

Unit 05 Problem Set Submission Form

Overview

Your Name	Jainish Savaliya
Your SU Email	jsavaliy@syr.edu

Instructions

Put your name and SU email at the top. Answer these questions all from the lab. When asked to include screenshots, please follow the screen shot guidelines from the first lab.

Remember as you complete the problem sets it is not only about getting it right / correct. We will discuss the answers in class so it's important to articulate anything you would like to contribute to the discussion in your answer:

- If you feel the question is vague, include any assumptions you've made.
- If you feel the answer requires interpretation or justification provide it.
- If you do not know the answer to the question, articulate what you tried and how you are stuck.

This how you receive credit for answering questions which might not be correct.

Questions

Answer these questions using the problem set submission template. You will need to consult the logical model in the overview section for details. For any screenshots provided, please follow the guidelines for submitting a screenshot.

Write the following as SQL queries. If the query is ambiguous, fill in the gaps yourself and justify your reasoning. For each, include the SQL as a screenshot with the output of the query.

1. How many item types are there? Perform an analysis of each item type. For each item type, provide the count of items in that type, the minimum, average, and maximum item reserve prices for that type. Sort the output by item type.

The screenshot shows the SQL Server Enterprise Manager interface. The query editor contains the following SQL code:

```
--1
select item_type, count(item_reserve) as count_of_types, MIN(item_reserve) as Min_item_type,
MAX(item_reserve) as Max_item_type, avg(item_reserve) as Avg_item_type
from vb_items
group by item_type
order by item_type

--2
SELECT item_name, item_type, item_reserve, COUNT(item_reserve) as count_item_reserve,
MIN(item_reserve) over (partition by item_type) as Min_Item_reserve,
MAX(item_reserve) over (partition by item_type) as Max_Item_reserve,
avg(item_reserve) over (partition by item_type) as Avg_Item_reserve
from vb_items
where item_type='Antiques' or item_type = 'collectables'
```

The Results pane displays the following table:

item_type	count_of_types	Min_item_type	Max_item_type	Avg_item_type
All Other	4	0.99	10000000.00	2500004.86
Antiques	6	9.00	250.00	81.5833
Books	3	4.50	10.99	8.48
Collectables	14	5.00	500.00	105.3828
Electronics	1	15.00	15.00	15.00
Jewelry	2	6.95	599.99	303.47
Sporting Goods	2	12.50	12.75	12.625
Tickets	2	5.00	750.00	377.50

- Perform an analysis of each item in the “Antiques” and “Collectables” item types. For each item display the name, item type and item reserve. Include the min, max and average item reserve over each item type so that the current item reserve can be compared to these values.

The screenshot shows the SQL Server Enterprise Manager interface. The query editor contains the following SQL code:

```
--2
SELECT item_name, item_type, item_reserve, COUNT(item_reserve) as count_item_reserve,
MIN(item_reserve) over (partition by item_type) as Min_Item_reserve,
MAX(item_reserve) over (partition by item_type) as Max_Item_reserve,
avg(item_reserve) over (partition by item_type) as Avg_Item_reserve
from vb_items
where item_type='Antiques' or item_type = 'collectables'
group by item_type, item_name, item_reserve
order by item_type

--3
--select u.user_firstname, u.user_lastname, COUNT(*) as number_of_ratings ,
--avg(r.rating) as avg_rating
--from vb_items i
--join vb_users u on i.user_id = u.user_id
--join vb_ratings r on i.item_id = r.item_id
```

The Results pane displays the following table:

item_name	item_type	item_reserve	count_item_reserve	Min_Item_reserve	Max_Item_reserve	Avg_Item_reserve
a Toaster	Antiques	20.00	1	9.00	250.00	81.5833
Antique Desk	Antiques	250.00	1	9.00	250.00	81.5833
Brass French Press	Antiques	45.00	1	9.00	250.00	81.5833
case of vintage tube socks	Antiques	9.00	1	9.00	250.00	81.5833
Dusty Vase	Antiques	100.00	1	9.00	250.00	81.5833
Original Coke Bottle from 19...	Antiques	65.00	1	9.00	250.00	81.5833
Alf Alarm Clock	Collectables	5.00	1	5.00	500.00	105.3828
Autographed Mik Jagger Poster	Collectables	75.00	1	5.00	500.00	105.3828
Carlos Villalba BobbleHead	Collectables	49.95	1	5.00	500.00	105.3828
Dukes Of Hazard ashtray	Collectables	149.99	1	5.00	500.00	105.3828
Farrah Fawcett poster	Collectables	50.00	1	5.00	500.00	105.3828
Joe Montana Figurine	Collectables	200.00	1	5.00	500.00	105.3828
Kleenex used by Dr. Dre	Collectables	500.00	1	5.00	500.00	105.3828
Mike Fudge BobbleHead	Collectables	49.95	1	5.00	500.00	105.3828
PacMan Fever lunchbox	Collectables	29.99	1	5.00	500.00	105.3828
Pez dispensers	Collectables	10.00	1	5.00	500.00	105.3828
Rare Mint Snow Globe	Collectables	30.50	1	5.00	500.00	105.3828

