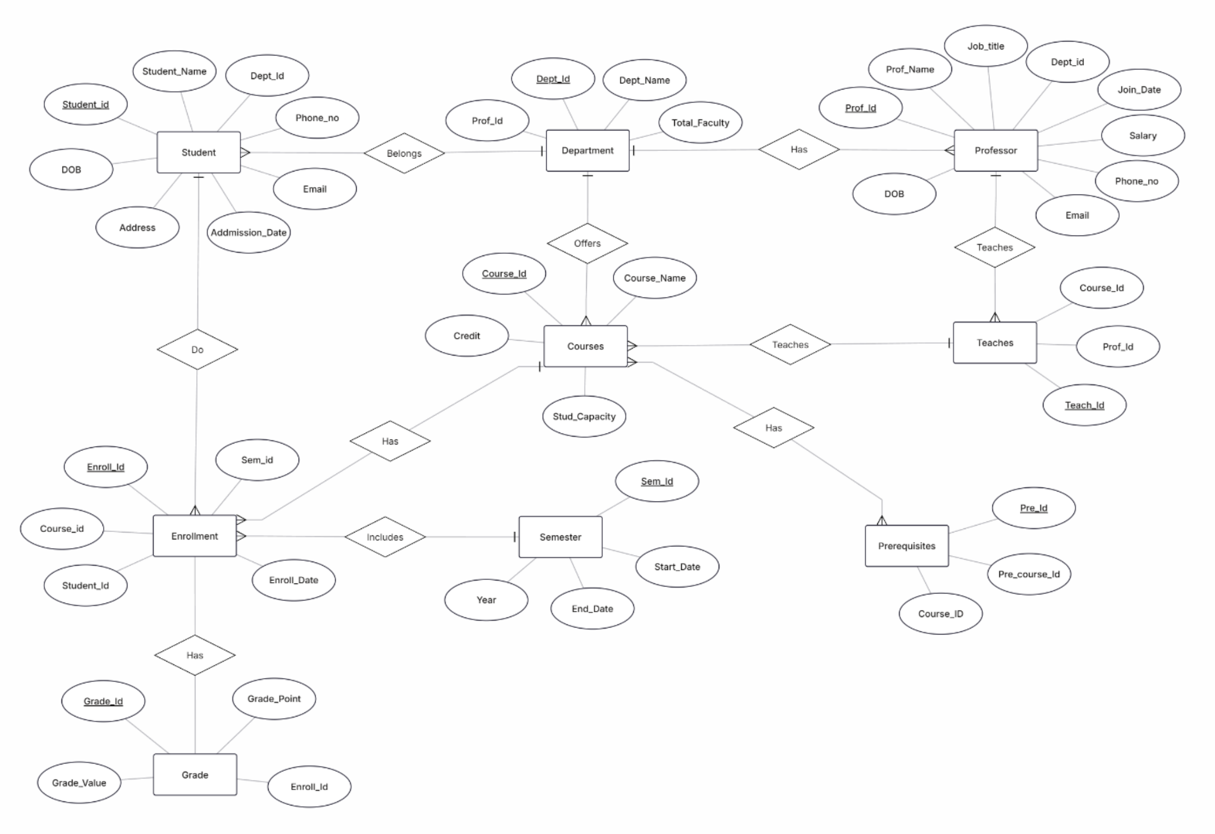
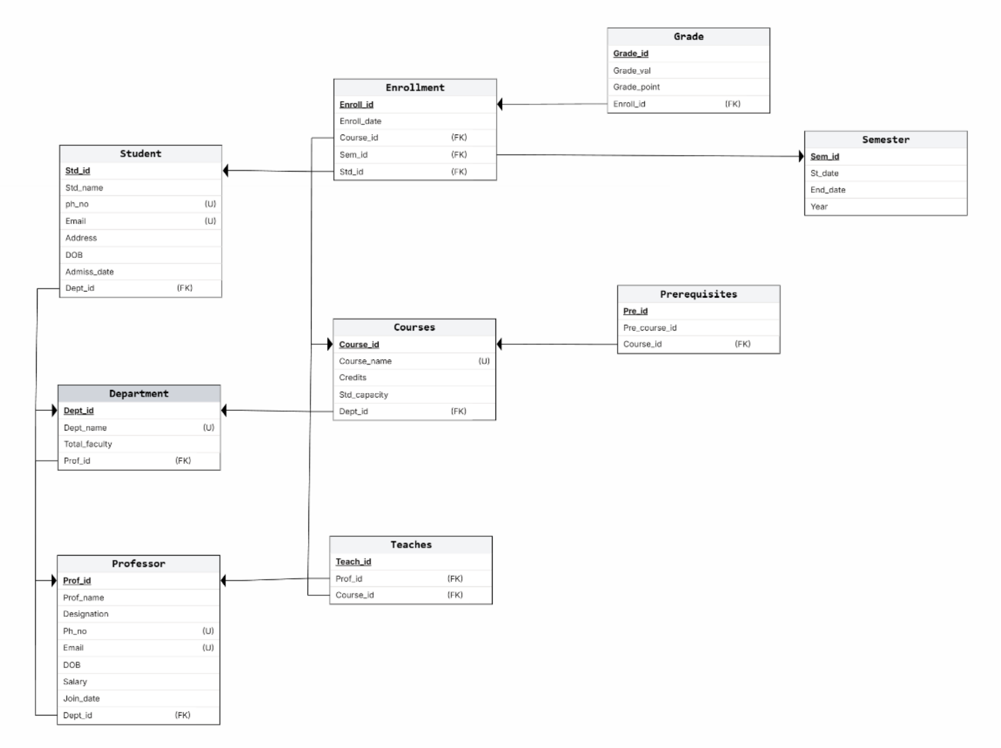
COLLEGE MANAGEMENT SYSTEM
ER DIAGRAM:



SCHEMA DIAGRAM:



Grade	
Grade_id	
Grade_val	
Grade_point	
Enroll_id	(FK)

Enrollment	
Enroll_id	
Enroll_date	
Course_id	(FK)
Sem_id	(FK)
Std_id	(FK)

Student	
Std_id	
Std_name	
ph_no	(U)
Email	(U)
Address	
DOB	
Admiss_date	
Dept_id	(FK)

Semester	
Sem_id	
St_date	
End_date	
Year	

Prerequisites	
Pre_id	
Pre_course_id	
Course_id	(FK)

Courses	
Course_id	
Course_name	(U)
Credits	
Std_capacity	
Dept_id	(FK)

Department	
Dept_id	
Dept_name	(U)
Total_faculty	
Prof_id	(FK)

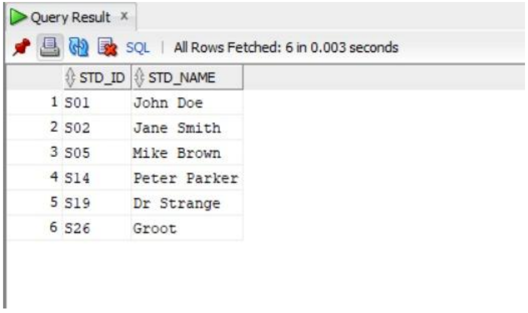
Teaches	
Teach_id	
Prof_id	(FK)
Course_id	(FK)

Professor	
Prof_id	
Prof_name	
Designation	
Ph_no	(U)
Email	(U)
DOB	
Salary	
Join_date	
Dept_id	(FK)

SQL CODES:

Basic Queries

1. List students enrolled in the "Computer Science" department.
   Query:
   Select std_id,std_name from student where dept_id=(Select dept_id from Department where dept_name='Computer Science');
   Output:

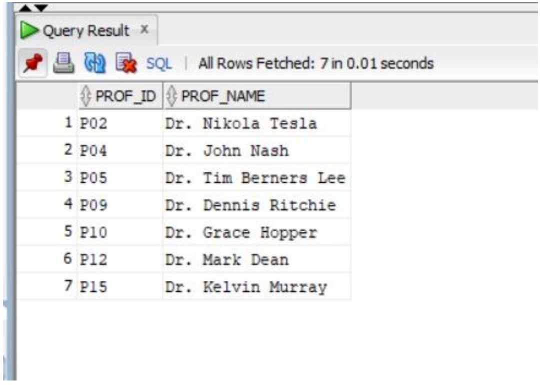


All Rows Fetched: 6 in 0.003 seconds

	STD_ID	STD_NAME
1	S01	John Doe
2	S02	Jane Smith
3	S05	Mike Brown
4	S14	Peter Parker
5	S19	Dr Strange
6	S26	Groot

   It returns all student data in "Computer Science" department

2. Find professors who joined after 2020.
   Query:
   Select prof_id,prof_name from Professor where join_date<DATE '2020-12-31';
   Output:

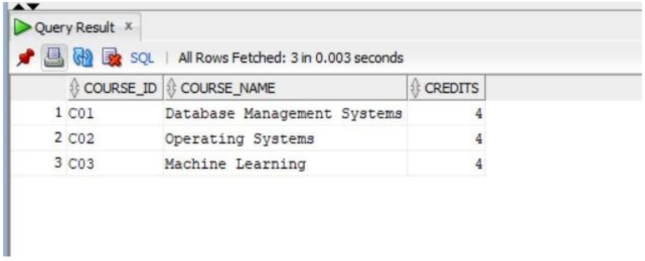


All Rows Fetched: 7 in 0.01 seconds

	PROF_ID	PROF_NAME
1	P02	Dr. Nikola Tesla
2	P04	Dr. John Nash
3	P05	Dr. Tim Berners Lee
4	P09	Dr. Dennis Ritchie
5	P10	Dr. Grace Hopper
6	P12	Dr. Mark Dean
7	P15	Dr. Kelvin Murray

   This query filters the professors based on their join date. Any professor whose join_date falls after the year 2020 will be included in the result.

3. Retrieve courses that are worth 4 credits
   Query:
   select course_id,course_name,credits from courses where credits=4;
   Output:



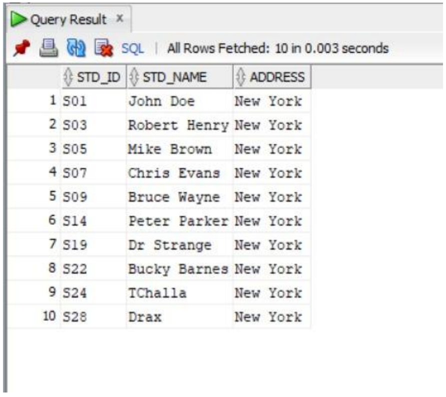
All Rows Fetched: 3 in 0.003 seconds

	COURSE_ID	COURSE_NAME	CREDITS
1	C01	Database Management Systems	4
2	C02	Operating Systems	4
3	C03	Machine Learning	4

   This simply selects courses where the value in the credits column equals 4

4. List students who live in "New York" (based on address).
   Query:
   select std_id,std_name,address from student where address='New York';

Output:



All Rows Fetched: 10 in 0.003 seconds

	STD_ID	STD_NAME	ADDRESS
1	S01	John Doe	New York
2	S03	Robert Henry	New York
3	S05	Mike Brown	New York
4	S07	Chris Evans	New York
5	S09	Bruce Wayne	New York
6	S14	Peter Parker	New York
7	S19	Dr Strange	New York
8	S22	Bucky Barnes	New York
9	S24	TChalla	New York
10	S28	Drax	New York

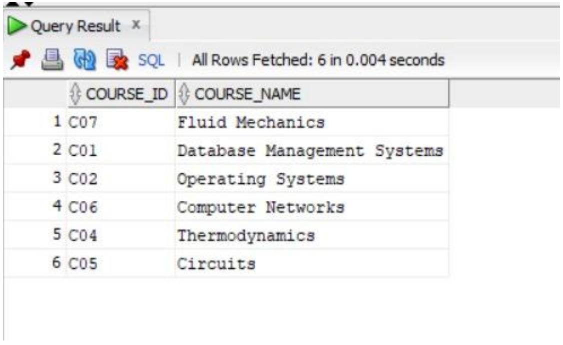
This retrieves all students whose address matches “New York”

5. Find courses that have no prerequisites.

   Query:

   select course_id,course_name from courses where course_id NOT IN (select course_id from prerequisites);

   Output:



All Rows Fetched: 6 in 0.004 seconds

	COURSE_ID	COURSE_NAME
1	C07	Fluid Mechanics
2	C01	Database Management Systems
3	C02	Operating Systems
4	C06	Computer Networks
5	C04	Thermodynamics
6	C05	Circuits

This retrives all the courses that have NOT have prerequisites

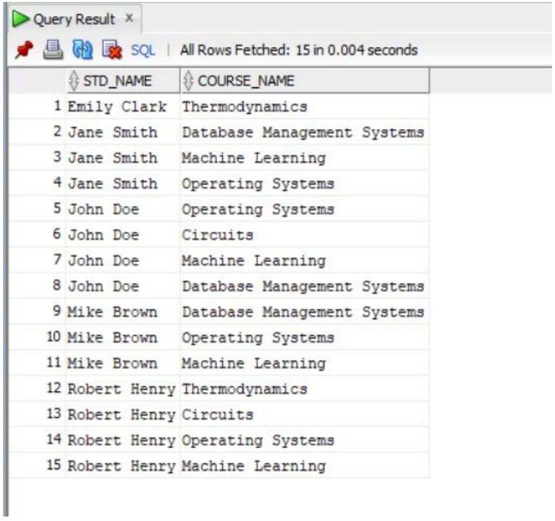
Joins & Subqueries

6. Display student names along with the courses they are currently enrolled in.

   Query:

   select s.std_name,c.course_name from student s JOIN enrollment e ON s.std_id=e.std_id JOIN courses c ON c.course_id=e.course_id ORDER BY s.std_name;

   Output:



All Rows Fetched: 15 in 0.004 seconds

	STD_NAME	COURSE_NAME
1	Emily Clark	Thermodynamics
2	Jane Smith	Database Management Systems
3	Jane Smith	Machine Learning
4	Jane Smith	Operating Systems
5	John Doe	Operating Systems
6	John Doe	Circuits
7	John Doe	Machine Learning
8	John Doe	Database Management Systems
9	Mike Brown	Database Management Systems
10	Mike Brown	Operating Systems
11	Mike Brown	Machine Learning
12	Robert Henry	Thermodynamics
13	Robert Henry	Circuits
14	Robert Henry	Operating Systems
15	Robert Henry	Machine Learning

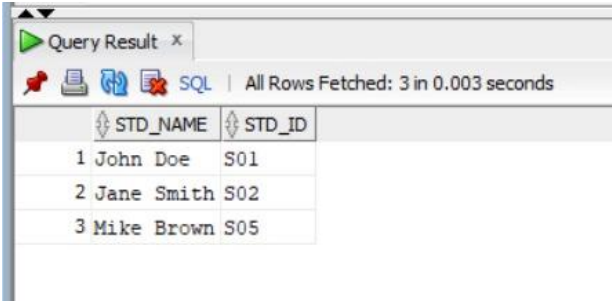
A join between student ,enrolment and courses shows which student is taking which course recently

7. Find students who received an 'A' grade in 'Database Management Systems'.

Query:

select s.std_name,s.std_id from grade g JOIN enrollment e ON g.enroll_id=e.enroll_id JOIN student s ON e.std_id=s.std_id JOIN courses c ON e.course_id=c.course_id where g.grade_val='A' AND c.course_name='Database Management Systems';

Output:



All Rows Fetched: 3 in 0.003 seconds

	STD_NAME	STD_ID
1	John Doe	S01
2	Jane Smith	S02
3	Mike Brown	S05

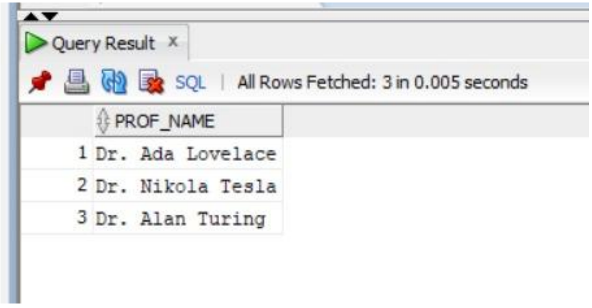
This query retrieves the students records whose grade is ‘A’ in DBMS

