



BSc/HND Computing Portfolio
Databases and Application
Development ACCA5026

Nicolae Ciprian Panait 2311205



Part A

Guidelines:

1. Database Design (30%)

Main Entities and Attributes

Students:

-Attributes: Student_ID (PK), University_ID, First_Name, Last_Name, Email, Enrollment_Year

-Purpose: Acts as a representative for individual students who might engage in activities and join several clubs.

Clubs:

Attributes: Club_ID (PK), Club_Name, Category_ID (FK), Founding_Date, Active_Status

Purpose: The purpose is to store information about every student club or society, such as its categorisation and current state of operation.

Club Categories:

Attributes: Category_ID (PK), Category_Name, Description

Purpose: Describes classifications for clubs, such as Academic, Sports, Cultural, or Special Interest.

Memberships:

Attributes: Membership_ID (PK), Student_ID (FK), Club_ID (FK), Role, Start_Date, End_Date, Status

Purpose: The goal of the junction table is to enable many-to-many linkages between clubs and students. keeps track of membership periods and positions (such as officer and member).

Events:

Attributes: Event_ID (PK), Club_ID (FK), Title, Description, Start_Time, End_Time, Venue, Capacity, Restricted

Purpose: Symbolises club-organized events, such as meetings, workshops, or games.

Event Participation:

Attributes: Participation_ID (PK), Event_ID (FK), Student_ID (FK), RSVP_Status, Checked_In_At

Purpose: Monitors student participation in events, including attendance and degree of engagement.

Announcements:

Attributes: Announcement_ID (PK), Club_ID (FK), Subject, Body, Sent_At, Scope, Medium

Purpose: Documents official messages that groups send to their members or the general student body.

Announcement Recipients

Attributes: Recipient_ID (PK), Announcement_ID (FK), Student_ID (FK), Delivery_Method, Delivered_At

Purpose: The goal is to create an audit trail that shows which pupils received announcements and how they were delivered.

Relationships Between Entities

Students ↔ Memberships ↔ Clubs

Many-to-many relationship: A student may participate in more than one club, and each club may have more than one member.

Implemented via the Memberships table.

Clubs ↔ Club Categories

One-to-many relationship: Every club is a member of a single category, although numerous clubs may be included in each category.

Clubs ↔ Events

One-to-many relationship: A club may plan several events, but only one club owns each one.

Students ↔ Event Participation ↔ Events

Many-to-many relationship: A student may participate in more than one club, and each club may have more than one member.

Implemented via the Event Participation table.

Clubs ↔ Announcements

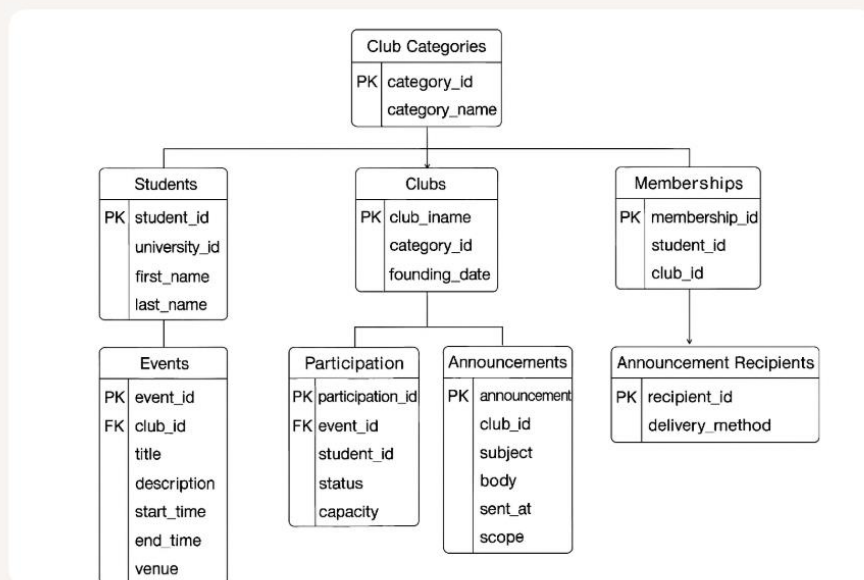
One-to-many relationship: Every club is a member of a single category, although numerous clubs may be included in each category.

Announcements ↔ Announcement Recipients ↔ Students

Many-to-many relationship: A club may plan several events, but only one club owns each one.

Implemented via the Announcement Recipients table.

Entity Relationship Diagram (ERD)



Develop the database according to the ER model by creating tables and defining constraints to ensure data integrity and proper relational links.

```
db-class-exercise"  SQL File 3"  SQL File 4"
Limit to 1000 rows

1  create database ccms2;
2  use ccms2;
3  CREATE TABLE club_categories (
4      category_id INT AUTO_INCREMENT PRIMARY KEY,
5      category_name VARCHAR(50) NOT NULL UNIQUE,
6      description TEXT
7  );
8  CREATE TABLE clubs (
9      club_id INT AUTO_INCREMENT PRIMARY KEY,
10     club_name VARCHAR(100) NOT NULL UNIQUE,
11     category_id INT NOT NULL,
12     founding_date DATE NOT NULL,
13     active BOOLEAN NOT NULL DEFAULT TRUE,
14     FOREIGN KEY (category_id) REFERENCES club_categories(category_id)
15     ON UPDATE CASCADE ON DELETE RESTRICT
16 );
17 CREATE TABLE students (
18     student_id INT AUTO_INCREMENT PRIMARY KEY,
19     university_id VARCHAR(20) NOT NULL UNIQUE,
20     first_name VARCHAR(50) NOT NULL,
21     last_name VARCHAR(50) NOT NULL,
22     email VARCHAR(120) NOT NULL UNIQUE,
23     enrollment_year SMALLINT NOT NULL
24 );
25 CREATE TABLE memberships (
26     membership_id INT AUTO_INCREMENT PRIMARY KEY,
27     student_id INT NOT NULL,
28     club_id INT NOT NULL,
29     role VARCHAR(30) NOT NULL DEFAULT 'member',
30     start_date DATE NOT NULL,
31     end_date DATE DEFAULT NULL,
32     status VARCHAR(20) NOT NULL DEFAULT 'active'
```

2.Database Implementation (35%)

