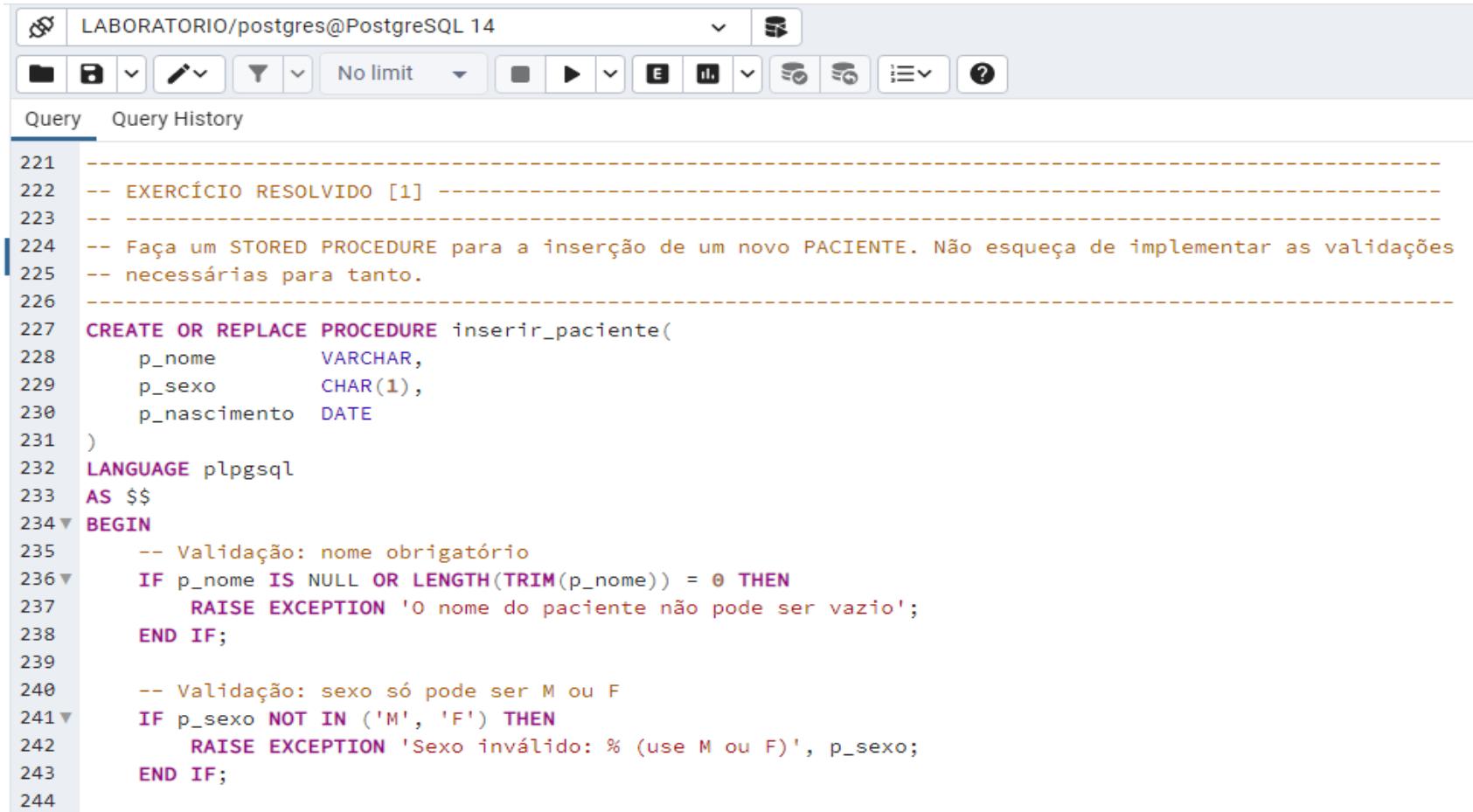


Banco de Dados 2

07 – Segurança.

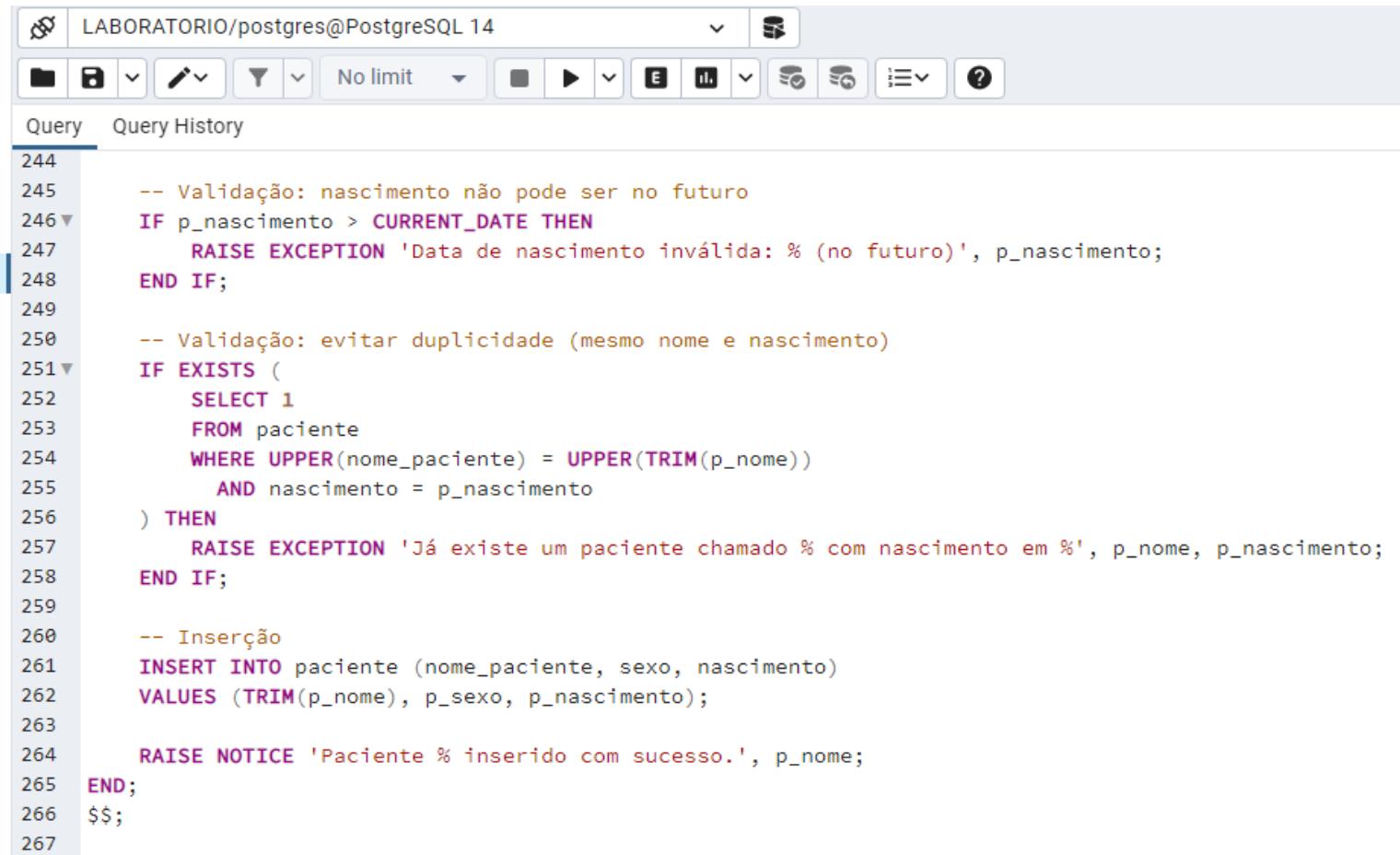
Exercícios Resolvidos



The screenshot shows a PostgreSQL query editor interface. The title bar indicates the connection is to 'LABORATORIO/postgres@PostgreSQL 14'. The toolbar includes various icons for file operations, search, and execution. Below the toolbar, the tabs 'Query' and 'Query History' are visible, with 'Query' being the active tab.

```
221 ---  
222 -- EXERCÍCIO RESOLVIDO [1] ---  
223 ---  
224 -- Faça um STORED PROCEDURE para a inserção de um novo PACIENTE. Não esqueça de implementar as validações  
225 -- necessárias para tanto.  
226 ---  
227 CREATE OR REPLACE PROCEDURE inserir_paciente(  
228     p_nome        VARCHAR,  
229     p_sexo        CHAR(1),  
230     p_nascimento  DATE  
231 )  
232 LANGUAGE plpgsql  
233 AS $$  
234 BEGIN  
235     -- Validação: nome obrigatório  
236     IF p_nome IS NULL OR LENGTH(TRIM(p_nome)) = 0 THEN  
237         RAISE EXCEPTION 'O nome do paciente não pode ser vazio';  
238     END IF;  
239  
240     -- Validação: sexo só pode ser M ou F  
241     IF p_sexo NOT IN ('M', 'F') THEN  
242         RAISE EXCEPTION 'Sexo inválido: % (use M ou F)', p_sexo;  
243     END IF;  
244
```

Exercícios Resolvidos



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

- Connection:** LABORATORIO/postgres@PostgreSQL 14
- Toolbar:** Includes icons for file operations, search, and various database functions.
- Tab:** The "Query" tab is selected, while "Query History" is also visible.
- Code Content:** A PostgreSQL script (script.sql) containing the following code:

```
244
245    -- Validação: nascimento não pode ser no futuro
246    IF p_nascimento > CURRENT_DATE THEN
247        RAISE EXCEPTION 'Data de nascimento inválida: % (no futuro)', p_nascimento;
248    END IF;
249
250    -- Validação: evitar duplicidade (mesmo nome e nascimento)
251    IF EXISTS (
252        SELECT 1
253        FROM paciente
254        WHERE UPPER(nome_paciente) = UPPER(TRIM(p_nome))
255            AND nascimento = p_nascimento
256    ) THEN
257        RAISE EXCEPTION 'Já existe um paciente chamado % com nascimento em %', p_nome, p_nascimento;
258    END IF;
259
260    -- Inserção
261    INSERT INTO paciente (nome_paciente, sexo, nascimento)
262    VALUES (TRIM(p_nome), p_sexo, p_nascimento);
263
264    RAISE NOTICE 'Paciente % inserido com sucesso.', p_nome;
265
266
267$$;
```

Exercícios Resolvidos

```
268  
269 CALL inserir_paciente('', 'M', '1990-05-10'); -- ERRO: nome vazio  
270 CALL inserir_paciente('João Pedro', 'X', '2010-03-05'); -- ERRO: sexo inválido  
271 CALL inserir_paciente('Maria Silva', 'F', '2030-01-01'); -- ERRO: data no futuro
```

Data output Messages

```
ERROR: O nome do paciente não pode ser vazio  
CONTEXT: função PL/pgSQL inserir_paciente(character varying,character,date) linha 5 em RAISE  
SQL state: P0001
```

Exercícios Resolvidos

```
268  
269 CALL inserir_paciente('', 'M', '1990-05-10'); -- ERRO: nome vazio  
270 CALL inserir_paciente('João Pedro', 'X', '2010-03-05'); -- ERRO: sexo inválido  
271 CALL inserir_paciente('Maria Silva', 'F', '2030-01-01'); -- ERRO: data no futuro
```

Data output Messages

```
ERROR: Sexo inválido: X (use M ou F)  
CONTEXT: função PL/pgSQL inserir_paciente(character varying,character,date) linha 10 em RAISE  
SQL state: P0001
```

Exercícios Resolvidos

```
268  
269 CALL inserir_paciente('', 'M', '1990-05-10'); -- ERRO: nome vazio  
270 CALL inserir_paciente('João Pedro', 'X', '2010-03-05'); -- ERRO: sexo inválido  
271 CALL inserir_paciente('Maria Silva', 'F', '2030-01-01'); -- ERRO: data no futuro  
272  
273 CALL inserir_paciente('Carlos Alberto', 'M', '1990-05-10');  
274 CALL inserir_paciente('Fernanda Lima', 'F', '2000-12-25');  
275
```

Data output Messages

```
ERROR: Data de nascimento inválida: 2030-01-01 (no futuro)  
CONTEXT: função PL/pgSQL inserir_paciente(character varying,character,date) linha 15 em RAISE  
SQL state: P0001
```

Exercícios Resolvidos

```
268
269 CALL inserir_paciente('', 'M', '1990-05-10'); -- ERRO: nome vazio
270 CALL inserir_paciente('João Pedro', 'X', '2010-03-05'); -- ERRO: sexo inválido
271 CALL inserir_paciente('Maria Silva', 'F', '2030-01-01'); -- ERRO: data no futuro
272
273 CALL inserir_paciente('Carlos Alberto', 'M', '1990-05-10');
274 CALL inserir_paciente('Fernanda Lima', 'F', '2000-12-25');
275
```

Data output Messages

NOTICE: Paciente Carlos Alberto inserido com sucesso.

CALL

Query returned successfully in 130 msec.

Exercícios Resolvidos

```
268
269 CALL inserir_paciente('', 'M', '1990-05-10');    -- ERRO: nome vazio
270 CALL inserir_paciente('João Pedro', 'X', '2010-03-05');  -- ERRO: sexo inválido
271 CALL inserir_paciente('Maria Silva', 'F', '2030-01-01'); -- ERRO: data no futuro
272
273 CALL inserir_paciente('Carlos Alberto', 'M', '1990-05-10');
274 CALL inserir_paciente('Fernanda Lima', 'F', '2000-12-25');
275
```

Data output Messages

NOTICE: Paciente Fernanda Lima inserido com sucesso.

CALL

Query returned successfully in 89 msec.

