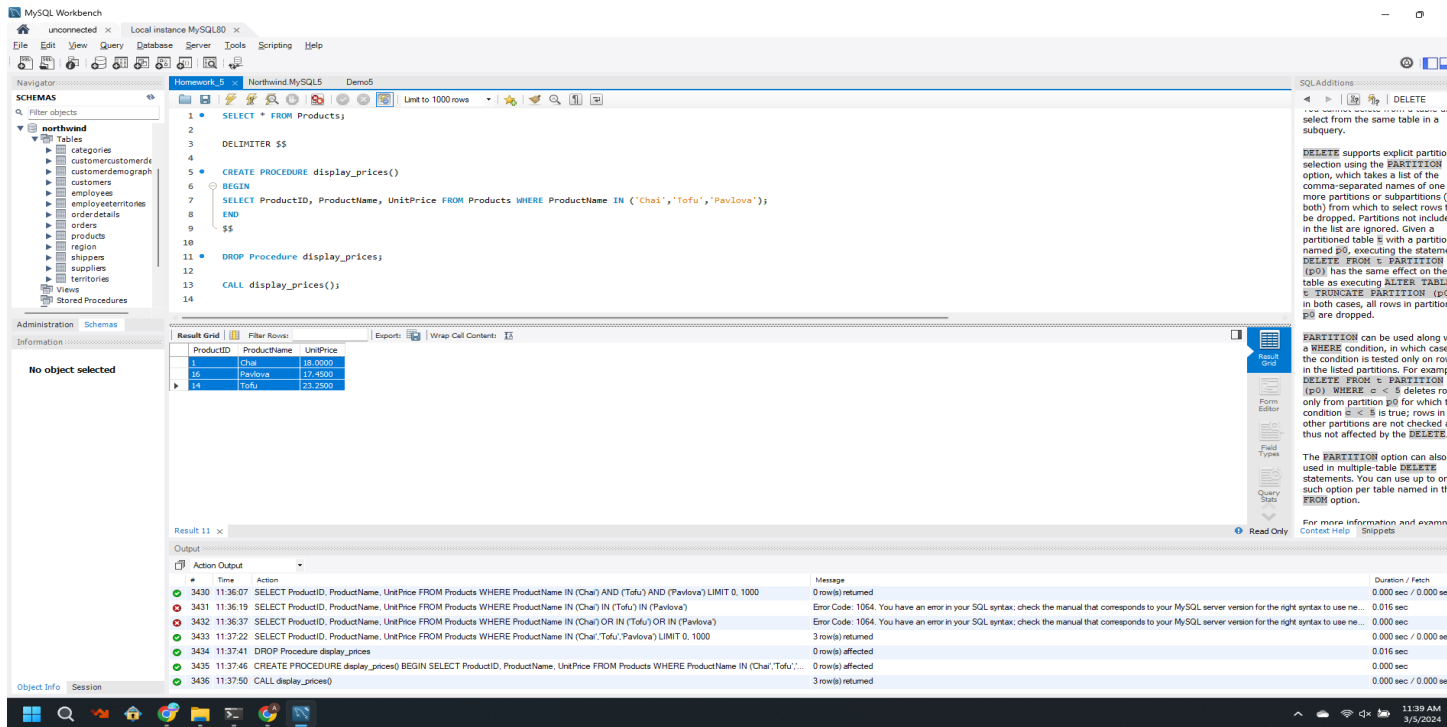


Homework_5_AparnaBharathi_Suresh

Question 1:

Write a stored procedure for Prices of Chai, Tofu, and Pavlova from the Products table in the **Northwind database**.

Screenshot 1:



Query 1:

```
DELIMITER $$
```

```
CREATE PROCEDURE display_prices()
```

```
BEGIN
```

```
SELECT ProductID, ProductName, UnitPrice FROM Products WHERE ProductName IN ('Chai','Tofu','Pavlova');
```

```
END
```

```
$$
```

```
#Call the procedure:
```

```
CALL display_prices();
```

Question 2:

Create an AFTER UPDATE trigger that promotes all students in the next class, i.e., 6 will be 7, 7 will be 8, and so on. Whenever an update is performed on a single row in the "students" table, a new row will be inserted in the "students_log" table.

