



Instituto Politécnico Viana do Castelo
Escola Superior de Tecnologia e Gestão

Engenharia Informática
2023/2024

Tiago Alberto Tristão Borlido Baptista N°28998

Pedro Soares Poças N°28976

Relatório de Projeto
da
Unidade Curricular de **Bases de Dados**

Gestão de Conferências

Dezembro de 2023

RESUMO

Este relatório tem como objetivo a criação de uma base de dados sobre o tema “Gestão de Conferencias”.

Para a realização da base de dados foi necessária a criação do modelo relacional, para se aplicar a informação no software PgAdmin 4. Também se abordou sobre inserção, eliminação, atualização, consultas e triggers para a gestão dos dados na base de dados.

Conteúdo

Índice de Figuras	ii
1. Introdução.....	1
2. Desenvolvimento do trabalho	2
2.1 Levantamento de requisitos	2
2.2 Especificação e design	2
O modelo de Entidade e Relacionamentos – Modelo lógico	2
O esquema de tabelas	2
O esquema de tabelas na 3 FN	2
O modelo Relacional – Modelo físico	4
2.3 Implementação da base de dados	5
Script de criação de tabelas.....	5
Restrições de integridade	13
Script de criação de vistas	14
2.4 Gestão dos dados na base de dados	15
Inserção de dados.....	15
Atualização de dados.....	15
Eliminação de dados.....	16
Consultas simples	16
Consultas com JOIN de tabelas	19
Consultas recorrendo ao uso das vistas	20
Triggers	21
3. Conclusão.....	23
Referências.....	24

ÍNDICE DE FIGURAS

FIGURA 1 - MODELO DE ENTIDADE E RELACIONAMENTO	2
FIGURA 2 - MODELO FÍSICO FINAL	4
FIGURA 3 - TABELAS.....	5
FIGURA 4 - TABELA "TIPODESPESA"	5
FIGURA 5 - TABELA "TIPOSALA"	6
FIGURA 6 - TABELA "TIPOBILHETE"	6
FIGURA 7 - TABELA "METODOPAGAMENTO"	7
FIGURA 8 - TABELA "ORADOR"	7
FIGURA 9 - TABELA "PARTICIPANTE"	8
FIGURA 10 - TABELA "CONFERENCIA"	8
FIGURA 11 - TABELA "DESPESACONFERENCIA"	9
FIGURA 12 - TABELA "SALA"	10
FIGURA 13 - TABELA "SESSAO"	11
FIGURA 14 - TABELA "SESSAOORADOR"	11
FIGURA 15 - TABELA "CONFPARTI"	12
FIGURA 16 - TABELA "BILHETE"	13
FIGURA 17 - INSERÇÃO DE DADOS	15
FIGURA 18 - ATUALIZAÇÃO DE DADOS	16
FIGURA 19 - ELIMINAÇÃO DE DADOS	16
FIGURA 20 - CONSULTA 1	16
FIGURA 21 - CONSULTA 2	17
FIGURA 22 - CONSULTA 3	17
FIGURA 23 - CONSULTA 4	17
FIGURA 24 - CONSULTA 5	18
FIGURA 25 - CONSULTA 6	18
FIGURA 26 - CONSULTA 7	18
FIGURA 27 - CONSULTA 8	18
FIGURA 28 - CONSULTA COM "GROUP BY"	19
FIGURA 29 - CONSULTA COM "HAVING"	19
FIGURA 30 - CONSULTA COM "ORDER BY"	20
FIGURA 31 - CONSULTA COM A VIEW "VIEW_PARTICIPANTE"	20
FIGURA 32 - CONSULTA COM A VIEW "VIEW_SALA_ORDEM_CAPACIDADE"	21

1. INTRODUÇÃO

Neste trabalho prático, foi-nos proposta a criação de uma base de dados eficiente para organizar conferências. As conferências são eventos muito detalhados, como apresentações, participantes, datas e locais. Queremos uma base de dados que facilite a organização destas informações, tornando a gestão das conferências mais fácil e eficaz.

No cerne deste projeto, não queremos apenas construir uma base de dados organizada, mas também usar técnicas avançadas para garantir que ela funcione da melhor forma possível. Queremos simplificar e melhorar os processos relacionados às conferências, tornando a administração destes eventos mais coesa e eficaz.

Ao atingir este objetivo, esperamos não só melhorar a capacidade de guardar, organizar e encontrar informações, mas também contribuir para que toda a experiência das conferências seja melhor, tanto para quem organiza como para quem participa.

2. DESENVOLVIMENTO DO TRABALHO

2.1 LEVANTAMENTO DE REQUISITOS

Foi decidido a implementação do software PostgreSQL para que fosse possível o armazenamento das diversas conferências e os diferentes aspetos presentes no tema presente “Gestão de Conferencias”.

Deste modo, surgiram os esquemas e todo o processo intermédio que permitiram a criação desta base de dados.

