**CREATE STATEMENTS:**

1. CREATE TABLE ANSWERS (QUESTIONID NUMBER, ISRIGHT CHAR(1 BYTE), EXPLANATION VARCHAR2(500 BYTE), ANSWERID NUMBER(2,0), ANSWERTEXT VARCHAR2(500 BYTE) NOT NULL, SETID NUMBER(30,0), ADD CONSTRAINT ISRIGHT\_CHK CHECK (ISRIGHT IN ('Y','N')), ADD CONSTRAINT FK\_QUESTIONBANK\_ANSWERS FOREIGN KEY (QUESTIONID) REFERENCES QUESTIONBANK(QUESTIONID))

2. CREATE TABLE CHAPTER (TEXTBOOKISBN VARCHAR2(30 BYTE), CHAPTERID NUMBER(10,0), CHAPTERTITLE VARCHAR2(50 BYTE) NOT NULL, ADD CONSTRAINT PK\_CHAPTER PRIMARY KEY (TEXTBOOKISBN, CHAPTERID), ADD CONSTRAINT FK\_ISBN FOREIGN KEY (TEXTBOOKISBN) REFERENCES TEXTBOOKS(TEXTBOOKISBN))

3. CREATE TABLE COURSE (COURSEID NUMBER(10,0), COURSENAME VARCHAR2(100 BYTE) NOT NULL, COURSELEVEL VARCHAR2(100 BYTE), ADD CONSTRAINT PK\_COURSEID PRIMARY KEY (COURSEID), ADD CONSTRAINT CHECK\_LEVEL CHECK (COURSELEVEL IN ('UNDERGRADUATE', 'GRADUATE’)))

4. CREATE TABLE COURSEOFFERINGS (COURSETOKENID VARCHAR2(15 BYTE), COURSEID NUMBER(10,0), STARTDATE DATE, ENDDATE DATE, MAXIMUMENROLLMENTNUMBER NUMBER(5,0), ADD CONSTRAINT PK\_COURSEOFFERING PRIMARY KEY (COURSETOKENID), ADD CONSTRAINT FK\_COURSEOFFER\_COURSEID FOREIGN KEY (COURSEID) REFERENCES COURSE(COURSEID))

5. CREATE TABLE COURSETEACHING (USERID NUMBER(10,0), COURSEID NUMBER(10,0), COURSETOKENID VARCHAR2(15 BYTE), ROLEID NUMBER(2,0), ADD CONSTRAINT PK\_COURSETEACHING PRIMARY KEY (USERID, COURSEID, COURSETOKENID), ADD CONSTRAINT CHECK\_ROLEID CHECK (ROLEID IN (1,3)), ADD CONSTRAINT FK\_USERS\_COURSETEACHING FOREIGN KEY (USERID) REFERENCES USERS (USERID), ADD CONSTRAINT FK\_COURSE\_COURSETEACHING FOREIGN KEY (COURSEID) REFERENCES COURSE (COURSEID), ADD CONSTRAINT FK\_CO\_CT FOREIGN KEY (COURSETOKENID) REFERENCES COURSEOFFERINGS(COURSETOKENID))

6. CREATE TABLE COURSETEXTBOOKMAPPING (TEXTBOOKISBN VARCHAR2(30 BYTE) NOT NULL, COURSETOKENID VARCHAR2(30 BYTE) NOT NULL, CONSTRAINT COURSETEXTBOOKMAPPING\_PK PRIMARY KEY (TEXTBOOKISBN, COURSETOKENID), ADD CONSTRAINT FK\_TEXTBOOKISBN FOREIGN KEY (TEXTBOOKISBN) REFERENCES TEXTBOOKS(TEXTBOOKISBN), ADD CONSTRAINT FK\_COURSETOKEN FOREIGN KEY (COURSETOKENID) REFERENCES COURSEOFFERINGS (COURSETOKENID))

7. CREATE TABLE COURSETOPICS (COURSETOPICID NUMBER(10,0), COURSETOPICNAME VARCHAR2(100 BYTE) NOT NULL, COURSEID NUMBER(10,0), ADD CONSTRAINT PK\_COURSETOPICID PRIMARY KEY (COURSETOPICID), ADD CONSTRAINT FK\_COURSE\_COURSEID FOREIGN KEY (COURSEID) REFERENCES COURSE (COURSEID))

8. CREATE TABLE EXERCISELIST(EXERCISEID NUMBER, COURSETOPICID NUMBER NOT NULL, COURSETOKENID VARCHAR2(15 BYTE), ADD CONSTRAINT PK\_EL PRIMARY KEY (EXERCISEID, COURSETOKENID), ADD CONSTRAINT FK\_EXEID FOREIGN KEY (EXERCISEID) REFERENCES EXERCISES (EXERCISEID))

9. CREATE TABLE EXERCISEQUESTION (EXERCISEID NUMBER(10,0), QUESTIONID NUMBER(10,0), ADD CONSTRAINT PK\_EXERCISEQUESTION PRIMARY KEY (EXERCISEID, QUESTIONID), ADD CONSTRAINT FK\_EXERCISES\_EXERQUESTION FOREIGN KEY (EXERCISEID) REFERENCES EXERCISES(EXERCISEID), ADD CONSTRAINT FK\_QUESTIONBANK\_EXERQUESTION FOREIGN KEY(QUESTIONID) REFERENCES QUESTIONBANK(QUESTIONID))

10. CREATE TABLE EXERCISES (EXERCISEID NUMBER(10,0), DIFFICULTYLEVEL NUMBER(2,0) NOT NULL, NUMBEROFRETRIES NUMBER(2,0) NOT NULL, MARKSPERCORRECTANSWER NUMBER(2,0) NOT NULL, MARKSPERINCORRECTANSWER NUMBER(2,0) NOT NULL, STARTDATETIME DATE NOT NULL, ENDDATETIME DATE NOT NULL, SCORINGTYPE VARCHAR2(20 BYTE) NOT NULL, NUMBEROFQUESTIONS NUMBER NOT NULL, ADD CONSTRAINT PK\_EXERCISES PRIMARY KEY (EXERCISEID), ADD CONSTRAINT EXERCISES\_CHK1 CHECK (DIFFICULTYLEVEL IN (1,2,3,4,5,6)))

11. CREATE TABLE NOTIFICATIONS (NOTIFICATIONID NUMBER(3,0), NOTIFICATIONTEXT VARCHAR2(1000 BYTE), COURSETOKENID VARCHAR2(25 BYTE))

12. CREATE TABLE NOTIFICATIONUSERMAPPING (USERID NUMBER(5,0), NOTIFICATIONID NUMBER(2,0), READFLAG NUMBER, ADD CONSTRAINT CHECK\_0\_1 CHECK (READFLAG IN (0,1)))

13. CREATE TABLE QUESTIONANSWERMAPPING (EXERCISEID NUMBER(10,0), QUESTIONID NUMBER(10,0), ANSWERID NUMBER(2,0), ADD CONSTRAINT PK\_QUESTIONASNWERMAPPING PRIMARY KEY (EXERCISEID, QUESTIONID, ANSWERID, ADD CONSTRAINT FK\_EXERCISES\_QUESTIONANRMAP FOREIGN KEY (EXERCISEID) REFERENCES EXERCISES (EXERCISEID), ADD CONSTRAINT FK\_QUESTIONBANK\_QUESANSRMAP FOREIGN KEY (QUESTIONID) REFERENCES QUESTIONBANK (QUESTIONID))

14. CREATE TABLE QUESTIONBANK (QUESTIONID NUMBER(10,0), QUESTIONTEXT VARCHAR2(500 BYTE), COURSETOPICID NUMBER(10,0), DIFFICULTYLEVEL NUMBER(2,0) NOT NULL, HINT VARCHAR2(500 BYTE), EXPLANATION VARCHAR2(500 BYTE), ADD CONSTRAINT CHECK\_DIFFICULTYLEVEL CHECK (DIFFICULTYLEVEL BETWEEN 1 AND 6), ADD CONSTRAINT PK\_QUESTIONBANK PRIMARY KEY (QUESTIONID), ADD CONSTRAINT FK\_COURSETOPICS\_QUESTIONBANK FOREIGN KEY (COURSETOPICID) REFERENCES COURSETOPICS (COURSETOPICID))

15. CREATE TABLE QUESTIONPARAM (QUESTIONID NUMBER(30,0) NOT NULL, PARAMETERVALUE VARCHAR2(100 BYTE) NOT NULL, SETID NUMBER(30,0) NOT NULL, PARAMETERID NUMBER(5,0), ADD CONSTRAINT QUESTIONPARAM\_PK PRIMARY KEY (QUESTIONID, PARAMETERVALUE, SETID))

16. CREATE TABLE ROLES\_TBL (ROLEID NUMBER(10,0), ROLENAME VARCHAR2(20 BYTE) NOT NULL, ADD CONSTRAINT PK\_ROLEID PRIMARY KEY (ROLEID))

17. CREATE TABLE SECTIONS (SECTIONID NUMBER(10,0), SECTIONNAME VARCHAR2(100 BYTE) NOT NULL, TEXTBOOKISBN VARCHAR2(30 BYTE), CHAPTERID NUMBER(10,0), CONTENTVAL VARCHAR2(4000 BYTE) NOT NULL, ADD CONSTRAINT SECTIONS\_PK PRIMARY KEY (SECTIONID, TEXTBOOKISBN, CHAPTERID), ADD CONSTRAINT FK\_TXTISBN FOREIGN KEY (TEXTBOOKISBN) REFERENCES TEXTBOOKS (TEXTBOOKISBN))

18. CREATE TABLE STUDENTATTEMPT (EXERCISEID NUMBER(10,0), USERID NUMBER(10,0), ATTEMPTID NUMBER(10,0), MARKSOBTAINED NUMBER(10,0), SUBMISSIONTIME DATE, CONSTRAINT PK\_STUDENTATTEMPT PRIMARY KEY (EXERCISEID, USERID, ATTEMPTID), ADD CONSTRAINT FK\_EXERCISES\_STUDENTATTEMPT FOREIGN KEY (EXERCISEID) REFERENCES EXERCISES (EXERCISEID), ADD CONSTRAINT FK\_USERS\_STUDENTATTEMPT FOREIGN KEY (USERID) REFERENCES USERS (USERID))

19. CREATE TABLE STUDENTATTEMPTQUESTIONS(EXERCISEID NUMBER(10,0) NOT NULL, USERID NUMBER(10,0) NOT NULL, ATTEMPTID NUMBER(10,0) NOT NULL, QUESTIONID NUMBER(10,0) NOT NULL, OPTION1 NUMBER(2,0), OPTION2 NUMBER(2,0), OPTION3 NUMBER(2,0), OPTION4 NUMBER(2,0), SELECTEDOPTION NUMBER(2,0), SETID NUMBER(5,0), ADD CONSTRAINT PK\_STUDATTEMPTQUEST PRIMARY KEY (EXERCISEID, USERID, QUESTIONID, ATTEMPTID), ADD CONSTRAINT STUDENTATTEMPTQUESTIONS\_FK1 FOREIGN KEY (QUESTIONID) REFERENCES QUESTIONBANK (QUESTIONID), ADD CONSTRAINT STUDENTATTEMPTQUESTIONS\_FK2 FOREIGN KEY (USERID) REFERENCES USERS (USERID), ADD CONSTRAINT STUDENTATTEMPTQUESTIONS\_FK3 FOREIGN KEY (EXERCISEID) REFERENCES EXERCISES(EXERCISEID))

20. CREATE TABLE STUDENTENROLLMENT (USERID NUMBER(10,0), COURSETOKENID VARCHAR2(15 BYTE), ADD CONSTRAINT PK\_STDENR PRIMARY KEY (COURSETOKENID, USERID), ADD CONSTRAINT FK\_STDENR\_COURSETOKENID FOREIGN KEY (COURSETOKENID) REFERENCES COURSEOFFERINGS (COURSETOKENID), ADD CONSTRAINT FK\_STENR\_USERID FOREIGN KEY (USERID) REFERENCES USERS (USERID))

21. CREATE TABLE SUBSECTIONS(SUBSECTIONIDNUMBER(10,0), SECTIONID NUMBER(10,0), CHAPTERID NUMBER(10,0), TEXTBOOKISBN VARCHAR2(30 BYTE), SUBSECTIONNAME VARCHAR2(40 BYTE), CONTENT VARCHAR2(4000 BYTE) NOT NULL, ADD CONSTRAINT PK\_SUBSECTIONS PRIMARY KEY (SUBSECTIONID, SECTIONID, CHAPTERID, TEXTBOOKISBN, SUBSECTIONNAME), ADD CONSTRAINT FK\_ISB FOREIGN KEY (TEXTBOOKISBN) REFERENCES TEXTBOOKS (TEXTBOOKISBN))

22. CREATE TABLE TEXTBOOKS (TEXTBOOKISBN VARCHAR2(30 BYTE), TITLE VARCHAR2(100 BYTE) NOT NULL, AUTHOR VARCHAR2(100 BYTE) NOT NULL, ADD CONSTRAINT PK\_TEXTBOOKS PRIMARY KEY (TEXTBOOKISBN))

23. CREATE TABLE USERS (USERID NUMBER(10,0), FIRSTNAME VARCHAR2(40 BYTE), LASTNAME VARCHAR2(20 BYTE), EMAILADDRESS VARCHAR2(50 BYTE), ROLEID NUMBER(10,0), LEVELVAL VARCHAR2(20 BYTE), LOGINID VARCHAR2(10 BYTE), PASSWORDVAL VARCHAR2(20 BYTE), ADD CONSTRAINT PK\_USERS PRIMARY KEY (USERID), ADD CONSTRAINT UNIQUE\_LOGINID UNIQUE (LOGINID), ADD CONSTRAINT CHK\_LEVEL CHECK (LEVELVAL IN ('GRADUATE','UNDERGRADUATE','PROFESSOR')))

**PROCEDURES:**

1

CREATE OR REPLACE PROCEDURE ADD\_TA(COURSETOKEN\_ID IN VARCHAR2

, USER\_ID IN NUMBER, RESULT\_VAL OUT NUMBER)

AS

ISTA NUMBER;

COURSE\_ID NUMBER;

BEGIN

SELECT COUNT(\*) INTO ISTA FROM USERS WHERE USERID = USER\_ID;

IF(ISTA>0)

THEN

SELECT COUNT(\*) INTO ISTA FROM USERS WHERE USERID = USER\_ID AND ROLEID=2;

RESULT\_VAL :=1;

ELSE

RESULT\_VAL := -1;

END IF;

IF(ISTA=0)

THEN

RESULT\_VAL := -1;

ELSE

RESULT\_VAL := 1;

END IF;

IF(RESULT\_VAL =1)

THEN

SELECT COURSEID INTO COURSE\_ID FROM COURSEOFFERINGS WHERE COURSETOKENID = COURSETOKEN\_ID;

INSERT INTO COURSETEACHING VALUES(USER\_ID,COURSE\_ID,COURSETOKEN\_ID,3);

END IF;

END;

2

CREATE OR REPLACE PROCEDURE ADDCOURSE

(

LOGIN\_ID IN VARCHAR2

, COURSE\_ID IN NUMBER

, COURSE\_TOKENID IN VARCHAR2

, BEGIN\_DATE IN DATE

, END\_DATE IN DATE

, MAX\_ENROL IN NUMBER

, COURSE\_NAME IN VARCHAR2

, COURSE\_LEVEL IN VARCHAR2

, RESULT\_VALUE OUT NUMBER

) AS

USER\_ID NUMBER;

TEMPCOUNT1 NUMBER;

ROLE\_ID NUMBER;

COURSE\_COUNT NUMBER;

BEGIN

RESULT\_VALUE := -9;

USER\_ID := MAPLOGINIDTOUSERID(LOGIN\_ID);

SELECT COUNT(\*) INTO COURSE\_COUNT FROM COURSE WHERE COURSEID = COURSE\_ID;

IF (COURSE\_COUNT=0)

THEN

INSERT INTO COURSE VALUES(COURSE\_ID,COURSE\_NAME,COURSE\_LEVEL);

RESULT\_VALUE := 1;

ELSE

RESULT\_VALUE := -1;

END IF;

SELECT COUNT(\*) INTO TEMPCOUNT1 FROM COURSEOFFERINGS WHERE COURSETOKENID = COURSE\_TOKENID;

IF(TEMPCOUNT1 = 0)

THEN

INSERT INTO COURSEOFFERINGS VALUES(COURSE\_TOKENID,COURSE\_ID,BEGIN\_DATE,END\_DATE,MAX\_ENROL);

SELECT R.ROLEID INTO ROLE\_ID FROM ROLES\_TBL R, USERS U

WHERE U.ROLEID = R.ROLEID

AND U.LOGINID = LOGIN\_ID;

INSERT INTO COURSETEACHING VALUES (USER\_ID,COURSE\_ID,COURSE\_TOKENID,ROLE\_ID);

RESULT\_VALUE := 1;

ELSE

RESULT\_VALUE := -1;

END IF;

END;

3

CREATE OR REPLACE PROCEDURE ADDNOTIFICATION AS

COUNT\_VAL NUMBER;

NOTIFICATIONCOUNT NUMBER;

CURSOR C1 IS SELECT DISTINCT COURSETOKENID FROM COURSEOFFERINGS ;

CURSOR C2 IS SELECT DISTINCT USERID FROM USERS;

COU\_TOK VARCHAR2(200);

USER\_ID NUMBER;

BEGIN

OPEN C1;

SELECT MAX(NOTIFICATIONID) INTO COUNT\_VAL FROM NOTIFICATIONS;

COUNT\_VAL := COUNT\_VAL+1;

LOOP

FETCH C1 INTO COU\_TOK;

EXIT WHEN C1%NOTFOUND;

INSERT INTO NOTIFICATIONS VALUES(COUNT\_VAL,'PLEASE CHANGE YOUR PASSWORD',COU\_TOK);

END LOOP;

CLOSE C1;

OPEN C2;

SELECT MAX(NOTIFICATIONID) INTO NOTIFICATIONCOUNT FROM NOTIFICATIONS;

LOOP

FETCH C2 INTO USER\_ID;

EXIT WHEN C2%NOTFOUND;

INSERT INTO NOTIFICATIONUSERMAPPING VALUES(USER\_ID,NOTIFICATIONCOUNT,1);

END LOOP;

CLOSE C2;

END;

4

CREATE OR REPLACE PROCEDURE ATTEMPT\_ZERO (HW\_ID IN NUMBER, COURSETOKEN\_ID IN VARCHAR2) AS

CURSOR C1 IS SELECT USERID

FROM USERS WHERE USERID IN

(SELECT USERID FROM STUDENTENROLLMENT WHERE COURSETOKENID=COURSETOKEN\_ID) AND ROLEID=2;

J NUMBER;

K NUMBER;

BEGIN

OPEN C1;

K:=0;

LOOP

FETCH C1 INTO J;

EXIT WHEN C1%NOTFOUND;

IF C1%FOUND

THEN

INSERT INTO STUDENTATTEMPT VALUES(HW\_ID,J,K,NULL,NULL);

END IF;

END LOOP;

CLOSE C1;

END;

5

CREATE OR REPLACE PROCEDURE ATTEMPT\_ZEROSIGNUP (LOGIN\_ID IN VARCHAR2,COURSETOKEN\_ID IN VARCHAR2) AS

CURSOR C1 IS SELECT EXERCISEID

FROM EXERCISELIST WHERE COURSETOKENID = COURSETOKEN\_ID;

J NUMBER;

K NUMBER;

USER\_ID NUMBER;

BEGIN

OPEN C1;

SELECT USERID INTO USER\_ID FROM USERS WHERE LOGINID = LOGIN\_ID;

K:=0;

LOOP

FETCH C1 INTO J;

EXIT WHEN C1%NOTFOUND;

IF C1%FOUND

THEN

INSERT INTO STUDENTATTEMPT VALUES(J,USER\_ID,K,NULL,NULL);

END IF;

END LOOP;

CLOSE C1;

END;

6

CREATE OR REPLACE PROCEDURE CHECK\_COURSES

(LOGIN\_ID IN VARCHAR2,

COURSE\_ID IN INT,

RETVALUE OUT INT,

COURSE\_TOKENID OUT VARCHAR2

) AS

USER\_ID NUMBER;

COURSECOUNT NUMBER;

BEGIN

COURSECOUNT := 0;

USER\_ID := MAPLOGINIDTOUSERID(LOGIN\_ID);

SELECT COUNT(S.COURSETOKENID) INTO COURSECOUNT FROM STUDENTENROLLMENT S, COURSEOFFERINGS CO, COURSE CS

WHERE

S.COURSETOKENID = CO.COURSETOKENID

AND CO.COURSEID = COURSE\_ID

AND CO.COURSEID = CS.COURSEID

AND SYSDATE BETWEEN CO.STARTDATE AND CO.ENDDATE

AND S.USERID = USER\_ID ;

IF(COURSECOUNT > 0)

THEN

RETVALUE := 1;

SELECT COURSETOKENID INTO COURSE\_TOKENID FROM COURSEOFFERINGS WHERE COURSEID = COURSE\_ID;

ELSE

SELECT COUNT(CT.COURSEID) INTO COURSECOUNT FROM COURSETEACHING CT, COURSEOFFERINGS CO, COURSE CS

WHERE CT.COURSETOKENID = CO.COURSETOKENID

AND CT.COURSEID = COURSE\_ID

AND CO.COURSEID = CS.COURSEID

AND SYSDATE BETWEEN CO.STARTDATE AND CO.ENDDATE

AND CT.USERID = USER\_ID

AND CT.ROLEID = 3;

IF(COURSECOUNT > 0)

THEN

RETVALUE := 2;

SELECT COURSETOKENID INTO COURSE\_TOKENID FROM COURSEOFFERINGS WHERE COURSEID = COURSE\_ID;

ELSE

RETVALUE := 3;

END IF;

END IF;

END;

7

CREATE OR REPLACE PROCEDURE CHECKROLE(LOGIN\_ID IN VARCHAR2,ROLEVAL OUT NUMBER) AS

BEGIN

SELECT ROLEID INTO ROLEVAL

FROM USERS

WHERE LOGINID=LOGIN\_ID;

END;

8

CREATE OR REPLACE PROCEDURE CREATE\_USER(FIRST\_NAME IN VARCHAR2, LAST\_NAME IN VARCHAR2,

EMAIL\_ADDRESS IN VARCHAR2, LEVELVAL IN VARCHAR2, LOGIN\_ID IN VARCHAR2, PASSWORDVAL IN VARCHAR2,

RET\_VALUE OUT NUMBER)

AS

MAXUSERID NUMBER(5);

ROLETEMP NUMBER(5);

COUNTE NUMBER(5);

BEGIN

RET\_VALUE := 0;

SELECT MAX(USERID)INTO MAXUSERID FROM USERS;

MAXUSERID := MAXUSERID + 1;

IF FIRST\_NAME IS NULL

THEN

RET\_VALUE := -1;

ELSIF LAST\_NAME IS NULL

THEN

RET\_VALUE := -2;

ELSIF EMAIL\_ADDRESS IS NULL

THEN

RET\_VALUE := -3;

ELSIF LEVELVAL IS NULL

THEN

RET\_VALUE := -4;

ELSIF LOGIN\_ID IS NULL

THEN

RET\_VALUE := -5;

ELSIF PASSWORDVAL IS NULL

THEN

RET\_VALUE := -6;

END IF;

IF (LEVELVAL = 'GRADUATE' OR LEVELVAL = 'UNDERGRADUATE')

THEN

SELECT ROLEID INTO ROLETEMP FROM ROLES\_TBL WHERE ROLENAME = 'STUDENT';

ELSE

IF (RET\_VALUE = 0)

THEN

RET\_VALUE := -7;

END IF;

END IF;

SELECT COUNT(\*) INTO COUNTE FROM USERS WHERE EMAILADDRESS = EMAIL\_ADDRESS;

IF(COUNTE != 0)

THEN

IF (RET\_VALUE = 0)

THEN

RET\_VALUE := -8;

END IF;

END IF;

COUNTE := -1;

SELECT COUNT(\*) INTO COUNTE FROM USERS WHERE LOGINID = LOGIN\_ID;

IF(COUNTE != 0)

THEN

IF (RET\_VALUE = 0)

THEN

RET\_VALUE := -9;

END IF;

END IF;

IF (RET\_VALUE = 0)

THEN

INSERT INTO USERS VALUES(MAXUSERID,FIRST\_NAME, LAST\_NAME, EMAIL\_ADDRESS,ROLETEMP,LEVELVAL,LOGIN\_ID,PASSWORDVAL);

RET\_VALUE := 1;

END IF;

END;

9

CREATE OR REPLACE PROCEDURE DELETE\_QUESTIONS

(

INPUTQUESTIONS\_LIST IN QUESTION\_LIST,

HW\_ID INT,

RET\_VALUE OUT NUMBER

)

AS

--CURRENTQUESTIONCOUNT INT := 0;

--TOTALNUMBEROFQUESTIONS INT := 0;

BEGIN

-- SELECT COUNT(\*) INTO CURRENTQUESTIONCOUNT FROM EXERCISEQUESTION WHERE EXERCISEID = HW\_ID;

-- SELECT NUMBEROFQUESTIONS INTO TOTALNUMBEROFQUESTIONS FROM EXERCISES;

FOR I IN 1 .. INPUTQUESTIONS\_LIST.COUNT

LOOP

DELETE FROM EXERCISEQUESTION WHERE QUESTIONID = INPUTQUESTIONS\_LIST(I) AND EXERCISEID = HW\_ID;

END LOOP;

RET\_VALUE := 0;

END DELETE\_QUESTIONS;

10

CREATE OR REPLACE PROCEDURE EDIT\_HW (EXERCISE\_ID IN NUMBER, C1 OUT SYS\_REFCURSOR) AS

BEGIN

OPEN C1 FOR

SELECT STARTDATETIME,ENDDATETIME,NUMBEROFRETRIES,MARKSPERCORRECTANSWER,MARKSPERINCORRECTANSWER,SCORINGTYPE,DIFFICULTYLEVEL FROM EXERCISES

WHERE EXERCISEID = EXERCISE\_ID;

END;

11

CREATE OR REPLACE PROCEDURE EXERCISECREATION(COURSETOPIC\_IP IN COURSETOPIC\_LIST,

STARTDATE IN DATE,

ENDDATE IN DATE,

NUMBEROFATTEMPTS IN NUMBER,

DIFFICULTYRANGE IN NUMBER,

SCORESELECTIONSCHEME IN VARCHAR2,

NUMQUESTIONS IN NUMBER,

CORRECTANSWERPOINTS IN NUMBER ,

COURSETOKENID IN VARCHAR2,

INCORRECTANSWERPOINTS IN NUMBER,

RET\_VALUE OUT NUMBER)

AS

COU\_TOK VARCHAR2(40);

COU\_TOPID NUMBER;

LEN NUMBER;

USER\_ID NUMBER;

K NUMBER;

COUNT\_VAL NUMBER;

NOT\_STRING VARCHAR2(200);

CURSOR C2 IS SELECT DISTINCT USERID FROM STUDENTENROLLMENT WHERE COURSETOKENID = COURSETOKENID;

BEGIN

RET\_VALUE := 0;

SELECT MAX(EXERCISEID) INTO K FROM EXERCISES;

INSERT INTO EXERCISES VALUES(K+1,DIFFICULTYRANGE,NUMBEROFATTEMPTS, CORRECTANSWERPOINTS, INCORRECTANSWERPOINTS, STARTDATE, ENDDATE, SCORESELECTIONSCHEME,NUMQUESTIONS);

FOR I IN 1 .. COURSETOPIC\_IP.COUNT

LOOP

SELECT COURSETOKENID INTO COU\_TOK FROM COURSEOFFERINGS WHERE COURSEID IN (SELECT COURSEID FROM COURSETOPICS WHERE COURSETOPICID = COURSETOPIC\_IP(I));

/\*SELECT COURSETOPICID INTO COU\_TOPID FROM COURSETOPICS WHERE COURSETOPICNAME = COURSETOPIC\_IP(I); \*/

INSERT INTO EXERCISELIST VALUES(K+1, COURSETOPIC\_IP(I), COU\_TOK);

END LOOP;

RET\_VALUE := K+1;

NOT\_STRING := CONCAT ('ADDED HOMEWORK : ',K+1);

SELECT COUNT(\*) INTO COUNT\_VAL FROM NOTIFICATIONS;

COUNT\_VAL := COUNT\_VAL+1;

INSERT INTO NOTIFICATIONS VALUES(COUNT\_VAL,NOT\_STRING,COU\_TOK);

OPEN C2;

LOOP

FETCH C2 INTO USER\_ID;

EXIT WHEN C2%NOTFOUND;

INSERT INTO NOTIFICATIONUSERMAPPING VALUES(USER\_ID,COUNT\_VAL,1);

END LOOP;

CLOSE C2;

END;

12

CREATE OR REPLACE PROCEDURE FETCH\_COURSES(LOGINID IN VARCHAR2) AS

ROLEVAL NUMBER;

BEGIN

SELECT ROLEID INTO ROLEVAL

FROM USERS

WHERE LOGINID=LOGINID;

DBMS\_OUTPUT.PUT\_LINE(ROLEVAL);

END;

13

CREATE OR REPLACE PROCEDURE FETCH\_COURSETOPICLIST

(

COURSE\_TOKENID IN VARCHAR2

, COURSETOPICS OUT SYS\_REFCURSOR

) AS

BEGIN

OPEN COURSETOPICS FOR

SELECT CT.COURSEID,CT.COURSETOPICID,CT.COURSETOPICNAME FROM COURSETOPICS CT, COURSEOFFERINGS CO

WHERE CT.COURSEID = CO.COURSEID AND CO.COURSETOKENID = COURSE\_TOKENID;

END;

14

CREATE OR REPLACE PROCEDURE FETCH\_QUESTIONS

(COURSETOKEN\_ID VARCHAR2,

HW\_ID NUMBER,

ISADD NUMBER,

C1 OUT SYS\_REFCURSOR)

AS

BEGIN

IF(ISADD = 1)

THEN

OPEN C1 FOR

SELECT QUESTIONID,QUESTIONTEXT FROM QUESTIONBANK WHERE COURSETOPICID IN (

SELECT COURSETOPICID FROM EXERCISELIST WHERE COURSETOKENID =COURSETOKEN\_ID AND EXERCISEID = HW\_ID);

ELSIF(ISADD = 0)

THEN

OPEN C1 FOR

SELECT QUESTIONID,QUESTIONTEXT FROM QUESTIONBANK WHERE QUESTIONID IN (

SELECT EXERCISEQUESTION.QUESTIONID FROM EXERCISEQUESTION WHERE EXERCISEQUESTION.EXERCISEID = HW\_ID);

END IF;

END FETCH\_QUESTIONS;

15

CREATE OR REPLACE PROCEDURE FETCHEXERCISEDETAILS(EXID NUMBER, EXDETAILS OUT SYS\_REFCURSOR)

AS

BEGIN

OPEN EXDETAILS FOR

SELECT E.EXERCISEID,E.MARKSPERCORRECTANSWER,E.MARKSPERINCORRECTANSWER,E.NUMBEROFRETRIES,E.SCORINGTYPE,E.STARTDATETIME,E.ENDDATETIME,E.DIFFICULTYLEVEL,E.NUMBEROFQUESTIONS

FROM EXERCISES E WHERE EXERCISEID = EXID;

END;

16

CREATE OR REPLACE PROCEDURE FETCHPASTQUESTIONSSUBMISSIONS(

LOGIN\_ID IN VARCHAR2,

EXERCISE\_ID IN NUMBER,

ATTEMPT\_ID IN NUMBER,

PASTSUB OUT SYS\_REFCURSOR)

AS

USER\_ID NUMBER;

QUESNUM NUMBER;

OP1 NUMBER;

OP2 NUMBER;

OP3 NUMBER;

OP4 NUMBER;

SELOP NUMBER;

QUESTEXT VARCHAR2(500);

OPT1 VARCHAR2(500);

OPT2 VARCHAR2(500);

OPT3 VARCHAR2(500);

OPT4 VARCHAR2(500);

SELOPT VARCHAR2(500);

HINTVAL VARCHAR2(500);

EXPVAL VARCHAR2(500);

CRCTANS VARCHAR2(500);

SET\_ID NUMBER;

TEMPPARAM SYS\_REFCURSOR;

TEMPPARAMID NUMBER(5);

TEMPPARAMVALUE VARCHAR2(30);

BEGIN

EXECUTE IMMEDIATE 'TRUNCATE TABLE FETCH\_TEMP';

SELECT USERID INTO USER\_ID FROM USERS WHERE LOGINID = LOGIN\_ID;

OPEN PASTSUB FOR

SELECT QUESTIONID,OPTION1,OPTION2,OPTION3,OPTION4,SELECTEDOPTION,SETID

FROM STUDENTATTEMPTQUESTIONS

WHERE EXERCISEID=EXERCISE\_ID AND USERID=USER\_ID AND ATTEMPTID = ATTEMPT\_ID;

LOOP

FETCH PASTSUB INTO QUESNUM,OP1,OP2,OP3,OP4,SELOP,SET\_ID;

EXIT WHEN PASTSUB%NOTFOUND;

IF SET\_ID = 0

THEN

SELECT QUESTIONTEXT INTO QUESTEXT FROM QUESTIONBANK WHERE QUESTIONID = QUESNUM;

ELSE

SELECT QUESTIONTEXT INTO QUESTEXT FROM QUESTIONBANK WHERE QUESTIONID = QUESNUM;

OPEN TEMPPARAM FOR

SELECT QP.PARAMETERID, QP.PARAMETERVALUE FROM QUESTIONPARAM QP WHERE QP.QUESTIONID = QUESNUM AND QP.SETID = SET\_ID;

LOOP

FETCH TEMPPARAM INTO TEMPPARAMID,TEMPPARAMVALUE;

EXIT WHEN TEMPPARAM%NOTFOUND;

QUESTEXT := REPLACE(QUESTEXT,CONCAT('PARAM',TO\_CHAR(TEMPPARAMID)),TEMPPARAMVALUE);

END LOOP;

END IF;

SELECT ANSWERTEXT INTO OPT1 FROM ANSWERS WHERE ANSWERID=OP1 AND QUESTIONID = QUESNUM AND SETID = SET\_ID;

SELECT ANSWERTEXT INTO OPT2 FROM ANSWERS WHERE ANSWERID=OP2 AND QUESTIONID = QUESNUM AND SETID = SET\_ID;

SELECT ANSWERTEXT INTO OPT3 FROM ANSWERS WHERE ANSWERID=OP3 AND QUESTIONID = QUESNUM AND SETID = SET\_ID;

SELECT ANSWERTEXT INTO OPT4 FROM ANSWERS WHERE ANSWERID=OP4 AND QUESTIONID = QUESNUM AND SETID = SET\_ID;

SELECT ANSWERTEXT INTO CRCTANS FROM ANSWERS WHERE ISRIGHT= 'Y' AND QUESTIONID = QUESNUM AND ANSWERID IN (OP1,OP2,OP3,OP4) AND SETID = SET\_ID;

SELECT ANSWERTEXT INTO SELOPT FROM ANSWERS WHERE ANSWERID=SELOP AND QUESTIONID = QUESNUM AND SETID = SET\_ID;

SELECT HINT INTO HINTVAL FROM QUESTIONBANK WHERE QUESTIONID = QUESNUM;

SELECT EXPLANATION INTO EXPVAL FROM QUESTIONBANK WHERE QUESTIONID = QUESNUM;

INSERT INTO FETCH\_TEMP VALUES(QUESTEXT,OPT1,OPT2,OPT3,OPT4,SELOPT,HINTVAL,EXPVAL,CRCTANS);

END LOOP;

OPEN PASTSUB FOR

SELECT \* FROM FETCH\_TEMP;

END;

17

CREATE OR REPLACE PROCEDURE FETCHPASTSUBMISSIONS(LOGIN\_ID IN VARCHAR2,COURSE\_TOKEN IN VARCHAR2, PASTSUB OUT SYS\_REFCURSOR)

AS

USER\_ID NUMBER;

BEGIN

SELECT USERID INTO USER\_ID FROM USERS WHERE LOGINID = LOGIN\_ID;

OPEN PASTSUB FOR

SELECT SA.EXERCISEID,SA.USERID,SA.MARKSOBTAINED,SA.SUBMISSIONTIME,SA.ATTEMPTID FROM STUDENTATTEMPT SA, EXERCISELIST EL

WHERE SA.EXERCISEID = EL.EXERCISEID

AND SA.USERID = USER\_ID AND EL.COURSETOKENID = COURSE\_TOKEN AND SA.ATTEMPTID <> 0;

END;

18

CREATE OR REPLACE PROCEDURE FORGOT\_PWD(LOGIN\_ID IN VARCHAR2,

EMAIL\_ID IN VARCHAR2,

RESULTTEXT OUT VARCHAR2,

RESULTVAL OUT NUMBER,

PASSWORD\_VAL OUT VARCHAR2) AS

CVAL NUMBER;

LOCALEMAILID VARCHAR2(40);

BEGIN

RESULTVAL :=0;

SELECT COUNT(\*) INTO CVAL FROM USERS WHERE LOGINID = LOGIN\_ID;

IF CVAL > 0

THEN

SELECT 1 INTO RESULTVAL FROM USERS WHERE LOGINID = LOGIN\_ID;

END IF;

IF RESULTVAL = 1

THEN

SELECT EMAILADDRESS INTO LOCALEMAILID FROM USERS WHERE LOGINID = LOGIN\_ID;

END IF;

IF LOCALEMAILID = EMAIL\_ID

THEN

SELECT 2 INTO RESULTVAL FROM USERS WHERE LOGINID = LOGIN\_ID AND EMAILADDRESS = EMAIL\_ID;

SELECT PASSWORDVAL INTO PASSWORD\_VAL FROM USERS WHERE LOGINID = LOGIN\_ID AND EMAILADDRESS = EMAIL\_ID;

END IF;

IF RESULTVAL = 0

THEN

RESULTTEXT := 'INVALID LOGINID';

ELSE IF RESULTVAL = 1

THEN

RESULTTEXT := 'INVALID EMAIL ID. PLEASE ENTER THE EMAIL ID ';

ELSE

RESULTTEXT := 'A NEW PASSWORD HAS BEEN SENT TO THE EMAIL ID PROVIDED';

END IF;

END IF;

END;

->

CREATE OR REPLACE PROCEDURE HOMEWORKREMAINDER AS

COUNT\_VAL NUMBER;

NOTIFICATIONCOUNT NUMBER;

CURSOR C1 IS SELECT SE.USERID,E.EXERCISEID,EL.COURSETOKENID ,E.ENDDATETIME

FROM EXERCISES E, EXERCISELIST EL,STUDENTENROLLMENT SE

WHERE E.EXERCISEID = EL.EXERCISEID AND EL.COURSETOKENID = SE.COURSETOKENID

AND E.ENDDATETIME > SYSDATE AND E.STARTDATETIME <SYSDATE;

COU\_TOK VARCHAR2(200);

USER\_ID NUMBER;

EXERCISE\_ID NUMBER;

NOTIFICATION\_TEXT VARCHAR(200);

ENDDATE\_TIME DATE;

DATEDIFFVAL NUMBER;

BEGIN

OPEN C1;

SELECT MAX(NOTIFICATIONID) INTO COUNT\_VAL FROM NOTIFICATIONS;

COUNT\_VAL := COUNT\_VAL+1;

LOOP

FETCH C1 INTO USER\_ID, EXERCISE\_ID,COU\_TOK,ENDDATE\_TIME;

EXIT WHEN C1%NOTFOUND;

SELECT ENDDATE\_TIME - TRUNC(SYSDATE) INTO DATEDIFFVAL FROM DUAL;

IF(DATEDIFFVAL = 1)

THEN

INSERT INTO NOTIFICATIONS VALUES(COUNT\_VAL,'CURRENT EXERCISE IS DUE BY NEXT DATE. NO EXTENSIONS IN THE DEADLINE',COU\_TOK);

END IF;

END LOOP;

CLOSE C1;

OPEN C1;

SELECT MAX(NOTIFICATIONID) INTO NOTIFICATIONCOUNT FROM NOTIFICATIONS;

LOOP

FETCH C1 INTO USER\_ID,EXERCISE\_ID,COU\_TOK,ENDDATE\_TIME;

EXIT WHEN C1%NOTFOUND;

SELECT ENDDATE\_TIME - TRUNC(SYSDATE) INTO DATEDIFFVAL FROM DUAL;

IF(1=DATEDIFFVAL)

THEN

INSERT INTO NOTIFICATIONUSERMAPPING VALUES(USER\_ID,NOTIFICATIONCOUNT,1);

END IF;

END LOOP;

CLOSE C1;

END;

->

CREATE OR REPLACE PROCEDURE HW\_ATTEMPTS(COURSETOKEN\_ID IN VARCHAR2, LOGIN\_ID IN VARCHAR2, C1 OUT SYS\_REFCURSOR) AS

U\_ID NUMBER;

BEGIN

SELECT USERID INTO U\_ID FROM USERS WHERE LOGINID=LOGIN\_ID;

OPEN C1 FOR

SELECT E.EXERCISEID, E.NUMBEROFRETRIES, COUNT(S.ATTEMPTID)

FROM EXERCISES E INNER JOIN STUDENTATTEMPT S ON E.EXERCISEID = S.EXERCISEID

WHERE S.USERID = U\_ID AND E.EXERCISEID IN (SELECT EXERCISEID FROM EXERCISELIST WHERE COURSETOKENID=COURSETOKEN\_ID)

AND SYSDATE BETWEEN E.STARTDATETIME AND E.ENDDATETIME

GROUP BY S.EXERCISEID, E.EXERCISEID, E.NUMBEROFRETRIES;

END;

->

CREATE OR REPLACE PROCEDURE HW\_LIST(LOGIN\_ID IN VARCHAR2, COURSETOKEN\_ID IN VARCHAR2, C1 OUT SYS\_REFCURSOR) AS

BEGIN

OPEN C1 FOR

SELECT EXERCISEID, ATTEMPTID, MARKSOBTAINED, SUBMISSIONTIME

FROM STUDENTATTEMPT WHERE USERID IN

(SELECT USERID FROM USERS WHERE LOGINID=LOGIN\_ID)

AND EXERCISEID IN (SELECT EXERCISEID FROM EXERCISELIST WHERE COURSETOKENID=COURSETOKEN\_ID);

END;

->

CREATE OR REPLACE PROCEDURE HWNOTIFICATION (HW\_ID IN NUMBER,COURSETOKEN\_ID IN VARCHAR2)AS

COUNT\_VAL NUMBER;

NOTIFICATIONCOUNT NUMBER;

NOTIFICATION\_TEXT VARCHAR2(100);

CURSOR C2 IS SELECT DISTINCT USERID FROM STUDENTENROLLMENT WHERE COURSETOKENID = COURSETOKEN\_ID;

USER\_ID NUMBER;

BEGIN

SELECT MAX(NOTIFICATIONID) INTO COUNT\_VAL FROM NOTIFICATIONS;

COUNT\_VAL := COUNT\_VAL+1;

NOTIFICATION\_TEXT := CONCAT('ADDED HOMEWORK:',HW\_ID);

INSERT INTO NOTIFICATIONS VALUES(COUNT\_VAL,NOTIFICATION\_TEXT,COURSETOKEN\_ID);

SELECT MAX(NOTIFICATIONID) INTO NOTIFICATIONCOUNT FROM NOTIFICATIONS;

OPEN C2;

LOOP

FETCH C2 INTO USER\_ID;

EXIT WHEN C2%NOTFOUND;

INSERT INTO NOTIFICATIONUSERMAPPING VALUES(USER\_ID,NOTIFICATIONCOUNT,1);

END LOOP;

CLOSE C2;

END;

->

CREATE OR REPLACE PROCEDURE INSERT\_QUESTIONS

(

INPUTQUESTIONS\_LIST IN QUESTION\_LIST,

HW\_ID INT,

RET\_VALUE OUT NUMBER

)

AS

--CURRENTQUESTIONCOUNT INT := 0;

--TOTALNUMBEROFQUESTIONS INT := 0;

BEGIN

FOR I IN 1 .. INPUTQUESTIONS\_LIST.COUNT

LOOP

INSERT INTO EXERCISEQUESTION VALUES(HW\_ID,INPUTQUESTIONS\_LIST(I));

END LOOP;

RET\_VALUE := 1;

END INSERT\_QUESTIONS;

->

CREATE OR REPLACE PROCEDURE LIST\_TA(COURSETOKEN\_ID IN VARCHAR2,C1 OUT SYS\_REFCURSOR)

AS

BEGIN

OPEN C1 FOR

SELECT U.LOGINID

FROM USERS U,COURSETEACHING C

WHERE U.USERID = C.USERID

AND C.ROLEID = 3

AND C.COURSETOKENID = COURSETOKEN\_ID;

END;

->

CREATE OR REPLACE PROCEDURE LOGIN\_AUTHENTICATE(LOGIN\_ID IN VARCHAR2, PASSWORD\_IN IN VARCHAR2, AUTHENTICATED OUT NUMBER)

AS

PASSWORD\_DB VARCHAR2(10);

ID VARCHAR2(10);

COUNTVAL NUMBER;

BEGIN

SELECT COUNT(\*) INTO COUNTVAL FROM USERS WHERE LOGINID = LOGIN\_ID;

IF COUNTVAL >0

THEN

ID :=LOGIN\_ID;

AUTHENTICATED := -1;

SELECT PASSWORDVAL INTO PASSWORD\_DB FROM USERS WHERE LOGINID = ID;

IF PASSWORD\_DB = PASSWORD\_IN

THEN

AUTHENTICATED := 1;

ELSE

AUTHENTICATED := -1;

END IF;

ELSE

AUTHENTICATED := -1;

END IF;

END;

->

CREATE OR REPLACE PROCEDURE MAIL\_ADDHW(COURSETOKEN\_ID IN VARCHAR2, C1 OUT SYS\_REFCURSOR) AS

BEGIN

OPEN C1 FOR

SELECT EMAILADDRESS FROM USERS WHERE USERID IN(SELECT USERID FROM STUDENTENROLLMENT WHERE COURSETOKENID = COURSETOKEN\_ID);

END;

->

CREATE OR REPLACE PROCEDURE PROFCOURSE\_LIST(LOGIN\_ID IN VARCHAR2, C1 OUT SYS\_REFCURSOR)

AS

BEGIN

OPEN C1 FOR

SELECT C1.COURSENAME,C2.COURSETOKENID

FROM COURSE C1,COURSETEACHING C2,COURSEOFFERINGS C3

WHERE C1.COURSEID=C2.COURSEID

AND C1.COURSEID = C3.COURSEID

AND C2.USERID IN

(SELECT USERID

FROM USERS

WHERE LOGINID = LOGIN\_ID)

AND SYSDATE BETWEEN C3.STARTDATE AND C3.ENDDATE;

END;

->

CREATE OR REPLACE PROCEDURE PROFHW\_LIST(LOGIN\_ID IN VARCHAR2,COURSETOKEN\_ID IN VARCHAR2, C1 OUT SYS\_REFCURSOR) AS

BEGIN

OPEN C1 FOR

SELECT DISTINCT(EXERCISEID) FROM EXERCISELIST WHERE COURSETOKENID=COURSETOKEN\_ID ORDER BY EXERCISEID;

END;

->

CREATE OR REPLACE PROCEDURE QUES\_LIST(COURSETOPIC\_ID IN NUMBER, C1 OUT SYS\_REFCURSOR)

AS

BEGIN

OPEN C1 FOR

SELECT QUESTIONID, QUESTIONTEXT FROM QUESTIONBANK WHERE COURSETOPICID=COURSETOPIC\_ID;

END;

CREATE OR REPLACE PROCEDURE QUES\_LIST\_HWID(HWID IN NUMBER, C1 OUT SYS\_REFCURSOR)

AS

BEGIN

OPEN C1 FOR

SELECT QUESTIONTEXT FROM QUESTIONBANK WHERE QUESTIONID IN (SELECT QUESTIONID FROM EXERCISEQUESTION WHERE EXERCISEID=HWID);

END;

->

CREATE OR REPLACE PROCEDURE QUESTIONTOANSWER(QUESTION\_ID IN NUMBER,SET\_ID IN NUMBER, ANSWERIDS OUT SYS\_REFCURSOR)

AS

BEGIN

OPEN ANSWERIDS FOR

SELECT ANSWERID FROM ANSWERS WHERE ISRIGHT = 'Y' AND QUESTIONID = QUESTION\_ID AND SETID = SET\_ID;

END;

->

CREATE OR REPLACE PROCEDURE REGISTERED\_COURSES(LOGIN\_ID IN VARCHAR2,

P\_RECORDSET OUT SYS\_REFCURSOR) AS

USER\_ID NUMBER;

BEGIN

USER\_ID := MAPLOGINIDTOUSERID(LOGIN\_ID);

OPEN P\_RECORDSET FOR

SELECT CO.COURSEID,CS.COURSENAME,S.COURSETOKENID FROM STUDENTENROLLMENT S, COURSEOFFERINGS CO, COURSE CS

WHERE

S.COURSETOKENID = CO.COURSETOKENID AND CO.COURSEID = CS.COURSEID

AND SYSDATE BETWEEN CO.STARTDATE AND CO.ENDDATE

AND S.USERID = USER\_ID

UNION

SELECT CO.COURSEID,CS.COURSENAME,CT.COURSETOKENID FROM COURSETEACHING CT, COURSEOFFERINGS CO, COURSE CS

WHERE

CT.COURSETOKENID = CO.COURSETOKENID AND CO.COURSEID = CS.COURSEID

AND SYSDATE BETWEEN CO.STARTDATE AND CO.ENDDATE

AND CT.USERID = USER\_ID

AND CT.ROLEID = 3;

END;

->

CREATE OR REPLACE PROCEDURE REMOVE\_TA(COURSETOKEN\_ID IN VARCHAR2

, LOGIN\_ID IN VARCHAR2,RET\_VALUE OUT NUMBER)

AS

USER\_ID NUMBER;

BEGIN

RET\_VALUE := 0;

SELECT USERID INTO USER\_ID FROM USERS WHERE LOGINID = LOGIN\_ID;

DELETE FROM COURSETEACHING WHERE USERID=USER\_ID;

RET\_VALUE := 1;

END;

->

CREATE OR REPLACE PROCEDURE RETRIEVEANSWERS(EXERCISEID IN NUMBER,QUESTIONIDS IN QUESTION\_LIST ,SETIDS IN SETS\_LIST, ANSWERS OUT SYS\_REFCURSOR)

AS

CURSOR QUESTIONS IS

SELECT DISTINCT EQ.QUESTIONID

FROM EXERCISEQUESTION EQ, QUESTIONBANK QB

WHERE EQ.QUESTIONID = QB.QUESTIONID

AND EQ.EXERCISEID = EXERCISEID

ORDER BY EQ.QUESTIONID;

QUESTION\_ID QUESTIONBANK.QUESTIONID%TYPE;

BEGIN

EXECUTE IMMEDIATE 'TRUNCATE TABLE ANSWERS\_TEMP';

OPEN QUESTIONS;

LOOP

FETCH QUESTIONS INTO QUESTION\_ID;

EXIT WHEN QUESTIONS%NOTFOUND;

INSERT INTO ANSWERS\_TEMP(ANSWERID,ANSWERTEXT,EXPLANATION,ISRIGHT,QUESTIONID)

SELECT ANSWERID,ANSWERTEXT,EXPLANATION,ISRIGHT,QUESTIONID FROM (

SELECT ANSWERID,ANSWERTEXT,EXPLANATION,ISRIGHT,QUESTIONID FROM

(SELECT A.ANSWERID,A.ANSWERTEXT,A.EXPLANATION, A.ISRIGHT, A.QUESTIONID

FROM ANSWERS A

WHERE A.QUESTIONID = QUESTION\_ID AND A.ISRIGHT = 'N' AND A.SETID = 0

ORDER BY DBMS\_RANDOM.RANDOM)

WHERE ROWNUM < 4

UNION

SELECT ANSWERID,ANSWERTEXT,EXPLANATION,ISRIGHT,QUESTIONID FROM

(SELECT A.ANSWERID,A.ANSWERTEXT,A.EXPLANATION, A.ISRIGHT, A.QUESTIONID

FROM ANSWERS A

WHERE A.QUESTIONID = QUESTION\_ID AND A.ISRIGHT = 'Y' AND A.SETID = 0

ORDER BY DBMS\_RANDOM.RANDOM)

WHERE ROWNUM < 2

)

ORDER BY SYS.DBMS\_RANDOM.RANDOM;

END LOOP;

CLOSE QUESTIONS;

FOR I IN 1..QUESTIONIDS.COUNT

LOOP

IF(SETIDS(I) > 0)

THEN

INSERT INTO ANSWERS\_TEMP(ANSWERID,ANSWERTEXT,EXPLANATION,ISRIGHT,QUESTIONID)

SELECT ANSWERID,ANSWERTEXT,EXPLANATION,ISRIGHT,QUESTIONID FROM (

SELECT ANSWERID,ANSWERTEXT,EXPLANATION,ISRIGHT,QUESTIONID FROM

(SELECT A.ANSWERID,A.ANSWERTEXT,A.EXPLANATION, A.ISRIGHT, A.QUESTIONID

FROM ANSWERS A

WHERE A.QUESTIONID = QUESTIONIDS(I) AND A.ISRIGHT = 'N' AND A.SETID = SETIDS(I)

ORDER BY DBMS\_RANDOM.RANDOM)

WHERE ROWNUM < 4

UNION

SELECT ANSWERID,ANSWERTEXT,EXPLANATION,ISRIGHT,QUESTIONID FROM

(SELECT A.ANSWERID,A.ANSWERTEXT,A.EXPLANATION, A.ISRIGHT, A.QUESTIONID

FROM ANSWERS A

WHERE A.QUESTIONID = QUESTIONIDS(I) AND A.ISRIGHT = 'Y' AND A.SETID = SETIDS(I)

ORDER BY DBMS\_RANDOM.RANDOM)

WHERE ROWNUM < 2

)

ORDER BY SYS.DBMS\_RANDOM.RANDOM;

END IF;

END LOOP;

OPEN ANSWERS FOR

SELECT \* FROM ANSWERS\_TEMP ORDER BY QUESTIONID;

END;

->

CREATE OR REPLACE PROCEDURE RETRIEVECONTENTVAL(COURSETOKEN\_ID IN VARCHAR2,

SECTION\_NAME IN VARCHAR2,

CONTENT\_VAL OUT VARCHAR2)

AS

COURSE\_ID NUMBER;

BEGIN

SELECT S.CONTENTVAL INTO CONTENT\_VAL FROM SECTIONS S, TEXTBOOKS T, COURSETEXTBOOKMAPPING CTM

WHERE S.TEXTBOOKISBN = T.TEXTBOOKISBN AND CTM.TEXTBOOKISBN = T.TEXTBOOKISBN AND CTM.COURSETOKENID=COURSETOKEN\_ID AND S.SECTIONNAME = SECTION\_NAME;

END;

CREATE OR REPLACE PROCEDURE RETRIEVECOURSECONTENT

(COURSETOKEN\_ID IN VARCHAR2,

COURSETOPIC\_NAME OUT SYS\_REFCURSOR) AS

COURSE\_ID NUMBER;

BEGIN

OPEN COURSETOPIC\_NAME FOR

SELECT S.SECTIONID,S.SECTIONNAME,S.CONTENTVAL FROM SECTIONS S,TEXTBOOKS T, COURSETEXTBOOKMAPPING CTM

WHERE S.TEXTBOOKISBN = T.TEXTBOOKISBN AND CTM.TEXTBOOKISBN = T.TEXTBOOKISBN AND CTM.COURSETOKENID = COURSETOKEN\_ID;

END;

CREATE OR REPLACE PROCEDURE RETRIEVEEXERCISEQUESTIONS(EXERCISE\_ID IN NUMBER, QUESTIONS OUT SYS\_REFCURSOR, COUNTREC OUT NUMBER) AS

NUMBEROFQUESTIONS INT;

TEMPQUESTIONS SYS\_REFCURSOR;

TEMPQUESTIONIDS SYS\_REFCURSOR;

TEMPPARAM SYS\_REFCURSOR;

TEMPSETID INT;

TEMPQID INT;

TEMPPARAMID INT;

TEMPPARAMVALUE VARCHAR2(50);

TEMPQUESTIONTEXT VARCHAR2(500);

BEGIN

EXECUTE IMMEDIATE 'TRUNCATE TABLE TEMP\_QUESTIONSINFO';

INSERT INTO TEMP\_QUESTIONSINFO

SELECT DISTINCT EQ.QUESTIONID,QB.QUESTIONTEXT,QB.HINT,QB.EXPLANATION ,0 AS SETID

FROM EXERCISEQUESTION EQ, QUESTIONBANK QB

WHERE EQ.QUESTIONID = QB.QUESTIONID

AND EQ.EXERCISEID = EXERCISE\_ID

AND NOT EXISTS

(

SELECT 1 FROM QUESTIONPARAM QP WHERE QP.QUESTIONID = QB.QUESTIONID

);

OPEN TEMPQUESTIONIDS FOR

SELECT DISTINCT QP.QUESTIONID FROM QUESTIONPARAM QP, EXERCISEQUESTION EQ

WHERE EQ.QUESTIONID = QP.QUESTIONID AND EQ.EXERCISEID = EXERCISE\_ID;

LOOP

FETCH TEMPQUESTIONIDS INTO TEMPQID;

EXIT WHEN TEMPQUESTIONIDS%NOTFOUND;

OPEN TEMPQUESTIONS FOR

SELECT SETID FROM

(SELECT DISTINCT QP.SETID FROM QUESTIONPARAM QP, EXERCISEQUESTION EQ

WHERE EQ.QUESTIONID = TEMPQID AND EQ.EXERCISEID = EXERCISE\_ID

ORDER BY DBMS\_RANDOM.RANDOM) WHERE ROWNUM <=1;

SELECT QUESTIONTEXT INTO TEMPQUESTIONTEXT FROM QUESTIONBANK WHERE QUESTIONID = TEMPQID;

LOOP

FETCH TEMPQUESTIONS INTO TEMPSETID;

EXIT WHEN TEMPQUESTIONS%NOTFOUND;

OPEN TEMPPARAM FOR

SELECT QP.PARAMETERID, QP.PARAMETERVALUE FROM QUESTIONPARAM QP WHERE QP.QUESTIONID = TEMPQID AND QP.SETID = TEMPSETID;

LOOP

FETCH TEMPPARAM INTO TEMPPARAMID,TEMPPARAMVALUE;

EXIT WHEN TEMPPARAM%NOTFOUND;

TEMPQUESTIONTEXT := REPLACE(TEMPQUESTIONTEXT,CONCAT('PARAM',TO\_CHAR(TEMPPARAMID)),TEMPPARAMVALUE);

END LOOP;

INSERT INTO TEMP\_QUESTIONSINFO

SELECT TEMPQID,TEMPQUESTIONTEXT,QB.HINT,QB.EXPLANATION,TEMPSETID FROM QUESTIONBANK QB

WHERE QB.QUESTIONID = TEMPQID;

END LOOP;

END LOOP;

SELECT COUNT(\*) INTO COUNTREC FROM TEMP\_QUESTIONSINFO;

OPEN QUESTIONS FOR

SELECT \* FROM TEMP\_QUESTIONSINFO

ORDER BY QUESTIONID;

END;

CREATE OR REPLACE PROCEDURE RETRIEVENOTIFICATIONS(LOGIN\_ID IN VARCHAR2, COURSETOKEN\_ID IN VARCHAR2, NOTIFICATION\_TEXT OUT SYS\_REFCURSOR) AS

USER\_ID NUMBER;

BEGIN

SELECT USERID INTO USER\_ID FROM USERS WHERE LOGINID = LOGIN\_ID;

OPEN NOTIFICATION\_TEXT FOR

SELECT N.NOTIFICATIONTEXT,N.NOTIFICATIONID

FROM NOTIFICATIONS N, NOTIFICATIONUSERMAPPING NUM

WHERE N.NOTIFICATIONID = NUM.NOTIFICATIONID

AND NUM.USERID = USER\_ID AND N.COURSETOKENID=COURSETOKEN\_ID AND NUM.READFLAG =1;

END;

CREATE OR REPLACE PROCEDURE SEND\_MAIL (FROMVAL IN VARCHAR2) AS

MSG\_FROM VARCHAR2(100);

MSG\_TO VARCHAR2(100) := 'SPULIMA@NCSU.EDU';

MSG\_SUBJECT VARCHAR2(100) := 'E-MAIL MESSAGE FROM YOUR DATABASE';

MSG\_TEXT VARCHAR2(100) := 'HELLO';

C UTL\_TCP.CONNECTION;

RC INTEGER;

BEGIN

C := UTL\_TCP.OPEN\_CONNECTION('127.0.0.1', 25); -- OPEN THE SMTP PORT 25 ON LOCAL MACHINE

DBMS\_OUTPUT.PUT\_LINE(UTL\_TCP.GET\_LINE(C, TRUE));

RC := UTL\_TCP.WRITE\_LINE(C, 'HELO LOCALHOST');

DBMS\_OUTPUT.PUT\_LINE(UTL\_TCP.GET\_LINE(C, TRUE));

RC := UTL\_TCP.WRITE\_LINE(C, 'MAIL FROM: '||MSG\_FROM);

DBMS\_OUTPUT.PUT\_LINE(UTL\_TCP.GET\_LINE(C, TRUE));

RC := UTL\_TCP.WRITE\_LINE(C, 'RCPT TO: '||MSG\_TO);

DBMS\_OUTPUT.PUT\_LINE(UTL\_TCP.GET\_LINE(C, TRUE));

RC := UTL\_TCP.WRITE\_LINE(C, 'DATA'); -- START MESSAGE BODY

DBMS\_OUTPUT.PUT\_LINE(UTL\_TCP.GET\_LINE(C, TRUE));

RC := UTL\_TCP.WRITE\_LINE(C, 'SUBJECT: '||MSG\_SUBJECT);

RC := UTL\_TCP.WRITE\_LINE(C, MSG\_TEXT);

RC := UTL\_TCP.WRITE\_LINE(C, '.'); -- END OF MESSAGE BODY

DBMS\_OUTPUT.PUT\_LINE(UTL\_TCP.GET\_LINE(C, TRUE));

RC := UTL\_TCP.WRITE\_LINE(C, 'QUIT');

DBMS\_OUTPUT.PUT\_LINE(UTL\_TCP.GET\_LINE(C, TRUE));

UTL\_TCP.CLOSE\_CONNECTION(C); -- CLOSE THE CONNECTION

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR( -20000, 'UNABLE TO SEND E-MAIL MESSAGE FROM PL/SQL BECAUSE OF: '|| SQLERRM); END;

CREATE OR REPLACE PROCEDURE SEND\_MAIL\_SKB AS

V\_FROM VARCHAR2(80) := 'SBOBBA3@NCSU.EDU';

V\_RECIPIENT VARCHAR2(80) := 'SKARAKA@NCSU.EDU';

V\_SUBJECT VARCHAR2(80) := 'TEST SUBJECT';

V\_MAIL\_HOST VARCHAR2(30) := 'SMTP.GMAIL.COM';

V\_MAIL\_CONN UTL\_SMTP.CONNECTION;

CRLF VARCHAR2(2) := CHR(13)||CHR(10);

BEGIN

V\_MAIL\_CONN := UTL\_SMTP.OPEN\_CONNECTION(V\_MAIL\_HOST, 587);

UTL\_SMTP.HELO(V\_MAIL\_CONN, V\_MAIL\_HOST);

UTL\_SMTP.MAIL(V\_MAIL\_CONN, V\_FROM);

UTL\_SMTP.RCPT(V\_MAIL\_CONN, V\_RECIPIENT);

UTL\_SMTP.DATA(V\_MAIL\_CONN,

'DATE: ' || TO\_CHAR(SYSDATE, 'DY, DD MON YYYY HH24:MI:SS') || CRLF ||

'FROM: ' || V\_FROM || CRLF ||

'SUBJECT: '|| V\_SUBJECT || CRLF ||

'TO: ' || V\_RECIPIENT || CRLF ||

CRLF ||

'SOME MESSAGE TEXT'|| CRLF || -- MESSAGE BODY

'MORE MESSAGE TEXT'|| CRLF

);

UTL\_SMTP.QUIT(V\_MAIL\_CONN);

EXCEPTION

WHEN UTL\_SMTP.TRANSIENT\_ERROR OR UTL\_SMTP.PERMANENT\_ERROR THEN

RAISE\_APPLICATION\_ERROR(-20000, 'UNABLE TO SEND MAIL: '||SQLERRM);

END;

CREATE OR REPLACE PROCEDURE STUDENT\_ENROLLMENT(LOGIN\_ID IN VARCHAR2, COURSE\_TOKEN\_ID IN VARCHAR2, RESULT\_VALUE OUT NUMBER)

IS

USER\_ID NUMBER(5);

COUNTER NUMBER(2);

NUMBERENROLLED NUMBER(3);

DATECHECK NUMBER(2);

COURSECHECK NUMBER(2);

MAXIMUMENROLLMENT NUMBER(2);

BEGIN

RESULT\_VALUE := -10;

USER\_ID := MAPLOGINIDTOUSERID(LOGIN\_ID);

SELECT COUNT(\*) INTO DATECHECK FROM COURSEOFFERINGS WHERE SYSDATE BETWEEN STARTDATE AND ENDDATE AND COURSETOKENID = COURSE\_TOKEN\_ID;

SELECT COUNT(\*) INTO COUNTER FROM STUDENTENROLLMENT WHERE USERID = USER\_ID AND COURSETOKENID = COURSE\_TOKEN\_ID;

SELECT COUNT(\*) INTO NUMBERENROLLED FROM STUDENTENROLLMENT WHERE COURSETOKENID = COURSE\_TOKEN\_ID;

SELECT COUNT(\*) INTO COURSECHECK FROM COURSEOFFERINGS WHERE COURSETOKENID = COURSE\_TOKEN\_ID;

SELECT MAXIMUMENROLLMENTNUMBER INTO MAXIMUMENROLLMENT FROM COURSEOFFERINGS WHERE COURSETOKENID = COURSE\_TOKEN\_ID;

IF (COUNTER = 0 AND NUMBERENROLLED = MAXIMUMENROLLMENT)

THEN

RESULT\_VALUE := -4;

ELSIF (COUNTER = 0 AND DATECHECK = 1)

THEN

INSERT INTO STUDENTENROLLMENT VALUES(USER\_ID,COURSE\_TOKEN\_ID);

RESULT\_VALUE := 1;

ELSIF (COUNTER > 0 AND DATECHECK = 1)

THEN

RESULT\_VALUE :=-1;

ELSIF ( (COUNTER = 0 OR COUNTER = 1) AND DATECHECK = 0)

THEN

RESULT\_VALUE := -2;

END IF;

IF (COURSECHECK != 1)

THEN

RESULT\_VALUE := -3;

END IF;

END;

CREATE OR REPLACE PROCEDURE SUBMIT\_CLCK(HWID IN NUMBER,

LOGIN\_ID IN VARCHAR2,

MARKS\_OBTAINED IN NUMBER,

COURSETOKEN\_ID IN VARCHAR2) AS

ATMPT NUMBER;

TIMESTAMPVAL DATE;

USER\_ID NUMBER;

GPA\_VAL NUMBER(20,2);

CGPA\_VAL NUMBER(20,2);

MAX\_MARKS NUMBER;

STUDENTMARKS NUMBER;

STUDENTTOTALMARKS NUMBER;

TOTALMARKS NUMBER;

TOTALMAXMARKS NUMBER;

BEGIN

SELECT USERID INTO USER\_ID FROM USERS WHERE LOGINID =LOGIN\_ID;

SELECT MAX(ATTEMPTID) INTO ATMPT FROM STUDENTATTEMPT WHERE EXERCISEID =HWID AND USERID = USER\_ID;

SELECT SYSDATE INTO TIMESTAMPVAL FROM DUAL;

ATMPT := ATMPT + 1;

INSERT INTO STUDENTATTEMPT VALUES(HWID, USER\_ID, ATMPT, MARKS\_OBTAINED, TIMESTAMPVAL);

END;

CREATE OR REPLACE PROCEDURE SUBMIT\_CLCK2(

HWID IN NUMBER,

LOGIN\_ID IN VARCHAR2,

QUESTION\_ID IN NUMBER,

OPTION\_1 IN NUMBER,

OPTION\_2 IN NUMBER,

OPTION\_3 IN NUMBER,

OPTION\_4 IN NUMBER,

OPTION\_SEL IN NUMBER,

SET\_ID IN NUMBER

)

AS

ATMPT NUMBER;

USER\_ID NUMBER;

BEGIN

SELECT USERID INTO USER\_ID FROM USERS WHERE LOGINID =LOGIN\_ID;

SELECT MAX(ATTEMPTID) INTO ATMPT FROM STUDENTATTEMPT WHERE EXERCISEID =HWID AND USERID = USER\_ID;

INSERT INTO STUDENTATTEMPTQUESTIONS VALUES(HWID, USER\_ID, ATMPT, QUESTION\_ID, OPTION\_1, OPTION\_2, OPTION\_3, OPTION\_4, OPTION\_SEL,SET\_ID);

END;

CREATE OR REPLACE PROCEDURE UPDATE\_ACCOUNT(LOGINID IN VARCHAR2, PASSWORDVAL IN VARCHAR2, FIRSTNAME IN VARCHAR2, LASTNAME IN VARCHAR2, EMAILADDRESS IN VARCHAR2,UPDATEDVAL OUT NUMBER) AS

BEGIN

IF (LOGINID = LOGINID)

THEN

UPDATE USERS

SET PASSWORDVAL=PASSWORDVAL, FIRSTNAME=FIRSTNAME, LASTNAME = LASTNAME, EMAILADDRESS=EMAILADDRESS

WHERE LOGINID=LOGINID;

UPDATEDVAL:=1;

ELSE

UPDATEDVAL:=-1;

END IF;

IF(UPDATEDVAL>0)

THEN

DBMS\_OUTPUT.PUT\_LINE('SUCCESSFUL');

ELSE

DBMS\_OUTPUT.PUT\_LINE('UNSUCCESSFUL');

END IF;

END;

CREATE OR REPLACE PROCEDURE UPDATE\_HW (

EXERCISE\_ID IN NUMBER,

STARTDATE IN DATE,

ENDDATE IN DATE,

NUMRETRIES IN NUMBER,

CRCTANS IN NUMBER,

INCRCTANS IN NUMBER,

SCORSELSCHEME IN VARCHAR2,

DIFFLVL IN NUMBER) AS

BEGIN

UPDATE EXERCISES

SET STARTDATETIME = STARTDATE,

ENDDATETIME = ENDDATE,

NUMBEROFRETRIES = NUMRETRIES,

MARKSPERCORRECTANSWER = CRCTANS,

MARKSPERINCORRECTANSWER = INCRCTANS,

SCORINGTYPE = SCORSELSCHEME,

DIFFICULTYLEVEL = DIFFLVL

WHERE EXERCISEID = EXERCISE\_ID;

END;

CREATE OR REPLACE PROCEDURE UPDATE\_PWD(LOGIN\_ID IN VARCHAR2, NEWPASSWORD IN VARCHAR2, OLDPASSWORD IN VARCHAR2,UPDATEDVAL OUT NUMBER) AS

OLDPASSWORD\_DB VARCHAR(10);

BEGIN

SELECT PASSWORDVAL INTO OLDPASSWORD\_DB FROM USERS WHERE LOGINID = LOGIN\_ID;

IF(OLDPASSWORD = OLDPASSWORD\_DB AND OLDPASSWORD <> NEWPASSWORD)

THEN

UPDATE USERS

SET PASSWORDVAL=NEWPASSWORD

WHERE LOGINID=LOGIN\_ID;

UPDATEDVAL:=1;

ELSE

UPDATEDVAL:=-1;

END IF;

IF(UPDATEDVAL>0)

THEN

DBMS\_OUTPUT.PUT\_LINE('SUCCESSFULLY UPDATED PASSWORD');

ELSE

DBMS\_OUTPUT.PUT\_LINE('UNSUCCESSFUL IN UPDATING PASSWORD');

END IF;

END;

CREATE OR REPLACE PROCEDURE UPDATENOTIFICATIONS(LOGIN\_ID IN VARCHAR2, COURSETOKEN\_ID IN VARCHAR2) AS

USER\_ID NUMBER;

BEGIN

SELECT USERID INTO USER\_ID FROM USERS WHERE LOGINID = LOGIN\_ID;

UPDATE NOTIFICATIONUSERMAPPING SET READFLAG =0 WHERE USERID = USER\_ID AND NOTIFICATIONID IN (

SELECT N.NOTIFICATIONID

FROM NOTIFICATIONS N, NOTIFICATIONUSERMAPPING NUM

WHERE N.NOTIFICATIONID = NUM.NOTIFICATIONID

AND NUM.USERID = USER\_ID AND N.COURSETOKENID=COURSETOKEN\_ID AND NUM.READFLAG =1);

END;

CREATE OR REPLACE PROCEDURE VIEW\_HW(HWID IN NUMBER, C1 OUT SYS\_REFCURSOR) AS

BEGIN

OPEN C1 FOR

SELECT E.\*, (SELECT COUNT(EL.COURSETOPICID)

FROM EXERCISES E, EXERCISELIST EL

WHERE E.EXERCISEID = EL.EXERCISEID AND E.EXERCISEID = HWID

GROUP BY (EL.EXERCISEID)) AS NUMTOPICS, (SELECT COUNT(EQ.QUESTIONID)

FROM EXERCISES E, EXERCISEQUESTION EQ

WHERE E.EXERCISEID = EQ.EXERCISEID AND E.EXERCISEID = HWID

GROUP BY (EQ.EXERCISEID)) AS NUMQUES

FROM EXERCISES E

WHERE E.EXERCISEID = HWID;

END;

CREATE OR REPLACE PROCEDURE VIEWHW\_TOPICS(HWID IN NUMBER,C1 OUT SYS\_REFCURSOR) AS

BEGIN

OPEN C1 FOR

SELECT COURSETOPICNAME FROM COURSETOPICS WHERE COURSETOPICID IN (SELECT EXERCISELIST.COURSETOPICID FROM EXERCISELIST WHERE EXERCISEID = HWID);

END;

**FUNCTIONS :**

CREATE OR REPLACE FUNCTION MAPLOGINIDTOUSERID(LOGIN\_ID VARCHAR2)RETURN NUMBER

IS

USER\_ID NUMBER;

BEGIN

SELECT USERID INTO USER\_ID FROM USERS WHERE LOGINID = LOGIN\_ID;

RETURN USER\_ID;

END;

**TYPES**

CREATE OR REPLACE TYPE COURSE\_LIST AS TABLE OF NUMBER;

CREATE OR REPLACE TYPE COURSETOPIC\_LIST AS TABLE OF NUMBER;

CREATE OR REPLACE TYPE PROC\_OUTPUT AS TABLE OF VARCHAR2(50);

CREATE OR REPLACE TYPE QUESTION\_LIST AS TABLE OF NUMBER;

CREATE OR REPLACE TYPE REPCAT$\_OBJECT\_NULL\_VECTOR AS OBJECT

(

-- TYPE OWNER, NAME, HASHCODE FOR THE TYPE REPRESENTED BY NULL\_VECTOR

TYPE\_OWNER VARCHAR2(30),

TYPE\_NAME VARCHAR2(30),

TYPE\_HASHCODE RAW(17),

-- NULL\_VECTOR FOR A PARTICULAR OBJECT INSTANCE

-- ROBJ REVISIT: SHOULD ONLY CONTAIN THE NULL IMAGE, AND NOT VERSION#

NULL\_VECTOR RAW(2000)

)

CREATE OR REPLACE TYPE SETS\_LIST AS TABLE OF NUMBER;

**JOBS**

BEGIN

DBMS\_SCHEDULER.CREATE\_JOB(

JOB\_NAME => 'PASSWORD\_CHANGE',

JOB\_TYPE => 'STORED\_PROCEDURE',

JOB\_ACTION => 'ADDNOTIFICATION',

START\_DATE => SYSDATE ,

REPEAT\_INTERVAL => 'FREQ=QUARTERLY;INTERVAL=1' ,

ENABLED => TRUE,

COMMENTS => 'DEMO FOR JOB SCHEDULE.');

END;

BEGIN

DBMS\_SCHEDULER.CREATE\_JOB(

JOB\_NAME => 'PASSWORD\_CHANGE',

JOB\_TYPE => 'STORED\_PROCEDURE',

JOB\_ACTION => HOMEWORKNOTIFICATION,

START\_DATE => SYSDATE ,

REPEAT\_INTERVAL => 'FREQ=MINUTELY;INTERVAL=1' ,

ENABLED => TRUE,

COMMENTS => 'DEMO FOR JOB SCHEDULE.');

END;