

Ex. No.: 9
Date: / / 2024

Meghana Kumar
3122235002070
Nikhil S
3122235002080

PROJECT – MEDICAL SALES DATABASE MANAGEMENT

AIM:

SCHEMA DIAGRAM:

CODE SNIPPET:

```
10 class DatabaseManager:
11     _instance = None
12
13     def __new__(cls, *args, **kwargs):
14         if not cls._instance:
15             cls._instance = super(DatabaseManager, cls).__new__(cls)
16         return cls._instance
17
18     def __init__(self, username, password):
19         if not hasattr(self, 'initialized'): # Avoid re-initialization
20             try:
21                 dsn = "localhost:1521"
22                 self.connection = cx_Oracle.connect(user=username, password=password, dsn=dsn)
23                 self.cursor = self.connection.cursor()
24                 print("Database connection established successfully.")
25
26                 self.check_and_add_constraint(
27                     'SALES_ITEMS',
28                     'check_sales_item_quantity_positive',
29                     'CHECK (quantity > 0)'
30                 )
31                 # Drop existing trigger if it exists
32                 try:
33                     self.execute_query("DROP TRIGGER update_medicine_stock", show_success=False)
34                 except cx_Oracle.DatabaseError:
35                     pass
36                 trigger_code = """
37                 CREATE OR REPLACE TRIGGER update_medicine_stock
38                 AFTER INSERT ON SALES_ITEMS
39                 FOR EACH ROW
40                 BEGIN
41                     -- Ensure quantity does not go below zero
42                     UPDATE MEDICINE
43                     SET quantity = quantity - :NEW.quantity
44                     WHERE medicine_id = :NEW.medicine_id
45                     AND quantity >= :NEW.quantity;
46
47                     -- Raise an error if quantity would go below zero
48                     IF SQL%ROWCOUNT = 0 THEN
49                         RAISE_APPLICATION_ERROR(-20001, 'Insufficient stock in MEDICINE table.');
```

```
50                     END IF;
51                 END;
52                 self.execute_query(trigger_code, show_success=False)
53
54                 self.initialized = True
55             except cx_Oracle.DatabaseError as e:
56                 messagebox.showerror("Database Error", str(e))
57
58     def execute_query(self, query, params=(), show_success=True):
59         try:
60             self.cursor.execute(query, params)
61             self.connection.commit()
62         except cx_Oracle.DatabaseError as e:
63             error_message = str(e)
64             if "ORA-20001" in error_message:
65                 messagebox.showerror("Trigger Error", "Trigger prevented the operation: Insufficient stock in MEDICINE table.")
66             else:
67                 messagebox.showerror("Database Error", error_message)
68
69
```

```

69
70 def fetch_query(self, query, params=()):
71     try:
72         self.cursor.execute(query, params)
73         return self.cursor.fetchall()
74     except cx_Oracle.DatabaseError as e:
75         messagebox.showerror("Database Error", str(e))
76         return []
77
78 def close(self):
79     self.cursor.close()
80     self.connection.close()
81
82 def check_and_add_constraint(self, table_name, constraint_name, constraint_definition):
83     check_query = """
84     SELECT COUNT(*)
85     FROM all_constraints
86     WHERE table_name = :table_name
87     AND constraint_name = :constraint_name
88     """
89     result = self.fetch_query(check_query, {'table_name': table_name.upper(), 'constraint_name': constraint_name.upper()})
90
91     if result and result[0][0] == 0:
92         add_constraint_query = f"ALTER TABLE {table_name} ADD CONSTRAINT {constraint_name} {constraint_definition}"
93         self.execute_query(add_constraint_query, show_success=False)
94         print(f"Constraint {constraint_name} added successfully.")
95     else:
96         print(f"Constraint {constraint_name} already exists.")

```

```

203 class MedicineInsert(InsertTemplate):
204     def __init__(self, medicine_id, m_name, brand, batch_number, expiry_date, quantity, price, supplier_id):
205         self.medicine_id = medicine_id
206         self.m_name = m_name
207         self.brand = brand
208         self.batch_number = batch_number
209         self.expiry_date = expiry_date
210         self.quantity = quantity
211         self.price = price
212         self.supplier_id = supplier_id
213
214     def validate(self):
215         if not (self.medicine_id or self.m_name or self.brand or self.batch_number or self.expiry_date or self.quantity or self.price or self.supplier_id):
216             messagebox.showerror('Error!', 'All fields are required.')
217             return False
218         if int(self.quantity) <= 0:
219             messagebox.showerror('Error!', 'Quantity must be positive.')
220             return False
221         if float(self.price) <= 0:
222             messagebox.showerror('Error!', 'Price must be positive.')
223             return False
224         if not re.match(r"^[M]\d{3}$", self.medicine_id):
225             messagebox.showerror('Error!', 'Invalid Medicine ID format. It should start with "M" followed by three digits.')
226             return False
227         if not re.match(r"^[BATCH]\d{3}$", self.batch_number):
228             messagebox.showerror('Error!', 'Invalid Batch Number format. It should start with "BATCH" followed by three digits.')
229             return False
230         return True
231
232     def perform_insert(self):
233         query = """INSERT INTO MEDICINE (medicine_id, m_name, brand, batch_number, expiry_date, quantity, price, supplier_id)
234         VALUES (:medicine_id, :m_name, :brand, :batch_number, :expiry_date, :quantity, :price, :supplier_id)"""
235         params = {
236             'medicine_id': self.medicine_id,
237             'm_name': self.m_name,
238             'brand': self.brand,
239             'batch_number': self.batch_number,
240             'expiry_date': self.expiry_date,
241             'quantity': self.quantity,
242             'price': self.price,
243             'supplier_id': self.supplier_id
244         }
245         dbms.execute_query(query, params)
246         dbms.execute_query("COMMIT")

```

```

249 class CustomerInsert(InsertTemplate):
250     def __init__(self, customer_id, customer_name, contact_number, email, address):
251         self.customer_id = customer_id
252         self.customer_name = customer_name
253         self.contact_number = contact_number
254         self.email = email
255         self.address = address
256
257     def validate(self):
258         if not self.customer_id or not self.customer_name or not self.contact_number or not self.email or not self.address:
259             messagebox.showerror('Error!', 'All fields are required.')
260             return False
261         if not re.match(r"^\d{3}$", self.customer_id):
262             messagebox.showerror('Error!', 'Invalid Customer ID format. It should start with "C" followed by three digits.')
263             return False
264         return True
265
266     def perform_insert(self):
267         query = """INSERT INTO CUSTOMER (customer_id, c_name, contact_number, email, address)
268             VALUES (:customer_id, :customer_name, :contact_number, :email, :address)"""
269         params = {
270             'customer_id': self.customer_id,
271             'customer_name': self.customer_name,
272             'contact_number': self.contact_number,
273             'email': self.email,
274             'address': self.address
275         }
276         dbms.execute_query(query, params)
277         dbms.execute_query("COMMIT")
278
279 class PrescriptionInsert(InsertTemplate):
280     def __init__(self, prescription_id, customer_id, doctor_name, prescription_date, dosage, frequency, duration, additional_instructions):
281         self.prescription_id = prescription_id
282         self.customer_id = customer_id
283         self.doctor_name = doctor_name
284         self.prescription_date = prescription_date
285         self.dosage = dosage
286         self.frequency = frequency
287         self.duration = duration
288         self.additional_instructions = additional_instructions
289
290     def validate(self):
291         if not (self.prescription_id or self.customer_id or self.doctor_name or self.prescription_date or self.dosage or self.frequency or self.duration or self.additional_instructions):
292             messagebox.showerror('Error!', 'All fields are required.')
293             return False
294
295         # Validate Prescription ID format (should start with "P" followed by three digits)
296         if not re.match(r"^\d{3}$", self.prescription_id):
297             messagebox.showerror('Error!', 'Invalid Prescription ID format. It should start with "P" followed by three digits.')
298             return False
299
300         # Check if the prescription date is in the correct format (DD-MON-YY)
301         try:
302             datetime.strptime(self.prescription_date, "%d-%b-%y")
303         except ValueError:
304             messagebox.showerror('Error!', 'Invalid date format. Please enter the prescription date in DD-MON-YY format.')
305             return False
306
307         # Assuming check_customer_id is defined to validate customer ID
308         if not check_customer_id(self.customer_id):
309             messagebox.showerror('Error!', 'Customer ID does not exist.')
310             return False
311
312         return True
313
314     def perform_insert(self):
315         query = """INSERT INTO PRESCRIPTION (prescription_id, customer_id, doctor_name, prescription_date, dosage, frequency, duration, additional_instructions)
316             VALUES (:prescription_id, :customer_id, :doctor_name, :prescription_date, :dosage, :frequency, :duration, :additional_instructions)"""
317         params = {
318             'prescription_id': self.prescription_id,
319             'customer_id': self.customer_id,
320             'doctor_name': self.doctor_name,
321             'prescription_date': self.prescription_date,
322             'dosage': self.dosage,
323             'frequency': self.frequency,
324             'duration': self.duration,
325             'additional_instructions': self.additional_instructions
326         }
327         dbms.execute_query(query, params)
328         dbms.execute_query("COMMIT")

