

Table of Contents

Pre-requisito	2
Atividade 1	11
Atividade 2 - Parte 1: Conexão com Banco de Dados usando JDBC	19
Atividade 2 - Parte 2: Uso de ORMs	25
Conclusão	34
Referências	37

Pre-requisito

⚠ Link para o repositório com os códigos:
<http://github.com/mrpunkdasilva/learn-jdbc-and-hibernate>
(<http://github.com/mrpunkdasilva/learn-jdbc-and-hibernate>)

Para o laboratório, foi escolhido um sistema de biblioteca para a continuidade do projeto.

Iniciar container

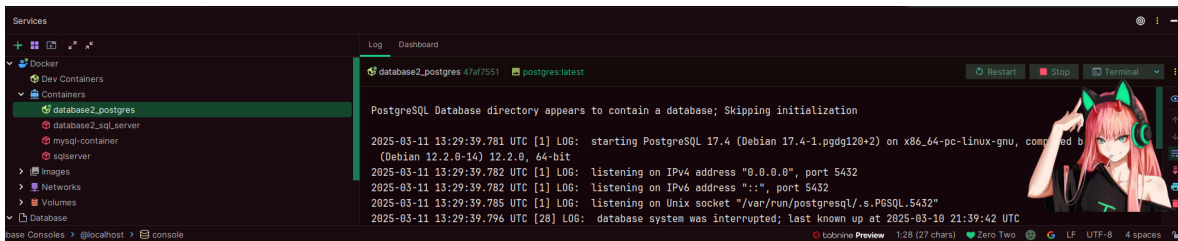
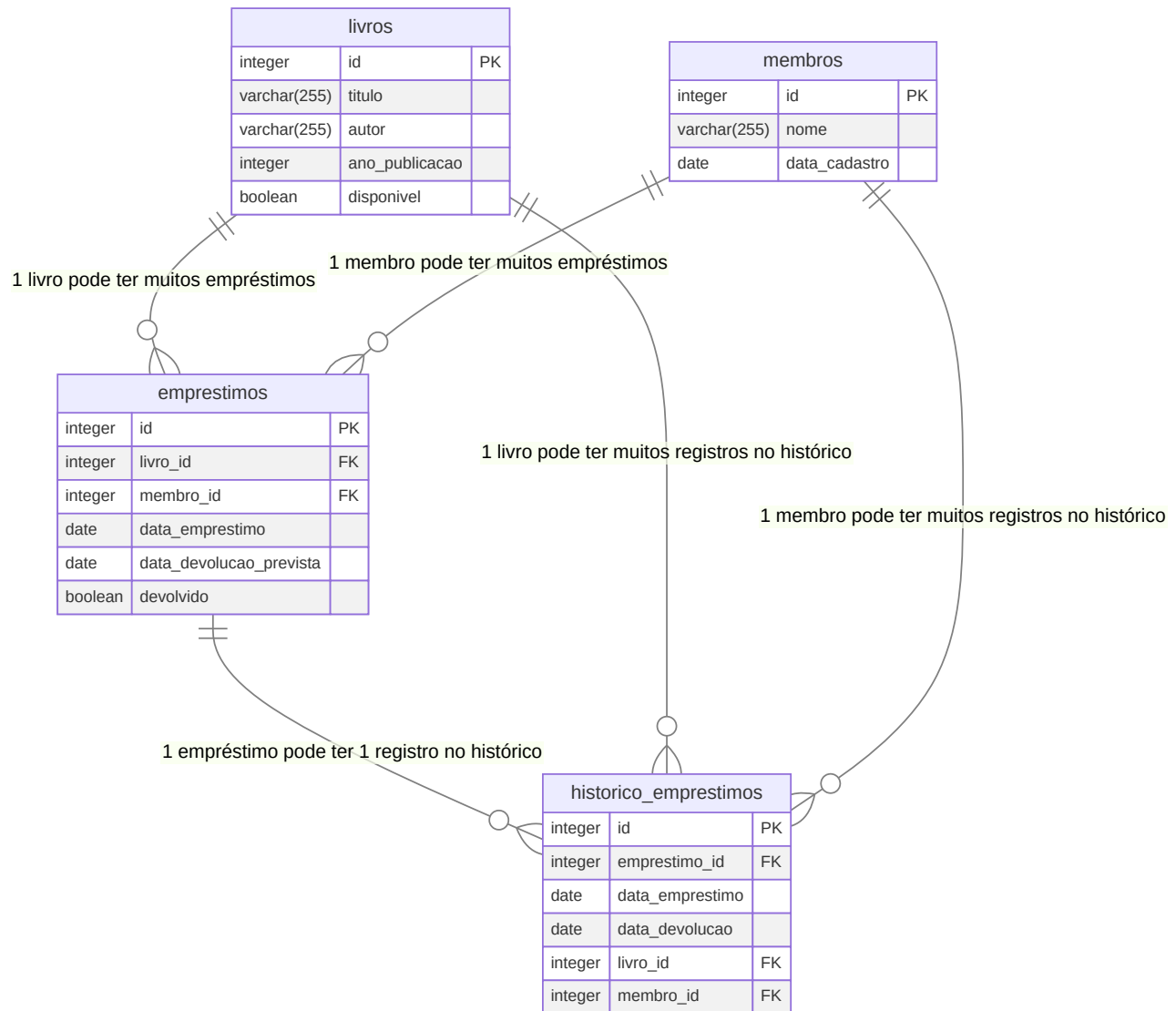


Imagem ilustrando o início do container

Modelo Entidade-Relacionamento



Criação do banco de dados

```
CREATE DATABASE BIBLIOTECA;
```

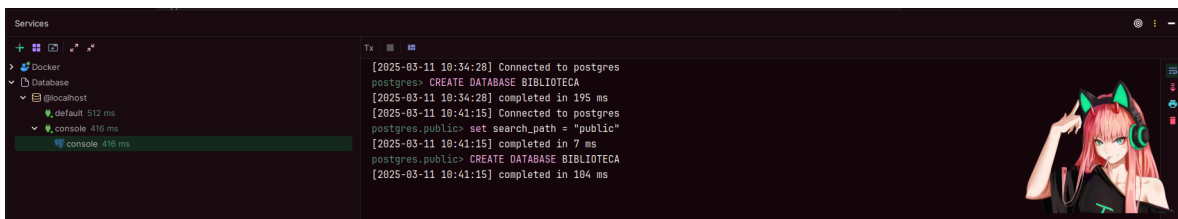
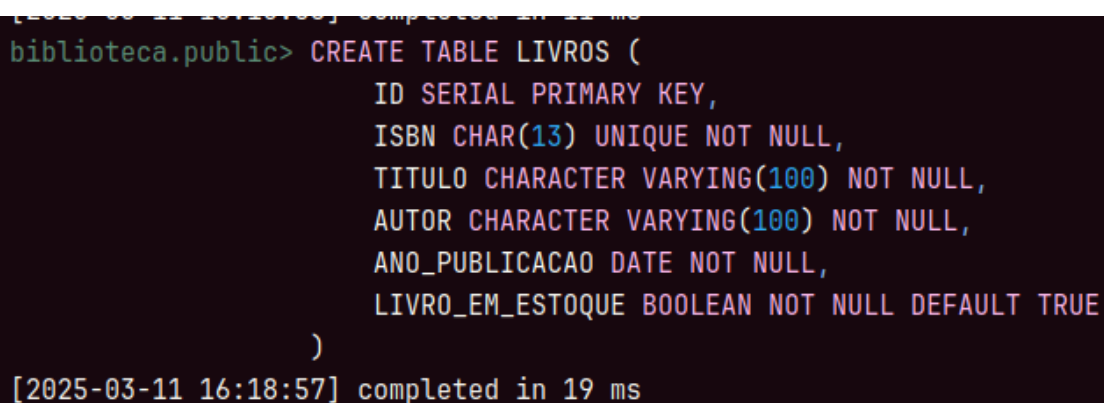