Screenshot 2:

The screenshot displays the MySQL Workbench interface. The central editor shows a SQL script for creating and testing a trigger. The script includes a trigger named `after_update_rows` that fires after an update on the `students` table, inserting a log entry into the `students_log` table. The script also includes a `SELECT` statement to view the contents of `students_log`.

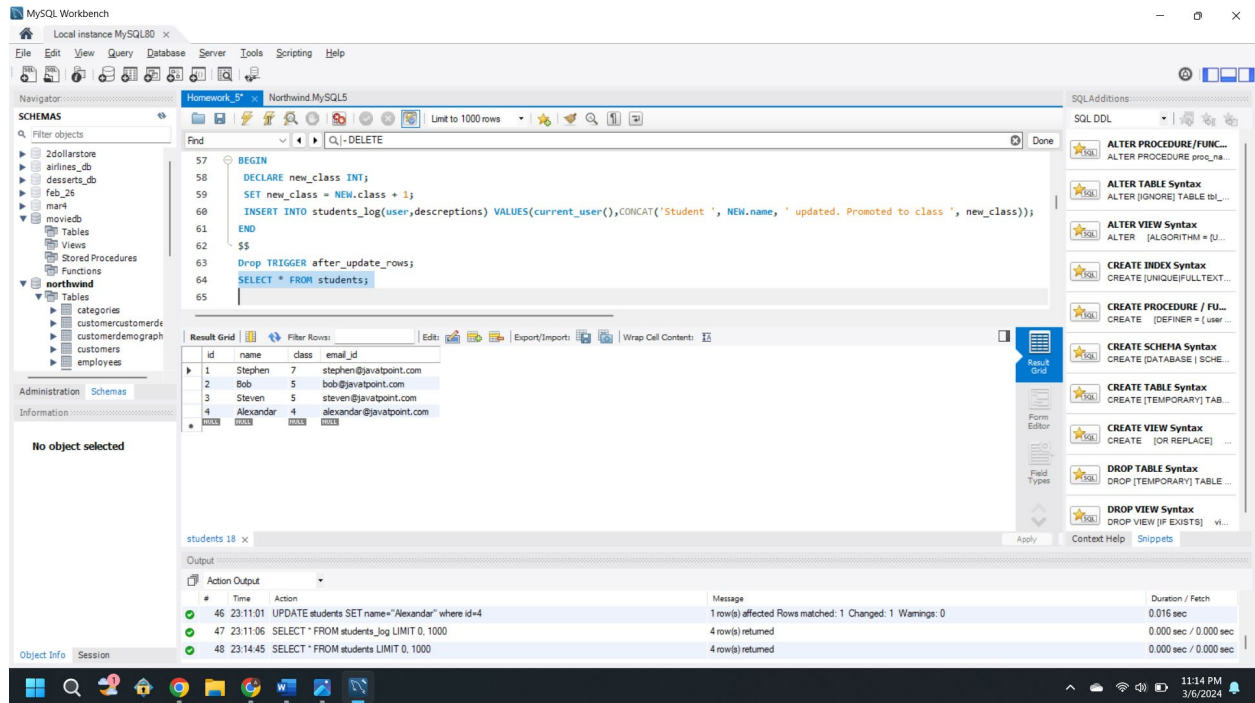
```
59 SET new_class = NEW.class + 1;
60 INSERT INTO students_log(user,descriptions) VALUES(current_user()),CONCAT('Student ', NEW.name, ' updated. Promoted to class ', new_class));
61 END
62 $$
63 Drop TRIGGER after_update_rows;
64 SELECT * FROM students;
65
66 SELECT * FROM students_log;
67
```

The `Result Grid` shows the output of the `SELECT * FROM students;` query, displaying four rows of student data:

user	descriptions
root@localhost	Student Stephen updated. Promoted to class 8
root@localhost	Student Bob updated. Promoted to class 6
root@localhost	Student Steven updated. Promoted to class 6
root@localhost	Student Alexander updated. Promoted to class 5

The `Output` pane at the bottom shows the execution log, including the `UPDATE` statements and the `SELECT` query results.