```

```

331 class SalesInsert(InsertTemplate):
332     def __init__(self, sales_id, customer_id, sales_date, total_amount, payment_method):
333         self.sales_id = sales_id
334         self.customer_id = customer_id
335         self.sales_date = sales_date
336         self.total_amount = total_amount
337         self.payment_method = payment_method
338
339     def validate(self):
340         if not self.sales_id or not self.customer_id or not self.sales_date or not self.total_amount or not self.payment_method:
341             messagebox.showerror('Error!', 'All fields are required.')
342             return False
343         if self.payment_method not in ['Cash', 'Credit Card', 'Debit Card', 'Online', 'UPI']:
344             messagebox.showerror('Error!', 'Invalid payment method. Choose from: Cash, Credit Card, Debit Card, Online, or UPI.')
345             return False
346         return True
347
348     def perform_insert(self):
349         query = """INSERT INTO SALES (sale_id, customer_id, sale_date, total_amount, payment_method)
350             VALUES (:sales_id, :customer_id, :sales_date, :total_amount, :payment_method)"""
351         params = {
352             'sales_id': self.sales_id,
353             'customer_id': self.customer_id,
354             'sales_date': self.sales_date,
355             'total_amount': self.total_amount,
356             'payment_method': self.payment_method
357         }
358         dbms.execute_query(query, params)
359         dbms.execute_query("COMMIT")
360
361 class SalesItemInsert(InsertTemplate):
362     def __init__(self, item_id, sales_id, medicine_id, quantity, price):
363         self.item_id = item_id
364         self.sales_id = sales_id
365         self.medicine_id = medicine_id
366         self.quantity = quantity
367         self.price = price
368         self.subtotal = self.calculate_subtotal()
369
370     def calculate_subtotal(self):
371         return self.quantity * self.price
372
373     def validate(self):
374         # Validate that quantity and price are positive
375         if self.quantity <= 0:
376             messagebox.showerror('Error!', 'Quantity must be greater than 0.')
377             return False
378         if self.price <= 0:
379             messagebox.showerror('Error!', 'Price must be greater than 0.')
380             return False
381         return True
382
383     def perform_insert(self):
384         query = """INSERT INTO SALES_ITEMS (sale_item_id, sale_id, medicine_id, quantity, price_per_unit, subtotal)
385             VALUES (:item_id, :sales_id, :medicine_id, :quantity, :price, :subtotal)"""
386         params = {
387             'item_id': self.item_id,
388             'sales_id': self.sales_id,
389             'medicine_id': self.medicine_id,
390             'quantity': self.quantity,
391             'price': self.price,
392             'subtotal': self.subtotal
393         }
394         dbms.execute_query(query, params)
395         dbms.execute_query("COMMIT")
396
397
398 # Creating an object for Database to Python link
399 dbms = DatabaseManager(username='system', password='tiger')

```

```

816 def check_customer_history():
817     # Create tkinter page for viewing customer's purchase history
818     root_history = Tk()
819     root_history.geometry('1000x700+250+50')
820     root_history.title('Customer Purchase History')
821     root_history.resizable(0, 0)
822     root_history.config(bg='gray')
823     bgimg = Image.open(R"C:\Meghana\SSN\sem 3\Database Lab\Mini Project\bgpic.jpg")
824     bgtk = ImageTk.PhotoImage(bgimg)
825     bglabel = Label(root_history, image=bgtk, height=750, width=1000)
826     bglabel.place(x=0, y=0)
827
828     # Create top frame for displaying title
829     Topframe = Frame(root_history, bg='black', width=1000, height=150)
830     Topframe.place(x=0, y=0)
831
832     # Title text with increased font size
833     Introtext = Label(Topframe, text='Customer Data',
834                       font=('Georgia', 30, 'bold'), bg='black', fg='white')
835     Introtext.place(x=50, y=40, width=900)
836
837     DetailsFrame = Frame(root_history, bg='black', width=1000, height=540)
838     DetailsFrame.place(x=0, y=180)
839
840     # Label and entry for Customer ID with realignment
841     Label(Topframe, text='Customer ID', font=('Georgia', 14), fg='white', bg='black').place(x=250, y=95)
842     customerid_entry = Entry(Topframe, font=('Georgia', 14), width=15)
843     customerid_entry.place(x=375, y=95)
844
845     # Define Treeview table for displaying results
846     columns = ("Customer ID", "Customer Name", "Prescription ID", "Medicine Name", "Quantity Sold", "Sale Date", "Most Recent Prescription")
847     tree = ttk.Treeview(DetailsFrame, columns=columns, show="headings", style="Treeview")
848     for col in columns:
849         tree.heading(col, text=col)
850         tree.column(col, width=1000 // len(columns), anchor="center")
851     tree.place(x=0, y=30, width=980, height=480)
852
853     v_scroll = Scrollbar(DetailsFrame, orient=VERTICAL, command=tree.yview)
854     h_scroll = Scrollbar(DetailsFrame, orient=HORIZONTAL, command=tree.xview)
855     tree.config(yscrollcommand=v_scroll.set, xscrollcommand=h_scroll.set)
856     v_scroll.place(x=980, y=30, height=480)
857     h_scroll.place(x=0, y=490, width=980)
858
859     def resetfield():
860         customerid_entry.delete(0, END)
861
862     def fetch_and_display_customer_history():
863         customer_id = customerid_entry.get()
864
865         # Check if Customer ID is in the correct format (e.g., C001)
866         if not re.match(r'^C\d{3}$', customer_id):
867             messagebox.showerror('Error!', 'Customer ID format is invalid. It should be in the format Cxxx (e.g., C001).')
868             return
869         try:
870             # Check if Customer ID exists in the CUSTOMER table
871             check_query = "SELECT COUNT(*) FROM CUSTOMER WHERE customer_id = :1"
872             print(f"Executing query: {check_query} with {customer_id}")
873             result = dbms.execute_query(check_query, (customer_id,))
874             print(f"Query result: {result}")
875
876             if result and result[0][0] == 0:
877                 messagebox.showerror('Error!', 'Customer ID does not exist in the database.')
878                 customerid_entry.delete(0, END)
879                 return
880         except Exception as e:
881             print(f"Error during customer check: {e}")
882             messagebox.showerror('Error!', f'Error checking Customer ID: {e}')
883             customerid_entry.delete(0, END)
884             return

```

```

885     try:
886         query = """
887             SELECT C.customer_id, C.c_name AS customer_name,
888                   P.prescription_id, M.m_name AS medicine_name,
889                   SI.quantity, S.sale_date,
890                   ( SELECT MAX(P1.prescription_id) -- Subquery to get the most recent prescription
891                     FROM PRESCRIPTION P1
892                     WHERE P1.customer_id = C.customer_id
893                   ) AS most_recent_prescription_id
894             FROM CUSTOMER C
895             JOIN PRESCRIPTION P ON C.customer_id = P.customer_id
896             JOIN SALES S ON C.customer_id = S.customer_id
897             JOIN SALES_ITEMS SI ON SI.sale_id = S.sale_id
898             JOIN MEDICINE M ON M.medicine_id = SI.medicine_id
899             WHERE C.customer_id = :1
900         """
901         print(f"Executing query: {query} with {customer_id}")
902         records = dbms.fetch_query(query, (customer_id,))
903         print(f"Fetch records: {records}")
904
905         # Ensure records are not None before accessing
906         if not records or len(records) == 0:
907             messagebox.showinfo('No Data', 'This customer has no sales or prescription history.')
908             resetfield()
909             return
910         tree.delete(*tree.get_children()) # Clear previous records
911
912         for record in records:
913             # Debug: Print each record to verify its format
914             print(f"Inserting record: {record}")
915
916             if isinstance(record, tuple):
917                 tree.insert("", "end", values=record) # Insert into Treeview
918             else:
919                 messagebox.showerror('Data Error', 'Record format is incorrect.')
920                 resetfield()
921                 return
922         except Exception as e:
923             print(f"Error during fetch: {e}")
924             messagebox.showerror('Query Failed', f'Error fetching customer history: {e}')
925             resetfield()
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

```

```

2158 def updatebysupplier():
2159     def update_supplier_details():
2160         se = sidentry.get() # Supplier ID (primary key for identification)
2161         sne = snameentry.get() # Supplier Name
2162         ne = numberentry.get() # Contact Number
2163         ee = emailentry.get() # Email
2164         ae = addressentry.get() # Address
2165
2166         # Validate email format
2167         def email_check(email):
2168             pattern = r"^[a-zA-Z0-9_+-.]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-]+$"
2169             return bool(re.match(pattern, email))
2170
2171         # Validate Supplier ID format
2172         def pattern():
2173             pattern_regex = r"^S\d{3}$"
2174             if not re.match(pattern_regex, se):
2175                 messagebox.showerror('Error!', 'Invalid Supplier ID format. It should start with "S" followed by three digits.')
2176                 return False
2177             return True
2178
2179         # Validate Supplier Name
2180         def name_check():
2181             pattern_regex = r"^[a-zA-Z ]+$"
2182             if not re.match(pattern_regex, sne):
2183                 messagebox.showerror('Error!', 'Invalid Supplier Name format. It should contain only letters and spaces.')
2184                 return False
2185             return True
2186
2187         # Validate Contact Number
2188         def number_check():
2189             pattern_regex = r"^[0-9]{10}$"
2190             if not re.match(pattern_regex, ne):
2191                 messagebox.showerror('Error!', 'Invalid Contact Number format. It should be a 10-digit number.')
2192                 return False
2193             return True
2194
2195         # Validate Address
2196         def address_check():
2197             pattern_regex = r"^[a-zA-Z0-9 ]+$"
2198             if not re.match(pattern_regex, ae):
2199                 messagebox.showerror('Error!', 'Invalid Address format. It should contain only letters, numbers, and spaces.')
2200                 return False
2201             return True
2202
2203         # Validate all fields
2204         if not (pattern() and name_check() and number_check() and address_check()):
2205             return
2206
2207         # Update supplier details
2208         sql = """
2209             UPDATE SUPPLIER
2210             SET SNAME = ?, SCONTACT = ?, SADDRESS = ?
2211             WHERE SID = ?
2212         """
2213         cursor.execute(sql, (sne, ne, ae, se))
2214         db.commit()
2215         messagebox.showinfo('Success', 'Supplier details updated successfully.')
2216         resetfield()
2217
2218     update_supplier_details()
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000

