Midterm Project
Part I
Instructions for Instructors

The midterm project is divided into 3 parts. You can use the 3 parts in various ways depending on how much time you wish to devote to this project or on the technical level of your students.

- All 3 parts can be assigned to individual students. The order of completion among the 3 parts does not matter.
- The class can be divided into teams, with each team completing Part 1 or Part2
- Part 3 can be used in conjunction with Parts 1 or 2 for more advanced students/classes.

Make sure the students save their work in files, so you can check them and so that they can easily be amended to packages, functions and packages in later projects.

Project setup: The Data

This project will use a case study called STUDENT ADMINISTRATION or SA. A set of database tables is used to manage a school's course offerings as delivered by instructors in many classes over time. Information is stored about classes that are offered, the students who take classes, and the grades the students receive on various assessments. The school administrators can use the SA database to manage the class offerings and to assign instructors. Teachers can also use the SA database to track student performance.

The database objects for this project are already in your accounts and they are as follows:

Tables:

INSTRUCTORS
SECTIONS
COURSES
CLASSES
ASSESSMENTS
STUDENTS
ENROLLMENTS
CLASS_ASSESSMENTS
ERROR_LOG
GRADE_CHANGES

Sequence:

ASSESSMENT_ID_SEQ

Synonyms:

- sect FOR sections
- instr FOR instructors
- enroll FOR enrollments
- stu FOR students
- cl_assess FOR class_assessments
- cl FOR classes
- cour FOR courses
- assess FOR assessments

Part 1: Student Information

1. Create an anonymous PL/SQL block to enroll a student in a particular class. Make sure the students save it in a file called *enroll_student_in_class.sql* . You will use the ENROLLMENTS table. Accept a STU_ID and CLASS_ID as input parameters. Use “today’s date” for the ENROLLMENT_DATE and the string ‘Enrolled’ for the STATUS.

   Tables used: ENROLLMENTS
   Topics Incorporated ==> Scalar variable; %TYPE, INSERT; COMMIT;

   Suggested Solution:

```
DECLARE
v_stu_id           enrollments.stu_id%TYPE := :student_id;
v_class_id         enrollments.class_id%TYPE := :class_id;
v_times_enrolled  PLS_INTEGER;
BEGIN
 SELECT COUNT(*) INTO v_times_enrolled
  FROM enrollments
  WHERE class_id = v_class_id
  AND stu_id   = v_stu_id;

 INSERT INTO enrollments
  (enrollment_date, class_id, stu_id, status)
 VALUES
  (SYSDATE, v_class_id, v_stu_id, 'Enrolled');
COMMIT;

END;
```

2. Create an anonymous block to drop a student from a class. Save the block in a file called *drop_student_from_class.sql*. You will use the ENROLLMENTS table. Accept a STU_ID and CLASS_ID as input parameters.

   Tables used: ENROLLMENTS
   Topics Incorporated ==> %TYPE; DELETE; SQL%ROWCOUNT; COMMIT;

Suggested Solution:

```
DECLARE
v_stu_id    enrollments.stu_id%TYPE := :stu_id;
v_class_id  enrollments.class_id%TYPE := :class_id;
BEGIN
 DELETE FROM enrollments
   WHERE class_id = v_class_id AND stu_id = v_stu_id;
 IF SQL%ROWCOUNT = 0 THEN
  DBMS_OUTPUT.PUT_LINE('Student '|| v_stu_id ||
  ' is not enrolled in class '|| v_class_id);
 END IF;
 COMMIT;

END;
```

3. Create an anonymous block that displays all of the classes a student has been enrolled in within the most recent 6 years. Save the block in a file *student_class_list.sql*. You will use the ENROLLMENTS table. For example: If you run your program on May 10, 2006, you should display all enrollments between May 10, 2000 and May 10, 2006. Accept the STU_ID as an input parameter. For each enrollment, display the ENROLLMENT_DATE, CLASSS_ID and STATUS.

