

Final Project: Advanced SQL Techniques

Objectives

1. Using joins to query data from multiple tables
2. Creating and querying views
3. Writing and running stored procedures
4. Using transactions

Scenario

In this project, you will work with three datasets that are available on the City of Chicago's Data Portal:

- Socioeconomic indicators in Chicago
- Chicago public schools
- Chicago crime data

You must download each dataset, create a table for each one, and load the appropriate dataset through the Db2 console. If you have already completed the hands-on Joins, you can reuse the tables you created for that hands-on lab. However, you should not reuse similar tables with other names from other exercises or labs, as they may not create the correct results.

Exercise 1: Using Joins

You have been asked to produce some reports about the communities and crimes in the Chicago area. You will need to use SQL join queries to access the data stored across multiple tables.

Question 1

- Write and execute a SQL query to list the school names, community names and average attendance for communities with a hardship index of 98.

Take a screenshot showing the SQL query and its results.

Untitled - 1

```

1 SELECT S.name_of_school, C.community_area_number, S.average_student_attendan
2 FROM CHICAGO_PUBLIC_SCHOOLS S
3 INNER JOIN CENSUS_DATA C
4 ON S.community_area_number = C.community_area_number
5 WHERE hardship_index = .98;

```

Run all

Remember my selection

Result - Jul 13, 2021 2:14:46 PM

SELECT 5...

Run time: 0.046 s

Result set 1

NAME_OF_SCHOOL	C
George Washington Carver Military Academy High School	54
George Washington Carver Primary School	54
Ira F Aldridge Elementary School	54
William E B Dubois Elementary School	54

Question 2

- Write and execute a SQL query to list all crimes that took place at a school. Include case number, crime type and community name.

Take a screenshot showing the SQL query and its results.

on Cloud

Run SQL

Untitled - 1

```

1 SELECT a.case_number, a.primary_type, b.community_area_name
2 FROM CHICAGO_CRIME_DATA a
3 LEFT JOIN CENSUS_DATA b
4 ON a.community_area_number = b.community_area_number
5 WHERE location_description LIKE 'SCHOOL';

```

Run all

Remember my selection

Result set 1

CASE_NUMBER	PRIMARY_TYPE	COMMUNITY_AREA_NAME
HS200939	CRIMINAL DAMAGE	Austin
HT315369	ASSAULT	East Garfield Park
HP716225	BATTERY	Douglas
HL353697	BATTERY	South Shore
HS305355	NARCOTICS	Brighton Park
JA460432	BATTERY	Ashburn
HR585012	CRIMINAL TRESPA	Ashburn
HH292682	PUBLIC PEACE VI	
G635735	PUBLIC PEACE VI	

Exercise 2: Creating a View

For privacy reasons, you have been asked to create a view that enables users to select just the school name and the icon fields from the CHICAGO_PUBLIC_SCHOOLS table. By providing a view, you can ensure that users cannot see the actual scores given to a school, just the icon associated with their score. You should define new names for the view columns to obscure the use of scores and icons in the original table.

Question 1

- Write and execute a SQL statement to create a view showing the columns listed in the following table, with new column names as shown in the second column.

Column name in CHICAGO_PUBLIC_SCHOOLS	Column name in view
---------------------------------------	---------------------

NAME_OF_SCHOOL	School_Name
Safety_Icon	Safety_Rating
Family_Involvement_Icon	Family_Rating
Environment_Icon	Environment_Rating
Instruction_Icon	Instruction_Rating
Leaders_Icon	Leaders_Rating
Teachers_Icon	Teachers_Rating

- Write and execute a SQL statement that returns all of the columns from the view.
- Write and execute a SQL statement that returns just the school name and leaders rating from the view.

Take a screenshot showing the last SQL query and its results.

The screenshot shows a SQL IDE interface. On the left, the SQL editor contains the following code:

```
1 CREATE VIEW chicago_schools (School_Name, Safety_Rating, Family_Rating, Environment_Rating, Instruction_Rating, Leaders_Rating, Teachers_Rating)
2 AS
3 SELECT NAME_OF_SCHOOL, Safety_Icon, Family_Involvement_Icon, Environment_Icon, Instruction_Icon, Leaders_Icon, Teachers_Icon
4 FROM CHICAGO_PUBLIC_SCHOOLS;
5
6
7
8 select School_Name, Leaders_Rating from chicago_schools;
```

At the bottom of the editor, there is a "Run selected" button and a checkbox labeled "Remember my selection" which is checked.