- Write a query to include the names, counts (number of ratings) and average seller ratings (as a decimal) of users. For reference, User Carrie Dababbbi has 4 seller ratings and an average rating of 4.75.

The screenshot shows the SQL Azure Data Studio interface. The query editor contains the following SQL code:

```

16 order by item_type
17
18 --3
19 select u.user_firstname, u.user_lastname, COUNT(*) as number_of_ratings,
20 AVG(cast(rating_value as decimal (15,2))) as avg_rating_value
21 from vb_user_ratings r
22 join vb_users u on u.user_id = r.rating_for_user_id
23 group by u.user_firstname, u.user_lastname
24
25 --4
26 with ct as(
27 SELECT v.item_name, COUNT(*) as count_of_bids from vb_bids b
28 left join vb_items v on b.bid_item_id = v.item_id
29
30 Results
31 Messages
32
33 Results grid
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

The results grid displays the following data:

user_firstname	user_lastname	number_of_ratings	avg_rating_value	
1	Rose	Abou-Duresst	3	1.000000
2	Ty	Anott	2	2.500000
3	Barb	Barion	4	4.250000
4	Carrie	Dababbi	4	4.750000
5	Barry	Delatchett	2	5.000000
6	Martin	Eyazing	2	5.000000
7	Isabelle	Gunninger	2	4.000000
8	Les	Ismoore	8	1.875000
9	Anita	Job	1	3.000000
10	Abby	Kuss	5	4.200000
11	Mary	Melator	4	4.500000
12	Victor	Rhee	1	4.000000
13	Gus	Toffwind	2	3.000000

4. Create a list of "Collectable" item types with more than 1 bid. Include the name of the item and the number of bids making sure the item with the most bids appear first.

The screenshot shows the SQL Azure Data Studio interface. The query editor contains the following SQL code:

```

23 --join vb_users u on u.user_id = r.rating_for_user_id
24 --group by u.user_firstname, u.user_lastname
25
26 --4
27 with ct as(
28 SELECT v.item_name, COUNT(*) as count_of_bids from vb_bids b
29 left join vb_items v on b.bid_item_id = v.item_id
30 where v.item_type='collectables'
31 group by v.item_name,b.bid_item_id )
32
33 select ct.item_name, ct.count_of_bids from ct
34 where ct.count_of_bids > 1
35 order by count_of_bids DESC
36
37 Results
38 Messages
39
40 Results grid
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

The results grid displays the following data:

item_name	count_of_bids	
1	Dukes Of Hazard ashtray	9
2	Autographed Mik Jagger Poster	6
3	Shatner's old Toupee	5
4	Rare Mint Snow Globe	3
5	Farrah Fawcett poster	3
6	Pez dispensers	2

5. Generate a valid bidding history for any given item of your choice. Display the item id, item name a number representing the order the bid was placed, the bid amount and the bidder's name. Here's

an example showing the first 3 bids on item 11.

item_id	item_name	bid_order	bid_amount	bidder
11	Dukes Of Hazard ashtray	1	150.0000	Dan Delyons
11	Dukes Of Hazard ashtray	2	175.0000	Al Fresco
11	Dukes Of Hazard ashtray	3	200.0000	Carrie Dababbi

