 $$ LANGUAGE plpgsql;
60
61 SELECT inserir_disciplina_ofertada(1085, 2026, 1, 40, 'Bloco A - Sala 301');
62
```

Data Output Messages Notifications



	inserir_disciplina_ofertada	text	lock icon
1	Inserção realizada com sucesso.		

- Efetuando um **teste** da função **inserir_disciplina_ofertada()**.
- Inserimos a oferta da disciplina **BANCO DE DADOS (1085)** para o **semestre 2026-1**. Ela tem **40 vagas** para a matrícula de alunos e será ministrada no **Bloco A - Sala 301**.

Funções PLPGSQL

```
60
61 SELECT inserir_disciplina_ofertada(1085, 2026, 1, 40, 'Bloco A - Sala 301');
62
63 SELECT * FROM DISCIPLINA_OFERTADA;
64
```

Data Output Messages Notifications

SQL

	id_disciplina_ofertada [PK] integer	ano [PK] integer	semestre [PK] integer	vagas integer	local character varying (45)
1	1079	2025	1	40	SALA 1001
2	1081	2025	1	45	SALA 8902
3	1083	2025	1	30	LAB. 05
4	1080	2025	2	40	SALA 5901
5	1084	2025	2	30	LAB. 04
6	1085	2025	2	30	LAB. 03
7	1083	2025	2	30	LAB. 05
8	1085	2026	1	40	Bloco A - Sala 301

- A comprovação de que a inserção foi bem sucedida.
- Corresponde ao oitavo registro da tabela.

Funções PLPGSQL

```
60
61  SELECT inserir_disciplina_ofertada(1085, 2026, 1, 40, 'Bloco A - Sala 301');
62
63  SELECT * FROM DISCIPLINA_OFERTADA;
64
65  SELECT inserir_disciplina_ofertada(1085, 2026, 1, 40, 'Bloco A - Sala 301');
66
67
```

Data Output Messages Notifications

	inserir_disciplina_ofertada	
1	text	Erro: essa disciplina já foi ofertada nesse ano e semestre.

- Repare que, ao tentarmos replicar o mesmo registro geramos uma mensagem de erro (**consistência 4: checagem de duplicidade de ofertas**).
- O registro duplicado não foi inserido.

Funções PLPGSQL

```
64  
65 SELECT inserir_disciplina_ofertada(1085, 2026, 1, 40, 'Bloco A - Sala 301');  
66  
67 SELECT inserir_disciplina_ofertada(2099, 2026, 1, 40, 'Bloco C - Sala 101');  
68  
69
```

Data Output Messages Notifications



	inserir_disciplina_ofertada	lock
1	Erro: ID_DISCIPLINA_OFERTADA não existe na tabela DISCIPLINA.	

- Aqui geramos um novo erro: a DISCIPLINA com ID_DISCIPLINA = 2099 não existe na tabela DISCIPLINA.

Funções PLPGSQL

```
Query Query History
69 CREATE OR REPLACE FUNCTION buscar_disciplinas_ofertadas(
70     p_ano INT,
71     p_semestre INT
72 )
73 RETURNS TABLE (
74     id_disciplina_ofertada INT,
75     ano INT,
76     semestre INT,
77     vagas INT,
78     local VARCHAR
79 ) AS $$*
80 BEGIN
81     -- Validação do semestre
82     IF p_semestre NOT IN (1, 2) THEN
83         RAISE EXCEPTION 'Semestre inválido. Use 1 ou 2.';
84     END IF;
85
86     -- Consulta com aliases para evitar ambiguidade
87     RETURN QUERY
88     SELECT
89         d.ID_DISCIPLINA_OFERTADA,
90         d.ANO,
91         d.SEMESTRE,
92         d.VAGAS,
93         d.LOCAL
94     FROM
95         DISCIPLINA_OFERTADA d
96     WHERE
```

A função PLPGSQL `buscar_disciplinas_ofertadas()` executa uma consulta SQL com base nos parâmetros repassados pelo usuário (ano e semestre).

As palavras reservadas `RETURN QUERY` indica que o código SQL apresentado em seguida irá gerar a saída esperada (TABLE).

Funções PLPGSQL

- Na **chamada** para a **execução** da função delimitamos a busca para o **semestre 2025-2**.

```
96      WHERE
97          d.ANO = p_ano AND d.SEMESTRE = p_semestre;
98
99      END;
100
101
102      $$ LANGUAGE plpgsql;
103
104
105      SELECT * FROM buscar_disciplinas_ofertadas(2025, 2);
```

Data Output Messages Notifications

	id_disciplina_ofertada integer	ano integer	semestre integer	vagas integer	local character varying
1	1080	2025	2	40	SALA 5901
2	1083	2025	2	30	LAB. 05
3	1084	2025	2	30	LAB. 04
4	1085	2025	2	30	LAB. 03

Funções PLPGSQL

- Melhorando o código da função PLPGSQL.
- Foi acrescentado um JOIN na consulta (QUERY) para trazer o nome da Disciplina ofertada.

```
Query Query History
103
104 DROP FUNCTION IF EXISTS buscar_disciplinas_ofertadas(INT, INT);
105
106 CREATE OR REPLACE FUNCTION buscar_disciplinas_ofertadas(
107     p_ano INT,
108     p_semestre INT
109 )
110 RETURNS TABLE (
111     id_disciplina_ofertada INT,
112     ano INT,
113     semestre INT,
114     vagas INT,
115     local VARCHAR,
116     nome_disciplina VARCHAR,
117     carga_horaria INT,
118     ementa VARCHAR
119 ) AS $$
120 BEGIN
121     -- Validação do semestre
122     IF p_semestre NOT IN (1, 2) THEN
123         RAISE EXCEPTION 'Semestre inválido. Use 1 ou 2.';
124     END IF;
125 
```

Funções PLPGSQL

The screenshot shows a PostgreSQL query editor interface. The main area displays a PL/pgSQL function definition. The code is color-coded for syntax highlighting, with numbers on the left indicating line numbers from 125 to 145.