2.2 ESPECIFICAÇÃO E DESIGN

O MODELO DE ENTIDADE E RELACIONAMENTOS – MODELO LÓGICO

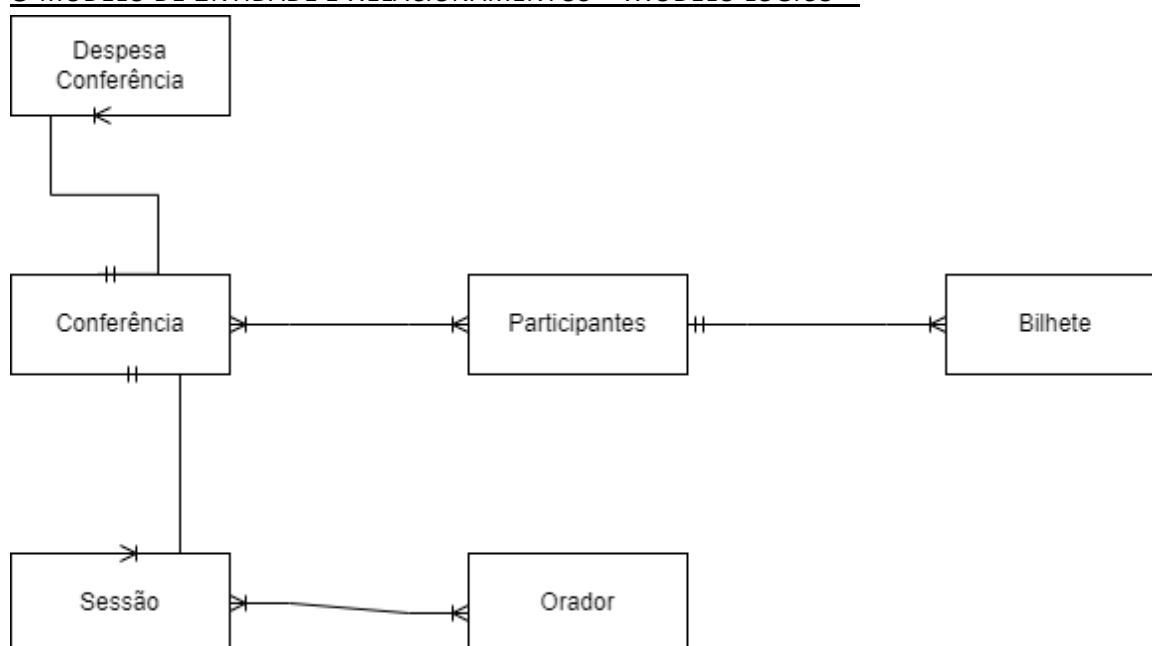


Figura 1 - Modelo de Entidade e Relacionamento

O ESQUEMA DE TABELAS

conferencia (**id_conf**, tema, horário, localidade)

participante (**id_participante**, nome, morada, data_nasc, telefone)

sessao (**id_sessao**, título, horário, descrição, id_conf, sala)

despesaconferencia (**id_despesa**, datadespesa, valordespesa, tipo, id_conf)

bilhete (**id_bilhete**, preço compra, data compra, tipo, método pagamento, id_conf, id_participante)

orador (**id_orador**, nome, profissão, telefone)

O ESQUEMA DE TABELAS NA 3 FN

A normalização de tabelas numa base de dados é um processo que visa organizar e estruturar a informação de forma eficiente e sem redundâncias desnecessárias. Este método ajuda a garantir que os dados estão bem organizados, evitando repetições e melhorando a consistência das informações.

Para isso vamos verificar se as tabelas estão na 3FN:

- **1FN** (Eliminar a repetição de grupos de informação):

Uma relação está na 1FN quando:

- Não existem atributos compostos;
- Não existem atributos multivalores.

Então:

Participante (**id_participante**, nome, rua, nºporta, codPostal, data_nascimento, telefone)

- **2FN** (Evitar a redundância de dados):

Uma relação está na 2FN quando:

- Está na **1FN**;
- Todos os atributos que não sejam chaves dependerem de todo da chave primária.

Então:

confparti (**id_conf, id_participante**)
sessaoorador (**id_sessao, id_orador**)

- **3FN** (Eliminar dependências transitivas):

Uma relação está na 3FN quando:

- Está na **2FN**;
- Não existem dependências funcionais.

tipobilhete (**id_tipobilhete**, precoatual, preco)
metodopagamento (**id_metodo**, método)
tiposala (**id_sala**, tipo)
tipodespesa (**id_tipodespesa**, descrição)