On the right, the "Result set 1" pane displays the results of the last query. It shows a table with two columns: "SCHOOL_NAME" and "LEADERS_RAT". The results are as follows:

SCHOOL_NAME	LEADERS_RAT
Abraham Lincoln Elementary School	Weak
Adam Clayton Powell Paideia Community Academy Elementary School	Weak
Adlai E Stevenson Elementary School	Weak
Agustin Lara Elementary Academy	Weak
Air Force Academy High School	Weak
Albany Park Multicultural Academy	Weak
Albert G Lane Technical High School	Weak

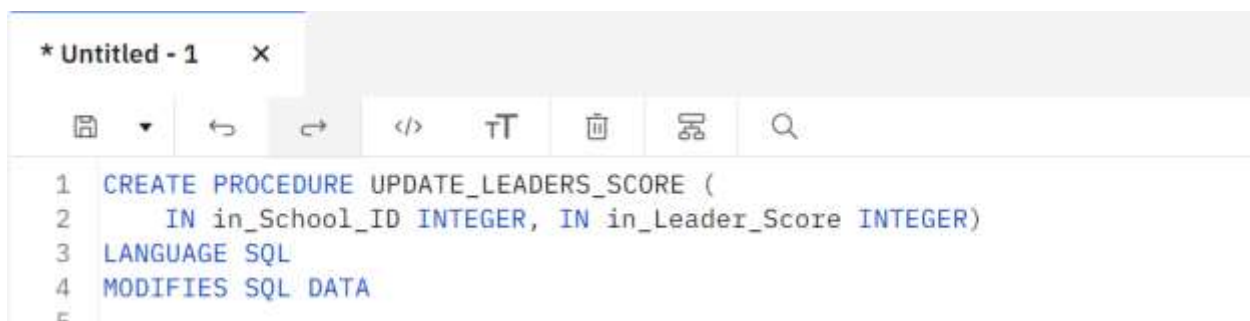
Exercise 3: Creating a Stored Procedure

The icon fields are calculated based on the value in the corresponding score field. You need to make sure that when a score field is updated, the icon field is updated too. To do this, you will write a stored procedure that receives the school id and a leaders score as input parameters, calculates the icon setting and updates the fields appropriately.

Question 1

- Write the structure of a query to create or replace a stored procedure called UPDATE_LEADERS_SCORE that takes a in_School_ID parameter as an integer and a in_Leader_Score parameter as an integer. Don't forget to use the #SET TERMINATOR statement to use the @ for the CREATE statement terminator.

Take a screenshot showing the SQL query.

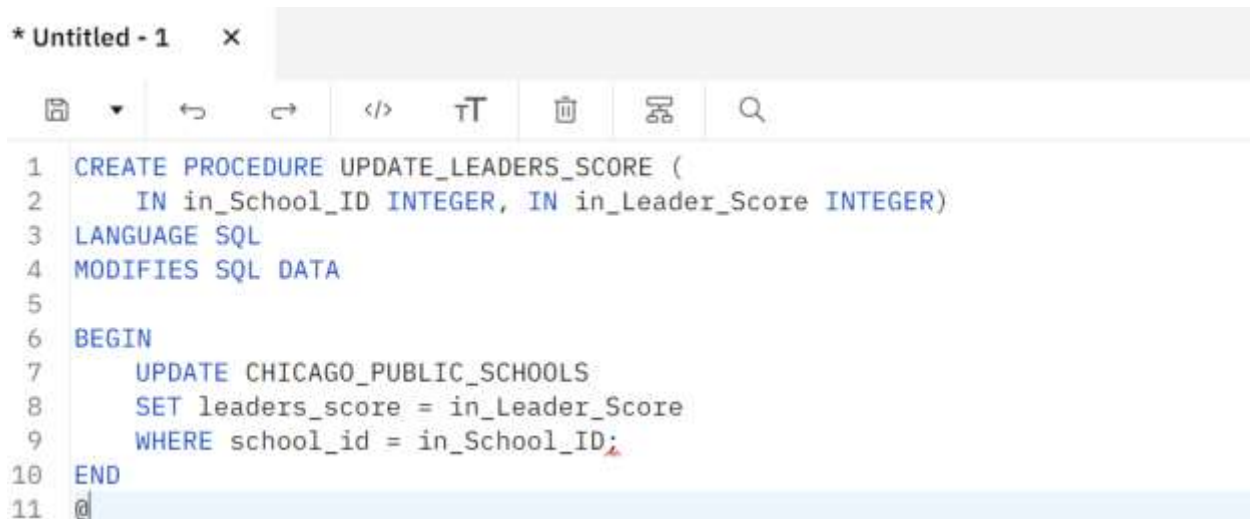


```
* Untitled - 1  X
1 CREATE PROCEDURE UPDATE_LEADERS_SCORE (
2     IN in_School_ID INTEGER, IN in_Leader_Score INTEGER)
3 LANGUAGE SQL
4 MODIFIES SQL DATA
5
```

Question 2

- Inside your stored procedure, write a SQL statement to update the Leaders_Score field in the CHICAGO_PUBLIC_SCHOOLS table for the school identified by in_School_ID to the value in the in_Leader_Score parameter.

Take a screenshot showing the SQL query.



```
* Untitled - 1  X
1 CREATE PROCEDURE UPDATE_LEADERS_SCORE (
2     IN in_School_ID INTEGER, IN in_Leader_Score INTEGER)
3 LANGUAGE SQL
4 MODIFIES SQL DATA
5
6 BEGIN
7     UPDATE CHICAGO_PUBLIC_SCHOOLS
8     SET leaders_score = in_Leader_Score
9     WHERE school_id = in_School_ID;
10 END
11 @
```

Question 3

- Inside your stored procedure, write a SQL IF statement to update the Leaders_Icon field in the CHICAGO_PUBLIC_SCHOOLS table for the school identified by in_School_ID using the following information.