Imagem mostrando a criação do banco de dados da biblioteca

Criando as tabelas

- Livros:

```
CREATE TABLE LIVROS (  
    ID SERIAL PRIMARY KEY,  
    ISBN CHAR(13) UNIQUE NOT NULL,  
    TITULO CHARACTER VARYING(100) NOT NULL,  
    AUTOR CHARACTER VARYING(100) NOT NULL,  
    ANO_PUBLICACAO DATE NOT NULL,  
    LIVRO_EM_ESTOQUE BOOLEAN NOT NULL DEFAULT TRUE  
);
```



```
[2025-03-11 16:18:57] completed in 19 ms  
biblioteca.public> CREATE TABLE LIVROS (  
    ID SERIAL PRIMARY KEY,  
    ISBN CHAR(13) UNIQUE NOT NULL,  
    TITULO CHARACTER VARYING(100) NOT NULL,  
    AUTOR CHARACTER VARYING(100) NOT NULL,  
    ANO_PUBLICACAO DATE NOT NULL,  
    LIVRO_EM_ESTOQUE BOOLEAN NOT NULL DEFAULT TRUE  
)  
[2025-03-11 16:18:57] completed in 19 ms
```

Imagem da tabela de livros criada

- Membros:

```
CREATE TABLE MEMBROS (  
    ID SERIAL PRIMARY KEY,  
    NOME CHARACTER VARYING(120) NOT NULL,  
    DATA_CADASTRO DATE NOT NULL  
);
```

```
11
12 ✓ CREATE TABLE MEMBROS (
13     ID SERIAL PRIMARY KEY,
14     NOME CHARACTER VARYING(120) NOT NULL,
15     DATA_CADASTRO DATE NOT NULL
16 );
17
```

Services

- Docker
- Database
 - @localhost
 - default 512 ms
 - console 106 ms
 - console 106 ms

Tx

```
[2025-03-11 10:34:28] Connected to postgres
postgres> CREATE DATABASE BIBLIOTECA
[2025-03-11 10:34:28] completed in 195 ms
[2025-03-11 10:41:15] Connected to postgres
postgres.public> set search_path = "public"
[2025-03-11 10:41:15] completed in 7 ms
postgres.public> CREATE DATABASE BIBLIOTECA
[2025-03-11 10:41:15] completed in 104 ms
[2025-03-11 10:58:53] Disconnected
[2025-03-11 10:58:53] Connected to biblioteca
[2025-03-11 10:58:53] completed in 97 ms
biblioteca> CREATE TABLE MEMBROS (
    ID SERIAL PRIMARY KEY,
    NOME CHARACTER VARYING(120) NOT NULL,
    DATA_CADASTRO DATE NOT NULL
)
[2025-03-11 11:39:51] completed in 19 ms
```

Imagem da tabela de membros criada

- Empréstimos:

```
CREATE TABLE EMPRESTIMOS (
    ID SERIAL PRIMARY KEY,
    ID_LIVRO INTEGER,
    CONSTRAINT FK_LIVRO FOREIGN KEY (ID_LIVRO) REFERENCES LIVROS
(ID),
    ID_MEMBRO INTEGER,
    CONSTRAINT FK_MEMBRO FOREIGN KEY (ID_MEMBRO) REFERENCES
MEMBROS (ID),
    DATA_EMPRESTIMO DATE DEFAULT CURRENT_DATE,
    DATA_DEVOLUCAO_PREVISTA DATE,
    DEVOLVIDO BOOLEAN DEFAULT FALSE
);
```

```
console 50 ms
[2025-03-11 11:44:46] completed in 20 ms
biblioteca.public> CREATE TABLE EMPRESTIMOS (
    ID SERIAL PRIMARY KEY,
    ID_LIVRO INTEGER,
    CONSTRAINT FK_LIVRO FOREIGN KEY (ID_LIVRO) REFERENCES LIVROS (ID),
    ID_MEMBRO INTEGER,
    CONSTRAINT FK_MEMBRO FOREIGN KEY (ID_MEMBRO) REFERENCES MEMBROS (ID),
    DATA_EMPRESTIMO DATE DEFAULT CURRENT_DATE,
    DATA_DEVOLUCAO_PREVISTA DATE,
    DEVOLVIDO BOOLEAN DEFAULT FALSE
)
[2025-03-11 12:00:03] completed in 11 ms
```

Imagem da tabela de empréstimos criada

- **Histórico de Empréstimos:**

```
CREATE TABLE HISTORICO_EMPRESTIMOS (
    ID SERIAL PRIMARY KEY,
    DATA_DEVOLUCAO DATE,
    DATA_EMPRESTIMO DATE,

    ID_EMPRESTIMO INTEGER,
    CONSTRAINT FK_EMPRESTIMO FOREIGN KEY (ID_EMPRESTIMO)
REFERENCES LIVROS (ID),
    ID_LIVRO INTEGER,
    CONSTRAINT FK_EMPRESTIMO_LIVRO FOREIGN KEY (ID_LIVRO)
REFERENCES LIVROS (ID),
    ID_MEMBRO INTEGER,
    CONSTRAINT FK_EMPRESTIMO_MEMBRO FOREIGN KEY (ID_MEMBRO)
REFERENCES MEMBROS (ID)
);
```

```
28
29 CREATE TABLE HISTORICO_EMPRESTIMOS (
30     ID SERIAL PRIMARY KEY,
31     DATA_DEVOLUCAO DATE,
32     DATA_EMPRESTIMO DATE,
33
34     ID_EMPRESTIMO INTEGER,
35     CONSTRAINT FK_EMPRESTIMO FOREIGN KEY (ID_EMPRESTIMO) REFERENCES LIVROS (ID),
36     ID_LIVRO INTEGER,
37     CONSTRAINT FK_EMPRESTIMO_LIVRO FOREIGN KEY (ID_LIVRO) REFERENCES LIVROS (ID),
38     ID_MEMBRO INTEGER,
39     CONSTRAINT FK_EMPRESTIMO_MEMBRO FOREIGN KEY (ID_MEMBRO) REFERENCES MEMBROS (ID)
40 );
```

Services

+ Docker

Database

▼ @localhost

- default 134 ms
- ▼ console 41 ms

Tx

```
biblioteca.public> CREATE TABLE HISTORICO_EMPRESTIMOS (
    ID SERIAL PRIMARY KEY,
    DATA_DEVOLUCAO DATE,
    DATA_EMPRESTIMO DATE,
    ID_EMPRESTIMO INTEGER,
    CONSTRAINT FK_EMPRESTIMO FOREIGN KEY (ID_EMPRESTIMO) REFERENCES LIVROS (ID),
    ID_LIVRO INTEGER,
    CONSTRAINT FK_EMPRESTIMO_LIVRO FOREIGN KEY (ID_LIVRO) REFERENCES LIVROS (ID),
    ID_MEMBRO INTEGER,
    CONSTRAINT FK_EMPRESTIMO_MEMBRO FOREIGN KEY (ID_MEMBRO) REFERENCES MEMBROS (ID)
)
```

