ONLINE EDUCATION

Final Database Design Project Report, Fall 2016

BY:

Akhilesh Joshi: adj160230

Ankita Ahir: axa165030

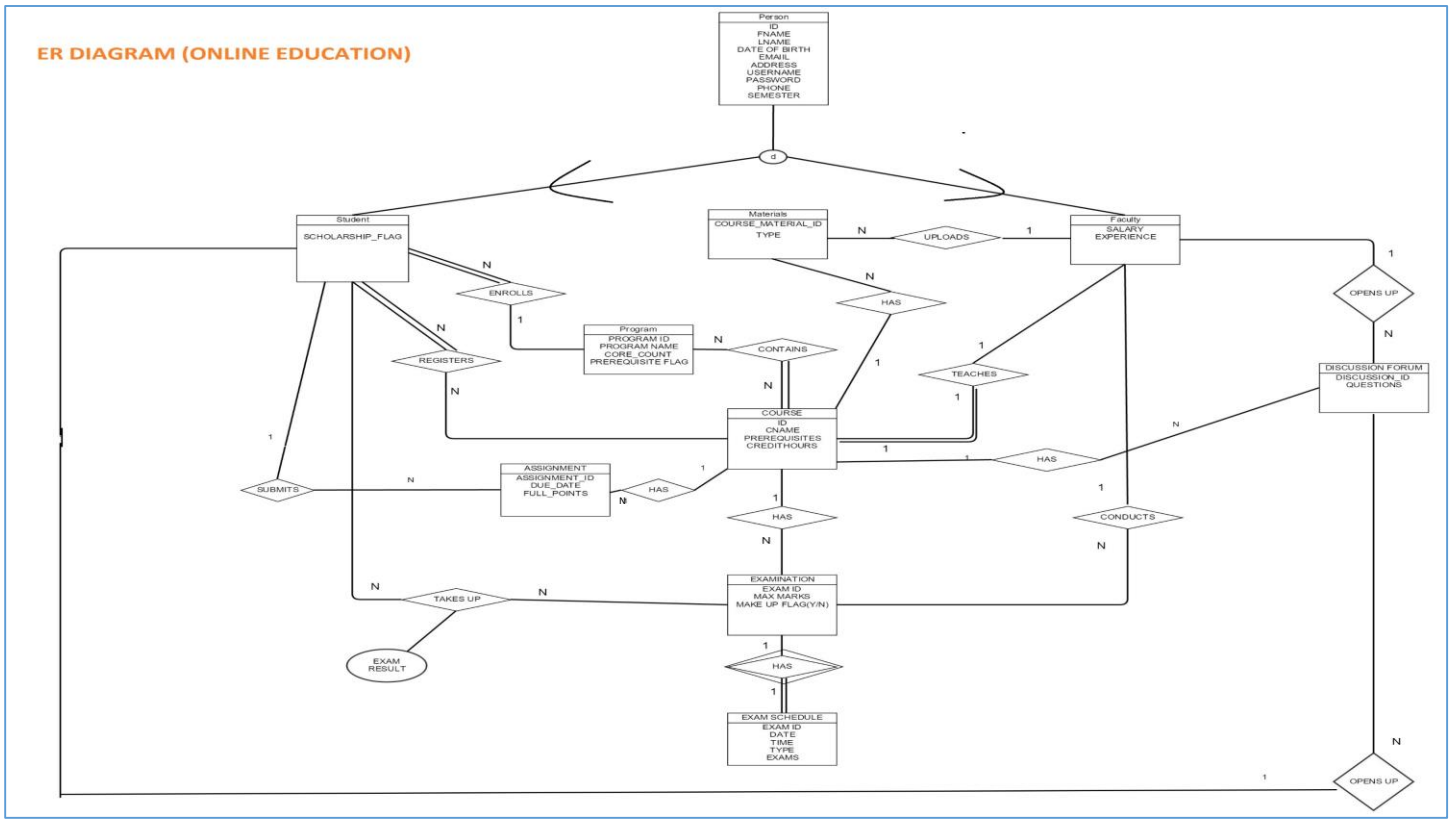
CONTENTS

1. REQUIREMENT ANALYSIS

The online classroom is a teaching and learning environment located within a computer-mediated communication system. The objective of online classroom is to improve access to advanced educational experiences by allowing students and instructors to participate in remote learning communities using personal computers at home or at work; and to improve the quality and effectiveness of education by using the computer to support a collaborative learning process. Collaborative learning is a learning process that emphasizes group or cooperative efforts among faculty and students, active participation, and interaction on the part of both students and instructors, and new knowledge that emerges from an active dialog among those who are sharing ideas and information.