```
125
126    -- Consulta com JOIN para trazer detalhes da disciplina
127    RETURN QUERY
128    SELECT
129        d.ID_DISCIPLINA_OFERTADA,
130        d.ANO,
131        d.SEMESTRE,
132        d.VAGAS,
133        d.LOCAL,
134        dis.NOME,
135        dis.CARGA_HORARIA,
136        dis.EMENTA
137    FROM
138        DISCIPLINA_OFERTADA d
139    JOIN
140        DISCIPLINA dis ON dis.ID_DISCIPLINA = d.ID_DISCIPLINA_OFERTADA
141    WHERE
142        d.ANO = p_ano AND d.SEMESTRE = p_semestre;
143    END;
144    $$ LANGUAGE plpgsql;
145
```

Below the code, there are tabs for "Data Output", "Messages", and "Notifications". The "Messages" tab is selected, showing the message "CREATE FUNCTION". At the bottom, it says "Query returned successfully in 166 msec."

Funções PLPGSQL

```
145  
146  SELECT * FROM buscar_disciplinas_ofertadas(2025, 2);  
147
```

Data Output Messages Notifications

Showing rows: 1 to 4  Page No: 1       

	id_disciplina_ofertada integer	ano integer	semestre integer	vagas integer	local character varying	nome_disciplina character varying	carga_horaria integer	ementa character varying	
1	1080	2025	2	40	SALA 5901	CALCULO 2	120	CALCULO NUMERICO.	
2	1083	2025	2	30	LAB. 05	PROGRAMACAO	120	COMANDOS CONDICIONAIS, COMANDOS DE REPETICAO E PROCEDIMENTOS RECURSIVOS.	
3	1084	2025	2	30	LAB. 04	ESTRUTURA DE DADOS	120	LISTAS ENCADEADAS. FILAS. PILHAS. ARVORES. GRAFOS	
4	1085	2025	2	30	LAB. 03	BANCO DE DADOS	90	MODELAGEM. SQL.	

Funções PLPGSQL

- Mas, cuidado!
- Se a escrita desta função for alterada, no sentido de ser mais simplificada que a anterior, a chamada da mesma automaticamente se tornará mais complicada.
- Vejamos um exemplo a seguir:

Funções PLPGSQL

Eis uma versão simplificada da função plpgsql
`buscar_disciplinas_ofertadas()`.

```
Query Query History
1 DROP FUNCTION BUSCAR_DISCIPLINAS_OFERTADAS(p_ano INT,
2                                         p_semestre INT);
3
4
5
6
7 )
8 RETURNS SETOF RECORD AS $$

9 BEGIN
10    -- Validação do semestre
11    IF p_semestre NOT IN (1, 2) THEN
12        RAISE EXCEPTION 'Semestre inválido. Use 1 ou 2.';
13    END IF;
14
15    RETURN QUERY
16    SELECT
17        d.ID_DISCIPLINA_OFERTADA,
18        d.ANO,
19        d.SEMESTRE,
20        d.VAGAS,
21        d.LOCAL,
22        dis.NOME AS nome_disciplina,
23        dis.CARGA_HORARIA,
24        dis.EMENTA
25    FROM
26        DISCIPLINA_OFERTADA d
27    JOIN
28        DISCIPLINA dis ON dis.ID_DISCIPLINA = d.ID_DISCIPLINA_OFERTADA
29    WHERE
30        d.ANO = p_ano AND d.SEMESTRE = p_semestre;
31    END;
32 $$ LANGUAGE plpgsql;
```

Data Output Messages Notifications

CREATE FUNCTION

Query returned successfully in 92 msec.

Funções PLPGSQL

Código de Versão mais Complexa (usando RETURNS TABLE com descrição detalhada das colunas).

```
48
49 CREATE OR REPLACE FUNCTION buscar_disciplinas_ofertadas(
50     p_ano INT,
51     p_semestre INT
52 )
53 RETURNS TABLE (
54     id_disciplina_ofertada INT,
55     ano INT,
56     semestre INT,
57     vagas INT,
58     local VARCHAR,
59     nome_disciplina VARCHAR,
60     carga_horaria INT,
61     ementa VARCHAR
62 ) AS $$
```

63 **BEGIN**

```
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89 SELECT * FROM buscar_disciplinas_ofertadas(2025, 2);
```

Código de Versão mais Simples (usando RETURNS SETOF RECORD sem descrição detalhada das colunas).

```
3
4 CREATE OR REPLACE FUNCTION buscar_disciplinas_ofertadas(
5     p_ano INT,
6     p_semestre INT
7 )
8 RETURNS SETOF RECORD AS $$
9 BEGIN

34
35     SELECT * FROM buscar_disciplinas_ofertadas(2025, 2)
36     AS resultado (
37         id_disciplina_ofertada INT,
38         ano INT,
39         semestre INT,
40         vagas INT,
41         local VARCHAR,
42         nome_disciplina VARCHAR,
43         carga_horaria INT,
44         ementa VARCHAR
45     );
46
```

Funções PLPGSQL

```
34
35 SELECT * FROM buscar_disciplinas_ofertadas(2025, 2)
36 AS resultado (
37     id_disciplina_ofertada INT,
38     ano INT,
39     semestre INT,
40     vagas INT,
41     local VARCHAR,
42     nome_disciplina VARCHAR,
43     carga_horaria INT,
44     ementa VARCHAR
45 );
```

Data Output Messages Notifications

The screenshot shows a PostgreSQL database interface with a query editor and a results table. The query editor contains a PL/pgSQL function definition. The results table displays four rows of data from the function's output.

	id_disciplina_ofertada	ano	semestre	vagas	local	nome_disciplina	carga_horaria	ementa
1	1080	2025	2	40	SALA 5901	CALCULO 2	120	CALCULO NUMERICO.
2	1083	2025	2	30	LAB. 05	PROGRAMACAO	120	COMANDOS CONDICIONAIS, COMANDOS DE REPETICAO E PROCEDIMENTOS RECURSIVOS.
3	1084	2025	2	30	LAB. 04	ESTRUTURA DE DADOS	120	LISTAS ENCADEADAS. FILAS. PILHAS. ARVORES. GRAFOS
4	1085	2025	2	30	LAB. 03	BANCO DE DADOS	90	MODELAGEM. SQL.

Ou seja, quanto mais simplificado o código mais complexa a chamada e vice versa.

Funções PLPGSQL

- Outra versão alternativa do mesmo código.
- Uma versão simplificada também.
- Em vez de **RETURNS TABLE** ou de **RETURNS SETOF RECORD** agora temos simplesmente **RETURNS SETOF DISCIPLINA_OFERTADA** (nome da tabela).
- A Query não deve ter JOIN.

```
Query Query History
1 DROP FUNCTION BUSCAR_DISCIPLINAS_OFERTADAS(p_ano INT,
2   p_semestre INT);
3
4 CREATE OR REPLACE FUNCTION buscar_disciplinas_ofertadas(
5   p_ano INT,
6   p_semestre INT
7 )
8 RETURNS SETOF DISCIPLINA_OFERTADA AS $$ BEGIN
9   -- Validação do semestre
10  IF p_semestre NOT IN (1, 2) THEN
11    RAISE EXCEPTION 'Semestre inválido. Use 1 ou 2.';
12  END IF;
13
14  RETURN QUERY
15  SELECT *
16  FROM
17    DISCIPLINA_OFERTADA
18  WHERE
19    ANO = p_ano AND SEMESTRE = p_semestre;
20
21 END;
22 $$ LANGUAGE plpgsql;
23
```

Data Output Messages Notifications

CREATE FUNCTION

Query returned successfully in 79 msec.

Funções PLPGSQL

Repare que não foi necessário detalhar as colunas na chamada da função.

Assim como não foi necessário detalhar essas colunas no código da função.

Todavia, o uso de **RETURNS SETOF DISCIPLINA_OFERTADA** em vez de **RETURNS SETOF RECORD** implica na impossibilidade de usar **JOIN** ou um **FROM** com mais de uma tabela na **QUERY** que gera a saída da função.