Imagem da tabela de histórico de empréstimos criada

Inserindo alguns dados iniciais na tabela

- Livros:

```
INSERT INTO LIVROS (ISBN, TITULO, AUTOR, ANO_PUBLICACAO,
LIVRO_EM_ESTOQUE)
VALUES
('9788535914849', '1984', 'George Orwell', '1949-06-08', TRUE),
('9788535914863', 'A Revolução dos Bichos', 'George Orwell',
'1945-08-17', TRUE),
('9788535914870', 'O Senhor dos Anéis', 'J.R.R. Tolkien', '1954-
07-29', TRUE),
('9788535914887', 'O Hobbit', 'J.R.R. Tolkien', '1937-09-21',
FALSE),
('9788535914894', 'Cem Anos de Solidão', 'Gabriel García Márquez',
'1967-05-30', TRUE);
```

```
41  -- INSERTS
42
43
44  INSERT INTO LIVROS (ISBN, TITULO, AUTOR, ANO_PUBLICACAO, LIVRO_EM_ESTOQUE)
45  VALUES
46  ( ISBN '9788535914849', TITULO '1984', AUTOR 'George Orwell', ANO_PUBLICACAO '1949-06-08', LIVRO_EM_ESTOQUE TRUE),
47  ( ISBN '9788535914863', TITULO 'A Revolução dos Bichos', AUTOR 'George Orwell', ANO_PUBLICACAO '1945-08-17', LIVRO_EM_ESTOQUE TRUE),
48  ( ISBN '9788535914870', TITULO 'O Senhor dos Anéis', AUTOR 'J.R.R. Tolkien', ANO_PUBLICACAO '1954-07-29', LIVRO_EM_ESTOQUE TRUE),
49  ( ISBN '9788535914887', TITULO 'O Hobbit', AUTOR 'J.R.R. Tolkien', ANO_PUBLICACAO '1937-09-21', LIVRO_EM_ESTOQUE FALSE),
50  ( ISBN '9788535914894', TITULO 'Cem Anos de Solidão', AUTOR 'Gabriel García Márquez', ANO_PUBLICACAO '1967-05-30', LIVRO_EM_ESTOQUE TRUE);
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

Imagem mostrando dados de livros inseridos

- **Membros:**

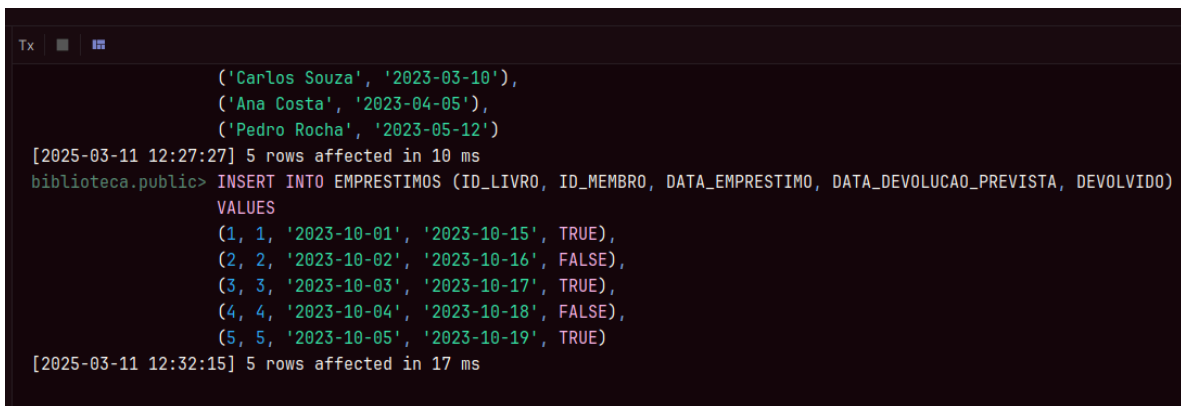
```
INSERT INTO MEMBROS (NOME, DATA_CADASTRO)
VALUES
('João Silva', '2023-01-15'),
('Maria Oliveira', '2023-02-20'),
('Carlos Souza', '2023-03-10'),
('Ana Costa', '2023-04-05'),
('Pedro Rocha', '2023-05-12');
```

```
Tx  [2025-03-11 12:23:15] 5 rows affected in 14 ms
biblioteca.public> INSERT INTO MEMBROS (NOME, DATA_CADASTRO)
VALUES
('João Silva', '2023-01-15'),
('Maria Oliveira', '2023-02-20'),
('Carlos Souza', '2023-03-10'),
('Ana Costa', '2023-04-05'),
('Pedro Rocha', '2023-05-12')
[2025-03-11 12:27:27] 5 rows affected in 10 ms
```

Imagem mostrando dados de membros inseridos

- Empréstimos:

```
INSERT INTO EMPRESTIMOS (ID_LIVRO, ID_MEMBRO, DATA_EMPRESTIMO,
DATA_DEVOLUCAO_PREVISTA, DEVOLVIDO)
VALUES
(1, 1, '2023-10-01', '2023-10-15', TRUE),
(2, 2, '2023-10-02', '2023-10-16', FALSE),
(3, 3, '2023-10-03', '2023-10-17', TRUE),
(4, 4, '2023-10-04', '2023-10-18', FALSE),
(5, 5, '2023-10-05', '2023-10-19', TRUE);
```



```
Tx [ ] [ ]
      ('Carlos Souza', '2023-03-10'),
      ('Ana Costa', '2023-04-05'),
      ('Pedro Rocha', '2023-05-12')
[2025-03-11 12:27:27] 5 rows affected in 10 ms
biblioteca.public> INSERT INTO EMPRESTIMOS (ID_LIVRO, ID_MEMBRO, DATA_EMPRESTIMO, DATA_DEVOLUCAO_PREVISTA, DEVOLVIDO)
VALUES
(1, 1, '2023-10-01', '2023-10-15', TRUE),
(2, 2, '2023-10-02', '2023-10-16', FALSE),
(3, 3, '2023-10-03', '2023-10-17', TRUE),
(4, 4, '2023-10-04', '2023-10-18', FALSE),
(5, 5, '2023-10-05', '2023-10-19', TRUE)
[2025-03-11 12:32:15] 5 rows affected in 17 ms
```

Imagem mostrando dados de empréstimos inseridos

- Histórico de Empréstimos:

```
INSERT INTO HISTORICO_EMPRESTIMOS (DATA_DEVOLUCAO,
DATA_EMPRESTIMO, ID_EMPRESTIMO, ID_LIVRO, ID_MEMBRO)
VALUES
('2023-10-15', '2023-10-01', 1, 1, 1),
('2023-10-16', '2023-10-02', 2, 2, 2),
('2023-10-17', '2023-10-03', 3, 3, 3),
('2023-10-18', '2023-10-04', 4, 4, 4),
('2023-10-19', '2023-10-05', 5, 5, 5);
```