#	Time	Action	Message	Duration / Fetch
45	23:11:01	UPDATE students SET name="Steven" where id=3	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.000 sec
46	23:11:01	UPDATE students SET name="Alexander" where id=4	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.016 sec
47	23:11:06	SELECT * FROM students_log LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec



Query 2:

DELIMITER \$\$

CREATE TRIGGER after_update_rows

AFTER UPDATE

ON students FOR EACH ROW

BEGIN

DECLARE new_class INT;

SET new_class = NEW.class + 1;

INSERT INTO students_log(user,descreptions) VALUES(current_user(),CONCAT('Student ', NEW.name, ' updated. Promoted to class ', new_class));

END

\$\$

Question 3:

Write 1 **useful** stored procedure and trigger using the **Northwind Database**. (No restrictions for the type of stored procedure and trigger)

Screenshot 3:

Procedure:

I created an employee sales report Procedure which displays the EmployeeID, Name, and the Total Sales by each employee that is $\text{Sum}(\text{UnitPrice} * \text{quantity})$.

The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' panel with the 'northwind' database selected. The main editor window shows the SQL script for creating the stored procedure. The script includes a comment, a delimiter change, the procedure definition, and a call to the procedure. The 'Result Grid' shows the output of the procedure call, displaying a table with columns EmployeeID, FirstName, and TotalSales. The 'Output' panel at the bottom shows the execution log, including the creation of the procedure and the successful execution of the call.

```
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
```

EmployeeID	FirstName	TotalSales
1	Nancy	202143.7100
2	Andrew	177746.2600
3	Janet	213051.3000
4	Margaret	250187.4500
5	Steven	75567.7500
6	Michael	78198.1000
7	Robert	141295.9900
8	Laura	133301.0300
9	Anne	82964.0000

Output

#	Time	Action	Message	Duration / Patch
206	11:41:13	CREATE PROCEDURE Employee_SalesReport()		0.000 sec
207	11:41:22	CREATE PROCEDURE Employee_SalesReport()	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'EM...'	0.000 sec
208	11:41:35	CALL Employee_SalesReport()	0 row(s) affected	0.000 sec / 0.000 sec

Query 3:

```
DELIMITER %%
CREATE PROCEDURE Employee_SalesReport()
BEGIN
Select E.EmployeeID,E.FirstName,SUM(OD.UnitPrice*OD.Quantity) AS TotalSales FROM
EMPLOYEES E
```

```

JOIN Orders O ON E.EmployeeID = O.EmployeeID
JOIN `Order Details` OD ON O.OrderID=OD.OrderID GROUP BY E.EMPLOYEEID;
END
%%

```

```
CALL Employee_SalesReport();
```

Screenshot 3:

Trigger:

I created a trigger to prevent price reduction for each product. If you update the unit price of a product to a lesser value than the actual value, then it will throw error stating that Price reduction is not allowed.

The screenshot displays the MySQL Workbench interface. The main editor shows a SQL script with the following content:

```

188 JOIN `Order Details` OD ON O.OrderID=OD.OrderID
189 WHERE E.EmployeeID = employeeID;
190 END
191 %%
192
193 CALL Employee_SalesReport(2);
194
195 DELIMITER %%
196
197 • CREATE TRIGGER PreventPriceReduction
198 BEFORE UPDATE
199 ON Products FOR EACH ROW
200 BEGIN
201 DECLARE error_msg VARCHAR(200);
202 SET error_msg = 'Price Reduction is not allowed';
203
204 IF new.unitprice < old.unitprice THEN SIGNAL SQLSTATE '45000'
205
206 SET MESSAGE_TEXT = error_msg;
207 END IF;
208 END
209 %%
210
211 • UPDATE Products SET UnitPrice=10.00 WHERE ProductID=1;
212

```

The left sidebar shows the 'Schemas' panel with a tree view of the database structure. The 'Table: order details' is selected, showing its columns: OrderID (int PK), ProductID (int PK), UnitPrice (decimal(10,4)), Quantity (smallint), and Discount (double(8,0)).

