PL/SQL Examples

Eric J. Schwabe

CSC 453 Spring 2014

(This document is for study and review purposes only. Copying of any part of the examples in this document into a submitted assignment constitutes plagiarism.)

=====

-- Hello.sql

-- Eric J. Schwabe

-- CSC 453 Spring 2014

declare

-- no variables declared...

begin

dbms\_output.put\_line( 'Hello Everyone!' );

dbms\_output.put\_line( 'This is very simple PL/SQL.' );

end;

/

=====

-- OutputScript.sql

-- Eric J. Schwabe

-- CSC 453 Spring 2014

declare

X NUMBER(3,0) := 12;

Y X%type := 5;

Z X%type;

begin

DBMS\_OUTPUT.PUT\_LINE( 'X is ' || X );

DBMS\_OUTPUT.PUT\_LINE( 'Y is ' || Y );

Z := X \* Y;

DBMS\_OUTPUT.PUT\_LINE( 'X\*Y is ' || Z );

end;

/

=====

-- Branching.sql

-- Eric J. Schwabe

-- CSC 453 Spring 2014

DECLARE

requirement NUMBER(2,0) := 18;

age requirement%type := 28;

BEGIN

DBMS\_OUTPUT.PUT( 'Age ' || age || ': ' );

IF (age >= requirement) THEN

DBMS\_OUTPUT.PUT\_LINE( 'old enough to vote.');

ELSE

DBMS\_OUTPUT.PUT\_LINE( 'not old enough to vote.');

END IF;

END;

/

=====

-- BasicLoop.sql

-- Eric J. Schwabe

-- CSC 453 Spring 2014

declare

counter INTEGER := 5;

begin

loop

dbms\_output.put\_line(counter);

if (counter = 0) then

dbms\_output.put\_line('Blast off!');

exit;

else

counter := counter - 1;

end if;

end loop;

end;

/

=====

-- WhileLoop.sql

-- Eric J. Schwabe

-- CSC 453 Spring 2014

declare

counter INTEGER := 5;

begin

while (counter >= 0) loop

dbms\_output.put\_line(counter);

counter := counter - 1;

end loop;

dbms\_output.put\_line('Blast off!');

end;

/

=====

-- ForLoop.sql

-- Eric J. Schwabe

-- CSC 453 Spring 2014

declare

-- counter is declared implicitly by for loop...

begin

for counter in reverse 0..5 loop

dbms\_output.put\_line(counter);

end loop;

dbms\_output.put\_line('Blast off!');

end;

/

=====

-- SimpleQuery.sql

-- Eric J. Schwabe

-- CSC 453 Spring 2014

-- First, build and display the STUDENT and TEST tables...

DROP TABLE STUDENT CASCADE CONSTRAINTS;

DROP TABLE TEST CASCADE CONSTRAINTS;

CREATE TABLE STUDENT

(

ID CHAR(4),

Name VARCHAR2(20),

PRIMARY KEY (ID)

);

CREATE TABLE TEST

(

ID CHAR(4),

Exam VARCHAR2(10),

Score NUMBER(3,0)

CHECK (Score >= 0 AND Score <= 100),

PRIMARY KEY (ID, Exam),

FOREIGN KEY (ID)

REFERENCES STUDENT(ID)

);

INSERT INTO STUDENT VALUES('1716', 'Eric');

INSERT INTO STUDENT VALUES('2804', 'Hank');

INSERT INTO STUDENT VALUES('1996', 'Janine');

INSERT INTO STUDENT VALUES('1130', 'Jacob');

INSERT INTO TEST VALUES('1716', 'Midterm', 100);

INSERT INTO TEST VALUES('1996', 'Final', 90);

INSERT INTO TEST VALUES('2804', 'Midterm', 98);

INSERT INTO TEST VALUES('1716', 'Final', 99);

SELECT \* FROM STUDENT;

SELECT \* FROM TEST;

-- This anonymous PL/SQL block defines two variables to hold

-- the values of a record returned by a query, executes the

-- query and stored the results in the variables, and then

-- displays them...

declare

studentName STUDENT.Name%type;

studentID STUDENT.ID%type;

targetID STUDENT.ID%type := '1716';

begin

DBMS\_OUTPUT.PUT\_LINE( 'Querying STUDENT table for ID ' || targetID || ':');

-- The next statement stores the result of the query in the two variables

SELECT Name, ID

INTO studentName, studentID

FROM STUDENT

WHERE ID = targetID;

DBMS\_OUTPUT.PUT\_LINE( 'Query done!' );

DBMS\_OUTPUT.PUT\_LINE('For ID ' || studentID || ' , Name is ' || studentName);

end;

/

=====

-- FancyQuery.sql

-- Eric J. Schwabe

-- CSC 453 Spring 2014

-- First, build and display the STUDENT and TEST tables...

