CS 328 - Homework 5

Deadline

11:59 pm on **Friday, February 27, 2026**

Purpose

To practice more with PL/SQL procedures and functions, including more practice with parameters, cursor-controlled `for` loops, and (possibly) exception-handling.

How to submit

You complete **Problems 1 and 2** on the course Canvas site (more short-answer questions on PL/SQL), so that you can see if you are on the right track.

Each time you wish to submit files for **Problems 3 onward**, submit your files using `~st10/328submit` on nrs-projects, with a homework number of **5**.

Problem 1 - on Canvas - 5 points

Purpose: to review PL/SQL syntax for parameters, functions, and if statements

Task: Answer the questions "HW 5 - Problem 1 - PL/SQL parameters, functions, and if statements" on the course Canvas site. You may make as many attempts as you would like, and your highest score is the one kept.

Criteria: You will receive full credit for giving the correct answer for each question.

Problem 2 - on Canvas - 5 points

Purpose: to review PL/SQL syntax for loops and exception-handling

Task: Answer the questions "HW 5 - Problem 2 - PL/SQL loops and exception-handling" on the course Canvas site. You may make as many attempts as you would like, and your highest score is the one kept.

Criteria: You will receive full credit for giving the correct answer for each question.

Requirements/Set-up for Problems 3 onward

Create a file `328hw5.sql`. Give this file permissions of `600` (`rw-------`) by typing this at the nrs-projects prompt:

```
chmod 600 328hw5.sql
```

Start this file with the following:

- comments containing at least your name, `CS 328 - Homework 5`, and the last-modified date.
- include the command to `set serveroutput on`

- followed by a SQL*Plus **`spool`** command to spool the results of running this SQL script to a file named **`328hw5-out.txt`**
- followed by a **`prompt`** command including your name

Be sure to **`spool off`** at the **end** of this script (after your statements for the remaining problems).

ASIDE: string-related Oracle functions WORK in PL/SQL!

Recall that you discussed several lovely and useful string- and date- and time-related Oracle functions in CS 325, such as **`upper`**, which expects a character string argument and returns an all-uppercase version of that argument. You can find a little handout describing some of these posted with the selection of PL/SQL and SQL references at:

https://nrs-projects.humboldt.edu/~st10/s25cs328/plsql-refs.php

Look for the link "Some string- and date- and time-related SQL functions".

Happily, these functions also can be used in PL/SQL **outside** of **`select`** statements!

For example, if the following were included in a PL/SQL procedure:

```
    dbms_output.put_line(lpad('Moo', 12, '.'));
```

...and `set serveroutput on` has been done, the following would be displayed to the screen:

```
.........Moo
```

Problem 3 - PL/SQL stored procedure `silly_shout`

Purpose: to give you practice writing a PL/SQL **procedure** that includes parameters, an **`IF`** statement, and a loop

Task: In your PL/SQL script **`328hw5.sql`**, write a PL/SQL stored procedure **`silly_shout`** that:

- expects **two** parameters, a desired message and how many times it is to be "shouted" to the screen
- prints one of the following to the screen:
  - If the number of times to shout given is less than 0, the procedure should print a message to the screen saying that it cannot show the desired message that many times (and include both the desired message and the "bad" number of times in that message)
  - Otherwise, it should print an **all-uppercase** version of the given message to the screen that many times, once per line, each time concatenating **TWO** exclamation point characters (**`'!!'`**) to the end (get it? so it is "shouting" that message to the screen? 8-))
- returns nothing (since it is a procedure!)