O MODELO RELACIONAL – MODELO FÍSICO

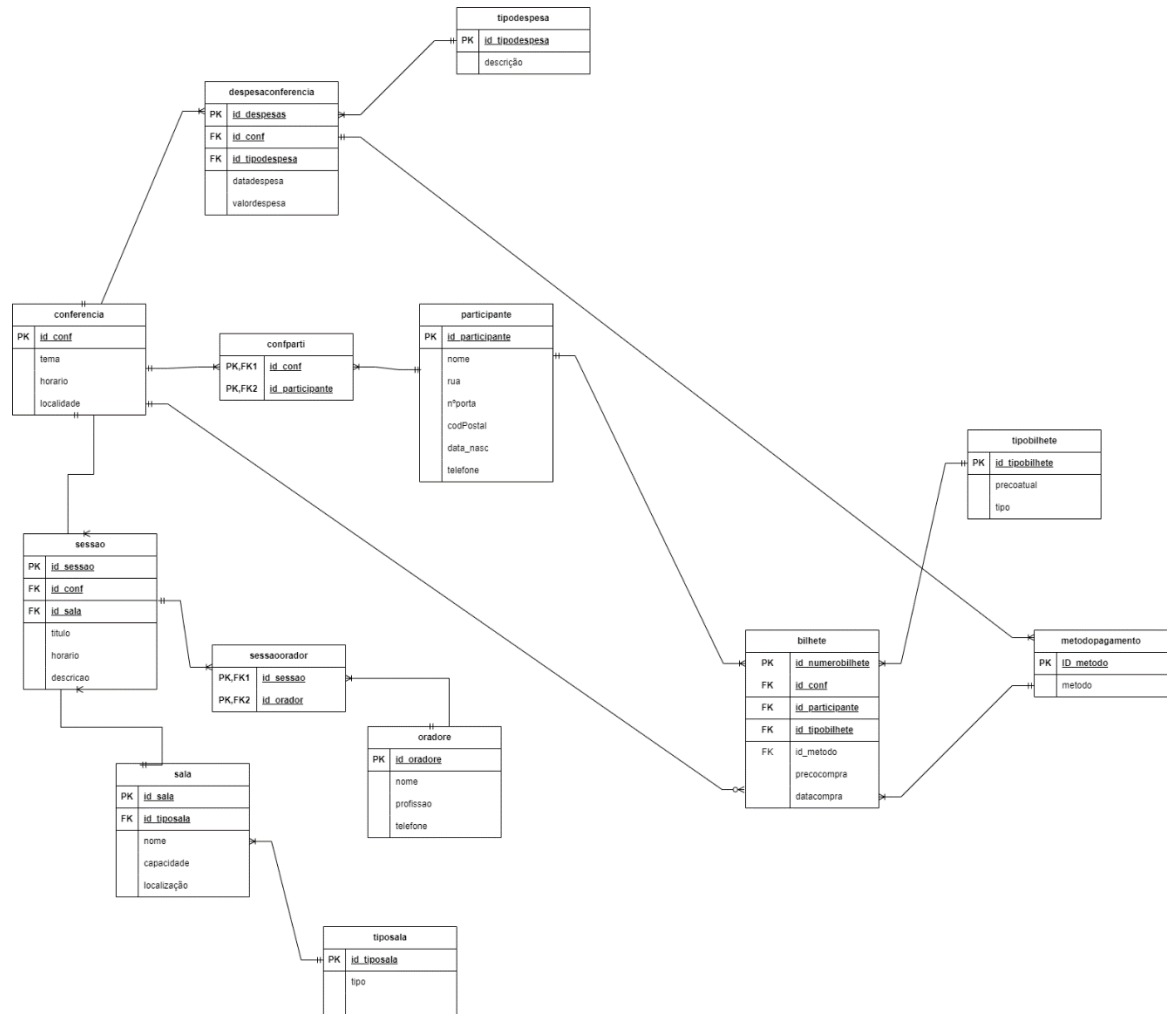


Figura 2 - Modelo Físico Final

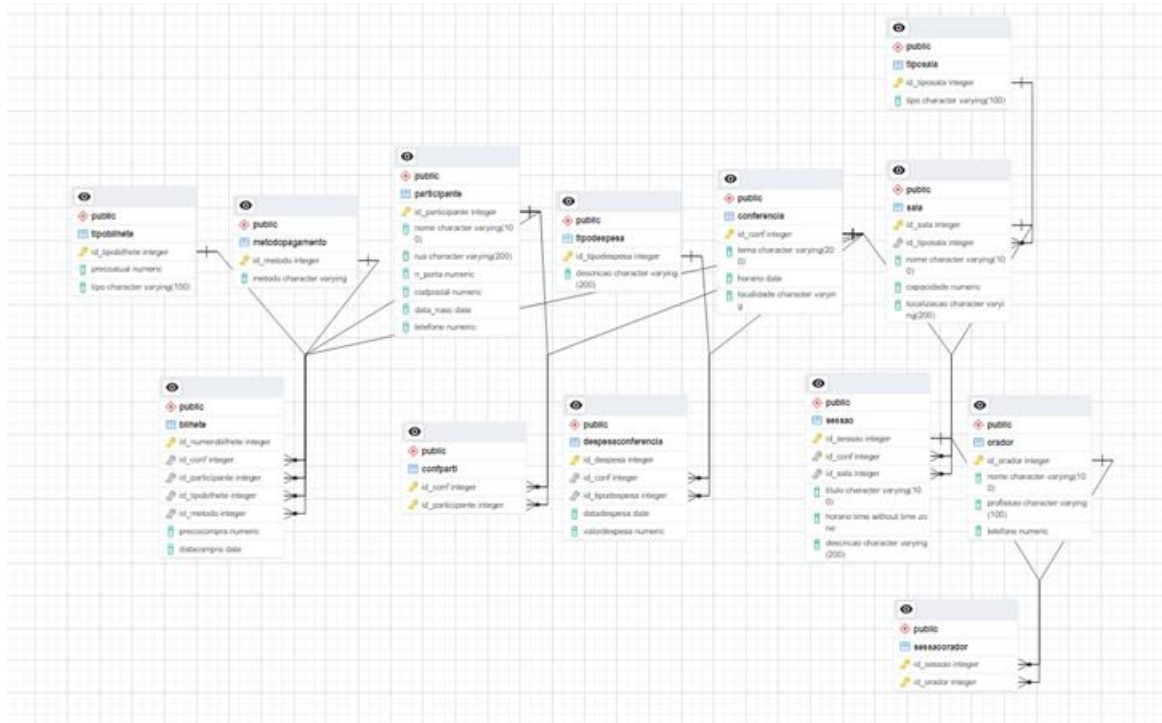


Figura 3 - Tabelas

2.3 IMPLEMENTAÇÃO DA BASE DE DADOS

SCRIPT DE CRIAÇÃO DE TABELAS

CREATE TABLE IF NOT EXISTS **public.tipodespesa**

```
(
    id_tipodespesa          integer          NOT NULL          DEFAULT
    nextval('tipodespesa_id_tipodespesa_seq'::regclass),
    descricao character varying (200) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT tipodespesa_pkey PRIMARY KEY (id_tipodespesa)
);
```

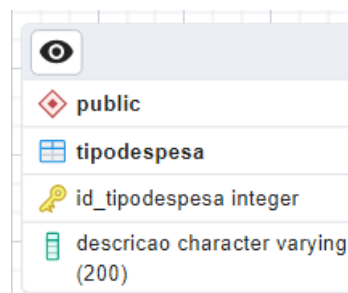


Figura 4 - Tabela "tipodespesa"

```
CREATE TABLE IF NOT EXISTS public.tiposala
(
    id_tiposala integer NOT NULL DEFAULT nextval('tiposala_id_tiposala_seq'::regclass),
    tipo character varying (100) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT tiposala_pkey PRIMARY KEY (id_tiposala)
);
```



Figura 5 - Tabela "tiposala"

```
CREATE TABLE IF NOT EXISTS public.tipobilhete
(
    id_tipobilhete integer NOT NULL DEFAULT
    nextval('tipobilhete_id_tipobilhete_seq'::regclass),
    precoatual numeric NOT NULL,
    tipo character varying(100) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT tipobilhete_pkey PRIMARY KEY (id_tipobilhete)
);
```

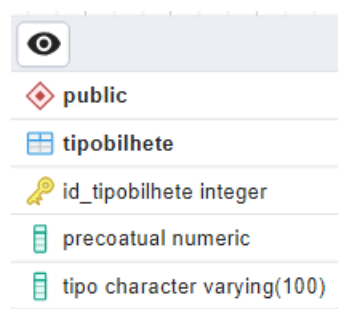


Figura 6 - Tabela "tipobilhete"

```
CREATE TABLE IF NOT EXISTS public.metodopagamento
(
    id_metodo            integer            NOT            NULL            DEFAULT
    nextval('metodopagamento_id_metodo_seq'::regclass),
    metodo character varying COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT metodopagamento_pkey PRIMARY KEY (id_metodo)
);
```



Figura 7 - Tabela "metodopagamento"

```
CREATE TABLE IF NOT EXISTS public.orador
(
    id_orador integer NOT NULL DEFAULT nextval('orador_id_orador_seq'::regclass),
    nome character varying(100) COLLATE pg_catalog."default" NOT NULL,
    profissao character varying(100) COLLATE pg_catalog."default" NOT NULL,
    telefone numeric NOT NULL,
    CONSTRAINT orador_pkey PRIMARY KEY (id_orador)
);
```