The screenshot shows the SQL Server Enterprise Manager interface. The query editor displays the following SQL code:

```
--5
select item_id, item_name,
ROW_NUMBER() over (partition by item_id order by bid_datetime) as bid_order,
bid_amount, user_firstname + ' ' + user_lastname as bidder_name
from vb_items i
join vb_bids b on b.bid_item_id = i.item_id
join vb_users u on b.bid_user_id = u.user_id
where bid_status = 'ok' and item_id = 32

--6
select i.item_id, i.item_name,
ROW_NUMBER() over (partition by item_id order by b.bid_datetime) as bid_order,
b.bid_amount, u.user_firstname + ' ' + u.user_lastname as bidder,
LAG(u.user_firstname + ' ' + u.user_lastname) over (partition by i.item_id order by b.bid_datetime) as prev_bidder,
```

The Results pane shows the following data:

	item_id	item_name	bid_order	bid_amount	bidder_name
1	32	Ten Speed Bike	1	13.00	Barry DeHatchett
2	32	Ten Speed Bike	2	14.00	Isabelle Gunnering
3	32	Ten Speed Bike	3	15.00	Barry DeHatchett
4	32	Ten Speed Bike	4	16.00	Isabelle Gunnering
5	32	Ten Speed Bike	5	17.00	Barry DeHatchett
6	32	Ten Speed Bike	6	18.00	Isabelle Gunnering
7	32	Ten Speed Bike	7	19.00	Barry DeHatchett
8	32	Ten Speed Bike	8	20.00	Isabelle Gunnering
9	32	Ten Speed Bike	9	21.00	Barry DeHatchett
10	32	Ten Speed Bike	10	22.00	Isabelle Gunnering

A Notepad window is open in the background, displaying the email address jsavaliy@syr.edu.

- Re-Write your query in the previous question to include the names of the next and previous bidders, like this example again showing the first 3 bids for item 11.

SQLQuery_2 - localhost.vbay (sa) - Azure Data Studio [Administrator]

```

43 join vb_users u on b.bid_user_id = u.user_id
44 where bid_status = 'ok' and item_id = 32
45
46 --6
47 select i.item_id, i.item_name,
48 ROW_NUMBER() over (partition by item_id order by b.bid_datetime) as bid_order,
49 b.bid_amount, u.user_firstname + ' ' + u.user_lastname as bidder,
50 LAG(u.user_firstname + ' ' + u.user_lastname) over (partition by i.item_id order by b.bid_datetime) as prev_bidder,
51 LEAD(u.user_firstname + ' ' + u.user_lastname) over (partition by i.item_id order by b.bid_datetime) as next_bidder
52 from vb_bids as b
53 join vb_items as i on i.item_id = b.bid_item_id
54 join vb_users as u on u.user_id = b.bid_user_id
55 where i.item_id = 11 and b.bid_status = 'ok'
56

```

Results Messages

	item_id	item_name	bid_order	bid_amount	bidder	prev_bidder	next_bidder
1	11	Dukes Of Hazard ashtray	1	150.00	DanDelyons	NULL	AlFresco
2	11	Dukes Of Hazard ashtray	2	175.00	AlFresco	DanDelyons	CarrieDababbi
3	11	Dukes Of Hazard ashtray	3	200.00	CarrieDababbi	AlFresco	GusToffwind
4	11	Dukes Of Hazard ashtray	4	225.00	GusToffwind	CarrieDababbi	IsabelleGunninger
5	11	Dukes Of Hazard ashtray	5	250.00	IsabelleGunninger	GusToffwind	DanDelyons
6	11	Dukes Of Hazard ashtray	6	275.00	DanDelyons	IsabelleGunninger	CarrieDababbi
7	11	Dukes Of Hazard ashtray	7	300.00	CarrieDababbi	DanDelyons	IsabelleGunninger
8	11	Dukes Of Hazard ashtray	8	325.00	IsabelleGunninger	CarrieDababbi	NULL

Ln 47, Col 1 (574 selected) Spaces: 4 UTF-8 CRLF 8 rows MSSQL 00:00:00 localhost: vbay 8:33 PM 2/21/2023

item_name	bid_order	bid_amount	prev_bidder	bidder	next_bidder
Dukes Of Hazard ashtray	1	150.0000	NULL	Dan Delyons	Al Fresco
Dukes Of Hazard ashtray	2	175.0000	Dan Delyons	Al Fresco	Carrie Dababbi
Dukes Of Hazard ashtray	3	200.0000	Al Fresco	Carrie Dababbi	Gus Toffwind

7. Find the names and emails of the users who give out the worst ratings (lower than the overall average rating) to either buyers or sellers (no need to differentiate whether the user rated a buyer or seller), and only include those users who have submitted more than 1 rating.

