

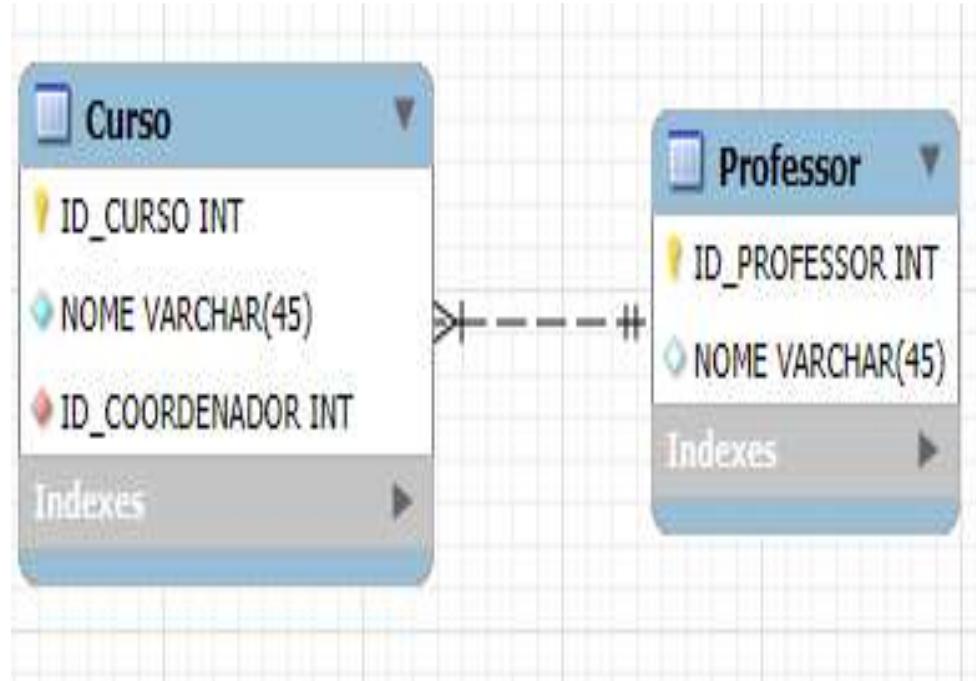
Banco de Dados 2

01 – Funções SQL (PostgreSQL)

PostgreSQL

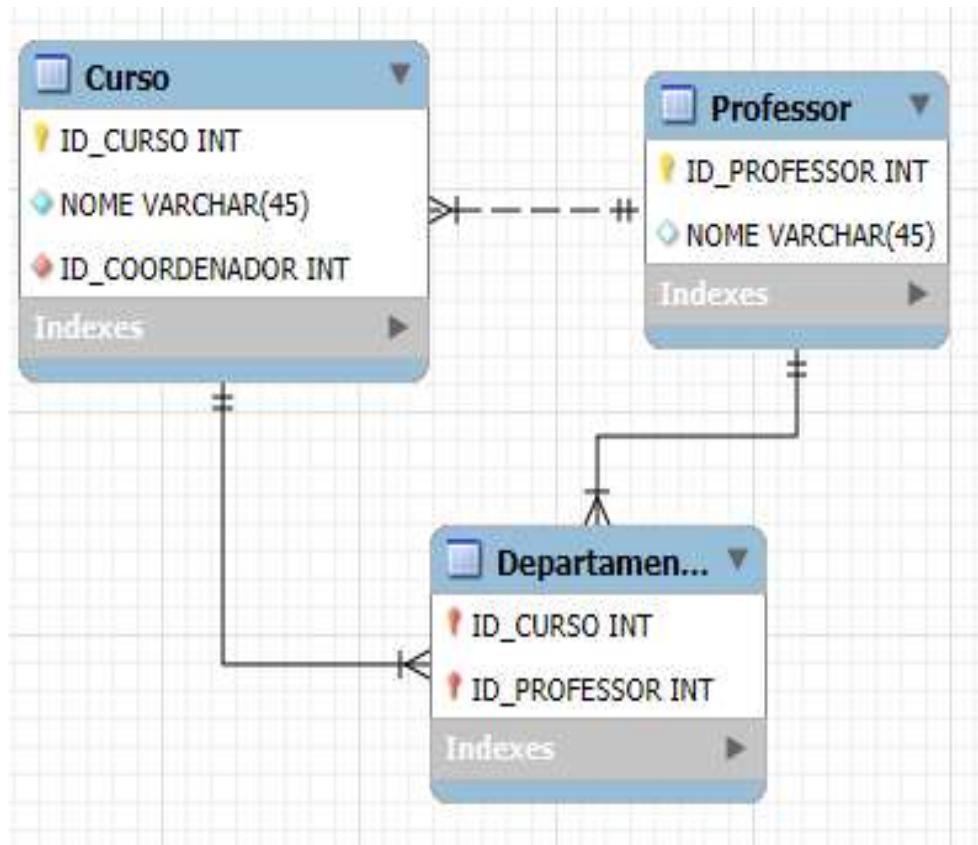
- O **PostgreSQL** oferece diversos recursos para ajudar desenvolvedores a criar aplicativos, administradores a proteger a integridade dos dados e criar ambientes tolerantes a falhas, além de ajudar você a gerenciar seus dados, independentemente do tamanho do conjunto de dados.
- Além de ser gratuito e de código aberto, o PostgreSQL é altamente extensível. Por exemplo, você pode definir seus próprios tipos de dados, criar funções personalizadas e até mesmo escrever código de diferentes linguagens de programação sem precisar recompilar seu banco de dados!
- O **PostgreSQL** busca se adequar ao padrão SQL, desde que tal conformidade não contradiga os recursos tradicionais ou possa levar a decisões arquitetônicas inadequadas. Muitos dos recursos exigidos pelo padrão SQL são suportados, embora às vezes com sintaxe ou função ligeiramente diferentes. Novas mudanças em direção à conformidade podem ser esperadas ao longo do tempo.
- Desde o lançamento da **versão 16**, em **setembro de 2023**, o **PostgreSQL** está em **conformidade com pelo menos 170 dos 177 recursos obrigatórios para a conformidade com o SQL:2023 Core**. Na verdade, nenhum banco de dados relacional atende à conformidade total com este padrão.

Modelando um Banco de Dados



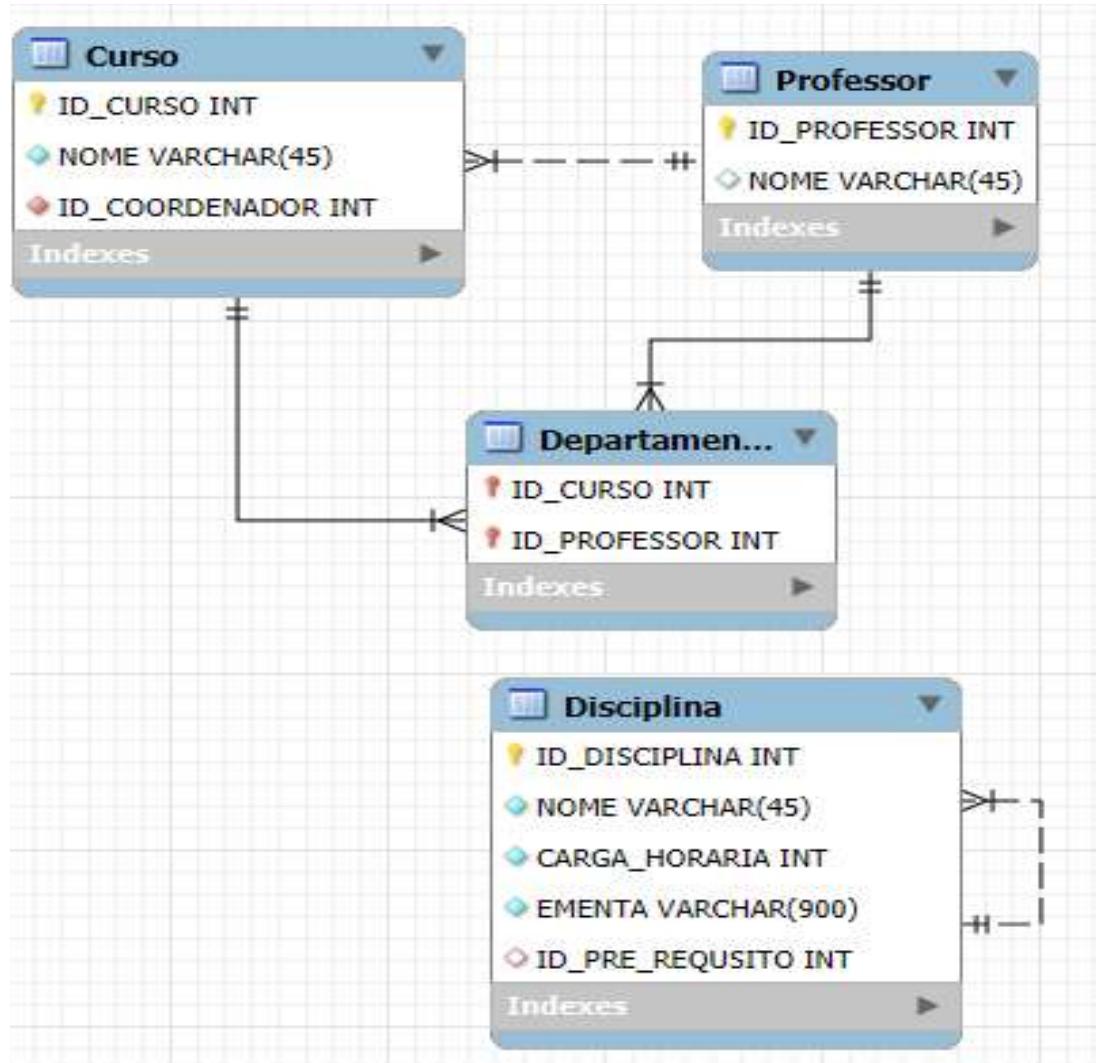
- Comecemos pela **modelagem** de um **Banco de Dados** para uma **Instituição de Ensino Superior (IES)**.
- A referida IES oferece inúmeros cursos de educação superior.
- Cada curso terá um professor designado como seu coordenador.