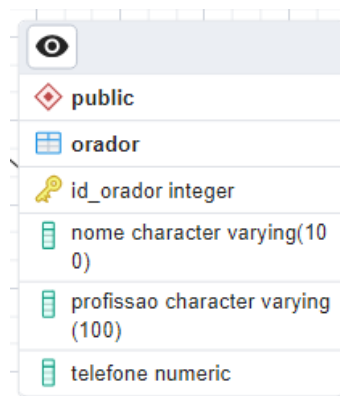


Figura 8 - Tabela "orador"

```
CREATE TABLE IF NOT EXISTS public.participante
(
    id_participante      integer            NOT            NULL            DEFAULT
    nextval('participante_id_participante_seq'::regclass),
    nome character varying(100) COLLATE pg_catalog."default" NOT NULL,
    rua character varying(200) COLLATE pg_catalog."default" NOT NULL,
```

```

n_porta numeric NOT NULL,
codpostal numeric NOT NULL,
data_nasc date NOT NULL,
telefone numeric NOT NULL,
CONSTRAINT participante_pkey PRIMARY KEY (id_participante)
);

```

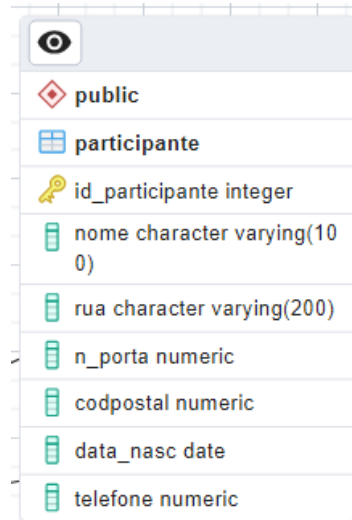


Figura 9 - Tabela "participante"

```

CREATE TABLE IF NOT EXISTS public.conferencia
(
    id_conf integer NOT NULL DEFAULT nextval('conferencia_id_conf_seq'::regclass),
    tema character varying(200) COLLATE pg_catalog."default" NOT NULL,
    horario date NOT NULL,
    localidade character varying COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT conferencia_pkey PRIMARY KEY (id_conf)
);

```

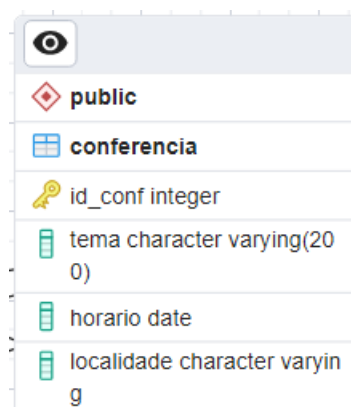


Figura 10 - Tabela "conferencia"

```

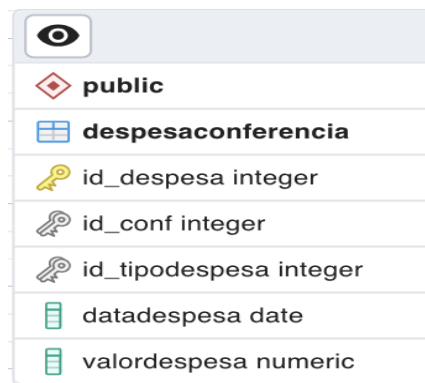
CREATE TABLE IF NOT EXISTS public.despesaconferencia
(
    id_despesa integer NOT NULL DEFAULT
    nextval('despesaconferencia_id_despesa_seq'::regclass),

```

```

id_conf integer NOT NULL,
id_tipodespesa integer NOT NULL,
datadespesa date NOT NULL,
valordespesa numeric NOT NULL,
CONSTRAINT despesaconferencia_pkey PRIMARY KEY (id_despesa),
CONSTRAINT fk_id_conf FOREIGN KEY (id_conf)
    REFERENCES public.conferencia (id_conf) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID,
CONSTRAINT fk_if_tipodespesa FOREIGN KEY (id_tipodespesa)
    REFERENCES public.tipodespesa (id_tipodespesa) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID
);

```



Schema	Table Name	Column Name	Data Type	Constraints
public	despesaconferencia	id_despesa	integer	PRIMARY KEY
public	despesaconferencia	id_conf	integer	FOREIGN KEY (REFERENCES public.conferencia (id_conf) MATCH SIMPLE ON UPDATE NO ACTION ON DELETE NO ACTION NOT VALID)
public	despesaconferencia	id_tipodespesa	integer	FOREIGN KEY (REFERENCES public.tipodespesa (id_tipodespesa) MATCH SIMPLE ON UPDATE NO ACTION ON DELETE NO ACTION NOT VALID)
public	despesaconferencia	datadespesa	date	
public	despesaconferencia	valordespesa	numeric	

Figura 11 - Tabela "despesaconferencia"

```

CREATE TABLE IF NOT EXISTS public.sala
(
    id_sala integer NOT NULL DEFAULT nextval('sala_id_sala_seq'::regclass),
    id_tiposala integer NOT NULL,
    nome character varying(100) COLLATE pg_catalog."default" NOT NULL,
    capacidade numeric NOT NULL,
    localizacao character varying(200) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT sala_pkey PRIMARY KEY (id_sala),
    CONSTRAINT fk_id_tiposala FOREIGN KEY (id_tiposala)
        REFERENCES public.tiposala (id_tiposala) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID);

```









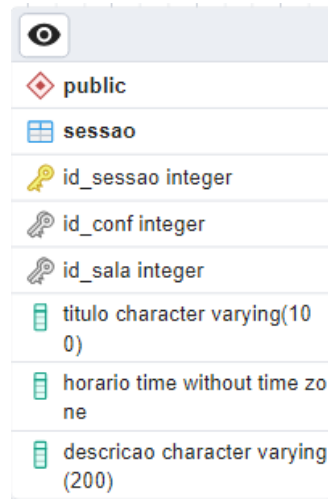

 public
 sala
 id_sala integer
 id_tiposala integer
 nome character varying(100)
 capacidade numeric
 localizacao character varying(200)

Figura 12' - Tabela "sala"

```
CREATE TABLE IF NOT EXISTS public.sessao
(
    id_sessao integer NOT NULL DEFAULT nextval('sessao_id_sessao_seq'::regclass),
    id_conf integer NOT NULL,
    id_sala integer NOT NULL,
    titulo character varying(100) COLLATE pg_catalog."default" NOT NULL,
    horario time without time zone NOT NULL,
    descricao character varying(200) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT sessao_pkey PRIMARY KEY (id_sessao),
    CONSTRAINT fk_id_conf FOREIGN KEY (id_conf)
        REFERENCES public.conferencia (id_conf) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT fk_id_sala FOREIGN KEY (id_sala)
        REFERENCES public.sala (id_sala) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
);
```