Populate your tables with realistic sample data to simulate real-world operation of the system.

```
db-class-exercise SQL File 3 SQL File 4
Limit to 1000 rows
89 })
90 INSERT INTO club_categories (category_name, description) VALUES
91 ('Academic', 'Subject-focused societies and study groups'),
92 ('Sports', 'Teams and recreational sports clubs'),
93 ('Cultural', 'Cultural and international associations'),
94 ('Special Interest', 'Hobby and community groups');
95 INSERT INTO clubs (club_name, category_id, founding_date, active) VALUES
96 ('Data Science Society', 1, '2015-10-01', TRUE),
97 ('Football Club', 2, '1999-09-01', TRUE),
98 ('International Students Association', 3, '2008-02-15', TRUE),
99 ('Board Games Guild', 4, '2012-11-20', TRUE);
100 INSERT INTO students (university_id, first_name, last_name, email, enrollment_year) VALUES
101 ('U001234', 'Ana', 'Popescu', 'ana.popescu@university.edu', 2023),
102 ('U001235', 'Mihai', 'Ionescu', 'mihai.ionescu@university.edu', 2022),
103 ('U001236', 'Elena', 'Marin', 'elena.marin@university.edu', 2024),
104 ('U001237', 'Radu', 'Dumitrescu', 'radu.dumitrescu@university.edu', 2021);
105 INSERT INTO memberships (student_id, club_id, role, start_date, status) VALUES
106 (1, 1, 'member', '2023-10-05', 'active'),
107 (2, 1, 'officer', '2022-10-01', 'active'),
108 (2, 2, 'member', '2022-10-10', 'active'),
109 (3, 3, 'member', '2024-02-20', 'active'),
110 (4, 4, 'president', '2021-11-25', 'active');
111 INSERT INTO events (club_id, title, description, start_time, end_time, venue, capacity, restricted) VALUES
112 (1, 'Intro to Python', 'Beginner workshop on Python basics', '2025-12-05 10:00:00', '2025-12-05 20:00:00', 'Lab A', 40, TRUE),
113 (1, 'Data Viz Night', 'Evening of data visualization demos', '2025-12-12 18:00:00', '2025-12-12 20:30:00', 'Hall 2', 100, FALSE),
114 (2, 'Friendly Match', 'Casual football match', '2025-12-03 17:00:00', '2025-12-03 18:30:00', 'Pitch 1', 22, TRUE),
115 (4, 'Board Game Marathon', 'Open event for board game enthusiasts', '2025-12-10 16:00:00', '2025-12-10 22:00:00', 'Student Union Hall', 80, FALSE);
116 INSERT INTO event_participation (event_id, student_id, rsvp_status, checked_in_at) VALUES
117 (1, 1, 'going', NULL),
118 (1, 2, 'going', NULL),
119 (1, 3, 'interested', NULL),
120 (2, 2, 'notgoing', NULL);
Output
Action Output
# Time Action Message
78 11:12:50 INSERT INTO event_participation (event_id, student_id, rsvp_status, checked_in_at) VALUES (1, 1, 'going', NULL), (1, 2, 'going', NULL), (2, 1, 'interest... 5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0
79 11:12:59 INSERT INTO announcements (club_id, subject, body, scope, medium) VALUES (1, 'Workshop Python', 'Registration open for Python workshop.', 'all... 3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0
80 11:13:16 INSERT INTO announcements (club_id, subject, body, scope, medium) VALUES (1, 'Workshop Python', 'Registration open for Python workshop.', 'all... 3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0
81 11:13:26 INSERT INTO announcement_recipients (announcement_id, student_id, delivery_method, delivered_at) VALUES (1, 1, 'email', '2025-11-28 10:00:00') (... 4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0
```

Use this data to test and validate the database design.

Import/Restore

Up / Shutdown

Logs

File

NCE

board

formance Reports

rmance Schema Setup

```
120 (3, 2, 'going', NULL),
121 (4, 4, 'going', '2025-12-10 16:10:00');
122 INSERT INTO announcements (club_id, subject, body, scope, medium) VALUES
123 (1, 'Workshop Python', 'Registration open for Python workshop.', 'all_members', 'email'),
124 (2, 'Team Selections', 'Looking for players for Friday match.', 'all_members', 'portal'),
125 (4, 'Board Game Marathon', 'Event open to all students.', 'public', 'social');
126 INSERT INTO announcement_recipients (announcement_id, student_id, delivery_method, delivered_at) VALUES
127 (1, 1, 'email', '2025-11-28 10:00:00'),
128 (1, 2, 'email', '2025-11-28 10:01:00'),
129 (2, 2, 'portal', '2025-11-28 10:02:00'),
130 (3, 4, 'social', '2025-11-28 10:03:00');
131 SELECT s.first_name, s.last_name, c.club_name, m.role
132 FROM memberships m
133 JOIN students s ON s.student_id = m.student_id
134 JOIN clubs c ON c.club_id = m.club_id;
```

Result Grid

Filter Rows

Export

Wrap Cell Contents

first_name	last_name	club_name	role
Ana	Popescu	Data Science Society	member
Mihai	Ionescu	Data Science Society	officer
Mihai	Ionescu	Football Club	member
Elena	Marin	International Students Association	member
Radu	Dumitrescu	Board Games Guild	president

Result 3

Output

Action Output

Time Action Message

79 11:12:59 INSERT INTO announcements (club_id, subject, body, scope, medium) VALUES (1, 'Workshop Python', 'Registration open for Python workshop.', 'all... 3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0

80 11:13:16 INSERT INTO announcements (club_id, subject, body, scope, medium) VALUES (1, 'Workshop Python', 'Registration open for Python workshop.', 'all... 3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0

81 11:13:26 INSERT INTO announcement_recipients (announcement_id, student_id, delivery_method, delivered_at) VALUES (1, 1, 'email', '2025-11-28 10:00:00') (... 4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0

82 11:16:05 SELECT s.first_name, s.last_name, c.club_name, m.role FROM memberships m JOIN students s ON s.student_id = m.student_id JOIN clubs c ON c.clu... 5 row(s) returned

hutdown
s
e

```

127 (1, 1, 'email', '2025-11-28 10:00:00'),
128 (1, 2, 'email', '2025-11-28 10:01:00'),
129 (2, 2, 'portal', '2025-11-28 10:02:00'),
130 (3, 4, 'social', '2025-11-28 10:03:00');
131 • SELECT s.first_name, s.last_name, c.club_name, m.role
132 FROM memberships m
133 JOIN students s ON s.student_id = m.student_id
134 JOIN clubs c ON c.club_id = m.club_id;
135 -- List upcoming events with club names
136 • SELECT e.title, e.start_time, e.venue, c.club_name
137 FROM events e
138 JOIN clubs c ON c.club_id = e.club_id
139 WHERE e.start_time > NOW()
140 ORDER BY e.start_time;

```

ce Reports
ce Schema Setup

Schemas

title	start_time	venue	club_name
Friendly Match	2025-12-03 17:00:00	Pitch 1	Football Club
Intro to Python	2025-12-05 18:00:00	Lab A	Data Science Society
Board Game Marathon	2025-12-10 16:00:00	Student Union Hall	Board Games Guild
Data Viz Night	2025-12-12 18:00:00	Hall 2	Data Science Society

electd

Result 4 x

Output

Action Output

#	Time	Action	Message
80	11:13:16	INSERT INTO announcement_recipients (announcement_id, student_id, delivery_method, delivered_at) VALUES (1, 1, 'email', '2025-11-28 10:00:00'), (... 3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0	
81	11:13:26	INSERT INTO announcement_recipients (announcement_id, student_id, delivery_method, delivered_at) VALUES (1, 1, 'email', '2025-11-28 10:00:00'), (... 4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0	

Implement stored procedures for CRUD operations with embedded transaction handling (BEGIN, COMMIT, ROLLBACK).

