```
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;
```