   Tables used: ENROLLMENTS
   Topics Incorporated ==> %TYPE; SYSDATE, ADD_MONTHS, Explicit cursor to find all courses for the provided Student ID;DBMS_OUTPUT to display the list of classes.

   Suggested Solution:

```
DECLARE
 v_stu_id   enrollments.stu_id%TYPE := :stu_id;
CURSOR stu_class_cur IS
 SELECT enrollment_date, class_id, status
  FROM enrollments
  WHERE stu_id = v_stu_id
  AND enrollment_date
       between ADD_MONTHS (SYSDATE,-72) and SYSDATE;
BEGIN
 DBMS_OUTPUT.PUT_LINE('Student ' || v_stu_id ||
        ' is enrolled in the following classes:');
 FOR stu_class_rec IN stu_class_cur LOOP
   DBMS_OUTPUT.PUT_LINE('Class: ' ||stu_class_rec.class_id ||
'  Enrolled on: ' || stu_class_rec.enrollment_date ||
' and has a status of: '|| stu_class_rec.status );
 END LOOP;
 END;
```

4. Create an anonymous block to add “n” new classes for a particular course. Save the block in a file called *add_new_classes.sql* Accept the following IN parameter:
   - Number of new classes required. Set a default value of 1.
   - Course id; For each new class, use “today”as the START_DATE.
   - Period, to specify what days the class meets.
   - Frequency, to specify how often it meets.
   - Instructor id, who is teaching the class(s).

   There are 2 ways you can generate a new CLASS_ID.
   - First use a SELECT to find out what is currently the highest CLASS_ID. Increment this number by 1 for each new class you add. For example: When you run your program, the highest CLASS_ID is 12 and you want to create 3 new classes. Your new CLASS_IDs would be 13, 14, and 15.
   - Create a sequence number that starts with a number higher than the maximum CLASS_ID. Use sequence_name.NEXTVAL in your INSERT statement.
   - Test your program by adding 4 new classes for COURSE_ID #1001. Also test your default by calling the program to add only one class for COURSE_ID #1002.

   Note for the teacher: It is probably more likely and safer to use a sequence number for the class_id. If 2 sessions are calling this program at the same time, there is a small chance the same MAX(class_id) could be found. This would cause the second set of INSERTS to fail. The first option is okay for this exercise, but you might discuss this interesting subtlety with your students.

   Tables used: CLASSES
   Topics Incorporated ==> SELECT; INSERT; COMMIT; DEFAULT value; Using a LOOP to only insert “n” new rows.

   Suggested Solution 1:

```
DECLARE
 v_number_new_classes      PLS_INTEGER :=
    NVL(:number_of_new_cl, 1);
 v_course_id               classes.course_id%TYPE :=
    :course_id;
 v_period             classes.period%TYPE := :period;
 v_frequency               classes.frequency%TYPE :=
    :frequency;
 v_instructor              classes.instr_id%TYPE :=
    :instr_id;
 v_current_max_class_id    classes.class_id%TYPE;
BEGIN
 SELECT MAX(class_id)
  INTO v_current_max_class_id
  FROM classes;
 FOR loop_counter IN 1..v_number_new_classes LOOP
  INSERT INTO classes (class_id, start_date,course_id,
                       period, frequency,instr_id)
  VALUES (v_current_max_class_id + loop_counter,
                        SYSDATE, v_course_id,
                        v_period,v_frequency,v_instructor);
  COMMIT;
 END LOOP;

END;
```