```

```

2198     if se == '' or sne == '' or ne == '' or ee == '' or ae == '':
2199         messagebox.showerror('Error!', 'Enter all the required values.')
2200         resetfield()
2201         return
2202     elif not pattern(): # Check if Supplier ID format is correct
2203         resetfield()
2204         return
2205     elif len(ne) != 10 or not ne.isdigit() or ne[0] == '0': # Validate Contact Number format
2206         messagebox.showerror('Error!', 'Enter a valid 10-digit mobile number that does not start with 0.')
2207         resetfield()
2208         return
2209     elif not email_check(ee): # Check if email format is correct
2210         messagebox.showerror('Error', 'Enter a valid Gmail ID (e.g., example@gmail.com).')
2211         resetfield()
2212         return
2213     else:
2214         # Check if Supplier ID exists in the table before updating
2215         try:
2216             query_check = "SELECT supplier_id FROM SUPPLIER WHERE supplier_id = :1"
2217             params_check = (se,)
2218             result = dbms.fetch_query(query_check, params_check)
2219
2220             if not result:
2221                 messagebox.showerror('Error!', f'Supplier ID {se} does not exist in the SUPPLIER table.')
2222                 resetfield()
2223                 return
2224
2225             # Update Supplier details if all validations pass
2226             query = """
2227             UPDATE SUPPLIER
2228             SET s_name=:1, contact_number=:2, email=:3, address=:4
2229             WHERE supplier_id=:5
2230             """
2231             params = (sne, ne, ee, ae, se)
2232             dbms.execute_query(query, params)
2233             dbms.execute_query("COMMIT")
2234             messagebox.showinfo('Success!', f'Supplier ID {se} updated successfully.')
2235             resetfield()
2236         except Exception as e:
2237             messagebox.showerror('Update Failed', str(e))
2238             resetfield()
2239
2240 def updatebymedicine():
2241
2242     def update_medicine_details():
2243         me = midentry.get() # Medicine ID (primary key for identification)
2244         mne = mnameentry.get() # Medicine Name
2245         bre = brandentry.get() # Brand
2246         be = batchnoentry.get() # Batch Number
2247         ee = expdateentry.get() # Expiry Date
2248         qe = qtyentry.get() # Quantity
2249         pe = priceentry.get() # Price
2250         se = sidentry.get() # Supplier ID
2251
2252         # Check if Medicine ID is valid and exists
2253         if not validate_medicine_id(me):
2254             messagebox.showerror('Error!', 'Please enter a valid Medicine ID (e.g., M001).')
2255             resetfield()
2256             return

```



```

2354     try:
2355         query = "SELECT COUNT(*) FROM MEDICINE WHERE medicine_id = :1"
2356         params = (me,)
2357         result = dbms.execute_query(query, params)
2358         if result[0][0] == 0:
2359             messagebox.showerror('Error!', f'Medicine ID {me} does not exist in the table.')
2360             resetfield()
2361             return
2362     except Exception as e:
2363         messagebox.showerror('Error!', f'Failed to check Medicine ID: {str(e)}')
2364         resetfield()
2365         return
2366     # Validate Batch Number
2367     if not validate_batch_number(be):
2368         messagebox.showerror('Error!', 'Please enter a valid Batch Number (e.g., BATCH123).')
2369         resetfield()
2370         return
2371     # Validate Expiry Date
2372     if not validate_expiry_date(ee):
2373         messagebox.showerror('Error!', 'Please enter a valid Expiry Date (dd-mon-yy).')
2374         resetfield()
2375         return
2376     # Validate Quantity and Price (Non-negative)
2377     if not (qe.isdigit() and int(qe) >= 0):
2378         messagebox.showerror('Error!', 'Please enter a valid non-negative Quantity.')
2379         resetfield()
2380         return
2381     if not (pe.replace('.', '', 1).isdigit() and float(pe) >= 0):
2382         messagebox.showerror('Error!', 'Please enter a valid non-negative Price.')
2383         resetfield()
2384         return
2385     # Validate Supplier ID
2386     if not validate_supplier_id(se):
2387         messagebox.showerror('Error!', 'Please enter a valid Supplier ID (e.g., S001).')
2388         resetfield()
2389         return
2390     # Check if Supplier ID exists in the supplier table

2391     try:
2392         query = "SELECT COUNT(*) FROM SUPPLIER WHERE supplier_id = :1"
2393         params = (se,)
2394         result = dbms.execute_query(query, params)
2395         if result[0][0] == 0:
2396             messagebox.showerror('Error!', f'Supplier ID {se} does not exist in the Supplier table.')
2397             resetfield()
2398             return
2399     except Exception as e:
2400         messagebox.showerror('Error!', f'Failed to check Supplier ID: {str(e)}')
2401         resetfield()
2402         return
2403     # If all validations pass, update the record
2404     try:
2405         query = """
2406             UPDATE MEDICINE
2407             SET m_name=:1, brand=:2, batch_number=:3, expiry_date=:4, quantity=:5, price=:6, supplier_id=:7
2408             WHERE medicine_id=:8
2409         """
2410         params = (mne, bre, be, ee, qe, pe, se, me)
2411         dbms.execute_query(query, params)
2412         messagebox.showinfo('Success!', f'Medicine ID {me} updated successfully.')
2413         resetfield()
2414     except Exception as e:
2415         messagebox.showerror('Update Failed', str(e))
2416         resetfield()

```

```

3395 def deletebycustomer():
3407     def delete_customer():
3408         cid = cidentry.get() # Get the customer_id from the entry field
3409
3410         # Check if Customer ID is in the correct format (e.g., C002)
3411         if not re.match(r'^C\d{3}$', cid):
3412             messagebox.showerror('Error!', 'Customer ID format is invalid. It should be in the format Cxxx (e.g., C002).')
3413             return
3414
3415         if cid == '':
3416             messagebox.showerror('Error!', 'Please enter a Customer ID to delete.')
3417             return
3418         else:
3419             # Check if Customer ID exists in the CUSTOMER table
3420             try:
3421                 check_query = "SELECT COUNT(*) FROM CUSTOMER WHERE customer_id=:1"
3422                 result = dbms.fetch_query(check_query, (cid,))
3423
3424                 # Check if result is valid and contains data
3425                 if result and result[0][0] > 0:
3426                     customer_count = result[0][0]
3427                 else:
3428                     messagebox.showerror('Error!', 'Customer ID does not exist in the database.')
3429                     cidentry.delete(0, END)
3430                     return
3431             except Exception as e:
3432                 messagebox.showerror('Error!', f'Error checking Customer ID: {e}')
3433                 cidentry.delete(0, END)
3434                 return
3435
3436             # If all checks pass, proceed with deletion
3437             try:
3438                 query = "DELETE FROM CUSTOMER WHERE customer_id = :1"
3439                 dbms.execute_query(query, (cid,))
3440                 messagebox.showinfo('Success!', f'Customer ID {cid} deleted successfully.')
3441                 cidentry.delete(0, END)
3442             except Exception as e:
3443                 messagebox.showerror('Delete Failed', str(e))
3490
3490 def deletebyprescription():
3502     def delete_prescription():
3503         pid = pidentry.get() # Get the prescription_id from the entry field
3504
3505         # Check if Prescription ID is in the correct format (e.g., P001)
3506         if not re.match(r'^P\d{3}$', pid):
3507             messagebox.showerror('Error!', 'Prescription ID format is invalid. It should be in the format Pxxx (e.g., P001).')
3508             return
3509
3510         if pid == '':
3511             messagebox.showerror('Error!', 'Please enter a Prescription ID to delete.')
3512             return
3513         else:
3514             # Check if Prescription ID exists in the PRESCRIPTION table
3515             try:
3516                 check_query = "SELECT COUNT(*) FROM PRESCRIPTION WHERE prescription_id=:1"
3517                 result = dbms.fetch_query(check_query, (pid,))
3518
3519                 # Check if result is valid and contains data
3520                 if result and result[0][0] > 0:
3521                     prescription_count = result[0][0]
3522                 else:
3523                     messagebox.showerror('Error!', 'Prescription ID does not exist in the database.')
3524                     pidentry.delete(0, END)
3525                     return
3526             except Exception as e:
3527                 messagebox.showerror('Error!', f'Error checking Prescription ID: {e}')
3528                 pidentry.delete(0, END)
3529                 return

```

```

3531         # If all checks pass, proceed with deletion
3532         try:
3533             query = "DELETE FROM PRESCRIPTION WHERE prescription_id = :1"
3534             dbms.execute_query(query, (pid,))
3535             messagebox.showinfo('Success!', f'Prescription ID {pid} deleted successfully.')
3536             pidentry.delete(0, END)
3537         except Exception as e:
3538             messagebox.showerror('Delete Failed', str(e))

3585 def deletebysales():
3597     def delete_sale():
3598         sale_id = saleidentry.get() # Get the sale_id from the entry field
3599
3600         # Check if Sale ID is in the correct format (e.g., S003)
3601         if not re.match(r'^S\d{3}$', sale_id):
3602             messagebox.showerror('Error!', 'Sale ID format is invalid. It should be in the format Sxxx (e.g., S003).')
3603             return
3604
3605         if sale_id == '':
3606             messagebox.showerror('Error!', 'Please enter a Sale ID to delete.')
3607         else:
3608             # Check if Sale ID exists in the SALES table
3609             try:
3610                 check_query = "SELECT COUNT(*) FROM SALES WHERE sale_id=:1"
3611                 result = dbms.fetch_query(check_query, (sale_id,)) # Use fetch_query instead of execute_query for retrieval
3612                 if result[0][0] == 0:
3613                     messagebox.showerror('Error!', 'Sale ID does not exist in the database.')
3614                     saleidentry.delete(0, END)
3615                     return
3616             except Exception as e:
3617                 messagebox.showerror('Error!', f'Error checking Sale ID: {e}')
3618                 saleidentry.delete(0, END)
3619                 return
3620
3621             # If all checks pass, proceed with deletion
3622             try:
3623                 delete_query = "DELETE FROM SALES WHERE sale_id = :1"
3624                 dbms.execute_query(delete_query, (sale_id,))
3625                 messagebox.showinfo('Success!', f'Sale ID {sale_id} deleted successfully.')
3626                 saleidentry.delete(0, END)
3627             except Exception as e:
3628                 messagebox.showerror('Delete Failed', str(e))
3629             ~~~~

3676 def deletebysalesitems():
3688     def delete_sale_item():
3689         sale_item_id = saleitementry.get() # Get the sale_item_id from the entry field
3690
3691         # Check if Sale Item ID is in the correct format (e.g., SI002)
3692         if not re.match(r'^SI\d{3}$', sale_item_id):
3693             messagebox.showerror('Error!', 'Sale Item ID format is invalid. It should be in the format SIxxx (e.g., SI002).')
3694             return
3695
3696         if sale_item_id == '':
3697             messagebox.showerror('Error!', 'Please enter a Sale Item ID to delete.')
3698         else:
3699             # Check if Sale Item ID exists in the SALES_ITEMS table
3700             try:
3701                 check_query = "SELECT COUNT(*) FROM SALES_ITEMS WHERE sale_item_id=:1"
3702                 result = dbms.fetch_query(check_query, (sale_item_id,))
3703                 if result[0][0] == 0:
3704                     messagebox.showerror('Error!', 'Sale Item ID does not exist in the database.')
3705                     saleitementry.delete(0, END)
3706                     return
3707             except Exception as e:
3708                 messagebox.showerror('Error!', f'Error checking Sale Item ID: {e}')
3709                 saleitementry.delete(0, END)
3710                 return