```
Tx ■ ■ ■
(3, 3, '2023-10-03', '2023-10-17', TRUE),
(4, 4, '2023-10-04', '2023-10-18', FALSE),
(5, 5, '2023-10-05', '2023-10-19', TRUE)
[2025-03-11 12:32:15] 5 rows affected in 17 ms
biblioteca.public> INSERT INTO HISTORICO_EMPRESTIMOS (DATA_DEVOLUCAO, DATA_EMPRESTIMO, ID_EMPRESTIMO, ID_LIVRO, ID_MEMBRO)
VALUES
('2023-10-15', '2023-10-01', 1, 1, 1),
('2023-10-16', '2023-10-02', 2, 2, 2),
('2023-10-17', '2023-10-03', 3, 3, 3),
('2023-10-18', '2023-10-04', 4, 4, 4),
('2023-10-19', '2023-10-05', 5, 5, 5)
[2025-03-11 12:34:52] 5 rows affected in 13 ms
```

Imagem mostrando dados de histórico de empréstimos inseridos

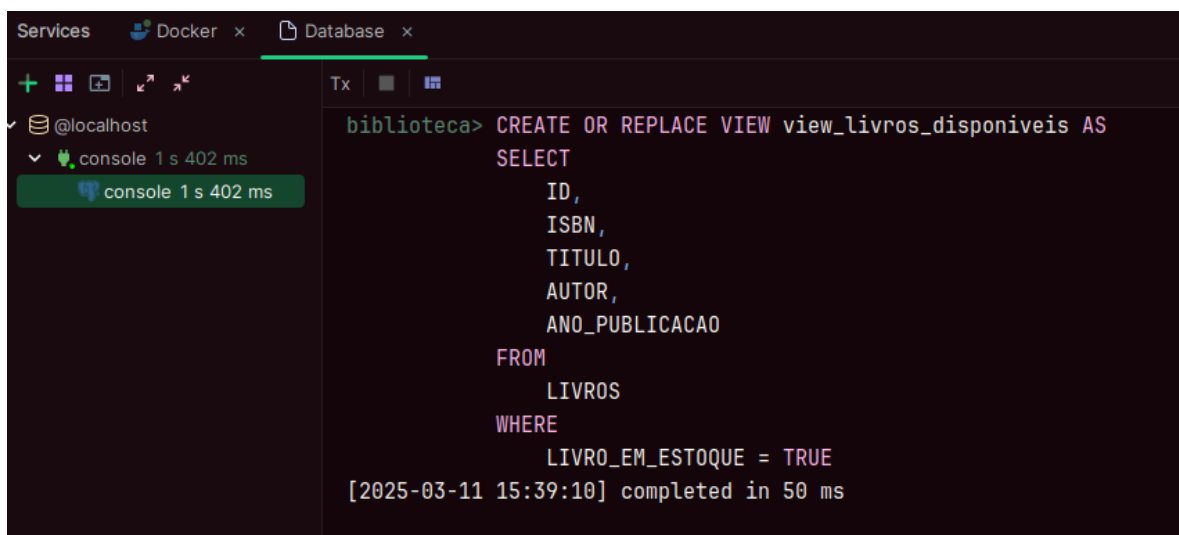
Atividade 1

Views

1. View para Visualização de Livros Disponíveis

Essa view retorna apenas os livros que estão disponíveis para empréstimo (LIVRO_EM_ESTOQUE = TRUE).

```
CREATE OR REPLACE VIEW view_livros_disponiveis AS
SELECT
    ID,
    ISBN,
    TITULO,
    AUTOR,
    ANO_PUBLICACAO
FROM
    LIVROS
WHERE
    LIVRO_EM_ESTOQUE = TRUE;
```

A screenshot of a database terminal window. The window has a dark background with a sidebar on the left showing a file explorer and a list of connections. The main area displays a SQL command being executed in a terminal. The command is to create or replace a view named 'view_livros_disponiveis' which selects columns ID, ISBN, TITULO, AUTOR, and ANO_PUBLICACAO from the LIVROS table where LIVRO_EM_ESTOQUE is TRUE. The command is preceded by 'biblioteca>'. The output shows the command completed successfully in 50 ms on 2025-03-11 at 15:39:10.

```
biblioteca> CREATE OR REPLACE VIEW view_livros_disponiveis AS
SELECT
    ID,
    ISBN,
    TITULO,
    AUTOR,
    ANO_PUBLICACAO
FROM
    LIVROS
WHERE
    LIVRO_EM_ESTOQUE = TRUE
[2025-03-11 15:39:10] completed in 50 ms
```

Resultado da execução da view de livros disponíveis

2. Procedimento Armazenado para Empréstimo de Livros

Esse procedimento armazenado gerencia o empréstimo de um livro, verificando se o livro está disponível e atualizando o status do livro.

```
CREATE OR REPLACE PROCEDURE realizar_emprestimo(
    p_id_livro INTEGER,
    p_id_membro INTEGER,
    p_data_devolucao_prevista DATE
)
LANGUAGE plpgsql
AS $$
BEGIN
    -- Verifica se o livro está disponível
    IF NOT EXISTS (SELECT 1 FROM LIVROS WHERE ID = p_id_livro AND
LIVRO_EM_ESTOQUE = TRUE) THEN
        RAISE EXCEPTION 'Livro não está disponível para
empréstimo.';
    END IF;

    -- Insere o empréstimo na tabela EMPRESTIMOS
    INSERT INTO EMPRESTIMOS (ID_LIVRO, ID_MEMBRO,
DATA_DEVOLUCAO_PREVISTA)
    VALUES (p_id_livro, p_id_membro, p_data_devolucao_prevista);

    -- Atualiza o status do livro para indisponível
    UPDATE LIVROS
    SET LIVRO_EM_ESTOQUE = FALSE
    WHERE ID = p_id_livro;

    RAISE NOTICE 'Empréstimo realizado com sucesso.';
END;
$$;
```

```
72
93 -- TRIGGERS
94 CREATE OR REPLACE PROCEDURE realizar_emprestimo(
95     p_id_livro INTEGER,
96     p_id_membro INTEGER,
97     p_data_devolucao_prevista DATE
98 )
99 LANGUAGE plpgsql
100 AS $$
101 BEGIN
102     -- Verifica se o livro está disponível
103     IF NOT EXISTS (SELECT 1 FROM LIVROS WHERE ID = p_id_livro AND LIVRO_EM_ESTOQUE = TRUE) THEN
104         RAISE EXCEPTION 'Livro não está disponível para empréstimo.';
105     END IF;
106
107     -- Insere o empréstimo na tabela EMPRESTIMOS
108     INSERT INTO EMPRESTIMOS (ID_LIVRO, ID_MEMBRO, DATA_DEVOLUCAO_PREVISTA)
109     VALUES (p_id_livro, p_id_membro, p_data_devolucao_prevista);
110
111     -- Atualiza o status do livro para indisponível
112     UPDATE LIVROS
113     SET LIVRO_EM_ESTOQUE = FALSE
114     WHERE ID = p_id_livro;
115
116     RAISE NOTICE 'Empréstimo realizado com sucesso.';
117 END;
118 $$
[2025-03-11 15:41:44] completed in 16 ms
```

Confirmação da criação do procedimento de empréstimo

3. Procedimento Armazenado para Devolução de Livros

Esse procedimento armazenado gerencia a devolução de um livro, atualizando o status do livro e registrando a devolução no histórico.