[...redundant table creation removed...]

-- This anonymous PL/SQL block defines two variables to hold

-- the values of a record returned by a query and a cursor to

-- traverse a set of records, executes a query, and then uses

-- the cursor to retrieve the records one at a time and display them...

declare

studentName STUDENT.Name%type;

studentID STUDENT.ID%type;

-- Sets up the cursor

cursor studentPtr is SELECT \* FROM STUDENT;

begin

dbms\_output.put\_line( 'Traversing STUDENT table with a cursor:' );

-- Opens the cursor

open studentPtr;

-- Loop fetches one record at a time until no more records are found

loop

fetch studentPtr into studentID, studentName;

if studentPtr%found then

dbms\_output.put\_line( studentPtr%rowcount ||

' : For ID ' || studentID || ' , Name is ' || studentName);

else

exit;

end if;

end loop;

-- Closes the cursor

close studentPtr;

dbms\_output.put\_line( 'Done!' );

end;

/

=====

--- Projects.sql

--- Eric J. Schwabe

--- CSC 453 Spring 2014

-- get rid of any existing tables

DROP TABLE ASSIGNMENT CASCADE CONSTRAINTS;

DROP TABLE PROJECT CASCADE CONSTRAINTS;

-- create the new tables, with primary and foreign keys

CREATE TABLE PROJECT

(

Code NUMBER(3),

Name VARCHAR2(30),

CONSTRAINT PK\_PROJECT

PRIMARY KEY (Code)

);

CREATE TABLE ASSIGNMENT

(

EmpID CHAR(5),

EmpLName VARCHAR2(30),

EmpFName VARCHAR2(20),

Dept VARCHAR2(20),

ProjCode NUMBER(3),

Hours NUMBER(\*,0),

CONSTRAINT PK\_ASSIGNMENT

PRIMARY KEY (EmpID, ProjCode),

CONSTRAINT FK\_ASSIGNMENT\_PROJECT

FOREIGN KEY (ProjCode)

REFERENCES PROJECT (Code)

);

-- populate tables (PROJECT first, then ASSIGNMENT)

INSERT INTO PROJECT

VALUES ( 101, 'Four-Dimensional Phones' );

INSERT INTO PROJECT

VALUES ( 252, 'Matter Transporters' );

INSERT INTO PROJECT

VALUES ( 995, 'Anti-Gravity Clothing' );

INSERT INTO ASSIGNMENT

VALUES ( '550', 'Smith', 'Winston', 'Accounting', 101, 20 );

INSERT INTO ASSIGNMENT

VALUES ( '550', 'Smith', 'Winston', 'Accounting', 252, 10 );

INSERT INTO ASSIGNMENT

VALUES ( '601', 'Smith', 'Barney', 'Finance', 252, 5 );

INSERT INTO ASSIGNMENT

VALUES ( '390', 'Hammond', 'Evey', 'Personnel', 995, 25 );

INSERT INTO ASSIGNMENT

VALUES ( '001', 'Preston', 'Bill', 'Special Events', 995, 5 );

INSERT INTO ASSIGNMENT

VALUES ( '100', 'Logan', 'Ted', 'Special Events', 995, 5 );

INSERT INTO ASSIGNMENT

VALUES ( '007', 'Bond', 'James', 'Personnel', 252, 20 );

INSERT INTO ASSIGNMENT

VALUES ( '505', 'Lane', 'Lois', 'Media Relations', 101, 10 );

INSERT INTO ASSIGNMENT

VALUES ( '505', 'Lane', 'Lois', 'Media Relations', 252, 10 );