BEGIN:

```

152
153 • CREATE PROCEDURE add_student (
154     IN p_university_id VARCHAR(20),
155     IN p_first_name VARCHAR(50),
156     IN p_last_name VARCHAR(50),
157     IN p_email VARCHAR(120),
158     IN p_enrollment_year SMALLINT
159 )
160 BEGIN
161     DECLARE EXIT HANDLER FOR SQLEXCEPTION
162     BEGIN
163         ROLLBACK;
164     END;
165
166     START TRANSACTION;
167
168     INSERT INTO students (university_id, first_name, last_name, email, enrollment_year)
169     VALUES (p_university_id, p_first_name, p_last_name, p_email, p_enrollment_year);
170
171     COMMIT;
172 END$$
173
174 DELIMITER ;
175 DELIMITER $$
176
177 • CREATE PROCEDURE get_student (
178     IN p_student_id INT
179 )
180 BEGIN
181     SELECT student_id, university_id, first_name, last_name, email, enrollment_year
182     FROM students
183     WHERE student_id = p_student_id;

```

Output

Action Output

COMMIT:

The screenshot displays a SQL IDE interface with a script editor and an output pane. The script contains SQL procedures for managing students and events.

```
170 COMMIT;
171 END$$
172
173
174 DELIMITER ;
175 DELIMITER $$
176
177 CREATE PROCEDURE get_student (
178     IN p_student_id INT
179 )
180 BEGIN
181     SELECT student_id, university_id, first_name, last_name, email, enrollment_year
182     FROM students
183     WHERE student_id = p_student_id;
184 END$$
185
186 DELIMITER ;
187 DELIMITER $$
188
189 CREATE PROCEDURE update_student_email (
190     IN p_student_id INT,
191     IN p_new_email VARCHAR(120)
192 )
193 BEGIN
194     DECLARE EXIT HANDLER FOR SQLEXCEPTION
195     BEGIN
196         ROLLBACK;
197     END;
198
199     START TRANSACTION;
200
201     UPDATE students
```

The output pane shows the execution results of the script:

#	Time	Action	Message
94	11:24:09	CALL delete_student(4)	0 row(s) affected
95	11:24:14	CALL add_student('U001230', 'Ioana', 'Georgescu', 'ioana.georgescu@university.edu', 2025)	0 row(s) affected
96	11:24:17	CALL get_student(1)	1 row(s) returned
97	11:24:27	CALL delete_student(4)	0 row(s) affected

Below the first output, another SQL query is shown in the script editor:

```
136 FROM events e
137 JOIN clubs c ON c.club_id = e.club_id
138 WHERE e.start_time > NOW()
139 ORDER BY e.start_time;
140 SELECT e.title,
141     SUM(CASE WHEN p.rsvp_status = 'going' THEN 1 ELSE 0 END) AS going,
142     SUM(CASE WHEN p.rsvp_status = 'interested' THEN 1 ELSE 0 END) AS interested
143 FROM events e
144 JOIN event_participation p ON p.event_id = e.event_id
145 WHERE e.event_id = 1
146 GROUP BY e.title;
147 SELECT a.subject, a.scope, COUNT(r.recipient_id) AS recipients
148 FROM announcements a
149 LEFT JOIN announcement_recipients r ON r.announcement_id = a.announcement_id
150 GROUP BY a.announcement_id;
```

The output pane shows the results of this query in a table format:

subject	scope	recipients
Workshop Python	all_members	2
Team Selections	all_members	1
Board Game Marathon	public	1
Workshop Python	all_members	0
Team Selections	all_members	0
Board Game Marathon	public	0

Below the table, the output pane shows the execution results of the script:

#	Time	Action	Message
82	11:16:05	SELECT s.first_name, s.last_name, c.club_name, m.role FROM memberships m JOIN students s ON s.student_id = m.student_id JOIN clubs c ON c.club_id = m.club_id	5 row(s)
83	11:17:17	SELECT e.title, e.start_time, e.venue, c.club_name FROM events e JOIN clubs c ON c.club_id = e.club_id WHERE e.start_time > NOW() ORDER BY ...	4 row(s)
84	11:18:02	SELECT e.title, SUM(CASE WHEN p.rsvp_status = 'going' THEN 1 ELSE 0 END) AS going, SUM(CASE WHEN p.rsvp_status = 'interested' T...	1 row(s)
85	11:19:23	SELECT a.subject, a.scope, COUNT(r.recipient_id) AS recipients FROM announcements a LEFT JOIN announcement_recipients r ON r.announcement...	6 row(s)

ROLLBACK

The screenshot shows a SQL IDE with a script editor and an output window. The script editor contains the following SQL code:

```
203 WHERE student_id = p_student_id;
204
205 COMMIT;
206 END$$
207
208 DELIMITER ;
209 DELIMITER $$
210
211 CREATE PROCEDURE delete_student (
212     IN p_student_id INT
213 )
214 BEGIN
215     DECLARE EXIT HANDLER FOR SQLEXCEPTION
216     BEGIN
217         ROLLBACK;
218     END;
219
220     START TRANSACTION;
221
222     DELETE FROM students
223     WHERE student_id = p_student_id;
224
225     COMMIT;
226 END$$
227
228 DELIMITER ;
229 CALL add_student('U001238', 'Ioana', 'Georgescu', 'ioana.georgescu@university.edu', 2025);
230
231 CALL get_student(1);
232
233 CALL update_student_email(1, 'ana.popescu.new@university.edu');
```

The output window shows the following results:

#	Time	Action	Message
94	11:24:09	CALL delete_student(4)	0 row(s) affected
95	11:24:14	CALL add_student('U001238', 'Ioana', 'Georgescu', 'ioana.georgescu@university.edu', 2025)	0 row(s) affected
96	11:24:17	CALL get_student(1)	1 row(s) returned
97	11:24:27	CALL delete_student(4)	0 row(s) affected

Testing operation:

```

227
228 DELIMITER ;
229 • CALL add_student('U001238', 'Ioana', 'Georgescu', 'ioana.georgescu@university.edu', 2025);
230
231 • CALL get_student(1);
232
233 • CALL update_student_email(1, 'ana.popescu.new@university.edu');
234
235 • CALL delete_student(4);
236
237
238
239
240

```

Output				
Action Output				
#	Time	Action	Message	
✓ 91	11:23:54	CALL add_student('U001238', 'Ioana', 'Georgescu', 'ioana.georgescu@university.edu', 2025)	0 row(s) affected	
✓ 92	11:23:58	CALL get_student(1)	1 row(s) returned	
✓ 93	11:24:06	CALL update_student_email(1, 'ana.popescu.new@university.edu')	0 row(s) affected	
✓ 94	11:24:09	CALL delete_student(4)	0 row(s) affected	
✓ 95	11:24:14	CALL add_student('U001238', 'Ioana', 'Georgescu', 'ioana.georgescu@university.edu', 2025)	0 row(s) affected	
✓ 96	11:24:17	CALL get_student(1)	1 row(s) returned	
✓ 97	11:24:27	CALL delete_student(4)	0 row(s) affected	
✓ 98	11:28:22	CALL get_student(1)	1 row(s) returned	
✓ 99	11:29:14	CALL add_student('U001238', 'Ioana', 'Georgescu', 'ioana.georgescu@university.edu', 2025)	0 row(s) affected	
✓ 100	11:29:16	CALL get_student(1)	1 row(s) returned	
✓ 101	11:29:19	CALL update_student_email(1, 'ana.popescu.new@university.edu')	0 row(s) affected	
✓ 102	11:29:22	CALL delete_student(4)	0 row(s) affected	

Create triggers and functions to implement auditing features that automatically track changes to your data.

