**Technical requirements**

**FUNCTIONS & PROCEDURES**

-- This procedure shows the average grade for the given function among students

create or replace procedure avg\_grade

(v\_course IN number,

v\_average OUT number) as

begin

select round(avg(numeric\_grade))

into v\_average

from grade join section

on grade.section\_id = section.section\_id

join course on section.course\_no = course.course\_no

where course.course\_no = v\_course;

dbms\_output.put\_line('course number = ' || v\_course || ', average grade = ' || v\_average);

end avg\_grade;

-- test it

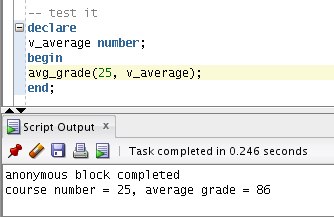
declare

v\_average number;

begin

avg\_grade(25, v\_average);

end;



-- This procedure shows the status of specified function or procedure

create or replace procedure checkStatus (

procedureName IN varchar)as

xstatus varchar2(10);

begin

select status into xstatus

from user\_objects

where object\_type in ('FUNCTION','PROCEDURE')

and object\_name = procedureName;

dbms\_output.put\_line('Procedure NAME: ' || procedureName);

if xstatus = 'VALID' then

dbms\_output.put\_line('The procedure is valid');

end if;

exception

when no\_data\_found then

dbms\_output.put\_line('The procedure is invalid');

when others then

dbms\_output.put\_line('Some error occured: ' || SQLERRM);

end checkStatus;create or replace

procedure checkStatus (

procedureName IN varchar)

as

xstatus varchar2(10);

begin

select status

into xstatus

from user\_objects

where object\_type in ('FUNCTION','PROCEDURE')

and object\_name = procedureName;

dbms\_output.put\_line('Procedure NAME: ' || procedureName);

if xstatus = 'VALID' then

dbms\_output.put\_line('The procedure is valid');

end if;

exception

when no\_data\_found then

dbms\_output.put\_line('The procedure is invalid');

when others then

dbms\_output.put\_line('Some error occured: ' || SQLERRM);

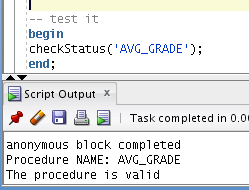
end checkStatus;

-- test it

begin

checkStatus('AVG\_GRADE');

end;



-- This function checks student GPA and if it is greater than 2

-- then a student is passed to the next year, otherwise a student has to be expelled

create or replace function CheckGPA (

xstudent IN number) RETURN VARCHAR IS

type mygrade is record(

numeric\_grade grade.numeric\_grade%type);

type xgrades is table of mygrade;

xfingrade xgrades := xgrades();

int number := 0;

xgpa number;

xfinalgpa float;

begin

for x in (select numeric\_grade

from grade

join student

on grade.student\_id=student.student\_id

join section on

grade.section\_id=section.section\_id

join course on

course.course\_no=section.course\_no

where grade.student\_id=xstudent and grade\_type\_code='FI') loop

xfingrade.extend;

int := int+1;

xfingrade(int).numeric\_grade := x.numeric\_grade;

end loop;

for x in 1 .. int loop

dbms\_output.put\_line(xfingrade(x).numeric\_grade);

end loop;

xgpa := (xfingrade(1).numeric\_grade + xfingrade(2).numeric\_grade) / 2;

dbms\_output.put\_line('Average GPA: ' || xgpa);

case

when xgpa >= 90 then

xfinalgpa := 3.67;

when xgpa >= 80 and xgpa < 90 then

xfinalgpa := 3.0;

when xgpa >= 70 and xgpa < 80 then

xfinalgpa := 2.67;

when xgpa >= 60 and xgpa < 70 then

xfinalgpa := 2.33;

when xgpa >= 50 and xgpa < 60 then

xfinalgpa := 2.0;

else xfinalgpa := 0;

end case;

dbms\_output.put\_line('Final GPA: ' || xfinalgpa);

if xfinalgpa > 2.0 then

return 'Passed to the next year';

else

return 'Expelled from the university';

end if;

exception

when no\_data\_found then

return 'No data found for that student';

end CheckGPA;