INSERT INTO ASSIGNMENT

VALUES ( '505', 'Lane', 'Lois', 'Media Relations', 995, 10 );

-- display the tables' contents

SELECT \* FROM PROJECT;

SELECT \* FROM ASSIGNMENT;

-- anonymous block to report project codes and names

declare

projectCode NUMBER(3,0);

projectName VARCHAR(30);

cursor projectPtr is SELECT \* FROM PROJECT;

begin

dbms\_output.put\_line('Projects:');

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

open projectPtr;

loop

fetch projectPtr into projectCode, projectName;

if (projectPtr%found) then

dbms\_output.put\_line( projectCode || ': ' || projectName );

else

exit;

end if;

end loop;

close projectPtr;

end;

/

-- anonymous block to report project codes and names,

-- and list the IDs and names of employees on each project

/\*

declare

projectCode NUMBER(3,0);

projectName VARCHAR(30);

cursor projectPtr is SELECT \* FROM PROJECT;

id CHAR(5);

firstname VARCHAR2(20);

lastname VARCHAR2(30);

-- this is a parameterized cursor, where we must

-- supply the missing parameter when it is opened

cursor personPtr (codeIn in PROJECT.Code%type) is

SELECT EmpId, EmpFName, EmpLName FROM ASSIGNMENT

WHERE ProjCode = codeIn

ORDER BY EmpId;

begin

dbms\_output.put\_line('Projects and Personnel:');

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

open projectPtr;

loop

-- retrieve one project

fetch projectPtr into projectCode, projectName;

if (projectPtr%found) then

dbms\_output.put\_line( projectCode || ': ' || projectName );

-- for the current project, retrive all employees

-- working on that project

open personPtr(projectCode);

loop

fetch personPtr into id, firstname, lastname;

if (personPtr%found) then

dbms\_output.put\_line( ' ' || id || firstname ||

' ' || lastname );

else

exit;

end if;

end loop;

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

close personPtr;

else

exit;

end if;

end loop;

close projectPtr;

end;

/

\*/

=====

-- Records.sql

-- Eric J. Schwabe

-- CSC 453 Spring 2014

-- First, build and display the STUDENT and TEST tables...

[...redundant table creation removed...]

-- This anonymous PL/SQL block defines a record to hold

-- the result returned by a query, executes the query

-- and stores the results in the record, and then

-- displays its fields...

declare

studentInfo STUDENT%rowtype;

targetID STUDENT.ID%type := '1716';

begin

DBMS\_OUTPUT.PUT\_LINE( 'Querying STUDENT table for ID ' || targetID || ':' );

-- The next statement stores the result of the query in the record

SELECT Name, ID

INTO studentInfo

FROM STUDENT

WHERE ID = targetID;

DBMS\_OUTPUT.PUT\_LINE( 'Query done!' );

DBMS\_OUTPUT.PUT\_LINE( 'For ID ' || studentInfo.ID || ' , Name is ' ||

studentInfo.Name );

end;

/

=====

-- Records2.sql

-- Eric J. Schwabe

-- CSC 453 Spring 2014

-- This anonymous PL/SQL block defines a record to hold

-- the result returned by a query, executes the query

-- and uses a cursor to traverse the records in the result...

-- Note the use of a cursor for loop, where open, fetch, and close are implicit...

declare

studentInfo STUDENT%rowtype;

cursor studentPtr is SELECT \* FROM STUDENT;

begin

DBMS\_OUTPUT.PUT\_LINE( 'Querying STUDENT table' );

for studentInfo in studentPtr

loop

DBMS\_OUTPUT.PUT\_LINE( studentInfo.ID || ' ' || studentInfo.Name );

end loop;

DBMS\_OUTPUT.PUT\_LINE( 'Done!' );

end;

/

=====

-- Table.sql

-- Eric J. Schwabe

-- CSC 453 Spring 2014

-- This anonymous PL/SQL block defines a record to hold

-- the result returned by a query, executes the query

-- and uses a cursor to traverse the records in the result...