Create trigger:

```

234
235 * CALL delete_student();
236 * CREATE TABLE audit_log (
237     audit_id INT AUTO INCREMENT PRIMARY KEY,
238     table_name VARCHAR(50) NOT NULL,
239     operation VARCHAR(10) NOT NULL,
240     record_id INT NOT NULL,
241     changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
242     old_data TEXT,
243     new_data TEXT
244 );
245 DELIMITER $$
246
247 * CREATE TRIGGER trg_students_insert
248 AFTER INSERT ON students
249 FOR EACH ROW
250 BEGIN
251     INSERT INTO audit_log (table_name, operation, record_id, new_data)
252     VALUES ('students', 'INSERT', NEW.student_id,
253             CONCAT('Name=', NEW.first_name, ' ', NEW.last_name, ', Email=', NEW.email));
254     END$$
255
256 DELIMITER ;
257 DELIMITER $$
258
259 * CREATE TRIGGER trg_students_update
260 AFTER UPDATE ON students
261 FOR EACH ROW
262 BEGIN
263     INSERT INTO audit_log (table_name, operation, record_id, old_data, new_data)
264     VALUES ('students', 'UPDATE', NEW.student_id,
265             CONCAT('Old Email=', OLD.email),
266             CONCAT('New Email=', NEW.email));
267     END$$
268
269 DELIMITER ;
270 DELIMITER $$
271
272 * CREATE TRIGGER trg_students_delete
273 AFTER DELETE ON students
274 FOR EACH ROW
275 BEGIN
276     INSERT INTO audit_log (table_name, operation, record_id, old_data)
277     VALUES ('students', 'DELETE', OLD.student_id,
278             CONCAT('Deleted Student=', OLD.first_name, ' ', OLD.last_name));
279     END$$
280
281 DELIMITER ;
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

```

#	Time	Action	Message
104	11:32:12	CREATE TRIGGER trg_students_insert AFTER INSERT ON students FOR EACH ROW BEGIN INSERT INTO audit_log (table_name, operation, record_id, new_data) VALUES ('students', 'INSERT', NEW.student_id, CONCAT('Name=', NEW.first_name, ' ', NEW.last_name, ', Email=', NEW.email)); END\$\$	0 row(s) affected
105	11:32:47	CREATE TRIGGER trg_students_update AFTER UPDATE ON students FOR EACH ROW BEGIN INSERT INTO audit_log (table_name, operation, record_id, old_data, new_data) VALUES ('students', 'UPDATE', NEW.student_id, CONCAT('Old Email=', OLD.email), CONCAT('New Email=', NEW.email)); END\$\$	0 row(s) affected
106	11:33:15	CREATE TRIGGER trg_students_update AFTER UPDATE ON students FOR EACH ROW BEGIN INSERT INTO audit_log (table_name, operation, record_id, old_data, new_data) VALUES ('students', 'UPDATE', NEW.student_id, CONCAT('Old Email=', OLD.email), CONCAT('New Email=', NEW.email)); END\$\$	Error Code: 1359. Trigger already exists
107	11:33:25	CREATE TRIGGER trg_students_delete AFTER DELETE ON students FOR EACH ROW BEGIN INSERT INTO audit_log (table_name, operation, record_id, old_data) VALUES ('students', 'DELETE', OLD.student_id, CONCAT('Deleted Student=', OLD.first_name, ' ', OLD.last_name)); END\$\$	0 row(s) affected

Delete trigger:

```

3     INSERT INTO audit_log (table_name, operation, record_id, old_data, new_data)
4     VALUES ('students', 'UPDATE', NEW.student_id,
5             CONCAT('Old Email=', OLD.email),
6             CONCAT('New Email=', NEW.email));
7     END$$
8
9     DELIMITER ;
10    DELIMITER $$
11
12 * CREATE TRIGGER trg_students_delete
13 AFTER DELETE ON students
14 FOR EACH ROW
15 BEGIN
16     INSERT INTO audit_log (table_name, operation, record_id, old_data)
17     VALUES ('students', 'DELETE', OLD.student_id,
18             CONCAT('Deleted Student=', OLD.first_name, ' ', OLD.last_name));
19     END$$
20
21    DELIMITER ;
22
23
24
25
26
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

```

put

#	Time	Action	Message
104	11:32:12	CREATE TRIGGER trg_students_insert AFTER INSERT ON students FOR EACH ROW BEGIN INSERT INTO audit_log (table_name, operation, record_id, new_data) VALUES ('students', 'INSERT', NEW.student_id, CONCAT('Name=', NEW.first_name, ' ', NEW.last_name, ', Email=', NEW.email)); END\$\$	0 row(s) affected
105	11:32:47	CREATE TRIGGER trg_students_update AFTER UPDATE ON students FOR EACH ROW BEGIN INSERT INTO audit_log (table_name, operation, record_id, old_data, new_data) VALUES ('students', 'UPDATE', NEW.student_id, CONCAT('Old Email=', OLD.email), CONCAT('New Email=', NEW.email)); END\$\$	0 row(s) affected
106	11:33:15	CREATE TRIGGER trg_students_update AFTER UPDATE ON students FOR EACH ROW BEGIN INSERT INTO audit_log (table_name, operation, record_id, old_data, new_data) VALUES ('students', 'UPDATE', NEW.student_id, CONCAT('Old Email=', OLD.email), CONCAT('New Email=', NEW.email)); END\$\$	Error Code: 1359. Trigger already exists
107	11:33:25	CREATE TRIGGER trg_students_delete AFTER DELETE ON students FOR EACH ROW BEGIN INSERT INTO audit_log (table_name, operation, record_id, old_data) VALUES ('students', 'DELETE', OLD.student_id, CONCAT('Deleted Student=', OLD.first_name, ' ', OLD.last_name)); END\$\$	0 row(s) affected

Insert/delete:

Testing trigger:

```

297 FOR EACH ROW
298 BEGIN
299     INSERT INTO audit_log (table_name, operation, record_id, new_data)
300     VALUES ('students', 'INSERT', NEW.student_id,
301             format_student_data(NEW.student_id, NEW.first_name, NEW.last_name, NEW.email));
302 END$$
303
304 DELIMITER ;
305 • INSERT INTO students (university_id, first_name, last_name, email, enrollment_year)
306 VALUES ('U001241', 'Cristina', 'Popa', 'cristina.popa@university.edu', 2025);
307
308 • SELECT * FROM audit_log ORDER BY changed_at DESC;
309 DELIMITER $$
310 • DELETE FROM students WHERE university_id = 'U001240';
311 SELECT * FROM audit_log ORDER BY changed_at DESC;
312 INSERT INTO students (university_id, first_name, last_name, email, enrollment_year)
313 VALUES ('U001241', 'Cristina', 'Popa', 'cristina.popa@university.edu', 2025);
314
315 SELECT * FROM audit_log ORDER BY changed_at DESC;
316

```

audit_id	table_name	operation	record_id	changed_at	old_data	new_data
3	students	INSERT	17	2025-11-28 11:43:34	NULL	Name=Cristina Popa, Email=cristina.popa@univ...
2	students	DELETE	12	2025-11-28 11:42:32	Deleted Student=George Matei	NULL
1	students	INSERT	12	2025-11-28 11:40:40	NULL	Name=George Matei, Email=george.matei@uni...