```

```

3712         # If all checks pass, proceed with deletion
3713         try:
3714             delete_query = "DELETE FROM SALES_ITEMS WHERE sale_item_id = :1"
3715             dbms.execute_query(delete_query, (sale_item_id,))
3716             messagebox.showinfo('Success!', f'Sale Item ID {sale_item_id} deleted successfully.')
3717             saleitementry.delete(0, END)
3718         except Exception as e:
3719             messagebox.showerror('Delete Failed', str(e))
3720
3721
3722
3723
3724
3725
3726
3727
3728
3729
3730
3731
3732
3733
3734
3735
3736
3737
3738
3739
3740
3741
3742
3743
3744
3745
3746
3747
3748
3749
3750
3751
3752
3753
3754
3755
3756
3757
3758
3759
3760
3761
3762
3763
3764
3765
3766
3767
3768
3769
3770
3771
3772
3773
3774
3775
3776
3777
3778
3779
3780
3781
3782
3783
3784
3785
3786
3787
3788
3789
3790
3791
3792
3793
3794
3795
3796
3797
3798
3799
3800
3801
3802
3803
3804
3805
3806
3807
3808
3809
3810
3811
3812
3813
3814
3815
3816
3817
3818
3819
3820
3821
3822
3823
3824
3825
3826
3827
3828
3829
3830
3831
3832
3833
3834
3835
3836
3837
3838
3839
3840
3841
3842
3843
3844
3845
3846
3847
3848
3849
3850
3851
3852
3853
3854
3855
3856
3857
3858
3859
3860
3861
3862
3863
3864
3865
3866
3867
3868
3869
3870
3871
3872
3873
3874
3875
3876
3877
3878
3879
3880
3881
3882
3883
3884
3885
3886
3887
3888
3889
3890
3891
3892
3893
3894
3895
3896
3897
3898
3899
3900
3901
3902
3903
3904
3905
3906
3907
3908
3909
3910
3911
3912
3913
3914
3915
3916
3917
3918
3919
3920
3921
3922
3923
3924
3925
3926
3927
3928
3929
3930
3931
3932
3933
3934
3935
3936
3937
3938
3939
3940
3941
3942
3943
3944
3945
3946
3947
3948
3949
3950
3951
3952
3953
3954
3955
3956
3957
3958
3959
3960
3961
3962
3963
3964
3965
3966
3967
3968
3969
3970
3971
3972
3973
3974
3975
3976
3977
3978
3979
3980
3981
3982
3983
3984
3985
3986
3987
3988
3989
3990
3991
3992
3993
3994
3995
3996
3997
3998
3999
4000
4001
4002
4003
4004
4005
4006
4007
4008
4009
4010
4011
4012
4013
4014
4015
4016
4017
4018
4019
4020
4021
4022
4023
4024
4025
4026
4027
4028
4029
4030
4031
4032
4033
4034
4035
4036
4037
4038
4039
4040
4041
4042
4043
4044
4045
4046
4047
4048
4049
4050
4051
4052
4053
4054
4055
4056
4057
4058
4059
4060
4061
4062
4063
4064
4065
4066
4067
4068
4069
4070
4071
4072
4073
4074
4075
4076
4077
4078
4079
4080
4081
4082
4083
4084
4085
4086
4087
4088
4089
4090
4091
4092
4093
4094
4095
4096
4097
4098
4099
4100
4101
4102
4103
4104
4105
4106
4107
4108
4109
4110
4111
4112
4113
4114
4115
4116
4117
4118
4119
4120
4121
4122
4123
4124
4125
4126
4127
4128
4129
4130
4131
4132
4133
4134
4135
4136
4137
4138
4139
4140
4141
4142
4143
4144
4145
4146
4147
4148
4149
4150
4151
4152
4153
4154
4155
4156
4157
4158
4159
4160
4161
4162
4163
4164
4165
4166
4167
4168
4169
4170
4171
4172
4173
4174
4175
4176
4177
4178
4179
4180
4181
4182
4183
4184
4185
4186
4187
4188
4189
4190
4191
4192
4193
4194
4195
4196
4197
4198
4199
4200
4201
4202
4203
4204
4205
4206
4207
4208
4209
4210
4211
4212
4213
4214
4215
4216
4217
4218
4219
4220
4221
4222
4223
4224
4225
4226
4227
4228
4229
4230
4231
4232
4233
4234
4235
4236
4237
4238
4239
4240
4241
4242
4243
4244
4245
4246
4247
4248
4249
4250
4251
4252
4253
4254
4255
4256
4257
4258
4259
4260
4261
4262
4263
4264
4265
4266
4267
4268
4269
4270
4271
4272
4273
4274
4275
4276
4277
4278
4279
4280
4281
4282
4283
4284
4285
4286
4287
4288
4289
4290
4291
4292
4293
4294
4295
4296
4297
4298
4299
4300
4301
4302
4303
4304
4305
4306
4307
4308
4309
4310
4311
4312
4313
4314
4315
4316
4317
4318
4319
4320
4321
4322
4323
4324
4325
4326
4327
4328
4329
4330
4331
4332
4333
4334
4335
4336
4337
4338
4339
4340
4341
4342
4343
4344
4345
4346
4347
4348
4349
4350
4351
4352
4353
4354
4355
4356
4357
4358
4359
4360
4361
4362
4363
4364
4365
4366
4367
4368
4369
4370
4371
4372
4373
4374
4375
4376
4377
4378
4379
4380
4381
4382
4383
4384
4385
4386
4387
4388
4389
4390
4391
4392
4393
4394
4395
4396
4397
4398
4399
4400
4401
4402
4403
4404
4405
4406
4407
4408
4409
4410
4411
4412
4413
4414
4415
4416
4417
4418
4419
4420
4421
4422
4423
4424
4425
4426
4427
4428
4429
4430
4431
4432
4433
4434
4435
4436
4437
4438
4439
4440
4441
4442
4443
4444
4445
4446
4447
4448
4449
4450
4451
4452
4453
4454
4455
4456
4457
4458
4459
4460
4461
4462
4463
4464
4465
4466
4467
4468
4469
4470
4471
4472
4473
4474
4475
4476
4477
4478
4479
4480
4481
4482
4483
4484
4485
4486
4487
4488
4489
4490
4491
4492
4493
4494
4495
4496
4497
4498
4499
4500
4501
4502
4503
4504
4505
4506
4507
4508
4509
4510
4511
4512
4513
4514
4515
4516
4517
4518
4519
4520
4521
4522
4523
4524
4525
4526
4527
4528
4529
4530
4531
4532
4533
4534
4535
4536
4537
4538
4539
4540
4541
4542
4543
4544
4545
4546
4547
4548
4549
4550
4551
4552
4553
4554
4555
4556
4557
4558
4559
4560
4561
4562
4563
4564
4565
4566
4567
4568
4569
4570
4571
4572
4573
4574
4575
4576
4577
4578
4579
4580
4581
4582
4583
4584
4585
4586
4587
4588
4589
4590
4591
4592
4593
4594
4595
4596
4597
4598
4599
4600
4601
4602
4603
4604
4605
4606
4607
4608
4609
4610
4611
4612
4613
4614
4615
4616
4617
4618
4619
4620
4621
4622
4623
4624
4625
4626
4627
4628
4629
4630
4631
4632
4633
4634
4635
4636
4637
4638
4639
4640
4641
4642
4643
4644
4645
4646
4647
4648
4649
4650
4651
4652
4653
4654
4655
4656
4657
4658
4659
4660
4661
4662
4663
4664
4665
4666
4667
4668
4669
4670
4671
4672
4673
4674
4675
4676
4677
4678
4679
4680
4681
4682
4683
4684
4685
4686
4687
4688
4689
4690
4691
4692
4693
4694
4695
4696
4697
4698
4699
4700
4701
4702
4703
4704
4705
4706
4707
4708
4709
4710
4711
4712
4713
4714
4715
4716
4717
4718
4719
4720
4721
4722
4723
4724
4725
4726
4727
4728
4729
4730
4731
4732
4733
4734
4735
4736
4737
4738
4739
4740
4741
4742
4743
4744
4745
4746
4747
4748
4749
4750
4751
4752
4753
4754
4755
4756
4757
4758
4759
4760
4761
4762
4763
4764
4765
4766
4767
4768
4769
4770
4771
4772
4773
4774
4775
4776
4777
4778
4779
4780
4781
4782
4783
4784
4785
4786
4787
4788
4789
4790
4791
4792
4793
4794
4795
4796
4797
4798
4799
4800
4801
4802
4803
4804
4805
4806
4807
4808
4809
4810
4811
4812
4813
4814
4815
4816
4817
4818
4819
4820
4821
4822
4823
4824
4825
4826
4827
4828
4829
4830
4831
4832
4833
4834
4835
4836
4837
4838
4839
4840
4841
4842
4843
4844
4845
4846
4847
4848
4849
4850
4851
4852
4853
4854
4855
4856
4857
4858
4859
4860
4861
4862
4863
4864
4865
4866
4867
4868
4869
4870
4871
4872
4873
4874
4875
4876
4877
4878
4879
4880
4881
4882
4883
4884
4885
4886
4887
4888
4889
4890
4891
4892
4893
4894
4895
4896
4897
4898
4899
4900
4901
4902
4903
4904
4905
4906
4907
4908
4909
4910
4911
4912
4913
4914
4915
4916
4917
4918
4919
4920
4921
4922
4923
4924
4925
4926
4927
4928
4929
4930
4931
4932
4933
4934
4935
4936
4937
4938
4939
4940
4941
4942
4943
4944
4945
4946
4947
4948
4949
4950
4951
4952
4953
4954
4955
4956
4957
4958
4959
4960
4961
4962
4963
4964
4965
4966
4967
4968
4969
4970
4971
4972
4973
4974
4975
4976
4977
4978
4979
4980
4981
4982
4983
4984
4985
4986
4987
4988
4989
4990
4991
4992
4993
4994
4995
4996
4997
4998
4999
5000