Suggested Solution 2:

```
SELECT MAX(class_id)
 FROM classes;
CREATE sequence class_id_seq
       START with number_returned_from_SELECT + 1
       INCREMENT BY 1
       NOCACHE;

DECLARE
 v_number_new_classes  PLS_INTEGER :=
    NVL(:no_of_new_classes, 1);
 v_course_id           classes.course_id%TYPE := :course_id;
 v_period              classes.period%TYPE := :period;
 v_frequency           classes.frequency%TYPE := frequency;
 v_instructor          classes.instr_id%TYPE := :instr_id;
BEGIN
 FOR loop_counter IN 1..v_number_new_classes LOOP
  INSERT INTO classes (class_id, start_date,course_id,
                       period, frequency,instr_id)
  VALUES (class_id_seq.NEXTVAL,SYSDATE, v_course_id,
          v_period,v_frequency,v_instructor);

  COMMIT;
 END LOOP;

END;
```

Part 2: Teacher Tools

1. Create an anonymous block that a teacher can run to see the students in a course across all classes of that course. Save the block in a file called *course_roster.sql*. Accept the INSTR_ID and COURSE_ID. For each ENROLLMENT, display: CLASS_ID, STATUS, Student FIRST_NAME and LAST_NAME.

   Tables used: ENROLLMENTS; CLASSES; STUDENTS
   Topics Incorporated ==> SELECT with JOIN; Explicit Cursor

   Suggested Solution:

```
DECLARE
 v_instr_id     classes.instr_id%TYPE := :instr_id;
 v_course_id    classes.course_id%TYPE := :class_id;
 CURSOR stu_course_cur IS
  SELECT e.class_id, e.status, s.first_name, s.last_name
  FROM enrollments e, classes c, students s
  WHERE e.class_id = c.class_id
  AND e.stu_id = s.stu_id
  AND c.course_id = v_course_id
  AND c.instr_id = v_instr_id;

BEGIN
 DBMS_OUTPUT.PUT_LINE('Course ' || v_course_id   ||
        ' has the following students:');
 FOR stu_course_rec IN stu_course_cur LOOP
  DBMS_OUTPUT.PUT_LINE('Class: ' || stu_course_rec.class_id ||
'Student:  '|| stu_course_rec.first_name || ' '  ||
stu_course_rec.last_name ||
' with a status of: ' || stu_course_rec.status );

 END LOOP;

END;
```

2. Create an anonymous block which will convert a number grade to a letter grade. Save your work in a file called *convert_grade.sql*. Prompt for a number grade. RETURN a CHAR value. Use the following rules: A:90 or above, B: >=80 and<90 , C: >=70 and < 80, D: >=60 and < 70, F:<60.

   Tables used: None
   Topics Incorporated ==> Scalar variables...NUMBER IN; CHAR RETURNED; IF or CASE Statement (A=90 to 100, B=80-90, C=70-80, D=60-70, F=<60;

Suggested Solution 1:

```
DECLARE
 v_numeric_grade NUMBER := :Number_Grade;
 v_letter_grade CHAR(1);
BEGIN
IF v_numeric_grade >= 90.0 THEN v_letter_grade := 'A';
  ELSIF v_numeric_grade >= 80.0 THEN v_letter_grade := 'B';
  ELSIF v_numeric_grade >= 70.0 THEN v_letter_grade := 'C';
  ELSIF v_numeric_grade >= 60.0 THEN v_letter_grade := 'D';
  ELSE v_letter_grade := 'F';  END IF;
DBMS_OUTPUT.PUT_LINE(v_letter_grade);
END;
```

Suggested Solution 2:

```
DECLARE
 v_numeric_grade NUMBER := :Number_Grade;
 v_letter_grade CHAR(1);
BEGIN
v_letter_grade :=
  CASE WHEN v_numeric_grade >= 90.0 THEN 'A'
       WHEN v_numeric_grade >= 80.0 THEN 'B'
       WHEN v_numeric_grade >= 70.0 THEN 'C'
       WHEN v_numeric_grade >= 60.0 THEN 'D'
      ELSE 'F'
  END;
DBMS_OUTPUT.PUT_LINE(v_letter_grade);
END;
```

3. Create an anonymous block that will RETURN the number of students in a particular class. Save your work in a file called *student_count.sql*. Accept a CLASS_ID as a parameter.