```
3
4 CREATE OR REPLACE FUNCTION buscar_disciplinas_ofertadas(
5   p_ano INT,
6   p_semestre INT
7 )
8 RETURNS SETOF DISCIPLINA_OFERTADA AS $$ 
9 BEGIN
10   -- Validação do semestre
11   IF p_semestre NOT IN (1, 2) THEN
12     RAISE EXCEPTION 'Semestre inválido. Use 1 ou 2.';
13   END IF;
14
15   RETURN QUERY
16   SELECT id_disciplina_ofertada, ano, semestre, vagas, local
17   FROM
18     DISCIPLINA_OFERTADA
19   WHERE
20     ANO = p_ano AND SEMESTRE = p_semestre;
21 END;
22 $$ LANGUAGE plpgsql;
23
24 SELECT * FROM buscar_disciplinas_ofertadas(2025, 2);
```

Data Output Messages Notifications

	id_disciplina_ofertada integer	ano integer	semestre integer	vagas integer	local character varying (45)
1	1080	2025	2	40	SALA 5901
2	1083	2025	2	30	LAB. 05
3	1084	2025	2	30	LAB. 04
4	1085	2025	2	30	LAB. 03

Cursor

- Um **cursor** em PostgreSQL é um **objeto de banco de dados** que permite **navegar linha a linha** (linha por linha) sobre o resultado de uma **consulta SQL**.
- É especialmente útil quando você precisa **processar registros individualmente** em vez de todos de uma vez — por exemplo, dentro de um **laço LOOP** em uma **função PLPGSQL**.
- É especialmente vantajoso quando:
 - Precisa-se de **controle total sobre o fluxo de leitura** dos dados (como pular registros, voltar, etc).
 - Executa-se **lógica condicional** a cada linha de resultado.
 - Utiliza-se **transações complexas** que exigem interação parcial com o conjunto de dados.

Cursor

- Declarando Variáveis de Cursor:
- Todo o acesso a **cursores** em **PL/pgSQL** passa por **variáveis de cursor**, que são sempre do **tipo de dado especial refcursor**.
- Uma maneira de criar uma **variável de cursor** é simplesmente declará-la como **uma variável do tipo refcursor**. Outra maneira é usar a **sintaxe de declaração de cursor**, que em geral é:

```
name [ [ NO ] SCROLL ] CURSOR [ ( arguments ) ] FOR query;
```

Cursor

```
DECLARE  
    curs1 refcursor;  
    curs2 CURSOR FOR SELECT * FROM tenk1;  
    curs3 CURSOR (key integer) FOR SELECT * FROM tenk1 WHERE unique1 = key;
```

- Todas essas três variáveis têm o **tipo de dado refcursor**, mas a **primeira pode ser usada com qualquer consulta**, enquanto a **segunda tem uma consulta totalmente especificada já vinculada a ela**, e a **última tem uma consulta parametrizada vinculada a ela**.
- **Key** será substituído por um valor de parâmetro inteiro quando o cursor for aberto.
- A variável **curs1** é considerada **não vinculada**, pois não está vinculada a nenhuma consulta específica.

Cursor

- A opção SCROLL não pode ser usada quando a consulta do cursor usa FOR UPDATE/SHARE. Além disso, é melhor usá-la NO SCROLL com uma consulta que envolva funções voláteis.
- A implementação de SCROLL pressupõe que a releitura da saída da consulta fornecerá resultados consistentes, o que uma função volátil pode não fazer.
- A opção **SCROLL** ou **NO SCROLL** na **declaração de cursores** no **PostgreSQL** define **como o cursor pode ser percorrido (navegado)**:

Cursor

SCROLL Permite movimentar-se **em qualquer direção** dentro do cursor. Você pode:

- Ler os dados para frente (como **FETCH NEXT**)
- Ler para trás (como **FETCH PRIOR**)
- Saltar para posições específicas (como **FETCH ABSOLUTE 5** ou **FETCH RELATIVE -2**).

• EXEMPLO:

DECLARE

cur SCROLL CURSOR FOR SELECT * FROM minha_tabela;

Cursor

- **NO SCROLL** Permite apenas leitura **sequencial para frente** (um registro após o outro):
 - Você **não pode voltar** ao registro anterior
 - Não pode saltar para posições específicas
- **EXEMPLO:**

DECLARE

cur NO SCROLL CURSOR FOR SELECT * FROM minha_tabela;

Cursor Implícito

The screenshot shows a PostgreSQL query editor interface. The top bar has tabs for 'Query' (selected) and 'Query History'. The main area contains a code snippet:

```
1 DO $$  
2 DECLARE  
3     aluno_nome TEXT;  
4 BEGIN  
5     FOR aluno_nome IN SELECT nome FROM aluno LOOP  
6         RAISE NOTICE 'Aluno: %', aluno_nome;  
7     END LOOP;  
8 END;  
9 $$;  
10
```

Below the code, there are tabs for 'Data Output' (selected), 'Messages', and 'Notifications'. The 'Data Output' tab displays the results of the query:

NOTA: Aluno: FRANCISCO BUARQUE DE HOLANDA
NOTA: Aluno: CAETANO VELOSO
NOTA: Aluno: MILTON NASCIMENTO
NOTA: Aluno: AGNETHA FALTSKOG
NOTA: Aluno: JOHN LENNON
NOTA: Aluno: ANNY FRID LYNGSTAD
DO

At the bottom, a message says 'Query returned successfully in 86 msec.'

- Eis um **exemplo** de **Cursor Implícito** usando o comando de repetição **FOR**.
- Aqui, o **cursor** é **implícito** — o **PostgreSQL** **gerencia** **automaticamente**.
- Um **cursor implícito** é criado automaticamente pelo PostgreSQL **dentro de um laço FOR em PL/pgSQL**. O programador **não precisa declarar, abrir ou fechar o cursor manualmente**.
- **Não pode ser reutilizado ou controlado fora do FOR.**

Cursor Explícito

- Aqui, o **cursor** é **declarado** com **CURSOR FOR**.
- O **programador** **controla** **explicitamente** cada detalhe da **navegação no cursor**:
 - OPEN;
 - FETCH;
 - CLOSE;

```
Query  Query History
1  DO $$ 
2  DECLARE
3      cur_alunos CURSOR FOR SELECT matricula, nome FROM aluno;
4      v_id INTEGER;
5      v_nome TEXT;
6  BEGIN
7      OPEN cur_alunos;
8
9  LOOP
10     FETCH cur_alunos INTO v_id, v_nome;
11     EXIT WHEN NOT FOUND;
12
13     RAISE NOTICE 'ID: %, Nome: %', v_id, v_nome;
14 END LOOP;
15
16     CLOSE cur_alunos;
17 END;
18 $$;
19
```

Data Output Messages Notifications