```

```

3989 def check_password_strength(password):
3990     # Regular expression to check password strength
3991     if (len(password) >= 8 and
3992         re.search(r'[A-Z]', password) and # At least one uppercase letter
3993         re.search(r'[a-z]', password) and # At least one lowercase letter
3994         re.search(r'[0-9]', password) and # At least one digit
3995         re.search(r'!@#$%^&*()_+', password)): # At least one special character
3996         return True
3997     return False
3998
3999 def create_account():
4000     # Handles the creation of a new account
4001     c_username = c_username_entry.get()
4002     c_password = c_password_entry.get()
4003
4004     # Check if username already exists
4005     if dbms.fetch_query("SELECT * FROM LOGIN WHERE username = :1", (c_username,)):
4006         messagebox.showerror("Account Creation Failed", "Username already exists.")
4007         return
4008
4009     # Check password strength
4010     if not check_password_strength(c_password):
4011         messagebox.showerror("Weak Password", "Password must be at least 8 characters long,\
4012         contain uppercase and lowercase letters, a digit, and a special character.")
4013         return
4014
4015     # Insert new credentials into the LOGIN table
4016     dbms.execute_query("INSERT INTO LOGIN (username, password) VALUES (:1, :2)", (c_username, c_password))
4017
4018     messagebox.showinfo("Account Created", "Your account has been successfully created. Please log in.")
4019     create_login_window.destroy()
4020     show_login_window()
4021
4022 def validate_login(username, password):
4023     # Validates the username and password with the LOGIN table
4024     result = dbms.fetch_query("SELECT * FROM LOGIN WHERE username = :1 AND password = :2", (username, password))
4025     return bool(result)

```

DATABASE – BACKEND :

```
REM Medical Management System Database

SET ECHO ON;

DROP TABLE IF EXISTS SALES_ITEMS;
DROP TABLE IF EXISTS SALES;
DROP TABLE IF EXISTS PRESCRIPTION;
DROP TABLE IF EXISTS MEDICINE;
DROP TABLE IF EXISTS SUPPLIER;
DROP TABLE IF EXISTS CUSTOMER cascade constraints;

REM Creating table SUPPLIER

CREATE TABLE SUPPLIER(
supplier_id VARCHAR2(10) PRIMARY KEY,
s_name VARCHAR2(50) NOT NULL,
contact_number INT,
email VARCHAR2(100),
address VARCHAR2(150)
);

REM Creating table MEDICINE

CREATE TABLE MEDICINE(
medicine_id VARCHAR2(10) PRIMARY KEY,
m_name VARCHAR2(20) NOT NULL,
brand VARCHAR2(20),
batch_number VARCHAR2(10),
expiry_date DATE,
quantity INT DEFAULT 0,
price DECIMAL(10,2) NOT NULL,
supplier_id VARCHAR2(10),
FOREIGN KEY (supplier_id) REFERENCES SUPPLIER(supplier_id) ON DELETE CASCADE
);

REM Creating table CUSTOMER

CREATE TABLE CUSTOMER(
customer_id VARCHAR2(10) PRIMARY KEY,
c_name VARCHAR2(30) NOT NULL,
contact_number INT,
email VARCHAR2(40),
address VARCHAR2(50)
);

REM Creating table PRESCRIPTION

CREATE TABLE PRESCRIPTION(
prescription_id VARCHAR2(15) PRIMARY KEY,
customer_id VARCHAR2(10),
doctor_name VARCHAR2(25),
prescription_date DATE,
dosage VARCHAR2(30),
frequency VARCHAR2(40),
duration VARCHAR2(10),
additional_instructions VARCHAR2(75),
FOREIGN KEY (customer_id) REFERENCES CUSTOMER(customer_id) ON DELETE CASCADE
);

REM Creating table SALES

CREATE TABLE SALES(
sale_id VARCHAR2(10) PRIMARY KEY,
customer_id VARCHAR2(10),
sale_date DATE NOT NULL,
total_amount DECIMAL(10,2) NOT NULL,
payment_method VARCHAR2(20),
FOREIGN KEY (customer_id) REFERENCES CUSTOMER(customer_id) ON DELETE CASCADE
);
```


REM Creating table SALES_ITEMS

```
CREATE TABLE SALES_ITEMS (  
  sale_item_id VARCHAR2(10) PRIMARY KEY,  
  sale_id VARCHAR2(10),  
  medicine_id VARCHAR2(10),  
  quantity INT DEFAULT 1,  
  price_per_unit DECIMAL(10, 2),  
  subtotal DECIMAL(10,2),  
  FOREIGN KEY (sale_id) REFERENCES SALES(sale_id) ON DELETE CASCADE,  
  FOREIGN KEY (medicine_id) REFERENCES MEDICINE(medicine_id) ON DELETE CASCADE  
);
```

REM Creating login table

```
CREATE TABLE IF NOT EXISTS LOGIN (  
  username VARCHAR2(50) PRIMARY KEY,  
  password VARCHAR2(50) NOT NULL);
```

REM Inserting into SUPPLIER Table

```
INSERT INTO SUPPLIER (supplier_id, s_name, contact_number, email, address)  
VALUES
```

```
('S005', 'Ramesh Kumar', 9876543210, 'ramesh.kumar@example.com', '123, Kottivakkam, Chennai, Tamil Nadu'),  
( 'S012', 'Kavitha Nair', 9654321098, 'kavitha.nair@example.com', '456, R.S. Puram, Coimbatore, Tamil Nadu'),  
( 'S021', 'Arunachalam V', 9843214567, 'arunachalam.v@example.com', '789, Simmakal, Madurai, Tamil Nadu'),  
( 'S009', 'Lakshmi S', 9887654321, 'lakshmi.s@example.com', '321, Kilpauk, Chennai, Tamil Nadu'),  
( 'S014', 'Mohan Raj', 9776543210, 'mohan.raj@example.com', '654, Tirunelveli Road, Tirunelveli, Tamil Nadu'),  
( 'S001', 'Anjali Devi', 9567890123, 'anjali.devi@example.com', '111, Perundurai, Erode, Tamil Nadu'),  
( 'S017', 'Suresh Babu', 9443216789, 'suresh.babu@example.com', '222, Thiru Vi Ka Nagar, Tiruchirappalli, Tamil Nadu'),  
( 'S018', 'Vani Duraisamy', 9334567890, 'vani.duraisamy@example.com', '333, Marappalam, Salem, Tamil Nadu'),  
( 'S004', 'Ganesh Prasad', 9198765432, 'ganesh.prasad@example.com', '444, Alangulam, Dindigul, Tamil Nadu'),  
( 'S011', 'Aarti Sundar', 9654321987, 'aarti.sundar@example.com', '555, Ekkatuthangal, Vellore, Tamil Nadu'),  
( 'S022', 'Karthik Subramanian', 9478561230, 'karthik.subramanian@example.com', '666, Thanjavur Road, Thanjavur, Tamil Nadu'),  
( 'S023', 'Vidhya Ramesh', 9345678901, 'vidhya.ramesh@example.com', '777, Karur Bypass Road, Karur, Tamil Nadu'),  
( 'S003', 'Rajesh Kannan', 9845621789, 'rajesh.kannan@example.com', '888, Kanyakumari Road, Kanyakumari, Tamil Nadu'),  
( 'S015', 'Sankar Raghavan', 9654321098, 'sankar.raghavan@example.com', '999, Palayamkottai, Thoothukudi, Tamil Nadu'),  
( 'S019', 'Naveen Kumar', 9567890234, 'naveen.kumar@example.com', '101, Dindigul Main Road, Namakkal, Tamil Nadu'),  
( 'S006', 'Srinivasan Mani', 9445698743, 'srinivasan.mani@example.com', '202, Thiruvalluvar Street, Pudukkottai, Tamil Nadu'),  
( 'S020', 'Priya Natarajan', 9334567890, 'priya.natarajan@example.com', '303, Veeravanallur, Tiruppur, Tamil Nadu'),  
( 'S008', 'Balaji Reddy', 9198765432, 'balaji.reddy@example.com', '404, Pollachi Road, Pollachi, Tamil Nadu'),  
( 'S002', 'Deepika Saravanan', 9887654321, 'deepika.saravanan@example.com', '505, Aruppukottai, Tamil Nadu'),  
( 'S016', 'Vikram Selvam', 9478561234, 'vikram.selvam@example.com', '606, Kallakurichi, Virudhunagar, Tamil Nadu'),  
( 'S010', 'Aditi Raghavan', 9345678901, 'aditi.raghavan@example.com', '707, Chengalpattu Road, Chengalpattu, Tamil Nadu'),  
( 'S013', 'Gayathri Menon', 9567890123, 'gayathri.menon@example.com', '808, Thiruvallur, Perambalur, Tamil Nadu'),  
( 'S024', 'Harish Karthik', 9443216789, 'harish.karthik@example.com', '909, Sathy Road, Karamadai, Tamil Nadu'),  
( 'S007', 'Manoj Krishnan', 9345126780, 'manoj.krishnan@example.com', '888, Gandhi Road, Thanjavur, Tamil Nadu'),  
( 'S025', 'Vasanth Kumar', 9743214567, 'vasanth.kumar@example.com', '010, Thiruvalankadu, Cuddalore, Tamil Nadu');
```

