

データベース最終課題 プレゼン資料

2442005 五十嵐玲有

2442012 岩本光多郎

2442082 三浦篤史

2442083 村瀬優直

2442087 山口漣麗

<https://github.com/graceripple/Burg-burg>

アプリの 実装コード

```

3   講義内容の実装:
4   - 第4回: トランザクション処理・ACID特性
5   - 第3回: 関係演算(選択・射影・結合)
6   - 第11回: 正規化されたデータベース設計
7   """
8
9
10  from flask import Flask, jsonify, request, render_template_string
11  from database import get_db_cursor, execute_transaction, init_connection_pool
12  import os
13  from dotenv import load_dotenv
14
15  load_dotenv()
16
17  app = Flask(__name__)
18
19  # データベース接続プール初期化
20  init_connection_pool()
21
22  # =====
23  # API: 商品一覧取得
24  # 第3回: SELECT演算(選択・射影)
25  # =====
26  # =====

45      # カラム名を取得
46      columns = [desc[0] for desc in cursor.description]
47      # 結果を辞書形式に変換
48      products = [dict(zip(columns, row)) for row in cursor.fetchall()]
49
50      return jsonify({'success': True, 'data': products})
51
52  except Exception as e:
53      print(f'Error fetching products: {e}')
54      return jsonify({'success': False, 'error': 'Failed to fetch products'}), 500
55
56
57  # =====
58  # API: 商品詳細取得
59  # =====
60 @app.route('/api/products/<int:product_id>', methods=['GET'])
61 def get_product(product_id):
62     """特定の商品の詳細を取得"""
63     try:
64         with get_db_cursor() as cursor:
65             cursor.execute("""
66                 SELECT * FROM products WHERE product_id = %s
67             """, (product_id,))
68
69     except Exception as e:
70         print(f'Error fetching product: {e}')
71         return jsonify({'success': False, 'error': 'Failed to fetch product'}), 500
72
73
74  # =====
75  # API: 注文作成(トランザクション処理)
76  # 第4回: ACID特性の実装例
77  # =====
78 @app.route('/api/orders', methods=['POST'])
79 def create_order():
80     """
81     注文を作成する(トランザクション処理)
82
83     講義第4回の送金例と同じ構造:
84     1. 在庫チェック(ロック取得)
85     2. 注文レコード作成
86     3. 注文明細作成
87     4. 在庫更新
88     5. 在庫履歴記録
89     6. 合計金額更新
90
91     エラー時は自動的にロールバック(Atomicity: 原子性)
92
93     1. 在庫チェック(ロック取得)
94     2. 注文レコード作成
95     3. 注文明細作成
96     4. 在庫更新
97     5. 在庫履歴記録
98     6. 合計金額更新
99
100    エラー時は自動的にロールバック(Atomicity: 原子性)
101
102    try:
103
104        # トランザクション実行
105        def order_transaction(cursor):
106            # 1. 在庫チェック(ロック取得)
107            for item in items:
108                cursor.execute("""
109                    SELECT product_id, product_name, stock_quantity, unit_price
110                    FROM products
111                    WHERE product_id = %s
112                    FOR UPDATE -- 行レベルロック取得(Isolation: 隔離性)
113                """, (item['product_id'],))
114
115                product = cursor.fetchone()
116                if not product:
117                    raise Exception(f'商品ID {item["product_id"]} が見つかりません')
118
119                if product[2] < item['quantity']:
120                    raise Exception(
121                        f'商品 [{product[1]}] の在庫が不足しています。'
122                        f' (在庫: {product[2]}個、注文: {item["quantity"]}個)')
123
124
125                # 2. 注文レコード登録
126                cursor.execute("""
127                    INSERT INTO orders (customer_name, total_amount, status)
128                    VALUES (%s, 0, 'pending')
129                    RETURNING order_id
130                """, (customer_name,))
131
132                order_id = cursor.fetchone()[0]
133                total_amount = 0
134
135                # 3. 注文明細作成と在庫更新
136                for idx, item in enumerate(items, 1):
137                    # 商品情報取得
138                    cursor.execute("""
139                        SELECT unit_price, stock_quantity FROM products
140                        WHERE product_id = %s
141                    """, (item['product_id'],))
142
143                    unit_price, stock_before = cursor.fetchone()
144
145                    # 注文明細追加
146                    cursor.execute("""
147                        INSERT INTO order_details (order_id, detail_number, product_id, quantity, unit_price)
148                        VALUES (%s, %s, %s, %s, %s)
149                    """, (order_id, idx, item['product_id'], item['quantity'], unit_price))
150
151                # 在庫更新
152                stock_after = stock_before - item['quantity']
153
154                # 在庫履歴記録
155                cursor.execute("""
156                    INSERT INTO stock_history (product_id, change_type, quantity_change, stock_before, stock_after, order_id)
157                    VALUES (%s, 'sale', %s, %s, %s, %s)
158                """, (item['product_id'], -item['quantity'], stock_before, stock_after, order_id))
159
160                total_amount += unit_price * item['quantity']
161
162            # 4. 注文合計金額更新
163            cursor.execute("""
164                UPDATE orders
165                SET total_amount = %s,
166                status = 'completed'
167                WHERE order_id = %s
168            """, (total_amount, order_id))
169
170            return {'orderId': order_id, 'totalAmount': float(total_amount)}
171
172        # トランザクション実行
173        result = execute_transaction(order_transaction)
174
175        return jsonify({
176            'success': True,
177            'message': '注文が完了しました',
178            'data': result
179        })
180
181    except Exception as e:
182        print(f'Order creation error: {e}')
183        return jsonify({
184            'success': False,
185            'error': str(e)
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
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
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999