Result 13 Result 14 x

Output

#	Time	Action	Message
121	11:42:40	DELETE FROM students WHERE university_id = 'U001240'; SELECT * FROM audit_log ORDER BY changed_at DESC;	2 row(s) returned
122	11:43:34	DELETE FROM students WHERE university_id = 'U001240'; SELECT * FROM audit_log ORDER BY changed_at DESC; INSERT INTO students (univ...	0 row(s) affected
123	11:43:34	DELETE FROM students WHERE university_id = 'U001240'; SELECT * FROM audit_log ORDER BY changed_at DESC; INSERT INTO students (univ...	2 row(s) returned
124	11:43:34	DELETE FROM students WHERE university_id = 'U001240'; SELECT * FROM audit_log ORDER BY changed_at DESC; INSERT INTO students (univ...	3 row(s) returned

Design a transactional scenario using stored procedures that:

- Includes multiple related operations executed as a single transaction.
- Uses rollback to undo all operations if any step fails.
- Provides a clear justification for commit or rollback decisions to ensure data integrity.

```

1316 DELIMITER $$
1317
1318 CREATE PROCEDURE register_student_for_event (
1319     IN p_student_id INT,
1320     IN p_event_id INT
1321 )
1322 BEGIN
1323     DECLARE v_club_id INT;
1324     DECLARE v_capacity INT;
1325
1326     DECLARE EXIT HANDLER FOR SQLSTATE
1327     BEGIN
1328         ROLLBACK;
1329     END;
1330
1331     START TRANSACTION;
1332
1333     SELECT club_id, capacity INTO v_club_id, v_capacity
1334     FROM events
1335     WHERE event_id = p_event_id
1336     FOR UPDATE;
1337
1338
1339     IF NOT EXISTS (
1340         SELECT 1 FROM memberships
1341         WHERE student_id = p_student_id AND club_id = v_club_id AND status = 'active'
1342     ) THEN
1343         SIGNAL SQLSTATE '45000'
1344         SET MESSAGE_TEXT = 'Student is not a member of the hosting club';
1345     END IF;
1346
1347     IF v_capacity <= 0 THEN
1348
1349
1350
1351
1352     INSERT INTO event_participation (event_id, student_id, rsvp_status)
1353     VALUES (p_event_id, p_student_id, 'going');
1354
1355     UPDATE events
1356     SET capacity = capacity - 1
1357     WHERE event_id = p_event_id;
1358
1359     COMMIT;
1360 END$$
1361
1362 DELIMITER ;
1363 CALL register_student_for_event(1, 1);
1364 CALL register_student_for_event(3, 1);
1365 CALL register_student_for_event(2, 3);
1366
1367
1368
1369
1370
1371
1372
1373
1374

```

Output

#	Time	Action	Message
122	11:43:34	DELETE FROM students WHERE university_id = 'U001240'; SELECT * FROM audit_log ORDER BY changed_at DESC; INSERT INTO students (univ...	0 row(s) affected
123	11:43:34	DELETE FROM students WHERE university_id = 'U001240'; SELECT * FROM audit_log ORDER BY changed_at DESC; INSERT INTO students (univ...	2 row(s) returned
124	11:43:34	DELETE FROM students WHERE university_id = 'U001240'; SELECT * FROM audit_log ORDER BY changed_at DESC; INSERT INTO students (univ...	3 row(s) returned
125	11:46:40	CREATE PROCEDURE register_student_for_event (IN p_student_id INT, IN p_event_id INT) BEGIN DECLARE v_club_id INT; DECLARE ...	0 row(s) affected

```

345     END IF;
346
347     IF v_capacity <= 0 THEN
348         SIGNAL SQLSTATE '45000'
349         SET MESSAGE_TEXT = 'Event is full';
350     END IF;
351
352     INSERT INTO event_participation (event_id, student_id, rsvp_status)
353     VALUES (p_event_id, p_student_id, 'going');
354
355     UPDATE events
356     SET capacity = capacity - 1
357     WHERE event_id = p_event_id;
358
359     COMMIT;
360 END$$
361
362 DELIMITER ;
363 CALL register_student_for_event(1, 1);
364 CALL register_student_for_event(3, 1);
365 CALL register_student_for_event(2, 3);
366
367
368
369
370
371
372
373
374

```

Output

#	Time	Action	Message
125	11:46:40	CREATE PROCEDURE register_student_for_event (IN p_student_id INT, IN p_event_id INT) BEGIN DECLARE v_club_id INT; DECLARE ...	0 row(s) affected
126	11:47:43	CALL register_student_for_event(1, 1)	0 row(s) affected
127	11:47:59	CALL register_student_for_event(3, 1)	0 row(s) affected
128	11:48:12	CALL register_student_for_event(2, 3)	0 row(s) affected

Justification for Commit/Rollback:

Commit: Only if all requirements are met, such as the student being a legitimate member,

the event having enough space, and the successful insertion of the participation record.

Rollback: In the event that a condition is not met, rollback guarantees that no partial changes occur (e.g., capacity cut but no participation record). This avoids conflicting states and maintains referential integrity.

Identify key queries or operations that are frequent or critical to business performance.

```
363 • CALL register_student_for_event(1, 1);
364 • CALL register_student_for_event(3, 1);
365 • CALL register_student_for_event(2, 3);
366 • SELECT s.first_name, s.last_name, m.role
367 FROM memberships m
368 JOIN students s ON s.student_id = m.student_id
369 WHERE m.club_id = 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	first_name	last_name	role
▶	Ana	Popescu	member
▶	Mihai	Ionescu	officer

Result 15 ×

Output

Action Output

#	Time	Action
✓ 126	11:47:43	CALL register_student_for_event(1, 1)
✓ 127	11:47:59	CALL register_student_for_event(3, 1)
✓ 128	11:48:12	CALL register_student_for_event(2, 3)
✓ 129	11:51:47	SELECT s.first_name, s.last_name, m.role FROM memberships m JOIN students s ON s.student_id = m.student_id WHERE m.club_id = 1 LIMIT 0, 1

ip

```
377 GROUP BY e.title;
378 • SELECT c.club_name, COUNT(m.student_id) AS total_members
379 FROM clubs c
380 LEFT JOIN memberships m ON c.club_id = m.club_id AND m.status = 'active'
381 GROUP BY c.club_name
382 ORDER BY total_members DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	club_name	total_members
▶	Data Science Society	2
▶	Football Club	1
▶	International Students Association	1
▶	Board Games Guild	0

Result 17 ×

```

379 FROM clubs c
380 LEFT JOIN memberships m ON c.club_id = m.club_id AND m.status = 'active'
381 GROUP BY c.club_name
382 ORDER BY total_members DESC;
383 • SELECT a.subject, COUNT(r.recipient_id) AS recipients
384 FROM announcements a
385 LEFT JOIN announcement_recipients r ON a.announcement_id = r.announcement_id
386 GROUP BY a.announcement_id;
387

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

subject	recipients
Workshop Python	2
Team Selections	1
Board Game Marathon	0
Workshop Python	0
Team Selections	0
Board Game Marathon	0

Result 18 x

Output

Action Output

#	Time	Action	Message
✓ 129	11:51:47	SELECT s.first_name, s.last_name, m.role FROM memberships m JOIN students s ON s.student_id = m.student_id WHERE m.club_id = 1 LIMIT 0, 1000	2 row(s) return
✓ 130	11:52:52	SELECT e.title, SUM(CASE WHEN p.rsvp_status = 'going' THEN 1 ELSE 0 END) AS going, SUM(CASE WHEN p.rsvp_status = 'interested' THEN 1 ELSE 0 END) AS interested FROM event_participation p JOIN events e ON e.event_id = p.event_id	1 row(s) return
✓ 131	11:53:03	SELECT c.club_name, COUNT(m.student_id) AS total_members FROM clubs c LEFT JOIN memberships m ON c.club_id = m.club_id AND m.status = 'active'	4 row(s) return
✓ 132	11:53:49	SELECT a.subject, COUNT(r.recipient_id) AS recipients FROM announcements a LEFT JOIN announcement_recipients r ON a.announcement_id = r.announcement_id	6 row(s) return