-- As the records are retrieved, they are placed into a table…

declare

studentInfo STUDENT%rowtype;

cursor studentPtr is SELECT \* FROM STUDENT;

-- declare type of table

type studentTable is table of STUDENT%rowtype

index by binary\_integer;

-- declare actual table of the new type

students studentTable;

-- declare index variable

i binary\_integer := 1;

begin

DBMS\_OUTPUT.PUT\_LINE( 'Querying STUDENT table' );

for studentInfo in studentPtr

loop

DBMS\_OUTPUT.PUT\_LINE( studentInfo.ID || ' ' || studentInfo.Name );

DBMS\_OUTPUT.PUT\_LINE( ' [Adding record to table in position ' || i ||

' ...]' );

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

students(i) := studentInfo;

-- can vary spacing to illustrate sparsity of table...

i := i + 1;

end loop;

DBMS\_OUTPUT.PUT\_LINE( 'Done!' );

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

-- now looking at table limits and one element...

DBMS\_OUTPUT.PUT\_LINE('First element is in position: ' || students.first);

DBMS\_OUTPUT.PUT\_LINE('Last element is in position : ' || students.last);

DBMS\_OUTPUT.PUT('Element 3 is : ');

DBMS\_OUTPUT.PUT\_LINE(students(3).ID || ' ' || students(3).Name);

/\*

-- have to do this to avoid "no data found" error if there is no element 3

if (students.exists(3)) then

DBMS\_OUTPUT.PUT\_LINE(students(3).ID || ' ' || students(3).Name);

else

DBMS\_OUTPUT.PUT\_LINE('Not there!');

end if;

\*/

end;

/

=====

-- Procedure.sql

-- Eric J. Schwabe

-- CSC 453 Spring 2014

-- Creating a procedure

CREATE OR REPLACE PROCEDURE

raisetopower (base IN INTEGER, exponent IN INTEGER, result OUT INTEGER)

AS

x INTEGER:= 1;

BEGIN

FOR i IN 1..exponent LOOP

x := x \* base;

END LOOP;

result := x;

END;

/

-- Calling the procedure from an anonymous block

DECLARE

a INTEGER := 2;

b INTEGER := 10;

c INTEGER;

BEGIN

DBMS\_OUTPUT.PUT\_LINE( 'a is ' || a );

DBMS\_OUTPUT.PUT\_LINE( 'b is ' || b );

DBMS\_OUTPUT.PUT\_LINE( 'Calling procedure raisetopower...' );

raisetopower(a, b, c);

DBMS\_OUTPUT.PUT\_LINE( 'c is ' || c );

END;

/

=====

--- Function.sql

--- Eric J. Schwabe

--- CSC 453 Spring 2014

-- Creating a function

CREATE OR REPLACE FUNCTION

raisetopower2 (base IN INTEGER, exponent IN INTEGER) RETURN INTEGER

AS

x INTEGER:= 1;

BEGIN

FOR i IN 1..exponent LOOP

x := x \* base;

END LOOP;

return x;

END;

/

-- Calling the function from an anonymous block

DECLARE

a INTEGER := 10;

b INTEGER := 2;

BEGIN

DBMS\_OUTPUT.PUT\_LINE( 'a is ' || a );

DBMS\_OUTPUT.PUT\_LINE( 'b is ' || b );

DBMS\_OUTPUT.PUT\_LINE( 'Calling function raisetopower2...' );

DBMS\_OUTPUT.PUT\_LINE( 'result is ' || raisetopower2(a, b) );

END;

/

=====

=====

-- Workers.sql

-- Eric J. Schwabe

-- CSC 453 Spring 2014

-- First, set up table of workers' names (just letters),

-- department numbers (just digits), and salaries, and

-- display both the table and the total of the salaries

DROP TABLE WORKERS CASCADE CONSTRAINTS;

CREATE TABLE WORKERS

(

Name CHAR(1),

DeptNumber NUMBER(1,0),

Salary NUMBER(8,2),

CONSTRAINT PK\_WORKERS

PRIMARY KEY (Name)

);

INSERT INTO WORKERS VALUES('A', 5, 30000);

INSERT INTO WORKERS VALUES('B', 2, 45000);

INSERT INTO WORKERS VALUES('C', 5, 70000);

INSERT INTO WORKERS VALUES('D', 2, 55000);

INSERT INTO WORKERS VALUES('E', 1, 25000);

INSERT INTO WORKERS VALUES('F', 3, 90000);

INSERT INTO WORKERS VALUES('G', 4, 90000);