-- test it

declare

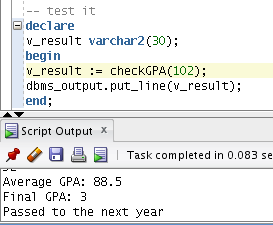
v\_result varchar2(30);

begin

v\_result := checkGPA(102);

dbms\_output.put\_line(v\_result);

end;



-- This function counts the number of students enrolled into course, if less than

-- 5 then a course gets discount of 5%

create or replace

function course\_discount

(v\_course\_no IN number)

RETURN varchar2

IS

v\_count\_students NUMBER;

begin

SELECT count(\*)

INTO v\_count\_students

FROM section

JOIN enrollment

ON section.section\_id = enrollment.section\_id

JOIN course

ON course.course\_no = section.course\_no

WHERE course.course\_no = v\_course\_no;

IF v\_count\_students < 5 THEN

UPDATE course

SET cost = (cost \* 95) / 100

WHERE course\_no = v\_course\_no;

RETURN 'This course got discount of 5%';

ELSE

RETURN 'No discount for this course';

END IF;

end course\_discount;

-- test it

declare

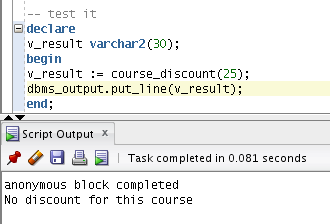
v\_result varchar2(30);

begin

v\_result := course\_discount(25);

dbms\_output.put\_line(v\_result);

end;



**COLLECTIONS (ARRAYS AND RECORDS)**

-- Showing any three sections with more than 5 students enrolled

declare

type mysection is record (

section\_id enrollment.section\_id%type,

num\_rows number);

type myarray is table of mysection index by binary\_integer;

v\_array myarray;

int number :=0;

begin

for i in ( select \* from(select section\_id, count(student\_id) my\_count

from enrollment

group by section\_id

having count(student\_id) > 5) where rownum <=3 ) loop

int := int+1;

v\_array(int).section\_id :=i.section\_id;

v\_array(int).num\_rows :=i.my\_count;

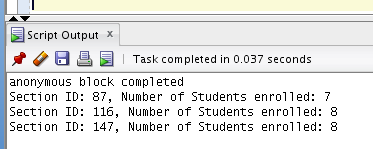
end loop;

for i in 1 .. int loop

dbms\_output.put\_line('Section ID: ' || v\_array(i).section\_id || ', Number of Students enrolled: ' || v\_array(i).num\_rows);

end loop;

end;



-- Create a phone book for instructors

declare

type InstrRec

is record (

myinstr instructor.instructor\_id%type,

myname varchar2(50),

myzip instructor.zip%type,

myphone instructor.phone%type

);

v\_instructor\_one InstrRec;

v\_instructor\_two InstrRec;

v\_instructor\_three InstrRec;

begin

select instructor\_id, first\_name || ' ' || last\_name, zip, phone

into v\_instructor\_one

from instructor

where salutation = 'Ms'

and rownum <=1;

select instructor\_id, first\_name || ' ' || last\_name, zip, phone

into v\_instructor\_two

from instructor

where salutation = 'Ms'

and rownum <=1 and instructor\_id = 105;

select instructor\_id, first\_name || ' ' || last\_name, zip, phone

into v\_instructor\_three

from instructor

where salutation = 'Ms'

and rownum <=1 and instructor\_id = 110;

dbms\_output.put\_line(v\_instructor\_one.myinstr || ' ' || v\_instructor\_one.myname ||