```

```
205 # =====
206 # API: 注文一覧取得
207 # =====
208 @app.route('/api/orders', methods=['GET'])
209 def get_orders():
210     """注文履歴一覧を取得"""
211     try:
212         with get_db_cursor() as cursor:
213             cursor.execute("""
214                 SELECT
215                     order_id,
216                     customer_name,
217                     order_date,
218                     total_amount,
219                     status
220                 FROM orders
221                 ORDER BY order_date DESC
222                 LIMIT 50
223             """)
224
225         columns = [desc[0] for desc in cursor.description]
226         orders = [dict(zip(columns, row)) for row in cursor.fetchall()]
227
228         return jsonify({'success': True, 'data': orders})
229
230     except Exception as e:
231         print(f'Error fetching orders: {e}')
232         return jsonify({'success': False, 'error': 'Failed to fetch orders'}), 500
233
```

```
230 # =====
231 # 第3回: JOIN演算(結合)
232 # =====
233 @app.route('/api/orders/<int:order_id>', methods=['GET'])
234 def get_order(order_id):
235     """注文詳細を取得(商品情報を結合)"""
236     try:
237         with get_db_cursor() as cursor:
238             cursor.execute("""
239                 SELECT * FROM orders WHERE order_id = %s
240             """, (order_id,))
241
242             row = cursor.fetchone()
243             if not row:
244                 return jsonify({'success': False, 'error': 'Order not found'}), 404
245
246             columns = [desc[0] for desc in cursor.description]
247             order = dict(zip(columns, row))
248
249             # 注文明細取得(商品情報を結合)
250             cursor.execute("""
251                 SELECT
252                     od.*,
253                     p.product_name
254                 FROM order_details od
255                 INNER JOIN products p ON od.product_id = p.product_id
256                 WHERE od.order_id = %s
257                 ORDER BY od.detail_number
258             """, (order_id,))
259
260             columns = [desc[0] for desc in cursor.description]
261             details = [dict(zip(columns, row)) for row in cursor.fetchall()]
262
263             order['details'] = details
264
265             return jsonify({'success': True, 'data': order})
266
267     except Exception as e:
268         print(f'Error fetching order: {e}')
269         return jsonify({'success': False, 'error': 'Failed to fetch order'}), 500
270
271
272
273
```

```
279 # =====
280 # フロントエンド: バインページ
281 # =====
282 @app.route('/')
283 def index():
284     """バインページを表示"""
285     html = open('/home/user/webapp-python/templates/index.html', 'r', encoding='utf-8').read()
286
287
288     if __name__ == "__main__":
289         # host="0.0.0.0"が重要です! これがないとブラウザから見えません。
290         app.run(host="0.0.0.0", port=3000, debug=True)
291
292
```