Schema	Table	Column	Data Type	Constraints
public	sessao	id_sessao	integer	Primary Key
		id_conf	integer	Foreign Key
		id_sala	integer	Foreign Key
		titulo	character varying(100)	
		horario	time without time zone	
		descricao	character varying(200)	

Figura 13 - Tabela "sessao"

```
CREATE TABLE IF NOT EXISTS public.sessaorador
(
    id_sessao integer NOT NULL DEFAULT nextval('sessaorador_id_sessao_seq'::regclass),
    id_orador integer NOT NULL DEFAULT
nextval('sessaorador_id_orador_seq'::regclass),
    CONSTRAINT sessaorador_pkey PRIMARY KEY (id_sessao, id_orador),
    CONSTRAINT fk_id_orador FOREIGN KEY (id_orador)
        REFERENCES public.orador (id_orador) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT fk_id_sessao FOREIGN KEY (id_sessao)
        REFERENCES public.sessao (id_sessao) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
);
```



Schema	Table	Column	Data Type	Constraints
public	sessaorador	id_sessao	integer	Primary Key
		id_orador	integer	Primary Key

Figura 14 - Tabela "sessaorador"

```
CREATE TABLE IF NOT EXISTS public.confparti
(
    id_conf integer NOT NULL DEFAULT nextval('confparti_id_conf_seq'::regclass),
    id_participante integer NOT NULL DEFAULT
nextval('confparti_id_participante_seq'::regclass),
    CONSTRAINT confparti_pkey PRIMARY KEY (id_conf, id_participante),
```

```

CONSTRAINT fk_id_conf FOREIGN KEY (id_conf)
REFERENCES public.conferencia (id_conf) MATCH SIMPLE
ON UPDATE NO ACTION
ON DELETE NO ACTION
NOT VALID,
CONSTRAINT fk_id_participante FOREIGN KEY (id_participante)
REFERENCES public.participante (id_participante) MATCH SIMPLE
ON UPDATE NO ACTION
ON DELETE NO ACTION
NOT VALID

```

);



Figura 15 - Tabela "confparti"

```

CREATE TABLE IF NOT EXISTS public.bilhete
(
    id_numerobilhete          integer          NOT          NULL          DEFAULT
    nextval('bilhete_id_numerobilhete_seq'::regclass),
    id_conf integer NOT NULL,
    id_participante integer NOT NULL,
    id_tipobilhete integer NOT NULL,
    id_metodo integer NOT NULL,
    precocompra numeric NOT NULL,
    datacompra date NOT NULL,
    CONSTRAINT bilhete_pkey PRIMARY KEY (id_numerobilhete),
    CONSTRAINT fk_id_conf FOREIGN KEY (id_conf)
    REFERENCES public.conferencia (id_conf) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID,
    CONSTRAINT fk_id_metodo FOREIGN KEY (id_metodo)
    REFERENCES public.metodopagamento (id_metodo) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION

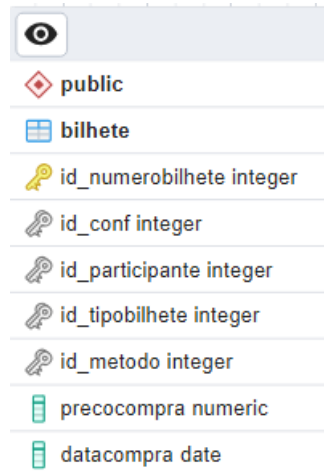
```



```

NOT VALID,
CONSTRAINT fk_id_participante FOREIGN KEY (id_participante)
REFERENCES public.participante (id_participante) MATCH SIMPLE
ON UPDATE NO ACTION
ON DELETE NO ACTION
NOT VALID,
CONSTRAINT fk_id_tipobilhete FOREIGN KEY (id_tipobilhete)
REFERENCES public.tipobilhete (id_tipobilhete) MATCH SIMPLE
ON UPDATE NO ACTION
ON DELETE NO ACTION
NOT VALID
);

```



Schema	Table	Column	Data Type
public	bilhete	id_numerobilhete	integer
		id_conf	integer
		id_participante	integer
		id_tipobilhete	integer
		id_metodo	integer
		precocompra	numeric
		datacompra	date

RESTRICÇÕES DE INTEGRIDADE

Figura 16 - Tabela "bilhete"

- **Chave Primária (Primary Key)**

Tem como objetivo garantir que numa tabela seja exclusivamente identificada por uma chave única.

Exemplo:

```

CREATE TABLE IF NOT EXISTS public.tiposala
(
    id_tiposala integer NOT NULL DEFAULT
    nextval('tiposala_id_tiposala_seq'::regclass),
    tipo character varying (100) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT tiposala_pkey PRIMARY KEY (id_tiposala)
);

```

- **Chave Estrangeira (Foreign Key)**

Manter relação entre duas tabelas, para que os valores de uma coluna correspondam aos valores de outra coluna.

Exemplo:

```
CREATE TABLE IF NOT EXISTS public.sala
(
    id_sala integer NOT NULL DEFAULT nextval('sala_id_sala_seq'::regclass),
    id_tiposala integer NOT NULL,
    nome character varying(100) COLLATE pg_catalog."default" NOT NULL,
    capacidade numeric NOT NULL,
    localizacao character varying(200) COLLATE pg_catalog."default" NOT
NULL,
    CONSTRAINT sala_pkey PRIMARY KEY (id_sala),
    CONSTRAINT fk_id_tiposala FOREIGN KEY (id_tiposala)
);
```

- **Restrição de Verificação (Check Constraint)**

Garante que os dados introduzidos satisfaçam aos critérios específicos.

Exemplo: Adicionar uma restrição de verificação a um atributo de uma tabela para só seja possível adicionar se satisfazer a restrição.

- **Restrição NOT NULL**

Garante que os valores introduzidos não sejam nulos.

Exemplo:

```
CREATE TABLE IF NOT EXISTS public.sala
(
    id_sala integer NOT NULL DEFAULT nextval('sala_id_sala_seq'::regclass),
    id_tiposala integer NOT NULL,
);
```

SCRIPT DE CRIAÇÃO DE VISTAS

Este comando SQL cria uma visão denominada "**view_participante**" com base numa consulta à tabela "**participante**". A visão resultante irá conter apenas as colunas "**nome**", "**rua**" e "**telefone**" das linhas em que a data de nascimento ("**data_nasc**") é igual ou posterior a '2000-01-01':

- Create view view_participante as select nome,rua,telefone from participante where data_nasc>='2000-01-01';

Este comando SQL cria uma visão designada por "**view_sala_ordem_capacidade**" mediante uma consulta à tabela "**sala**". Na visão gerada, estarão presentes as colunas "**id_sala**", "**nome**", "**capacidade**" e "**localização**" de todas as entradas da tabela "**sala**", organizadas de maneira decrescente de acordo com a informação contida na coluna "**capacidade**":

- Create view view_sala_ordem_capacidade as select id_sala,nome,capacidade,localização from sala order by capacidade desc;

2.4 GESTÃO DOS DADOS NA BASE DE DADOS

A base de dados pode ser usada para armazenar a informação por um longo período, para isso deverá ser possível inserir, atualizar, eliminar e consultar os dados armazenados.