8. List professors who teach students from other departments.

Query:

select DISTINCT p.prof_name from professor p JOIN teaches t ON p.prof_id=t.prof_id JOIN enrollment e ON t.course_id=e.course_id JOIN student s ON e.std_id=s.std_id where p.dept_id<>s.dept_id;

Output:



All Rows Fetched: 3 in 0.005 seconds

	PROF_NAME
1	Dr. Ada Lovelace
2	Dr. Nikola Tesla
3	Dr. Alan Turing

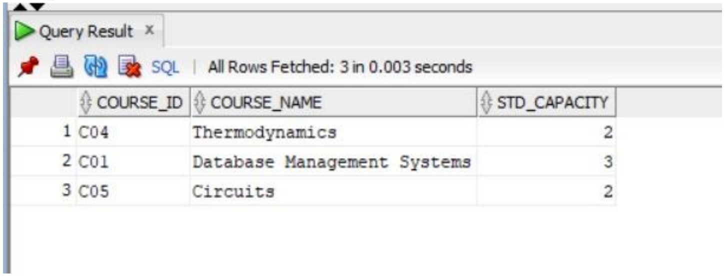
This query checks if prof’s dept differs from student dept .If yes it means prof is from another department

9. Identify courses that have reached their full capacity (Enrollment = Capacity).

Query:

select c.course_id,c.course_name, c.std_capacity from courses c JOIN enrollment e ON c.course_id=e.course_id GROUP BY c.course_id,c.course_name,c.std_capacity HAVING COUNT(*)=c.std_capacity;

Output:



All Rows Fetched: 3 in 0.003 seconds

	COURSE_ID	COURSE_NAME	STD_CAPACITY
1	C04	Thermodynamics	2
2	C01	Database Management Systems	3
3	C05	Circuits	2

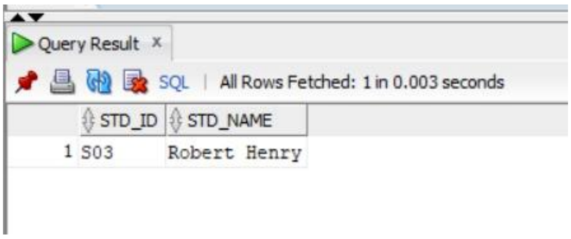
Counts how many students enrolled in a course and compares with the course capacity

10.List students who have failed (Grade 'F') in more than 2 subjects.

Query:

select s.std_id,s.std_name from grade g JOIN enrollment e ON g.enroll_id=e.enroll_id JOIN student s ON e.std_id=s.std_id where g.grade_val='F' GROUP BY s.std_id,s.std_name HAVING COUNT(*)>2;

Output:



```
Query Result
SQL | All Rows Fetched: 1 in 0.003 seconds
    STD_ID  STD_NAME
  1 S03     Robert Henry
```

Grops results by student and course how many ‘F’ grade, if count is more than 2 the student data listed

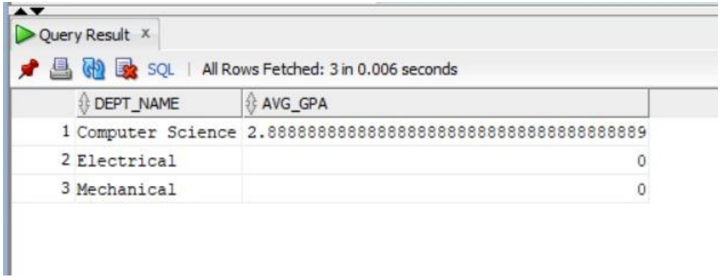
Aggregation & Reports

11.Calculate the average GPA per department.

Query:

```
select d.dept_name,AVG(g.grade_point) AS Avg_GPA from grade g JOIN enrollment e ON
g.enroll_id=e.enroll_id JOIN student s ON e.std_id=s.std_id JOIN department d ON s.dept_id=d.dept_id
GROUP BY d.dept_name;
```

Output:



```
Query Result
SQL | All Rows Fetched: 3 in 0.006 seconds
    DEPT_NAME         AVG_GPA
  1 Computer Science  2.88888888888888888888888888888888888889
  2 Electrical                                               0
  3 Mechanical                                               0
```

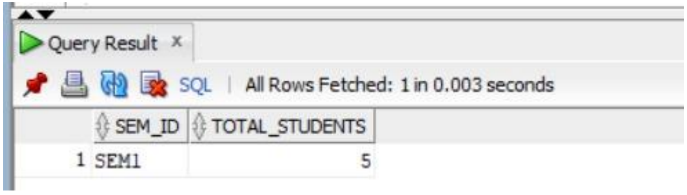
Calculate average GPA per department using Join ,Aggregate ,GroupBy

12.Total number of students enrolled in each semester.

Query:

```
select e.sem_id,COUNT(DISTINCT e.std_id) AS Total_Students from enrollment e GROUP BY e.sem_id;
```

Output:



```
Query Result
SQL | All Rows Fetched: 1 in 0.003 seconds
    SEM_ID  TOTAL_STUDENTS
  1 SEM1                 5
```

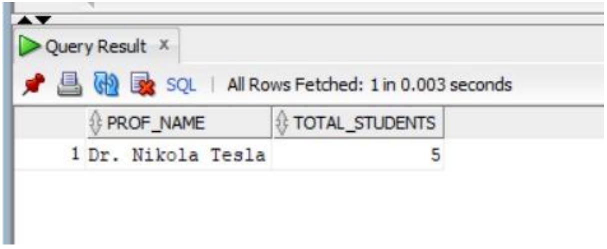
Count distinct students enrolled in semester

13.Identify the professor teaching the highest number of students.

Query:

```
select prof_name, total_Students from(select p.prof_name,COUNT(DISTINCT e.std_id) AS
Total_Students from Professor p JOIN Teaches t ON p.prof_id = t.prof_id JOIN enrollment e ON
t.course_id = e.course_id GROUP BY p.prof_name ORDER BY Total_Students
DESC)where ROWNUM = 1;
```

Output:



```
Query Result
SQL | All Rows Fetched: 1 in 0.003 seconds
    PROF_NAME         TOTAL_STUDENTS
  1 Dr. Nikola Tesla               5
```

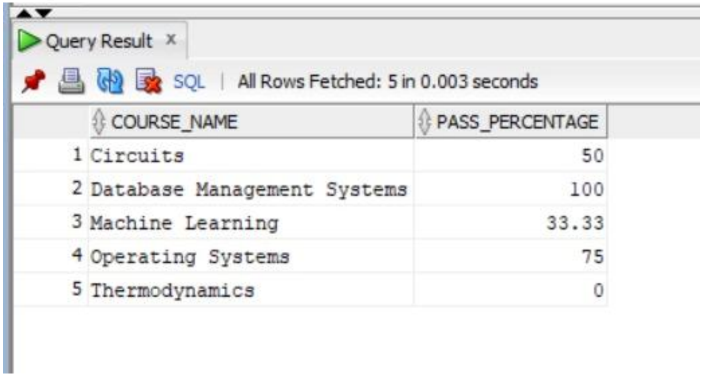
Counts how many unique student each professor teaches and selects the professor with high count

14.Percentage of students passing per course.