Modelagem da IES



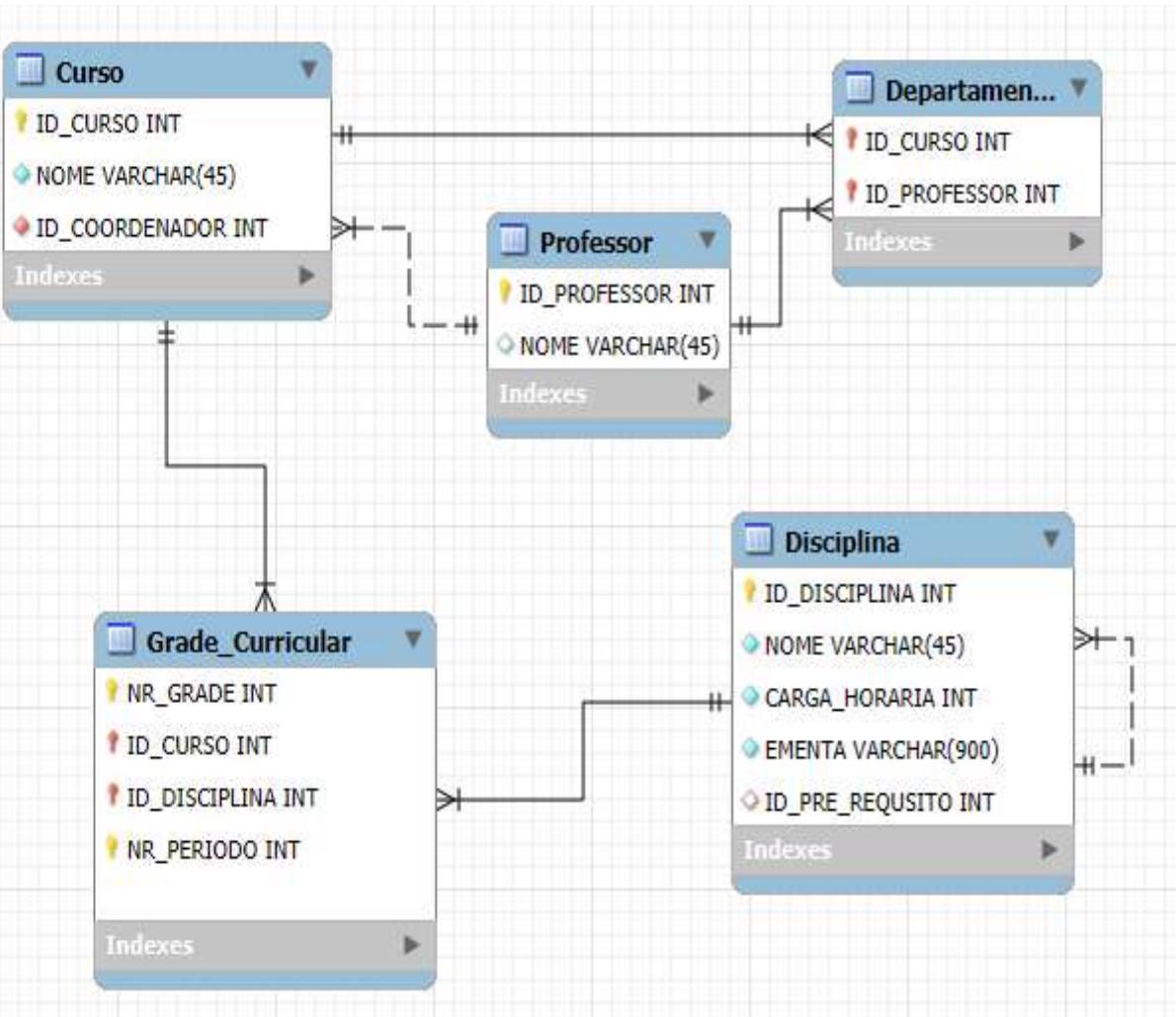
- Cada curso possui um departamento que é formado pelo conjunto de professores inscritos no curso.
- Cada professor do IES pode estar inscrito em mais de um curso existente.

Modelagem da IES



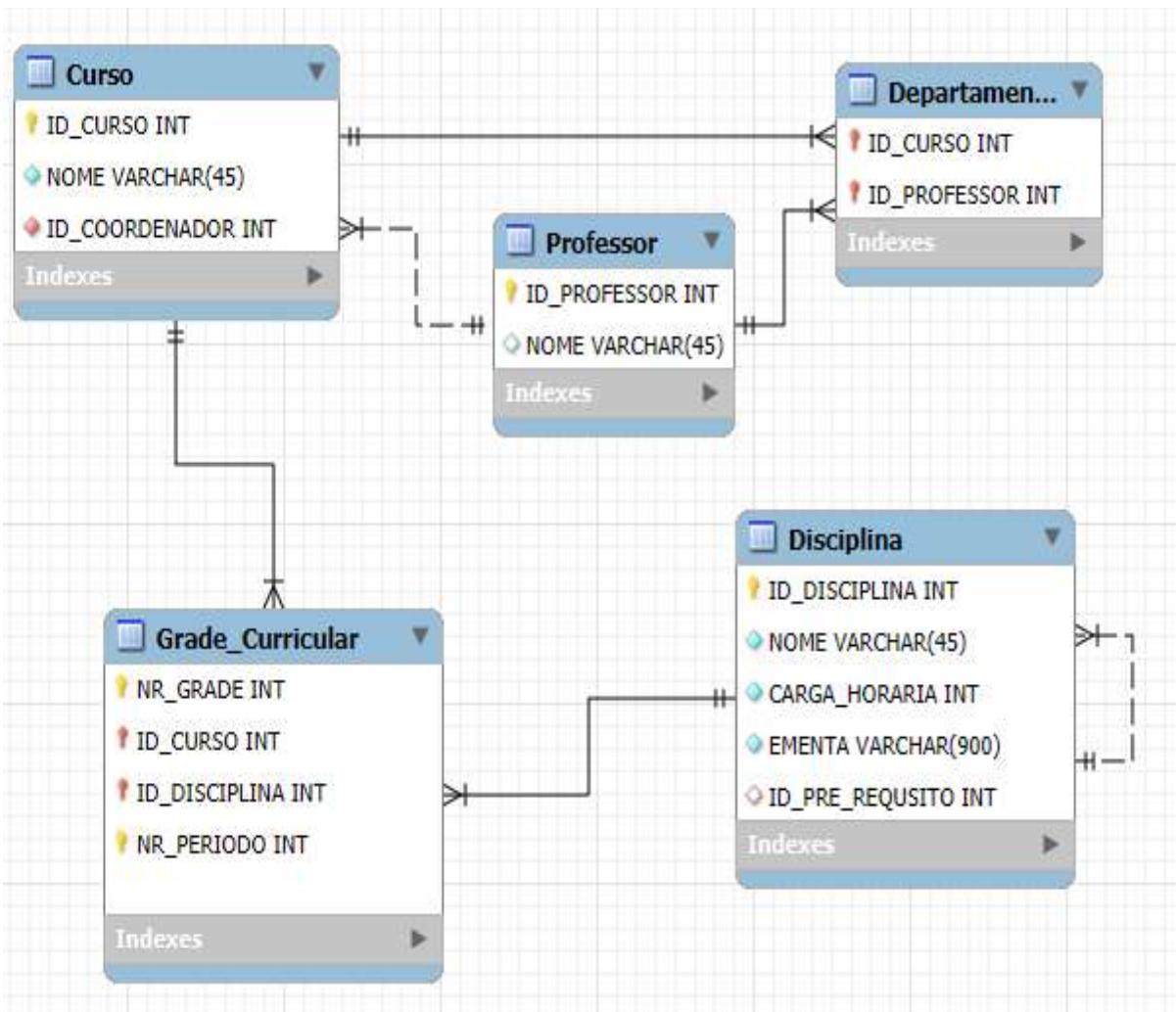
- Existe um conjunto de **disciplinas** caracterizadas por um **nome**, **carga horária** e **ementa** (conteúdos obrigatórios específicos).
- Uma disciplina, para ser cursada, pode exigir a prévia aprovação pelo aluno em outra disciplina (Pré-requisito).
- Considere que uma disciplina pode apresentar, no máximo, um pré-requisito.