INSERÇÃO DE DADOS

- Insert into Orador (ID_orador,nome,profissão,telefone) values ('101','Manuel','Data Base Maneger','123456789');

Figura 17 - Inserção de dados

```
trabalhodados=# select * from orador where id_orador=101;
 id_orador | nome |      profissao      | telefone
-----+-----+-----+-----
      101 | Manuel | Data Base Maneger | 123456789
(1 row)
```

ATUALIZAÇÃO DE DADOS

Este comando atualiza o valor da coluna "**tema**" para '**Cripto**' apenas no registo onde o "**id_conf**" é igual a 22 na tabela "**conferencia**". Isto pode ser útil quando for preciso modificar informações específicas em um registo com base em critérios específicos:

- Update conferencia set tema = 'Cripto' where id_conf=22;

id_conf	tema	horario	localidade
22	Cripto	2024-01-02	Moose Jaw

(1 row)

Figura 18 - Atualização de dados

ELIMINAÇÃO DE DADOS

Este comando SQL é empregue para remover registos da tabela "**orador**" nos quais o valor da coluna "**id_orador**" é igual a 101:

- Delete from orador where id_orador = 101;

98	Orlando Lingerfeldt	Systems Administrator I	504770022
99	Moyna Ginnaly	Social Worker	5954405293
100	Valry Brandsma	VP Quality Control	4344525105

(100 rows)

Figura 19 - Eliminação de dados

CONSULTAS SIMPLES

Este comando SQL efetua uma consulta na tabela "**participante**" e apresenta todas as colunas de cada linha em que o campo da coluna "**nome**" não está vazio:

- Select * from participante where nome is not null;

id_participante	nome	rua	n_porta	codpostal	data_nasc	telefone
1	Sophey Cornes	Schiller	977	4929897	1914-02-15	5697762003
2	Townie Sives	Division	4005	4947547	1916-03-30	8203607549
3	Bonnie Steptow	Graedel	94	4975128	1903-05-17	1043537433
4	Darline Jurek	Erie	88	4915210	1928-10-04	8399852941
5	Holly Moyne	Parkside	582	4951464	1971-02-21	8364115158
6	Merrill Bertrand	Dennis	98379	4969502	1912-07-15	5751508595
7	Dorothy Johann	Eagle Crest	35101	4919716	1911-04-30	7497406830
8	Aldin Waddington	Schlimgen	5	4913619	1941-04-12	1137320660
9	Solomon Hassell	Arrowood	3	4965572	1935-11-06	2444076567
10	Reynolds Gammidge	Derek	51494	4929459	1902-01-19	3407622647
11	Briny Cavie	Acker	5552	4953814	1918-02-16	8771226234
12	Garrot Janecek	Bobwhite	8602	4902971	1961-03-12	4679676924
13	Lissie Lippard	Nelson	3797	4942252	1939-12-27	7891317250
14	Anne-marie Myhan	Fieldstone	298	4910912	1916-02-19	1381456334
15	Kelly Crofts	Tennessee	4338	4941660	1943-12-03	4784606233
16	Daniella Smullen	Florence	455	4923775	1916-11-20	4193043968
17	Ellissa Mounsey	Mandrake	4232	4916835	1980-05-21	1558052127

Figura 20 - Consulta 1

Este comando SQL realiza uma consulta na tabela “**conferencia**” e retorna todas as colunas de todas as linhas onde a coluna “**localidade**” começa com a letra 'A':

- Select * from conferencia where localidade like 'A%';

```
trabalhadores=# select * from conferencia where localidade like 'A%';
```

id_conf	tema	horario	localidade
6	Dazzlesphere	2024-01-09	Astorga
33	Roombo	2024-01-01	Aulnay-sous-Bois
43	Jaxworks	2024-01-19	Ambilobe
47	Lazzy	2024-01-30	At-Bashi

(4 rows)

Figura 21 - Consulta 2

O comando SQL a seguir realiza uma consulta na tabela “**bilhete**” e extrai o valor máximo presente na coluna “**precocompra**”:

- Select MAX (precocompra) FROM bilhete;

max
4.99

(1 row)

Figura 22 - Consulta 3

Este comando SQL procura a data de compra mais antiga registrada na tabela “**bilhete**”:

- Select MIN (datacompra) FROM bilhete;

min
2023-12-20

(1 row)

Figura 23 - Consulta 4

Este comando SQL efetua uma consulta na tabela “**sessao**”, trazendo todas as colunas de todas as linhas em que o valor na coluna “**id_sessao**” está no intervalo entre 35 e 42, incluindo ambos os extremos:

- Select * from sessao where id_sessao between ‘35’ and ‘42’;

id_sessao	id_conf	id_sala	descricao	titulo	horario
35	46	12	Amblyodon Moss		00:11:00
36	49	12	Drooping Star Of Bethlehem	10:04:00	ultrices posuere cubilia curae donec pharetra magna vestibulum aliquet ultrices erat
37	26	3	Medusa Clover	16:00:00	sed interdum venenatis turpis enim blandit mi in porttitor pede justo eu massa donec
38	5	40	Robust Saltbush	17:17:00	massa volutpat convallis morbi odio odio elementum eu interdum eu tincidunt in leo
39	27	33	Menzies' Anacolia Moss	16:23:00	sed augue aliquam erat volutpat in congue etiam justo etiam pretium iaculis justo in
40	6	50	Queen-devil Hawkweed	00:37:00	vivamus metus arcu adipiscing molestie hendrerit at vulputate vitae nisl aenean lectus
41	48	1	Purple Poppymallow	23:01:00	non pretium quis lectus suspendisse potenti in eleifend quam a odio in hac habitasse
42	48	1	Purple Poppymallow	23:01:00	non pretium quis lectus suspendisse potenti in eleifend quam a odio in hac habitasse