   Tables used: ENROLLMENTS
   Topics Incorporated ==> Scalar variable...INTEGER; SELECT COUNT(*) INTO variable FROM enrollments where class_id = ...; using a User defined function in SQL.

Suggested Solution:

```
DECLARE
 v_class_id classes.class_id%TYPE := :class_id;
 v_student_count PLS_INTEGER;
BEGIN
 SELECT COUNT(*)
  INTO v_student_count
  FROM enrollments
  WHERE class_id = v_class_id;
 DBMS_OUTPUT.PUT_LINE(v_student_count);

END;
```

4. Create an anonymous block which a teacher can run to insert a new assignment (ASSESSMENT) including a DESCRIPTION. Save the program in a file called *create_assignment.sql*. Use the ASSESSMENT_ID_SEQ sequence to generate the class_assessment_id.

   Tables used: ASSESSMENTS,
   Topics Incorporated ==> SQL INSERT; Use of sequence in INSERT;

   Suggested Solution:

```
DECLARE
  v_assign_descrip assessments.description%TYPE :=
:assessment_description;
BEGIN
 INSERT INTO assessments (assessment_id, description)
  VALUES (assessment_id_seq.NEXTVAL, v_assign_descrip);
 COMMIT;

END;
```

5. Create an anonymous block that a teacher can run to insert the student's grade on a particular assignment. Save the program in a file called *enter_student_grade.sql*. Accept a NUMERIC_GRADE, CLASS_ASSESSMENT_ID, CLASS_ID and STU_ID. Use "today's" date for the DATE_TURNED_IN.

   Tables used: CLASS_ASSESSMENTS
   Topics Incorporated ==> SQL INSERT; SYSDATE.

Suggested Solution:

```
DECLARE
  v_cl_assessment_id
        class_assessments.class_assessment_id%TYPE :=
:class_assessment;
  v_numeric_grade
        class_assessments.numeric_grade%TYPE := :num_grade;
  v_class_id
        class_assessments.class_id%TYPE := :class_id;
  v_stu_id
        class_assessments.stu_id%TYPE := :std_id;
  v_assessment_id
        class_assessments.assessment_id%TYPE := :ass_id;
BEGIN
 INSERT INTO class_assessments
  (class_assessment_id, date_turned_in, numeric_grade,
   class_id, stu_id, assessment_id)
 VALUES
(v_cl_assessment_id, SYSDATE, v_numeric_grade, v_class_id,
 v_stu_id, v_assessment_id);
 COMMIT;

END;
```

Part 3: School Administrator's Tools

1. Create an anonymous block to list any enrollments which have NEITHER a FINAL_NUMERIC_GRADE or FINAL_LETTER_GRADE. Save the program in a file called *show_missing_grades.sql*. Accept a start_date and end_date to establish a date range. Display only enrollments between those 2 dates. Write your program so the start_date and end_date are optional. If both dates are not entered, display all applicable enrollments for the past year, and include a note about the date range. For each enrollment, list the CLASS_ID, STU_ID, and STATUS. Order the output by ENROLLMENT_DATE with the most recent enrollments first.

   Tables used: ENROLLMENTS
   Topics Incorporated ==> SQL with BETWEEN; Dates; NULL values in a database column, MONTHS_BETWEEN or other way to find the "past year."