REM Inserting into MEDICINE Table

```
INSERT INTO MEDICINE (medicine_id, m_name, brand, batch_number, expiry_date, quantity, price, supplier_id)  
VALUES
```

```
('M005', 'Paracetamol', 'Acetaminophen', 'BATCH123', TO_DATE('2026-05-01', 'YYYY-MM-DD'), 10, 50.00, 'S023'),  
( 'M018', 'Amoxicillin', 'Amoxil', 'BATCH543', TO_DATE('2025-09-15', 'YYYY-MM-DD'), 15, 120.00, 'S010'),  
( 'M011', 'Ibuprofen', 'Advil', 'BATCH213', TO_DATE('2026-02-20', 'YYYY-MM-DD'), 200, 75.50, 'S018'),  
( 'M002', 'Lisinopril', 'Prinivil', 'BATCH321', TO_DATE('2025-12-10', 'YYYY-MM-DD'), 90, 80.00, 'S002'),  
( 'M024', 'Cetirizine', 'Zyrtec', 'BATCH999', TO_DATE('2026-04-05', 'YYYY-MM-DD'), 120, 45.00, 'S017'),  
( 'M014', 'Metformin', 'Glucophage', 'BATCH888', TO_DATE('2025-08-30', 'YYYY-MM-DD'), 75, 60.00, 'S022'),  
( 'M007', 'Amlodipine', 'Norvasc', 'BATCH456', TO_DATE('2026-01-25', 'YYYY-MM-DD'), 60, 100.00, 'S005'),  
( 'M022', 'Omeprazole', 'Prilosec', 'BATCH111', TO_DATE('2025-07-18', 'YYYY-MM-DD'), 80, 55.00, 'S004'),  
( 'M016', 'Simvastatin', 'Zocor', 'BATCH222', TO_DATE('2024-11-30', 'YYYY-MM-DD'), 50, 90.00, 'S015'),  
( 'M001', 'Clonidine', 'Plavix', 'BATCH333', TO_DATE('2025-10-20', 'YYYY-MM-DD'), 40, 85.00, 'S019'),  
( 'M025', 'Levothyroxine', 'Synthroid', 'BATCH777', TO_DATE('2026-08-14', 'YYYY-MM-DD'), 110, 70.00, 'S021'),  
( 'M013', 'Doxycycline', 'Vibramycin', 'BATCH444', TO_DATE('2025-03-11', 'YYYY-MM-DD'), 30, 40.00, 'S006'),  
( 'M003', 'Hydrochlorothiazide', 'Hydrodiuril', 'BATCH555', TO_DATE('2026-03-12', 'YYYY-MM-DD'), 95, 65.00, 'S014'),  
( 'M019', 'Furosemide', 'Lasix', 'BATCH888', TO_DATE('2026-12-25', 'YYYY-MM-DD'), 70, 50.00, 'S012'),  
( 'M008', 'Montelukast', 'Singulair', 'BATCH666', TO_DATE('2025-05-09', 'YYYY-MM-DD'), 85, 60.00, 'S009'),  
( 'M012', 'Pantoprazole', 'Protonix', 'BATCH999', TO_DATE('2025-11-01', 'YYYY-MM-DD'), 150, 95.00, 'S008'),  
( 'M004', 'Cetirizine', 'Zyrtec', 'BATCH444', TO_DATE('2026-10-30', 'YYYY-MM-DD'), 65, 45.00, 'S020'),  
( 'M015', 'Venlafaxine', 'Effexor', 'BATCH777', TO_DATE('2025-06-17', 'YYYY-MM-DD'), 120, 150.00, 'S001'),  
( 'M006', 'Escitalopram', 'Lexapro', 'BATCH555', TO_DATE('2026-09-09', 'YYYY-MM-DD'), 75, 80.00, 'S003'),  
( 'M020', 'Sertraline', 'Zoloft', 'BATCH222', TO_DATE('2025-02-02', 'YYYY-MM-DD'), 50, 70.00, 'S011'),  
( 'M009', 'Bupropion', 'Wellbutrin', 'BATCH888', TO_DATE('2024-12-05', 'YYYY-MM-DD'), 80, 65.00, 'S016'),  
( 'M010', 'Azithromycin', 'Zithromax', 'BATCH112', TO_DATE('2025-08-15', 'YYYY-MM-DD'), 100, 90.00, 'S005'),  
( 'M017', 'Atorvastatin', 'Lipitor', 'BATCH334', TO_DATE('2026-04-22', 'YYYY-MM-DD'), 85, 75.00, 'S010'),  
( 'M021', 'Gabapentin', 'Neurontin', 'BATCH556', TO_DATE('2025-10-05', 'YYYY-MM-DD'), 70, 120.00, 'S012'),  
( 'M023', 'Losartan', 'Cozaar', 'BATCH778', TO_DATE('2026-02-18', 'YYYY-MM-DD'), 60, 65.00, 'S018');
```

REM Inserting into CUSTOMER TABLE

```
INSERT INTO CUSTOMER (customer_id, c_name, contact_number, email, address)
VALUES
```

```
('C001', 'Rajesh Sharma', 9876543210, 'rajesh.sharma@example.com', '123, Anna Nagar, Chennai, Tamil Nadu'),
('C002', 'Sita Rao', 9988776655, 'sita.rao@example.com', '456, Kotturpuram, Chennai, Tamil Nadu'),
('C003', 'Vikram Singh', 9701234567, 'vikram.singh@example.com', '789, T. Nagar, Chennai, Tamil Nadu'),
('C004', 'Anjali Gupta', 9598765432, 'anjali.gupta@example.com', '321, Mylapore, Chennai, Tamil Nadu'),
('C005', 'Karan Mehta', 9843216789, 'karan.mehta@example.com', '654, Adyar, Chennai, Tamil Nadu'),
('C006', 'Pooja Verma', 9871234560, 'pooja.verma@example.com', '987, Saidapet, Chennai, Tamil Nadu'),
('C007', 'Rahul Jain', 9912345678, 'rahul.jain@example.com', '654, Kodambakkam, Chennai, Tamil Nadu'),
('C008', 'Suresh Babu', 9865432109, 'suresh.babu@example.com', '321, T. Nagar, Chennai, Tamil Nadu'),
('C009', 'Meena Reddy', 9809876543, 'meena.reddy@example.com', '432, Alwarpet, Chennai, Tamil Nadu'),
('C010', 'Nitin Sharma', 9798765432, 'nitin.sharma@example.com', '543, Tambaram, Chennai, Tamil Nadu'),
('C011', 'Aditi Raghavan', 9638527410, 'aditi.raghavan@example.com', '876, Mambalam, Chennai, Tamil Nadu'),
('C012', 'Sunita Menon', 9743214567, 'sunita.menon@example.com', '234, Velachery, Chennai, Tamil Nadu'),
('C013', 'Rajkumar Iyer', 9865321470, 'rajkumar.iyer@example.com', '345, Kottivakkam, Chennai, Tamil Nadu'),
('C014', 'Lakshmi Nair', 9765432109, 'lakshmi.nair@example.com', '456, Nungambakkam, Chennai, Tamil Nadu'),
('C015', 'Vivek Krishnan', 9812345670, 'vivek.krishnan@example.com', '567, Ashok Nagar, Chennai, Tamil Nadu'),
('C016', 'Priya Das', 9709876543, 'priya.das@example.com', '678, Sholinganallur, Chennai, Tamil Nadu'),
('C017', 'Niharika Ramesh', 9898765432, 'niharika.ramesh@example.com', '789, Ekkatuthangal, Chennai, Tamil Nadu'),
('C018', 'Ganesh Kumar', 9945678901, 'ganesh.kumar@example.com', '890, Kotturpuram, Chennai, Tamil Nadu'),
('C019', 'Srinivas Balakrishnan', 9901234567, 'srinivas.balakrishnan@example.com', '321, Thiruvannamiyur, Chennai, Tamil Nadu'),
('C020', 'Rita Choudhury', 9856789012, 'rita.choudhury@example.com', '432, Kottivakkam, Chennai, Tamil Nadu'),
('C021', 'Arvind Kumar', 9812347650, 'arvind.kumar@example.com', '543, Anna Nagar, Chennai, Tamil Nadu'),
('C022', 'Deepak Reddy', 9701253486, 'deepak.reddy@example.com', '654, T. Nagar, Chennai, Tamil Nadu'),
('C023', 'Samantha Joshi', 9897654321, 'samantha.joshi@example.com', '765, Besant Nagar, Chennai, Tamil Nadu'),
('C024', 'Akash Bhatia', 9912345678, 'akash.bhatia@example.com', '876, Kottivakkam, Chennai, Tamil Nadu'),
('C025', 'Gita Pillai', 9909876543, 'gita.pillai@example.com', '987, Mylapore, Chennai, Tamil Nadu');
```

REM Inserting into SALES Table

```
INSERT INTO SALES (sale_id, customer_id, sale_date, total_amount, payment_method)
VALUES
```

```
('S001', 'C007', TO_DATE('2024-10-10', 'YYYY-MM-DD'), 201.00, 'Credit Card'),
('S002', 'C004', TO_DATE('2024-10-11', 'YYYY-MM-DD'), 255.00, 'UPI'),
('S003', 'C010', TO_DATE('2024-10-12', 'YYYY-MM-DD'), 220.00, 'Cash'),
('S004', 'C001', TO_DATE('2024-10-13', 'YYYY-MM-DD'), 165.50, 'UPI'),
('S005', 'C003', TO_DATE('2024-10-14', 'YYYY-MM-DD'), 175.00, 'Debit Card'),
('S006', 'C012', TO_DATE('2024-10-15', 'YYYY-MM-DD'), 140.00, 'Cash'),
('S007', 'C015', TO_DATE('2024-10-16', 'YYYY-MM-DD'), 50.00, 'Cash'),
('S008', 'C022', TO_DATE('2024-10-17', 'YYYY-MM-DD'), 100.00, 'Cash'),
('S009', 'C005', TO_DATE('2024-10-18', 'YYYY-MM-DD'), 170.00, 'UPI'),
('S010', 'C020', TO_DATE('2024-10-19', 'YYYY-MM-DD'), 95.00, 'Debit Card');
```