実装コード②

アプリのデモ動画

在庫商品一覧

ノートパソコン

¥89,800

商品ID: 1

在庫: 8

 カートに追加

ワイヤレスマウス

¥2,980

商品ID: 2

在庫: 47

 カートに追加

USB-Cケーブル

¥1,280

商品ID: 3

在庫: 100

 カートに追加

モニターアーム

¥8,900

商品ID: 4

在庫: 15

 カートに追加

ペルソナ定義

- 名前：中村裕子（34歳）
- 職業：個人経営の電化製品の店
- 人物像：20～50代、店舗運営の実務担当（レジ、発注、在庫確認、簡単な経理も兼任しがち）、専任のシステム/データ担当はない、“正しい入力”よりも“止まらず回る”ことが最重要（忙しい時間帯に使う）
- 達成したいこと（Goals / Jobs）：いま売れる在庫があるかを即確認したい、注文を受けたら、迷わず最短で登録して確定したい、過去の取引（いつ・誰に・何を・いくらで）をすぐ見返したい
- 困りごと（Pains）：在庫が実態とズレる（販売登録漏れ、二重入力、返品の反映漏れ）、商品名が曖昧で検索しづらい（似た商品が多い）、入力項目が多いと運用が破綻する（忙しい時に入力できない）、合計金額の計算ミスが怖い（割引/税/端数など）
- 利用状況（Context）：ピーク時：1件あたり30～60秒で処理したい、PC/タブレット中心、画面は広くないこともある、“紙や口頭の注文”→“あとでまとめ入力”も起きる

プロジェクトの目的と概要

開発目的: 効率的な販売管理を実現するためのWebアプリケーション構築

主要機能（動画の実装内容）：

- ・在庫商品の一覧表示と管理
- ・新規注文の作成（顧客名入力、商品選択、数量指定、合計金額の自動計算）
- ・注文履歴の保持と詳細情報の確認機能

ビジネス要件定義

ターゲット（ペルソナ）：小規模な店舗の在庫・販売管理を簡略化したい担当者

ストーリーボード：

1. 担当者が現在の在庫一覧を確認する。
2. 新規注文を受け、顧客名と商品を入力する。
3. システムが自動計算した合計金額を確認し、注文を確定させる。
4. 履歴画面で過去の取引内容を振り返る。

システムアーキテクチャ

構成: 標準的な Web 3 Layer 構成を採用

構成要素:

- Web/Client層: ブラウザベースのユーザーインターフェース（動画内の操作画面）
- Application層: Python (Flask等) を利用したビジネスロジック処理
- Database層: RDB を用いた永続的なデータ管理

データベース設計

データモデル: RDB を利用し、ER図に基づいたテーブル設計を実施

主なエンティティ:

- Users (注文者/顧客)
- Products (在庫商品: ノートパソコン、ワイヤレスマウス等)
- Orders (注文履歴: ID、注文日時、合計金額)

CRUD操作の実装: 在庫の読み取り (Read) と新規注文の登録 (Create) を網羅

実装機能の解説（デモ動画に基づく）

在庫商品一覧画面:

- 商品名、価格、在庫数、および個別の画像を表示。

新規注文プロセス:

- ドロップダウンリストから商品を選択（例：HDMIケーブル 2m ¥1,580）。
- 数量の増減に応じた小計・合計のリアルタイム更新。

注文履歴と詳細:

- 「注文を確定」ボタン押下後、履歴一覧へ即時反映。
- ステータス管理（「完了」等）とモーダルによる詳細表示機能。

インフラ構成と非機能要件

環境構築: Webサーバー、アプリケーションサーバー、DBの連携設定

環境変数管理: .env ファイルを用いたセキュアなDB接続情報の保持

今後の課題: RPO/RTO（目標復旧時間）の定義やバックアップ体制、パフォーマンスの最適化

プロジェクト管理と進捗

GitHub Project の活用: Kanban Board を用いたタスク・チケット管理

開発手法: Scrum マスターを中心とした Sprint 管理による段階的な実装

まとめ

本プロジェクトの成果

- ・全工程の完遂: ビジネス要件定義からインフラ構築、アプリケーション実装に至る開発工程を統合的に遂行した。
- ・Web 3層構造の実装: FlaskとRDBを用い、保守性の高い3層構造アーキテクチャを実現した。
- ・実用的な販売管理機能: 動画の通り、在庫確認から注文登録、履歴管理までの主要なビジネスロジックを具現化した。

まとめ

技術的習得事項

- ・アジャイル管理の実践: GitHub ProjectとKanban Boardを活用し、タスクの可視化と工程管理の重要性を習得した。
- ・実務的設計視点:.envによるセキュリティ管理や、RPO/RTOを意識した非機能要件の設計手法を理解した。

まとめ

今後の課題

- ・機能の高度化: 在庫の自動減算処理や売上統計機能の実装による、利便性の向上を図る。
- ・運用の自動化: CI/CDパイプラインの導入により、開発からデプロイまでの効率化を検討する。