Exercícios Resolvidos

```
280 -- EXERCÍCIO RESOLVIDO [2] -----
281 -----
282 -- Escreva um STORED PROCEDURE com as devidas validações dos dados de entrada para a geração de um
283 -- novo PEDIDO.
284 -----
285 -----
286
287 CREATE OR REPLACE PROCEDURE inserir_pedido(
288     p_dt_pedido DATE,
289     p_id_paciente INTEGER,
290     p_id_plano INTEGER,
291     p_crm TEXT
292 )
293 LANGUAGE plpgsql
294 AS $$
295 DECLARE
296     v_exists INTEGER;
297 ▼ BEGIN
298     -- Validação: data obrigatória
299 ▼ IF p_dt_pedido IS NULL THEN
300         RAISE EXCEPTION 'A data do pedido não pode ser nula';
301     END IF;
302
```

Exercícios Resolvidos

Query Query History

```
303      -- Validação: data não pode ser no futuro
304  ▼   IF p_dt_pedido > CURRENT_DATE THEN
305      RAISE EXCEPTION 'A data do pedido não pode ser futura (%).', p_dt_pedido;
306  END IF;
307
308      -- Validação: paciente deve existir
309  SELECT COUNT(*) INTO v_exists
310  FROM paciente
311  WHERE id_paciente = p_id_paciente;
312  ▼   IF v_exists = 0 THEN
313      RAISE EXCEPTION 'Paciente com ID % não existe', p_id_paciente;
314  END IF;
315
316      -- Validação: plano de saúde deve existir
317  SELECT COUNT(*) INTO v_exists
318  FROM plano_saude
319  WHERE id_plano = p_id_plano;
320  ▼   IF v_exists = 0 THEN
321      RAISE EXCEPTION 'Plano de saúde com ID % não existe', p_id_plano;
322  END IF;
323
```

Exercícios Resolvidos

Query Query History

```
323
324      -- Validação: médico deve existir
325      SELECT COUNT(*) INTO v_exists
326      FROM medico
327      WHERE crm = p_crm;
328 ▼   IF v_exists = 0 THEN
329          RAISE EXCEPTION 'Médico com CRM % não existe', p_crm;
330      END IF;
331
332      -- Validação: evitar duplicidade de pedido (mesmo paciente, plano, médico e data)
333 ▼   IF EXISTS (
334      SELECT 1 FROM pedido
335      WHERE dt_pedido = p_dt_pedido
336      AND id_paciente = p_id_paciente
337      AND id_plano = p_id_plano
338      AND crm = p_crm
339  ) THEN
340      RAISE EXCEPTION 'Já existe um pedido idêntico (paciente %, plano %, médico %, data %)', 
341                  p_id_paciente, p_id_plano, p_crm, p_dt_pedido;
342  END IF;
343
```

Exercícios Resolvidos

```
Query  Query History
342      END IF;
343
344      -- Inserção
345      INSERT INTO pedido (dt_pedido, id_paciente, id_plano, crm)
346      VALUES (p_dt_pedido, p_id_paciente, p_id_plano, p_crm);
347
348      RAISE NOTICE 'Pedido inserido com sucesso para paciente % na data %.', p_id_paciente, p_dt_pedido;
349  END;
350 $$;
351
352
353      -- Inserção válida
354  CALL inserir_pedido('2025-08-18', 3, 2, 'SP 125900');
355
```

Data output Messages

NOTICE: Pedido inserido com sucesso para paciente 3 na data 2025-08-18.

CALL

Query returned successfully in 77 msec.

Exercícios Resolvidos

Query Query History

```
355  
356 -- Tentativa inválida: paciente inexistente  
357 CALL inserir_pedido('2025-08-19', 99, 2, 'SP 125900');  
358  
359 -- Tentativa inválida: médico inexistente  
360 CALL inserir_pedido('2025-08-19', 1, 2, 'SP 9999');  
361  
362 -- Tentativa inválida: data futura  
363 CALL inserir_pedido('2030-01-01', 1, 2, 'SP 125900');  
364
```

Data output Messages

```
ERROR: Paciente com ID 99 não existe  
CONTEXT: função PL/pgSQL inserir_pedido(date,integer,integer,text) linha 20 em RAISE  
SQL state: P0001
```

Exercícios Resolvidos

Query Query History

```
355  
356 -- Tentativa inválida: paciente inexistente  
357 CALL inserir_pedido('2025-08-19', 99, 2, 'SP 125900');  
358  
359 -- Tentativa inválida: médico inexistente  
360 CALL inserir_pedido('2025-08-19', 1, 2, 'SP 9999');  
361  
362 -- Tentativa inválida: data futura  
363 CALL inserir_pedido('2030-01-01', 1, 2, 'SP 125900');  
364
```

Data output Messages

```
ERROR: Médico com CRM SP 9999 não existe  
CONTEXT: função PL/pgSQL inserir_pedido(date,integer,integer,text) linha 36 em RAISE  
SQL state: P0001
```

Exercícios Resolvidos

Query

Query History

```
355  
356 -- Tentativa inválida: paciente inexistente  
357 CALL inserir_pedido('2025-08-19', 99, 2, 'SP 125900');  
358  
359 -- Tentativa inválida: médico inexistente  
360 CALL inserir_pedido('2025-08-19', 1, 2, 'SP 9999');  
361  
362 -- Tentativa inválida: data futura  
363 CALL inserir_pedido('2030-01-01', 1, 2, 'SP 125900');  
364
```

Data output

Messages

```
ERROR: A data do pedido não pode ser futura (2030-01-01).  
CONTEXT: função PL/pgSQL inserir_pedido(date,integer,integer,text) linha 12 em RAISE  
SQL state: P0001
```

Exercícios Resolvidos

Query Query History

```
367 -- EXERCÍCIO RESOLVIDO [3] -----
368 -- Produza um STORED PROCEDURE que valide os dados de entrada e insira um novo EXAME_PEDIDO.
369 -----
370 -----
371 CREATE OR REPLACE PROCEDURE inserir_exame_pedido(
372     p_nr_pedido INTEGER,
373     p_id_exame INTEGER,
374     p_dt_coleta DATE DEFAULT NULL
375 )
376 LANGUAGE plpgsql
377 AS $$
378 DECLARE
379     v_id_plano      INTEGER;
380     v_dt_pedido     DATE;
381     v_percentual    NUMERIC;
382     v_valor_ch      NUMERIC;
383     v_valor_unidade NUMERIC;
384     v_valor_total   NUMERIC;
385     v_valor_paciente NUMERIC;
386     v_valor_plano   NUMERIC;
387 ▼ BEGIN
388     -- Validação: pedido deve existir
```

Query Query History

```
387 ▼ BEGIN
388     -- Validação: pedido deve existir
389     SELECT id_plano, dt_pedido
390     INTO v_id_plano, v_dt_pedido
391     FROM pedido
392     WHERE nr_pedido = p_nr_pedido;
393
394 ▼ IF NOT FOUND THEN
395     RAISE EXCEPTION 'Pedido % não existe', p_nr_pedido;
396 END IF;
397
398     -- Validação: exame deve existir
399     PERFORM 1 FROM exame WHERE id_exame = p_id_exame;
400 ▼ IF NOT FOUND THEN
401     RAISE EXCEPTION 'Exame % não existe', p_id_exame;
402 END IF;
403
404     -- Validação: exame deve estar autorizado para o plano
405     SELECT valor_ch
406     INTO v_valor_ch
407     FROM exame_autorizado
408     WHERE id_exame_autorizado = p_id_exame
409         AND id_plano = v_id_plano;
410
```

Exercícios Resolvidos

Query Query History

```
410
411 ▼   IF NOT FOUND THEN
412     RAISE EXCEPTION 'Exame % não autorizado para o plano %', p_id_exame, v_id_plano;
413 END IF;
414
415 -- Validação: unidade de serviço vigente
416 SELECT valor
417 INTO v_valor_unidade
418 FROM unidade_servico
419 WHERE id_plano = v_id_plano
420   AND v_dt_pedido >= dt_inicio_vigencia
421   AND (dt_fim_vigencia IS NULL OR v_dt_pedido <= dt_fim_vigencia);
422
423 ▼   IF NOT FOUND THEN
424     RAISE EXCEPTION 'Não há unidade de serviço vigente para o plano % na data %', v_id_plano, v_dt_pedido;
425 END IF;
426
427 -- Validação: data de coleta
428 ▼   IF p_dt_coleta IS NOT NULL AND p_dt_coleta < v_dt_pedido THEN
429     RAISE EXCEPTION 'Data de coleta % não pode ser anterior ao pedido %', p_dt_coleta, v_dt_pedido;
430 END IF;
431
```

Exercícios Resolvidos

Query Query History

```
432      -- Buscar percentual do paciente
433      SELECT percentual_paciente
434      INTO v_percentual
435      FROM plano_saude
436      WHERE id_plano = v_id_plano;
437
438      -- Cálculo dos valores
439      v_valor_total    := v_valor_ch * v_valor_unidade;
440      v_valor_paciente := ROUND(v_valor_total * v_percentual, 2);
441      v_valor_plano    := ROUND(v_valor_total - v_valor_paciente, 2);
442
443      -- Inserir exame pedido
444      INSERT INTO exame_pedido (nr_pedido, id_exame_pedido, dt_coleta, valor_paciente, valor_plano)
445      VALUES (p_nr_pedido, p_id_exame, p_dt_coleta, v_valor_paciente, v_valor_plano);
446
447      RAISE NOTICE 'Exame % inserido no pedido % com sucesso. Valor total: %, Paciente: %, Plano: %',
448                  p_id_exame, p_nr_pedido, v_valor_total, v_valor_paciente, v_valor_plano;
449
450      END;
451      $$;
```

Exercícios Resolvidos

Query Query History

```
452  
453    -- Inserir hemograma (1099) no pedido 1  
454    CALL inserir_exame_pedido(1, 1099, '2025-08-19');  
455  
456    -- Inserir glicemia (1100) no pedido 1 sem data de coleta  
457    CALL inserir_exame_pedido(1, 1100, NULL);  
458  
459    -- Tentativa inválida: exame não autorizado  
460    CALL inserir_exame_pedido(1, 2000, '2025-08-19');  
461  
462    -- Tentativa inválida: coleta antes da data do pedido  
463    CALL inserir_exame_pedido(1, 1099, '2025-08-01');  
464
```

Data output Messages

```
NOTICE: Exame 1099 inserido no pedido 1 com sucesso. Valor total: 6.0000, Paciente: 1.20, Plano: 4.80  
CALL  
  
Query returned successfully in 90 msec.
```

Exercícios Resolvidos

Query Query History

```
452  
453  -- Inserir hemograma (1099) no pedido 1  
454  CALL inserir_exame_pedido(1, 1099, '2025-08-19');  
455  
456  -- Inserir glicemia (1100) no pedido 1 sem data de coleta  
457  CALL inserir_exame_pedido(1, 1100, NULL);  
458  
459  -- Tentativa inválida: exame não autorizado  
460  CALL inserir_exame_pedido(1, 2000, '2025-08-19');  
461  
462  -- Tentativa inválida: coleta antes da data do pedido  
463  CALL inserir_exame_pedido(1, 1099, '2025-08-01');  
464
```

Data output Messages

```
NOTICE: Exame 1100 inserido no pedido 1 com sucesso. Valor total: 10.0000, Paciente: 2.00, Plano: 8.00  
CALL
```

```
Query returned successfully in 74 msec.
```

Exercícios Resolvidos

Query Query History

```
452  
453 -- Inserir hemograma (1099) no pedido 1  
454 CALL inserir_exame_pedido(1, 1099, '2025-08-19');  
455  
456 -- Inserir glicemia (1100) no pedido 1 sem data de coleta  
457 CALL inserir_exame_pedido(1, 1100, NULL);  
458  
459 -- Tentativa inválida: exame não autorizado  
460 CALL inserir_exame_pedido(1, 2000, '2025-08-19');  
461  
462 -- Tentativa inválida: coleta antes da data do pedido  
463 CALL inserir_exame_pedido(1, 1099, '2025-08-01');  
464
```

Data output Messages

```
ERROR: Exame 2000 não autorizado para o plano 2  
CONTEXT: função PL/pgSQL inserir_exame_pedido(integer,integer,date) linha 36 em RAISE  
SQL state: P0001
```

Exercícios Resolvidos

Query

```
452
453 -- Inserir hemograma (1099) no pedido 1
454 CALL inserir_exame_pedido(1, 1099, '2025-08-19');
455
456 -- Inserir glicemia (1100) no pedido 1 sem data de coleta
457 CALL inserir_exame_pedido(1, 1100, NULL);
458
459 -- Tentativa inválida: exame não autorizado
460 CALL inserir_exame_pedido(1, 2000, '2025-08-19');
461
462 -- Tentativa inválida: coleta antes da data do pedido
463 CALL inserir_exame_pedido(1, 1099, '2025-08-01');
464
```

Data output Messages

```
ERROR: Data de coleta 2025-08-01 não pode ser anterior ao pedido 2025-08-18
CONTEXT: função PL/pgSQL inserir_exame_pedido(integer,integer,date) linha 53 em RAISE
SQL state: P0001
```

Exercícios Resolvidos

Query Query History

```
465 --
466 --
467 -- [4] Elabore um TRIGGER que impeça um PACIENTE do sexo MASCULINO solicitar um EXAME_PEDIDO de "Beta
468 -- HCG" (Exame de Gravidez).
469 --
470 --
471 CREATE OR REPLACE FUNCTION fn_impedir_beta_hcg_masculino()
472 RETURNS TRIGGER
473 LANGUAGE plpgsql
474 AS $$ 
475 DECLARE
476     v_sexo CHAR(1);
477     v_nome_exame TEXT;
478 BEGIN
479     -- Obter sexo do paciente e nome do exame
480     SELECT pa.sexo, e.nome_exame
481     INTO v_sexo, v_nome_exame
482     FROM pedido p
483     JOIN paciente pa ON pa.id_paciente = p.id_paciente
484     JOIN exame e ON e.id_exame = NEW.id_exame_pedido
485     WHERE p.nr_pedido = NEW.nr_pedido;
486 
487     -- Validar
488     IF v_sexo = 'M' AND UPPER(v_nome_exame) LIKE 'BETA HCG%' THEN
489         RAISE EXCEPTION 'Paciente do sexo masculino não pode solicitar exame BETA HCG.';
490     END IF;
491 
492     RETURN NEW;
493 END;
```

Exercícios Resolvidos

Query Query History

```
493 END;
494 $$;
495
496 -- Criar o trigger
497 CREATE TRIGGER trg_impedir_beta_hcg_masculino
498 BEFORE INSERT ON exame_pedido
499 FOR EACH ROW
500 EXECUTE FUNCTION fn_impedir_beta_hcg_masculino();
501
502
```

Exercícios Resolvidos

Query Query History

```
503  
504  -- Paciente masculino tenta inserir Beta HCG  
505  CALL inserir_exame_pedido(2, 1600, '2025-09-20');  
506  -- ERRO: Paciente do sexo masculino não pode solicitar exame BETA HCG.  
507  
508  -- Paciente feminino solicita Beta HCG  
509  CALL inserir_exame_pedido(1, 1600, '2025-08-19');  
510  -- OK: inserido normalmente  
511  
512
```

Data output Messages

```
NOTICE: Exame 1600 inserido no pedido 2 com sucesso. Valor total: 13.5000, Paciente: 0.00, Plano: 13.50  
CALL
```

```
Query returned successfully in 87 msec.
```

Exercícios Resolvidos

```
512 SELECT * FROM PACIENTE;
```

```
512
```

Data output Messages

	id_paciente [PK] integer	nome_paciente character varying (45)	sexo character (1)	nascimento date
1	1	LIVIA XAVIER	F	2015-10-02
2	2	RENATO SOUZA JR	M	2002-03-30
3	3	CASSANDRA SILVA	F	1961-06-17
4	4	Carlos Alberto	M	1990-05-10
5	5	Fernanda Lima	F	2000-12-25

Ele é o solicitante do PEDIDO 2 de 18 de setembro 2025.