Modelagem da IES



- Todo curso possui uma grade curricular que descreve, período a período, as disciplinas que precisam ser cursadas para a obtenção do diploma.
- Logicamente, é necessário que cada pré-requisito de uma disciplina esteja na mesma grade que a disciplina que dele precise.

Modelagem da IES



- Por enquanto vamos parar por aqui em nossa modelagem.
- Ela será terminada mais adiante.
- Já temos o que será necessário para prosseguirmos em nosso conteúdo.

Data Definition Language (DDL)

- A ***Data Definition Language (DDL)*** define a estrutura dos objetos do banco de dados, mas não manipula os dados em si (isso é papel da ***DML — Data Manipulation Language***).
- Comandos principais da DDL:

Comando

CREATE

ALTER

DROP

Função

Cria um novo objeto no banco, como uma tabela, índice, ou esquema.

Modifica a estrutura de um objeto existente (ex: adicionar coluna a uma tabela).

Exclui um objeto do banco (ex: remover tabela ou índice).

Data Definition Language (DDL)

Query Query History

```
1 -- SCRIPT de criação do Banco de Dados IES
2 -- 23-07-2025
3 --
4 ▼ CREATE TABLE PROFESSOR (
5     ID_PROFESSOR INTEGER NOT NULL PRIMARY KEY,
6     NOME VARCHAR(45) NOT NULL
7 );
8
9 ▼ CREATE TABLE CURSO (
10    ID_CURSO SERIAL NOT NULL PRIMARY KEY,
11    NOME VARCHAR(45) NOT NULL,
12    ID_COORDENADOR INTEGER NOT NULL,
13
14    FOREIGN KEY (ID_COORDENADOR) REFERENCES PROFESSOR (ID_PROFESSOR)
15    ON DELETE RESTRICT
16 );
17
18 ▼ CREATE TABLE DEPARTAMENTO (
19    ID_CURSO INTEGER NOT NULL,
20    ID_PROFESSOR INTEGER NOT NULL,
21
22    PRIMARY KEY(ID_CURSO, ID_PROFESSOR),
23
24    FOREIGN KEY (ID_CURSO) REFERENCES CURSO (ID_CURSO) ON DELETE RESTRICT,
25    FOREIGN KEY (ID_PROFESSOR) REFERENCES PROFESSOR (ID_PROFESSOR) ON DELETE RESTRICT
26 );
27
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 92 msec.

Data Definition Language (DDL)

```
17
18 ✓ CREATE TABLE DEPARTAMENTO (
19     ID_CURSO INTEGER NOT NULL,
20     ID_PROFESSOR INTEGER NOT NULL,
21
22     PRIMARY KEY(ID_CURSO, ID_PROFESSOR),
23
24     FOREIGN KEY (ID_CURSO) REFERENCES CURSO (ID_CURSO) ON DELETE RESTRICT,
25     FOREIGN KEY (ID_PROFESSOR) REFERENCES PROFESSOR (ID_PROFESSOR) ON DELETE RESTRICT
26 );
27
28 ✓ CREATE TABLE DISCIPLINA (
29     ID_DISCIPLINA SERIAL NOT NULL PRIMARY KEY,
30     NOME VARCHAR(45) NOT NULL,
31     CARGA_HORARIA INTEGER NOT NULL,
32     EMENTA VARCHAR(900) NOT NULL,
33     ID_PRE_REQUISITO INTEGER,
34
35     FOREIGN KEY (ID_PRE_REQUISITO) REFERENCES DISCIPLINA (ID_DISCIPLINA)
36     ON DELETE RESTRICT
37 );
38
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 83 msec.

Data Definition Language (DDL)

```
28 ✓ CREATE TABLE DISCIPLINA (
29     ID_DISCIPLINA SERIAL NOT NULL PRIMARY KEY,
30     NOME VARCHAR(45) NOT NULL,
31     CARGA_HORARIA INTEGER NOT NULL,
32     EMENTA VARCHAR(900) NOT NULL,
33     ID_PRE_REQUISITO INTEGER,
34
35     FOREIGN KEY (ID_PRE_REQUISITO) REFERENCES DISCIPLINA (ID_DISCIPLINA)
36     ON DELETE RESTRICT
37 );
38
39 ✓ CREATE TABLE GRADE_CURRICULAR (
40     NR_GRADE INTEGER NOT NULL,
41     NR_PERIODO INTEGER NOT NULL,
42     ID_CURSO INTEGER NOT NULL,
43     ID_DISCIPLINA INTEGER NOT NULL,
44
45     PRIMARY KEY(NR_GRADE, NR_PERIODO, ID_CURSO, ID_DISCIPLINA),
46
47     FOREIGN KEY (ID_CURSO) REFERENCES CURSO (ID_CURSO) ON DELETE RESTRICT,
48     FOREIGN KEY (ID_DISCIPLINA) REFERENCES DISCIPLINA (ID_DISCIPLINA) ON DELETE RESTRICT
49 );
50
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 75 msec.

A Carga Inicial de Dados

Query

Query History



```
1 INSERT INTO PROFESSOR VALUES (10,'ASDRUBAL SOARES RIBEIRO'), (22,'JAMBIRA TIMBIRAS'), (31,'DESIDERIO MONFORTE');  
2  
3 ✓ INSERT INTO PROFESSOR VALUES (15,'SUETONIO ROMAO SOUZA'), (21,'CARLOTA FLAVINIAS'), (33,'RONALDO CARVALHO CAMPOS'),  
4 (34,'EDUARDO LICOLINI'), (35,'CINTHIA RIBEIRO'),(36,'RENATO AURELIO MARCONDES'),(37,'ANA PAULA XAVIER DE ABREU'),  
5 (38,'CLAUDIA LUCIA OLIVEIRA OHMS'),(39,'JOSE MAGALHAES NETO'),(40,'JOSE MAURICIO MADEIRO'), (41,'ANA PAULA LASCASAS'),  
6 (42,'CLAUDIA GONCALVES RIBEIRO'),(43,'TITO CARVALHO'),(44,'ROXANNA ALVES MANGABEIRA'), (45,'LEONARDO PEREIRA');  
7  
8 INSERT INTO CURSO VALUES (100,'CIENCIAS ECONOMICAS', 22), (200,'ARQUITETURA E URBANISMO', 31), (300,'SISTEMAS DE INFORMACAO',10);  
9  
10  
11 ✓ INSERT INTO DISCIPLINA VALUES (1079,'CALCULO', 120,'LIMITE, DIFERENCIAL E INTEGRAL.', NULL),  
12 (1080,'CALCULO 2', 120,'CALCULO NUMERICO.', 1079),  
13 (1081,'LOGICA', 120, 'CALCULO PROPOSICIONAL, INFERENCE E LOGICA NAO CLASSICA.', NULL),  
14 (1082,'MATEMATICA DISCRETA', 90,'GRAFOS, AUTOMATOS FINITOS DETERMINISTICOS.', NULL),  
15 (1083,'PROGRAMACAO', 120,'COMANDOS CONDICIONAIS, COMANDOS DE REPETICAO E PROCEDIMENTOS RECURSIVOS.', NULL);  
16  
17 -----
```

A Carga Inicial de Dados

Query Query History