' ' || v\_instructor\_one.myzip || ' ' || v\_instructor\_one.myphone);

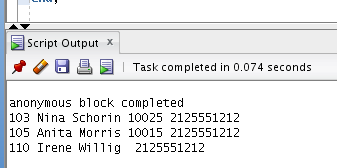
dbms\_output.put\_line(v\_instructor\_two.myinstr || ' ' || v\_instructor\_two.myname ||

' ' || v\_instructor\_two.myzip || ' ' || v\_instructor\_two.myphone);

dbms\_output.put\_line(v\_instructor\_three.myinstr || ' ' || v\_instructor\_three.myname ||

' ' || v\_instructor\_three.myzip || ' ' || v\_instructor\_three.myphone);

end;



**CURSORS**

DECLARE

CURSOR c\_group\_discount

IS

SELECT distinct s.course\_no, c.description

FROM section s, enrollment e, course c

WHERE s.section\_id = e.section\_id

AND c.course\_no = s.course\_no

GROUP BY s.course\_no, c.description, e.section\_id, s.section\_id

HAVING COUNT(\*)<=8;

BEGIN

FOR i IN c\_group\_discount

LOOP

UPDATE course

SET cost = cost\*0.95

WHERE course\_no = i.course\_no;

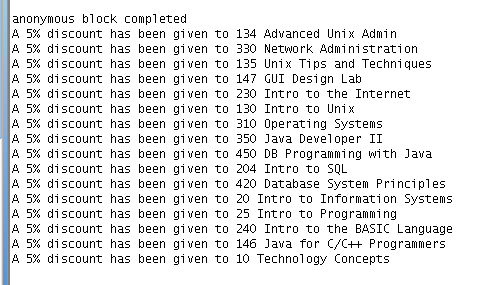
dbms\_output.put\_line('A 5% discount has been given to ' ||

i.course\_no || ' ' ||

i.description);

END LOOP;

END;



-- GET ALL MALE STUDENTS

DECLARE

CURSOR Students IS

SELECT \*

FROM student

where salutation = 'Mr.';

StudRecords student%ROWTYPE;

BEGIN

OPEN Students;

LOOP

FETCH Students INTO StudRecords;

EXIT WHEN Students%NOTFOUND;

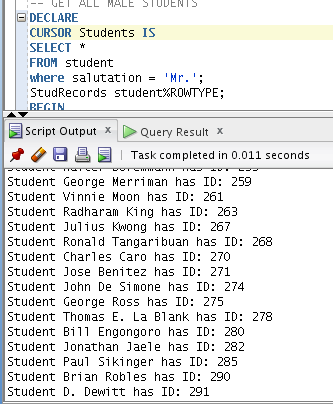
dbms\_output.put\_line('Student ' || StudRecords.first\_name ||

' ' || StudRecords.last\_name || ' has ID: ' || StudRecords.student\_id);

END LOOP;

CLOSE Students;

END;



-- TOP 3 STUDENTS OF THE SPECIFIED COURSE

DECLARE

CURSOR TopStudents (SelectCourse section.course\_no%type) IS

select section.course\_no, description, grade.numeric\_grade, grade.student\_id

from course

join section

on course.course\_no=section.course\_no

join enrollment

on section.course\_no=SelectCourse

join grade

on enrollment.student\_id=grade.student\_id

where grade\_type\_code='FI' and section.course\_no = SelectCourse

order by grade.numeric\_grade desc;

StudentSect section%rowtype;

StudentCourse course%rowtype;

StudentGrade grade%rowtype;

SelectCourse course.course\_no%type := &enter\_course\_no;

invalid\_exc exception;

BEGIN

IF NOT (TopStudents%ISOPEN) THEN

OPEN TopStudents(SelectCourse);

END IF;

IF SelectCourse <= 0

THEN RAISE invalid\_exc;