O PACIENTE 2 (Renato Souza JR) é do sexo masculino.

```
511
```

```
512
```

```
SELECT * FROM PEDIDO;
```

```
512
```

Data output Messages

	nr_pedido [PK] integer	dt_pedido date	id_paciente integer	id_plano integer	crm text
1	1	2025-08-18	1	2	SP 125900
2	2	2025-09-18	2	1	ES 2356
3	3	2025-11-03	1	2	SP 125900
4	4	2025-08-18	3	2	SP 125900

Exercícios Resolvidos

```
512 SELECT * FROM EXAME;
```

Data output Messages

	id_exame [PK] integer	nome_exame character varying (50)	id_material integer
3	1101	GLICEMIA	110
4	1200	GLICEMIA POS-PRAN...	100
5	1300	CREATININA	100
6	1400	PROTEINURIA	110
7	1500	UREIA	100
8	1600	BETA HCG	100
9	1700	PSA - ANTIGENO PRO...	100

Nosso TRIGGER falhou em bloquear a INSERÇÃO do EXAME de GRAVIDEZ para o RENATO.

O EXAME 1600 (“BETA HCG”) É UM EXAME DE GRAVIDEZ.

```
511
```

```
512 SELECT * FROM EXAME_PEDIDO WHERE NR_PEDIDO = 2;
```

```
513
```

Data output Messages

	nr_pedido integer	id_exame_pedido integer	dt_coleta date	valor_paciente numeric (9,2)	valor_plano numeric (9,2)
1	2	1099	2025-09-18	0.00	70.00
2	2	1200	2025-09-18	0.00	40.00
3	2	1600	2025-09-20	0.00	13.50

Exercícios Resolvidos

```
561  
562 delete from exame_pedido where id_exame_pedido = 1600 and nr_pedido = 2;  
563  
564
```

Data output Messages

DELETE 1

Query returned successfully in 104 msec.

Exercícios Resolvidos

```
517 CREATE OR REPLACE FUNCTION fn_impedir_beta_hcg_masculino()
518 RETURNS TRIGGER
519 LANGUAGE plpgsql
520 AS $$ 
521 DECLARE
522     v_sexo CHAR(1);
523     v_nome_exame TEXT;
524 BEGIN
525     -- Obter sexo do paciente e nome do exame
526     SELECT pa.sexo, e.nome_exame
527     INTO v_sexo, v_nome_exame
528     FROM pedido p
529     JOIN paciente pa ON pa.id_paciente = p.id_paciente
530     JOIN exame e ON e.id_exame = NEW.id_exame_pedido
531     WHERE p.nr_pedido = NEW.nr_pedido;
532
533     -- Validar restrição
534 IF v_sexo = 'M' AND UPPER(v_nome_exame) LIKE 'BETA HCG%' THEN
535     RAISE EXCEPTION 'Paciente do sexo masculino não pode solicitar exame BETA HCG.';
536 END IF;
537
538     RETURN NEW;
539 END;
540 $$;
```

Exercícios Resolvidos

```
549
550  -- Paciente masculino
551  CALL inserir_exame_pedido(2, 1600, '2025-09-19');
552  -- ERRO: Paciente do sexo masculino não pode solicitar exame BETA HCG.
553
554  -- Paciente feminino
555  CALL inserir_exame_pedido(1, 1600, '2025-08-20');
556  -- Inserido normalmente
557
```

Data output Messages

```
ERROR: Paciente do sexo masculino não pode solicitar exame BETA HCG.
CONTEXT: função PL/pgSQL fn_impedir_beta_hcg_masculino() linha 16 em RAISE
SQL statement "INSERT INTO exame_pedido (nr_pedido, id_exame_pedido, dt_coleta, valor_paciente, valor_plano)
VALUES (p_nr_pedido, p_id_exame, p_dt_coleta, v_valor_paciente, v_valor_plano)"
função PL/pgSQL inserir_exame_pedido(integer,integer,date) linha 68 em comando SQL
SQL state: P0001
```

Exercícios Resolvidos

```
553  
554 -- Paciente feminino  
555 CALL inserir_exame_pedido(1, 1600, '2025-08-20');  
556 -- Inserido normalmente  
557
```

Data output Messages

```
NOTICE: Exame 1600 inserido no pedido 1 com sucesso. Valor total: 16.0000, Paciente: 3.20, Plano: 12.80  
CALL
```

Query returned successfully in 81 msec.

Exercícios Resolvidos

```
557  
558 select * from exame_pedido where nr_pedido in (1,2);  
559
```

Data output Messages

	nr_pedido integer	id_exame_pedido integer	dt_coleta date	valor_paciente numeric (9,2)	valor_plano numeric (9,2)
1	1	1099	2025-08-18	0.00	60.00
2	1	1500	[null]	0.00	40.50
3	1	1100	2025-08-20	0.00	35.00
4	2	1099	2025-09-18	0.00	70.00
5	2	1200	2025-09-18	0.00	40.00
6	1	1099	2025-08-19	1.20	4.80
7	1	1100	[null]	2.00	8.00
8	1	1600	2025-08-20	3.20	12.80

- Sucesso!
- O EXAME 1600 não foi inserido no PEDIDO 2 (do Renato).
- Mas foi no PEDIDO 1 (da Livia).

Exercícios Resolvidos

```
563 -----
564 -----
565 -- EXERCÍCIO RESOLVIDO -----
566 -----
567 -- [5] Implemente uma RULE que impeça uma mulher de solicitar um exame de "PSA"
568 -- (Próstata).
569 -----
570 -----
571
572 CREATE OR REPLACE RULE impedir_psa_feminino AS
573 ON INSERT TO exame_pedido
574 WHERE EXISTS (
575     SELECT 1
576     FROM pedido p
577     JOIN paciente pa ON pa.id_paciente = p.id_paciente
578     JOIN exame e ON e.id_exame = NEW.id_exame_pedido
579     WHERE p.nr_pedido = NEW.nr_pedido
580         AND pa.sexo = 'F'
581         AND UPPER(e.nome_exame) LIKE 'PSA%'
582 )
583 DO INSTEAD NOTHING;
584
```

Exercícios Resolvidos

```
584  
585 -- Mulher solicitando PSA → NÃO insere nada  
586 CALL inserir_exame_pedido(1, 1700, '2025-08-25');  
587  
588 -- Homem solicitando PSA → inserção acontece normalmente  
589 CALL inserir_exame_pedido(2, 1700, '2025-08-25');  
590
```

Data output Messages

```
NOTICE: Exame 1700 inserido no pedido 1 com sucesso. Valor total: 16.0000, Paciente: 3.20, Plano: 12.80  
CALL
```

```
Query returned successfully in 60 msec.
```

Aparentemente nossa RULE fracassou e permitiu que uma mulher fizesse EXAME de Próstata! Mas, será mesmo?

Exercícios Resolvidos

```
630  
631 select * from exame_pedido where nr_pedido in (1);  
632
```

Data output Messages

	nr_pedido integer	id_exame_pedido integer	dt_coleta date	valor_paciente numeric (9,2)	valor_plano numeric (9,2)
1	1	1099	2025-08-18	0.00	60.00
2	1	1500	[null]	0.00	40.50
3	1	1100	2025-08-20	0.00	35.00
4	1	1099	2025-08-19	1.20	4.80
5	1	1100	[null]	2.00	8.00
6	1	1600	2025-08-20	3.20	12.80

Apesar da mensagem em contrário, a RULE impediu a inserção do EXAME 1700 para o PEDIDO 1 (de LÍVIA).

TRIGGER funciona melhor que RULE para esse tipo de Regra de Negócio.

Exercícios Resolvidos

```
630  
631 select * from exame_pedido where nr_pedido in (1);  
632
```

Data output Messages



	nr_pedido	id_exame_pedido	dt_coleta	valor_paciente	valor_plano
	integer	integer	date	numeric (9,2)	numeric (9,2)
1	1	1099	2025-08-18	0.00	60.00
2	1	1500	[null]	0.00	40.50
3	1	1100	2025-08-20	0.00	35.00
4	1	1099	2025-08-19	1.20	4.80
5	1	1100	[null]	2.00	8.00
6	1	1600	2025-08-20	3.20	12.80

- Porém, note que existem dois EXAMES_PEDID Os 1100 para o PEDIDO 1.
- Esquecemos de definir a chave primária de EXAME_PEDIDO.

Segurança da Informação

- A **Segurança da Informação** (SI) é a proteção de dados e sistemas contra acesso não autorizado, uso indevido, alteração, interrupção ou destruição.
- Ela se baseia em práticas, políticas e tecnologias que garantem a **confidencialidade, integridade** e **disponibilidade** das informações, sejam elas digitais ou físicas.

Segurança da Informação

- Os **princípios fundamentais da segurança da informação** são, frequentemente chamados de "**cinco pilares**" ou "**três pilares mais dois**":
 - **Confidencialidade:**
 - Garantir que a informação seja acessível apenas por usuários autorizados.
 - **Integridade:**

Segurança da Informação

- **Autenticidade:**
- Verificar se as informações ou um usuário são quem dizem ser.
- **Irretratabilidade (ou Não Repúdio):**
- O princípio de que um usuário não pode negar ter realizado uma ação, garantindo a autoria de transações de dados.

SQL Injection

- Um **SQL Injection** é um tipo de ataque de segurança em aplicações que utilizam bancos de dados relacionais.
- Ele acontece quando a aplicação não trata corretamente os dados fornecidos pelo usuário e esses dados acabam sendo inseridos diretamente em comandos SQL.
- **Isso permite que o invasor altere a lógica da consulta, manipule ou até destrua**

SQL Injection

- Imagine uma aplicação com o seguinte código (em pseudocódigo):
- **consulta = "SELECT * FROM usuarios WHERE login = " + usuario + " AND senha = " + senha + ";"**

SQL Injection

- Se o usuário preencher normalmente:

usuario = joao

senha = 1234

- A consulta fica:

SELECT * FROM usuarios WHERE login = 'joao' AND senha = '1234';

SQL Injection

- Mas, se o atacante colocar no campo de senha:
senha = ' OR '1'='1
- A consulta vira:
**SELECT * FROM usuarios WHERE login = 'joao'
AND senha = " OR '1'='1";**
- Como '**1='1**' é sempre verdadeiro, o invasor consegue **acesso indevido ao sistema**.

SQL Injection

- **Principais riscos de SQL Injection:**

- Obter acesso não autorizado (logar como outro usuário, até administrador).
- Exfiltrar dados confidenciais (senhas, números de cartão, dados pessoais).
- Alterar ou excluir dados.
- Em casos extremos, derrubar o banco ou executar comandos no servidor.

SQL Injection

Como se prevenir:

Usar queries parametrizadas / prepared statements (ex.: ? no JDBC, \$1 no PostgreSQL, :param no PHP PDO (PHP Data Objects)).

Validar e sanitizar entradas de usuários.

Usar ORM (Mapeamento Objeto-Relacional como Hibernate, Django ORM, etc.) que já tratam

SQL Injection



The screenshot shows the pgAdmin 4 interface. At the top, there's a connection bar with a gear icon, the text "Teste/postgres@PostgreSQL 14", and a dropdown arrow. Below the connection bar is a toolbar with various icons for file operations, search, and navigation. The main area is divided into two tabs: "Query" (which is selected) and "Query History". The "Query" tab contains the following SQL code:

```
1 -- Façamos um exemplo específico para o POSTGRESQL:  
2  
3 CREATE TABLE usuarios (  
4     id SERIAL PRIMARY KEY,  
5     login VARCHAR(50) NOT NULL,  
6     senha VARCHAR(50) NOT NULL  
7 );  
8  
9 INSERT INTO usuarios (login, senha) VALUES  
10    ('admin', '1234'),  
11    ('joao', 'abcd'),  
12    ('maria', 'senha');  
13
```

SQL Injection

```
13  
14 -- suponha que as variáveis são passadas direto pelo usuário  
15 -- login = joao  
16 -- senha = abcd  
17 -- Como visto anteriormente, NÂO devemos montar  
18 -- a query concatenando strings!!!  
19  
20 SELECT * FROM usuarios WHERE login = 'joao' AND senha = 'abcd';  
21
```

Data output Messages



	id [PK] integer	login character varying (50)	senha character varying (50)
1	2	joao	abcd

SQL Injection

Conforme demonstrado montar a query concatenando strings até funciona...

Mas se o atacante colocar:

login = admin

senha = ' OR '1'='1

A consulta final vira ...

SQL Injection

```
13  
14 -- suponha que as variáveis são passadas direto pelo usuário  
15 -- login = joao  
16 -- senha = abcd  
17 -- Como visto anteriormente, NÃO devemos montar  
18 -- a query concatenando strings!!!  
19  
20 SELECT * FROM usuarios WHERE login = 'joao' AND senha = 'abcd';  
21  
22 SELECT * FROM usuarios WHERE login = 'admin' AND senha = '' OR '1'='1';
```

Data output Messages

	id [PK] integer	login character varying (50)	senha character varying (50)
1	1	admin	1234

SQL Injection

```
23  
24  -- Modo CORRETO: usando PREPARE e EXECUTE:  
25  -- Criar consulta parametrizada  
26  PREPARE autenticar (text, text) AS  
27  SELECT * FROM usuarios WHERE login = $1 AND senha = $2;  
28  
29  -- Executar de forma segura  
30  EXECUTE autenticar('joao', 'abcd');  
31  
32
```

Data output Messages



	id [PK] integer	login character varying (50)	senha character varying (50)	
1	2	joao	abcd	
29				

SQL Injection

```
24 -- Modo CORRETO: usando PREPARE e EXECUTE:  
25 -- Criar consulta parametrizada  
26 PREPARE autenticar (text, text) AS  
27   SELECT * FROM usuarios WHERE login = $1 AND senha = $2;  
28  
29 -- Executar de forma segura  
30 EXECUTE autenticar('joao', 'abcd');  
31  
32  
33 -- Aqui, mesmo que o usuário tente injetar ' OR '1'='1, o PostgreSQL vai  
34 -- tratar esse valor como texto literal, e não como parte do comando SQL.  
35  
36 -- Executar de forma segura  
37 EXECUTE autenticar('admin', '' OR '1'='1');  
38
```

Data output Messages

```
ERROR: invalid input syntax for type boolean: ""  
LINE 6: EXECUTE autenticar('admin', '' OR '1'='1');  
          ^  
SQL state: 22P02  
Character: 207
```

SQL Injection

```
40 -- Caso empreguemos uma Função:  
41  
42 CREATE OR REPLACE FUNCTION autenticar_usuario(p_login text, p_senha text)  
43 RETURNS BOOLEAN AS $$  
44 DECLARE  
    existe BOOLEAN;  
45 BEGIN  
    SELECT TRUE  
    INTO existe  
    FROM usuarios  
    WHERE login = p_login AND senha = p_senha;  
51  
    RETURN COALESCE(existe, FALSE);  
53 END;  
54 $$ LANGUAGE plpgsql;  
55
```

Data output Messages

CREATE FUNCTION

Query returned successfully in 108 msec.