```
16
17 -----
18
19 ↴ INSERT INTO DISCIPLINA VALUES (1084,'ESTRUTURA DE DADOS',120,'LISTAS ENCADEADAS. FILAS. PILHAS. ARVORES. GRAFOS', 1083),
20                               (1085,'BANCO DE DADOS',90,'MODELAGEM. SQL.', NULL),
21                               (1086,'BANCO DE DADOS 2', 120,'TRANSACOES, VIEWS, FUNCTIONS, STORED PROCEDURE, RULES.',1085);
22
23
24 ↴ INSERT INTO GRADE_CURRICULAR VALUES (1,1,300,1079), (1,2,300,1080), (1,1,300, 1081), (1,1, 300, 1082), (1,2,300, 1083),
25                               (1,3,300,1084), (1,3,300, 1085),(1,4,300,1086);
26
27
28
29
30
```

Funções

- No **PostgreSQL** podemos ter **três** tipos de **funções**:
 - [1] **Funções em Linguagem SQL**
 - [2] **Funções de Linguagens Procedurais**
 - [3] **Funções Externas.**
- **Funções em Linguagem SQL:** As **funções** em **SQL** não possuem **variáveis** e **estruturas de comando** (*if, for* etc).
- Elas apenas consistem em uma **lista de comandos SQL** (**SELECT, INSERT, DELETE** ou **UPDATE**), devendo retornar, obrigatoriamente, um determinado valor.

Funções

- Assim, o **último comando** deve ser sempre um **SELECT**.
- Essas funções são carregadas juntamente com o serviço do **PostgreSQL**, não necessitando de nenhuma carga de módulo adicional.
- **Funções de Linguagens Procedurais:** Esse tipo de função **utiliza variáveis e estruturas de comandos**, além de executar **ações SQL**.
- Na versão atual do **PostgreSQL** temos **quatro tipos de linguagens procedurais: PL/PgSQL, PL/Tcl, PL/Perl e PL/Python**.
- A **Linguagem PL/PgSQL** é a **mais utilizada**, pois é **bem estruturada e fácil de aprender**.

Funções

- As **linguagens PL/Tcl**, **PL/Perl** e **PL/Python** têm sintaxe semelhante às linguagens das quais elas herdam sua implementação.
- A **linguagem PL/PGSQL** é semelhante a **linguagem PL/SQL**, do **Oracle**.
- **Funções Externas:** No **PostgreSQL** podemos utilizar **funções** desenvolvidas em uma **linguagem externa**, como **C++**.
- A vantagem é que passamos a contar com o poder de uma **linguagem de programação completa**, possibilitando a **implementação** de **rotinas complexas no banco de dados**.
- As funções devem ser empacotadas em bibliotecas compatilhadas que, por sua vez, devem ser registradas no SGBD.

Funções SQL

```
1 | CREATE [OR REPLACE] FUNCTION NomeDaFuncao([parâmetro 1, parâmetro 2, parâmetro n])
2 |   RETURNS RetornoTipoDeDados AS '
3 |     Corpo da Função;
4 |
5 |   LANGUAGE 'SQL';
```

- **CREATE FUNCTION** - Define o **nome da função** e seus respectivos **parâmetros**, caso existam. Esses **parâmetros** são **identificados internamente** como **\$1** (Parâmetro 1), **\$2** (Parâmetro 2), **\$3** (Parâmetro 3), e assim por diante.
- **RETURNS RetornoTipoDeDados** - Indica o **tipo de dado de retorno** da função. Uma função pode retornar um **tipo simples** como **integer**, **varchar** etc.
- **Funções em SQL** também podem **retornar** um **conjunto de valores** ou uma **estrutura composta de várias linhas** (**resultset**).

Funções SQL

- **Corpo da Função** - Contém a implementação da função e deve estar entre aspas simples.
- **LANGUAGE 'SQL'** - indica que a **linguagem utilizada** para **implementação** da **função** é **SQL** (se estivéssemos utilizando a linguagem **PL/PgSQL**, usaríamos **LANGUAGE 'PLPGSQL'**);
- A **função** é de “**propriedade**” do **usuário** que a **criou** e, para ser **acessada** por **outro usuário** do **banco**, é necessário que este possua o **grant** de **EXECUTE** na mesma.
- Esse procedimento pode ser efetuado com o comando **GRANT EXECUTE ON *nomedafuncao* TO GROUP *nomedousuario***.

Funções SQL

- Vejamos um **exemplo** de **criação** de uma função SQL.
- Esta **função** recebe como **parâmetro** um **único valor do tipo inteiro** (identificado como **\$1**), e também **retorna um valor do tipo inteiro**.
- O **valor de retorno** é definido pela execução do último comando da rotina (neste exemplo, o único comando existente):

```
1 CREATE FUNCTION incrementar(INTEGER)
2 RETURNS INTEGER AS '
3     SELECT $1 + 1 ;
4 '
5 LANGUAGE 'SQL';
```

Funções SQL

```
28
29
30 CREATE FUNCTION incrementar (INTEGER)
31 RETURNS INTEGER AS '
32     SELECT $1 + 1;
33 '
34 LANGUAGE SQL;
35
```

Data Output Messages Notifications

CREATE FUNCTION

Query returned successfully in 70 msec.

```
28
29
30 CREATE FUNCTION incrementar (INTEGER)
31 RETURNS INTEGER AS '
32     SELECT $1 + 1;
33 '
34 LANGUAGE SQL;
35
36 SELECT incrementar(27);
```

Data Output Messages Notifications

incrementar
integer

1	28
---	----

Funções SQL

- **CREATE FUNCTION** - Define o nome da função e seus respectivos parâmetros, caso existam. Esses parâmetros são identificados internamente como **\$1** (Parâmetro 1), **\$2** (Parâmetro 2), **\$3** (Parâmetro 3), e assim por diante.
- **RETURNS RetornoTipoDeDados** - Indica o tipo de dado de retorno da função. Uma função pode retornar um tipo simples como integer, varchar etc. Funções em SQL também podem retornar um conjunto de valores ou uma estrutura composta de várias linhas (**resultset**).

Funções SQL

- **LANGUAGE ‘SQL’** - indica que a linguagem utilizada para implementação da função é **SQL** (se estivéssemos utilizando a linguagem **PL/PgSQL**, usariámos **LANGUAGE ‘PLPGSQL’**);
- **NOTA:** A função é de “propriedade” do usuário que a criou e, para ser acessada por outro usuário do banco, é necessário que este possua o **GRANT** de **EXECUTE** para a mesma. Esse procedimento pode ser efetuado com o comando **GRANT EXECUTE ON nomedafuncao TO GROUP nomedousuario**.

Funções SQL

```
35
36 SELECT incrementar(27);
37
38 -----
39
40 CREATE FUNCTION totalDisciplinas(INTEGER)
41 RETURNS INT8 AS '
42     --PARÂMETRO DE ENTRADA $1 CORRESPONDE A ID_CURSO
43
44     SELECT COUNT(*) AS TOTAL_DE_DISCIPLINAS
45         FROM GRADE_CURRICULAR
46         WHERE ID_CURSO = $1;
47
48 '
49 LANGUAGE SQL;
50
51
52
```

Data Output Messages Notifications

CREATE FUNCTION

Query returned successfully in 211 msec.

```
38 -----
39
40 CREATE FUNCTION totalDisciplinas(INTEGER)
41 RETURNS INT8 AS '
42     --PARÂMETRO DE ENTRADA $1 CORRESPONDE A ID_CURSO
43
44     SELECT COUNT(*) AS TOTAL_DE_DISCIPLINAS
45         FROM GRADE_CURRICULAR
46         WHERE ID_CURSO = $1;
47
48 '
49 LANGUAGE SQL;
50
51 SELECT totalDisciplinas(300);
52
```

Data Output		Messages	Notifications
totaldisciplinas	bigint	1	8

Funções SQL

```
53
54
55     SELECT * FROM GRADE_CURRICULAR;
56
57
58
59
60
```

Data Output Messages Notifications

	nr_grade [PK] integer	nr_periodo [PK] integer	id_curso [PK] integer	id_disciplina [PK] integer
1	1	1	300	1079
2	1	2	300	1080
3	1	1	300	1081
4	1	1	300	1082
5	1	2	300	1083
6	1	3	300	1084
7	1	3	300	1085
8	1	4	300	1086

Funções SQL

```
60  
61  
62 SELECT * FROM PROFESSOR;  
63
```

Data Output Messages Notifications

	id_professor [PK] integer	nome character varying (45)
3	31	DESIDERIO MONFORTE
4	15	SUETONIO ROMAO SOUZA
5	21	CARLOTA FLAVINIAS
6	33	RONALDO CARVALHO CAMPOS
7	34	EDUARDO LICOLINI
8	35	CINTHIA RIBEIRO
9	36	RENATO AURELIO MARCONDES
10	37	ANA PAULA XAVIER DE ABREU
11	38	CLAUDIA LUCIA OLIVEIRA OHMS
12	39	JOSE MAGALHAES NETO
13	40	JOSE MAURICIO MADEIRO
14	41	ANA PAULA LASCASAS
15	42	CLAUDIA GONCALVES RIBEIRO
16	43	TITO CARVALHO
17	44	ROXANNA ALVES MANGABEIRA
18	45	LEONARDO PEREIRA

Total rows: 18 Query complete 00:00:00.118

Funções SQL

```
53
54
55 ✓ CREATE FUNCTION insereProfessor(INTEGER, VARCHAR(45))
56 RETURNS INTEGER AS '
57     INSERT INTO PROFESSOR (NOME, ID_PROFESSOR)
58     VALUES ($2, $1);
59
60     SELECT $1;
61 '
62 LANGUAGE SQL;
```

Data Output Messages Notifications

CREATE FUNCTION

Query returned successfully in 89 msec.