Query:

select c.course_name,ROUND((SUM(CASE WHEN g.grade_val <> 'F' THEN 1 ELSE 0 END) *100)/COUNT(*), 2) AS Pass_Percentage FROM courses c JOIN enrollment e ON c.course_id = e.course_id
JOIN grade g   ON g.enroll_id = e.enroll_id
GROUP BY c.course_name;

Output:



All Rows Fetched: 5 in 0.003 seconds

	COURSE_NAME	PASS_PERCENTAGE
1	Circuits	50
2	Database Management Systems	100
3	Machine Learning	33.33
4	Operating Systems	75
5	Thermodynamics	0

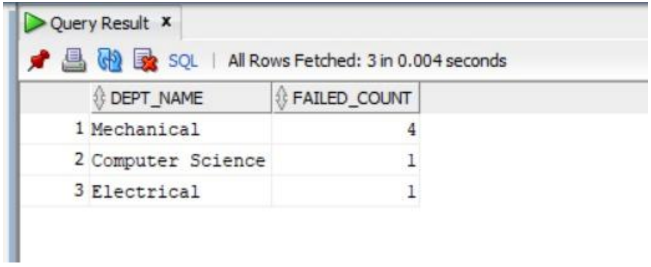
Calculate percentage of students using formula

15.Departments with the highest number of failed students.

Query:

select d.dept_name,COUNT(*) AS Failed_Count from department d JOIN student s ON d.dept_id = s.dept_id JOIN enrollment e ON s.std_id = e.std_id JOIN grade g ON e.enroll_id = g.enroll_id
where g.grade_val = 'F' GROUP BY d.dept_name
ORDER BY Failed_Count DESC;

Output:



All Rows Fetched: 3 in 0.004 seconds

	DEPT_NAME	FAILED_COUNT
1	Mechanical	4
2	Computer Science	1
3	Electrical	1

Counts all students who got an ‘F’ and groups them by department

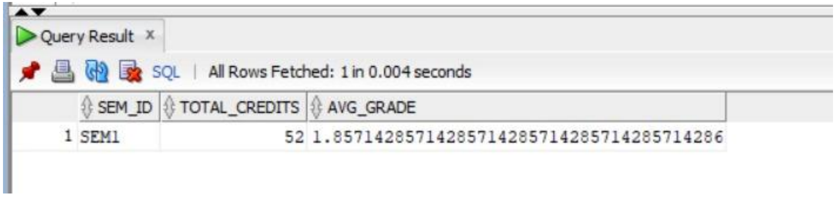
Advanced Queries

16.Generate a semester-wise report (Semester, Total Credits, Average Grade).

Query:

select sem_id,SUM(credits) AS Total_Credits,AVG(grade_point) AS Avg_Grade from enrollment JOIN courses USING (course_id) JOIN grade USING (enroll_id) GROUP BY sem_id;

Output:



All Rows Fetched: 1 in 0.004 seconds

	SEM_ID	TOTAL_CREDITS	AVG_GRADE
1	SEM1	52	1.85714285714285714285714285714285714286

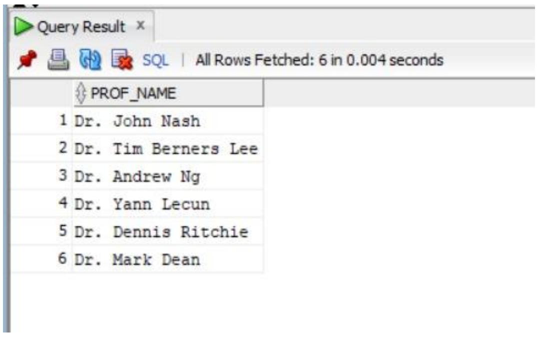
For each sem calculate Sum of credits and Avg of grade points

17.List professors earning above the average salary of the college.

Query:

select prof_name from professor where salary > (select AVG(salary) from professor);

Output:



All Rows Fetched: 6 in 0.004 seconds

	PROF_NAME
1	Dr. John Nash
2	Dr. Tim Berners Lee
3	Dr. Andrew Ng
4	Dr. Yann Lecun
5	Dr. Dennis Ritchie
6	Dr. Mark Dean

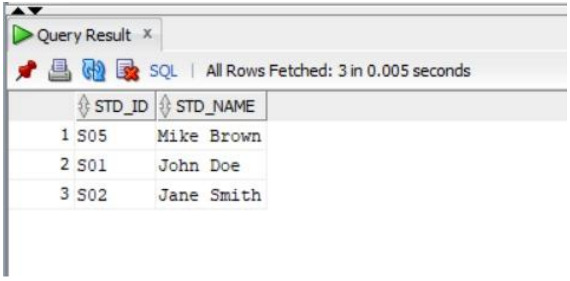
Compares each professor salary with average salary of all

18.Identify students who have completed all prerequisites for 'Machine Learning'.

Query:

select DISTINCT s.std_id, s.std_name from student s JOIN enrollment e ON s.std_id = e.std_id JOIN grade g ON e.enroll_id = g.enroll_id where e.course_id IN (select pre_course_id from prerequisites where course_id = 'C03')GROUP BY s.std_id, s.std_name HAVING COUNT(DISTINCT e.course_id) = (select COUNT(*) from prerequisites where course_id = 'C03');

Output:



All Rows Fetched: 3 in 0.005 seconds

	STD_ID	STD_NAME
1	S05	Mike Brown
2	S01	John Doe
3	S02	Jane Smith

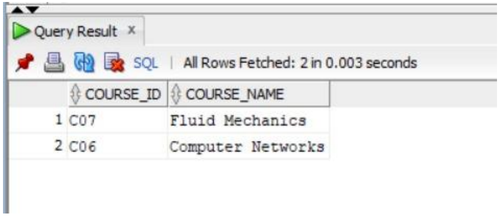
Checks if the student has completed every course as a prerequisite for ML

19.Find courses that have never been opted for by any student.

Query:

select c.course_id, c.course_name FROM courses c LEFT JOIN enrollment e ON c.course_id = e.course_id where e.course_id IS NULL;

Output:



All Rows Fetched: 2 in 0.003 seconds

	COURSE_ID	COURSE_NAME
1	C07	Fluid Mechanics
2	C06	Computer Networks

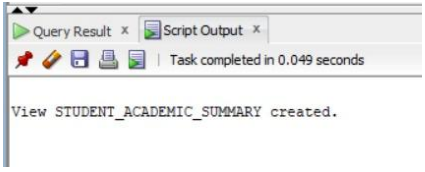
If there is no erollment for course it will displayed

20.Create a view showing: Student Name, Department, Total Credits, and CGPA.

Query:

create view Student_Academic_Summary AS select s.std_id,s.std_name,d.dept_name, SUM(c.credits) AS Total_Credits,AVG(g.grade_point) AS CGPA from student s JOIN enrollment e ON s.std_id = e.std_id JOIN courses c ON e.course_id = c.course_id JOIN grade g ON e.enroll_id = g.enroll_id JOIN department d ON s.dept_id = d.dept_id GROUP BY s.std_id, s.std_name, d.dept_name;

Output:



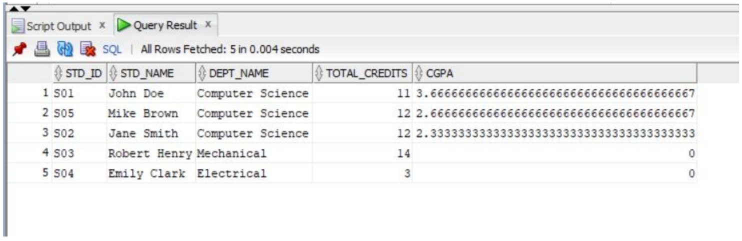
Task completed in 0.049 seconds

```
View STUDENT_ACADEMIC_SUMMARY created.
```

Query:

select * from Student_Academic_Summary;

Output:



All Rows Fetched: 5 in 0.004 seconds

	STD_ID	STD_NAME	DEPT_NAME	TOTAL_CREDITS	CGPA
1	S01	John Doe	Computer Science	11	3.66666666666666666666666666666666666667
2	S05	Mike Brown	Computer Science	12	2.66666666666666666666666666666666666667
3	S02	Jane Smith	Computer Science	12	2.33333333333333333333333333333333333333
4	S03	Robert Henry	Mechanical	14	0
5	S04	Emily Clark	Electrical	3	0

A view is created for each each student data including dept,total credits and cgpa

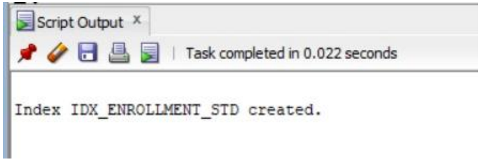
INDEXING AND QUERY OPTIMIZATION:

1.Create indexes on student_id,course_id,and grade.