SQL Injection

```
42 CREATE OR REPLACE FUNCTION autenticar_usuario(p_login text, p_senha text)
43 RETURNS BOOLEAN AS $$ 
44 DECLARE
45     existe BOOLEAN;
46 BEGIN
47     SELECT TRUE
48     INTO existe
49     FROM usuarios
50     WHERE login = p_login AND senha = p_senha;
51
52     RETURN COALESCE(existe, FALSE);
53 END;
54 $$ LANGUAGE plpgsql;
55
56 SELECT autenticar_usuario('joao', 'abcd');
```

Data output	Messages
autenticar_usuario boolean	
1	true

SQL Injection

Essa nossa função PLPGSQL é segura, pois o PostgreSQL substitui os parâmetros internamente, sem concatenar strings.

COALESCE() é uma função pronta do PostgreSQL (e também existe em outros SGBDs como Oracle, SQL Server, MySQL).

SQL Injection

COALESCE(valor1, valor2, valor3, ...) retorna o primeiro valor não-nulo da lista.

Exemplo:

```
SELECT COALESCE(NULL, NULL, 'teste',  
'outro');
```

Resultado: ‘teste’ (porque é o primeiro valor não nulo).

SQL Injection

SELECT COALESCE(NULL, 0, 100);

Resultado: 0.

SELECT COALESCE(NULL, NULL, NULL);

Resultado: NULL (todos os argumentos eram nulos).

SQL Injection

Na função que mostrei antes
(autenticar_usuario):

Usei **COALESCE(existe, FALSE)** porque a variável existe poderia não receber nenhum valor da consulta (se não encontrar usuário).

Assim:

Se encontrou → retorna TRUE.

Se não encontrou nada → NULL.

SQL Injection

- O **Dollar Quoting** (\$\$... \$\$ ou \$tag\$... \$tag\$) no **PostgreSQL** é uma **sintaxe especial para delimitar literais de string em funções PL/pgSQL e comandos SQL**.
- Ele **não** foi criado diretamente para **prevenir SQL Injection**, mas ajuda indiretamente porque:
 - evita a necessidade de ficar escapando aspas simples (') dentro de blocos de código;
 - reduz a chance de erros ao concatenar strings em funções.

SQL Injection

- Exemplo de **Dollar Quoting**: **SELECT TRUE FROM USERS WHERE nome \$nome\$Sondra\$nome\$ AND senha = \$senha\$1234\$senha\$;**
- A próxima barreira de defesa na luta contra a **injeção de SQL** é **checar os tipos de dados utilizados**. Um **tipo comum de ataque** é por exemplo injetar código em parâmetros GET do protocolo http.

SQL Injection

- Se você espera um número, tenha certeza de que ele é um número antes de substituir a sua variável no seu código SQL.
- Uma forma eficiente de fazer isso é utilizar a função ‘printf’. Quase toda linguagem de programação possui um comando semelhante ao printf da linguagem C. A nossa string SQL utilizando printf ficaria alguma coisa assim: **printf('SELECT TRUE FROM USERS WHERE nome = %s AND senha = %s', nome, senha)**
- A questão aqui é que a função printf vai **forçar o uso do tipo de dados correto na função SQL. Não é uma solução perfeita**, mas pode evitar alguns problemas além de evitar a conversão implícita de tipos de dados, fonte de muitas dores de cabeça no banco de dados.

SQL Injection

```
1 ✓ CREATE TABLE usuarios (
2     id SERIAL PRIMARY KEY,
3     login VARCHAR(50) NOT NULL,
4     senha VARCHAR(50) NOT NULL
5 );
6
7 ✓ INSERT INTO usuarios (login, senha) VALUES
8     ('admin', '1234'),
9     ('joao', 'abcd'),
10    ('maria', 'senha');
11
12
13 -- VULNERÁVEL
14 ✓ CREATE OR REPLACE FUNCTION autenticar_usuario_vulneravel(p_login text, p_senha text)
15 RETURNS SETOF usuarios AS $$
16 BEGIN
17     -- Aqui concatenamos as variáveis p_login e p_senha na query (ERRADO!)
18     RETURN QUERY
19     EXECUTE 'SELECT * FROM usuarios WHERE login = ''' || p_login || ''' AND senha = ''' || p_senha || '''';
20 END;
21 $$ LANGUAGE plpgsql;
22
```

SQL Injection

```
13 -- VULNERÁVEL
14 CREATE OR REPLACE FUNCTION autenticar_usuario_vulneravel(p_login text, p_senha text)
15 RETURNS SETOF usuarios AS $$ 
16 BEGIN
17     -- Aqui concatenamos as variáveis p_login e p_senha na query (ERRADO!)
18     RETURN QUERY
19     EXECUTE 'SELECT * FROM usuarios WHERE login = ''' || p_login || ''' AND senha = ''' || p_senha || '''';
20 END;
21 $$ LANGUAGE plpgsql;
22
23
24 -- Usuário válido
25 SELECT * FROM autenticar_usuario_vulneravel('joao','abcd');
26
27 -- Tentativa de injeção funcionando
28 SELECT * FROM autenticar_usuario_vulneravel('admin', '' OR 1=1 --');
```

Data Output Messages Notifications

The screenshot shows a PostgreSQL database interface. At the top, there is a code editor window displaying SQL code related to a vulnerable function named 'autenticar_usuario_vulneravel'. Below the code editor is a table viewer showing the contents of the 'usuarios' table. The table has three columns: 'id', 'login', and 'senha'. One row is visible, with values 8, 'joao', and 'abcd' respectively. The interface includes standard database navigation buttons (refresh, back, forward) and a toolbar with icons for file operations and SQL execution.

	id integer	login character varying (50)	senha character varying (50)
1	8	joao	abcd

SQL Injection

```
13 -- VULNERÁVEL
14 ✓ CREATE OR REPLACE FUNCTION autenticar_usuario_vulneravel(p_login text, p_senha text)
15 RETURNS SETOF usuarios AS $$
16 BEGIN
17     -- Aqui concatenamos as variáveis p_login e p_senha na query (ERRADO!)
18     RETURN QUERY
19     EXECUTE 'SELECT * FROM usuarios WHERE login = ''' || p_login || ''' AND senha = ''' || p_senha || '''';
20 END;
21 $$ LANGUAGE plpgsql;
22
23
24 -- Usuário válido
25 SELECT * FROM autenticar_usuario_vulneravel('joao','abcd');
26
27 -- Tentativa de injeção funcionando
28 SELECT * FROM autenticar_usuario_vulneravel('admin', '' OR 1=1 --');
```

Data Output Messages Notifications

The screenshot shows a PostgreSQL database interface with a table named 'usuarios'. The table has three columns: 'id' (integer), 'login' (character varying(50)), and 'senha' (character varying(50)). The data in the table is as follows:

	id integer	login character varying(50)	senha character varying(50)
1	7	admin	1234
2	8	joao	abcd
3	9	maria	senha

SQL Injection

```
29  
30  -- Versão segura com USING e Dollar Quoting  
31 ✓ CREATE OR REPLACE FUNCTION autenticar_usuario_seguro(p_login text, p_senha text)  
32   RETURNS SETOF usuarios AS $$  
33   BEGIN  
34     RETURN QUERY  
35     EXECUTE $sql$  
36       SELECT * FROM usuarios WHERE login = $1 AND senha = $2  
37     $sql$  
38     USING p_login, p_senha;  
39  
40   -- O Dollar Quoting ($sql$) delimita a string SQL.  
41   -- O USING injeta o valor de forma segura.  
42 END;  
43 $$ LANGUAGE plpgsql;  
44  
45
```

Data Output Messages Notifications

CREATE FUNCTION

Query returned successfully in 67 msec.

SQL Injection

```
30 -- Versão segura com USING e Dollar Quoting
31 ✓ CREATE OR REPLACE FUNCTION autenticar_usuario_seguro(p_login text, p_senha text)
32 RETURNS SETOF usuarios AS $$ 
33 BEGIN
34     RETURN QUERY
35     EXECUTE $sql$ 
36         SELECT * FROM usuarios WHERE login = $1 AND senha = $2
37     $sql$ 
38     USING p_login, p_senha;
39
40 -- O Dollar Quoting ($sql$) delimita a string SQL.
41 -- O USING injeta o valor de forma segura.
42 END;
43 $$ LANGUAGE plpgsql;
44
45 -- Usuário válido
46 SELECT * FROM autenticar_usuario_seguro('joao','abcd');
47
48 -- Tentativa de injeção
49 SELECT * FROM autenticar_usuario_seguro('admin',''' OR 1=1');
50
```

Data Output Messages Notifications

≡+ 📁 ↻ 🗂️ ↻ 🗃️ ↻ ↴ ↵ SQL

	id integer	login character varying (50)	senha character varying (50)
1	8	joao	abcd

SQL Injection

```
30  -- Versão segura com USING e Dollar Quoting
31  ✓ CREATE OR REPLACE FUNCTION autenticar_usuario_seguro(p_login text, p_senha text)
32    RETURNS SETOF usuarios AS $$
33    BEGIN
34      RETURN QUERY
35      EXECUTE $sql$ 
36          SELECT * FROM usuarios WHERE login = $1 AND senha = $2
37      $sql$ 
38      USING p_login, p_senha;
39
40  -- O Dollar Quoting ($sql$) delimita a string SQL.
41  -- O USING injeta o valor de forma segura.
42  END;
43  $$ LANGUAGE plpgsql;
44
45  -- Usuário válido
46  SELECT * FROM autenticar_usuario_seguro('joao','abcd');
47
48  -- Tentativa de injeção
49  SELECT * FROM autenticar_usuario_seguro('admin',''' OR 1=1');
50
```

Data Output Messages Notifications



	id integer	login character varying (50)	senha character varying (50)

Criação de Novas Contas de Usuários

- Até a **versão 8.1**, os conceitos de usuários e de grupos administrativos eram distintos em **PostgreSQL**. Esses conceitos foram abstraídos e absorvidos por uma única entidade chamada "**role**".
- A palavra "**role**", em inglês, significa cargo (no sentido de função) ou "papel" (no sentido de "como atua").
- Assim, uma **role** descreve seu próprio papel e quais as funções em que ela atua no contexto da segurança do banco de dados.
- De modo abstrato, uma role pode se comportar como um usuário, como um grupo ou ter ambos comportamentos ao mesmo tempo. A **role** pode conter e ser contida por outra **role**. Deste modo, fica claro que o conceito de **role** está relacionado com a definição de permissões, privilégios e garantias de acesso aos objetos do banco e aos dados.

Criação de Novas Contas de Usuários

- **Roles** podem ser donas de seus próprios objetos (tabelas) e podem delegar permissões ou direitos para outras **roles** através de herança e relacionamentos de confiança.
- Os **privilégios** dados em **roles**, são completamente distintos dos privilégios dados no sistema de arquivos. Muitas vezes, pode ser conveniente manter uma relação entre privilégios dos objetos do banco e privilégios do sistema de arquivos, mas isso não é obrigatório.
- As **roles** possuem um contexto global relativo ao **cluster do banco**. Isso significa que **elas existem em todos os bancos de dados criados no cluster**. Cada **role** é identificada por um nome exclusivo dentro do cluster, ou seja, cada role deve ser única.
- Os nomes das roles seguem o padrão léxico para criação de identificadores de objetos em PostgreSQL. O nome do identificador DEVE começar com uma letra [a-z], seguida de outros caracteres, incluindo letras acentuadas ou sinais não latinos, caracteres numéricos [0-9] e o sinal de sublinhado (underscore), que também é válido.

Criação de Novas Contas de Usuários

- Quando **PostgreSQL** é instalado, uma **role** com atribuições administrativas é automaticamente configurada.
- Todas as permissões administrativas são concedidas para essa **role**. Normalmente, pensamos nela como um **superusuário administrativo**. Essa **role** especial, quase sempre, é identificada pelo nome de **postgres**.
- No **PostgreSQL** devemos evitar o login com a **role postgres**. Crie e utilize outras **roles administrativas com poderes limitados ao contexto de uso de cada banco de dados**.