The right sidebar shows the 'SQL DDL' panel with various SQL syntax examples.

The bottom panel shows the 'Output' window with the following execution log:

#	Time	Action	Message	Duration / Fetch
3535	14:42:52	SELECT * FROM EMPLOYEES LIMIT 0, 1000	9 row(s) returned	0.000 sec / 0.000 sec
3536	14:43:04	CALL Employee_SalesReport(1)	345 row(s) returned	0.000 sec / 0.000 sec
3537	14:44:56	CALL Employee_SalesReport(2)	241 row(s) returned	0.000 sec / 0.000 sec
3538	15:01:44	CREATE TRIGGER PreventPriceReduction BEFORE UPDATE ON Products FOR EACH ROW BEGIN DECLARE error_msg VARCHAR(200); SET error...	0 row(s) affected	0.015 sec
3539	15:02:01	Select * from products LIMIT 0, 1000	77 row(s) returned	0.015 sec / 0.000 sec
3540	15:02:52	UPDATE Products SET UnitPrice=10.00 WHERE ProductID=1	Error Code: 1054. Unknown column 'ProductID' in 'where clause'	0.000 sec
3541	15:03:05	UPDATE Products SET UnitPrice=10.00 WHERE ProductID=1	Error Code: 1644. Price Reduction is not allowed	0.000 sec

Query 3:

```
DELIMITER %%
```

```
CREATE TRIGGER PreventPriceReduction
BEFORE UPDATE
ON Products FOR EACH ROW
BEGIN
DECLARE error_msg VARCHAR(200);
SET error_msg = 'Price Reduction is not allowed';

IF new.unitprice < old.unitprice THEN SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT = error_msg;
END IF;
END
%%
```

#Update Query:

```
UPDATE Products SET UnitPrice=10.00 WHERE ProductID=1;
```

Question 4:

Write the trigger logic that updates the total salary into the total_salary_budget table after a row is deleted from the salaries table.

Screenshot 4:

Before Delete Salaries table:

The screenshot displays the MySQL Workbench interface. The main editor shows a SQL script for creating a trigger and a table. The script is as follows:

```
104
105 DELIMITER $$
106 * CREATE TRIGGER after_delete_rows
107 AFTER DELETE
108 ON Salaries1 FOR EACH ROW
109 BEGIN
110 UPDATE total_salary_budget SET total_budget = (SELECT SUM(amount) FROM salaries);
111 END
112 $$
113
114 * Select * from Salaries1;
115
```

The left sidebar shows the 'Schemas' pane with a tree view of the database structure. The 'Table: products' is selected, showing its columns and data types.

The bottom pane shows the 'Output' tab with the execution results of the script. The results are as follows:

Time	Action	Message	Duration / Fetch
3468 12:20:19	CREATE TABLE total_salary_budget (total_budget DECIMAL(10,2) NOT NULL)	0 row(s) affected	0.016 sec
3469 12:20:19	INSERT INTO total_salary_budget (total_budget) SELECT SUM(amount) FROM salaries1	1 row(s) affected Records: 1 Duplicates: 0 Warnings: 0	0.000 sec
3470 12:25:43	UPDATE total_salary_budget SET total_budget=NEW_total_budget Where total_budget in(SELECT SUM(amount) AS NEW_total_budget FROM salaries1)	Error Code: 1054. Unknown column 'NEW_total_budget' in 'field list'	0.000 sec
3471 12:26:02	SELECT * FROM Products LIMIT 0, 1000	Error Code: 1146. Table 'salary_products' doesn't exist	0.000 sec
3472 12:28:57	UPDATE total_salary_budget SET total_budget = (SELECT SUM(amount) FROM salaries1)	0 row(s) affected Rows matched: 1 Changed: 0 Warnings: 0	0.000 sec
3473 12:29:36	CREATE TRIGGER after_delete_rows AFTER DELETE ON Salaries1 FOR EACH ROW BEGIN UPDATE total_salary_budget SET total_budget = (SELECT SUM(amount) FROM salaries1)	0 row(s) affected	0.016 sec
3474 12:29:53	Select * from Salaries1 LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec

Before delete total salary budget table:

The screenshot shows the MySQL Workbench interface. The SQL editor contains a trigger named `after_delete_rows` that updates the `total_salary_budget` table after deleting rows from the `Salaries1` table. The trigger code is as follows:

```
106 CREATE TRIGGER after_delete_rows
107 AFTER DELETE
108 ON Salaries1 FOR EACH ROW
109 BEGIN
110 UPDATE total_salary_budget SET total_budget = (SELECT SUM(amount) FROM salaries1);
111 END
112 $$
113
114 Select * from Salaries1;
115
116 select * from total_salary_budget;
117
```

The `total_salary_budget` table is shown in the Results grid with a single row:

total_budget
279000.00

The `products` table structure is also visible in the left sidebar:

Column	Type	PK	AI
ProductID	int	PK	AI
ProductName	varchar(40)		
SupplierID	int		
CategoryID	int		
QuantityPerUnit	varchar(20)		
UnitPrice	decimal(10, 1)		
UnitsInStock	smallint		
UnitsOnOrder	smallint		
ReorderLevel	smallint		
Discontinued	bit(1)		

The Output pane shows the execution of the trigger and the selection of data from the `Salaries1` and `total_salary_budget` tables.

Delete a row:

The screenshot shows the MySQL Workbench interface after deleting a row from the `Salaries1` table. The SQL editor contains the following code:

```
95 CREATE TABLE total_salary_budget(
96
97     total_budget DECIMAL(10,2) NOT NULL
98
99 );
100
101 INSERT INTO total_salary_budget (total_budget)
102
103     SELECT SUM(amount) FROM salaries1;
104
105 DELIMITER $$
106 CREATE TRIGGER after_delete_rows
107 AFTER DELETE
108 ON Salaries1 FOR EACH ROW
109 BEGIN
110 UPDATE total_salary_budget SET total_budget = (SELECT SUM(amount) FROM salaries1);
111 END
112 $$
113
114 Delete from salaries1 where emp_num=102;
115
116 Select * from Salaries1;
117
118 select * from total_salary_budget;
119
```

The Output pane shows the execution of the `DELETE` statement and the subsequent trigger execution, which updates the `total_salary_budget` table. The output shows that the row with `emp_num=102` was deleted from `Salaries1`, and the `total_salary_budget` table was updated accordingly.

Salary table after deleting:

The screenshot shows MySQL Workbench with a query window containing the following SQL code:

```
106 CREATE TRIGGER after_delete_rows
107 AFTER DELETE
108 ON Salaries1 FOR EACH ROW
109 BEGIN
110 UPDATE total_salary_budget SET total_budget = (SELECT SUM(amount) FROM salaries1);
111 END
112 $$
113
114 Delete from salaries1 where emp_num=102;
115
116 Select * from Salaries1;
117
118 select * from total_salary_budget;
```

The left sidebar shows the 'Schemas' pane with a tree view of the database structure. The 'Table: products' is selected, showing its columns: ProductID, ProductName, SupplierID, CategoryID, QuantityPerUnit, UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel, and Discontinued.

The 'Result Grid' shows the state of the 'Salaries1' table after deletion:

emp_num	valid_from	amount
103	2020-01-10	65000.00
105	2020-01-10	55000.00
107	2020-01-10	70000.00
109	2020-01-10	40000.00

The 'Output' pane shows the execution log, including the deletion of the row with emp_num=102 and the subsequent update of the total_salary_budget table.

Salary budget table after deleting:

The screenshot shows MySQL Workbench with a query window containing the following SQL code:

```
108 ON Salaries1 FOR EACH ROW
109 BEGIN
110 UPDATE total_salary_budget SET total_budget = (SELECT SUM(amount) FROM salaries1);
111 END
112 $$
113
114 Delete from salaries1 where emp_num=102;
115
116 Select * from Salaries1;
117
118 select * from total_salary_budget;
```

The left sidebar shows the 'Schemas' pane with a tree view of the database structure. The 'Table: products' is selected, showing its columns: ProductID, ProductName, SupplierID, CategoryID, QuantityPerUnit, UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel, and Discontinued.