Query:

CREATE INDEX idx_enrollment_std ON enrollment(std_id);

Output:



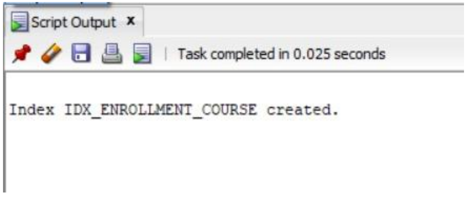
Task completed in 0.022 seconds

```
Index IDX_ENROLLMENT_STD created.
```

Query:

CREATE INDEX idx_enrollment_course ON enrollment(course_id);

Ouptut:



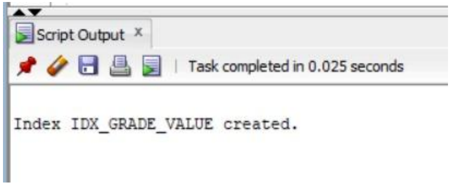
Task completed in 0.025 seconds

```
Index IDX_ENROLLMENT_COURSE created.
```

Query:

CREATE INDEX idx_grade_value ON grade(grade_val);

Output:



Task completed in 0.025 seconds

```
Index IDX_GRADE_VALUE created.
```

Index can be created for each table using ‘CREATE INDEX’

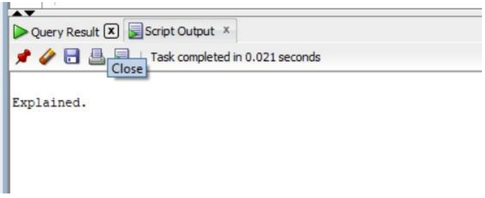
2.Comapre execution plans with vs without indexes.

Without indexes:

Query:

EXPLAIN PLAN FOR select s.std_name, s.std_id from grade g JOIN enrollment e ON g.enroll_id = e.enroll_id JOIN student s ON e.std_id = s.std_id JOIN courses c ON e.course_id = c.course_id where g.grade_val = 'A' AND c.course_name = 'Database Management Systems';

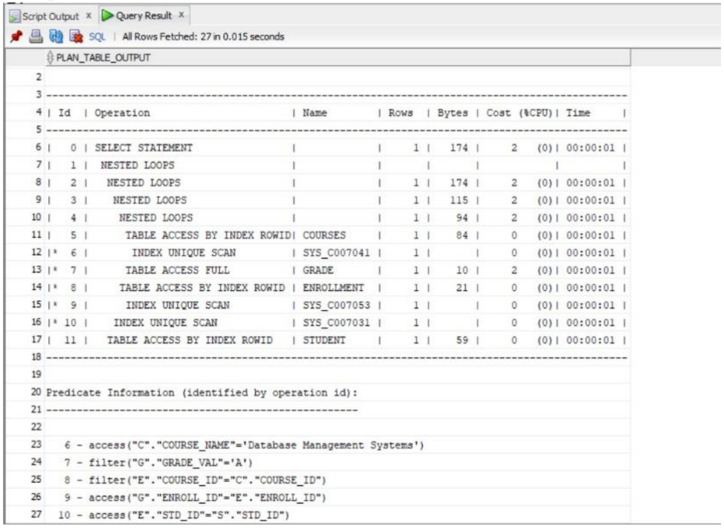
Output:



Task completed in 0.021 seconds

```
Explained.
```

Query:
select * from TABLE(DBMS_XPLAN.DISPLAY);
Output:



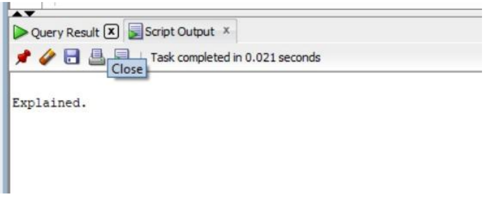
All Rows Fetched: 27 in 0.015 seconds

```

-----------------------------------------------------------------------------------------------
| Id  | Operation                       | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                |              |     1 |   174 |     2   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                   |              |       |       |            |          |
|   2 |   NESTED LOOPS                  |              |     1 |   174 |     2   (0)| 00:00:01 |
|   3 |    NESTED LOOPS                 |              |     1 |   115 |     2   (0)| 00:00:01 |
|   4 |     NESTED LOOPS                |              |     1 |    94 |     2   (0)| 00:00:01 |
|   5 |      TABLE ACCESS BY INDEX ROWID| COURSES      |     1 |    84 |     0   (0)| 00:00:01 |
|*  6 |       INDEX UNIQUE SCAN         | SYS_C007041  |     1 |       |     0   (0)| 00:00:01 |
|*  7 |      TABLE ACCESS FULL          | GRADE        |     1 |    10 |     2   (0)| 00:00:01 |
|*  8 |     TABLE ACCESS BY INDEX ROWID | ENROLLMENT   |     1 |    21 |     0   (0)| 00:00:01 |
|*  9 |      INDEX UNIQUE SCAN          | SYS_C007053  |     1 |       |     0   (0)| 00:00:01 |
|* 10 |    INDEX UNIQUE SCAN            | SYS_C007031  |     1 |       |     0   (0)| 00:00:01 |
|  11 |   TABLE ACCESS BY INDEX ROWID   | STUDENT      |     1 |    59 |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   6 - access("C"."COURSE_NAME"='Database Management Systems')
   7 - filter("G"."GRADE_VAL"='A')
   8 - filter("E"."COURSE_ID"="C"."COURSE_ID")
   9 - access("G"."ENROLL_ID"="E"."ENROLL_ID")
  10 - access("E"."STD_ID"="S"."STD_ID")
```

Tabels executes results slower because indexing can’t used in these tables right now

With indexes:

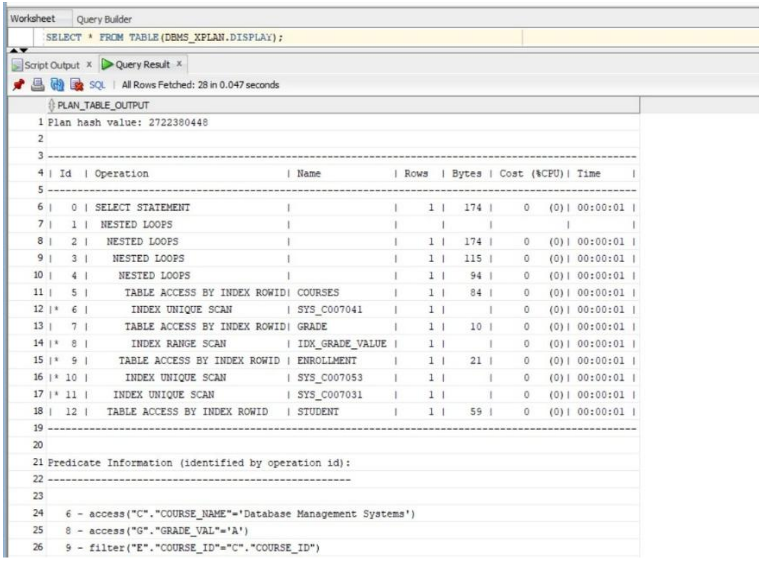
Query:
EXPLAIN PLAN FOR select s.std_name, s.std_id from grade g JOIN enrollment e ON g.enroll_id = e.enroll_id JOIN student s ON e.std_id = s.std_id JOIN courses c   ON e.course_id = c.course_id where g.grade_val = 'A' AND c.course_name = 'Database Management Systems';
Output:



Task completed in 0.021 seconds

```
Explained.
```

Query:
select * from TABLE(DBMS_XPLAN.DISPLAY);
Output:



```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

All Rows Fetched: 28 in 0.047 seconds