Criação de Novas Contas de Usuários

```
50
51 -- As ROLES de Banco de Dados são GLOBAIS em uma instalação de CLUSTER
52 -- de BANCO de DADOS e NÃO pertencentes a um Banco de Dados individual
53 -- como IES, LABORATORIO, PASSAGENS etc.
54 SELECT rolname FROM pg_roles;
55
```

Data Output Messages Notifications

SQL

	rolname name	lock
1	pg_database_owner	
2	pg_read_all_data	
3	pg_write_all_data	
4	pg_monitor	
5	pg_read_all_settings	
6	pg_read_all_stats	
7	pg_stat_scan_tables	
8	pg_read_server_files	
9	pg_write_server_files	
10	pg_execute_server_program	
11	pg_signal_backend	
12	pg_checkpoint	
13	pg_maintain	
14	pg_use_reserved_connections	
15	pg_create_subscription	
16	postgres	

Criação de Novas Contas de Usuários

```
51 -- As ROLES de Banco de Dados são GLOBAIS em uma instalação de CLUSTER
52 -- de BANCO de DADOS e NÃO pertencentes a um Banco de Dados individual
53 -- como IES, LABORATORIO, PASSAGENS etc.
54 SELECT ROLNAME FROM PG_ROLES;
55
56 -- A tabela interna do PostgreSQL PG_ROLES possui a coluna ROLNAME com os
57 -- nomes de todos os ROLES existentes no CLUSTER de BANCO de DADOS.
```

The screenshot shows a PostgreSQL client interface with a toolbar at the top and three tabs below it: Data, Output, Messages, and Notifications. The SQL tab is selected. Below the tabs is a table with 16 rows, each containing a number from 1 to 16 and a corresponding PostgreSQL role name. The first row is highlighted with a blue background.

	rolname name
1	pg_database_owner
2	pg_read_all_data
3	pg_write_all_data
4	pg_monitor
5	pg_read_all_settings
6	pg_read_all_stats
7	pg_stat_scan_tables
8	pg_read_server_files
9	pg_write_server_files
10	pg_execute_server_program
11	pg_signal_backend
12	pg_checkpoint
13	pg_maintain
14	pg_use_reserved_connections
15	pg_create_subscription
16	postgres

Object Explorer

Servers (1)

PostgreSQL 17

Databases (2)

> postgres

teste

- > Casts
- > Catalogs
- > Event Triggers
- > Extensions
- > Foreign Data Wrappers
- > Languages
- > Publications
- > Schemas
- > Subscriptions

Login/Group Roles (16)

- pg_checkpoint
- pg_create_subscription
- pg_database_owner
- pg_execute_server_program
- pg_maintain
- pg_monitor
- pg_read_all_data
- pg_read_all_settings
- pg_read_all_stats
- pg_read_server_files
- pg_signal_backend
- pg_stat_scan_tables
- pg_use_reserved_connections
- pg_write_all_data
- pg_write_server_files
- postgres

Tablespaces

Dashboard X Properties X SQL X Statistics X

teste/postgres@PostgreSQL 17

No limit

Query Query History

```

48 -- Tentativa de Injeção
49 SELECT * FROM autenticar_usuario_se
50
51 -- As ROLES de Banco de Dados são C
52 -- de BANCO de DADOS e NÃO pertence
53 -- como IES, LABORATORIO, PASSAGENS
54 SELECT ROLNAME FROM PG_ROLES;
55
56 -- A tabela interna do PostgreSQL F
57 -- nomes de todos os ROLES existentes

```

Data Output Messages Notifications

	rolname name	lock
1	pg_database_owner	
2	pg_read_all_data	
3	pg_write_all_data	
4	pg_monitor	
5	pg_read_all_settings	
6	pg_read_all_stats	
7	pg_stat_scan_tables	
8	pg_read_server_files	
9	pg_write_server_files	
10	pg_execute_server_program	
11	pg_signal_backend	
12	pg_checkpoint	
13	pg_maintain	
14	pg_use_reserved_connections	
15	pg_create_subscription	
16	postgres	

Criação de Novas Contas de Usuários

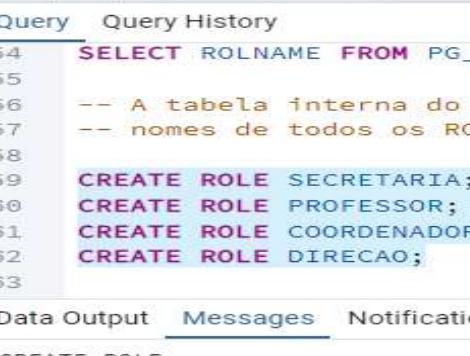
```
56 -- A tabela interna do PostgreSQL PG_ROLES possui a coluna ROLNAME com os  
57 -- nomes de todos os ROLES existentes no CLUSTER de BANCO de DADOS.  
58  
59 CREATE ROLE SECRETARIA;  
60 CREATE ROLE PROFESSOR;  
61 CREATE ROLE COORDENADOR;  
62 CREATE ROLE DIRECAO;  
63
```

Data Output Messages Notifications

CREATE ROLE

Query returned successfully in 59 msec.

- ✓ Servers (1)
 - ✓ PostgreSQL 17
 - ✓ Databases (2)
 - > postgres
 - ✓ teste
 - > Casts
 - > Catalogs
 - > Event Triggers
 - > Extensions
 - > Foreign Data Wrappers
 - > Languages
 - > Publications
 - > Schemas
 - > Subscriptions
- ✓ Login/Group Roles (20)
 - coordenador
 - direcao
 - pg_checkpoint
 - pg_create_subscription
 - pg_database_owner
 - pg_execute_server_program
 - pg_maintain
 - pg_monitor
 - pg_read_all_data
 - pg_read_all_settings
 - pg_read_all_stats
 - pg_read_server_files
 - pg_signal_backend
 - pg_stat_scan_tables
 - pg_use_reserved_connections
 - pg_write_all_data
 - pg_write_server_files
 - postgres
 - professor
 - secretaria



The screenshot shows the pgAdmin 4 interface with a PostgreSQL connection named "teste/postgres@PostgreSQL 17". The toolbar includes icons for file operations, copy, paste, and search. A dropdown menu indicates "No limit" for results. Below the toolbar, tabs for "Query" and "Query History" are visible, with "Query" selected. The main area contains a query editor with the following SQL code:

```
54 SELECT ROLNAME FROM PG_ROLE  
55  
56 -- A tabela interna do Post  
57 -- nomes de todos os ROLES  
58  
59 CREATE ROLE SECRETARIA;  
60 CREATE ROLE PROFESSOR;  
61 CREATE ROLE COORDENADOR;  
62 CREATE ROLE DIRECAO;  
63
```

Below the query editor, there are tabs for "Data Output", "Messages", and "Notifications", with "Messages" currently selected. The status bar at the bottom displays the message "CREATE ROLE" and "Query returned successfully in 59 ms".

Criação de Novas Contas de Usuários

```
248  
249 SELECT rolname FROM pg_roles;  
250  
251 CREATE ROLE ADDRESSA LOGIN PASSWORD '12345' IN ROLE SECRETARIA;  
252
```

Data output Messages

```
CREATE ROLE
```

Query returned successfully in 91 msec.

Criamos uma conta de usuário de nome **ADDRESSA** com senha '**12345**' com os mesmos privilégios e restrições que o **ROLE (Papel) SECRETARIA** – criado anteriormente.

Criação de Novas Contas de Usuários

```
249 SELECT rolname FROM pg_roles;  
250  
251 CREATE ROLE ANDRESSA LOGIN PASSWORD '12345' IN ROLE SECRETARIA;  
252  
253 CREATE ROLE LINA LOGIN PASSWORD '4567' IN ROLE PROFESSOR;  
254
```

Data output Messages

CREATE ROLE

Query returned successfully in 82 msec.

- Uma conta de usuário **LINA** foi criada com a senha '**4567**' e com o mesmo nível de privilégios que o **ROLE PROFESSOR**.

Criação de Novas Contas de Usuários

```
253 CREATE ROLE LINA LOGIN PASSWORD '4567' IN ROLE PROFESSOR;  
254  
255 -- Revogação (REVOKE) de todos (ALL) os privilégios de acesso sobre  
256 -- as tabelas para todos os usuários.  
257  
258 REVOKE ALL ON TABLE PROFESSOR FROM PUBLIC;  
259  
260
```

Data output Messages

REVOKE

Query returned successfully in 85 msec.

Criação de Novas Contas de Usuários

```
258 REVOKE ALL ON TABLE PROFESSOR FROM PUBLIC;
259
260 REVOKE ALL ON TABLE CURSO FROM PUBLIC;
261 REVOKE ALL ON TABLE DEPARTAMENTO FROM PUBLIC;
262 REVOKE ALL ON TABLE DISCIPLINA FROM PUBLIC;
263 REVOKE ALL ON TABLE GRADE_CURRICULAR FROM PUBLIC;
264 REVOKE ALL ON TABLE DISCIPLINA_OFERTADA FROM PUBLIC;
265 REVOKE ALL ON TABLE ALUNO FROM PUBLIC;
266 REVOKE ALL ON TABLE CONDICAO FROM PUBLIC;
267 REVOKE ALL ON TABLE HISTORICO FROM PUBLIC;
268
```

Data output Messages

REVOKE

Query returned successfully in 66 msec.

Criação de Novas Contas de Usuários

```
268
269 -- TODOS os USUÁRIOS (ROLEs) tiveram retirados os seus eventuais privilégios
270 -- sobre a totalidade das tabelas do Banco de Dados IES (Instituição de Ensino
271 -- Superior).
272
273 -- Como esse BANCO e suas TABELAS foram criados e são propriedade do ROLE postgres,
274 -- essa medida parece exagerada e quase desnecessária mas visa assegurar com absoluta
275 -- certeza que NENHUM outro USUÁRIO (Exceto postgres) pode ALTERAR, APAGAR, CONSULTAR
276 -- ou INSERIR dados nessas tabelas.
277
278 -- Criação de Função SQL para listar Professores.
279 CREATE FUNCTION LISTAR_PROFESSORES()
280 RETURNS SETOF PROFESSOR AS $$
281     SELECT ID_PROFESSOR, NOME
282     FROM PROFESSOR
283     ORDER BY NOME;
284 $$ LANGUAGE SQL;
285
```

Data output Messages

CREATE FUNCTION

Query returned successfully in 94 msec.

Criação de Novas Contas de Usuários

```
278 -- Criação de Função SQL para listar Professores.  
279 CREATE FUNCTION LISTAR_PROFESSORES()  
280 RETURNS SETOF PROFESSOR AS $$  
281     SELECT ID_PROFESSOR, NOME  
282     FROM PROFESSOR  
283     ORDER BY NOME;  
284 $$ LANGUAGE SQL;  
285  
286 REVOKE ALL ON FUNCTION LISTAR_PROFESSORES() FROM PUBLIC;  
287  
288
```

Data output Messages

REVOKE

Query returned successfully in 53 msec.

Atribuição de Privilégios

```
287  
288 -- Concedendo privilégio de execução para os ROLES SECRETARIA,  
289 -- DIRECAO  
290  
291 GRANT EXECUTE ON FUNCTION LISTAR_PROFESSORES() TO SECRETARIA;  
292  
293 GRANT EXECUTE ON FUNCTION LISTAR_PROFESSORES() TO DIRECAO;  
294  
295 GRANT EXECUTE ON FUNCTION LISTAR_PROFESSORES() TO COORDENADOR;  
296  
297 -- Apenas o ROLE PROFESSOR não tem permissão de executar a Função  
298 -- LISTAR_PROFESSOR( ).  
299
```

Data output Messages

GRANT

Query returned successfully in 104 msec.

Atribuição de Privilégios

```
307  
308 -- Atribuição de privilégios (GRANT) para consulta (SELECT) e inserção  
309 -- (INSERT) sobre as tabelas PROFESSOR, CURSO, DEPARTAMENTO, ALUNO e  
310 -- CONDICAO para o ROLE SECRETARIA.  
311  
312  
313 GRANT SELECT, INSERT ON PROFESSOR, CURSO, DEPARTAMENTO, ALUNO, CONDICAO TO  
314 SECRETARIA WITH GRANT OPTION;  
315  
316 -- O WITH GRANT OPTION permite que esse usuário (SECRETARIA) não apenas use  
317 -- os privilégios concedidos, mas também os repasse a outros usuários/roles.  
318
```

Data output Messages

GRANT

Query returned successfully in 60 msec.

Atribuição de Privilégios

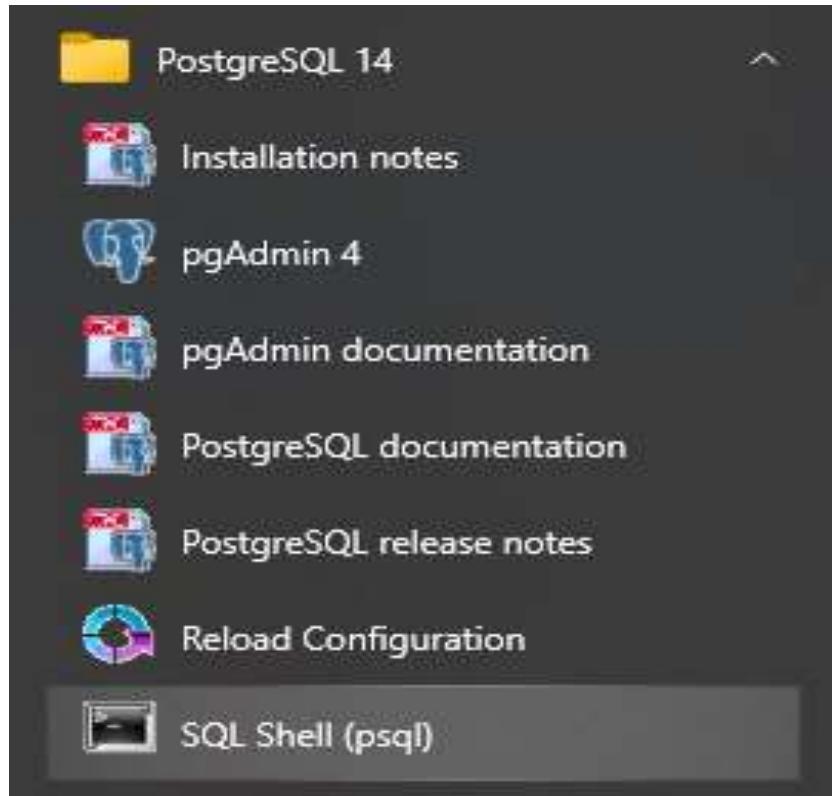
```
312  
313 GRANT SELECT, INSERT ON PROFESSOR, CURSO, DEPARTAMENTO, ALUNO, CONDICAO TO  
314 SECRETARIA WITH GRANT OPTION;  
315  
316 -- O WITH GRANT OPTION permite que esse usuário (SECRETARIA) não apenas use  
317 -- os privilégios concedidos, mas também os repasse a outros usuários/roles.  
318  
319 GRANT SELECT, INSERT, UPDATE, DELETE ON CURSO, DISCIPLINA, GRADE_CURRICULAR,  
320 DISCIPLINA_OFERTADA TO COORDENADOR;  
321  
322 -- Autoriza (GRANT) o ROLE COORDENADOR a efetuar consultas (SELECT), inserções  
323 -- (INSERT), exclusões (DELETE) e atualizações (UPDATE) sobre a lista de tabelas:  
324 -- CURSO, DISCIPLINA, GRADE_CURRICULAR e DISCIPLINA_OFERTADA.  
325
```

Data output Messages

GRANT

Query returned successfully in 76 msec.

Testando Privilégios Concedidos



Na pasta **PostgreSQL** selecione **SQL Shell (psql)**.

Testando Privilégios Concedidos



Um **prompt** é aberto perguntando pelo **Servidor (server)**.