Figura 24 - Consulta 5

À semelhança do comando anterior, este comando SQL efetua uma consulta na tabela "sessao", devolvendo todas as colunas de cada linha onde o valor na coluna "id_sessao" não se encontra no intervalo entre 35 e 42:

- `Select * from sessao where id_sessao not between '35' and '42';`

id_sessao	id_conf	id_sala	descricao	titulo	horario	
1	46	41	Cathedral Bells		14:37:00	vel nisl duis ac nibh fusce lacus purus aliquet at feugiat non pretium quis
2	25	21	Lecidea Lichen		11:56:00	at turpis donec posuere metus vitae ipsum aliquam non mauris morbi non lectu
3	30	17	Vine Hill Clarkia		02:54:00	fusce consequat nulla nisl nunc nisl duis bibendum felis sed interdum venena
4	26	26	Slender False Brome		01:42:00	eros elementum pellentesque quisque porta volutpat erat quisque erat eros vi
5	18	27	Beard Lichen		01:07:00	diam erat fermentum justo nec condimentum neque sapien placerat ante nulla j
6	23	28	Blue Mountain Catchfly		21:24:00	tempus vivamus in felis eu sapien cursus vestibulum proin eu mi nulla ac eni
7	30	46	Tiger Orchid		14:55:00	nunc rhoncus dui vel sem sed sagittis nam congue risus semper porta volutpat
8	20	48	'ohe Wiko 'ola		04:32:00	orci eget orci vehicula condimentum curabitur in libero ut massa volutpat co
9	22	42	Narrowfruit Horned Beaksedge		10:45:00	mauris morbi non lectus aliquam sit amet diam in magna bibendum imperdiet nu

Figura 25 - Consulta 6

O comando a seguir realiza uma consulta na tabela "sessao" e retorna todas as colunas de todas as linhas onde o valor na coluna "id_sala" é igual a 6 ou 7:

- `Select * from sessao where id_sala in (6,7);`

id_sessao	id_conf	id_sala	descricao	titulo	horario	
15	7	7	Blindia Moss		09:55:00	libero quis orci nullam molestie nibh in lectus pellentesque at nulla suspendisse potenti
18	31	7	Sweetleaf		03:23:00	eros viverra eget congue eget semper rutrum nulla nunc purus phasellus in felis donec sem
64	25	7	Japanese Violet		19:24:00	primis in faucibus orci luctus et ultrices posuere cubilia curae mauris viverra diam vitae
123	39	6	Buckwheat		09:09:00	nisl aenean lectus pellentesque eget nunc donec quis orci eget orci vehicula condimentum c
124	23	7	Toothed Onion		17:36:00	primis in faucibus orci luctus et ultrices posuere cubilia curae mauris viverra diam vitae
149	23	6	Yerba		17:03:00	odio condimentum id luctus nec molestie sed justo pellentesque viverra pede ac diam cras p
170	50	7	Wavyleaf Beeblossom		12:30:00	mi in porttitor pede justo eu massa donec dapibus duis at velit eu est congue elementum in

Figura 26 - Consulta 7

- `Select count (*) from bilhete;`

count
600
(1 row)

Figura 27 - Consulta 8

CONSULTAS COM JOIN DE TABELAS

Consultas com:

- Group by

Este comando SQL efetua uma consulta na tabela "**orador**" e apresenta duas colunas: uma com os valores distintos da coluna "**profissao**" e outra com a contagem de ocorrências para cada valor de "**profissao**":

- `Select profissao,count(*) from orador group by profissao;`

```
trabalhados=# select profissao,count(*) from orador group by profissao;
```

profissao	count
Occupational Therapist	2
Social Worker	2
Food Chemist	2
Human Resources Manager	1
Health Coach IV	1
Account Coordinator	1
Quality Engineer	1
Associate Professor	3
GIS Technical Architect	2
Database Administrator II	1
Sales Associate	1
Office Assistant III	1
Operator	1
Health Coach III	1

Figura 28 - Consulta com "group by"

- Having

Este comando SQL realiza uma consulta na tabela "**despesaconfidencia**", devolvendo duas colunas: "**id_tipodespesa**" e a soma dos valores presentes na coluna "**valordespesa**", agrupados pela coluna "**id_tipodespesa**". "HAVING" é utilizado para filtrar os resultados, incluindo apenas aqueles em que a soma dos valores de despesa (valordespesa) é superior a 5000:

- `Select id_tipodespesa, sum(valordespesa) from despesaconfidencia group by id_tipodespesa having sum(valordespesa) > 5000;`

```
trabalhados=# select id_tipodespesa, sum(valordespesa) from despesaconfidencia group by id_tipodespesa having sum(valordespesa) > 5000;
```

id_tipodespesa	sum
27	7926.88
23	9884.14
56	5462.12
91	5048.14
8	21656.60
87	6996.87
74	7329.12
54	14876.84
29	18540.90
71	8710.76
68	9664.47
4	6168.23
34	13295.4
96	14636.88
52	7436.66
88	7869.83
70	5328.24
67	7595.66
63	5839.38
10	13693.08
35	13070.53
6	6412.59
86	16190.81
93	7582.35
36	5671.59
69	7976.56
50	12981.70
60	7366.83
14	15685.47
22	2288.24

Figura 29 - Consulta com "having"

- Order by

Este comando SQL efetua uma consulta à tabela "**participante**", apresentando todas as colunas de cada registo, organizadas em ordem alfabética crescente com base na coluna "**nome**":

- Select * from participante order by nome;

```
trabalhados=# select * from participante order by nome;
```

id_participante	nome	rua	n_porta	codpostal	data_nasc	telefone
140	Abby Yurlov	Cherokee	7875	4900256	1988-12-15	1941085796
393	Adele Corzon	Calypso	676	4900964	1901-11-21	9887702228
196	Adelheid MacNelly	West	969	4965130	1953-10-31	6765982565
297	Adiana Lackemann	Barnett	3	4950566	1945-12-05	5903938084
235	Adrian Eyree	John Wall	355	4927555	1987-06-20	5745628922
75	Agathe Loveridge	Hollow Ridge	84	4971361	1974-02-22	4368025481
156	Ailsun Eplate	Grayhawk	8020	4997834	1924-11-12	2715443258
250	Aimee Durden	Cambridge	6	4906350	1940-12-18	6141003137
8	Aldin Waddington	Schlimgen	5	4913619	1941-04-12	1137320660
267	Alessandro Huchot	Di Loreto	8583	4983950	1905-05-14	5201309931
163	Alessandro Tybalt	Bunker Hill	5375	4997260	1966-02-05	7147805282
102	Alex O'Hartigan	Novick	357	4904621	1939-11-30	1214786266
408	Alfonso Houlston	Crest Line	45	4982954	1969-07-21	3443938050
134	Alfy Stanwix	Sycamore	636	4920679	1911-11-05	6307102125
402	Algernon Minot	Southridge	86149	4933613	1930-05-27	6304390531
493	Allen Becks	Doe Crossing	217	4936410	1954-04-03	4458280621
436	Allyn Diggle	Waywood	792	4986649	1979-08-23	9623884697