INSERT INTO WORKERS VALUES('H', 2, 39000);

INSERT INTO WORKERS VALUES('I', 1, 36000);

INSERT INTO WORKERS VALUES('J', 3, 60000);

INSERT INTO WORKERS VALUES('K', 5, 76000);

INSERT INTO WORKERS VALUES('L', 3, 40000);

INSERT INTO WORKERS VALUES('M', 4, 85000);

INSERT INTO WORKERS VALUES('N', 1, 39000);

INSERT INTO WORKERS VALUES('O', 2, 42000);

SELECT \* FROM WORKERS;

SELECT SUM(Salary) FROM WORKERS;

-- This trigger tests if inserting another employee

-- will yield a total of more than one million dollars

-- in salaries, and if it will, it raises an error

-- message and cancels the insert

CREATE OR REPLACE TRIGGER SalaryCap AFTER INSERT OR UPDATE ON WORKERS

DECLARE

total INTEGER;

BEGIN

SELECT SUM(WORKERS.Salary)

INTO total

FROM WORKERS;

DBMS\_OUTPUT.PUT\_LINE(total);

IF (total >= 1000000) THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Million Dollar Limit Exceeded');

END IF;

END;

/

=====

=====

-- Salaries.sql

-- Eric J. Schwabe

-- CSC 453 Spring 2014

-- Trigger example from E&N Sec 26.1

-- First, build the EMPLOYEE and DEPARTMENT tables

DROP TABLE EMPLOYEE CASCADE CONSTRAINTS;

DROP TABLE DEPARTMENT CASCADE CONSTRAINTS;

CREATE TABLE DEPARTMENT

(

DName VARCHAR2(20),

Dno NUMBER(1, 0),

TotalSalary NUMBER(9, 2),

PRIMARY KEY (Dno)

);

CREATE TABLE EMPLOYEE

(

Name VARCHAR2(20),

SSN CHAR(9),

Salary NUMBER(9,2),

Dno NUMBER(1, 0),

PRIMARY KEY (SSN),

FOREIGN KEY (DNo)

REFERENCES DEPARTMENT(Dno)

);

-- This trigger updates the total salary for a department

-- when an employee is added to that department

CREATE OR REPLACE TRIGGER TS1

AFTER INSERT ON EMPLOYEE

FOR EACH ROW

WHEN (new.Dno IS NOT NULL)

BEGIN

DBMS\_OUTPUT.PUT\_LINE('TS1 -- new employee');

UPDATE DEPARTMENT

SET TotalSalary = TotalSalary + :new.Salary

WHERE Dno = :new.Dno;

END;

/

-- This trigger updates the total salary for a department

-- when an employee in that department has their salary changed

CREATE OR REPLACE TRIGGER TS2

AFTER UPDATE OF Salary ON EMPLOYEE

FOR EACH ROW

WHEN (new.Dno IS NOT NULL)

BEGIN

DBMS\_OUTPUT.PUT\_LINE('TS2 -- change to salary');

UPDATE DEPARTMENT

SET TotalSalary = TotalSalary + :new.Salary - :old.Salary

WHERE Dno = :new.Dno;

END;

/

-- This trigger updates the total salary for two departments

-- when a employee is moved from one department to another

CREATE OR REPLACE TRIGGER TS3

AFTER UPDATE OF Dno ON EMPLOYEE

FOR EACH ROW

BEGIN

DBMS\_OUTPUT.PUT\_LINE('TS3 -- change to department');

UPDATE DEPARTMENT

SET TotalSalary = TotalSalary + :new.Salary

WHERE Dno = :new.Dno;

UPDATE DEPARTMENT

SET TotalSalary = TotalSalary - :old.Salary

WHERE Dno = :old.Dno;

END;

/

-- This trigger updates the total salary for a department

-- when a employee is deleted

CREATE OR REPLACE TRIGGER TS4

AFTER DELETE ON EMPLOYEE

FOR EACH ROW

WHEN (old.Dno IS NOT NULL)

BEGIN

DBMS\_OUTPUT.PUT\_LINE('TS4 -- employee leaving');

UPDATE DEPARTMENT

SET TotalSalary = TotalSalary - :old.Salary

WHERE Dno = :old.Dno;