Testando Privilégios Concedidos

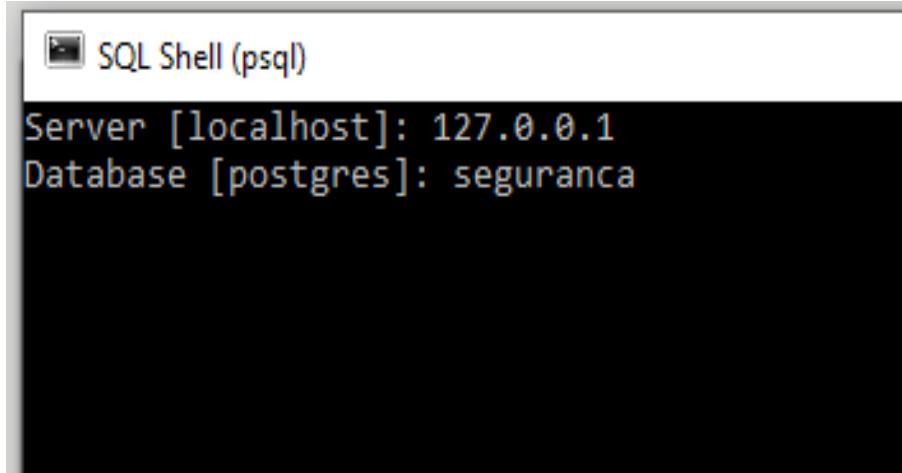


```
SQL Shell (psql)
Server [localhost]: 127.0.0.1
Database [postgres]:
```

O servidor informado foi o localhost:
127.0.0.1.

Agora ele solicita o Banco de Dados (Database).

Testando Privilégios Concedidos



A screenshot of a terminal window titled "SQL Shell (psql)". The window shows the following text:
Server [localhost]: 127.0.0.1
Database [postgres]: segurança

Database Segurança.

informado:

Testando Privilégios Concedidos

```
SQL Shell (psql)
Server [localhost]: 127.0.0.1
Database [postgres]: seguranca
Port [5432]: 5432
```

Porta de Acesso: 5432.

 SQL Shell (psql)

```
Server [localhost]: 127.0.0.1
Database [postgres]: segurança
Port [5432]: 5432
Username [postgres]: ANDRESSA
```

**Nome do usuário:
ANDRESSA.**

**Essa usuário possui o
ROLE
de
SECRETARIA.**

Testando Privilégios Concedidos

SQL Shell (psql)

```
Server [localhost]: 127.0.0.1
Database [postgres]: seguranca
Port [5432]: 5432
Username [postgres]: ANDRESSA
Password for user ANDRESSA: ■
```

Informamos a **SENHA**
12345.

Testando Privilégios Concedidos

SQL Shell (psql)

```
Server [localhost]: 127.0.0.1
Database [postgres]: segurança
Port [5432]: 5432
Username [postgres]: andressa
Password for user andressa:
psql (14.4)
WARNING: Console code page (850) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

segurança=> ■
```

Atenção: não escreva o nome do usuário em caixa alta. Use letras minúsculas.

Testando Privilégios Concedidos

```
SQL Shell (psql)
WARNING: Console code page (850) differs from Windows code page (1252)
          8-bit characters might not work correctly. See psql reference
          page "Notes for Windows users" for details.
Type "help" for help.

segurança=> SELECT * FROM PROFESSOR;
 id_professor | nome
-----+-----
 10 | ASDRUBAL SOARES RIBEIRO
 22 | JAMBIRA TIMBIRAS
 31 | DESIDERIO MONFORTE
 15 | SUETONIO ROMAO SOUZA
 21 | CARLOTA FLAVINIAS
 33 | RONALDO CARVALHO CAMPOS
 34 | EDUARDO LICOLINI
 35 | CINTHIA RIBEIRO
 36 | RENATO AURELIO MARCONDES
 37 | ANA PAULA XAVIER DE ABREU
 38 | CLAUDIA LUCIA OLIVEIRA OHMS
 39 | JOSE MAGALHAES NETO
 40 | JOSE MAURICIO MADEIRO
 41 | ANA PAULA LASCASAS
 42 | CLAUDIA GONCALVES RIBEIRO
 43 | TITO CARVALHO
 44 | ROXANNA ALVES MANGABEIRA
 45 | LEONARDO PEREIRA
(18 rows)

segurança=>
```

Note
andressa
que
tem
autorização
para
consultar
(SELECT)
tabela
PROFESSOR.

Testando Privilégios Concedidos

```
35 | CINTHIA RIBEIRO  
36 | RENATO AURELIO MARCONDES  
37 | ANA PAULA XAVIER DE ABREU  
38 | CLAUDIA LUCIA OLIVEIRA OHMS  
39 | JOSE MAGALHAES NETO  
40 | JOSE MAURICIO MADEIRO  
41 | ANA PAULA LASCASAS  
42 | CLAUDIA GONCALVES RIBEIRO  
43 | TITO CARVALHO  
44 | ROXANNA ALVES MANGABEIRA  
45 | LEONARDO PEREIRA
```

(18 rows)

```
seguranca=> DELETE FROM PROFESSOR WHERE ID_PROFESSOR = 44;  
ERROR: permission denied for table professor  
seguranca=> -
```

Todavia, **andressa**
não **possui**
autorização para
efetuar uma
exclusão na mesma
tabela.

Testando Privilépios Concedidos

(18 rows)

```
segurança=> DELETE FROM PROFESSOR WHERE ID_PROFESSOR = 44;
ERROR: permission denied for table professor
segurança=>
segurança=>
segurança=>
segurança=> SELECT * FROM DISCIPLINA;
ERROR: permission denied for table disciplina
segurança=>
```

Andressa também não possui autorização para consultar a tabela **DISCIPLINA**.

Testando Privilégios Concedidos

```
segurança=>
segurança=> SELECT * FROM LISTAR_PROFESSORES();
+-----+
| id_professor | nome          |
+-----+
|      41 | ANA PAULA LASCASAS
|      37 | ANA PAULA XAVIER DE ABREU
|      10 | ASDRUBAL SOARES RIBEIRO
|      21 | CARLOTA FLAVINIAS
|      35 | CINTHIA RIBEIRO
|      42 | CLAUDIA GONCALVES RIBEIRO
|      38 | CLAUDIA LUCIA OLIVEIRA OHMS
|      31 | DESIDERIO MONFORTE
|      34 | EDUARDO LICOLINI
|      22 | JAMBIRA TIMBIRAS
|      39 | JOSE MAGALHAES NETO
|      40 | JOSE MAURICIO MADEIRO
|      45 | LEONARDO PEREIRA
|      36 | RENATO AURELIO MARCONDES
|      33 | RONALDO CARVALHO CAMPOS
|      44 | ROXANNA ALVES MANGABEIRA
|      15 | SUETONIO ROMAO SOUZA
|      43 | TITO CARVALHO
+-----+
(18 rows)
```

Andressa também pode executar a Função SQL **LISTAR_PROFESSORES()**.

```
segurança=>
```

Testando Privilégios Concedidos

```
(18 rows)
```

```
segurança=> INSERT INTO PROFESSOR (NOME) VALUES ('SUETONIO SILVA');
ERROR: null value in column "id_professor" of relation "professor" violates not-null constraint
DETAIL: Failing row contains (null, SUETONIO SILVA).
segurança=>
```

ID_PROFESSOR na tabela PROFESSOR não foi definido como SERIAL mas sim como INTEGER.

Testando Privilégios Concedidos

```
(18 rows)

segurança=> INSERT INTO PROFESSOR (NOME) VALUES ('SUETONIO SILVA');
ERROR: null value in column "id_professor" of relation "professor" violates not-null constraint
DETAIL: Failing row contains (null, SUETONIO SILVA).
segurança=> INSERT INTO PROFESSOR VALUES (46,'SUETONIO SILVA');
INSERT 0 1
segurança=> ■
```

Andressa consegue inserir um novo PROFESSOR.

Testando Privilégios Concedidos

```
segurança=> INSERT INTO PROFESSOR (NOME) VALUES ('SUETONIO SILVA');
ERROR: null value in column "id_professor" of relation "professor" violates not-null constraint
DETAIL: Failing row contains (null, SUETONIO SILVA).
segurança=> INSERT INTO PROFESSOR VALUES (46,'SUETONIO SILVA');
INSERT 0 1
segurança=>
segurança=>
segurança=>
segurança=>
segurança=> UPDATE PROFESSOR SET NOME='SOSTENES EPAMINONDAS JUNIOR' WHERE ID_PROFESSOR = 46;
ERROR: permission denied for table professor
segurança=> .
```

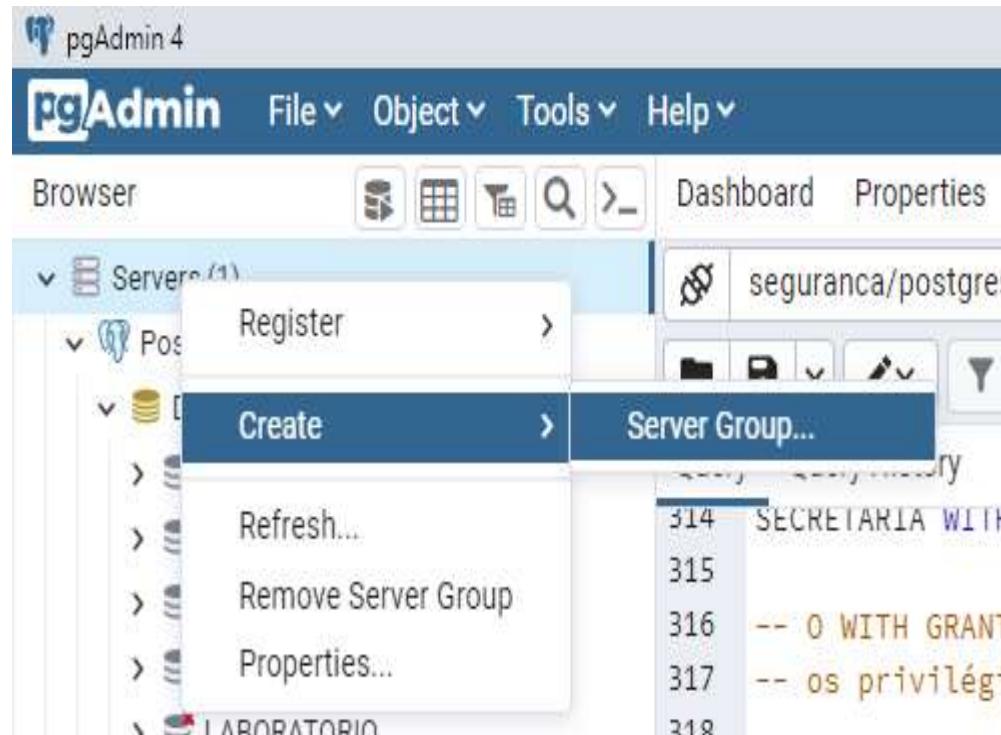
Ela não tem autorização para fazer alterações em registros da tabela PROFESSOR.

Testando Privilégios Concedidos

```
segurança=> UPDATE PROFESSOR SET NOME='SOSTENES EPAMINONDAS JUNIOR' WHERE ID_PROFESSOR = 46;
ERROR: permission denied for table professor
segurança=> GRANT EXECUTE ON FUNCTION LISTAR_PROFESSORES() TO LINA;
WARNING: no privileges were granted for "listar_professores"
GRANT
segurança=>
segurança=>
segurança=>
```

Repare que **ANDRESSA** consegue conceder autorização sobre a execução da função **LISTAR_PROFESSORES()** para **LINA** (**ROLE PROFESSOR**) em razão da cláusula **WITH**

Testando Privilégios Concedidos



De volta ao **PGAdmin 4** clique na aba **SERVERS** com o **botão direito do mouse**.

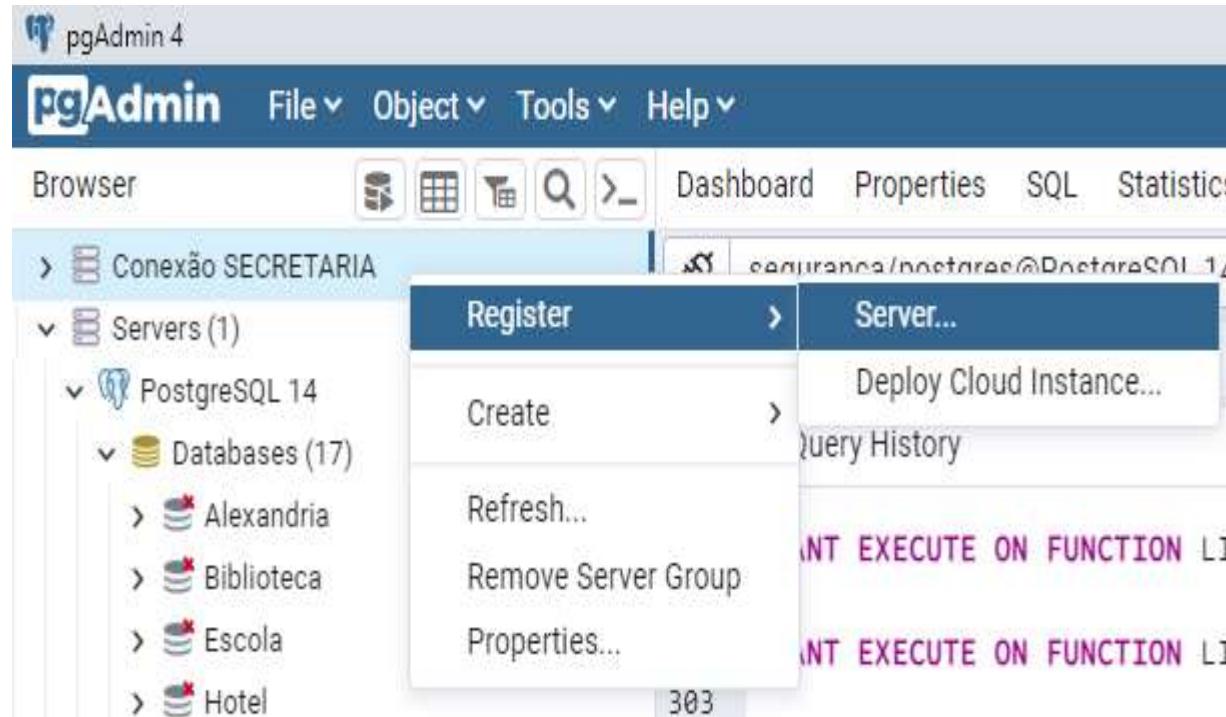
Selecione **CREATE SERVER GROUP**.

Testando Privilégios Concedidos



Informe **NAME** com
**“Conexão
SECRETARIA”.**
Clique em **SAVE**.