```
53
54
55 ✓ CREATE FUNCTION insereProfessor(INTEGER, VARCHAR(45))
56 RETURNS INTEGER AS '
57     INSERT INTO PROFESSOR (NOME, ID_PROFESSOR)
58     VALUES ($2, $1);
59
60     SELECT $1;
61 '
62 LANGUAGE SQL;
```

```
63
64 SELECT insereProfessor(46, 'ANTONIO JONAS PINOTTI');
```

Data Output Messages Notifications



insereprofessor	
	integer
1	46

Funções SQL

```
53
54
55 CREATE FUNCTION insereProfessor(INTEGER, VARCHAR(45))
56 RETURNS INTEGER AS '
57     INSERT INTO PROFESSOR (NOME, ID_PROFESSOR)
58     VALUES ($2, $1);
59
60     SELECT $1;
61 '
62 LANGUAGE SQL;
63
64 SELECT insereProfessor(46,'ANTONIO JONAS PINOTTI');
65
66 SELECT insereProfessor(47,'FLAVIO FALQUETO');
```

Data Output		
	insereprofessor	integer
1		47

68		
69		
70	SELECT * FROM PROFESSOR;	
71		
	Data Output	Messages
	Notifications	
	SQL	
	id_professor [PK] integer	nome character varying (45)
1	10	ASDRUBAL SOARES RIBEIRO
2	22	JAMBIRA TIMBIRAS
3	31	DESIDERIO MONFORTE
4	15	SUETONIO ROMAO SOUZA
5	21	CARLOTA FLAVINIAS
6	33	RONALDO CARVALHO CAMPOS
7	34	EDUARDO LICOLINI
8	35	CINTHIA RIBEIRO
9	36	RENATO AURELIO MARCONDES
10	37	ANA PAULA XAVIER DE ABREU
11	38	CLAUDIA LUCIA OLIVEIRA OHMS
12	39	JOSE MAGALHAES NETO
13	40	JOSE MAURICIO MADEIRO
14	41	ANA PAULA LASCASAS
15	42	CLAUDIA GONCALVES RIBEIRO
16	43	TITO CARVALHO
17	44	ROXANNA ALVES MANGABEIRA
18	45	LEONARDO PEREIRA
19	46	ANTONIO JONAS PINOTTI
20	47	FLAVIO FALQUETO

Funções SQL

```
67
68
69
70 CREATE FUNCTION exibeGrade(CURSO INTEGER, GRADE INTEGER)
71 RETURNS SETOF GRADE_CURRICULAR AS '
72     SELECT *
73         FROM GRADE_CURRICULAR
74         WHERE ID_CURSO = CURSO
75         AND NR_GRADE = GRADE
76         ORDER BY ID_CURSO, NR_GRADE, NR_PERIODO;
77
78 LANGUAGE SQL;
79
80 SELECT exibeGrade(300,1);
81
82
```

Data Output Messages Notifications

CREATE FUNCTION

Query returned successfully in 200 msec.

```
67
68
69
70 CREATE FUNCTION exibeGrade(CURSO INTEGER, GRADE INTEGER)
71 RETURNS SETOF GRADE_CURRICULAR AS '
72     SELECT *
73         FROM GRADE_CURRICULAR
74         WHERE ID_CURSO = CURSO
75         AND NR_GRADE = GRADE
76         ORDER BY ID_CURSO, NR_GRADE, NR_PERIODO;
77
78 LANGUAGE SQL;
79
80 SELECT exibeGrade(300,1);
81
82
```

Data Output Messages Notifications

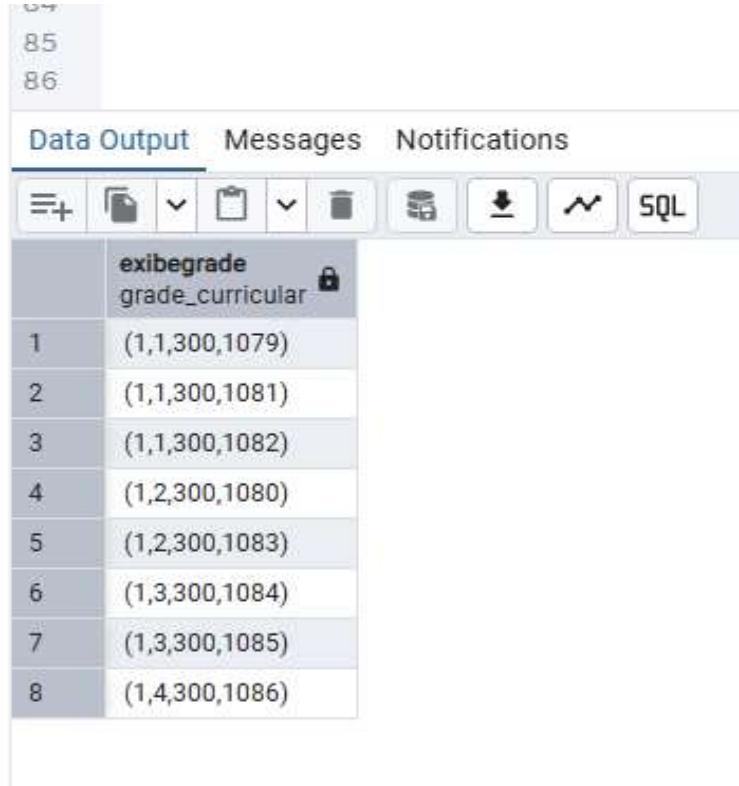
SQL

	exibegrade grade_curricular
1	(1,1,300,1079)
2	(1,1,300,1081)
3	(1,1,300,1082)
4	(1,2,300,1080)
5	(1,2,300,1083)
6	(1,3,300,1084)
7	(1,3,300,1085)
8	(1,4,300,1086)

Funções SQL

- Em **funções SQL** podemos **retornar um conjunto de linhas**; para isso, devemos acrescentar o **parâmetro SETOF** antes do tipo de dado a ser **retornado**.
- Em nosso **exemplo anterior** o **tipo de dado a ser retornado** é **GRADE_CURRICULAR**, ou seja, **registros** da **tabela GRADE_CURRICULAR**.
- **CREATE FUNCTION exibeGrade(CURSO INTEGER, GRADE INTEGER)**
RETURNS SETOF GRADE_CURRICULAR AS

Funções SQL



The screenshot shows a database interface with the following details:

- Code editor area with lines 85 and 86.
- Toolbar with icons for Data Output, Messages, Notifications, and SQL.
- Table named "exibegrade" with a column "grade_curricular".
- 8 rows of data:

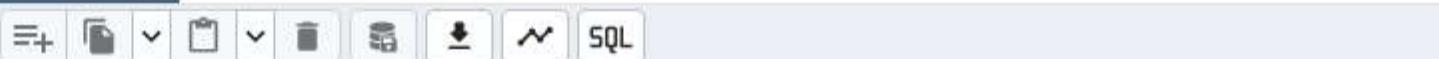
	grade_curricular
1	(1,1,300,1079)
2	(1,1,300,1081)
3	(1,1,300,1082)
4	(1,2,300,1080)
5	(1,2,300,1083)
6	(1,3,300,1084)
7	(1,3,300,1085)
8	(1,4,300,1086)

- Repare em como os registros recuperados pela função foram mostrados.
- Cada registro (linha) traz os valores das colunas entre parênteses e separados por vírgulas: **(1, 1, 300, 1079)**.
- O registro como um todo é apresentado como uma coluna única.
- Os nomes das colunas não são apresentados.
- Isso não é amigável para o usuário!
- **Como melhorar essa apresentação?**

Query Query History