Figura 30 - Consulta com "order by"

CONSULTAS RECORRENDO AO USO DAS VISTAS

Este comando SQL faz uma pesquisa na visão denominada "**view_participante**". Anteriormente, essa visão foi criada com base numa análise à tabela "**participante**", selecionando apenas as colunas "**nome**", "**rua**" e "**telefone**" das linhas onde a data de nascimento ("**data_nasc**") é igual ou posterior a '2000-01-01'. A instrução **SELECT *** recolhe todas as colunas disponíveis na visão. Dessa forma, o resultado desta consulta consistirá em todas as colunas de todas as entradas presentes na visão "**view_participante**":

```
trabalhados=# create view view_participante as select nome,rua,telefone from participante where data_nasc>='2000-01-01';
CREATE VIEW
trabalhados=# select * from view_participante;
```

nome	rua	telefone
Michaeline Lamberts	Acker	5323311055
Doralia Piffe	Blaine	4886731629
Nefen Guilliland	Summer Ridge	1951204742
Carine Patron	Jenna	9296060884
Ambur Kenyon	Emmet	9367351279
Hadria Sycamore	Monument	1737577632
Betteann O'Heagertie	Darwin	2894428622
Nancy Pacitti	Summerview	5421253347
Levy Latus	Coolidge	8231398991
Katinka Mcsarry	Randy	6441353091
Dorice Rubel	Ohio	4336577661
Inge Madeley	Dahle	4373859241
Kerwin Levison	Sullivan	9662479752
Merrielle Jerosch	Pine View	2485193551
Aurore Lutsch	Orin	4394072256
Marcos Chesnut	Browning	7861881260
Bennie MacRanald	Amoth	8152578626
Danny Galler	Eggdant	5875547492
Elayne Stronough	Michigan	4885361822
Matilda Muttitt	Monterey	6003689484
Pall Dondon	Acker	3239411780

(21 rows)

Figura 31 - Consulta com a view "view_participante"

Este comando SQL efetua uma pesquisa na visão intitulada “view_sala_orden_capacidade”.

Dessa forma, o resultado desta consulta incluirá todas as colunas de todos os registos presentes na visão "view_sala_orden_capacidade", apresentados em ordem decrescente de capacidade.

```
trabalhadores=# create view view_sala_orden_capacidade as select id_sala,nome,capacidade,localizacao from sala order by capacidade desc;
CREATE VIEW
trabalhadores=# select * from view_sala_orden_capacidade;
```

id_sala	nome	capacidade	localizacao
20	Subin	99	Apt 200
21	Vagram	97	Apt 477
26	Y-find	97	Suite 82
39	Y-Solowarm	96	Apt 1801
27	Tampflex	95	17th Floor
24	Tampflex	91	Room 1043
11	Domainer	91	Apt 96
42	Duoban	89	Suite 93
8	Zaan-Dox	89	Apt 364
17	Konklab	88	Room 1342
32	Greenlam	86	11th Floor
30	Prodder	85	Apt 1100
47	Zontrax	85	Suite 76
38	Temp	83	Room 830
23	Tresom	81	Suite 12
41	Pannier	78	PO Box 53022
2	Lotlux	77	PO Box 86166
13	Andalax	74	3rd Floor
15	Fix San	74	Room 649
29	Tin	72	Suite 17
14	Sonsing	71	Room 70
35	Tresom	69	Room 452
7	Zamit	68	Suite 47
46	Zontrax	64	Apt 1540

Figura 32 - Consulta com a view "view_sala_orden_capacidade"

TRIGGERS

Os **triggers** são utilizados para assegurar que certas ações ocorram automaticamente em resposta a alterações nos dados, preservando a integridade da base de dados ou realizando tarefas específicas sempre que condições particulares são satisfeitas.

Eis um Trigger para verificar se a data de nascimento corresponde á condição do mesmo:

```
CREATE OR REPLACE TRIGGER tri_participante
BEFORE INSERT OR UPDATE ON participante
FOR EACH ROW
BEGIN

    IF NEW.data_nasc >= '2000-01-01' THEN
        -- Realizar ação desejada, por exemplo, imprimir uma mensagem ou lançar
        uma exceção
        DBMS_OUTPUT.PUT_LINE (' data nascimento valida.');
```



```
        RAISE_APPLICATION_ERROR (-20001, 'Data de nascimento inválida. A data
deve ser anterior a 01/01/2000.');
```



```
    END IF;
END;
```

3. CONCLUSÃO

Com a execução deste projeto prático de bases de dados, foi aplicado de forma prática todo o conhecimento adquirido nas aulas da Unidade Curricular, onde foram elaborados diagramas de entidade-relacionamento, diagramas relacionais de tabelas e modelos relacionais. Adicionalmente, procedeu-se à implementação de uma base de dados utilizando o software PostgreSQL, recorrendo à linguagem SQL para administrar e exemplificar o funcionamento da base de dados.

Além dos conhecimentos previamente adquiridos em sala de aula, foi possível expandir significativamente a compreensão sobre o tema de Base de Dados. Para garantir a execução de uma base de dados funcional e alinhada com os objetivos do trabalho, foi necessário aprofundar a investigação nessa área, explorando conceitos mais avançados. Este processo não apenas enriqueceu o entendimento existente, mas também proporcionou uma boa oportunidade para a aplicação prática de novas abordagens e técnicas, contribuindo assim para uma experiência de aprendizado mais abrangente.

Com o desfecho deste projeto prático de bases de dados, não apenas consolidamos a aplicação prática do conhecimento adquirido nas aulas da Unidade Curricular, mas também testemunhamos a evolução contínua da nossa compreensão no domínio das bases de dados.

REFERÊNCIAS

- [1] Pdfs disponibilizados pelos docentes da disciplina, 18/12/2023
- [2] Mockaroo, 19/12/2023, <https://www.mockaroo.com>