Testando Privilégios Concedidos

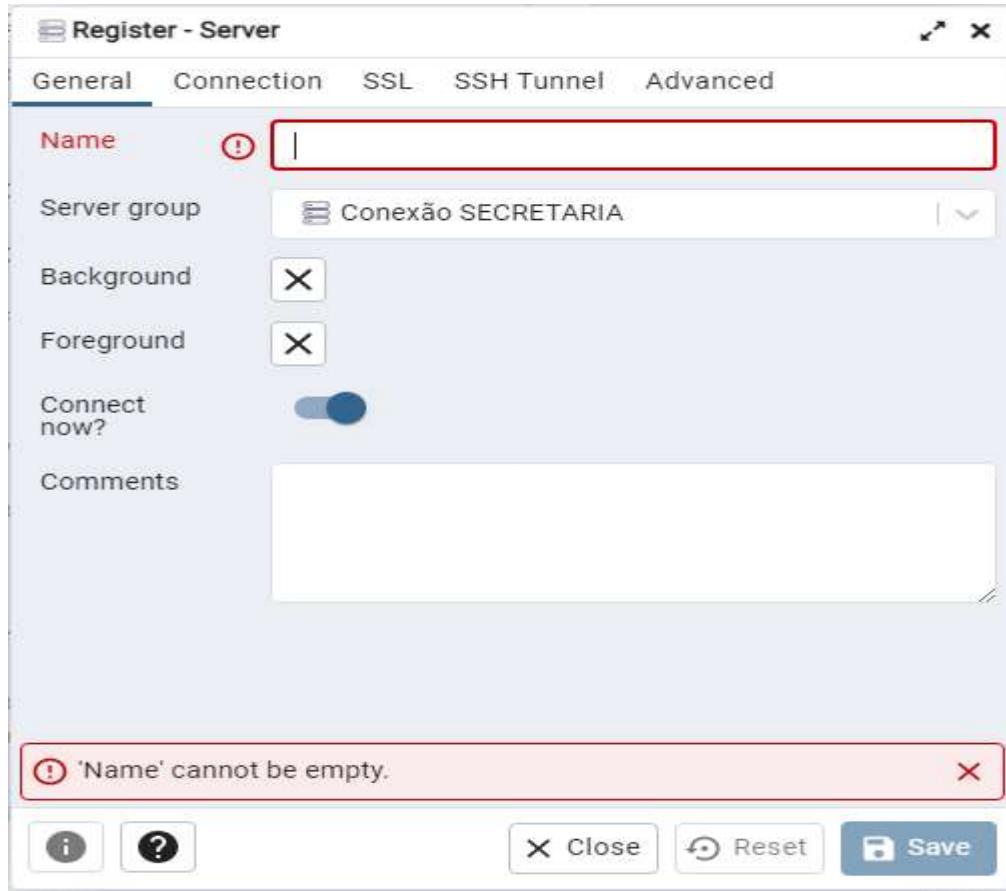


Clique com o botão direito do mouse sobre a conexão recém criada.

Selecione
REGISTER.

Selecione
SERVER...

Testando Privilégios Concedidos



Informe **NAME** do
SERVER.

Clique na aba
CONNECTION.

Testando Privilégios Concedidos



Testando Privilégios Concedidos

Register - Server

General Connection SSL SSH Tunnel Advanced

Host name/addresses: 127.0.0.1

Port: 5432

Maintenance database: postgres

Username: andressa

Kerberos authentication:

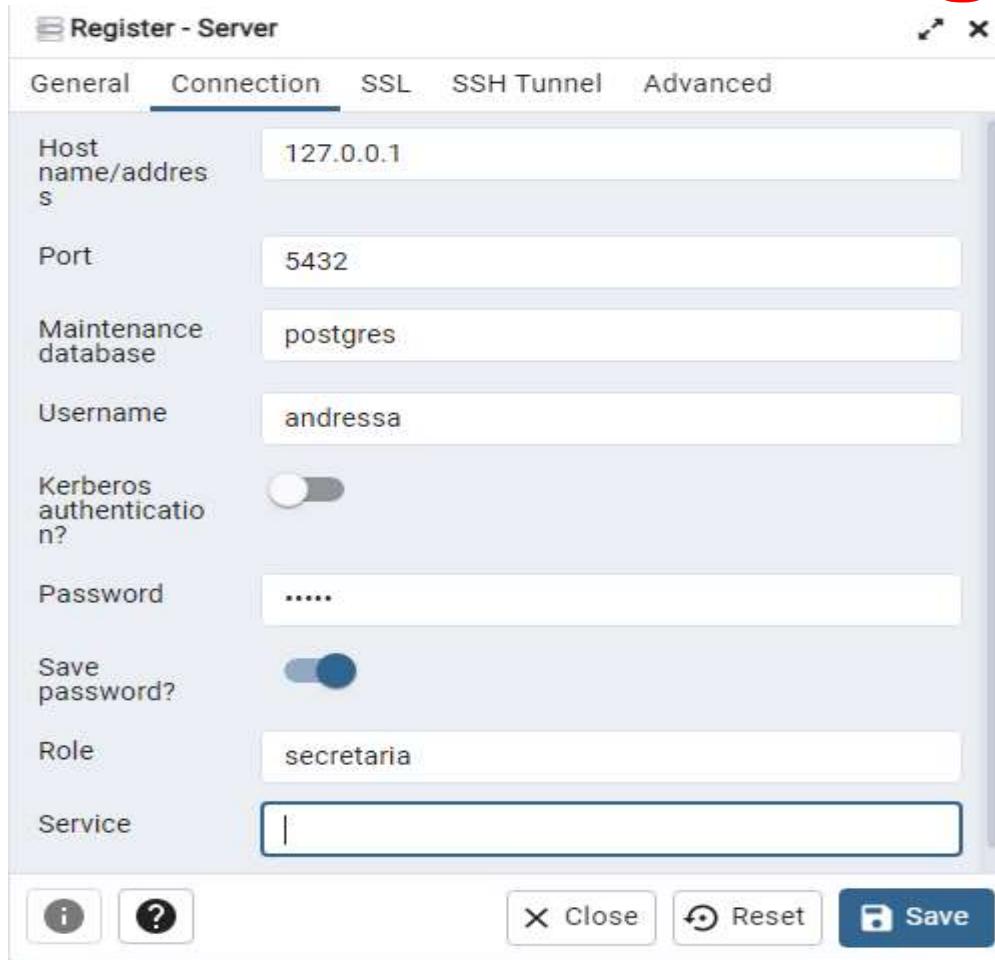
Password:

Save password?:

Role: secretaria

Service: |

Save



Clique no botão
salvar.

pgAdmin 4

pgAdmin File Object Tools

Browser

Conexão SECRETARIA (1)

Servers

Databases (17)

- > Alexandria
- > Biblioteca
- > Escola
- > Hotel
- > LABORATORIO
- > Loja
- > Passagens
- > Prova
- > Prova2
- > Prova7
- > Prova II M08 - 2023
- > S08
- > SUPERMERCADO
- > Teste
- > banco
- > postgres
- > segurança

Login/Group Roles

Tablespaces

Servers (1)

PostgreSQL 14



Clique sobre o **banco de dados** **seguranca** na nova conexão **(Conexão SECRETARIA)**.

Selecione **QUERY TOOL**.

Teste no PgAdmin

The screenshot shows the pgAdmin 4 interface. The left sidebar displays a tree view of database connections and objects. Under 'Conexão SECRETARIA (1) / Databases (17)', several tables are listed: Alexandria, Biblioteca, Escola, Hotel, LABORATORIO, Loja, Passagens, Prova, Prova2, Prova7, Prova II M08 - 2023, S08, SUPERMERCADO, Teste, banco, postgres, and segurança. The 'segurança' connection is currently selected. The main window shows a query editor with the following content:

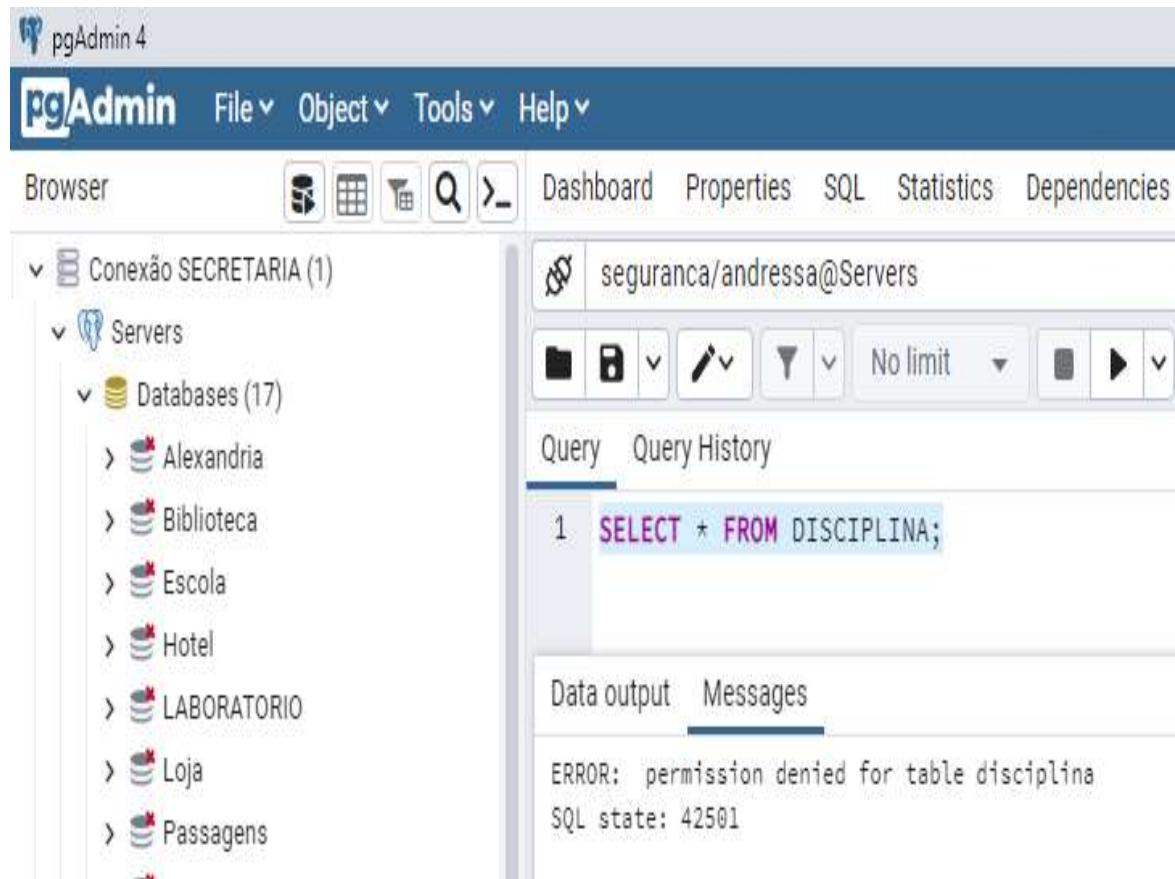
```
1  SELECT * FROM CURSO;
```

The results are displayed in a table titled 'Data output' under the 'Messages' tab:

	id_curso [PK] integer	nome character varying (45)	id_coordenador integer
1	100	CIENCIAS ECONOMICAS	22
2	200	ARQUITETURA E URBANISMO	31
3	300	SISTEMAS DE INFORMACAO	10

Andressa tem autorização de consulta sobre a tabela CURSO.

Teste no PgAdmin



Andressa continua
não tendo
autorização de
consulta sobre a
tabela **DISCIPLINA**.

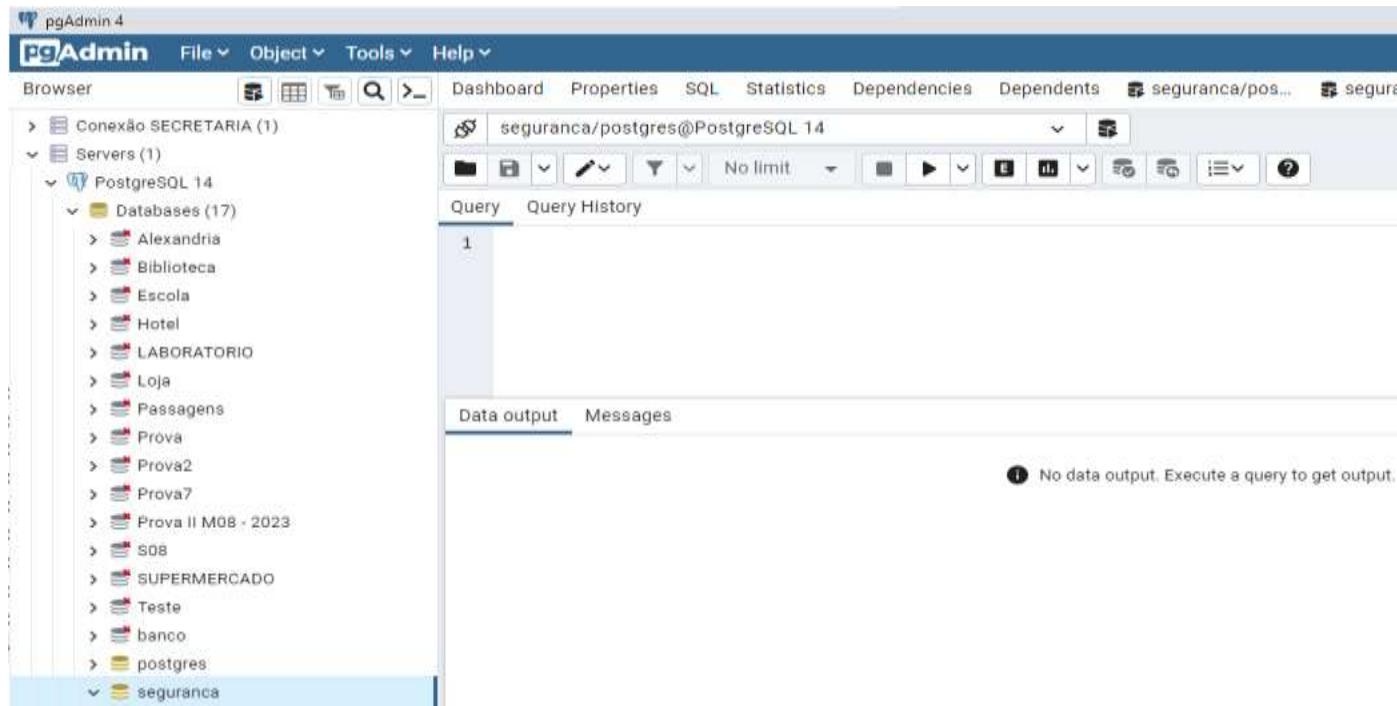
Teste no PgAdmin

No PostgreSQL dá para assumir temporariamente outro papel dentro da mesma conexão usando SET ROLE.