```
NOTA: ID: 2099, Nome: FRANCISCO BUARQUE DE HOLANDA
NOTA: ID: 2100, Nome: CAETANO VELOSO
NOTA: ID: 2101, Nome: MILTON NASCIMENTO
NOTA: ID: 2102, Nome: AGNETHA FALTSKOG
NOTA: ID: 2200, Nome: JOHN LENNON
NOTA: ID: 2300, Nome: ANNY FRID LYNGSTAD
DO
```

Query returned successfully in 78 msec.

Cursor Explícito

- Agora utilizamos um **CURSOR explícitamente** dentro de uma **Função PLPGSQL**.
- Além de explícito o referido **CURSOR** é dinâmico: utiliza **p_id** como **parâmetro**.

The screenshot shows a PostgreSQL query editor interface. The top navigation bar has tabs for 'Query' and 'Query History', with 'Query' currently selected. The main area contains the PL/pgSQL code for creating a function:

```
1 CREATE OR REPLACE FUNCTION listar_alunos_por_curso(p_curso_id INT)
2 RETURNS VOID AS $$
3 DECLARE
4     cur CURSOR(p_id INT) FOR
5         SELECT nome FROM aluno WHERE id_curso = p_id;
6     nome_aluno TEXT;
7 BEGIN
8     OPEN cur(p_curso_id);
9     LOOP
10        FETCH cur INTO nome_aluno;
11        EXIT WHEN NOT FOUND;
12        RAISE NOTICE 'Aluno: %', nome_aluno;
13    END LOOP;
14    CLOSE cur;
15 END;
16 $$ LANGUAGE plpgsql;
```

Below the code, there are three tabs: 'Data Output', 'Messages', and 'Notifications', with 'Messages' being the active tab. The message area displays the output of the function creation command:

CREATE FUNCTION

Query returned successfully in 97 msec.

Cursor Explícito

```
23  
24 select * from aluno;  
25
```

Data Output Messages Notifications

SQL

	matricula [PK] integer	nome character varying (45)	sexo character (1)	nascimento date	id_curso integer
1	2099	FRANCISCO BUARQUE DE HOLANDA	M	1992-09-22	200
2	2100	CAETANO VELOSO	M	1993-01-21	100
3	2101	MILTON NASCIMENTO	M	1994-04-03	100
4	2102	AGNETHA FALTSKOG	F	1995-05-01	300
5	2200	JOHN LENNON	M	1995-07-21	300
6	2300	ANNY FRID LYNGSTAD	F	1996-03-25	300

Query Query History

```
1 CREATE OR REPLACE FUNCTION listar_alunos_por_curso(p_curso_id INT)  
2 RETURNS VOID AS $$  
3 DECLARE  
4     cur CURSOR(p_id INT) FOR  
5         SELECT nome FROM aluno WHERE id_curso = p_id;  
6     nome_aluno TEXT;  
7 BEGIN  
8     OPEN cur(p_curso_id);  
9     LOOP  
10        FETCH cur INTO nome_aluno;  
11        EXIT WHEN NOT FOUND;  
12        RAISE NOTICE 'Aluno: %', nome_aluno;  
13    END LOOP;  
14    CLOSE cur;  
15 END;  
16 $$ LANGUAGE plpgsql;  
17  
18 -- Cursor parametrizado, que recebe um argumento ao ser aberto.  
19  
20 SELECT listar_alunos_por_curso(300);  
21
```

Data Output Messages Notifications

SQL

	listar_alunos_por_curso	locked
1		

Cursor Explícito

- Conforme a aba **Data Output** **nenhum nome dos três alunos matriculados no curso 300 apareceu.**
- Tudo bem: a função retorna **VOID** (ou seja, não retorna nada).
- Clique na aba **Messages** para ver o que foi descarregado de nosso **cursor (cur)**.

Query Query History

```
1 ✓ CREATE OR REPLACE FUNCTION listar_alunos_por_curso(p_curso_id INT)
2   RETURNS VOID AS $$ 
3   DECLARE
4     cur CURSOR(p_id INT) FOR
5       SELECT nome FROM aluno WHERE id_curso = p_id;
6       nome_aluno TEXT;
7   BEGIN
8     OPEN cur(p_curso_id);
9   LOOP
10    FETCH cur INTO nome_aluno;
11    EXIT WHEN NOT FOUND;
12    RAISE NOTICE 'Aluno: %', nome_aluno;
13  END LOOP;
14  CLOSE cur;
15 END;
16 $$ LANGUAGE plpgsql;
17
18 -- Cursor parametrizado, que recebe um argumento ao ser aberto.
19
20 SELECT listar_alunos_por_curso(300);
21
```

Data Output Messages Notifications

NOTA: Aluno: AGNETHA FALTSKOG
NOTA: Aluno: JOHN LENNON
NOTA: Aluno: ANNY FRID LYNGSTAD

Successfully run. Total query runtime: 76 msec.
1 rows affected.

Cursor com RETORNO para cliente externo

The screenshot shows a PostgreSQL query editor interface with two panes. The left pane displays the SQL code for creating a function named 'obter_alunos_cursor'. The right pane shows the results of executing this function, which returns a cursor to the client.

Query History:

```
1 CREATE OR REPLACE FUNCTION obter_alunos_cursor()
2 RETURNS refcursor AS $$ 
3 DECLARE
4     ref refcursor := 'alunos_cursor';
5 BEGIN
6     OPEN ref FOR SELECT * FROM aluno;
7     RETURN ref;
8 END;
9 $$ LANGUAGE plpgsql;
```

Data Output:

	matricula [PK] integer	nome character varying (45)	sexo character (1)	nascimento date	id_curso integer
1	2099	FRANCISCO BUARQUE DE HOLANDA	M	1992-09-22	200
2	2100	CAETANO VELOSO	M	1993-01-21	100
3	2101	MILTON NASCIMENTO	M	1994-04-03	100
4	2102	AGNETHA FALTSKOG	F	1995-05-01	300
5	2200	JOHN LENNON	M	1995-07-21	300
6	2300	ANNY FRID LYNGSTAD	F	1996-03-25	300

Cursor com RETORNO para cliente externo

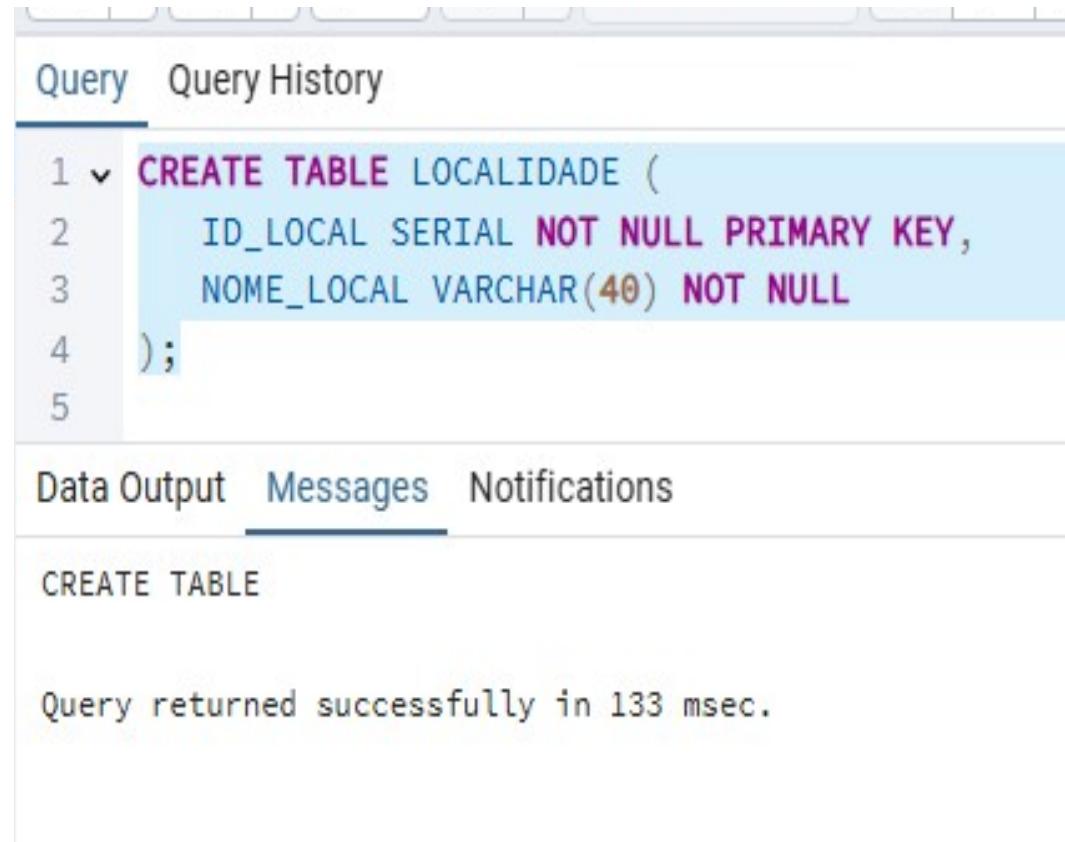
```
11  
12  
13 BEGIN;  
14  
15 -- Chama a função e abre o cursor  
16 SELECT obter_alunos_cursor();  
17  
18 -- Usa o nome retornado (literalmente 'alunos_cursor') para buscar os dados  
19 FETCH ALL FROM alunos_cursor;  
20  
21 COMMIT;  
22
```

Data Output Messages Notifications

COMMIT

Query returned successfully in 108 msec.

Criando o Banco de Dados Passagens



The screenshot shows a database query editor interface. The top navigation bar has tabs for "Query" and "Query History", with "Query" being the active tab. Below the tabs, there is a dropdown menu with the number "1" and a "▼" symbol, followed by a SQL code block. The code is a CREATE TABLE statement for a table named "LOCALIDADE". The table has two columns: "ID_LOCAL" (type SERIAL, NOT NULL, and PRIMARY KEY) and "NOME_LOCAL" (type VARCHAR(40), NOT NULL). The code ends with a closing parenthesis and a semicolon. There are five numbered lines below the code, corresponding to each line of the SQL statement. The bottom section of the editor shows tabs for "Data Output", "Messages", and "Notifications", with "Messages" being the active tab. The message area displays the text "CREATE TABLE" and "Query returned successfully in 133 msec.".

```
1 ▼ CREATE TABLE LOCALIDADE (
2     ID_LOCAL SERIAL NOT NULL PRIMARY KEY,
3     NOME_LOCAL VARCHAR(40) NOT NULL
4 );
5
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 133 msec.

Criando o Banco de Dados Passagens

Query Query History

```
1 ✓ CREATE TABLE LOCALIDADE (
2     ID_LOCAL SERIAL NOT NULL PRIMARY KEY,
3     NOME_LOCAL VARCHAR(40) NOT NULL
4 );
5
6 ✓ INSERT INTO LOCALIDADE (NOME_LOCAL) VALUES ('PEDRA AZUL'),('MARECHAL FLORIANO'),('DOMINGOS MARTINS'),
7 ('GUARAPARI'),('ANCHIETA'),('VITORIA'),('VILA VELHA'),
8 ('CARIACICA'),('VIANA'),('VENDA NOVA DO IMIGRANTE'),
9 ('SERRA'),('FUNDAO'),('IBIRACU'),('JOAO NEIVA'),('ARACRUZ'),
10 ('COLATINA'),('BAUNILHA'),('PENDANGA'),('MARILANDIA'),
11 ('BAIXO GUANDU'), ('LINHARES'),('SANTA TERESA'),('SAO MATEUS'),
12 ('JAGUARE'),('RIO BANANAL'),('SOORETAMA'),('ACIOLI'),('CAVALINHO'),
13 ('BOAPABA'),('ITAPINA'),('NOVA ALMEIDA'),('PRAIA GRANDE'),('TIMBUI'),
14 ('BARRA DO TRIUNFO'),('SAO ROQUE DO CANAA');
15
```

Data Output Messages Notifications

INSERT 0 35

Query returned successfully in 116 msec.

Criando o Banco de Dados Passagens

The screenshot shows a database query editor interface. The top navigation bar has tabs for "Query" and "Query History", with "Query" being the active tab. The main area displays a SQL script for creating a table named "LINHA". The code is numbered from 17 to 27. Lines 18 through 26 define the table structure and its foreign key constraints. Line 27 concludes the statement with a semicolon. Below the code, there are three tabs: "Data Output", "Messages" (which is active), and "Notifications". Under "Messages", the text "CREATE TABLE" is shown. At the bottom, a message indicates "Query returned successfully in 86 msec."

```
17
18 CREATE TABLE LINHA (
19     ID_LINHA INTEGER NOT NULL PRIMARY KEY,
20     ID_LOCAL_ORIGEM INTEGER NOT NULL,
21     ID_LOCAL_DESTINO INTEGER NOT NULL,
22
23     FOREIGN KEY (ID_LOCAL_ORIGEM) REFERENCES LOCALIDADE (ID_LOCAL) ON DELETE RESTRICT,
24
25     FOREIGN KEY (ID_LOCAL_DESTINO) REFERENCES LOCALIDADE (ID_LOCAL) ON DELETE RESTRICT
26 );
27
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 86 msec.

Criando o Banco de Dados Passagens

```
27
28 -- CORREÇÃO: A coluna ID_CLASSE (chave estrangeira) foi retirada da tabela LINHA e realocada na tabela VIAGEM.
29
30 INSERT INTO LINHA VALUES (100,35,16),(110,16,35),(120,35,22);
31
32
```

Data Output Messages Notifications

INSERT 0 3

Query returned successfully in 101 msec.

Criando o Banco de Dados Passagens

The screenshot shows a database query editor interface. At the top, there are tabs for "Query" and "Query History", with "Query" being the active tab. Below the tabs is a code editor area containing SQL code. The code is numbered from 32 to 45. Lines 33 through 45 define the structure of the "TRECHO" table, including columns for ID_LINHA, ID_TRECHO, ID_LOCAL, and KM, with primary key and foreign key constraints referencing "LINHA" and "LOCALIDADE" tables respectively. Line 45 ends the table definition with a semicolon. Below the code editor are three tabs: "Data Output", "Messages", and "Notifications", with "Messages" being the active tab. The message area displays the text "CREATE TABLE". At the bottom, a status message reads "Query returned successfully in 99 msec."