```

382 ORDER BY total_members DESC;
383 • SELECT a.subject, COUNT(r.recipient_id) AS recipients
384 FROM announcements a
385 LEFT JOIN announcement_recipients r ON a.announcement_id = r.announcement_id
386 GROUP BY a.announcement_id;
387 • SELECT s.first_name, s.last_name, e.title, e.start_time
388 FROM event_participation p
389 JOIN students s ON s.student_id = p.student_id
390 JOIN events e ON e.event_id = p.event_id
391 WHERE s.student_id = 2;
392

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

first_name	last_name	title	start_time
Mihai	Ionescu	Intro to Python	2025-12-05 18:00:00
Mihai	Ionescu	Friendly Match	2025-12-03 17:00:00

Result 19 x

Output

Action Output

#	Time	Action	Message
✓ 130	11:52:52	SELECT e.title, SUM(CASE WHEN p.rsvp_status = 'going' THEN 1 ELSE 0 END) AS going, SUM(CASE WHEN p.rsvp_status = 'interested' THEN 1 ELSE 0 END) AS interested FROM event_participation p JOIN events e ON e.event_id = p.event_id	1 row(s) return
✓ 131	11:53:03	SELECT c.club_name, COUNT(m.student_id) AS total_members FROM clubs c LEFT JOIN memberships m ON c.club_id = m.club_id AND m.status = 'active'	4 row(s) return
✓ 132	11:53:49	SELECT a.subject, COUNT(r.recipient_id) AS recipients FROM announcements a LEFT JOIN announcement_recipients r ON a.announcement_id = r.announcement_id	6 row(s) return
✓ 133	11:54:20	SELECT s.first_name, s.last_name, e.title, e.start_time FROM event_participation p JOIN students s ON s.student_id = p.student_id JOIN events e ON e.event_id = p.event_id	2 row(s) return

```

395 • SELECT table_name, operation, record_id, changed_at, old_data, new_data
396 FROM audit_log
397 ORDER BY changed_at DESC
398 LIMIT 10;
399

```

	table_name	operation	record_id	changed_at	old_data	new_data
▶	students	INSERT	17	2025-11-28 11:43:34	NULL	Name=Cristina Popa, Email=cristina.popa@univ...
	students	DELETE	12	2025-11-28 11:42:32	Deleted Student=George Matei	NULL
	students	INSERT	12	2025-11-28 11:40:40	NULL	Name=George Matei, Email=george.matei@uni...

audit_log 21 x

Output

#	Time	Action	Message
✓	132 11:53:49	SELECT a.subject, COUNT(r.recipient_id) AS recipients FROM announcements a LEFT JOIN announcement_recipients r ON a.announcement_id = r.a...	6 row(s) returned
✓	133 11:54:20	SELECT s.first_name, s.last_name, e.title, e.start_time FROM event_participation p JOIN students s ON s.student_id = p.student_id JOIN events e ON ...	2 row(s) returned
✓	134 11:54:41	SELECT title, venue, capacity FROM events WHERE capacity <= 5 LIMIT 0, 1000	0 row(s) returned
✓	135 11:54:51	SELECT table_name, operation, record_id, changed_at, old_data, new_data FROM audit_log ORDER BY changed_at DESC LIMIT 10	3 row(s) returned

Performance Tuning Enhancement: Query Optimization

Create secondary indexes on relevant columns to improve query performance.
Justify index choices based on expected query patterns and usage scenarios.
Discuss the query speed and overall system efficiency resulting from indexing.

Create index:

```

397 ORDER BY changed_at DESC
398 LIMIT 10;
399
400 • CREATE INDEX idx_students_university_id ON students(university_id);
401 • CREATE INDEX idx_students_email ON students(email);
402
403
404 • CREATE INDEX idx_memberships_student ON memberships(student_id);
405 • CREATE INDEX idx_memberships_club ON memberships(club_id);
406
407
408 • CREATE INDEX idx_clubs_category ON clubs(category_id);
409
410
411 • CREATE INDEX idx_events_club ON events(club_id);
412 • CREATE INDEX idx_events_start_time ON events(start_time);
413
414
415 • CREATE INDEX idx_participation_event ON event_participation(event_id);
416 • CREATE INDEX idx_participation_student ON event_participation(student_id);
417
418
419 • CREATE INDEX idx_announcements_club ON announcements(club_id);
420
421
422 • CREATE INDEX idx_recipients_announcement ON announcement_recipients(announcement_id);
423 • CREATE INDEX idx_recipients_student ON announcement_recipients(student_id);
424
425
426
427

```

Output

#	Time	Action	Message
✓	145 11:57:19	CREATE INDEX idx_announcements_club ON announcements(club_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
✗	146 11:57:22	CREATE INDEX idx_announcements_club ON announcements(club_id)	Error Code: 1061. Duplicate key name 'idx_announcements_cl'
✓	147 11:57:26	CREATE INDEX idx_recipients_announcement ON announcement_recipients(announcement_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
✓	148 11:57:28	CREATE INDEX idx_recipients_student ON announcement_recipients(student_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

Justification:

Students (university_id, email)

These are unique identifiers often used in lookups (e.g., login, search by email).

Indexing them speeds up authentication and student record retrieval.

Memberships (student_id, club_id)

Critical for joins when listing members of a club or clubs a student belongs to.

Indexes reduce join cost between students and clubs.

Clubs (category_id)

Clubs are often grouped or filtered by category (e.g., Sports, Academic).

Indexing improves queries like “show all sports clubs.”

Events (club_id, start_time)

Queries often filter by club or upcoming events.

Indexing start_time accelerates chronological queries (e.g., “next week’s events”).

Impact: Prior to indexing: Full table scans are necessary for queries involving joins across large databases (students, memberships, events), and these scans become sluggish as data volume increases.

Searches by student_id, club_id, or event_id become almost instantaneous after indexing.

By using indexed columns, joins can reduce execution times from seconds to milliseconds.

Scanning unnecessary rows is prevented by filtering by start_time or category_id.

Total Effectiveness:

Indexes greatly speed up query response times.

They increase the system's scalability, enabling it to manage thousands of students, clubs, and activities without experiencing a decline in performance.

Since read performance is the top concern in reporting and analytics, the trade-off of slightly more storage and slower inserts/updates (due to the need to maintain indexes) is acceptable.

3.Literature Support for Design and Implementation Choices (15%)

Provide references from academic and peer-reviewed literature that support your design and implementation decisions.

Literature to Back Up Your Design

1. 3NF normalisation

Kolahi & Libkin (Edinburgh/University of Toronto): According to their research, Third Normal Form (3NF) offers the optimal balance between removing redundancy and maintaining interdependence, guaranteeing integrity without being overly complicated.