```
Plan hash value: 2722380448

-----------------------------------------------------------------------------------------------
| Id  | Operation                       | Name            | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                |                 |     1 |   174 |     0   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                   |                 |       |       |            |          |
|   2 |   NESTED LOOPS                  |                 |     1 |   174 |     0   (0)| 00:00:01 |
|   3 |    NESTED LOOPS                 |                 |     1 |   115 |     0   (0)| 00:00:01 |
|   4 |     NESTED LOOPS                |                 |     1 |    94 |     0   (0)| 00:00:01 |
|   5 |      TABLE ACCESS BY INDEX ROWID| COURSES         |     1 |    84 |     0   (0)| 00:00:01 |
|*  6 |       INDEX UNIQUE SCAN         | SYS_C007041     |     1 |       |     0   (0)| 00:00:01 |
|   7 |      TABLE ACCESS BY INDEX ROWID| GRADE           |     1 |    10 |     0   (0)| 00:00:01 |
|*  8 |       INDEX RANGE SCAN          | IDX_GRADE_VALUE |     1 |       |     0   (0)| 00:00:01 |
|*  9 |     TABLE ACCESS BY INDEX ROWID | ENROLLMENT      |     1 |    21 |     0   (0)| 00:00:01 |
|* 10 |      INDEX UNIQUE SCAN          | SYS_C007053     |     1 |       |     0   (0)| 00:00:01 |
|* 11 |    INDEX UNIQUE SCAN            | SYS_C007031     |     1 |       |     0   (0)| 00:00:01 |
|  12 |   TABLE ACCESS BY INDEX ROWID   | STUDENT         |     1 |    59 |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   6 - access("C"."COURSE_NAME"='Database Management Systems')
   8 - access("G"."GRADE_VAL"='A')
   9 - filter("E"."COURSE_ID"="C"."COURSE_ID")
```

Tabels executes results faster by using indexes ,better than tables without indexing

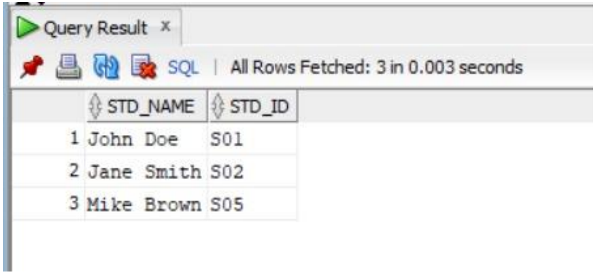
3. Rewrite 2 queries (from the list above) into optimized versions.

Students who got 'A' in “Database Management Systems”

Query:

```
SELECT s.std_name, s.std_id FROM grade g JOIN enrollment e ON g.enroll_id = e.enroll_id JOIN
student s ON e.std_id = s.std_id WHERE g.grade_val = 'A'  AND e.course_id = 'C01';
```

Output:

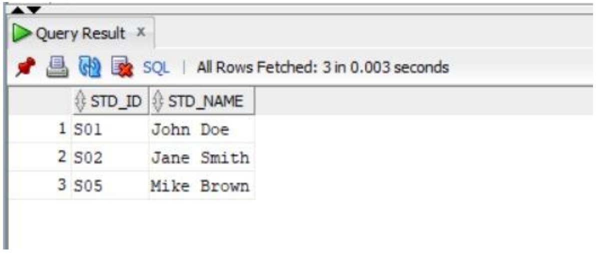


	STD_NAME	STD_ID
1	John Doe	S01
2	Jane Smith	S02
3	Mike Brown	S05

Query:

```
SELECT s.std_id, s.std_name FROM student s JOIN enrollment e ON s.std_id = e.std_id WHERE
e.course_id IN (SELECT pre_course_id FROM prerequisites WHERE course_id = 'C03') GROUP BY
s.std_id, s.std_name HAVING COUNT(DISTINCT e.course_id) = (SELECT COUNT(*) FROM
prerequisites WHERE course_id = 'C03');
```

Output:



	STD_ID	STD_NAME
1	S01	John Doe
2	S02	Jane Smith
3	S05	Mike Brown

These queries executes better ,because it optimized and results displayed

TRANSACTIONS & RECOVERY:

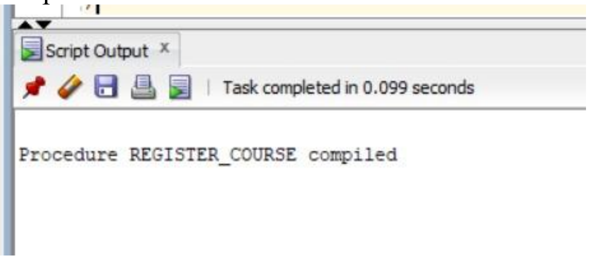
Implement a "Course Registration" Transaction

Query:

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE PROCEDURE register_course (p_std_id IN VARCHAR2, p_course_id IN
VARCHAR2, p_sem_id IN VARCHAR2 DEFAULT 'SEM1') IS v_seats_left Courses.Seats_left%TYPE;
v_prereq_count  NUMBER;
v_completed     NUMBER;
v_new_enroll_id VARCHAR2(10);
BEGIN
SAVEPOINT before_registration;
SELECT Seats_left INTO v_seats_left FROM course WHERE course_id = p_course_id FOR UPDATE;

IF v_seats_left <= 0 THEN
DBMS_OUTPUT.PUT_LINE('Course full. Rolling back.');
ROLLBACK TO before_registration;
RETURN;
END IF;

SELECT COUNT(*) INTO v_prereq_count FROM prerequisites WHERE course_id = p_course_id;

IF v_prereq_count > 0 THEN
```

```
SELECT COUNT(DISTINCT e.course_id) INTO v_completed FROM enrollment e grade g ON
g.enroll_id = e.enroll_id WHERE e.std_id = p_std_id  AND g.grade_val <> 'F' AND e.course_id IN (
SELECT pre_course_id FROM prerequisites WHERE course_id = p_course_id);

IF v_completed < v_prereq_count THEN
DBMS_OUTPUT.PUT_LINE('Prerequisites not completed. Rolling back.');
ROLLBACK TO before_registration;
RETURN;
END IF;
END IF;

v_new_enroll_id := 'E' || LPAD(ENROLL_SEQ.NEXTVAL, 3, '0');

INSERT INTO enrollment (enroll_id, enroll_date, course_id, sem_id, std_id)
VALUES (v_new_enroll_id,SYSDATE,p_course_id,p_sem_id,p _std_id);

UPDATE courses SET Seats_left = Seats_left – 1 WHERE course_id = p_course_id;
COMMIT;
DBMS_OUTPUT.PUT_LINE('Registration successful. enroll_id = ' || v_new_enroll_id);
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
      ROLLBACK;
END register_course;
/
```

Output:



```
Procedure REGISTER_COURSE compiled
```

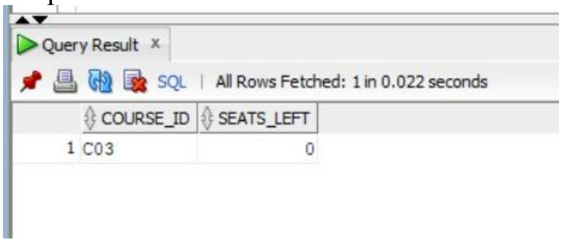
1. Check course capacity (Rollback if full)

Step 1: Force course to be full

Query:

```
UPDATE course SET Seats_left = 0 WHERE course_id = 'C03';
COMMIT;
SELECT course_id, Seats_left FROM course WHERE course_id = 'C03';
```

Output:



	COURSE_ID	SEATS_LEFT
1	C03	0

This confirms that the course has zero seats ,it means it is already full

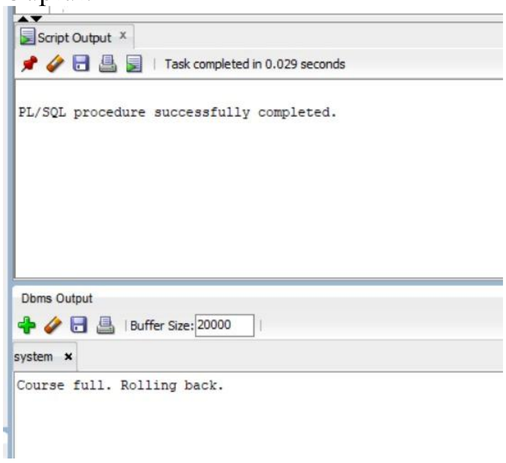
Step 2: Try to register

Query:

```
BEGIN
   register_course('S01', 'C03', 'SEM1');
END;
```

/

Ouptut:



```
PL/SQL procedure successfully completed.
```

```
Course full. Rolling back.
```