```
32
33 CREATE TABLE TRECHO (
34     ID_LINHA INTEGER NOT NULL,
35     ID_TRECHO INTEGER NOT NULL,
36     ID_LOCAL INTEGER NOT NULL,
37     KM INTEGER NOT NULL,
38
39     PRIMARY KEY (ID_LINHA, ID_TRECHO),
40
41     FOREIGN KEY (ID_LINHA) REFERENCES LINHA (ID_LINHA),
42
43     FOREIGN KEY (ID_LOCAL) REFERENCES LOCALIDADE (ID_LOCAL)
44 );
45
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 99 msec.

Criando o Banco de Dados Passagens

Query Query History

```
32 
33 CREATE TABLE TRECHO (
34     ID_LINHA INTEGER NOT NULL,
35     ID_TRECHO INTEGER NOT NULL,
36     ID_LOCAL INTEGER NOT NULL,
37     KM INTEGER NOT NULL,
38 
39     PRIMARY KEY (ID_LINHA, ID_TRECHO),
40 
41     FOREIGN KEY (ID_LINHA) REFERENCES LINHA (ID_LINHA),
42 
43     FOREIGN KEY (ID_LOCAL) REFERENCES LOCALIDADE (ID_LOCAL)
44 );
45 
46 INSERT INTO LOCALIDADE VALUES(36,'CARAPINA');
47 
48 INSERT INTO TRECHO VALUES (100,1, 36, 18), (100,2,11, 28), (100, 3,12, 63), (100, 4, 18, 71), (100, 5, 13, 75),
49 (100,6, 14, 81), (100,7,28, 85), (100,8,27, 104), (100,9,17, 121), (100,10,16,135);
50 
```

Data Output Messages Notifications

INSERT 0 10

Query returned successfully in 105 msec.

Criando o Banco de Dados Passagens

Query Query History

```
53 ✓ CREATE TABLE CLASSE (
54     ID_CLASSE INTEGER NOT NULL PRIMARY KEY,
55     NOME_CLASSE VARCHAR(35) NOT NULL
56 );
57
58 INSERT INTO CLASSE VALUES (10,'COMERCIAL'), (20,'EXECUTIVO'),(30,'LEITO');
59
60
```

Data Output Messages Notifications

INSERT 0 3

Query returned successfully in 86 msec.

Criando o Banco de Dados Passagens

Query Query History

```
60
61 ✓ CREATE TABLE TARIFA
62 (
63     id_tarifa integer NOT NULL,
64     id_classe integer NOT NULL,
65     dt_inicio_validade date NOT NULL,
66     dt_fim_validade date,
67     valor numeric(9,2) NOT NULL,
68     CONSTRAINT tarifa_pkey PRIMARY KEY (id_tarifa),
69
70     FOREIGN KEY(ID_CLASSE) REFERENCES CLASSE (ID_CLASSE)
71
72 );
73
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 68 msec.

Criando o Banco de Dados Passagens

```
73
74
75 v INSERT INTO tarifa(id_tarifa, id_classe, dt_inicio_validade, dt_fim_validade, valor)
76     VALUES (1, 10, '2024/01/02', '2025/01/01', 5.50),
77             (2, 10, '2025/01/02', NULL, 7.00),
78             (3, 20, '2025/01/02', NULL, 9.50),
79             (4,30,'2025/03/25', NULL, 12.0);
80
```

Data Output Messages Notifications

INSERT 0 4

Query returned successfully in 74 msec.

Criando o Banco de Dados Passagens

The screenshot shows a database query editor interface. The top navigation bar has tabs for "Query" and "Query History", with "Query" currently selected. The main area displays a multi-line SQL script. The script starts with a CREATE TABLE statement for a table named "VIAGEM". It defines five columns: "ID_VIAGEM" (SERIAL, NOT NULL, PRIMARY KEY), "ID_LINHA" (INTEGER, NOT NULL), "DATA_HORA" (TIMESTAMP, NOT NULL), "TOTAL_LUGARES" (INTEGER, NOT NULL), and "ID_CLASSE" (INTEGER, NOT NULL). It also includes two FOREIGN KEY constraints: one referencing the "LINHA" table and another referencing the "CLASSE" table. The script concludes with an INSERT INTO statement for the "VIAGEM" table, specifying values for all columns. Below the script, there are three tabs: "Data Output", "Messages", and "Notifications", with "Messages" currently selected. The message area shows the result of the insert operation: "INSERT 0 1". At the bottom, a status message indicates: "Query returned successfully in 81 msec."

```
81
82 CREATE TABLE VIAGEM (
83     ID_VIAGEM SERIAL NOT NULL PRIMARY KEY,
84     ID_LINHA INTEGER NOT NULL,
85     DATA_HORA TIMESTAMP NOT NULL,
86     TOTAL_LUGARES INTEGER NOT NULL,
87     ID_CLASSE INTEGER NOT NULL,
88
89     FOREIGN KEY (ID_LINHA) REFERENCES LINHA (ID_LINHA),
90
91     FOREIGN KEY (ID_CLASSE) REFERENCES CLASSE (ID_CLASSE)
92 );
93
94 INSERT INTO VIAGEM VALUES (1,100,'2025-07-30 17:00', 45, 10);
95
```

Data Output Messages Notifications

INSERT 0 1

Query returned successfully in 81 msec.

Criando o Banco de Dados Passagens

Query Query History

```
95
96
97 CREATE TABLE PASSAGEM (
98     ID_PASSAGEM SERIAL NOT NULL PRIMARY KEY,
99     ID_VIAGEM INTEGER NOT NULL,
100    ID_LOCAL_EMBARQUE INTEGER NOT NULL,
101    ID_LOCAL_DESEMBARQUE INTEGER NOT NULL,
102    DATA_HORA_COMPRA TIMESTAMP NOT NULL,
103    PRECO DECIMAL(9,2) NOT NULL,
104    POLTRONA INTEGER NOT NULL,
105    ID_TARIFA INTEGER NOT NULL,
106
107    FOREIGN KEY (ID_VIAGEM) REFERENCES VIAGEM (ID_VIAGEM) ON DELETE RESTRICT,
108
109    FOREIGN KEY (ID_LOCAL_EMBARQUE) REFERENCES LOCALIDADE (ID_LOCAL)
110    ON DELETE RESTRICT,
111
112    FOREIGN KEY (ID_LOCAL_DESEMBARQUE) REFERENCES LOCALIDADE (ID_LOCAL)
113    ON DELETE RESTRICT,
114
115    FOREIGN KEY (ID_TARIFA) REFERENCES TARIFA (ID_TARIFA) ON DELETE RESTRICT
116 );
117
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 88 msec.