END;

/

-- Now, with the triggers in place, populate the tables

INSERT INTO DEPARTMENT VALUES ('Adminstration', 1, 0);

INSERT INTO DEPARTMENT VALUES ('Computer Science', 2, 0);

INSERT INTO EMPLOYEE VALUES ('Epp', '111111111', 250000, 1);

INSERT INTO EMPLOYEE VALUES ('Miller', '222222222', 200000, 2);

INSERT INTO EMPLOYEE VALUES ('Schwabe', '333333333', 80000, 2);

SELECT \* FROM EMPLOYEE;

SELECT \* FROM DEPARTMENT;

=====

=====

/\*

Grades.sql

Eric J. Schwabe

CSC 453 Spring 2014

\*/

DROP TABLE GRADE CASCADE CONSTRAINTS;

DROP TABLE WEIGHT CASCADE CONSTRAINTS;

DROP TABLE TOTAL CASCADE CONSTRAINTS;

CREATE TABLE WEIGHT

(

Grade VARCHAR(2),

Points NUMBER(2,1),

PRIMARY KEY (Grade)

);

CREATE TABLE TOTAL

(

Name VARCHAR2(20),

GPA NUMBER(3,2),

PRIMARY KEY (Name)

);

CREATE TABLE GRADE

(

Name VARCHAR2(20),

Grade VARCHAR2(2),

CONSTRAINT FK\_GRADE\_WEIGHT

FOREIGN KEY (Grade)

REFERENCES WEIGHT(Grade)

);

INSERT INTO WEIGHT VALUES ( 'A', 4.0 );

INSERT INTO WEIGHT VALUES ( 'A-', 3.7 );

INSERT INTO WEIGHT VALUES ( 'B+', 3.3 );

INSERT INTO WEIGHT VALUES ( 'B', 3.0 );

INSERT INTO WEIGHT VALUES ( 'B-', 2.7 );

INSERT INTO WEIGHT VALUES ( 'C+', 2.3 );

INSERT INTO WEIGHT VALUES ( 'C', 2.0 );

INSERT INTO WEIGHT VALUES ( 'C-', 1.7 );

INSERT INTO WEIGHT VALUES ( 'D+', 1.3 );

INSERT INTO WEIGHT VALUES ( 'D', 1.0 );

INSERT INTO WEIGHT VALUES ( 'F', 0.0 );

SELECT \* FROM WEIGHT;

/\*

-- trigger to re-compute the GPA table any time the grade table is changed...

CREATE OR REPLACE TRIGGER UpdateTotals

AFTER INSERT OR UPDATE OR DELETE ON GRADE

DECLARE

CURSOR GPAPtr IS

SELECT Name, AVG(Points)

FROM GRADE INNER JOIN WEIGHT

ON GRADE.Grade = WEIGHT.Grade

GROUP BY Name

ORDER BY Name;

CurrName VARCHAR2(20);

CurrGPA NUMBER(3,2);

BEGIN

dbms\_output.put\_line('Trigger UpdateTotals firing...');

DELETE FROM TOTAL;

open GPAPtr;

loop

fetch GPAPtr into CurrName, CurrGPA;

if (GPAPtr%found) then

dbms\_output.put\_line(CurrName || ' : ' || CurrGPA);

INSERT INTO TOTAL VALUES (CurrName, CurrGPA);

else

exit;

end if;

end loop;

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

close GPAPtr;

END;

/

\*/

INSERT INTO GRADE VALUES ( 'Bob', 'C' );

INSERT INTO GRADE VALUES ( 'Bob', 'B-' );

INSERT INTO GRADE VALUES ( 'Alice', 'B' );

INSERT INTO GRADE VALUES ( 'Alice', 'A' );

INSERT INTO GRADE VALUES ( 'Charlie', 'A-' );

INSERT INTO GRADE VALUES ( 'Alice', 'A' );

INSERT INTO GRADE VALUES ( 'Charlie', 'B+' );

SELECT \* FROM GRADE;

SELECT \* FROM TOTAL;

SELECT Name, AVG(Points)

FROM GRADE INNER JOIN WEIGHT

ON GRADE.Grade = WEIGHT.Grade

GROUP BY Name

ORDER BY Name;

=====