ELSE

LOOP

FETCH TopStudents INTO StudentSect.course\_no, StudentCourse.description,

StudentGrade.numeric\_grade, StudentGrade.student\_id;

EXIT WHEN TopStudents%ROWCOUNT = 4;

dbms\_output.put\_line('Course No and Name: ' || StudentSect.course\_no || ' '

|| StudentCourse.description || ' Student with ID: ' || StudentGrade.student\_id

|| ' has final grade of ' || StudentGrade.numeric\_grade);

--dbms\_output.put\_line('Student ID: ' || StudentSect.section\_id);

END LOOP;

COMMIT;

IF (TopStudents%ISOPEN) THEN

CLOSE TopStudents;

END IF;

END IF;

EXCEPTION

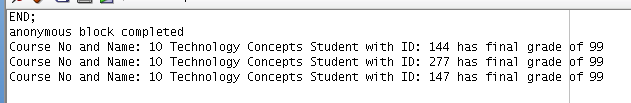
when invalid\_exc then

dbms\_output.put\_line('Invalid course number');

when value\_error then

dbms\_output.put\_line('Value error');

END;



-- CONVERTING STUDENTS NUMERIC GRADE INTO LETTER GRADE

DECLARE

stud\_id int(12):= &stud\_id;

stud\_name varchar(32);

course\_description varchar(32);

g\_numeric integer(3);

CURSOR Student IS

select student.first\_name, course.description, grade.numeric\_grade

from student

inner join grade on grade.student\_id = student.student\_id

inner join enrollment on grade.student\_id = enrollment.student\_id

inner join section on enrollment.section\_id = section.section\_id

inner join course on section.course\_no = course.course\_no

where stud\_id = student.student\_id and grade\_type\_code = 'FI'

group by student.first\_name, course.description, grade.numeric\_grade;

student\_id\_invalid EXCEPTION;

begin

if stud\_id <= 0 then

raise student\_id\_invalid;

else

open Student;

loop

fetch Student into stud\_name, course\_description, g\_numeric;

exit when Student%NOTFOUND;

dbms\_output.put\_line('Name:' || stud\_name || ', course name: '||course\_description);

case

when g\_numeric between 85 and 100 then dbms\_output.put\_line('A');

when g\_numeric between 75 and 85 then dbms\_output.put\_line('B');

when g\_numeric between 60 and 75 then dbms\_output.put\_line('C');

when g\_numeric between 50 and 60 then dbms\_output.put\_line('D');

when g\_numeric between 0 and 50 then dbms\_output.put\_line('F');

end case;

end loop;

close Student;

end if;

EXCEPTION

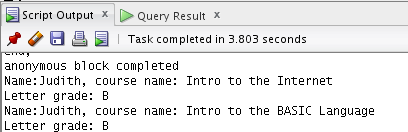
when student\_id\_invalid then

dbms\_output.put\_line('Invalid student id');

when others then

dbms\_output.put\_line('Other error!');

end;



**PACKAGES**

--GETS INSTRUCTOR DATA

create or replace PACKAGE instructor\_package AS

procedure DeleteInstructor (

xinstructor\_id IN instructor.instructor\_id%type);

procedure ListInstructors;

function totalInstructors

(xinstructor\_id IN section.instructor\_id%type,

xsection\_id IN section.section\_id%type)

RETURN VARCHAR;

END instructor\_package;

create or replace

PACKAGE BODY instructor\_package AS

procedure DeleteInstructor (

xinstructor\_id IN instructor.instructor\_id%type)

AS

invalid\_id exception;

BEGIN

DELETE FROM instructor

WHERE instructor\_id = xinstructor\_id;

IF xinstructor\_id < 0 THEN

RAISE invalid\_id;

END IF;

EXCEPTION

WHEN invalid\_id THEN

dbms\_output.put\_line('Invalid instructor ID');