The screenshot shows the SQL Server Enterprise Manager interface. The query window contains the following SQL code:

```

--R
with ratings_tables as (select rating_id, u.user_firstname as user_name, u.user_email, avg(r
over() as overall_average_rating, count(u.user_firstname)
OVER (partition by u.user_firstname) as count_of_reviews
from vb_users u
join vb_user_ratings r on u.user_id = r.rating_for_user_id
group by r.rating_id, u.user_firstname, u.user_email, rating_value)
select USER_NAME, user_email
from ratings_tables
join vb_user_ratings r on r.rating_id = ratings_tables.rating_id
where r.rating_value < overall_average_rating and count_of_reviews > 1
--R

```

The results grid displays the following data:

USER_NAME	user_email
1 Barb	bbarion@mail.org
2 Les	lismoore@mail.org
3 Les	lismoore@mail.org
4 Les	lismoore@mail.org
5 Les	lismoore@mail.org
6 Les	lismoore@mail.org
7 Martin	meveyzing@mail.org
8 Rose	rabovdu@mail.org
9 Rose	rabovdu@mail.org
10 Rose	rabovdu@mail.org
11 Ty	tanott@mail.org

- Produce a report of the KPI (key performance indicator) user bids per item. Show the user's name and email total number of valid bids, total count of items bid upon and then the ratio of bids to items. As a check, Anne Dewey's bids per item ratio is 1.666666

The screenshot shows the SQL Server Enterprise Manager interface. The query window contains the following SQL code:

```

--R
with uBid as (select u.user_firstname + ' ' + u.user_lastname as userName,
u.user_email as userEmail, count(b.bid_user_id) as totalCountOfBids,
COUNT(distinct b.bid_item_id) as total_items_bid
from vb_users u
join vb_bids b on u.user_id = b.bid_user_id
where b.bid_status = 'ok'
group by u.user_firstname + ' ' + u.user_lastname, u.user_email)
select userName, userEmail, totalCountOfBids, total_items_bid, cast(totalCountOfBids as decimal) / cast(total_items_bid as decimal) as KPI from uBid
--R

```

The results grid displays the following data:

userName	userEmail	totalCountOfBids	total_items_bid	KPI
1 Abby Kuss	abuss@mail.org	3	1	3.0000000000000000
2 Anne Dewey	adewey@mail.org	5	3	1.6666666666666666
3 Barb Barion	bbarion@mail.org	3	2	1.5000000000000000
4 Barry DeHatchett	bdehatchett@mail.org	5	1	5.0000000000000000
5 Bo Enarreau	benarreau@mail.org	2	2	1.0000000000000000
6 Gus Toffulnd	gtoffulnd@mail.org	2	2	1.0000000000000000
7 Isabelle Gunnering	igunner@mail.org	7	2	3.5000000000000000
8 Les Ismoore	lismoore@mail.org	3	3	1.0000000000000000
9 Martin Eyezing	meveyzing@mail.org	1	1	1.0000000000000000
10 Rose Abov-Duresst	rabovdu@mail.org	2	2	1.0000000000000000
11 Ray Ovligh	rovlight@mail.org	6	3	2.0000000000000000
12 Victor Rhee	vrhee@mail.org	2	2	1.0000000000000000
13 Seymour Ofewe	sofewe@mail.org	2	1	2.0000000000000000
14 Pete Moss	pmoss@mail.org	2	2	1.0000000000000000
15 Otto Moni	omoni@mail.org	2	2	1.0000000000000000
16 Oliver Stuffle	ostuff@mail.org	2	1	2.0000000000000000

- Among items not sold, show highest bidder name and the highest bid for each item. Make sure to include only valid bids.

The screenshot shows a SQL Server Enterprise Manager window with a query editor and a results pane. The query is as follows:

```
--9
select u.user_firstname + ' ' + user_lastname as highest_bidder_name, i.item_name,
FIRST_VALUE(b.bid_amount) over (partition by b.bid_item_id order by b.bid_amount desc)
as highest_bid,
FIRST_VALUE(b.bid_user_id) over (partition by b.bid_item_id order by b.bid_amount desc)
as highest_bid_user_id
from vb_items as i
join vb_bids as b on i.item_id = b.bid_item_id
join vb_users as u on u.user_id = b.bid_user_id
where i.item_sold = '0' and b.bid_status = 'ok'
with ratings as(
```