```
CREATE OR REPLACE PROCEDURE realizar_devolucao(
    p_id_emprestimo INTEGER
)
LANGUAGE plpgsql
AS $$
BEGIN
    -- Verifica se o empréstimo existe e se o livro ainda não foi devolvido
    IF NOT EXISTS (SELECT 1 FROM EMPRESTIMOS WHERE ID =
p_id_emprestimo AND DEVOLVIDO = FALSE) THEN
        RAISE EXCEPTION 'Empréstimo não encontrado ou livro já
```

```

devolvido.';
    END IF;

    -- Atualiza o status do livro para disponível
    UPDATE LIVROS
    SET LIVRO_EM_ESTOQUE = TRUE
    WHERE ID = (SELECT ID_LIVRO FROM EMPRESTIMOS WHERE ID =
p_id_emprestimo);

    -- Marca o empréstimo como devolvido
    UPDATE EMPRESTIMOS
    SET DEVOLVIDO = TRUE
    WHERE ID = p_id_emprestimo;

    -- Insere o registro no histórico de empréstimos
    INSERT INTO HISTORICO_EMPRESTIMOS (DATA_DEVOLUCAO,
DATA_EMPRESTIMO, ID_EMPRESTIMO, ID_LIVRO, ID_MEMBRO)
    SELECT
        CURRENT_DATE,
        DATA_EMPRESTIMO,
        ID,
        ID_LIVRO,
        ID_MEMBRO
    FROM
        EMPRESTIMOS
    WHERE
        ID = p_id_emprestimo;

    RAISE NOTICE 'Devolução realizada com sucesso.';
END;
$$;

```

```
119
120 CREATE OR REPLACE PROCEDURE realizar_devolucao(
121     p_id_emprestimo INTEGER
122 )
123 LANGUAGE plpgsql
124 $$
125 BEGIN
126     -- Verifica se o empréstimo existe e se o livro ainda não foi devolvido
127     IF NOT EXISTS (SELECT 1 FROM EMPRESTIMOS WHERE ID = p_id_emprestimo AND DEVOLVIDO = FALSE) THEN
128         RAISE EXCEPTION 'Empréstimo não encontrado ou livro já devolvido.';
129     END IF;
130
131     -- Atualiza o status do livro para disponível
132
133     INSERT INTO HISTORICO_EMPRESTIMOS (DATA_DEVOLUCAO, DATA_EMPRESTIMO, ID_EMPRESTIMO, ID_LIVRO, ID_MEMBRO)
134     SELECT
135         CURRENT_DATE,
136         DATA_EMPRESTIMO,
137         ID,
138         ID_LIVRO,
139         ID_MEMBRO
140     FROM
141         EMPRESTIMOS
142     WHERE
143         ID = p_id_emprestimo;
144
145     RAISE NOTICE 'Devolução realizada com sucesso.';
146 END;
147 $$
[2025-03-11 16:02:17] completed in 31 ms
```

Confirmação da criação do procedimento de devolução

4. Trigger para Registrar Empréstimos no Histórico

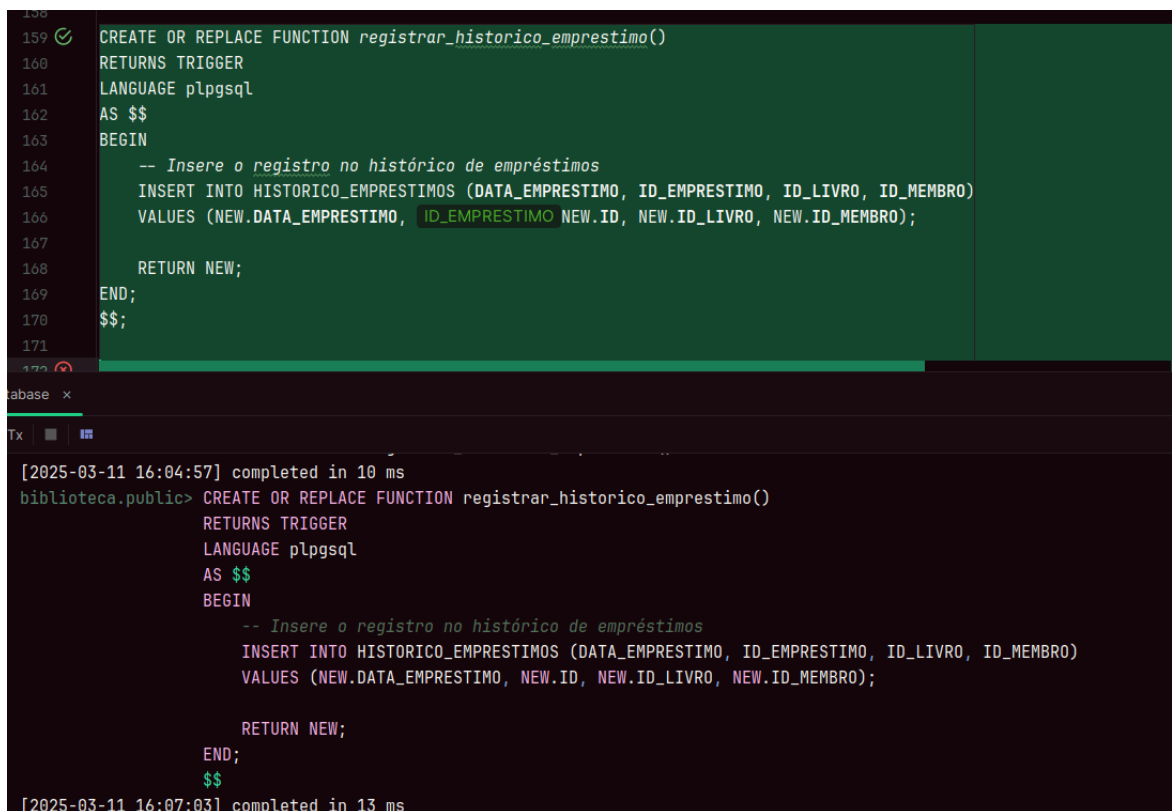
Esse trigger é acionado automaticamente após a inserção de um novo empréstimo na tabela `EMPRESTIMOS`, registrando o empréstimo no histórico.

```
CREATE OR REPLACE FUNCTION registrar_historico_emprestimo()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    -- Insere o registro no histórico de empréstimos
    INSERT INTO HISTORICO_EMPRESTIMOS (DATA_EMPRESTIMO,
    ID_EMPRESTIMO, ID_LIVRO, ID_MEMBRO)
    VALUES (NEW.DATA_EMPRESTIMO, NEW.ID, NEW.ID_LIVRO,
    NEW.ID_MEMBRO);

    RETURN NEW;
END;
```

```
$$;
```

```
CREATE TRIGGER trigger_registrar_historico_emprestimo  
AFTER INSERT ON EMPRESTIMOS  
FOR EACH ROW  
EXECUTE FUNCTION registrar_historico_emprestimo();
```



```
159 CREATE OR REPLACE FUNCTION registrar_historico_emprestimo()  
160 RETURNS TRIGGER  
161 LANGUAGE plpgsql  
162 AS $$  
163 BEGIN  
164     -- Insere o registro no histórico de empréstimos  
165     INSERT INTO HISTORICO_EMPRESTIMOS (DATA_EMPRESTIMO, ID_EMPRESTIMO, ID_LIVRO, ID_MEMBRO)  
166     VALUES (NEW.DATA_EMPRESTIMO, ID_EMPRESTIMO NEW.ID, NEW.ID_LIVRO, NEW.ID_MEMBRO);  
167  
168     RETURN NEW;  
169 END;  
170 $$;  
171
```

Database x

Tx

```
[2025-03-11 16:04:57] completed in 10 ms  
biblioteca.public> CREATE OR REPLACE FUNCTION registrar_historico_emprestimo()  
RETURNS TRIGGER  
LANGUAGE plpgsql  
AS $$  
BEGIN  
    -- Insere o registro no histórico de empréstimos  
    INSERT INTO HISTORICO_EMPRESTIMOS (DATA_EMPRESTIMO, ID_EMPRESTIMO, ID_LIVRO, ID_MEMBRO)  
    VALUES (NEW.DATA_EMPRESTIMO, NEW.ID, NEW.ID_LIVRO, NEW.ID_MEMBRO);  
  
    RETURN NEW;  
END;  
$$  
[2025-03-11 16:07:03] completed in 13 ms
```

Confirmação da criação do trigger para registro de empréstimos

5. View para Visualização do Histórico de Empréstimos

Essa view retorna o histórico completo de empréstimos, incluindo informações sobre os livros e membros.

