****

**American International University- Bangladesh**

**Faculty of Science and Technology**

**Department of Computer Science**

**Title: FINAL TERM PROJECT**

[**ADVANCE DATABASE MANAGEMENT SYSTEM**](https://portal.aiub.edu/Student/Section?q=H1lAiISjkj009O5Iwo%2FOWw%3D%3D)

**Section: A**

**Final Term Project**

**FALL 22-23**

|  |  |
| --- | --- |
| **Group Member Name** | **ID** |
| **Abid Hassan Emon** | **19-41207-2** |
| **Rubaba Binte Rahman** | **20-42182-1** |
| **Nubah Surrat Aparazita** | **20-43023-1** |
| **Sarzila Shahrin Jisha** | **20-42526-1** |

**Course Teacher: REZWAN AHMED**

**System summary:**

An online judge database system is a management system that acts for compiling, executing and evaluating programs submitted by users. Each member needs an account to login. A new member has to register an account. Admin will accept registration process. And then the member needs to log in their account, which is verified by system. If the entered password is incorrect, system requests for the password and the member enters password again. If the entered password is correct, the member gets access in this system. After login, a member can solve multiple problems. A member can select the problems from the problem sets and category-wise. There will be some problem setters, who will set, update, delete and check the problems and submissions. Each problem setter can set multiple problems. Different problem-set and categories have different problems. A member can submit multiple submissions. A problem has been submitted by many members. Finally, if the admin accepts the submission, that problem will be accepted as solved.

**Table Creation:**

CREATE TABLE member

(m\_id NUMBER(2) CONSTRAINT PK\_member PRIMARY KEY,

m\_language VARCHAR2(14) ,

m\_name VARCHAR2(17),

join\_date DATE) ;

CREATE TABLE U\_login

(login\_id NUMBER(2) CONSTRAINT PK\_login PRIMARY KEY,

user\_name VARCHAR2(14) ,

password VARCHAR2(17),

role VARCHAR2(14),

m\_id NUMBER(2) CONSTRAINT FK\_m\_id REFERENCES member ) ;

//sequence for member\_id

CREATE SEQUENCE m\_id\_seq

INCREMENT BY 1;

//sequence for login id

CREATE SEQUENCE l\_id\_seq

INCREMENT BY 1;

CREATE TABLE Category

(C\_id NUMBER(2) CONSTRAINT PK\_Category PRIMARY KEY,

Cname VARCHAR2(14)

) ;

CREATE TABLE PROBLEM\_SET(VOLUME\_NO NUMBER) ;

//sequence for volume

CREATE SEQUENCE VOL\_seq

INCREMENT BY 1;

CREATE TABLE PROBLEM\_SETTER (

PS\_id NUMBER(2) CONSTRAINT PK\_PROBLEM\_SETTER PRIMARY KEY,

PS\_NAME VARCHAR2(14)

) ;

CREATE TABLE problem

(p\_id NUMBER(2) CONSTRAINT PK\_problem PRIMARY KEY,

volume\_no number,

PS\_id NUMBER(2) CONSTRAINT FK\_ps\_id REFERENCES Problem\_setter ,

c\_id number(2) CONSTRAINT FK\_c\_id REFERENCES category,

P\_assigned\_date date, p\_NAME VARCHAR2(50)

) ;

CREATE TABLE submission

(s\_id NUMBER(2) CONSTRAINT PK\_submission PRIMARY KEY,

M\_id number(2),

p\_id number(2) CONSTRAINT FK\_p\_id REFERENCES problem,

s\_date date,

status number(2)

) ;

**SQL Queries:**

1.Find the details of member whose U\_name='' and pas=''.

select \* from U\_login l,member m where m.m\_id=l.m\_id and l.U\_name='abid' and l.U\_pass='1234';

2.select All problem setted by SARZILA JISHA.

select p.P\_name,ps.Ps\_name from problem p,problem\_setter ps where p.ps\_id=ps.ps\_id and ps.PS\_name='SARZILA JISHA';

3.show details of the member who have maximum submissions.

select \* from member where M\_ID in (

select M\_ID from submission group by M\_ID having count(\*)=(

select max(count(\*)) from submission group by M\_ID));

4.Find the category have the most number on problem.

select cname from category where c\_id in(

select c\_id from problem group by c\_id having count(\*)=(

select max(count(\*)) from problem group by c\_id));

5.select problem setted after abids last submissions.

select \* from problem where p\_assigned\_date>(

select max(S\_date) from submission s ,member m where s.M\_id=m.M\_ID and m.M\_NAME='abid hassan') ;

6.Show the info of member and problem whose submissions are accepted.

select M.M\_ID, M.M\_NAME, S.P\_ID, P.P\_NAME FROM MEMBER M, SUBMISSION S, PROBLEM P

where M.M\_ID=S.M\_ID AND S.P\_ID = P.P\_ID AND S.STATUS = 1;

7.Find the problem setter and problem info which problem is not solved.

select p.p\_id,p.p\_name, ps.ps\_id, ps.ps\_name from problem p, problem\_setter ps,

submission s

where p.p\_id = s.p\_id and p.ps\_id = ps.ps\_id and s.status = 0

group by p.p\_id, p.p\_name, ps.ps\_id, ps.ps\_name;

8. Find the info of the problems and volume number and member name which is solved by C# or php and problem is accepted.

select p.p\_id,p.p\_name, ps.volume\_no,m.m\_name from submission s,problem p, problem\_set ps ,member m

where p.p\_id = s.p\_id and ps.VOLUME\_NO = p.VOLUME\_NO and s.m\_id=m.m\_id and m.M\_language in ('c#','php') and s.status=1;

9. Find the info of member who has successfully solved 'RUBABA RAHMAN's problem.

select m.m\_id, m.m\_name from member m, problem p, submission s, problem\_setter ps

where m.m\_id = s.m\_id and s.p\_id = p.p\_id and s.status = 1

and p.ps\_id = ps.ps\_id and ps.ps\_name = 'RUBABA RAHMAN';

10. count last years total monthly submission.

select to\_char(S\_DATE, 'Month') as MONTH,count(\*) as Total\_Submissions from submission

where extract(year from S\_DATE)=extract(year from sysdate)

group by to\_char(S\_DATE, 'Month');

plsql:

create or replace function valid\_category(PC\_ID category.C\_ID%type)

return boolean

is

x number(2);

begin

select count(\*) into x from category where C\_ID=PC\_ID;

if(x=1) then

return true;

else

return false;

end if;

END;

create or replace function valid\_PSetter(PPS\_ID problem\_setter.PS\_ID%type)

return boolean

is

y number(2);

begin

select count(\*) into y from problem\_setter where ps\_ID=pps\_ID;

if(y=1) then

return true;

else

return false;

end if;

END;

create or replace function valid\_volume(pvolume\_no problem\_set.volume\_no%type)

return boolean

is

z number(2);

begin

select count(\*) into z from problem\_set where volume\_no=pvolume\_no;

if(z=1) then

return true;

else

return false;

end if;

END;

create or replace procedure Add\_problem(pp\_name in problem.p\_name%type,pps\_ID in problem.PS\_ID%type,pc\_ID in problem.C\_ID%type,pvolume\_no in problem.volume\_no%type)

is

begin

if(valid\_category(pc\_ID) and valid\_PSetter(PPS\_ID) and valid\_volume(pvolume\_no) )

then

insert into problem values(HR.p\_id\_seq.nextval,pvolume\_no,pps\_ID,pc\_ID,to\_date(sysdate,'dd-mm-yyyy'),pp\_name);

dbms\_output.put\_line('successfull');

else

raise\_application\_error(-20226,'Invalid category or problem or volume ID');

end if;

end;

begin

Add\_problem('abcd',3,3,4);

end;

select \* from problem\_set;

select \* from problem;

CREATE SEQUENCE p\_id\_seq

INCREMENT BY 1;

SELECT p\_id\_seq.NEXTVAL FROM DUAL;

SELECT sysdate FROM DUAL;

raise\_application\_error(-20226,'Invalid flight id');

create or replace function valid\_member(PM\_ID submission.M\_ID%type)

return boolean

is

p number(2);

begin

select count(\*) into p from submission where m\_id=PM\_ID;

if(p=1) then

return true;

else

return false;

end if;

END;

create or replace function valid\_problem(PP\_ID submission.M\_ID%type)

return boolean

is

o number(2);

begin

select count(\*) into o from submission where p\_id=PP\_ID;

if(o=1) then

return true;

else

return false;

end if;

END;

create or replace procedure submit\_problem(pp\_name in problem.p\_name%type,pps\_ID in problem.PS\_ID%type,pc\_ID in problem.C\_ID%type,pvolume\_no in problem.volume\_no%type)

is

begin

if(valid\_problem(5) and valid\_member(19))

then

--insert into problem values(HR.p\_id\_seq.nextval,pvolume\_no,pps\_ID,pc\_ID,to\_date(sysdate,'dd-mm-yyyy'),pp\_name);

dbms\_output.put\_line('successfull');

else

raise\_application\_error(-20228,'Invalid member or problem ID');

end if;

end;

begin

submit\_problem();

end;

1.

CREATE OR REPLACE TRIGGER update\_status

BEFORE INSERT OR UPDATE ON submission for each row

BEGIN

IF :new.status NOT IN(0,1)

then

raise\_application\_error(-20240,’status can be 0 or 1’);

END IF;

END;

2.

CREATE OR REPLACE TRIGGER update\_member

BEFORE UPDATE ON member for each row

BEGIN

IF :new.M\_language IS NULL

then

raise\_application\_error(-20241,'Must have to know a PL language');

END IF;

END;

3.

CREATE OR REPLACE TRIGGER update\_problem

BEFORE INSERT OR UPDATE ON submission for each row

BEGIN

IF :new.answer IS NULL

then

raise\_application\_error(-20241,'ANSWER BEFORE SUBMIT');

END IF;

END;

4.

CREATE OR REPLACE TRIGGER ADD\_problem

BEFORE INSERT OR UPDATE ON problem for each row

BEGIN

IF :new.ps\_id IS NULL

then

raise\_application\_error(-20241,'who is problem setter');

END IF;

END;