REM Inserting into SALES_ITEMS Table

```
INSERT INTO SALES_ITEMS (sale_item_id, sale_id, medicine_id, quantity, price_per_unit, subtotal)
VALUES
```

```
('SI001', 'S001', 'M005', 1, 50.00, 50.00),
('SI002', 'S001', 'M011', 2, 75.50, 151.00),
('SI003', 'S002', 'M018', 1, 120.00, 120.00),
('SI004', 'S002', 'M024', 3, 45.00, 135.00),
('SI005', 'S003', 'M005', 2, 50.00, 100.00),
('SI006', 'S003', 'M018', 1, 120.00, 120.00),
('SI007', 'S004', 'M024', 2, 45.00, 90.00),
('SI008', 'S004', 'M011', 1, 75.50, 75.50),
('SI009', 'S005', 'M013', 3, 40.00, 120.00),
('SI010', 'S005', 'M022', 1, 55.00, 55.00),
('SI011', 'S006', 'M020', 2, 70.00, 140.00),
('SI012', 'S007', 'M019', 1, 50.00, 50.00),
('SI013', 'S008', 'M007', 1, 100.00, 100.00),
('SI014', 'S009', 'M001', 2, 85.00, 170.00),
('SI015', 'S010', 'M012', 1, 95.00, 95.00);
```



```

COMMIT;

SELECT * FROM SUPPLIER;
SELECT * FROM CUSTOMER;
SELECT * FROM MEDICINE;
SELECT * FROM PRESCRIPTION;
SELECT * FROM SALES;
SELECT * FROM SALES_ITEMS;

CREATE OR REPLACE PROCEDURE check_near_expiry(expiry_meds OUT SYS_REFCURSOR)
AS
BEGIN
    OPEN expiry_meds FOR
    SELECT M_NAME, EXPIRY_DATE, ROUND(EXPIRY_DATE - SYSDATE) AS days_remaining
    FROM MEDICINE
    WHERE EXPIRY_DATE - SYSDATE <= 30;
END;
/

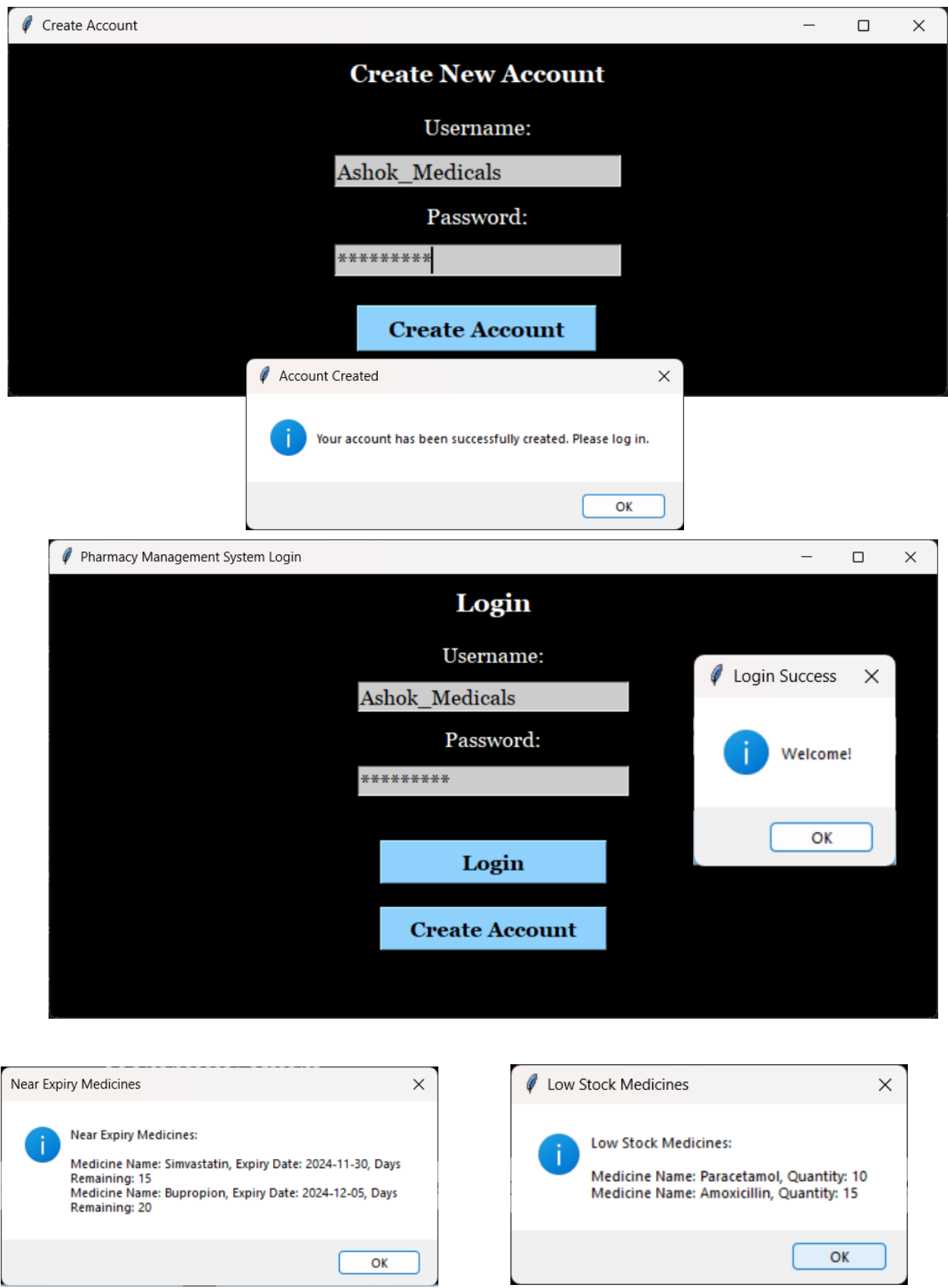
CREATE OR REPLACE PROCEDURE check_low_stock(low_stock_meds OUT SYS_REFCURSOR)
AS
BEGIN
    OPEN low_stock_meds FOR
    SELECT M_NAME, QUANTITY
    FROM MEDICINE
    WHERE QUANTITY < 25;
END;
/

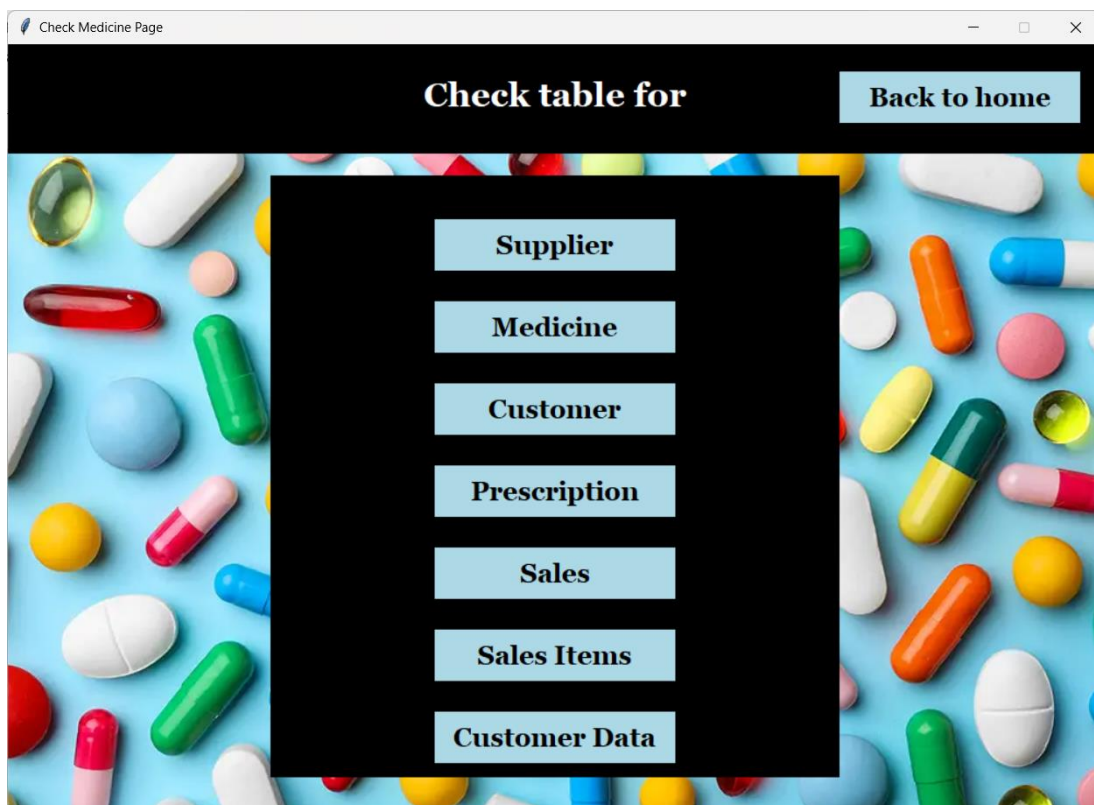
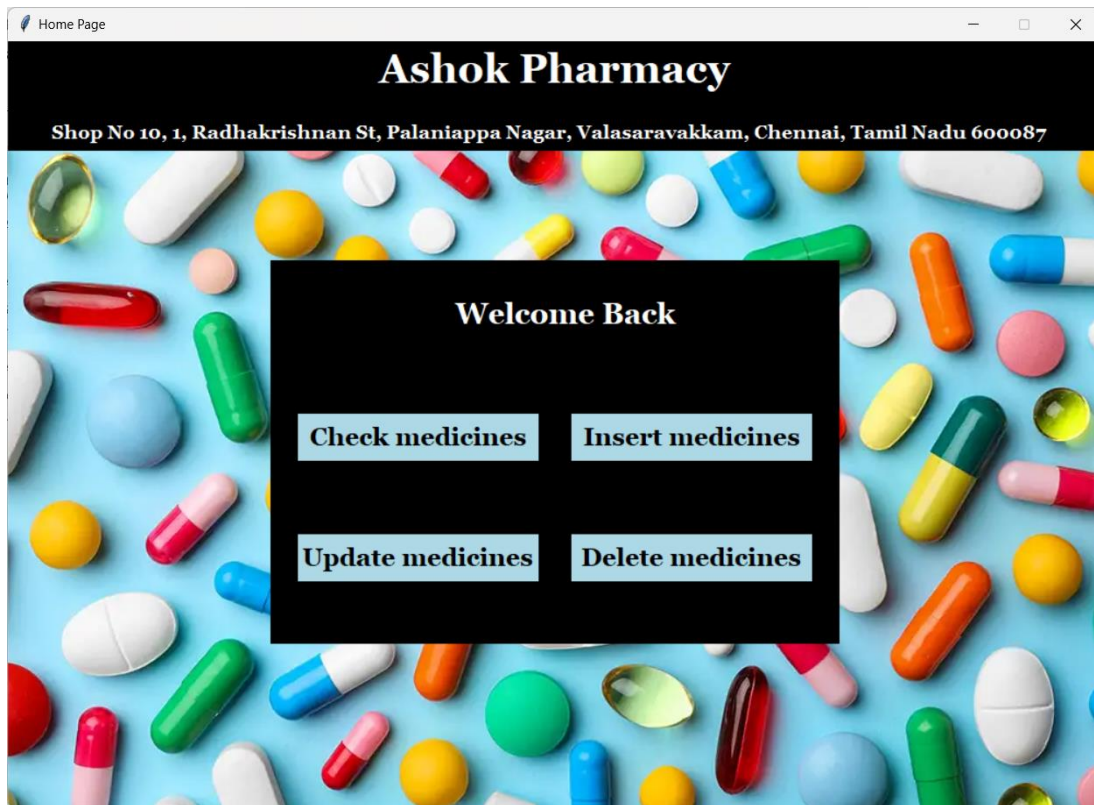
DROP TRIGGER update_medicine_stock;

CREATE OR REPLACE TRIGGER update_medicine_stock
AFTER INSERT ON SALES_ITEMS
FOR EACH ROW
BEGIN
    -- Check if the medicine exists before updating
    UPDATE MEDICINE
    SET quantity = quantity - :NEW.quantity
    WHERE medicine_id = :NEW.medicine_id
    AND EXISTS (SELECT 1 FROM MEDICINE WHERE medicine_id = :NEW.medicine_id);
END;
/