END DeleteInstructor;

procedure ListInstructors AS

type myinstructors is record (

first\_name instructor.first\_name%type,

last\_name instructor.last\_name%type,

modified\_date instructor.modified\_date%type);

type allInstructors is table of myinstructors;

xinstructors allInstructors := allInstructors();

int number := 0;

BEGIN

for x in (select first\_name, last\_name, modified\_date

from instructor

order by modified\_date) loop

xinstructors.extend;

int := int+1;

xinstructors(int).first\_name := x.first\_name;

xinstructors(int).last\_name := x.last\_name;

xinstructors(int).modified\_date := x.modified\_date;

end loop;

for x in 1 .. int loop

dbms\_output.put\_line(xinstructors(x).first\_name || ' ' || xinstructors(x).last\_name || ' modified date is ' ||

xinstructors(x).modified\_date);

end loop;

END ListInstructors;

function totalInstructors

(xinstructor\_id IN section.instructor\_id%type,

xsection\_id IN section.section\_id%type)

RETURN VARCHAR

IS

xcount number;

BEGIN

select count(\*)

into xcount

from section

where section\_id = xsection\_id AND instructor\_id = xinstructor\_id;

if xcount > 0 then

return xcount;

else

return 'This department has no instructors';

end if;

end totalInstructors;

end instructor\_package;

--CHECKS THE ENROLLMENT

CREATE OR REPLACE PACKAGE MyPackage AS

FUNCTION checkEnrollment

(xstudent\_id IN enrollment.student\_id%type)

RETURN varchar;

FUNCTION checkEnrollment

(xstudent\_id IN enrollment.student\_id%type,

xsection\_id IN enrollment.section\_id%type)

RETURN varchar;

END MyPackage;

create or replace

PACKAGE BODY MyPackage AS

FUNCTION checkEnrollment

(xstudent\_id IN enrollment.student\_id%type)

RETURN VARCHAR

IS

xcount number;

BEGIN

SELECT COUNT(\*)

INTO xcount

FROM ENROLLMENT

WHERE STUDENT\_ID = xstudent\_id;

IF xcount > 0 then

RETURN 'Student is enrolled';

ELSE

RETURN 'Student is not enrolled';

END IF;

EXCEPTION

when no\_data\_found then

return 'Student with that ID is not found';

when others then

return 'Some error occured';

END checkEnrollment;

FUNCTION checkEnrollment

(xstudent\_id IN enrollment.student\_id%type,

xsection\_id IN enrollment.section\_id%type)

RETURN VARCHAR

IS

xcount number;

BEGIN

SELECT COUNT(\*)

INTO xcount

FROM enrollment

WHERE student\_id = xstudent\_id AND section\_id = xsection\_id;

IF xcount > 0 then

RETURN 'Student is enrolled into specified section';

ELSE

RETURN 'Student is not enrolled into this section';

END IF;

EXCEPTION

when no\_data\_found then

return 'Student with that ID is not found';

when others then

return 'Some error occured';

END checkEnrollment;

END MyPackage;



create package package3 as

procedure proc1 (sect\_no in out section.section\_no%type,

cor\_no in out section.course\_no%type ,

s\_id out section.section\_id%type,

c\_no out section.course\_no%type,

counter number,

output varchar2);

function over1 (st\_id in out grade.student\_id%type,

sec\_id out grade.section\_id%type,

)

return varchar2;

function over1 (grade\_code in out grade\_type.grade\_type\_code%type,

des out grade\_type.description%type)

return varchar2;

end package3;

--4th package

CREATE OR REPLACE PACKAGE StudentFind AS

PROCEDURE FIND\_STUDENT(

V\_STUDENT IN NUMBER,

V\_FNAME OUT VARCHAR2,

V\_LNAME OUT VARCHAR2);

END StudentFind;

CREATE OR REPLACE PACKAGE BODY StudentFind AS

PROCEDURE FIND\_STUDENT(

V\_STUDENT IN NUMBER,

V\_FNAME OUT VARCHAR2,

V\_LNAME OUT VARCHAR2)