```
CREATE OR REPLACE VIEW view_historico_emprestimos AS  
SELECT  
    HE.ID AS HISTORICO_ID,  
    HE.DATA_EMPRESTIMO,  
    HE.DATA_DEVOLUCAO,  
    L.TITULO AS LIVRO,
```



```
M.NOME AS MEMBRO  
FROM  
  HISTORICO_EMPRESTIMOS HE  
JOIN  
  LIVROS L ON HE.ID_LIVRO = L.ID  
JOIN  
  MEMBROS M ON HE.ID_MEMBRO = M.ID;
```

```
176
177 ✓ CREATE OR REPLACE VIEW view_historico_emprestimos AS
178 SELECT
179     HE.ID AS HISTORICO_ID,
180     HE.DATA_EMPRESTIMO,
181     HE.DATA_DEVOLUCAO,
182     L.TITULO AS LIVRO,
183     M.NOME AS MEMBRO
184 FROM
185     HISTORICO_EMPRESTIMOS HE
186 JOIN
187     LIVROS L 1..n<->1: ON HE.ID_LIVRO = L.ID
188 JOIN
189     MEMBROS M 1..n<->1: ON HE.ID_MEMBRO = M.ID;
190
191
192
193
```

Database x

Tx ■ ■

```
SELECT
    HE.ID AS HISTORICO_ID,
    HE.DATA_EMPRESTIMO,
    HE.DATA_DEVOLUCAO,
    L.TITULO AS LIVRO,
    M.NOME AS MEMBRO
FROM
    HISTORICO_EMPRESTIMOS HE
JOIN
    LIVROS L ON HE.ID_LIVRO = L.ID
JOIN
    MEMBROS M ON HE.ID_MEMBRO = M.ID
[2025-03-11 16:10:58] completed in 11 ms
```

Resultado da execução da view de histórico de empréstimos

Atividade 2 - Parte 1: Conexão com Banco de Dados usando JDBC

Passo 1: Configuração do Projeto Java

- Adicione a dependência do driver JDBC do PostgreSQL no seu projeto. Se estiver usando Maven, adicione o seguinte ao seu `pom.xml`:

```
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <version>42.6.0</version>
</dependency>
```

Passo 2: Estabelecer uma Conexão

Aqui está um exemplo de como estabelecer uma conexão com o banco de dados PostgreSQL:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DatabaseConnection {
    private static final String URL =
"jdbc:postgresql://localhost:5432/nome_do_banco";
    private static final String USER = "usuario";
    private static final String PASSWORD = "senha";

    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }
}
```

Passo 3: Executar Operações CRUD

Aqui está um exemplo de como executar operações CRUD usando JDBC:

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class LivroDAO {

    public void inserirLivro(String isbn, String titulo, String
autor, String anoPublicacao) {
        String sql = "INSERT INTO LIVROS (ISBN, TITULO, AUTOR,
ANO_PUBLICACAO) VALUES (?, ?, ?, ?)";

        try (Connection conn = DatabaseConnection.getConnection();
            PreparedStatement pstmt = conn.prepareStatement(sql))
        {

            pstmt.setString(1, isbn);
            pstmt.setString(2, titulo);
            pstmt.setString(3, autor);
            pstmt.setDate(4,
java.sql.Date.valueOf(anoPublicacao));
            pstmt.executeUpdate();

            System.out.println("Livro inserido com sucesso!");

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public void listarLivros() {
        String sql = "SELECT * FROM LIVROS";
```

```

        try (Connection conn = DatabaseConnection.getConnection();
            PreparedStatement pstmt = conn.prepareStatement(sql);
            ResultSet rs = pstmt.executeQuery()) {

            while (rs.next()) {
                System.out.println("ID: " + rs.getInt("ID") +
                    ", Título: " + rs.getString("TITULO") +
                    ", Autor: " + rs.getString("AUTOR"));
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

```

Passo 4: Chamar Procedimentos Armazenados

Aqui está um exemplo de como chamar o procedimento armazenado `realizar_emprestimo`:

```

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.SQLException;

public class EmprestimoService {

    public void realizarEmprestimo(int idLivro, int idMembro,
        String dataDevolucaoPrevista) {
        String sql = "{call realizar_emprestimo(?, ?, ?)}";

        try (Connection conn = DatabaseConnection.getConnection();
            CallableStatement cstmt = conn.prepareCall(sql)) {

            cstmt.setInt(1, idLivro);
            cstmt.setInt(2, idMembro);
            cstmt.setDate(3,

```

```

java.sql.Date.valueOf(dataDevolucaoPrevista));
        pstmt.execute();

        System.out.println("Empréstimo realizado com
sucesso!");
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}

```

Passo 5: Usando as classes

```

public class Main {
    public static void main(String[] args) throws SQLException {
        LivroDAO livroDAO = new LivroDAO();

        // Inserir um livro
        livroDAO.inserirLivro("9788535914849", "1984", "George
Orwell", "1949-06-08");

        // Listar livros
        livroDAO.listarLivros();

        // Atualizar um livro
        livroDAO.atualizarLivro(1, "1984 - Edição Especial");

        // Excluir um livro
        livroDAO.excluirLivro(1);

        // Realizar empréstimo
        EmpréstimoService empréstimoService = new
EmpréstimoService();
        empréstimoService.realizarEmpréstimo(2, 1, "2023-10-20");
    }
}

```

```
}
}
```

Executando

	id	nome	data_cadastro
1	1	João Silva	2023-01-15
2	2	Maria Oliveira	2023-02-20
3	3	Carlos Souza	2023-03-10
4	4	Ana Costa	2023-04-05
5	5	Pedro Rocha	2023-05-12

Exibindo tabela MEMBROS

	id	isbn	titulo	autor	ano_publicacao	livro_em_estoque
1	7	9788535914863	A Revolução dos...	George Orwell	1945-08-17	• true
2	8	9788535914870	O Senhor dos An...	J.R.R. Tolkien	1954-07-29	• true
3	9	9788535914887	O Hobbit	J.R.R. Tolkien	1937-09-21	• false
4	10	9788535914894	Cem Anos de Sol...	Gabriel García...	1967-05-30	• true
5	12	123456789125	1984	George Orwell	1949-06-08	• true
6	5	123456789123	1984 - Edição E...	George Orwell	1949-06-08	• true

Exibindo tabela LIVROS

	id	data_devolucao	data_emprestimo	id_emprestimo	id_livro	id_membro
1	1	2023-10-20	2025-03-11	2	8	1

Exibindo tabela HISTORICO EMPRESTIMOS

```
Run Main x
>> [icons]
/home/MrPunkdaSilva/.jdk/openjdk-23.0.2/bin/java -javaagent:/home/MrPunkdaSilva/.local/share/JetBrains
-Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath /home/MrPunkdaSilva/IdeaProjects/
.m2/repository/org/postgresql/postgresql/42.2.14/postgresql-42.2.14.jar org.gustavojesus.Main
ID: 7, Título: A Revolução dos Bichos, Autor: George Orwell, Ano de Publicação: 1945-08-17
ID: 8, Título: O Senhor dos Anéis, Autor: J.R.R. Tolkien, Ano de Publicação: 1954-07-29
ID: 9, Título: O Hobbit, Autor: J.R.R. Tolkien, Ano de Publicação: 1937-09-21
ID: 10, Título: Cem Anos de Solidão, Autor: Gabriel García Márquez, Ano de Publicação: 1967-05-30
ID: 12, Título: 1984, Autor: George Orwell, Ano de Publicação: 1949-06-08
ID: 5, Título: 1984 - Edição Especial, Autor: George Orwell, Ano de Publicação: 1949-06-08
ID: 14, Título: 1984, Autor: George Orwell, Ano de Publicação: 1949-06-08
Empréstimo realizado com sucesso!