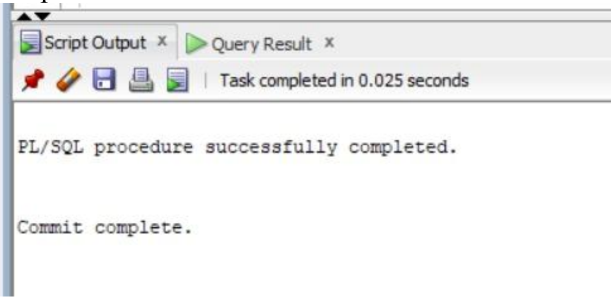
When trying to register S01 for C03,the procedure detects the course have filled and rollback to savepoint

2. Check prerequisite completion (Rollback if not met)

Query:

```
UPDATE course SET Seats_left=5 WHERE course_id = 'C03';
COMMIT;
```

Output:



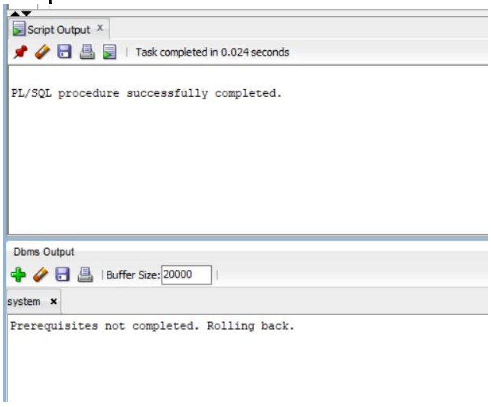
```
PL/SQL procedure successfully completed.


Commit complete.
```

Updates the seat capacity for course id C03 to 5 seats

Query:

```
BEGIN
   register_course('S04', 'C03', 'SEM1');
END;
/
```

Output:



```
PL/SQL procedure successfully completed.
```

```
Prerequisites not completed. Rolling back.
```

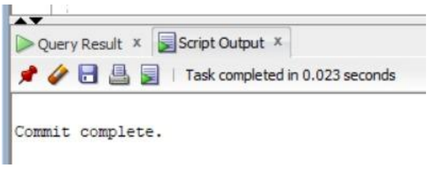
Student S04 register for course C03 ,it checks the student does complete all prerequisites, then procedure successfully creates a new enrollment

3. Insert enrollment record

Query:

```
UPDATE course SET Seats_left = 3 WHERE course_id = 'C03';
COMMIT;
```

Ouput:



Task completed in 0.023 seconds

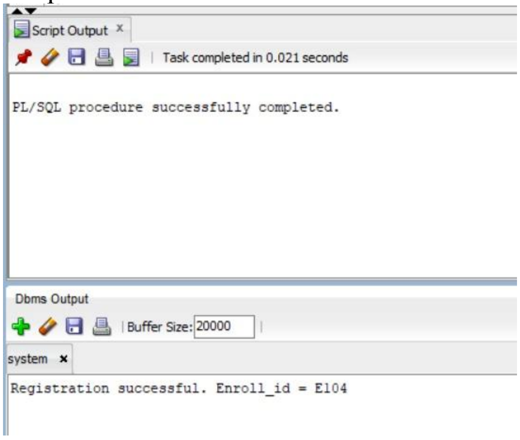
```
Commit complete.
```

Thus this reduces the seat capacity for the course C03, and it commits

Query:

```
BEGIN
   register_course('S01', 'C03', 'SEM1');
END;
/
```

Output:



Task completed in 0.021 seconds

```
PL/SQL procedure successfully completed.
```

Buffer Size: 20000

```
Registration successful. Enroll_id = E104
```

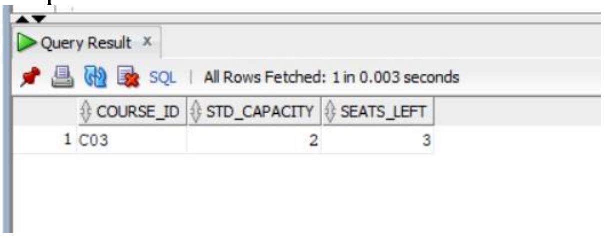
Student S01 register for course C03 ,it checks the student does complete all prerequisites, then procedure successfully creates a new enrollment

4.Update remaining seat count.

Query:

```
SELECT course_id, std_capacity, Seats_left FROM course WHERE course_id = 'C03';
```

Output:



All Rows Fetched: 1 in 0.003 seconds

	COURSE_ID	STD_CAPACITY	SEATS_LEFT
1	C03	2	3

This displays the available seats and student capacity for the course C03

DATABASE SECURITY:

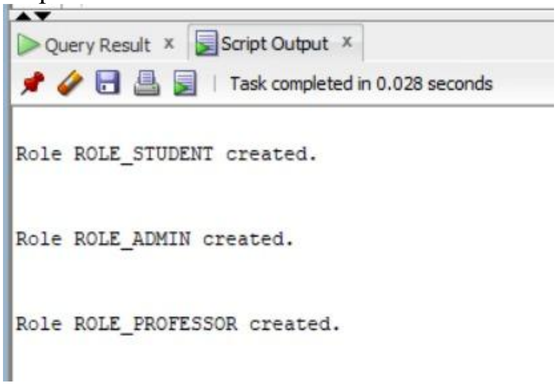
Define roles:

Admin → Full privileges.
Professor → UPDATE on Grades, SELECT on Roster
Student → SELECT on Transcript.

Creating Roles:

Query:

CREATE ROLE role_admin;
CREATE ROLE role_professor;
CREATE ROLE role_student;
Ouput:

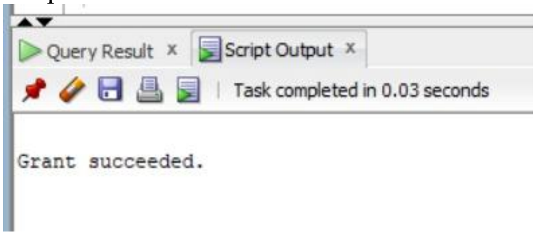


```
Role ROLE_STUDENT created.


Role ROLE_ADMIN created.


Role ROLE_PROFESSOR created.
```

Create roles for each tables Admin,Professor and Student

Privileges:

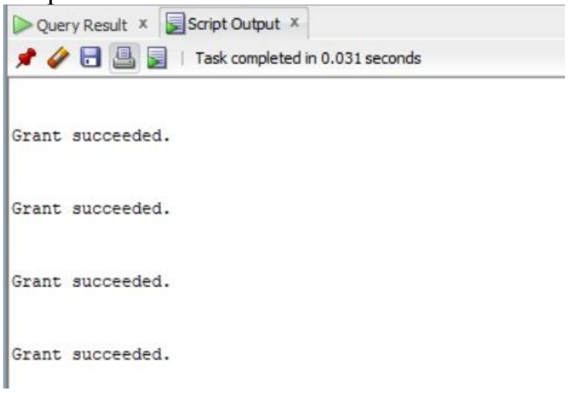
Query:
GRANT ALL PRIVILEGES TO role_admin;
Output:



```
Grant succeeded.
```

Provides all the access to the Admin ,like SELECT,UPDATE,ALTER,MODIFY,DELETE

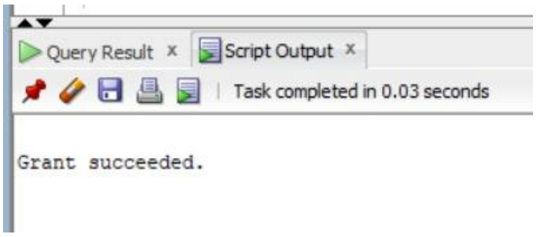
Query:
GRANT SELECT ON student TO role_professor;
GRANT SELECT ON enrollment TO role_professor;
GRANT SELECT ON courses TO role_professor;
GRANT UPDATE (grade_val,grade_point) ON grade TO role_professor;
Output:



```
Grant succeeded.


Grant succeeded.


Grant succeeded.


Grant succeeded.
```