   Suggested Solution:

```
DECLARE
  v_start_date DATE DEFAULT ADD_MONTHS(SYSDATE,-12);
  v_end_date   DATE DEFAULT SYSDATE;
  CURSOR no_grades_cur IS
   SELECT class_id, stu_id, status
   FROM enrollments
   WHERE final_numeric_grade IS NULL
   AND final_letter_grade IS NULL
   AND enrollment_date BETWEEN v_start_date and v_end_date
   ORDER BY enrollment_date DESC;
BEGIN
 IF v_start_date IS NULL OR v_end_date IS NULL
 THEN
  DBMS_OUTPUT.PUT_LINE
  ('You have not specified both dates. The listing will show
all enrollments for the past year.');
 END IF;
 DBMS_OUTPUT.PUT_LINE
  ('Date range: Between ' || v_start_date || ' and ' ||
v_end_date || '.');
 DBMS_OUTPUT.PUT_LINE
  ('The following enrollments have no grade.');
 FOR no_grades_rec IN no_grades_cur LOOP
  DBMS_OUTPUT.PUT_LINE ('Class ID ' || no_grades_rec.class_id
|| ' – Student ID ' || no_grades_rec.stu_id || ' with a status
of:  ' ||no_grades_rec.status);
 END LOOP;
     END;
```

2. Create an anonymous block to find the average grade for a class. Save the program in a file called *compute_average_grade* .sql. Assume the class has used numeric_grades. Accept a CLASS_ID. Return the average grade.

   Tables used: ENROLLMENTS
   Topics Incorporated ==> SQL with AVG function

   Suggested Solution:

```
DECLARE
 v_class_id enrollments.class_id%TYPE := :class_id;
 v_avg_grade enrollments.final_numeric_grade%TYPE;
BEGIN
 SELECT AVG(final_numeric_grade)
  INTO v_avg_grade
  FROM enrollments
  WHERE class_id = v_class_id;

 DBMS_OUTPUT.PUT_LINE('The average grade in class '||
v_class_id ||' is ' || v_avg_grade);

END;
```

3. Create an anonymous block to return the number of classes offered for a given course. Save the program in a file called *count_classes_per_course.sql.*

   Tables used: CLASSES
   Topics Incorporated ==> SQL SELECT with COUNT.

   Suggested Solution:

```
DECLARE
 v_course_id classes.course_id%TYPE := :course_id;
 v_num_classes PLS_INTEGER;
BEGIN

 SELECT COUNT(*)
  INTO v_num_classes
  FROM classes
  WHERE course_id = v_course_id;

 DBMS_OUTPUT.PUT_LINE('There are ' ||v_num_classes || '
students enrolled in class '|| v_course_id);

END;
```

4. Create an anonymous block to list all classes offered between a range of dates. Save your program in a file called *show_class_offerings.sql.* Accept a start date and end date. For each class found, display the CLASS_ID, START_DATE, instructor FIRST_NAME and LAST_NAME, course TITLE and SECTION_CODE.

   Tables used: CLASSES, INSTRUCTORS, COURSES,ENROLLMENTS
   Topics Incorporated ==> Using dates; SQL Join; %TYPE.

   Suggested Solution:

```
DECLARE
 v_start_date DATE := :Start_date;
 v_end_date   DATE := :End_date;
 CURSOR classes_info_cur IS
  SELECT cl.class_id, cl.start_date,
         i.first_name, i.last_name,
         cour.title, cour.section_code
   FROM classes cl, courses cour, instructors i
   WHERE  start_date BETWEEN v_start_date AND v_end_date
   AND cl.course_id = cour.course_id
   AND cl.instr_id = i.instructor_id
   ORDER BY 1,2,4;
--
BEGIN
 DBMS_OUTPUT.PUT_LINE
('Date range: Between ' || v_start_date || ' and ' ||
v_end_date || '.');
 DBMS_OUTPUT.PUT_LINE
('Classes Information.');
 FOR classes_info_rec IN classes_info_cur LOOP
  DBMS_OUTPUT.PUT_LINE (
'Class ID ' || classes_info_rec.class_id ||
' - Start Date' || classes_info_rec.start_date ||
' – Instructor ' || classes_info_rec.first_name || ' ' ||
classes_info_rec.last_name ||
' – Course Title '|| classes_info_rec.title ||
' – Section Code ' || classes_info_rec.section_code );
 END LOOP;
END;
```