Mbangata & Singh (2025, Springer): Stress that in relational databases, normalisation (1NF–3NF) is essential for lowering redundancy and enhancing data integrity.

Amato (2023, ResearchGate): Offers a thorough examination of normal forms, emphasising how 3NF guarantees dependency maintenance and atomic values.

Justification: Your 3NF schema design is in line with scholarly agreement that this form strikes a compromise between integrity and efficiency. [1]

2. Optimisation of queries and indexing

Anchlia (2024, IJCTT): Shows how indexing is an essential strategy for improving query performance and drastically cutting execution time.

Holubinka & Khudiyi (2024, Lviv Polytechnic): Examine indexing strategies and demonstrate how they affect the effectiveness of queries in big relational systems.

Khandal (2024, IJFANS) compares hash and B-tree indexes and demonstrates how well they speed up lookups and joins.

Justification: These results are directly reflected in your decision to establish secondary indexes on `student_id`, `club_id`, `event_id`, and `start_time`, guaranteeing scalability and quick query response. [2]

3. Handling Transactions (Commit/Rollback)

Springer (Transaction Rollback and Restart Recovery): Describes how rollback maintains ACID features while ensuring consistency by reversing unfinished transactions.

Rollback is crucial to avoiding partial updates and preserving integrity in transactional systems, according to Bhuyan (2023, DEV Community scholarly synthesis).

According to a systematic evaluation of distributed transaction management, commit/rollback techniques are essential for reliability (Lungu & Nyirenda, 2024, Open Journal of Applied

Sciences).

Rationale: Your stored procedures with COMMIT, ROLLBACK, and START TRANSACTION adhere to atomicity and consistency best practices.

4.Using Triggers for Auditing

Li et al. (2025, Springer): Demonstrate how database triggers may create a systematic audit trail by automatically recording and analysing operation logs.

The review of database security procedures by Omotunde et al. (2023, ResearchGate) highlights auditing as a crucial element for accountability and compliance.

Rationale: Your student audit triggers (INSERT, UPDATE, DELETE) are in line with research that suggests automated logging for traceability and transparency.

Justify critical database concepts relevant to your solution, including rollback, commit, triggers, stored procedures, transactions, and indexing, citing appropriate sources.

Literature Support for Critical Database Concepts

1. Commit, Rollback, and Transactions

Atomicity, consistency, isolation, and durability (ACID), the cornerstone of dependable database systems, are guaranteed by transactions.

Grey & Reuter (1993, Transaction Processing: Concepts and Techniques): This classic publication explains that while commit completes completed transactions, rollback is necessary to undo incomplete operations.

Lungu and Nyirenda (Open Journal of Applied Sciences, 2024): Commit/rollback techniques are essential for dependability and integrity, according to a systematic assessment of distributed transaction management.

Rationale: To ensure that multi-step activities (such as event registration) are all-or-nothing and maintain integrity, your stored procedures use START TRANSACTION, COMMIT, and ROLLBACK.

2. Procedures Stored

By encapsulating functionality inside the database, stored procedures enhance performance, security, and maintainability.

Emphasise stored procedures as a means of uniformly enforcing business rules at the database level (Elmasri & Navathe, 2016, Fundamentals of Database Systems).

According to Bhuyan (2023, International Journal of Computer Applications), stored procedures centralise transaction control and lower application–database communication costs.

Justification: Your CRUD processes for transactional scenarios and students guarantee uniform rule enforcement and minimise application code redundancy.

3.Triggers

Triggers facilitate auditing, constraint enforcement, and logging by automating responses to data changes.

According to Li et al. (2025, Springer, Database Security and Auditing), triggers are useful for creating audit trails and automatically logging operation logs.

According to Omotunde et al. (2023, ResearchGate, Database Security Review), auditing triggers are essential for relational system accountability and compliance.

Rationale: Your student audit triggers (INSERT, UPDATE, DELETE) are in line with best standards for traceability and transparency.

4. Indexing

By eliminating the requirement for complete table scans, indexes enhance query performance.

Anchlia (2024, International Journal of Computer Trends and Technology): Shows how indexing is an essential method for improving query performance and drastically cutting execution time.

Holubinka & Khudiyi (2024, Lviv Polytechnic National University): Examine indexing strategies and demonstrate how they affect the effectiveness of queries in big relational systems.

Khandal (2024, IJFANS) compares hash and B-tree indexes and demonstrates how well they speed up lookups and joins.

Rationale: These results are directly reflected in your secondary indexes on `student_id`, `club_id`, `event_id`, and `start_time`, guaranteeing scalability and quick query response.

Conclusion: Rollback/Commit: Endorsed by Lungu & Nyirenda (2024) and Grey & Reuter (1993).

Stored Procedures: Endorsed by Bhuyan (2023), Elmasri & Navathe (2016).

Triggers: Endorsed by Omotunde et al. (2023) and Li et al. (2025).

Indexing: Anchlia (2024), Holubinka & Khudiyi (2024), and Khandal (2024) provided support.

By demonstrating that your CCMS database adheres to best practices in relational database theory and practical research, these references offer scholarly support for your design and implementation.

4.Report Writing (20%)

Table of Contents

- Overview of a case study
- The importance of database implementation and design

Database Architecture (30%)

- 3NF normalisation
- Model of entities and relationships
- Definition of a schema

Implementation of databases (35%)

- Table construction and the population of sample data
- CRUD operations' stored routines
- Auditing functions and triggers
- Situations involving transactions with commit/rollback
- Performance tweaking (indexes, optimisation) and key queries

Support from Literature (15%)

- Academic support for stored procedures, transactions, triggers, indexing, and normalisation
- Citations from peer-reviewed publications

Writing Reports (20%)

- Documentation's organisation and clarity
- Combining technical implementation with a case study

Conclusion

- Synopsis of results

-Considering database performance, scalability, and integrity

References:

-APA/Harvard style academic and peer-reviewed sources

Introduction

The Campus Club Management System (CCMS), which was created to simplify the administration of student clubs, events, memberships, and communications in a university setting, is described in this study. The case study scenario illustrates the difficulties organisations encounter in upholding precise documentation, guaranteeing data integrity, and facilitating effective operations across several stakeholders. [3]

This project's importance stems from its capacity to illustrate best practices in relational database implementation and design. The system ensures integrity and removes redundancy by normalising the schema to Third Normal Form (3NF). The database guarantees dependable operations, accountability, and adherence to ACID principles through the use of stored procedures, triggers, and transactional controls. Additionally, query efficiency is improved through performance tuning with indexing, which makes the system scalable for practical application. [4]

This case study applies theoretical ideas like normalisation, transactions, and audits in a real-world setting while also validating them. The final approach offers a strong basis for overseeing student involvement, facilitating decision-making, and guaranteeing openness in campus club operations. [5]

Stored Procedure and Transactional Scenario

Explain practical use cases demonstrating conditional rollback and commit logic.

By creating operations that only finalise changes when all circumstances are met, I was able to illustrate the distinction between commit and rollback in the development of stored procedures and transactional scenarios. [6]

Commit Logic: A transaction is committed when a process, like adding a student record or

enrolling a student for an event, is completed successfully. This implies that every modification is stored in the database permanently. For instance, the event capacity is decreased and the participation record is added if a student is a legitimate member of the hosting club. After that, the transaction is committed, guaranteeing that the database shows the updated state.