Score	lower limit	Score	upper limit	Icon
80		99		Very strong
60		79		Strong
40		59		Average
20		39		Weak
0		19		Very weak

Take a screenshot showing the SQL query.

```

20 --SET TEMPORARY
21 CREATE PROCEDURE UPDATE_LEARNER_SCORE (IN p_School_ID INT, IN p_Learner_Score INT) -- Name of this stored proc
22
23 LANGUAGE SQL
24 -- Language and SQL dialect
25 MODIFIES SQL DATA
26 -- This routine will modify data
27
28
29
30
31
32
33
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
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
```

Question 4

- Run your code to create the stored procedure.

Take a screenshot showing the SQL query and its results.



- Write a query to call the stored procedure, passing a valid school ID and a leader score of 50, to check that the procedure works as expected.

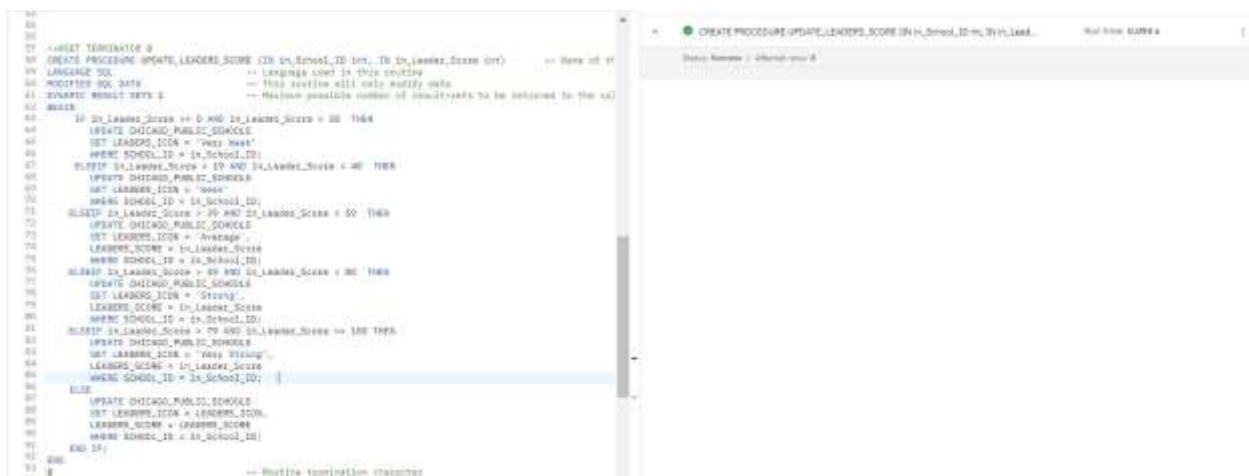
Exercise 4: Using Transactions

You realise that if someone calls your code with a score outside of the allowed range (0-99), then the score will be updated with the invalid data and the icon will remain at its previous value. There are various ways to avoid this problem, one of which is using a transaction.

Question 1

- Update your stored procedure definition. Add a generic ELSE clause to the IF statement that rolls back the current work if the score did not fit any of the preceding categories.

Take a screenshot showing the SQL query



Question 2

- Update your stored procedure definition again. Add a statement to commit the current unit of work at the end of the procedure.

Take a screenshot showing the SQL query.

```

--TEST TRANSACTION 2
--
CREATE PROCEDURE UPDATE_LEADER_SCORE (IN s_score INT IN, IN s_leader_score INT) -- Name of this stored proc
AS
BEGIN
    -- Update score in the database
    UPDATE LEADER_SCORE SET s_score = s_score WHERE s_leader_score = s_leader_score
    -- This update will only update the score
    -- Return the number of rows affected to the caller
    RETURN @@ROWCOUNT
END
GO

-- Test the stored procedure
--
DECLARE @s_score INT
DECLARE @s_leader_score INT
-- Set the score to 100
SET @s_score = 100
-- Set the leader score to 100
SET @s_leader_score = 100
-- Call the stored procedure
EXECUTE UPDATE_LEADER_SCORE @s_score, @s_leader_score
-- Return the number of rows affected
SELECT @@ROWCOUNT

```

- Run your code to replace the stored procedure.
- Write and run one query to check that the updated stored procedure works as expected when you use a valid score of 38.
- Write and run another query to check that the updated stored procedure works as expected when you use an invalid score of 101.

This was the final project for the honours part of the course 6: Databases and SQL for Data Science with Python. The code was developed by me, Saulo Villaseñor, and the challenges and questions were asked by IBM in order to complete the module. We can now write advanced SQL statements to query data from multiple tables, to obscure sensitive data from users, and to control how information is updated in our tables.