```
70 ✓ CREATE FUNCTION exibeGrade(CURSO INTEGER, GRADE INTEGER)
71   RETURNS SETOF GRADE_CURRICULAR AS '
72     SELECT *
73       FROM GRADE_CURRICULAR
74      WHERE ID_CURSO = CURSO
75      AND NR_GRADE = GRADE
76      ORDER BY ID_CURSO, NR_GRADE, NR_PERIODO;
77
78 LANGUAGE SQL;
79
80 SELECT exibeGrade(300,1);
81
82 SELECT ID_CURSO, NR_GRADE, NR_PERIODO, ID_DISCIPLINA FROM exibeGrade(300, 1);
83
```

Data Output Messages Notifications

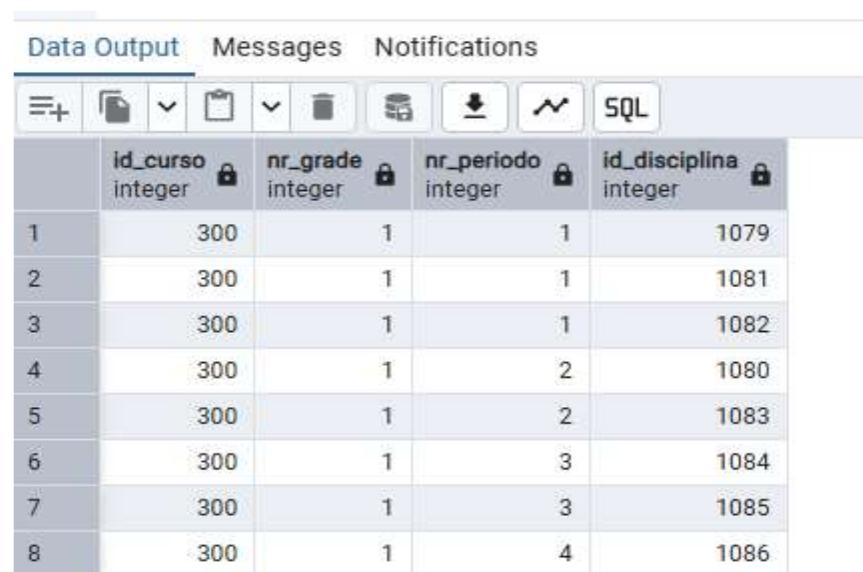


	id_curso integer	nr_grade integer	nr_periodo integer	id_disciplina integer
1	300	1	1	1079
2	300	1	1	1081
3	300	1	1	1082
4	300	1	2	1080
5	300	1	2	1083
6	300	1	3	1084
7	300	1	3	1085
8	300	1	4	1086

- Nenhuma alteração precisa ser feita no código de definição da função.
- Muda apenas a maneira de executar a chamada dessa função.

Funções SQL

- Porém, a maneira como são apresentados os registros da GRADE_CURRICULAR, continua não “amigável” para o usuário-final.
- Estamos exigindo dele um grande esforço de memorização: Qual é o CURSO de ID_CURSO = 300? E a DISCIPLINA de ID_DISCIPLINA = 1079?



The screenshot shows a database interface with a toolbar at the top labeled "Data Output", "Messages", and "Notifications". Below the toolbar is a table titled "GRADE_CURRICULAR". The table has five columns: "id_curso" (integer), "nr_grade" (integer), "nr_periodo" (integer), and "id_disciplina" (integer). The table contains eight rows of data:

	id_curso integer	nr_grade integer	nr_periodo integer	id_disciplina integer
1	300	1	1	1079
2	300	1	1	1081
3	300	1	1	1082
4	300	1	2	1080
5	300	1	2	1083
6	300	1	3	1084
7	300	1	3	1085
8	300	1	4	1086

Funções SQL

```
86 DROP FUNCTION exibeGrade(CURSO INTEGER, GRADE INTEGER);
87
88 ✓ CREATE FUNCTION exibeGrade(CURSOID INTEGER, GRADE INTEGER)
89 RETURNS SETOF RECORD AS '
90     SELECT CURSO.ID_CURSO, CURSO.NOME, NR_GRADE, NR_PERIODO,
91             DISCIPLINA.ID_DISCIPLINA, DISCIPLINA.NOME, CARGA_HORARIA
92
93     FROM GRADE_CURRICULAR, CURSO, DISCIPLINA
94
95     WHERE CURSO.ID_CURSO = CURSOID
96     AND NR_GRADE = GRADE
97     AND CURSO.ID_CURSO = GRADE_CURRICULAR.ID_CURSO
98     AND DISCIPLINA.ID_DISCIPLINA = GRADE_CURRICULAR.ID_DISCIPLINA
99     ORDER BY ID_CURSO, NR_GRADE, NR_PERIODO;
100
101 LANGUAGE SQL;
102
103 ✓ SELECT CURSO.ID_CURSO, CURSO.NOME, NR_GRADE, NR_PERIODO,
104         DISCIPLINA.ID_DISCIPLINA, DISCIPLINA.NOME, CARGA_HORARIA
105     FROM exibeGrade(300, 1);
```

Data Output Messages Notifications

CREATE FUNCTION

Query returned successfully in 89 msec.

- A consulta SQL (Query) foi alterada para buscar as descrições de cada identificador (ID).
- O retorno da função também foi alterado: agora temos **SETOF RECORD** em vez do **SETOF GRADE_CURRICULAR**.
- Isso é feito porque NÃO usamos mais exclusivamente as colunas da tabela GRADE_CURRICULAR.
- **CURSO.NOME** vem da tabela **CURSO** e **DISCIPLINA. NOME** vem da tabela **DISCIPLINA**.

Query Query History

```

86  DROP FUNCTION exibeGrade(CURSO INTEGER, GRADE INTEGER);
87
88  CREATE FUNCTION exibeGrade(CURSOID INTEGER, GRADE INTEGER)
89  RETURNS SETOF RECORD AS '
90      SELECT CURSO.ID_CURSO, CURSO.NOME AS NOME_CURSO, NR_GRADE, NR_PERIODO,
91             DISCIPLINA.ID_DISCIPLINA, DISCIPLINA.NOME AS NOME_DISCIPLINA, CARGA_HORARIA
92
93      FROM GRADE_CURRICULAR, CURSO, DISCIPLINA
94
95      WHERE CURSO.ID_CURSO = CURSOID
96      AND NR_GRADE = GRADE
97      AND CURSO.ID_CURSO = GRADE_CURRICULAR.ID_CURSO
98      AND DISCIPLINA.ID_DISCIPLINA = GRADE_CURRICULAR.ID_DISCIPLINA
99      ORDER BY ID_CURSO, NR_GRADE, NR_PERIODO;
100 '
101 LANGUAGE SQL;
102
103  SELECT * FROM exibeGrade(300, 1)
104  AS (ID_CURSO INTEGER, NOME_CURSO VARCHAR(45), NR_GRADE INTEGER, NR_PERIODO INTEGER,
105      ID_DISCIPLINA INTEGER, NOME_DISCIPLINA VARCHAR(45), CARGA_HORARIA INTEGER);
106
107

```

Data Output Messages Notifications

	id_curso integer	nome_curso character varying (45)	nr_grade integer	nr_periodo integer	id_disciplina integer	nome_disciplina character varying (45)	carga_horaria integer
1	300	SISTEMAS DE INFORMACAO	1	1	1079	CALCULO	120
2	300	SISTEMAS DE INFORMACAO	1	1	1081	LOGICA	120
3	300	SISTEMAS DE INFORMACAO	1	1	1082	MATEMATICA DISCRETA	90
4	300	SISTEMAS DE INFORMACAO	1	2	1080	CALCULO 2	120
5	300	SISTEMAS DE INFORMACAO	1	2	1083	PROGRAMACAO	120
6	300	SISTEMAS DE INFORMACAO	1	3	1084	ESTRUTURA DE DADOS	120
7	300	SISTEMAS DE INFORMACAO	1	3	1085	BANCO DE DADOS	90
8	300	SISTEMAS DE INFORMACAO	1	4	1086	BANCO DE DADOS 2	120

Functions SQL

- Observe como tivemos de alterar o código da própria função:
 - CURSO.NOME teve de receber uma alcunha ou apelido: NOME_CURSO.
 - DISCIPLINA.NOME também teve de ser alterado para NOME_DISCIPLINA.
 - O parâmetro de entrada CURSO (mesmo nome da tabela) teve de ser alterado para CURSOID.
- Também tivemos de alterar o comando de chamada da função.
- Nenhuma das duas outras chamadas usadas anteriormente funcionam nesse contexto.
- Tivemos de adicionar uma lista de colunas com seus respectivos tipos de dados.

Functions SQL

```
10  
11 INSERT INTO DISCIPLINA VALUES (1000,'DISCIPLINA GENERICA',60,' *** QUALQUER COISA ***', NULL);  
12  
13 SELECT * FROM DISCIPLINA;
```

Data Output Messages Notifications

	<code>id_disciplina</code> [PK] integer	<code>nome</code> character varying (45)	<code>carga_horaria</code> integer	<code>ementa</code> character varying (900)	<code>id_pre_requisito</code> integer
1	1079	CALCULO	120	LIMITE, DIFERENCIAL E INTEGRAL.	[null]
2	1080	CALCULO 2	120	CALCULO NUMERICO.	1079
3	1081	LOGICA	120	CALCULO PROPOSICIONAL, INFERENCE E LOGICA NAO CLASSICA.	[null]
4	1082	MATEMATICA DISCRE...	90	GRAFOS, AUTOMATOS FINITOS DETERMINISTICOS.	[null]
5	1083	PROGRAMACAO	120	COMANDOS CONDICIONAIS, COMANDOS DE REPETICAO E PROCEDIMENTOS RECURSIVOS.	[null]
6	1084	ESTRUTURA DE DADOS	120	LISTAS ENCADEADAS. FILAS. PILHAS. ARVORES. GRAFOS	1083
7	1085	BANCO DE DADOS	90	MODELAGEM. SQL.	[null]
8	1086	BANCO DE DADOS 2	120	TRANSACOES, VIEWS, FUNCTIONS, STORED PROCEDURE, RULES.	1085
9	1000	DISCIPLINA GENERICA	60	*** QUALQUER COISA ***	[null]

Function SQL

Query Query History