The results pane shows the following data:

	highest_bidder_name	item_name	highest_bid	highest_bid_user_id
1	Lesismore	Alf Alarm Clock	5.01	8
2	DanDelyons	Shatner's old Toupee	202.00	23
3	JeanPoole	Shatner's old Toupee	202.00	23
4	DanDelyons	Shatner's old Toupee	202.00	23
5	SeymourOfewe	Slightly-damaged Golf Bag	14.50	15
6	RoseAbov-Duresst	Slightly-damaged Golf Bag	14.50	15
7	SeymourOfewe	Slightly-damaged Golf Bag	14.50	15
8	Lesismore	Some Beanie Babies, New with...	250.00	8
9	IsabelleGunning	Dukes Of Hazard ashtray	325.00	7
10	CarrieDababbi	Dukes Of Hazard ashtray	325.00	7
11	DanDelyons	Dukes Of Hazard ashtray	325.00	7
12	IsabelleGunning	Dukes Of Hazard ashtray	325.00	7
13	GusToffvind	Dukes Of Hazard ashtray	325.00	7
14	CarrieDababbi	Dukes Of Hazard ashtray	325.00	7
15	Alfresco	Dukes Of Hazard ashtray	325.00	7
16	DanDelyons	Dukes Of Hazard ashtray	325.00	7
17	VictorRhee	case of vintage tube socks	9.02	13

10. Write a query with output similar to question 3, but also includes the overall average seller rating, and the difference between each user's average rating and the overall average. For reference, the overall average seller rating should be 3.2.

The screenshot shows a SQL Server Enterprise Manager window with a query editor and a results pane. The query is as follows:

```
--10
with rating as(
select cast(sum(rating_value) as decimal(6,2))/cast(count(rating_value) as decimal(6,2))
as overall_rating
from vb_user_ratings
where rating_astype = 'Seller'
)
select user_firstname + ' ' + user_lastname as username Rated, count(ur.rating_value)
as total_count_of_user_ratings,
(CAST(SUM(ur.rating_value) as decimal(4,2))/count(ur.rating_value)) as average_user_rating,
(CAST(SUM(ur.rating_value) as decimal(6,2))- r.overall_rating) as DIFFERENCE, r.overall_rating
from rating r, vb_users as u
join vb_user_ratings ur on u.user_id = ur.rating_for_user_id
group by user_firstname + ' ' + user_lastname, r.overall_rating
```

The results pane shows the following data:

	username Rated	total_count_of_user_ratings	average_user_rating	DIFFERENCE	overall_rating
1	AbbyKuss	5	4.20000000000000	17.8000000000	3.2000000000
2	AnitaJob	1	3.00000000000000	-0.2000000000	3.2000000000
3	BarbBarion	4	4.25000000000000	13.8000000000	3.2000000000
4	BarryDeHatchett	2	5.00000000000000	6.8000000000	3.2000000000
5	CarrieDababbi	4	4.75000000000000	15.8000000000	3.2000000000
6	GusToffvind	2	3.00000000000000	2.8000000000	3.2000000000
7	IsabelleGunning	2	4.00000000000000	4.8000000000	3.2000000000
8	Lesismore	8	1.87500000000000	11.8000000000	3.2000000000
9	MartinEyzing	2	2.50000000000000	1.8000000000	3.2000000000
10	MaryHelator	4	4.50000000000000	14.8000000000	3.2000000000
11	RoseAbov-Duresst	3	1.00000000000000	-0.2000000000	3.2000000000
12	TyAnott	2	2.50000000000000	1.8000000000	3.2000000000
13	VictorRhee	1	4.00000000000000	0.8000000000	3.2000000000

Reflection

Use this section to reflect on your learning. To achieve the highest grade on the assignment you must be as descriptive and personal as possible with your reflection.

1. What are the key things you learned through the process of completing this assignment?
The key thing I learnt was to use window functions like over and partition.
2. What were the challenges or roadblocks (if any) you encountered on the way to completing it?
I was encountering the query spacing issues because the spacing during the aggregation should be consistent.
3. Were you prepared for this assignment? What can you do to be better prepared?
I took the notes during I was watching the video and referred it during the assignment
4. Now that you have completed the assignment rate your comfort level with this week's material. This should be an honest assessment: (choose one)
4 ==> I understand this material and can explain it to others.