Função PLPGSQL

Aproveitemos o Banco de Dados recém criado (Passagens) para retomar funções PLPGSQL que usam e retornam CURSORES.

A função **verificar_poltronas_disponíveis()** recebe como **parâmetro de entrada** ID_VIAGEM e devolve em resposta um CURSOR que lista poltronas disponíveis para venda de passagem.

Query History

```
120 CREATE OR REPLACE FUNCTION verificar_poltronas_disponiveis(p_id_viagem INTEGER)
121 RETURNS refcursor AS $$ 
122 DECLARE
123     total_lugares INTEGER;
124     poltronas_cursor refcursor;
125 BEGIN
126
127     SELECT v.total_lugares
128     INTO total_lugares
129     FROM viagem v
130     WHERE v.id_viagem = p_id_viagem;
131
132     poltronas_cursor := 'poltronas_cursor';
133
134     OPEN poltronas_cursor FOR
135         SELECT
136             p_id_viagem AS id_viagem,
137             poltrona_numero AS poltrona,
138             NOT EXISTS (
139                 SELECT 1
140                 FROM passagem pa
141                 WHERE pa.id_viagem = p_id_viagem
142                     AND pa.poltrona = poltrona_numero
143             ) AS disponivel
144             FROM generate_series(1, total_lugares) AS poltrona_numero;
145
146     RETURN poltronas_cursor;
147 END;
148 $$ LANGUAGE plpgsql;
```

Data Output Messages Notifications

CREATE FUNCTION

Query returned successfully in 83 msec.

Função PLPGSQL

The screenshot shows a PostgreSQL query editor interface. The top navigation bar has tabs for "Query" and "Query History", with "Query" currently selected. Below the tabs is a code editor area containing the following PL/pgSQL code:

```
150
151 BEGIN;
152
153 -- Cria o cursor e retorna o nome
154 SELECT verificar_poltronas_disponiveis(1);
155
156 -- Consulta os dados do cursor corretamente
157 FETCH ALL FROM poltronas_cursor;
158
159 COMMIT;
160
```

Below the code editor is a toolbar with several icons: a plus sign, a file icon, a dropdown arrow, a clipboard icon, another dropdown arrow, a trash bin, a database icon, a download icon, a refresh icon, and an "SQL" button.

At the bottom of the interface is a table titled "Data Output". It contains two rows:

	verificar_poltronas_disponiveis	refcursor
1	poltronas_cursor	

Função PLPGSQL

Como nenhuma PASSAGEM foi vendida para essa VIAGEM (ID_VIAGEM = 1) todas as POLTRONAS estão disponíveis.

A coluna DISPONÍVEL é TRUE (verdadeiro) para cada uma das poltronas.

The screenshot shows a PostgreSQL query editor interface. The top section is the 'Query' tab, containing the following PL/pgSQL code:

```
151 BEGIN;
152
153 -- Cria o cursor e retorna o nome
154 SELECT verificar_poltronas_disponiveis(1);
155
156 -- Consulta os dados do cursor corretamente
157 FETCH ALL FROM poltronas_cursor;
158
159 COMMIT;
160
```

The bottom section is the 'Data Output' tab, displaying the results of the query. The table has four columns: id_viagem, poltrona, disponivel. All rows show disponivel as true.

	id_viagem integer	poltrona integer	disponivel boolean
1	1	1	true
2	1	2	true
3	1	3	true
4	1	4	true
5	1	5	true
6	1	6	true
7	1	7	true
8	1	8	true
9	1	9	true
10	1	10	true
11	1	11	true
12	1	12	true
13	1	13	true
14	1	14	true

Função PLPGSQL

```
Query  Query History
151    BEGIN;
152
153    -- Cria o cursor e retorna o nome
154    SELECT verificar_poltronas_disponiveis(1);
155
156    -- Consulta os dados do cursor corretamente
157    FETCH ALL FROM poltronas_cursor;
158
159    COMMIT;
160

Data Output  Messages  Notifications
                                         COMMIT

Query returned successfully in 155 msec.
```

Função PLPGSQL

```
Query Query History
162
163 CREATE OR REPLACE FUNCTION vender_passagem(
164     p_id_viagem INTEGER,
165     p_id_local_embarque INTEGER,
166     p_id_local_desembarque INTEGER,
167     p_poltrona INTEGER
168 ) RETURNS TEXT AS $$ 
169 DECLARE
170     v_id_linha INTEGER;
171     v_id_classe INTEGER;
172     v_id_tarifa INTEGER;
173     v_valor NUMERIC(9,2);
174     v_data_viagem TIMESTAMP;
175     v_ordem_embarque INTEGER;
176     v_ordem_desembarque INTEGER;
177     conflito INTEGER;
178 BEGIN
179     -- 1. Recuperar dados da viagem
180     SELECT id_linha, id_classe, data_hora
181     INTO v_id_linha, v_id_classe, v_data_viagem
182     FROM viagem
183     WHERE id_viagem = p_id_viagem;
184
185 IF NOT FOUND THEN
186     RAISE EXCEPTION 'Viagem % não encontrada.', p_id_viagem;
187 END IF;
188
```

Função PLPGSQL

```
Query Query History
188
189      -- 2. Validar se os locais pertencem à linha e recuperar ordem dos trechos
190      SELECT id_trecho INTO v_ordem_embarque
191      FROM trecho
192      WHERE id_linha = v_id_linha AND id_local = p_id_local_embarque;
193
194      IF NOT FOUND THEN
195          RAISE EXCEPTION 'Local de embarque % não pertence à linha %', p_id_local_embarque, v_id_linha;
196      END IF;
197
198      SELECT id_trecho INTO v_ordem_desembarque
199      FROM trecho
200      WHERE id_linha = v_id_linha AND id_local = p_id_local_desembarque;
201
202      IF NOT FOUND THEN
203          RAISE EXCEPTION 'Local de desembarque % não pertence à linha %', p_id_local_desembarque, v_id_linha;
204      END IF;
205
206      IF v_ordem_desembarque <= v_ordem_embarque THEN
207          RAISE EXCEPTION 'Desembarque deve ocorrer após o embarque.';
208      END IF;
209
210      -- 3. Verificar conflito com outra passagem já vendida para a mesma poltrona
211      SELECT COUNT(*) INTO conflito
212      FROM passagem pa
213      JOIN viagem v ON v.id_viagem = pa.id_viagem
214      JOIN trecho t1 ON t1.id_linha = v.id_linha AND t1.id_local = pa.id_local_embarque
215      JOIN trecho t2 ON t2.id_linha = v.id_linha AND t2.id_local = pa.id_local_desembarque
216      WHERE pa.id_viagem = p_id_viagem
217          AND pa.poltrona = p_poltrona
218          AND (
219              -- sobreposição de trechos
220              v_ordem_embarque < t2.id_trecho AND v_ordem_desembarque > t1.id_trecho
221          );
```