Provides view only option for tables Student,Enrollment, and courses ,but professor have access to modify grade value and point

Query:
GRANT SELECT ON student_transcript TO student_role;
Output:



```
Task completed in 0.03 seconds

Grant succeeded.
```

Provides on read only option for table Student

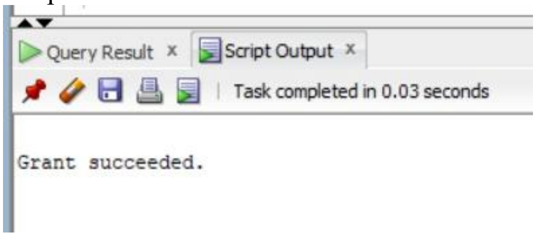
TASKS:

Apply RBAC with GRANT / REVOKE

Query:

```
GRANT ALL PRIVILEGES TO role_admin;
```

Output:



```
Task completed in 0.03 seconds

Grant succeeded.
```

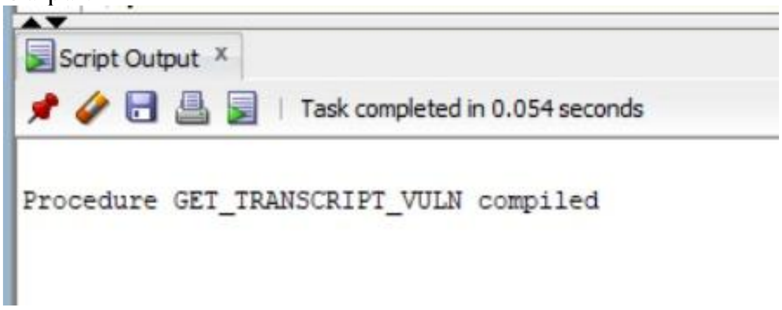
Provides all the access to the Admin ,like SELECT,UPDATE,ALTER,MODIFY,DELETE

Demonstrate a vulnerable SQL Injection and a secure fix.

Query:

```
CREATE OR REPLACE PROCEDURE get_transcript_vuln( p_std_id IN VARCHAR2) AS v_sql VARCHAR2(4000);
v_count NUMBER;
BEGIN
v_sql := 'SELECT COUNT(*) FROM Student_Transcript ' || 'WHERE Std_id = '''' || p_std_id || '''';
DBMS_OUTPUT.PUT_LINE('Running SQL: ' || v_sql);
EXECUTE IMMEDIATE v_sql INTO v_count;
DBMS_OUTPUT.PUT_LINE('Rows returned for this condition = ' || v_count);
END;
/
```

Output:



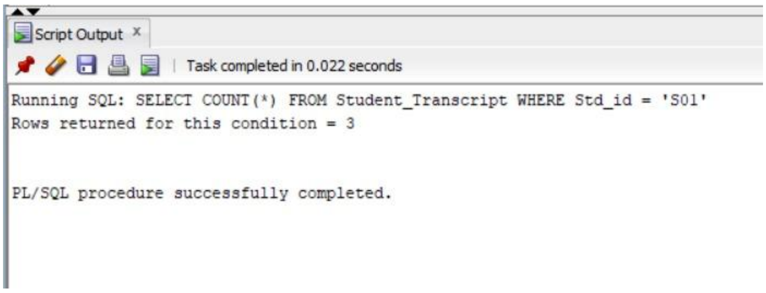
```
Task completed in 0.054 seconds

Procedure GET_TRANSCRIPT_VULN compiled
```

The vulnerability procedure has create successfully and it ready to sql injection

Query:

```
BEGIN
   get_transcript_vuln('S01');
END;
/
```

Output:

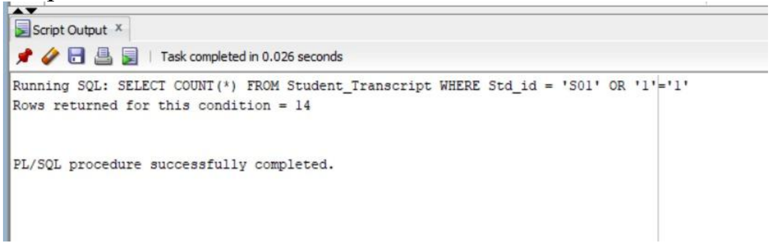


```
Running SQL: SELECT COUNT(*) FROM Student_Transcript WHERE Std_id = 'S01'
Rows returned for this condition = 3


PL/SQL procedure successfully completed.
```

Query:

```
BEGIN
    get_transcript_vuln('S01" OR "1"="1');
END;
/
```

Output:

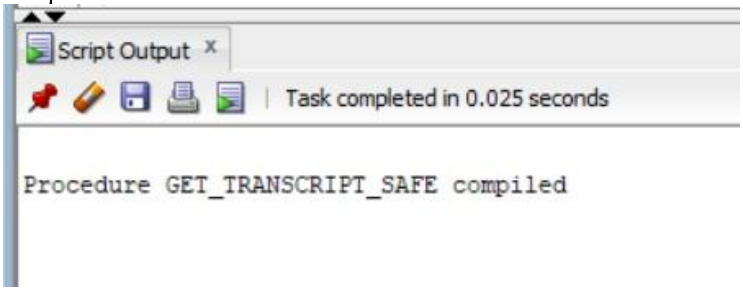


```
Running SQL: SELECT COUNT(*) FROM Student_Transcript WHERE Std_id = 'S01' OR '1'='1'
Rows returned for this condition = 14


PL/SQL procedure successfully completed.
```

Query:

```
CREATE OR REPLACE PROCEDURE get_transcript_safe(p_std_id IN VARCHAR2) AS v_count NUMBER;
BEGIN
SELECT COUNT(*) INTO v_count FROM Student_Transcript WHERE Std_id = p_std_id;
DBMS_OUTPUT.PUT_LINE('Safe rows for id = ' || p_std_id || ' = ' || v_count);
END;
/
```

Output:

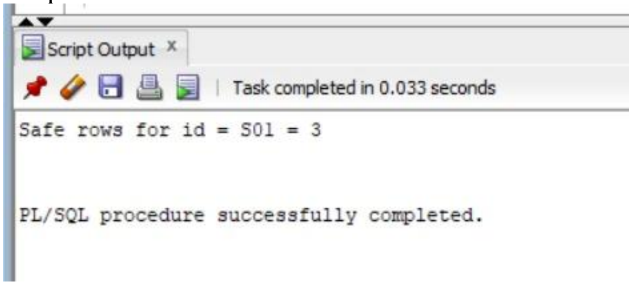


```
Procedure GET_TRANSCRIPT_SAFE compiled
```

Query:

```
BEGIN
    get_transcript_safe('S01');
END;
/
```

Output:

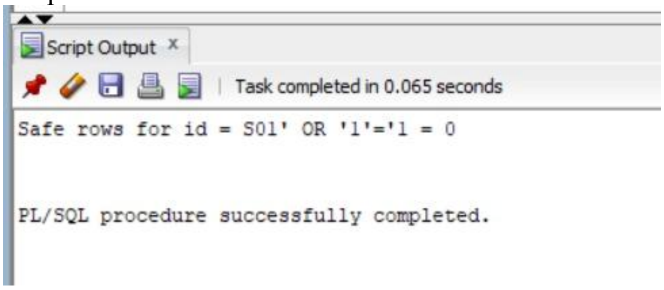


```
Safe rows for id = S01 = 3


PL/SQL procedure successfully completed.
```

Query:

```
BEGIN
    get_transcript_safe('S01" OR "1"="1');
END;
/
```

Output:



```
Safe rows for id = S01' OR '1'='1 = 0


PL/SQL procedure successfully completed.
```