Rollback Logic: All modifications made during a transaction are reversed if any condition fails. For example, the process initiates a rollback if a student attempts to register for an event but is not a member of the hosting club, or if the event is already filled. This maintains data integrity by preventing incomplete modifications, such as lowering capacity without actually registering the student. [7]

The system enforces all-or-nothing execution by merging these two mechanisms:

Commit guarantees the completion of legitimate procedures.

Rollback guarantees that incomplete or erroneous operations leave the database unaltered.

This example demonstrates how transactional control ensures consistency and atomicity, two ACID characteristics essential to dependable database system

Performance Tuning Enhancement

-Discuss your indexing strategy, focusing on performance-critical queries.

-Highlight how indexing benefits Campus Club's operational data handling.

-Explain expected improvements in query speed and efficiency.

Index strategy:

1. Performance-Critical Queries Indexing Strategy

Several queries using joins or filters on huge tables are often run in the CCMS database.

Secondary indexes were made on the following columns in order to optimise these queries:

Students:

Email and university_id are used for fast lookups and authentication.

Membership:

When listing members of a club or clubs that a student is a member of, student_id and club_id → are crucial for joins.

events:

Club_id → allows queries to retrieve events that are hosted by a certain club.

start_time → speeds up queries that filter past or future events.

Participation in the Event:

Attendance monitoring queries use event_id and student_id →.

Announcements

queries that retrieve announcements by club are supported by club_id →.

Announcement_Recipients:

Announcement delivery and reach are monitored using announcement_id and student_id →.

2.Benefits:

The Campus Club system's operating requirements are directly supported by indexing:

Membership Management: Because joins on student_id and club_id employ indexes, queries that list all club members or all clubs a student is a member of are quicker.

Event Attendance Tracking: Indexes on event_id and student_id optimise queries that count RSVP statuses or verify who attended an event.

Event Scheduling: Administrators can rapidly access forthcoming events without scanning the full table by filtering events by start_time.

Communication Effectiveness: Tracking which students got announcements is made efficient by indexes on announcement_id and student_id.

Authentication and Student Lookup: When logging in or looking up student data, quick validation is ensured using indexes on university_id and email.

3.Improvements:

The Campus Club system's operating requirements are directly supported by indexing:

Membership Management: Because joins on student_id and club_id employ indexes, queries that list all club members or all clubs a student is a member of are quicker.

Event Attendance Tracking: Indexes on event_id and student_id optimise queries that count RSVP statuses or verify who attended an event.

Event Scheduling: Administrators can rapidly access forthcoming events without scanning the full table by filtering events by start_time.

Communication Effectiveness: Tracking which students got announcements is made efficient by indexes on announcement_id and student_id.

Authentication and Student Lookup: When logging in or looking up student data, quick validation is ensured using indexes on university_id and email. [8]

Conclusion and Reflection

- Summarize how the final solution meets campus club's requirements.
- Reflect on challenges faced, solutions implemented and suggest future improvements.

Conclusion: Fulfilling Campus Club Requirements

The requirements listed in the case study are effectively met by the final Campus Club

Management System (CCMS):

Data Integrity: Redundancy was removed and consistency was guaranteed across all entities by normalising the schema to Third Normal Form (3NF).

Operational Efficiency: Reliable and atomic updates to important data, including memberships, events, and announcements, are ensured by stored procedures for CRUD operations in conjunction with transactional processing (commit/rollback).

Accountability: By automatically documenting modifications and facilitating compliance and troubleshooting, triggers and audit logs offer transparency.

Performance: By optimising query execution, secondary indexes on commonly searched columns (such as `student_id`, `club_id`, `event_id`, and `start_time`) guarantee scalability as the system expands.

Reflection:

A number of difficulties arose along the design and implementation process:

Constraint Errors: Initially, insert and remove operations failed due to duplicate entries and foreign key violations.

Solution: To avoid incomplete modifications and maintain integrity, transaction processing with rollback was put into place.

Auditing Complexity: It was necessary to balance performance and detail when designing triggers to collect useful audit data.

Solution: To ensure clarity in logs, helper functions were added to format audit entries consistently.

Performance bottlenecks: Without optimisation, queries involving joins across big tables ran slowly.

Solution: By creating secondary indexes on performance-critical columns, query speed was greatly increased.

The process of normalisation Trade-offs: Additional junction tables were occasionally needed to ensure 3NF, which raised the complexity of the schema.

Solution: Integrity was maintained and complexity was kept under control thanks to careful ERD design and reasoning.

Future improvements:

Although the existing method works well, there are a few improvements that could make the system even stronger:

Advanced Security: Limit sensitive operations to authorised users by implementing role-based access control (RBAC).

Analytics and Reporting: Include stored procedures to create dashboards on event success, club growth, and student engagement.

Scalability Improvements: To preserve performance, investigate indexing or partitioning

techniques for very big datasets.

Automation: Include scheduled tasks for automated alerts (like event reminders).

User Interface Integration: To give administrators and students a smooth experience, link the database to a web or mobile front-end. [9]

References:

[1]

S. Kolahi and L. Libkin, “On Redundancy vs Dependency Preservation in Normalization: An Information-Theoretic Study of 3NF.” Available:
<https://homepages.inf.ed.ac.uk/libkin/papers/pods06b.pdf>

[2]

L. Mbangata and U. G. Singh, “Relational Database Normalization Concepts Using Numbers: An Ethnomathematics Approach,” *Lecture Notes in Networks and Systems*, vol. 1, no. 1241, pp. 1–7, 2025, doi: https://doi.org/10.1007/978-981-97-9559-8_1.

[3]

C. J. Date and C. J. Date, *An introduction to database systems*, 7th ed. Reading, Mass: Addison-Wesley, 2000. Available:
https://openlibrary.org/books/OL24939170M/An_introduction_to_database_systems

[4]

J. Gray and A. Reuter, “Transaction Processing: Concepts and Techniques,” *Microsoft Research*, Dec. 31, 1991.

<https://www.microsoft.com/en-us/research/publication/transaction-processing-concepts-and-techniques/> (accessed Nov. 28, 2025).

[5]

A. Anchlia, “Enhancing Query Performance Through Relational Database Indexing,” *International Journal of Computer Trends and Technology*, vol. 72, no. 8, pp. 130–133, Aug. 2024, doi: <https://doi.org/10.14445/22312803/ijctt-v72i8p119>.

[6]

Pearson.com, “Fundamentals of Database Systems, 7th edition | eTextBook Subscription | Pearson+,” *Pearson.com*, 2021. <https://www.pearson.com/en-us/pearsonplus/p/9780137502523>

[7]

Y. Li, C. Li, and Z. Cui, “Intelligent Database Triggers Enable Advanced Analysis of Data Recorded in Audit Logs,” *Lecture Notes in Networks and Systems*, vol. 243, no. 213, pp. 184–193, 2025, doi: https://doi.org/10.1007/978-3-031-88287-6_17.

[8]

A. Anchlia, “Enhancing Query Performance Through Relational Database Indexing,” *International Journal of Computer Trends and Technology*, vol. 72, no. 8, pp. 130–133, Aug. 2024, doi: <https://doi.org/10.14445/22312803/ijctt-v72i8p119>.

[9]

A. Silberschatz, *Database system concepts*. New York McGraw-Hill Education, 2020.