- The system maintains the entity Person which maintains the information of all users with ID as the primary key. The entity of person is specialized into Student and Faculty. Faculty has additional attribute of experience and salary and student has an additional Scholarship Flag.
- A faculty teaches courses which are enrolled by the students. Faculty is also responsible for uploading the course material which can be referred by the students
- The Student enrolls to a program which contains many courses. The attributes of the Program include program_id which uniquely identifies the program, program name, the number of core subject present in the programs, does the program need any pre-requisite to be completed before registration.
- A program can have many courses. The attributes defined for course includes the course_id which uniquely identifies the course. Along with the Id it also has course name, Information about the credit hours offered by the course and the pre requisites which are needed to be completed before registration of the course.
- Each Course has assignment which needs to be completed by the students who enroll for the course.
- Each course has multiple exams which is conducted by the faculty and taken up by the Students. The attributes of exam contains the unique Exam_ID and maximum marks of the exam and whether the exam allows any make-up re-exam or not. Grades are maintained for each exam taken up by the student.
- Each Exam has an exam schedule which gives information about the exam. It stores the information like exam date, time, type of the exam.
- The system also has the facility such that the student can discuss the topics of the courses with the faculty through the Discussion Forums. The attributes are ID which uniquely identifies the discussion forum and the question asked by the students in the discussion forum

2. ENTITY RELATIONSHIP DIAGRAM



3. Cardinalities

a. two one-to-one binary relationships

- A faculty teaches to one Course.
- An Exams can have one exam Schedule

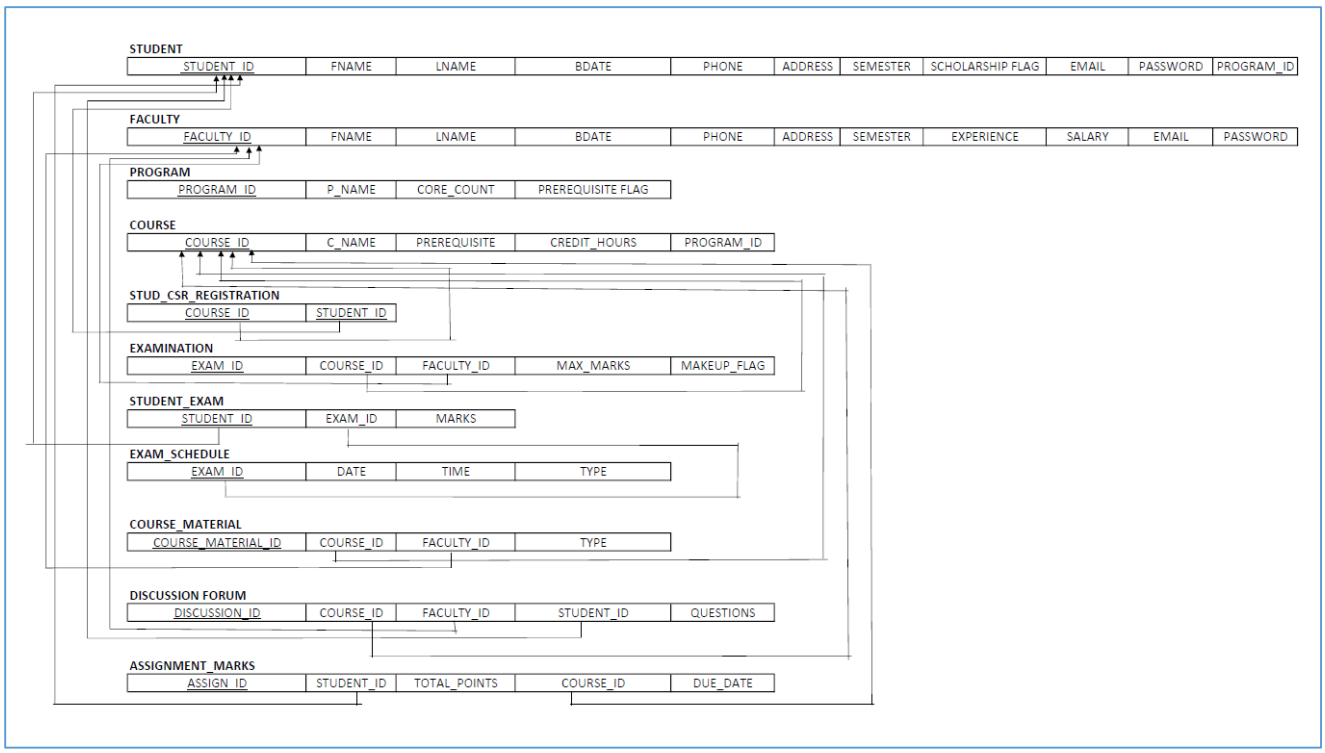
b. two one-to-many binary relationships

- A course can have many discussion Forums
- A Faculty can conduct N exams.
- A course can have multiple assignments
- A student can submit N assignments.
- A program can have N Students.
- A course can have multiple Course material.
- A course can have multiple Exam

c. two many-to-many binary relationships

- Multiple students can enroll to multiple course
- Multiple program can have multiple courses.