Process finished with exit code 0
```

Execução bem sucedida com JDBC

Atividade 2 - Parte 2: Uso de ORMs

Passo 1: Configuração do Hibernate

Adicionar Dependências no pom.xml

Adicione as dependências do Hibernate e do driver JDBC do PostgreSQL no arquivo `pom.xml` do seu projeto Maven:

```
<dependencies>
  <!-- Hibernate Core -->
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>6.3.1.Final</version>
  </dependency>

  <!-- Driver JDBC do PostgreSQL -->
  <dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.6.0</version>
  </dependency>

  <!-- Java Persistence API (JPA) -->
  <dependency>
    <groupId>jakarta.persistence</groupId>
    <artifactId>jakarta.persistence-api</artifactId>
    <version>3.1.0</version>
  </dependency>
</dependencies>
```

Passo 2: Configuração do hibernate.cfg.xml

Crie o arquivo `hibernate.cfg.xml` na pasta `src/main/resources`. Esse arquivo contém as configurações de conexão com o banco de dados e o mapeamento das entidades.

```

<hibernate-configuration>
  <session-factory>
    <!-- Configurações de conexão com o banco de dados -->
    <property
name="hibernate.connection.driver_class">org.postgresql.Driver</pr
operty>
    <property
name="hibernate.connection.url">jdbc:postgresql://localhost:5432/b
iblioteca</property>
    <property
name="hibernate.connection.username">usuario</property>
    <property
name="hibernate.connection.password">senha</property>

    <!-- Dialeto do PostgreSQL -->
    <property
name="hibernate.dialect">org.hibernate.dialect.PostgreSQLDialect</
property>

    <!-- Atualiza o schema do banco de dados automaticamente -
->
    <property name="hibernate.hbm2ddl.auto">update</property>

    <!-- Exibe as consultas SQL no console -->
    <property name="show_sql">true</property>

    <!-- Mapeamento das entidades -->
    <mapping class="com.exemplo.model.Livro"/>
    <mapping class="com.exemplo.model.Membro"/>
  </session-factory>
</hibernate-configuration>

```

Passo 3: Mapeamento de Entidades

Crie as classes de entidade que representam as tabelas do banco de dados. Essas classes devem ser anotadas com as anotações do JPA (Java Persistence API).

Classe Livro

```
package com.exemplo.model;

import jakarta.persistence.*;
import java.util.Date;

@Entity
@Table(name = "LIVROS")
public class Livro {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String isbn;
    private String titulo;
    private String autor;

    @Column(name = "ANO_PUBLICACAO")
    private Date anoPublicacao;

    @Column(name = "LIVRO_EM_ESTOQUE")
    private boolean livroEmEstoque;

    // Getters e Setters
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getIsbn() {
        return isbn;
    }
}
```

```
public void setIsbn(String isbn) {
    this.isbn = isbn;
}

public String getTitulo() {
    return titulo;
}

public void setTitulo(String titulo) {
    this.titulo = titulo;
}

public String getAutor() {
    return autor;
}

public void setAutor(String autor) {
    this.autor = autor;
}

public Date getAnoPublicacao() {
    return anoPublicacao;
}

public void setAnoPublicacao(Date anoPublicacao) {
    this.anoPublicacao = anoPublicacao;
}

public boolean isLivroEmEstoque() {
    return livroEmEstoque;
}

public void setLivroEmEstoque(boolean livroEmEstoque) {
    this.livroEmEstoque = livroEmEstoque;
}
}
```

Classe Membro

```
package com.exemplo.model;

import jakarta.persistence.*;
import java.util.Date;

@Entity
@Table(name = "MEMBROS")
public class Membro {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String nome;

    @Column(name = "DATA_CADASTRO")
    private Date dataCadastro;

    // Getters e Setters
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public Date getDataCadastro() {
```

```

        return dataCadastro;
    }

    public void setDataCadastro(Date dataCadastro) {
        this.dataCadastro = dataCadastro;
    }
}

```

Passo 4: Operações com Hibernate

Crie uma classe de serviço para realizar operações CRUD usando o Hibernate.

Classe LivroService

```

package com.exemplo.service;

import com.exemplo.model.Livro;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class LivroService {

    private static final SessionFactory sessionFactory = new
Configuration()
        .configure("hibernate.cfg.xml")
        .buildSessionFactory();

    // Método para inserir um livro
    public void inserirLivro(Livro livro) {
        try (Session session = sessionFactory.openSession()) {
            session.beginTransaction();
            session.persist(livro);
            session.getTransaction().commit();
            System.out.println("Livro inserido com sucesso!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```

    }
}

// Método para buscar um livro por ID
public Livro buscarLivroPorId(int id) {
    try (Session session = sessionFactory.openSession()) {
        return session.get(Livro.class, id);
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
}

```

Passo 5: Testando as Operações

Crie uma classe `Main` para testar as operações com Hibernate.

Classe Main

```

package com.exemplo;

import com.exemplo.model.Livro;
import com.exemplo.service.LivroService;

import java.util.Date;

public class Main {
    public static void main(String[] args) {
        LivroService livroService = new LivroService();

        // Criar um novo livro
        Livro livro = new Livro();
        livro.setIsbn("9788535914849");
        livro.setTitulo("1984");
        livro.setAutor("George Orwell");
        livro.setAnoPublicacao(new Date());
    }
}

```

```

        livro.setLivroEmEstoque(true);

        // Inserir o livro no banco de dados
        livroService.inserirLivro(livro);

        // Buscar o livro por ID
        Livro livroEncontrado = livroService.buscarLivroPorId(1);
        if (livroEncontrado != null) {
            System.out.println("Livro encontrado: " +
livroEncontrado.getTitulo());
        } else {
            System.out.println("Livro não encontrado.");
        }
    }
}

```

Executando



```

Run
Main
org.hibernate.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
mar. 11, 2025 7:49:54 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001003: Autocommit mode: false
mar. 11, 2025 7:49:54 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl$PooledConnections <init>
INFO: HHH10001115: Connection pool size: 20 (min=1)
mar. 11, 2025 7:49:54 PM org.hibernate.engine.jdbc.dialect.internal.DialectFactoryImpl constructDialect
WARN: HHH90000025: PostgreSQLDialect does not need to be specified explicitly using 'hibernate.dialect' (remove the property setting and it will be selected by default)
mar. 11, 2025 7:49:58 PM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initiateService
INFO: HHH0000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform integration)
mar. 11, 2025 7:49:58 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess] (non-JTA) DDL execution was not in auto-commit mode; the Connection 'local transaction' will be committed and the Connection will be set into auto-commit mode.
Hibernate: insert into livros (ANO_PUBLICACAO,autor,isbn,LIVRO_EM_ESTOQUE,titulo) values (?, ?, ?, ?, ?)
Livro inserido com sucesso!
Hibernate: select l1_0.id,l1_0.ANO_PUBLICACAO,l1_0.autor,l1_0.isbn,l1_0.LIVRO_EM_ESTOQUE,l1_0.titulo from livros l1_0 where l1_0.id=?
Livro encontrado: 1984