Função PLPGSQL

```
Query Query History
222
223     IF conflito > 0 THEN
224         RAISE EXCEPTION 'Poltrona % já está ocupada em trecho sobreposto.', p_poltrona;
225     END IF;
226
227     -- 4. Obter a tarifa válida mais recente
228     SELECT id_tarifa, valor INTO v_id_tarifa, v_valor
229     FROM tarifa
230     WHERE id_classe = v_id_classe
231     AND dt_inicio_validade <= v_data_viagem
232     AND (dt_fim_validade IS NULL OR dt_fim_validade >= v_data_viagem)
233     ORDER BY dt_inicio_validade DESC
234     LIMIT 1;
235
236     IF NOT FOUND THEN
237         RAISE EXCEPTION 'Não há tarifa válida para a classe %', v_id_classe;
238     END IF;
239
240     -- 5. Inserir a passagem
241     INSERT INTO passagem (
242         id_viagem, id_local_embarque, id_local_desembarque,
243         data_hora_compra, preco, poltrona, id_tarifa
244     )
245     VALUES (
246         p_id_viagem, p_id_local_embarque, p_id_local_desembarque,
247         now(), v_valor, p_poltrona, v_id_tarifa
248     );
249
250     RETURN 'Passagem vendida com sucesso.';
251
252 $$ LANGUAGE plpgsql;
253
```

Função PLPGSQL

```
253
254
255 -- Chamada da função:
256 SELECT vender_passagem(1, 36, 16, 12);
257
```

Data Output Messages Notifications

vender_passagem
text

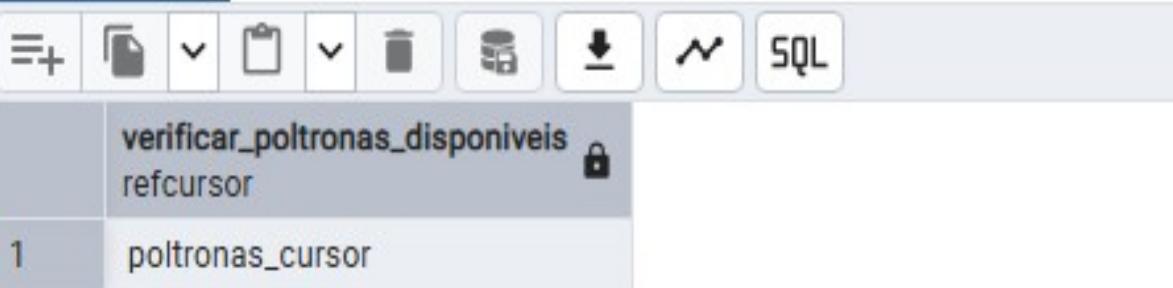
Passagem vendida com sucesso.

The screenshot shows a PostgreSQL client interface. In the top-left, there's a code editor window with several lines of SQL and PL/pgSQL code. Lines 255 and 256 are highlighted in light blue, containing a function call to 'vender_passagem' with parameters 1, 36, 16, and 12. Line 257 is also visible. Below this is a toolbar with various icons for data manipulation. Underneath the toolbar is a table with two rows. The first row is a header with columns for an index (1), the function name ('vender_passagem'), its return type ('text'), and a lock icon. The second row contains the value '1' in the index column and the message 'Passagem vendida com sucesso.' in the text column. The 'Data Output' tab is selected at the bottom of the interface.

Função PLPGSQL

```
150  
151 BEGIN;  
152  
153 -- Cria o cursor e retorna o nome  
154 SELECT verificar_poltronas_disponiveis(1);  
155  
156 -- Consulta os dados do cursor corretamente  
157 FETCH ALL FROM poltronas_cursor;  
158  
159 COMMIT;  
160
```

Data Output Messages Notifications



Função PLPGSQL

Observe que a POLTRONA 12 NÃO está mais disponível.

A COLUNA DISPONÍVEL do CURSOR onde POLTRONA = 12 tem valor FALSE.

Query Query History

```
145
146          RETURN poltronas_cursor;
147      END;
148  $$ LANGUAGE plpgsql;
149
150
151 BEGIN;
152
153 -- Cria o cursor e retorna o nome
154 SELECT verificar_poltronas_disponiveis(1);
155
156 -- Consulta os dados do cursor corretamente
157 FETCH ALL FROM poltronas_cursor;
158
159 COMMIT;
160
```

Data Output Messages Notifications

	id_viajem integer	poltrona integer	disponivel boolean
1	1	1	true
2	1	2	true
3	1	3	true
4	1	4	true
5	1	5	true
6	1	6	true
7	1	7	true
8	1	8	true
9	1	9	true
10	1	10	true
11	1	11	true
12	1	12	false
13	1	13	true
14	1	14	true

Exercício de Fixação

- [5] - Tendo por base o banco de dados Passagens suponha que você está desenvolvendo o sistema de vendas de uma empresa de transporte rodoviário. Cada passagem é vendida com base na **classe da viagem**, na **tarifa válida no momento da compra**, e também na **distância (em km)** percorrida entre os pontos de embarque e desembarque da linha.
- Crie uma **função** chamada **calcular_valor_passagem()** que receba os seguintes parâmetros: p_id_viagem (INTEGER), p_id_local_embarque (INTEGER), p_id_local_desembarque(INTEGER) e p_data_compra (DATE).

Exercício de Fixação

- A função deve retornar o preço da passagem como um valor numérico (numeric(9,2)), calculado da seguinte forma:
 - **valor = tarifa_base × (km_desembarque - km_embarque)**
- A função deve considerar as seguintes **regras de negócio**:
 - A viagem deve existir; caso contrário, dispare uma mensagem de erro com RAISE EXCEPTION.
 - A localidade de embarque e desembarque devem pertencer aos trechos atendidos pela linha da viagem.
 - O local de desembarque deve aparecer depois do local de embarque na ordem dos trechos.

Exercício de Fixação

- A tarifa válida mais recente deve ser selecionada com base na CLASSE da viagem e na DATA DA COMPRA.
- A função deve calcular a distância a ser percorrida com base nos valores de KM dos trechos da linha.
- Caso alguma informação não seja encontrada, a função deve lançar exceção com RAISE EXCEPTION
- Exemplo de **chamada da função:**
 - **SELECT calcular_valor_passagem(1, 36, 16, DATE '2025-08-01');**