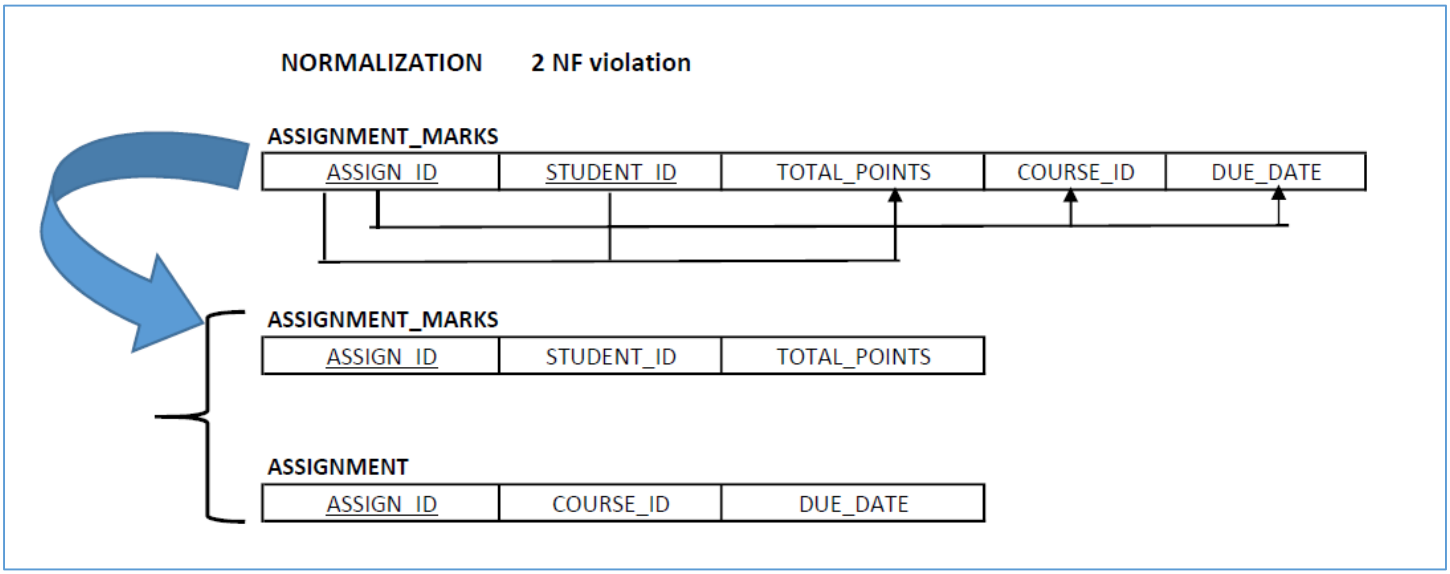
4. Mapping ER diagram into relational schema (Before Normalization)



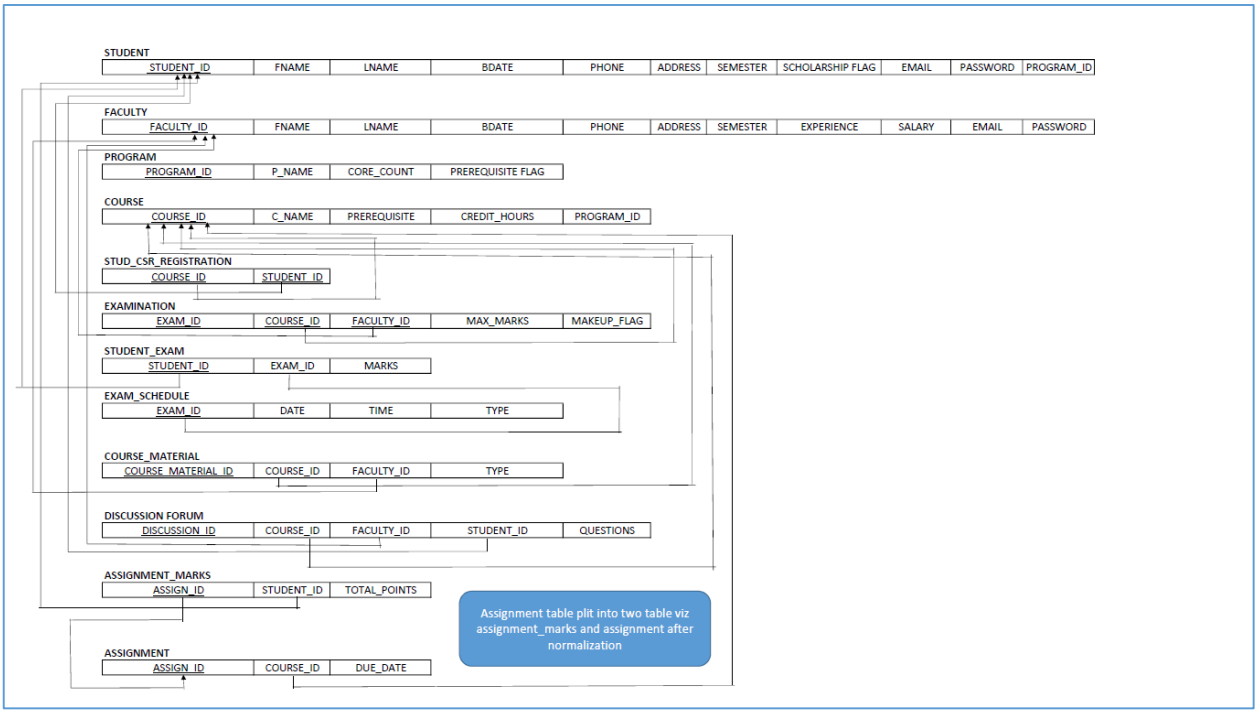
5. FUNCTIONAL DEPENDENCIES

student_id --> fname,lname,bdate,phone,address,semester,scholarship flag,email,password
student_id --> program_id
faculty_id --> fname,lname,bdate,phone,address,semester,experience,salary,email,password
program_id --> p_name,core_count,prerequisite flag
course_id --> prerequisite,credit_hours
exam_id --> max_marks
student_id,exam_id --> marks
exam_id --> date,time,type
course_material_id --> type
assign_id,student_id --> total_points
assign_id --> course_id,due_date

DATABASE NORMALIZATION RULE ILLUSTRATED



6. FINAL RELATIONAL SCHEMA AFTER NORMALIZATION



7. SQL QUERIES

```
CREATE TABLE PROGRAM(
PROGRAM_ID INTEGER ,
PNAME VARCHAR(50) ,
CORE_COUNT INTEGER,
PREREQ_FLAG INT,
PRIMARY KEY (PROGRAM_ID),
CHECK (PREREQ_FLAG=0 or PREREQ_FLAG=1)
);

CREATE TABLE STUDENT
(
STUDENT_ID      INT                     NOT NULL,
FNAME           VARCHAR (15)            NOT NULL,
LNAME           VARCHAR (15)            NOT NULL,
PROGRAM_ID      INT                     NOT NULL,
BDATE           DATE,
PHONE   varchar2 (15),
ADDRESS         VARCHAR2(50),
SEMESTER VARCHAR2(10),
SCHOLARSHIP_FLAG INT ,
EMAIL           VARCHAR (50)            NOT NULL,
PASSWRD         VARCHAR (20)            NOT NULL,
PRIMARY KEY (STUDENT_ID),
FOREIGN KEY (PROGRAM_ID) REFERENCES PROGRAM
(PROGRAM_ID) ON DELETE CASCADE,
CHECK (SCHOLARSHIP_FLAG=0 or SCHOLARSHIP_FLAG=1)
);

CREATE TABLE FACULTY
(
FACULTY_ID      INT NOT NULL,
FNAME           VARCHAR (30)    NOT NULL,
LNAME           VARCHAR (30)    NOT NULL,
BDATE           DATE,
PHONE   varchar2 (15),
ADDRESS         VARCHAR2(50),
SEMESTER VARCHAR2(10),
EXPERIENCE INT,
SALARY INT,
EMAIL   VARCHAR (30)    NOT NULL,
PASSWRD   VARCHAR (15)  NOT NULL,
CHECK (EXPERIENCE>2) ,
PRIMARY KEY (FACULTY_ID)
);

CREATE TABLE COURSE
(
COURSE_ID INT NOT NULL,
CNAME   VARCHAR (30)    NOT NULL,
PREREQUISITE VARCHAR (50) NOT NULL,
PROGRAM_ID INT NOT NULL,
CREDITHOURS INT ,
PRIMARY KEY (COURSE_ID),
FOREIGN KEY (PROGRAM_ID) REFERENCES PROGRAM
(PROGRAM_ID) ON DELETE CASCADE
);

CREATE TABLE STUD_CSR_REGISTRATION
(
COURSE_ID       INT                     NOT NULL,
STUDENT_ID      INT                     NOT NULL,
FOREIGN KEY (COURSE_ID) REFERENCES COURSE
(COURSE_ID)ON DELETE CASCADE,
FOREIGN KEY (STUDENT_ID) REFERENCES STUDENT
(STUDENT_ID)ON DELETE CASCADE
);

CREATE TABLE EXAMINATION
(
EXAM_ID         INT                     NOT NULL,
FACULTY_ID      INT                     NOT NULL,
COURSE_ID       INT                     NOT NULL,
PRIMARY KEY (EXAM_ID),
FOREIGN KEY (FACULTY_ID) REFERENCES FACULTY
(FACULTY_ID)ON DELETE CASCADE,
FOREIGN KEY (COURSE_ID) REFERENCES COURSE
(COURSE_ID) ON DELETE CASCADE
);


CREATE TABLE STUDENT_EXAM
(
STUDENT_ID      INT                     NOT NULL,
EXAM_ID         INT                     NOT NULL,
MARKS   INT                             NOT NULL,
FOREIGN KEY (STUDENT_ID) REFERENCES STUDENT
(STUDENT_ID)ON DELETE CASCADE,
FOREIGN KEY (EXAM_ID) REFERENCES EXAMINATION
(EXAM_ID)ON DELETE CASCADE
);


CREATE TABLE EXAM_SCHEDULE
(
EXAM_ID INT NOT NULL,
EXAM_DATE DATE,
```

```
EXAM_TYPE VARCHAR2(50),
FOREIGN KEY (EXAM_ID) REFERENCES EXAMINATION
(EXAM_ID) ON DELETE CASCADE
);

CREATE TABLE COURSE_MATERIAL
(
COURSE_MATERIAL_ID INT NOT NULL,
COURSE_ID INT NOT NULL,
FACULTY_ID INT   NOT NULL,
COURSE_MATERIAL_TYPE VARCHAR2(50),
PRIMARY KEY (COURSE_MATERIAL_ID),
FOREIGN KEY (COURSE_ID) REFERENCES COURSE
(COURSE_ID)ON DELETE CASCADE,
FOREIGN KEY (FACULTY_ID) REFERENCES FACULTY
(FACULTY_ID)ON DELETE CASCADE
);

CREATE TABLE DISCUSSION_FORUM
(
DISCUSSION_ID    INT NOT NULL,
COURSE_ID   INT NOT NULL,
FACULTY_ID INT    NOT NULL,
STUDENT_ID INT   NOT NULL,
TOPICS VARCHAR2(150),
PRIMARY KEY (DISCUSSION_ID),
FOREIGN KEY (COURSE_ID) REFERENCES COURSE
(COURSE_ID) ON DELETE CASCADE,
FOREIGN KEY (FACULTY_ID) REFERENCES FACULTY
(FACULTY_ID) ON DELETE CASCADE,
FOREIGN KEY (STUDENT_ID) REFERENCES STUDENT
(STUDENT_ID) ON DELETE CASCADE
);

CREATE TABLE ASSIGNMENT
(
ASSIGNMENT_ID  INT NOT NULL,
COURSE_ID INT NOT NULL,
DUE_DATE DATE,

PRIMARY KEY (ASSIGNMENT_ID),
FOREIGN KEY (COURSE_ID) REFERENCES COURSE
(COURSE_ID) ON DELETE CASCADE
);

CREATE TABLE ASSIGNMENT_MARKS
(
ASSIGNMENT_ID  INT NOT NULL,
STUDENT_ID INT  NOT NULL,
TOTAL_POINTS INT ,
FOREIGN KEY (STUDENT_ID) REFERENCES STUDENT
(STUDENT_ID)ON DELETE CASCADE,
FOREIGN KEY (ASSIGNMENT_ID) REFERENCES ASSIGNMENT
(ASSIGNMENT_ID)ON DELETE CASCADE
);
```

8. PROCEDURES

1. finding students who have more than 9 credit hours

```
create or replace procedure stud_credit(e out varchar2)
is
cursor c is
select s.fname as first_name , s.lname as last_name
from student s
where student_id in (
select scr.student_id as stud_id
from course c,stud_csr_registration scr
where  scr.course_id=c.course_id
group by scr.student_id
having sum(c.credithours) > 9 ) ;
cur_row c%rowtype;
begin
if not c%isopen then open c;
end if;

dbms_output.put_line('procedure for finding students who
have more than 9 credit hours' );
dbms_output.put_line(chr(10));
dbms_output.put_line('list of student(s) is as following' );
loop
fetch c into cur_row;
exit when c%notfound;
dbms_output.put_line( cur_row.first_name || ' ' ||
cur_row.last_name );
end loop;
exception
when others then
e := sqlerrm;
end ;
```

2. fixing credit hours

```
create or replace procedure update_credit(e out varchar2)
is
begin
update course set  credithours = 9
```

```
where course_id in
(select course_id
from stud_csr_registration scr
where scr.student_id in
(
select scr.student_id as stud_id
from course c,stud_csr_registration scr
where  scr.course_id=c.course_id
group by scr.student_id
having sum(c.credithours) > 9));
commit;

dbms_output.put_line(chr(10));
dbms_output.put_line('procedure for updating credit hours'
); dbms_output.put_line(chr(10) );
dbms_output.put_line('students who got more than 9 credit
hours are restricted to 9 credit hours' );
exception
when others then e := sqlerrm; end ; set serveroutput on;
declare
error varchar2(2000);
begin
stud_credit(error);update_credit(error);
if error is null then
dbms_output.put_line('no error encountered in processing');
else
dbms_output.put_line('error message : ' || error);
end if;
end;
```

3.students who have given exam on 1st november,2016 and scored above 50 marks

```
create or replace procedure student_marks as
cursor studentlist is
select fname,lname,marks,exam_date from
student,student_exam,exam_schedule
where student.student_id=student_exam.student_id and
student_exam.exam_id=exam_schedule.exam_id and
marks>50
and exam_date='01-nov-16';
student_data studentlist%rowtype;
begin
open studentlist;
loop
fetch studentlist into student_data;
exit when studentlist%notfound;
dbms_output.put_line ('student:'|| student_data.fname||'
'||student_data.lname||' has marks '|| student_data.marks
||' who has given exam on '|| student_data.exam_date );
end loop;
close studentlist;
end;
 /
begin
student_marks();
end;
```

9. TRIGGERS

1. deciding grades of students based on their marks

```
create or replace trigger student_grade
before insert or update of marks on student_exam
for each row
declare ispresent number;
begin
case
when inserting then
if :new.marks>70 then
insert into student_grades values(:new.student_id,'a');
end if;
if :new.marks<70 and :new.marks>50 then
insert into student_grades values(:new.student_id,'b');
end if;
if :new.marks<50 then
insert into student_grades values(:new.student_id,'c');
end if;

when updating then
select count(*) into ispresent from student_grades where
student_id=:new.student_id;
if(ispresent=0)then
if :new.marks>70 then
insert into student_grades values(:new.student_id,'a');   end
if;
if :new.marks<70 and :new.marks>50 then
insert into student_grades values(:new.student_id,'b');  end
if;
if :new.marks<50 then
insert into student_grades values(:new.student_id,'c');
end if;   end if;
if(ispresent>0)then if :new.marks>70 then
update student_grades set grade='a' where
student_id=:new.student_id;   end if;
if :new.marks<70 and :new.marks>50 then
update student_grades set grade='b' where
student_id=:new.student_id;
end if;
```

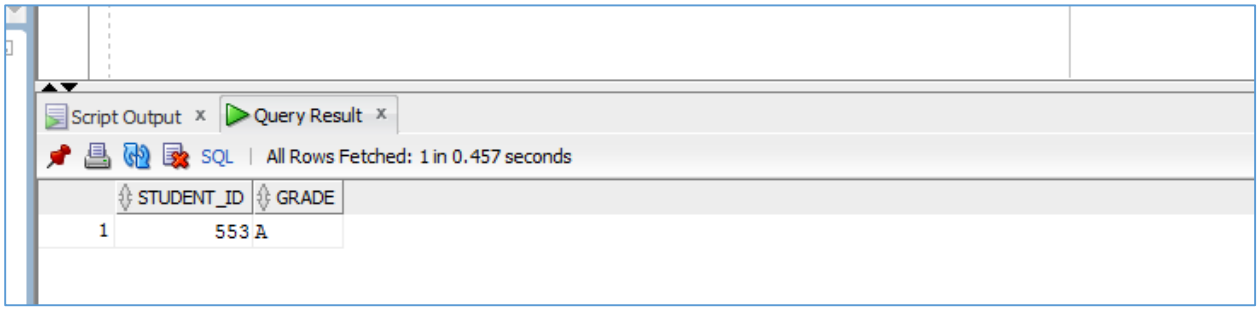
```
if :new.marks<50  then
update student_grades set grade='c' where
student_id=:new.student_id;
end if;
end if;
end case;
end;
```

2. deciding range of salary for faculty

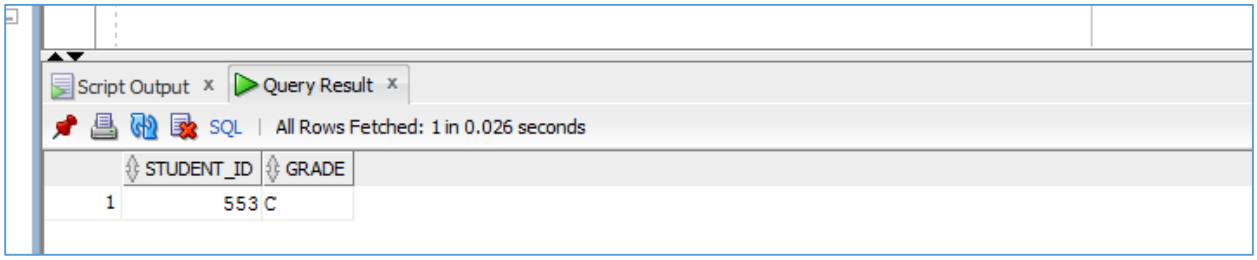
```
CREATE OR REPLACE TRIGGER
SALARY_RANGE_FACULTY
BEFORE INSERT OR UPDATE OF SALARY ON FACULTY
FOR EACH ROW
DECLARE ISPRESENT NUMBER;
BEGIN
CASE
WHEN INSERTING THEN
IF :NEW.SALARY>60000 THEN
INSERT INTO FACULTY_SALARY
VALUES(:NEW.FACULTY_ID,'HIGH');
END IF;
IF :NEW.SALARY<60000 AND :NEW.SALARY>40000
THEN
INSERT INTO FACULTY_SALARY
VALUES(:NEW.FACULTY_ID,'MEDIUM');
END IF;
IF :NEW.SALARY<40000 THEN
INSERT INTO FACULTY_SALARY
VALUES(:NEW.FACULTY_ID,'LOW');
END IF;
WHEN UPDATING THEN
SELECT COUNT(*) INTO ISPRESENT FROM
FACULTY_SALARY WHERE
FACULTY_ID=:NEW.FACULTY_ID;
IF(ISPRESENT=0)THEN
IF :NEW.SALARY>60000 THEN
INSERT INTO FACULTY_SALARY
VALUES(:NEW.FACULTY_ID,'HIGH');
END IF;
IF :NEW.SALARY<60000 AND :NEW.SALARY>40000
THEN
INSERT INTO FACULTY_SALARY
VALUES(:NEW.FACULTY_ID,'MEDIUM');
END IF;
IF :NEW.SALARY<40000 THEN
INSERT INTO FACULTY_SALARY
VALUES(:NEW.FACULTY_ID,'LOW');
END IF;
END IF;
IF(ISPRESENT>0)THEN
IF :NEW.SALARY>60000 THEN
UPDATE FACULTY_SALARY SET SALARY='HIGH' WHERE
FACULTY_ID=:NEW.FACULTY_ID;
END IF;
IF :NEW.SALARY<60000 AND :NEW.SALARY>40000
THEN
UPDATE FACULTY_SALARY SET SALARY='MEDIUM'
WHERE FACULTY_ID=:NEW.FACULTY_ID;
END IF;
IF :NEW.SALARY<40000  THEN
UPDATE FACULTY_SALARY SET SALARY='LOW' WHERE
FACULTY_ID=:NEW.FACULTY_ID;
END IF;
END IF;
END CASE;
END;
```

TRIGGER 1 (Screen Shots)

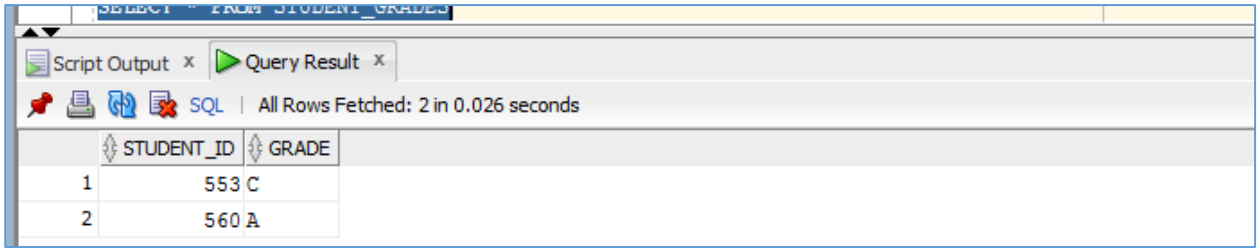
UPDATING MARKS TO 71 FOR STUDENT WITH STUDENT_ID=553
UPDATE STUDENT_EXAM SET MARKS = 71
WHERE STUDENT_ID=553;



2. UPDATING MARKS TO 49 FOR STUDENT WITH STUDENT_ID=553
UPDATE STUDENT_EXAM SET MARKS = 49
WHERE STUDENT_ID=553;



3. INSERTING A ROW WITH MARKS 91
INSERT INTO STUDENT_EXAM
VALUES(560,7,91)

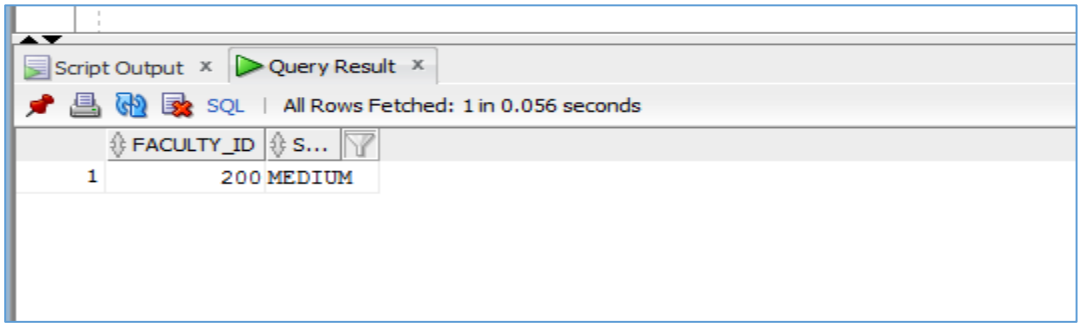


TRIGGER 2 (Screen Shots)

1.UPDATING SALARY TO 45000 FOR FACULTY WITH FACULTY_ID=200

UPDATE FACULTY SET SALARY = 45000

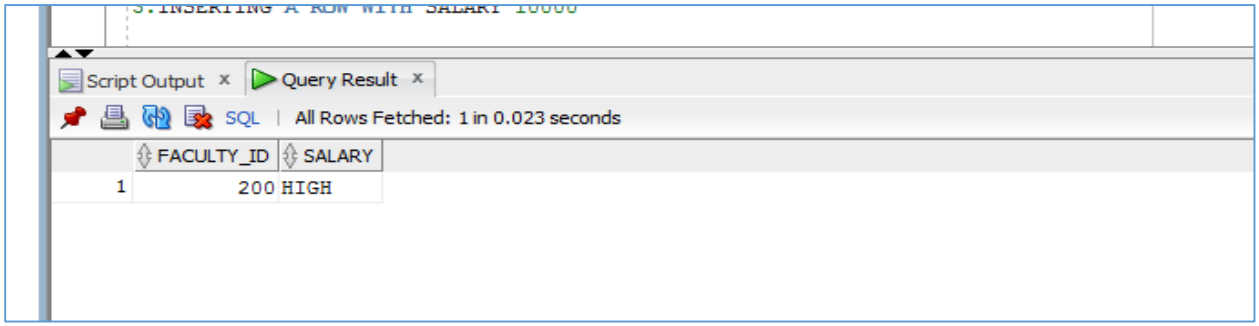
WHERE FACULTY_ID=200;



2. UPDATING SALARY TO 65000 FOR EMPLOYEE WITH FACULTY_ID=200

UPDATE FACULTY SET SALARY = 65000

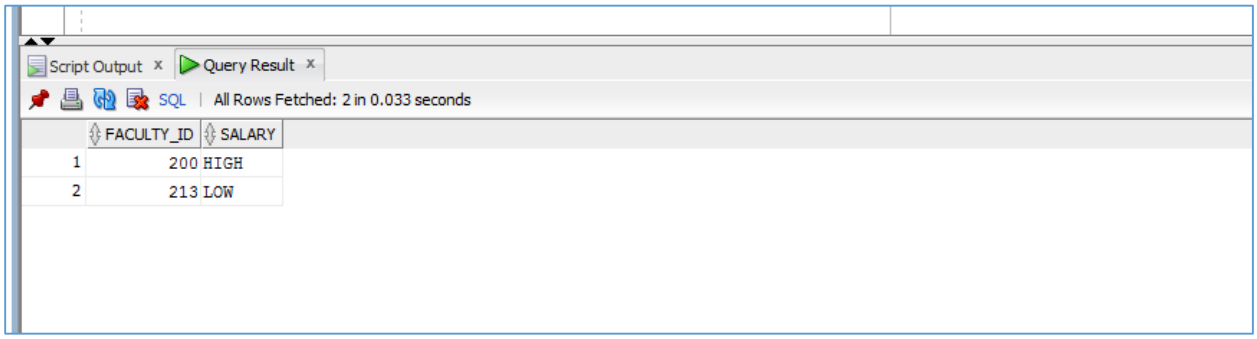
WHERE FACULTY_ID=200;



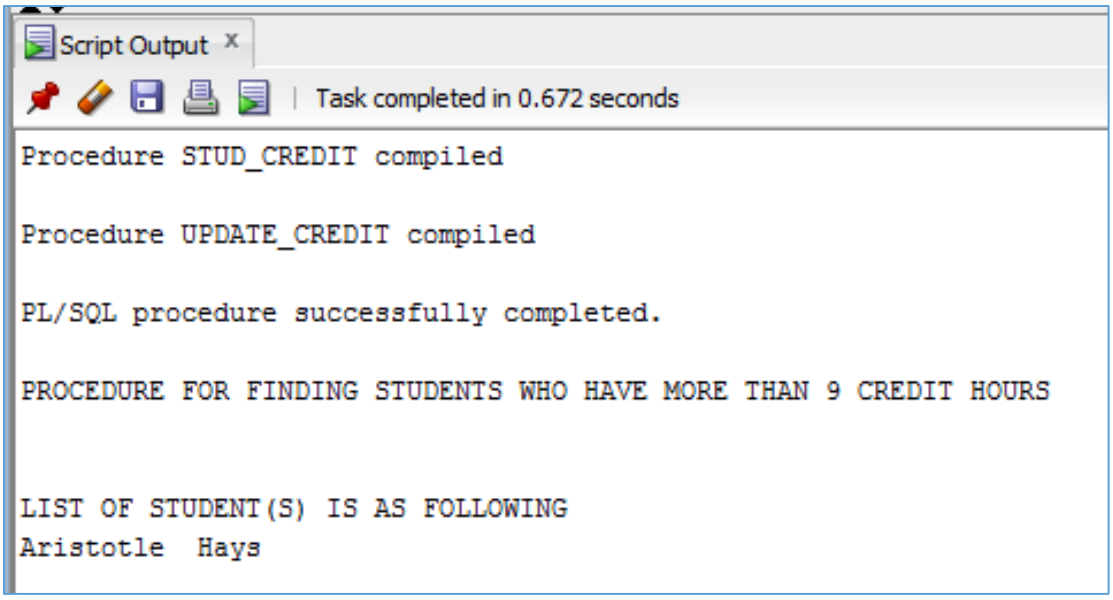
3.INSERTING A ROW WITH SALARY 10000

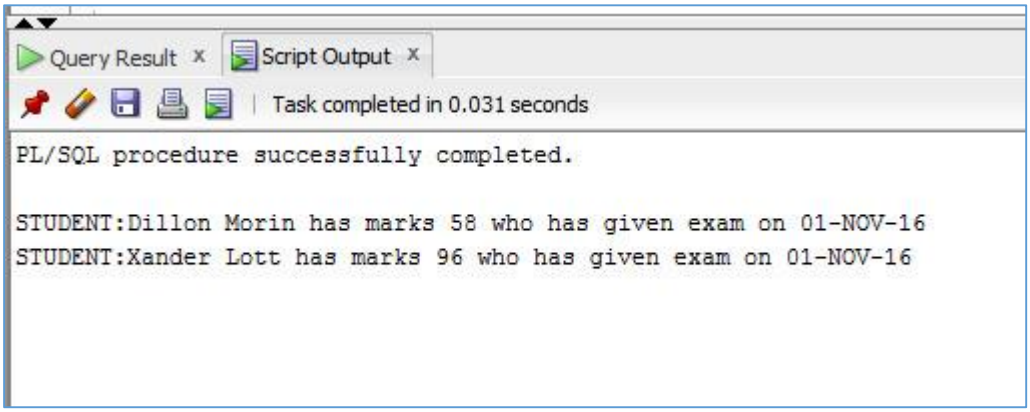
INSERT INTO FACULTY

VALUES(213,'ANKITA','AHIR','04-AUG-1991','l-972-762-6456','7720 MCCALLUM BLVD',1,4,1000,'ahirankita@hmail.com','1235')



Procedure (Screen Shots)





10. DEFINING RULES FOR YOUR SYSTEM AND THEIR IMPLEMENTATION USING CHECK CONSTRAINT/TRIGGERS

1. Every Faculty should have 2 years of experience:

- This is achieved by applying the check constraint of the experience column of the faculty.

2. The System maintains the grades of the Students base of the marks he She gets

- Range 70-100 is 'A' Grade.
- Range 50-69 is 'B' Grade.
- Range 0-49 is 'C' Grade.
- This is achieved by the trigger 'Student_Grade'

3. The System maintains the salary range of the faculty.

- Range >=6000 is 'High'.
- Range 4000-5999 is 'Medium'.
- Range 0-3999 is 'Low'
- This is achieved by the trigger 'SALARY_RANGE_FACULTY'