Process finished with exit code 0

```

Exeução do código com sucesso, o livro "1984" foi criado e achado dentro do banco

WHERE						
ORDER BY						
	id	isbn	titulo	autor	ano_publicacao	livro_em_estoque
1	9	9788535914887	O Hobbit	J.R.R. Tolkien	1937-09-21 00:00:00.000000	false
2	10	9788535914894	Cem Anos de Solidão	Gabriel Garcia Márquez	1967-05-30 00:00:00.000000	true
3	12	123456789125	1984	George Orwell	1949-06-08 00:00:00.000000	true
4	14	123456789155	1984	George Orwell	1949-06-08 00:00:00.000000	true
5	5	123456789123	1984 - Edição Especial	George Orwell	1949-06-08 00:00:00.000000	true
6	8	9788535914870	O Senhor dos Anéis	J.R.R. Tolkien	1954-07-29 00:00:00.000000	false
7	15	9788535914849	1984	George Orwell	2025-03-11 00:00:00.000000	true
8	17	9788535914149	1984	George Orwell	2025-03-11 00:00:00.000000	true
9	18	1234567899999	1984	George Orwell	2025-03-11 00:00:00.000000	true
10	21	1239967899999	1984	George Orwell	2025-03-11 19:48:39.457000	true
11	23	1239969999999	1984	George Orwell	2025-03-11 19:49:58.282000	true

Conferindo Tabela LIVROS

Conclusão

As duas atividades propostas — **integração com banco de dados usando JDBC** e **uso de ORMs como Hibernate** — permitiram explorar duas abordagens distintas para o gerenciamento de dados em aplicações Java. Ambas têm seus méritos e são úteis em diferentes cenários, dependendo das necessidades do projeto. Abaixo, resumo os principais pontos aprendidos e as conclusões gerais:

1. JDBC (Java Database Connectivity)

- **Controle Total:** Com JDBC, temos controle completo sobre as operações do banco de dados. Isso é útil em cenários onde é necessário otimizar consultas SQL complexas ou trabalhar com bancos de dados que exigem operações específicas.
- **Verbosidade:** A implementação com JDBC é mais verbosa, exigindo que o desenvolvedor escreva manualmente o código SQL, gerencie conexões, transações e trate exceções.
- **Aprendizado Fundamental:** Trabalhar com JDBC é essencial para entender como as operações de banco de dados funcionam "por baixo dos panos", o que é útil para depuração e otimização.
- **Desvantagens:** A necessidade de escrever muito código manualmente aumenta a complexidade e a probabilidade de erros, especialmente em projetos grandes.

Conclusão sobre JDBC: JDBC é uma ferramenta poderosa para quem precisa de controle total sobre as operações do banco de dados, mas pode se tornar trabalhosa e propensa a erros em projetos de grande escala ou com muitas operações CRUD.

2. ORMs (Object-Relational Mapping)

- **Produtividade:** ORMs como Hibernate simplificam drasticamente o acesso ao banco de dados, eliminando a necessidade de escrever SQL manualmente. Eles mapeiam automaticamente objetos Java para tabelas do banco de dados, o que acelera o desenvolvimento.

- **Menos Código:** Com ORMs, operações CRUD são realizadas com poucas linhas de código, e o framework cuida de transações, conexões e mapeamento objeto-relacional.
- **Abstração:** ORMs abstraem a complexidade do banco de dados, permitindo que o desenvolvedor se concentre na lógica de negócios em vez de detalhes de implementação.
- **Desvantagens:** Em cenários complexos, o SQL gerado automaticamente pelo ORM pode não ser otimizado, o que pode impactar o desempenho. Além disso, o uso de ORMs pode limitar o controle sobre as operações do banco de dados.

Conclusão sobre ORMs: ORMs são ideais para projetos que exigem alta produtividade e manutenção simplificada. Eles reduzem a quantidade de código e a complexidade, mas podem não ser a melhor escolha para cenários que exigem controle total sobre o SQL ou otimizações específicas.

Comparação Geral

Aspecto	JDBC	ORMs (Hibernate)
Controle	Total controle sobre o SQL e operações do banco de dados.	Menos controle direto sobre o SQL gerado.
Produtividade	Menos produtivo (mais código manual).	Mais produtivo (menos código e mais abstração).
Complexidade	Mais complexo e propenso a erros.	Menos complexo e menos propenso a erros.
Uso Ideal	Projetos pequenos ou que exigem otimizações específicas.	Projetos grandes ou que exigem alta produtividade.

Conclusão Final

Ambas as abordagens — **JDBC** e **ORMs** — têm seu lugar no desenvolvimento de software. A escolha entre elas depende das necessidades do projeto:

- **JDBC** é mais adequado para cenários onde o controle total sobre as operações do banco de dados é essencial, como em consultas complexas ou otimizações específicas.
- **ORMs** são ideais para projetos que exigem alta produtividade e manutenção simplificada, especialmente em aplicações com muitas operações CRUD.

Dominar ambas as abordagens é fundamental para um desenvolvedor Java, pois permite escolher a melhor ferramenta para cada situação. Enquanto o JDBC oferece um entendimento profundo das operações de banco de dados, os ORMs trazem agilidade e redução de complexidade, especialmente em projetos de grande escala.

Portanto, a atividade proporcionou uma visão abrangente das duas técnicas, preparando-nos para tomar decisões informadas no desenvolvimento de aplicações que integram bancos de dados relacionais.

Referências

- **ORACLE.** Java Database Connectivity (JDBC). Disponível em:
<https://docs.oracle.com/javase/tutorial/jdbc/>
(<https://docs.oracle.com/javase/tutorial/jdbc/>).
- **HIBERNATE.** Hibernate ORM: Documentation. Disponível em:
<https://hibernate.org/orm/documentation/>
(<https://hibernate.org/orm/documentation/>).
- **POSTGRESQL.** PostgreSQL JDBC Driver. Disponível em: <https://jdbc.postgresql.org/>
(<https://jdbc.postgresql.org/>).
- **DEV MEDIA.** Hibernate com JPA: Guia Completo. Disponível em:
<https://www.devmedia.com.br/hibernate-com-jpa-guia-completo/37244>
(<https://www.devmedia.com.br/hibernate-com-jpa-guia-completo/37244>).
- **TUTORIALS POINT.** Hibernate Tutorial. Disponível em:
<https://www.tutorialspoint.com/hibernate/index.htm>
(<https://www.tutorialspoint.com/hibernate/index.htm>).
- **ALURA.** Hibernate: Primeiros Passos. Disponível em:
<https://www.alura.com.br/artigos/hibernate-primeiros-passos>
(<https://www.alura.com.br/artigos/hibernate-primeiros-passos>).
- **ROCKETSEAT.** O que é ORM? YouTube, 2023. Disponível em:
<https://www.youtube.com/watch?v=6ZfhG8JpE6k> (<https://www.youtube.com/watch?v=6ZfhG8JpE6k>).
- **FIRESHIP.** ORM Explained in 100 Seconds. YouTube, 2023. Disponível em:
<https://www.youtube.com/watch?v=6ZfhG8JpE6k> (<https://www.youtube.com/watch?v=6ZfhG8JpE6k>).
- **PROGRAMMING WITH MOSH.** What is ORM? YouTube, 2023. Disponível em:
<https://www.youtube.com/watch?v=6ZfhG8JpE6k> (<https://www.youtube.com/watch?v=6ZfhG8JpE6k>).
- **LOIANE GRONER.** Entendendo ORM na Prática. YouTube, 2023. Disponível em:
<https://www.youtube.com/watch?v=6r2qBwZRvq4>

(<https://www.youtube.com/watch?v=6r2qBwZRvq4>).