```
1 ✓ CREATE FUNCTION apagaDisciplina(ID INTEGER)
2 RETURNS TEXT AS $$ 
3     DELETE FROM DISCIPLINA WHERE ID_DISCIPLINA = ID;
4 
5     SELECT 'A DISCIPLINA ' || ID || ' FOI EXCLUIDA COM SUCESSO!';
6 $$ 
7 LANGUAGE SQL;
8 
9 SELECT apagaDisciplina(1000);
10
```

Data Output Messages Notifications

CREATE FUNCTION

Query returned successfully in 95 msec.

Query Query History

```
1 ✓ CREATE FUNCTION apagaDisciplina(ID INTEGER)
2 RETURNS TEXT AS $$ 
3     DELETE FROM DISCIPLINA WHERE ID_DISCIPLINA = ID;
4 
5     SELECT 'A DISCIPLINA ' || ID || ' FOI EXCLUIDA COM SUCESSO!';
6 $$ 
7 LANGUAGE SQL;
8 
9 SELECT apagaDisciplina(1000);
10
```

Data Output Messages Notifications

≡+ 📁 ↻ 🗂️ ↻ 🗑️ 🔍 🔍 SQL

	apagadisciplina	text	🔒
1	A DISCIPLINA 1000 FOI EXCLUIDA COM SUCESSO!		

Function SQL