```

OUTPUT – FRONT END:






Checking Medicines in Medicine Page


Back

Viewing Medicine Table

Back to home



Medicine ID	Medicine Nam	Brand	Batch Number	Expiry Date	Quantity	Price	Supplier II
Mo05	Paracetamol	Acetaminophen	BATCH123	2026-05-01 00:00	10	50.0	So23
Mo18	Amoxicillin	Amoxil	BATCH543	2025-09-15 00:00	15	120.0	So10
Mo11	Ibuprofen	Advil	BATCH213	2026-02-20 00:00	200	75.5	So18
Mo02	Lisinopril	Prinivil	BATCH321	2025-12-10 00:00	90	80.0	So02
Mo24	Cetirizine	Zyrtec	BATCH999	2026-04-05 00:00	120	45.0	So17
Mo14	Metformin	Glucophage	BATCH888	2025-08-30 00:00	75	60.0	So22
Mo07	Amlodipine	Norvase	BATCH456	2026-01-25 00:00	60	100.0	So05
Mo22	Omeprazole	Prilosec	BATCH111	2025-07-18 00:00	80	55.0	So04
Mo16	Simvastatin	Zocor	BATCH222	2024-11-30 00:00	50	90.0	So15
Mo01	Clopidogrel	Plavix	BATCH333	2025-10-20 00:00	40	85.0	So19
Mo25	Levothyroxine	Synthroid	BATCH777	2026-08-14 00:00	110	70.0	So21
Mo13	Doxycycline	Vibramycin	BATCH444	2025-03-11 00:00	30	40.0	So06
Mo03	Hydrochlorothiaz	Hydrodiuril	BATCH555	2026-03-12 00:00	95	65.0	So14
Mo19	Furosemide	Lasix	BATCH888	2026-12-25 00:00	70	50.0	So12
Mo08	Montelukast	Singulair	BATCH666	2025-05-09 00:00	85	60.0	So09
Mo12	Pantoprazole	Protonix	BATCH999	2025-11-01 00:00	150	95.0	So08
Mo04	Cetirizine	Zyrtec	BATCH444	2026-10-30 00:00	65	45.0	So20
Mo15	Venlafaxine	Effexor	BATCH777	2025-06-17 00:00	120	150.0	So01
Mo06	Escitalopram	Lexapro	BATCH555	2026-09-09 00:00	75	80.0	So03
Mo20	Sertraline	Zoloft	BATCH222	2025-02-02 00:00	50	70.0	So11
Mo09	Bupropion	Wellbutrin	BATCH888	2024-12-05 00:00	80	65.0	So16
Mo10	Azithromycin	Zithromax	BATCH112	2025-08-15 00:00	100	90.0	So05



Customer Purchase History

Back


Customer Data

Back to home

Customer ID

C001

Check History



Customer ID	Customer Name	Prescription ID	Medicine Name	Quantity Sold	Sale Date	Most Recent Prescription
C001	Rajesh Sharma	P004	Cetirizine	2	2024-10-13 00:00:00	P004
C001	Rajesh Sharma	P004	Ibuprofen	1	2024-10-13 00:00:00	P004

Update Customer Page

[Back](#) **Update into Customer Table** [Back to home](#)

Customer ID

New Customer Name

New Contact Number

New Email

New Address

[Update Customer](#)

Success!

Customer ID C006 updated successfully.

[OK](#)

Insert in Supplier Page

[Back](#) **Insert into Supplier Table** [Back to home](#)

Supplier ID

Supplier Name

Contact Number

Email

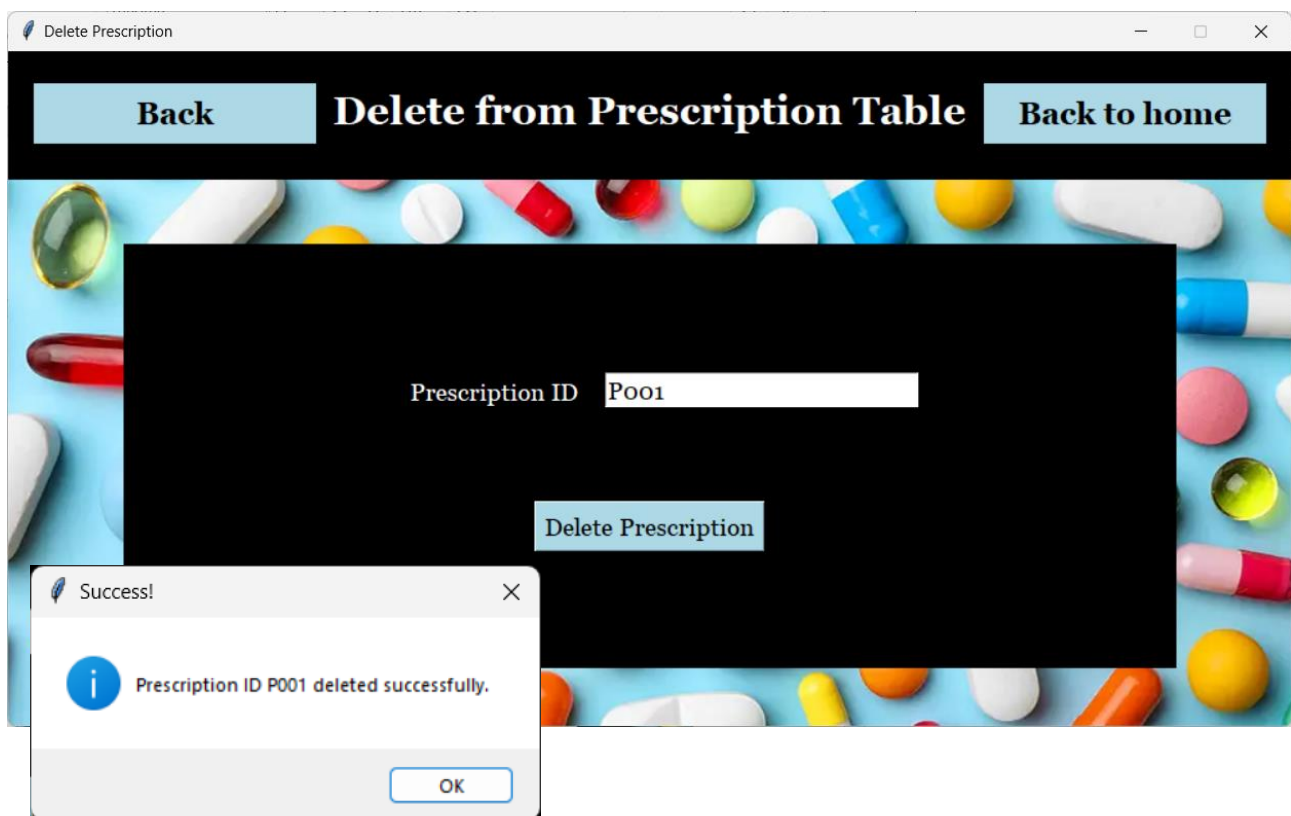
Address

[Enter](#)

Success!

Record inserted successfully into Supplier Table!

[OK](#)



RESULT: