create or replace PACKAGE "PACKAGE\_SRS"

AS

type ref\_show\_all\_tuples

IS

ref

CURSOR;

PROCEDURE proc\_show\_all\_tables(

in\_table\_name VARCHAR2,

out\_prc OUT sys\_refcursor );

PROCEDURE proc\_show\_student\_info(

in\_student\_B# IN students.B#%type,

out\_prc OUT sys\_refcursor ) ;

TYPE ref\_cursor

IS

REF

CURSOR;

PROCEDURE proc\_find\_dependent\_courses(

c\_dept\_code IN PREREQUISITES.pre\_dept\_code%TYPE,

c\_course# IN PREREQUISITES.pre\_course#%TYPE,

out\_prc OUT sys\_refcursor );

PROCEDURE enroll\_student(

stud\_B# IN STUDENTS.B#%TYPE,

stud\_classid IN CLASSES.CLASSID%TYPE,

msg out varchar2);

FUNCTION GIVE\_RESPECTIVE\_NGRADE

(

IN\_LGRADE IN VARCHAR2

) RETURN VARCHAR2;

PROCEDURE PROC\_DELETE\_STUDENT (p\_students\_b# IN STUDENTS.B#%TYPE);

PROCEDURE PROC\_DELETE\_ENROLLMENT(

p\_B# IN ENROLLMENTS.B#%TYPE,

p\_classid IN ENROLLMENTS.CLASSID%TYPE,

msg out varchar2);

procedure proc\_show\_class\_details (p\_class\_id in Classes.classid%type,

out\_prc OUT sys\_refcursor);

END;

create or replace PACKAGE BODY "PACKAGE\_SRS"

AS

--fucntion to check is class valid?

FUNCTION is\_class\_present(

p\_class\_classid IN CLASSES.CLASSID%TYPE)

RETURN INTEGER

IS

l\_class\_count INTEGER;

BEGIN

SELECT COUNT(\*)

INTO l\_class\_count

FROM CLASSES

WHERE CLASSID = p\_class\_classid;

RETURN (l\_class\_count);

END;

--fucntion to check is student valid?

FUNCTION is\_student\_present(

p\_student\_B# IN Students.B#%type)

RETURN INTEGER

IS

l\_student\_count INTEGER;

BEGIN

SELECT COUNT(\*) INTO l\_student\_count FROM Students WHERE B# = p\_student\_B#;

RETURN (l\_student\_count);

END;

--fucntion to check is student enrolled?

FUNCTION is\_student\_enrolled(

p\_class\_classid IN CLASSES.CLASSID%TYPE,

p\_student\_B# Students.B#%type)

RETURN INTEGER

IS

l\_enrollment\_count INTEGER;

BEGIN

SELECT COUNT(\*)

INTO l\_enrollment\_count

FROM enrollments

WHERE CLASSID = p\_class\_classid

AND B# = p\_student\_B#;

RETURN (l\_enrollment\_count);

END;

--fucntion to check is any student enrolled?

FUNCTION is\_any\_student\_enrolled(

p\_class\_classid IN CLASSES.CLASSID%TYPE)

RETURN INTEGER

IS

l\_enrollment\_count INTEGER;

BEGIN

SELECT COUNT(\*)

INTO l\_enrollment\_count

FROM enrollments

WHERE CLASSID = p\_class\_classid;

RETURN (l\_enrollment\_count);

END;

--fucntion to check is any class taken?

FUNCTION is\_any\_class\_taken(

p\_student\_B# Students.B#%type)

RETURN INTEGER

IS

l\_taken\_count INTEGER;

BEGIN

SELECT COUNT(\*)

INTO l\_taken\_count

FROM enrollments

WHERE B# = p\_student\_B#;

RETURN (l\_taken\_count);

END;

--fucntion to give respective ngrade from lgrade

FUNCTION GIVE\_RESPECTIVE\_NGRADE(

IN\_LGRADE IN VARCHAR2 )

RETURN VARCHAR2

AS

var\_out\_ngrade NUMBER;

BEGIN

SELECT ngrade INTO var\_out\_ngrade FROM grades WHERE LGRADE = IN\_LGRADE;

RETURN var\_out\_ngrade;

END GIVE\_RESPECTIVE\_NGRADE;

-- Q2

-- procedures to display the tuples in each of the eight tables

PROCEDURE proc\_show\_all\_tables(

in\_table\_name VARCHAR2,

out\_prc OUT sys\_refcursor )

IS

BEGIN

-- case stmt to select from table

CASE in\_table\_name

WHEN 'students' THEN

OPEN out\_prc FOR SELECT \* FROM students;

WHEN 'courses' THEN

OPEN out\_prc FOR SELECT \* FROM courses;

WHEN 'course\_credit' THEN

OPEN out\_prc FOR SELECT \* FROM course\_credit;

WHEN 'prerequisites' THEN

OPEN out\_prc FOR SELECT \* FROM prerequisites;

WHEN 'classes' THEN

OPEN out\_prc FOR SELECT \* FROM classes;

WHEN 'enrollments' THEN

OPEN out\_prc FOR SELECT \* FROM enrollments;

WHEN 'grades' THEN

OPEN out\_prc FOR SELECT \* FROM grades;

WHEN 'logs' THEN

OPEN out\_prc FOR SELECT \* FROM logs;

END CASE;

END proc\_show\_all\_tables;

-- Q 3

-- procedure for a given student (with B# provided as a parameter),

-- to list every class the student has taken or is taking

-- and report if any error

PROCEDURE proc\_show\_student\_info(

in\_student\_B# IN students.B#%type,

out\_prc OUT sys\_refcursor )

IS

student\_not\_available\_excp EXCEPTION;

course\_not\_taken\_excp EXCEPTION;

BEGIN

IF (IS\_STUDENT\_PRESENT(in\_student\_B#) <> 1) THEN

raise student\_not\_available\_excp;

END IF;

IF (is\_any\_class\_taken(in\_student\_B#) = 0) THEN

raise course\_not\_taken\_excp;

END IF;

OPEN out\_prc FOR

SELECT c.CLASSID

AS

CLASSID,

c.DEPT\_CODE

AS

DEPT\_CODE,

c.COURSE#

AS

COURSE#,

c.SECT#

AS

SECT#,

c.YEAR

AS

YEAR,

c.SEMESTER

AS

semester,

e.LGRADE

AS

LGRADE,

GIVE\_RESPECTIVE\_NGRADE(e.LGRADE)

AS

ngrade

FROM enrollments e INNER JOIN classes c ON e.classid = c.classid WHERE e.b# =in\_student\_B#;

Exception

-- raise required exception

WHEN student\_not\_available\_excp THEN

RAISE\_APPLICATION\_ERROR(-20001, 'The B# is invalid.');

WHEN course\_not\_taken\_excp THEN

RAISE\_APPLICATION\_ERROR(-20002, 'The student has not taken any course');

END proc\_show\_student\_info;

-- Q4

-- procedure, for a given course

-- (with dept\_code and course# as parameters),

-- can return all courses that need this course as a prerequisite

PROCEDURE proc\_find\_dependent\_courses(

c\_dept\_code IN PREREQUISITES.pre\_dept\_code%TYPE,

c\_course# IN PREREQUISITES.pre\_course#%TYPE,

out\_prc OUT sys\_refcursor)

IS

BEGIN

OPEN out\_prc FOR SELECT (dept\_code || course#)

AS

course FROM PREREQUISITES

START WITH pre\_dept\_code= c\_dept\_code AND pre\_course# = c\_course#

CONNECT BY PRIOR dept\_code = pre\_dept\_code AND PRIOR course# = pre\_course#;

END proc\_find\_dependent\_courses;

-- Q6

-- procedure, to entoll student

PROCEDURE enroll\_student(

stud\_B# IN STUDENTS.B#%TYPE,

stud\_classid IN CLASSES.CLASSID%TYPE,

msg out varchar2)

IS

tupleCount NUMBER(1);

invalid\_user EXCEPTION;

invalid\_class EXCEPTION;

duplicate\_enrollment EXCEPTION;

enrollment\_limit\_exceeded EXCEPTION;

enrollment\_overloaded EXCEPTION;

prereq\_requirement\_violation EXCEPTION;

BEGIN

SELECT COUNT(\*) INTO tupleCount FROM STUDENTS s WHERE s.B# = stud\_B#;

IF tupleCount = 0 THEN

RAISE invalid\_user;

END IF;

SELECT COUNT(\*) INTO tupleCount FROM CLASSES c WHERE c.CLASSID = stud\_classid;

IF tupleCount = 0 THEN

RAISE invalid\_class;

END IF;

SELECT COUNT(\*)

INTO tupleCount

FROM ENROLLMENTS e

WHERE e.B# = stud\_B#

AND e.classid = stud\_classid;

IF tupleCount = 1 THEN

RAISE duplicate\_enrollment;

END IF;

SELECT COUNT(\*)

INTO tupleCount

FROM ENROLLMENTS e,

CLASSES c

WHERE e.classid = c.classid

AND e.B# = stud\_B#

AND c.classid IN

(SELECT classid

FROM CLASSES

WHERE (semester, YEAR) =

(SELECT semester, YEAR FROM CLASSES WHERE classid = stud\_classid

)

);

IF tupleCount = 4 THEN

RAISE enrollment\_limit\_exceeded;

ELSIF tupleCount = 3 THEN

--RAISE enrollment\_overloaded;

DBMS\_OUTPUT.PUT\_LINE('You are overloaded');

msg := 'You are overloaded';

END IF;

SELECT COUNT(\*)

INTO tupleCount

FROM ENROLLMENTS e

WHERE e.B# = stud\_B#

AND classid IN

(SELECT classid

FROM CLASSES c

WHERE (dept\_code, course#) IN

(SELECT pre\_dept\_code,

pre\_course#

FROM PREREQUISITES

WHERE (dept\_code, course#) =

(SELECT dept\_code, course# FROM CLASSES WHERE classid = stud\_classid

)

)

)

AND LGRADE IN ('C-', 'D', 'F', 'I');

IF tupleCount > 0 THEN

RAISE prereq\_requirement\_violation;

END IF;

INSERT INTO ENROLLMENTS VALUES

(stud\_B#, stud\_classid, NULL

);

EXCEPTION

WHEN invalid\_user THEN

RAISE\_APPLICATION\_ERROR(-20001, 'The B# number is invalid.');

WHEN invalid\_class THEN

RAISE\_APPLICATION\_ERROR(-20002, 'The classid is invalid.');

WHEN duplicate\_enrollment THEN

RAISE\_APPLICATION\_ERROR(-20003, 'The student is already in the class.');

WHEN enrollment\_limit\_exceeded THEN

RAISE\_APPLICATION\_ERROR(-20004, 'Students cannot be enrolled in more than four classes in the semester.');

-- WHEN enrollment\_overloaded THEN

-- RAISE\_APPLICATION\_ERROR(2000, 'You are overloaded.');

WHEN prereq\_requirement\_violation THEN

RAISE\_APPLICATION\_ERROR(-20005, 'Prerequisite not satisfied.');

END enroll\_student;

-- funtion to check is prerequisite

FUNCTION check\_prerequisites

(

p\_classes\_classid IN CLASSES.CLASSID%type,

p\_student\_B# IN STUDENTS.B#%type

)

RETURN INTEGER

IS

l\_dept\_code CLASSES.DEPT\_CODE%type;

l\_course# Classes.course#%type;

CURSOR c1

IS

SELECT DISTINCT p.PRE\_COURSE#,

p.pre\_dept\_code

FROM

(SELECT \* FROM enrollments NATURAL JOIN classes WHERE B# = p\_student\_B#

) temp

JOIN prerequisites p

ON temp.course# = p.course#

AND temp.dept\_code = p.dept\_code;

BEGIN

SELECT dept\_code,

Course#

INTO l\_dept\_code,

l\_course#

FROM Classes NATURAL

JOIN Courses

WHERE classid = p\_classes\_classid;

FOR c1\_record IN c1

LOOP

IF(c1\_record.pre\_course# = l\_course# AND c1\_record.pre\_dept\_code = l\_dept\_code) THEN

RETURN -1;

end if;

END LOOP;

-- ELSE

RETURN 1;

--END IF;

END;

-- Q7

-- a procedure to drop a student from a class

-- and report respective error if applicable

PROCEDURE PROC\_DELETE\_ENROLLMENT(

p\_B# IN ENROLLMENTS.B#%TYPE,

p\_classid IN ENROLLMENTS.CLASSID%TYPE,

msg out varchar2)

IS

l\_student\_count INTEGER DEFAULT 0;

l\_enrollment\_count INTEGER DEFAULT 0;

l\_student\_classes\_count INTEGER DEFAULT 0;

l\_classes\_student\_count INTEGER DEFAULT 0;

class\_not\_available\_excp EXCEPTION;

student\_not\_available\_excp EXCEPTION;

student\_not\_enrolled\_excp EXCEPTION;

drop\_not\_permitted\_excp EXCEPTION;

BEGIN

IF (IS\_CLASS\_PRESENT(p\_classid) <> 1) THEN

raise class\_not\_available\_excp;

END IF;

IF (IS\_STUDENT\_PRESENT(p\_B#) <> 1) THEN

raise student\_not\_available\_excp;

END IF;

IF (IS\_STUDENT\_ENROLLED(p\_classid, p\_B#) = 0) THEN

raise student\_not\_enrolled\_excp;

END IF;

IF (check\_prerequisites(p\_classid, p\_B#) = -1) THEN

raise drop\_not\_permitted\_excp;

END IF;

DELETE FROM enrollments WHERE classid = p\_classid AND B# = p\_B#;

SELECT COUNT(\*) INTO l\_student\_classes\_count FROM enrollments WHERE B# = p\_B#;

IF (l\_student\_classes\_count = 0) THEN

DBMS\_OUTPUT.PUT\_LINE('the student is not enrolled in any classes');

msg := 'the student is not enrolled in any classes';

END IF;

SELECT COUNT(\*)

INTO l\_classes\_student\_count

FROM enrollments

WHERE classid = p\_classid;

IF (l\_classes\_student\_count = 0) THEN

DBMS\_OUTPUT.PUT\_LINE('the class has no more students enrolled');

msg := 'the class has no more students enrolled';

END IF;

EXCEPTION

WHEN class\_not\_available\_excp THEN

RAISE\_APPLICATION\_ERROR(-20001, 'The classid is invalid.');

WHEN student\_not\_available\_excp THEN

RAISE\_APPLICATION\_ERROR(-20002, 'The B# is invalid.');

WHEN student\_not\_enrolled\_excp THEN

RAISE\_APPLICATION\_ERROR(-20003, 'The student is not enrolled in the class.');

WHEN drop\_not\_permitted\_excp THEN

RAISE\_APPLICATION\_ERROR(-20004, 'The drop is not permitted because another class uses it as a prerequisite.');

END PROC\_DELETE\_ENROLLMENT;

-- Q8

-- a procedure to drop a student from students table

-- and report respective error if applicable

PROCEDURE PROC\_DELETE\_STUDENT(

p\_students\_b# IN STUDENTS.B#%TYPE)

IS

student\_not\_available\_excp EXCEPTION;

l\_classes\_student\_count INTEGER DEFAULT 0;

BEGIN

IF (IS\_STUDENT\_PRESENT(p\_students\_b#) <> 1) THEN

raise student\_not\_available\_excp;

END IF;

delete from students where B#= p\_students\_b#;

-- SELECT COUNT(\*)

-- INTO l\_classes\_student\_count

-- FROM enrollments

-- WHERE classid = p\_classid;

-- IF (l\_classes\_student\_count = 0) THEN

-- DBMS\_OUTPUT.PUT\_LINE('the class has no more students enrolled');

-- END IF;

EXCEPTION

WHEN student\_not\_available\_excp THEN

RAISE\_APPLICATION\_ERROR(-20002, 'The B# is invalid.');

END PROC\_DELETE\_STUDENT;

-- Q 5

-- procedure that, for a given class (with classid provided as

-- a parameter), can list the classid and course title of the class

-- as well as all the students (show B# and firstname) who took or are taking the class

PROCEDURE proc\_show\_class\_details (

p\_class\_id in Classes.classid%type,

out\_prc OUT sys\_refcursor)

IS

class\_invalid\_excp exception;

no\_student\_enrolled\_excp exception;

BEGIN

-- If the class is not in the classes table, report “The classid is invalid.

IF (IS\_CLASS\_PRESENT(p\_class\_id) <> 1) THEN

raise class\_invalid\_excp;

END IF;

--If no student took or is taking the class, report No student has enrolled in the class.

if( is\_any\_student\_enrolled(p\_class\_id) = 0 ) then

raise no\_student\_enrolled\_excp;

end if;

open out\_prc

for

SELECT Students.B# as B#,

Students.FIRSTNAME as FIRSTNAME,

Classes.Classid as classid,

COURSES.TITLE as title

FROM COURSES

JOIN CLASSES

ON CLASSES.DEPT\_CODE = COURSES.DEPT\_CODE

AND CLASSES.COURSE# = COURSES.COURSE#

JOIN Enrollments

ON Classes.Classid = Enrollments.Classid

JOIN Students

ON Students.B# = Enrollments.B#

WHERE Classes.classid = p\_class\_id;

EXCEPTION

WHEN class\_invalid\_excp THEN

RAISE\_APPLICATION\_ERROR(-20001, 'The classid is invalid');

WHEN no\_student\_enrolled\_excp THEN

RAISE\_APPLICATION\_ERROR(-20002, 'No student has enrolled in the class');

END proc\_show\_class\_details;

END;

CREATE SEQUENCE seq\_logs INCREMENT BY 1 START WITH 1000;

/

show errors;

CREATE OR REPLACE TRIGGER TRIGGER\_INSERT\_ENROLLMENTS

AFTER INSERT

ON ENROLLMENTS

FOR EACH ROW

DECLARE

t\_keyval LOGS.KEY\_VALUE%TYPE;

BEGIN

t\_keyval := :new.B# || ',' || :new.classid;

-- dbms\_output.put\_line(t\_keyval);

INSERT INTO LOGS (LOGID, WHO, TIME, TABLE\_NAME, OPERATION, KEY\_VALUE)

VALUES (seq\_logs.NEXTVAL, user, SYSDATE, 'Enrollments', 'Insert', t\_keyval);

END;

/

CREATE OR REPLACE TRIGGER TRIGGER\_DELETE\_ENROLLMENTS

AFTER DELETE

ON ENROLLMENTS

FOR EACH ROW

DECLARE

t\_keyval LOGS.KEY\_VALUE%TYPE;

BEGIN

t\_keyval := :old.B# || ',' || :old.classid;

-- dbms\_output.put\_line(t\_keyval);

INSERT INTO LOGS (LOGID, WHO, TIME, TABLE\_NAME, OPERATION, KEY\_VALUE)

VALUES (seq\_logs.NEXTVAL, user, SYSDATE, 'Enrollments', 'Delete', t\_keyval);

END;

/

--create trigger TRIGGER\_LOGS\_ENROLLMENTS

-- This trigger will log entry for student table insert

create or replace TRIGGER trigger\_insert\_student

AFTER INSERT

ON STUDENTS

FOR EACH ROW

DECLARE

t\_keyval LOGS.KEY\_VALUE%TYPE;

BEGIN

t\_keyval := :new.B#;

--dbms\_output.put\_line(t\_keyval);

INSERT INTO LOGS (LOGID, WHO, TIME, TABLE\_NAME, OPERATION, KEY\_VALUE)

VALUES (seq\_logs.NEXTVAL, user, SYSDATE, 'Students', 'Insert', t\_keyval);

END;

/

--create trigger class\_size

-- This trigger will log entry for class size increase

CREATE OR REPLACE TRIGGER TRIGGER\_INC\_CLASS\_SIZE

before INSERT ON ENROLLMENTS

FOR EACH ROW

DECLARE

old\_class\_size CLASSES.CLASS\_SIZE%TYPE;

l\_limit CLASSES.LIMIT%TYPE;

class\_full EXCEPTION;

BEGIN

SELECT class\_size, limit INTO old\_class\_size, l\_limit

FROM CLASSES

WHERE classid = :new.classid;

IF (old\_class\_size = l\_limit) THEN

raise class\_full;

ELSE

UPDATE CLASSES

SET class\_size = old\_class\_size + 1

WHERE classid = :new.classid;

END IF;

--dbms\_output.put\_line('trigger TRIGGER\_INC\_CLASS\_SIZE executed...');

EXCEPTION

WHEN class\_full

THEN RAISE\_APPLICATION\_ERROR(-20008, 'The class is full.');

END;

/

create or replace TRIGGER TRIGGER\_DEC\_CLASS\_SIZE AFTER

DELETE ON ENROLLMENTS FOR EACH ROW

DECLARE

old\_class\_size CLASSES.CLASS\_SIZE%TYPE;

BEGIN

SELECT class\_size

INTO old\_class\_size

FROM CLASSES

WHERE classid = :old.classid;

UPDATE CLASSES

SET class\_size = old\_class\_size - 1

WHERE classid = :old.classid;

--dbms\_output.put\_line('trigger TRIGGER\_DEC\_CLASS\_SIZE executed...');

END;

/

create or replace TRIGGER trigger\_delete\_student

AFTER DELETE

ON STUDENTS

FOR EACH ROW

DECLARE

l\_classid varchar2(200);

t\_keyval LOGS.KEY\_VALUE%TYPE;

CURSOR c1

IS

SELECT classid FROM ENROLLMENTS WHERE B# = :old.B#;

c1\_rec c1%rowtype;

BEGIN

if(not c1%isopen) then

open c1;

end if;

fetch c1 into c1\_rec;

while c1%found loop

l\_classid := c1\_rec.classid;

delete from enrollments where B# = :old.B# and classid = l\_classid;

fetch c1 into c1\_rec;

END LOOP;

close c1;

t\_keyval := :old.B#;

--dbms\_output.put\_line(t\_keyval);

INSERT INTO LOGS (LOGID, WHO, TIME, TABLE\_NAME, OPERATION, KEY\_VALUE)

VALUES (seq\_logs.NEXTVAL, user, SYSDATE, 'Students', 'Delete', t\_keyval);

END;

/

show errors;