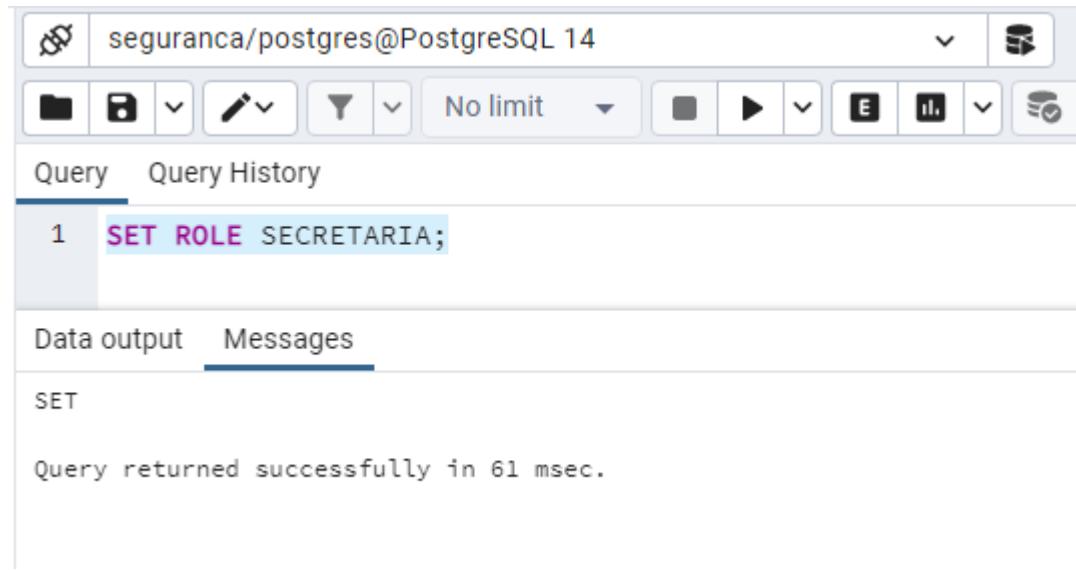
Isso funciona bem quando você já está conectado (por exemplo, como postgres) mas quer testar os privilégios de outro usuário, sem sair do pgAdmin.

Teste no PgAdmin

Regressemos ao usuário **postgres**.



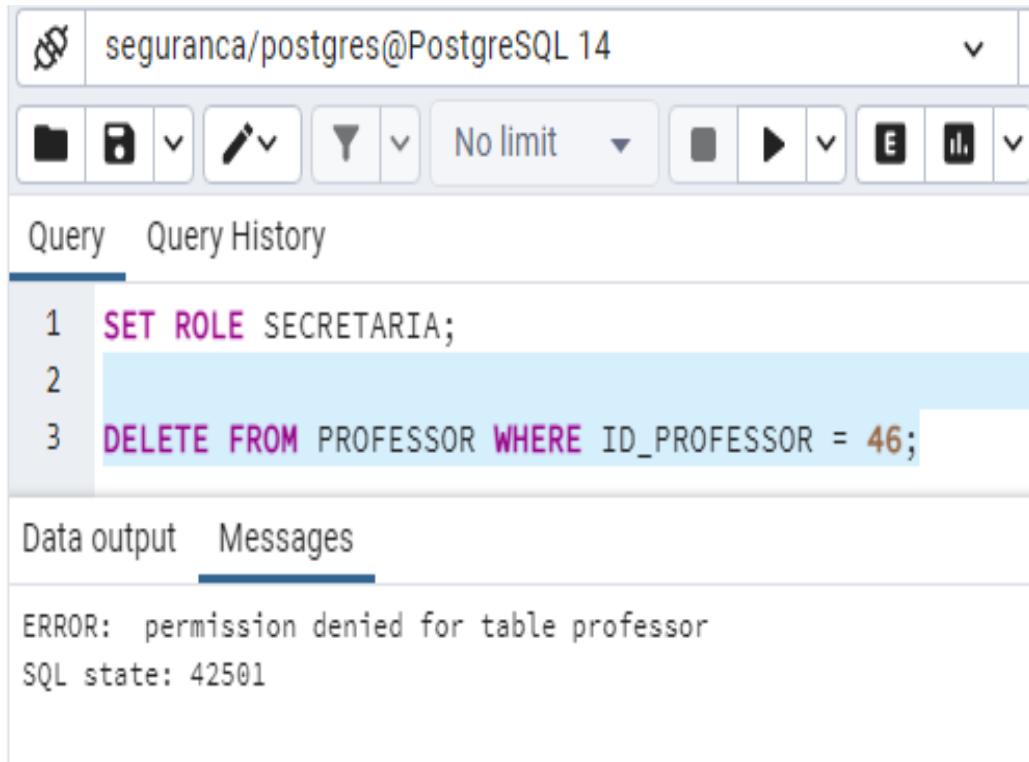
Teste no PgAdmin



A screenshot of the PgAdmin 4 interface. The title bar shows the connection details: 'segurança/postgres@PostgreSQL 14'. Below the title bar is a toolbar with various icons. The main area has two tabs: 'Query' (which is selected) and 'Query History'. In the 'Query' tab, there is a single line of code: '1 SET ROLE SECRETARIA;'. Below the code, under the 'Messages' tab, it says 'SET' and 'Query returned successfully in 61 msec.'

A partir desse momento, **todos os comandos que você executar serão feitos com os privilégios do usuário secretaria.**

Teste no PgAdmin



The screenshot shows the PgAdmin 4 interface. At the top, there's a connection bar with a user icon and the text "segurança/postgres@PostgreSQL 14". Below it is a toolbar with various icons for database management. The main area has two tabs: "Query" (selected) and "Query History". In the "Query" tab, there are three numbered SQL statements:

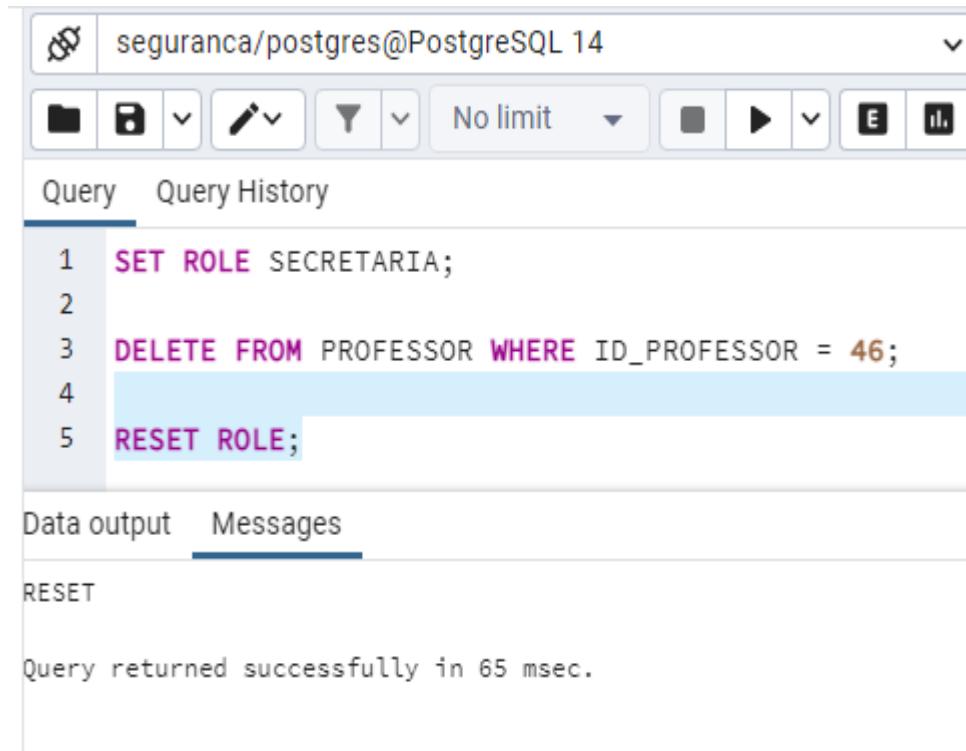
- 1 SET ROLE SECRETARIA;
- 2
- 3 DELETE FROM PROFESSOR WHERE ID_PROFESSOR = 46;

Below the query editor, there are two tabs: "Data output" and "Messages" (selected). The "Messages" tab displays an error message:

ERROR: permission denied for table professor
SQL state: 42501

SECRETARIA não tem permissão de efetuar exclusões na tabela **PROFESSOR**.

Teste no PgAdmin



A screenshot of the PgAdmin 4 interface. The title bar shows the connection information: 'segurança/postgres@PostgreSQL 14'. The toolbar includes icons for file operations, search, and execution. Below the toolbar, there are tabs for 'Query' (selected) and 'Query History'. The main area contains a numbered list of SQL commands:

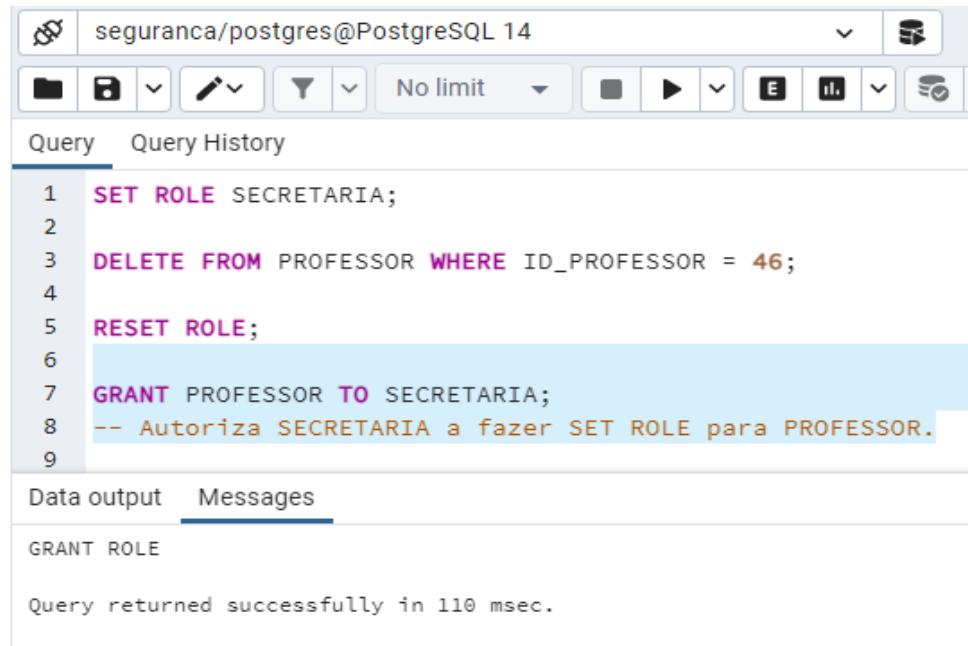
```
1 SET ROLE SECRETARIA;
2
3 DELETE FROM PROFESSOR WHERE ID_PROFESSOR = 46;
4
5 RESET ROLE;
```

Below the queries, there are tabs for 'Data output' and 'Messages'. The 'Messages' tab is selected, showing the message: 'Query returned successfully in 65 msec.'.

Agora regressamos ao nível de autorização do usuário **postgres**.

Teste no PgAdmin

Você só consegue usar SET ROLE outro_usuario se o usuário original tiver sido autorizado para tal.



The screenshot shows a PgAdmin 4 interface with a query editor. The connection is set to 'segurança/postgres@PostgreSQL 14'. The toolbar includes icons for file, copy, paste, search, and execution. The query history tab is selected. The main query window contains the following code:

```
1 SET ROLE SECRETARIA;
2
3 DELETE FROM PROFESSOR WHERE ID_PROFESSOR = 46;
4
5 RESET ROLE;
6
7 GRANT PROFESSOR TO SECRETARIA;
8 -- Autoriza SECRETARIA a fazer SET ROLE para PROFESSOR.
9
```

The line 'GRANT PROFESSOR TO SECRETARIA;' is highlighted with a light blue background. Below the query window, there are tabs for 'Data output' and 'Messages'. The 'Messages' tab is selected, showing the output 'GRANT ROLE' and the message 'Query returned successfully in 110 msec.'

Revogando Privilégios

```
1 SET ROLE SECRETARIA;
2
3 DELETE FROM PROFESSOR WHERE ID_PROFESSOR = 46;
4
5 RESET ROLE;
6
7 GRANT PROFESSOR TO SECRETARIA;
8 -- Autoriza SECRETARIA a fazer SET ROLE para PROFESSOR.
9
10 -- Remove todos os privilégios (SELECT, INSERT, UPDATE, DELETE) da tabela
11 -- PROFESSOR para o PUBLIC (ou seja, para todos os usuários).
12 -- Só o dono da tabela (neste caso, o postgres) ainda terá acesso.
13
14 REVOKE ALL ON TABLE PROFESSOR FROM PUBLIC;
15
```

Data output Messages

REVOKE

Query returned successfully in 54 msec.

Revogando Privilégios

```
9  
10 -- Remove todos os privilégios (SELECT, INSERT, UPDATE, DELETE) da tabela  
11 -- PROFESSOR para o PUBLIC (ou seja, para todos os usuários).  
12 -- Só o dono da tabela (neste caso, o postgres) ainda terá acesso.  
13  
14 REVOKE ALL ON TABLE PROFESSOR FROM PUBLIC;  
15  
16 -- O role SECRETARIA deixa de poder inserir registros na tabela ALUNO,  
17 -- mas mantém outros privilégios (como SELECT, se já tiver sido concedido).  
18 REVOKE INSERT ON TABLE ALUNO FROM SECRETARIA;  
19
```

Data output Messages

REVOKE

Query returned successfully in 97 msec.

Revogando Privilégios

```
10 -- Remove todos os privilégios (SELECT, INSERT, UPDATE, DELETE) da tabela
11 -- PROFESSOR para o PUBLIC (ou seja, para todos os usuários).
12 -- Só o dono da tabela (neste caso, o postgres) ainda terá acesso.
13
14 REVOKE ALL ON TABLE PROFESSOR FROM PUBLIC;
15
16 -- O role SECRETARIA deixa de poder inserir registros na tabela ALUNO,
17 -- mas mantém outros privilégios (como SELECT, se já tiver sido concedido).
18 REVOKE INSERT ON TABLE ALUNO FROM SECRETARIA;
19
20 -- O COORDENADOR poderá continuar lendo (SELECT) e inserindo (INSERT) cursos,
21 -- mas não poderá mais atualizar nem excluir registros dessa tabela.
22 REVOKE UPDATE, DELETE ON TABLE CURSO FROM COORDENADOR;
23
```

Data output

Messages

REVOKE

Query returned successfully in 53 msec.