AS

BEGIN

SELECT FIRST\_NAME, LAST\_NAME

INTO V\_FNAME, V\_LNAME

FROM STUDENT

WHERE STUDENT\_ID = V\_STUDENT;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

END FIND\_STUDENT;

END StudentFind;

**TRIGGERS**

-- Trigger to display cost changes

CREATE OR REPLACE TRIGGER display\_cost\_changes

BEFORE DELETE OR INSERT OR UPDATE ON course

FOR EACH ROW

WHEN (NEW.course\_no > 0)

DECLARE

cost\_diff number;

BEGIN

cost\_diff := :NEW.cost - :OLD.cost;

dbms\_output.put\_line('Old cost: ' || :OLD.cost);

dbms\_output.put\_line('New cost: ' || :NEW.cost);

dbms\_output.put\_line('Cost difference: ' || cost\_diff);

END;



-- Trigger to change the description of the course or delete it at all

CREATE OR REPLACE TRIGGER dept\_set\_null

AFTER DELETE OR UPDATE OF description ON course

FOR EACH ROW

-- Before row is deleted from dept or primary key (DEPTNO) of dept is updated,

-- set all corresponding dependent foreign key values in emp to NULL:

BEGIN

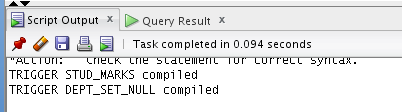
IF UPDATING AND :OLD.description != :NEW.description OR DELETING THEN

UPDATE course SET course.description = NULL

WHERE course.description = :OLD.description;

END IF;

END;



**DYNAMIC SQL**

-- first define a procedure

CREATE OR REPLACE PROCEDURE calc\_stats (

w NUMBER,x NUMBER,y NUMBER,z NUMBER)

IS

BEGIN

DBMS\_OUTPUT.PUT\_LINE(w + x + y + z);

END;

-- Test it, using dynamic SQL

DECLARE

a NUMBER := 5;

b NUMBER := 6;

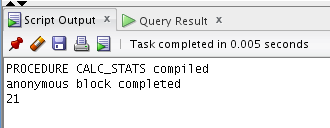
plsql\_block VARCHAR2(100);

BEGIN

plsql\_block := 'BEGIN calc\_stats(:x, :x, :y, :x); END;';

EXECUTE IMMEDIATE plsql\_block USING a, b;

END;



-- this script finds the difference between course costs

-- first define a procedure

CREATE OR REPLACE PROCEDURE cost\_diff (

x1 NUMBER,x2 NUMBER)

IS

BEGIN

DBMS\_OUTPUT.PUT\_LINE(x2-x1);

END;

-- Test it, using dynamic SQL

DECLARE

a number;

b number;

plsql\_block VARCHAR2(100);

BEGIN

select cost

into a

from course

where course\_no = 25;

select cost

into b

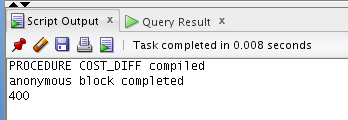
from course

where course\_no = 80;

plsql\_block := 'BEGIN cost\_diff(:x, :y); END;';

EXECUTE IMMEDIATE plsql\_block USING a, b;

END;



-- this script gets course information including ID and description

-- first define a procedure

CREATE OR REPLACE PROCEDURE course\_info (

x1 number,x2 varchar2)

IS

BEGIN

DBMS\_OUTPUT.PUT\_LINE(x1 || ' ' || x2);

END;

-- Test it, using dynamic SQL

DECLARE

a number;

b varchar2(30);

plsql\_block VARCHAR2(100);

BEGIN

select course\_no, description

into a, b

from course

where course\_no = 25;

plsql\_block := 'BEGIN course\_info(:x, :y); END;';

EXECUTE IMMEDIATE plsql\_block USING a, b;

END;