The 'Result Grid' shows the state of the 'total_salary_budget' table after deletion:

total_budget
2,000,00.00

The 'Output' pane shows the execution log, including the deletion of the row with emp_num=102 and the subsequent update of the total_salary_budget table.

Query 4:

#Trigger:

DELIMITER \$\$

CREATE TRIGGER after_delete_rows

AFTER DELETE

ON Salaries1 FOR EACH ROW

BEGIN

UPDATE total_salary_budget SET total_budget = (SELECT SUM(amount) FROM salaries1);

END

\$\$

#Delete Query

DELETE FROM salaries1 where emp_num=102;

Question 5:

Show customers(CompanyName,ContactName,ContactTitle,Phone) from Germany and Mexico using stored procedures.

Screenshot 5:

The screenshot displays the MySQL Workbench interface. The SQL editor contains the following code:

```
USE Northwind;
Select * from Customers;
DELIMITER $$
CREATE PROCEDURE Customers_Germany_Mexico()
BEGIN
SELECT CompanyName, ContactName, ContactTitle, Phone FROM customers WHERE Country IN ('Germany', 'Mexico');
END
$$
call Customers_Germany_Mexico();
```

The 'Schemas' pane on the left shows the 'northwind' database selected. The 'Result Grid' pane shows the output of the queries, including the execution of the stored procedure. The 'Output' pane at the bottom shows the execution log.

Time	Action	Message	Duration / Fetch
3479 12:34:29	select * from total_sales_budget LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
3480 12:45:48	USE Northwind	0 row(s) affected	0.000 sec
3481 12:46:12	SELECT * FROM customers LIMIT 0, 1000	93 row(s) returned	0.000 sec / 0.000 sec
3482 12:47:40	SELECT CompanyName, ContactName, ContactTitle, Phone FROM customers WHERE Country IN ('Germany', 'Mexico') LIMIT 0, 1000	16 row(s) returned	0.000 sec / 0.000 sec
3483 12:47:59	Select * from Customers LIMIT 0, 1000	93 row(s) returned	0.016 sec / 0.000 sec
3484 12:51:04	CREATE PROCEDURE Customers_Germany_Mexico() BEGIN SELECT CompanyName, ContactName, ContactTitle, Phone FROM customers WHERE	0 row(s) affected	0.000 sec
3485 12:51:09	call Customers_Germany_Mexico()	16 row(s) returned	0.015 sec / 0.000 sec

Query 5:

#Procedure:

DELIMITER \$\$

CREATE PROCEDURE Customers_Germany_Mexico()

BEGIN

SELECT CompanyName, ContactName, ContactTitle, Phone FROM customers WHERE Country IN ('Germany', 'Mexico');

END

\$\$

#Call Procedure:

call Customers_Germany_Mexico();

Question 6:

Using the schema below, write a query to display order id, order date, order total and running order total based on the order date, and sort the results using order date. (2 marks)

Screenshot 6:

The screenshot shows the MySQL Workbench interface. The 'Schemas' pane on the left lists various databases, with 'northwind' selected. The 'Query Editor' in the center contains the following SQL query:

```
SELECT Order_id, Order_date, Order_total, SUM(Order_Total) OVER (ORDER BY Order_Date) AS Running_total FROM ORDERS;
```

The 'Result Grid' below the query shows the results of the query. The columns are 'Order_id', 'Order_date', 'Order_total', and 'Running_total'. The results are sorted by 'Order_date'.

Order_id	Order_date	Order_total	Running_total
1	2020-04-03	100	470
2	2020-04-03	250	470
5	2020-04-03	120	470
3	2020-04-04	80	705
6	2020-04-04	90	705
7	2020-04-04	90	705
8	2020-04-04	15	705
4	2020-04-05	10	715

The 'Output' pane at the bottom shows the execution log, including the query execution time and the number of rows returned.

Query 6:

SELECT Order_id, Order_date, Order_total, SUM(Order_Total) OVER (ORDER BY Order_Date)
AS Running_total FROM Orders;