```
12  
13  
14 ✓ CREATE OR REPLACE FUNCTION dividir(val1 NUMERIC, val2 NUMERIC,  
15                               OUT quociente NUMERIC, OUT resto NUMERIC)  
16 LANGUAGE SQL  
17 AS $$  
18     SELECT val1 / val2, MOD(val1, val2);  
19 $$;  
20  
21  
22 SELECT * FROM dividir(10, 3);  
23
```

Data Output Messages Notifications

The screenshot shows a database management system interface. At the top, there is a code editor window displaying an SQL script for creating a function named 'dividir'. The script defines two output parameters: 'quociente' (of type NUMERIC) and 'resto' (of type NUMERIC). It uses the 'MOD' function to calculate the remainder of the division. Below the code editor is a toolbar with various icons for file operations like new, open, save, and export. Underneath the toolbar is a results grid. The grid has two columns: 'quociente numeric' and 'resto numeric'. A single row of data is shown, with the value '3.333333333333333' in the first column and '1' in the second column. The row number '1' is also present.

	quociente numeric	resto numeric
1	3.333333333333333	1

Function SQL

- Usar **functions** (ou **funções**) em um **banco de dados** traz uma série de **benefícios** que podem melhorar a performance, a manutenção e a segurança da sua aplicação. Aqui estão os principais motivos para utilizá-las:
 - **1. Encapsulamento de Lógica de Negócio:**
 - As **funções** permitem **encapsular regras de negócio** diretamente no **banco de dados**. Isso evita duplicação de lógica em várias partes da aplicação.
 - **Exemplo:** uma função **calcular_desconto(cliente_id)** pode centralizar o cálculo de desconto adotado pela organização. Contudo, isso faz mais sentido para **Funções PL/PGSQL** e **Stored Procedures**. Afinal, **Funções SQL** não usam variáveis, comandos condicionais ou de repetição.

Function SQL

- **2. Reutilização e Organização:**

- Você pode reutilizar a mesma função em várias queries, mantendo o código mais limpo, organizado e modular.

- **3. Performance e Eficiência:**

- Funções armazenadas são compiladas e otimizadas pelo PostgreSQL, o que pode melhorar o desempenho.
- Reduzem a quantidade de dados trafegados entre a aplicação e o banco, pois parte do processamento ocorre dentro do banco.

- **4. Atomicidade e Transações:**

- Funções podem ser usadas para executar **operações complexas** de forma **atômica**, ou seja, todas as operações são executadas por completo ou nenhuma é (rollback automático em erro). **Isso é verdadeiro apenas para Stored Procedures.**

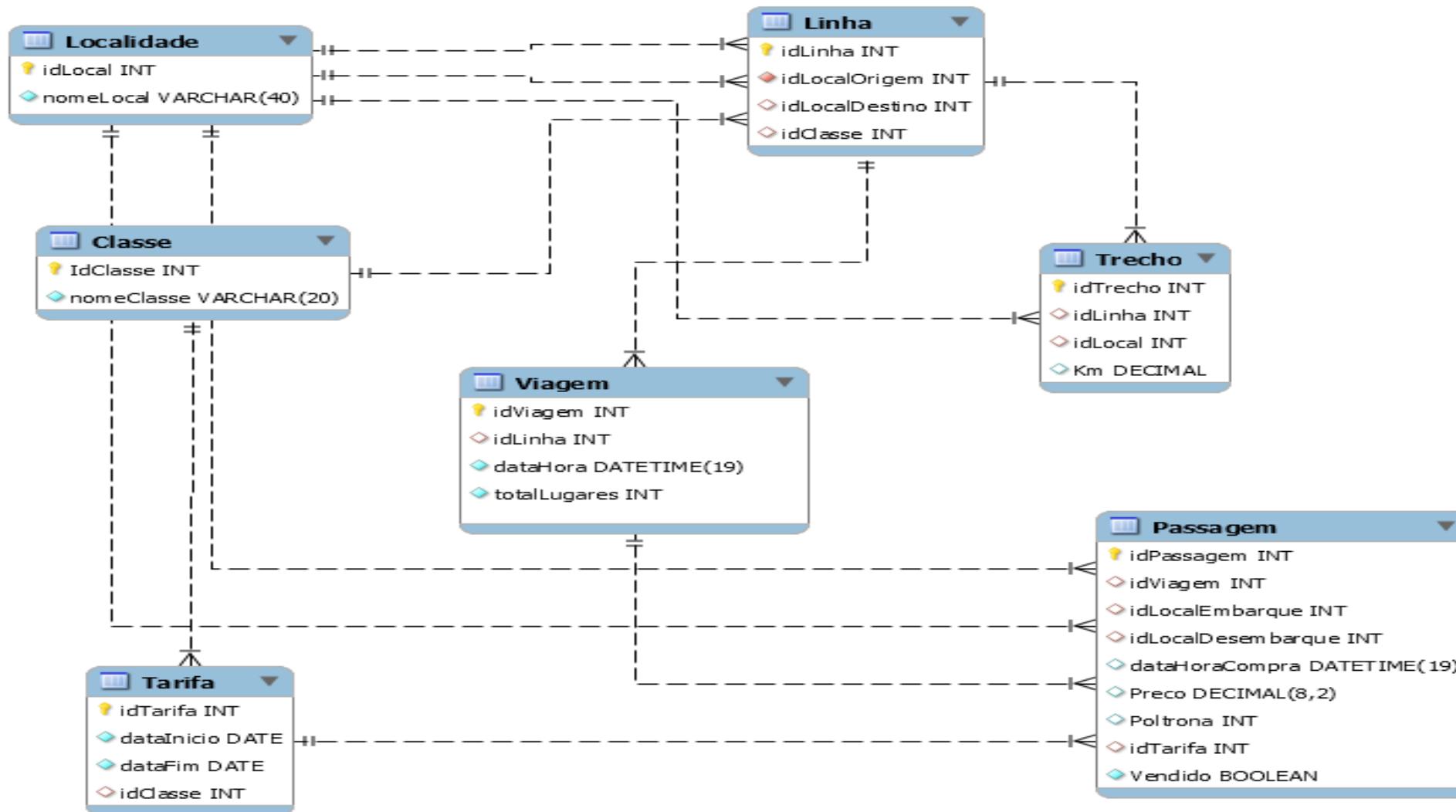
Function SQL

- **5. Segurança:**
 - Permite restringir o acesso direto a tabelas sensíveis. Usuários podem ter permissão apenas para executar funções, sem poder manipular diretamente os dados.
- **6. Suporte a Diversas Linguagens:**
 - No PostgreSQL, você pode criar funções em várias linguagens:
 - **PL/pgSQL** (nativo)
 - **SQL**
 - **PL/Python, PL/Perl, PL/Tcl, PL/R**, entre outras (com extensões)
 - **C++**

Function SQL

- **7. Automatização de Tarefas:**
 - Pode ser usada para automatizar rotinas como:
 - Geração de relatórios
 - Processamento em lote
 - Auditoria e logging
 - Validações personalizadas

Exercícios de Fixação



Exercícios de Fixação

- Considere um **sistema de informação** que suporte as operações de uma **Companhia de serviços de transporte rodoviário de pessoas**.
- Essa companhia atende a diferentes linhas de ônibus intermunicipais:
 - **Vitória X Colatina (origem: Vitória ... Destino: Colatina).**
 - **Colatina X Vitória (origem: Colatina ... Destino: Vitória).**
 - **Colatina X Linhares**
 - **Vitória X Santa Teresa**
 - **Domingos Martins X Vitória**
- Cada linha pode ser descrita em termos de uma sequência de trechos (ou pontos de embarque/desembarque que são atendidos).

Exercícios de Fixação

- As Linhas podem ser atendidas por diferentes classes de ônibus em horários específicos:
 - **Executivo (ônibus com aparelho de ar-condicionado e banheiro);**
 - **Comercial (ônibus sem ar-condicionado);**
 - **Leito (ônibus com assentos mais reclináveis, ar-condicionado e banheiro).**
- Ônibus de diferentes classes cobram tarifas diferenciadas. O “Executivo”, por exemplo, cobra passagens mais caras que o “Comercial” para um mesmo trecho em uma mesma linha.
- Como as Linhas possuem horários fixos distribuídos pelos dias da semana, viagens são programadas e atendidas por ônibus com diferentes quantidades de poltronas.

Exercícios de Fixação

- Quando uma viagem é inserida as passagens são geradas automaticamente com base no total de assentos para os passageiros.
- É gerada uma passagem para cada poltrona.
- Tais passagens, geradas a priori, não foram ainda vendidas para eventuais passageiros.
- Passagens não vendidas terão um número de poltrona mas não terão uma série de outras informações:
 - Local de Embarque;
 - Local de Desembarque;
 - Data e Hora de Compra da Passagem;

Exercícios de Fixação

- O preço e
- O identificador da tarifa vigente.
- Tais informações serão alimentadas no momento da venda da passagem.
- O comprador da passagem informará a poltrona escolhida, o local de seu embarque e de seu desembarque.
- Afinal, a linha “Vitória X Colatina”, por exemplo, atende as seguintes localidades:
 - Vitória (origem da linha),
 - Carapina (município de Serra),

Exercícios de Fixação

- Serra Sede;
- Fundão;
- Ibiraçu;
- João Neiva;
- Baunilha (distrito do município de Colatina);
- Colatina (Destino final da linha).
- Embora esteja comprando um passagem da linha “Vitória X Colatina” o passageiro não está obrigado a embarcar na rodoviária de Vitória e nem a desembarcar na rodoviária de Colatina.
- Ele pode subir, por exemplo, em Ibiraçu e descer em João Neiva.

Exercícios de Fixação

- O embarque e o desembarque previstos determinarão o preço a passagem.
- Se Vitória é o km zero, Ibiraçu o km 75,3 (sua distância em relação a Vitória) e João Neiva o km 84,5, então quem embarcou em Ibiraçu e desembarcou em João Neiva **percorreu 9,2 km**.
- Ele comprou a passagem em 22 de setembro de 2025 sua passagem se baseará na tarifa do período de 01/12/2024 a 30/11/2025. Vamos supor que seu valor é de R\$ 5,00 para a classe “Executivo” e de R\$ 3,50 para a classe “Comercial”.
- O cliente escolheu um horário atendido pela classe “Executivo”.

Exercícios de Fixação

- O valor da passagem seria R\$ 46,00.
- Afinal são R\$ 5,00 (tarifa vigente da classe executiva) multiplicada por 9,2 km.
- Passagem cara!!!
- E esse valor deve ser armazenado na tabela PASSAGEM.
- Mas, por quê?
- Esse valor não pode ser calculado a qualquer momento? Então, por que gastar espaço de armazenamento não volátil com isso?

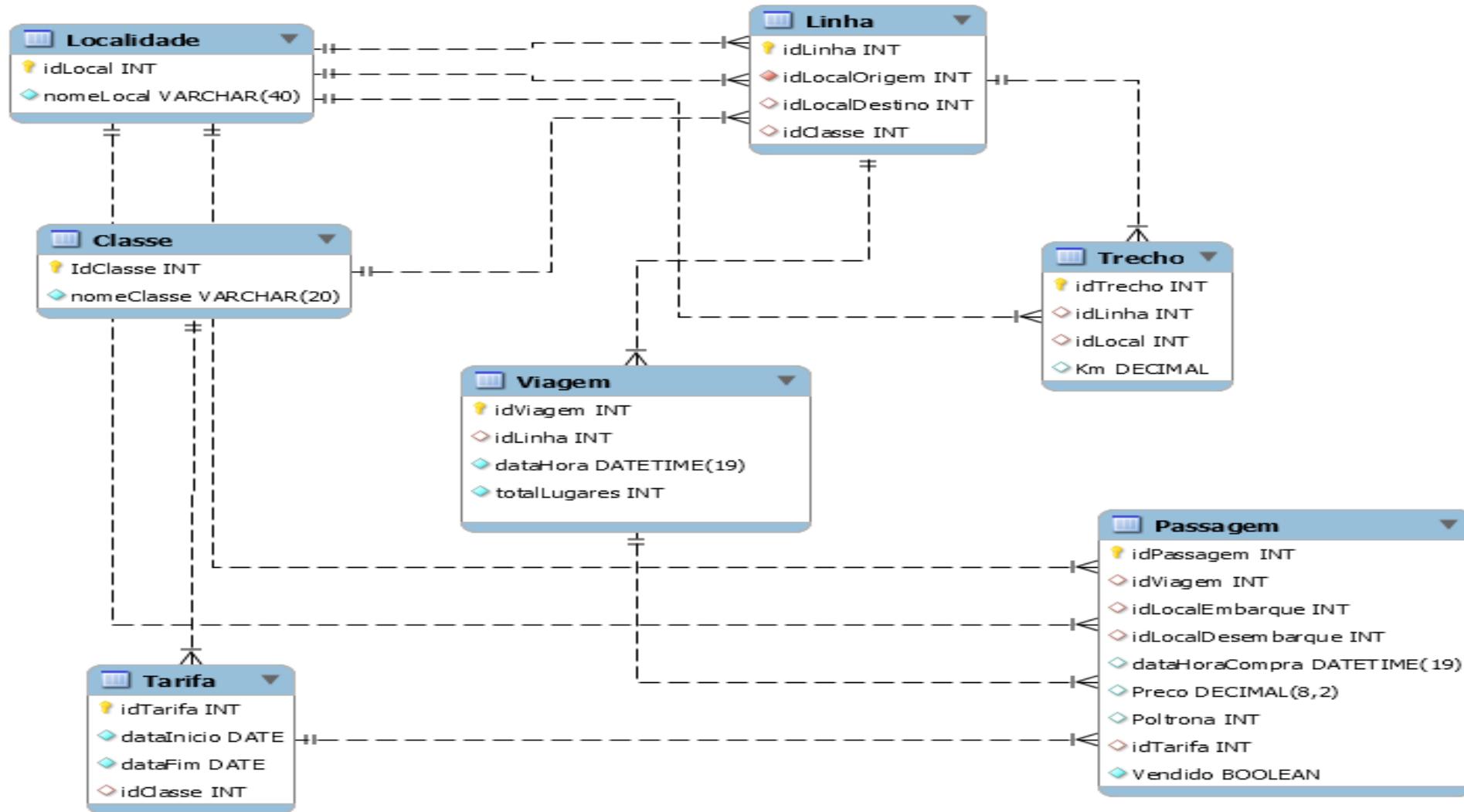
Exercícios de Fixação

- Imagine uma situação em que o passageiro compra a passagem no último dia de validade daquela tarifa.
- Se deixasse para comprar no dia seguinte já pagaria um valor majorado para a mesma passagem.
- Agora suponha que em uma data antes da viagem ele desista da viagem e queira ser ressarcido.
- Ora, se não armazenássemos o preço pago na passagem, ele poderia receber um valor maior que o desembolsado na compra da passagem.

Exercícios de Fixação

- Existe uma outra alternativa em vez de armazenar o preço na tabela PASSAGEM, poderíamos fazer uso da Data de Compra da Passagem.
- Com ela podemos efetuar uma rápida busca na tabela TARIFA e resgatar o valor da tarifa na ocasião da compra da passagem.
- ERRATA: falta na tabela TARIFA uma coluna VALOR de tipo DECIMAL(8,2).
- Sem essa coluna não teríamos como determinar o preço da passagem.

Exercícios de Fixação



Exercícios de Fixação

- Escreva um **SCRIPT** compatível com o **Banco de Dados PostgreSQL** que:
 - [1] – Crie todas as tabelas e seus respectivos relacionamentos.
 - [2] – Efetue uma carga inicial de dados para teste do Sistema.
 - [3] – Desenvolva uma função SQL que exiba todas as viagens programadas para uma data específica. O parâmetro de entrada deve ser a data pretendida para a consulta (data e hora da viagem). Essa função deverá retornar registros com data e hora da viagem, idViagem, idLinha, origem e destino da Linha, total de lugares.
 - [4] - Implemente uma Função SQL para exibir todas as passagens disponíveis para venda de uma determinada viagem. O parâmetro de entrada da função deve ser o identificador da viagem (um inteiro). Uma passagem disponível é aquela em que VENDIDO IS TRUE ou DataHoraCompra = NULL.