For example,

`exec silly_shout('howdy', 3)`

...should print to the screen:

```
HOWDY!!
HOWDY!!
```

```
HOWDY!!
```

Here are additional requirements for this problem:

- Create an opening comment block for your procedure that has a **`procedure:`** part and **`purpose:`** part in the same style as used in posted class examples. (You don't have to give an `examples:` part, but you can if you wish.)
  - Follow that with the PL/SQL code creating your procedure.
- Remember to follow your PL/SQL procedure with:

  ```
  /
  show errors
  ```
- Then put a comment saying you are about to **test** your procedure **`silly_shout`**.
- Follow that with at least **FOUR** tests of **`silly_shout`**:
  - at least **two** with ***different*** messages and ***different*** numbers of shouts that are each **greater than 1**
  - at least one with a number of shouts of **0**
  - at least one with a number of shouts that is **less than 0**

  ...**EACH** test including:
  - **`prompt`** command(s) stating that you are about to test **`silly_shout`** and **describing** what you should see if it is working properly.
    - (Your description should be specific enough that someone looking just at the spooled output can tell if the test passed or not.)
  - Then write a SQL*Plus command calling **`silly_shout`**.

Problem 4 - PL/SQL stored function `title_total_cost`

Purpose: to get more practice writing PL/SQL stored functions

Task: In your SQL script **`328hw5.sql`**, write a PL/SQL stored function **`title_total_cost`** that meets the following requirements:

- It expects a title's ISBN.
- It returns the total **COST** (**not** price!) of **all** of the current quantity for that title.
  - For example: **`title_total_cost('9780871507877')= 1137.5`**
  - (If there is **no** title with that ISBN, it should return **-1**, so the caller can know that there is no title with this ISBN in the **`title`** table.)
- Note that there is more than one reasonable way to implement this function; at least one of these approaches involves exception handling, and at least one of these approaches does not.
- Create an opening comment block for your function that has a **`function:`** part and **`purpose:`** part in the same style as used in posted class examples. (You don't have to give an `examples:` part, but you can if you wish.)

- Follow that with the PL/SQL code creating your function.

- Remember to follow your PL/SQL function with:

  `/`

  `show errors`

- Then put a comment saying you are about to **test** your function **`title_total_cost`**.
- Follow that with at least **THREE** tests of **`title_total_cost`**:
  - at least **two** with ***different*** ISBNs that exist in your **`title`** table
  - at least one with an ISBN that does **NOT exist** in your **`title`** table

  ...**EACH** test including:
  - **`prompt`** command(s) stating that you are about to test **`title_total_cost`** and **describe** what you should see if it is working properly.
    - (Your description should be specific enough that someone looking just at the spooled output can tell if the test passed or not.)
  - Remember that:
    - You will need to declare a SQL*Plus local variable to hold the result returned by your function.
    - The **`exec`** command is a little **different** when calling a function than when calling a procedure.
    - You can use the **`print`** command to display the value of a SQL*Plus local variable.

ALTERNATIVE option for TESTING `title_total_cost`:

- You may use Homework 4 - Problem 7's PL/SQL stored procedure **`print_test`** for your tests,

  first including a **`prompt`** command to state that you are testing **`title_total_cost`**, and then, for each of your tests:
  - using a first argument to **`print_test`** that is a **string** containing a boolean expression containing an example call to **`title_total_cost`** and what it should return,
  - and using a second argument to **`print_test`** that is that boolean expression.
- For example,

  (recalling that you print a single quote in SQL by putting two single quotes,

  and that when you want to **extend** a SQL*Plus command to a **next** line, you must end the **first** line with a **DASH**):

```
exec print_test('title_total_cost(''9780871507877'') = 1137.5',-
                title_total_cost('9780871507877') = 1137.5)
```

Problem 5 - PL/SQL stored procedure `titles_in_range`

Purpose: to get more practice writing a PL/SQL cursor-controlled **`for`** loop

Task: In your PL/SQL script **`328hw5.sql`**, write a PL/SQL stored procedure **`titles_in_range`** that meets the following requirements:

- It expects **two** parameters, the desired ***low*** end of a **price** range and the desired ***high*** end of a **price** range.
- It prints to the screen, for all titles whose title **price** (not cost!) are greater than or equal to the desired low end given, **and** less than or equal to the desired high end given:
  - a `'$'`
  - then the title's price,
  - then a blank, a dash, and a blank,
  - then the title's name,
  - then a colon and a blank,
  - and then the title's quantity,
  - ...in **first** order of the title price (lowest to highest) and in **secondary** order of title name.
  - (**NOTE**: For the case where there are **no** titles with prices in the given range, it is fine if your procedure simply prints nothing.)
- It returns nothing (since it is a procedure!).
- It appropriately uses a **cursor-controlled `for` loop**.
  - (Note that there are examples of these in `328lect05-2.sql`'s procedure `loopy`, its third and fourth loops.)

For example,

`exec titles_in_range(30, 40)`

...should print to the screen:

```
$31.5 - Financial Accounting: 10
$34.95 - Computers and Data Processing: 15
$35.95 - Operating Systems: A Systematic View: 5
$37.95 - An Introduction to Database Systems: 10
$37.95 - Data Base Management: 20
$37.95 - Problem Solving and Structured Programming: 12
$39.95 - The C Programming Language: 10
$40 - Software Engineering: 10
```

Here are additional requirements for this problem:

- Create an opening comment block for your procedure that has a **`procedure:`** part and **`purpose:`** part in the same style as used in posted class examples. (You don't have to give an `examples:` part, but you can if you wish.)

  - Follow that with the PL/SQL code creating your procedure.
- Remember to follow your PL/SQL procedure with:

  `/`

  `show errors`
- Then put a comment saying you are about to **test** your procedure **`titles_in_range`**.
- Follow that with at least **THREE** tests of **`titles_in_range`**:
  - at least **two** with ***different*** low-end prices and ***different*** high-end prices that **do** have some existing titles between them
  - at least **one** with a low-end price and a high-end price that do **NOT** have any existing titles between them

  ...**EACH** test including:
  - **`prompt`** command(s) stating that you are about to test **`titles_in_range`** and **describing** what you should see if it is working properly.
    - (Your description should be specific enough that someone looking just at the spooled output can tell if the test passed or not.)
  - Then write a SQL*Plus command calling **`titles_in_range`**.

Problem 6 - PL/SQL stored function `get_pub`

Purpose: to get still-more practice writing PL/SQL stored functions

Task: In your SQL script **`328hw5.sql`**, write a PL/SQL stored function **`get_pub`** that meets the following requirements:

- It expects a title's ISBN.
- It returns the **name** of the publisher (*not* the publisher ID) for the title with that ISBN.
  - For example: **`get_pub('9780262534802') = 'The MIT Press'`**
  - (If there is **no** title with that ISBN, it should return **`'ISBN not found'`**, so the caller can know there is no title with this ISBN in the **`title`** table.)
- Note that there is more than one reasonable way to implement this function; at least one of these approaches involves exception handling, and at least one of these approaches does not.
- Create an opening comment block for your function that has a **`function:`** part and **`purpose:`** part in the same style as used in posted class examples. (You don't have to give an `examples:` part, but you can if you wish.)
  - Follow that with the PL/SQL code creating your function.
- Remember to follow your PL/SQL function with:

  `/`

  `show errors`

- Then put a comment saying you are about to **test** your function **`get_pub`**.
- Follow that with at least **THREE** tests of **`get_pub`**:
  - at least **two** with *different* ISBNs (that have *different* publishers) that exist in your **`title`** table
  - at least one with an ISBN that does **NOT exist** in your **`title`** table

  ...**EACH** test including:
  - **`prompt`** command(s) stating that you are about to test **`get_pub`** and **describing** what you should see if it is working properly.
    - (Your description should be specific enough that someone looking just at the spooled output can tell if the test passed or not.)
  - Remember that:
    - You will need to declare a SQL*Plus local variable to hold the result returned by your function.
    - The **`exec`** command is a little **different** when calling a function than when calling a procedure.
    - You can use the **`print`** command to display the value of a SQL*Plus local variable.

ALTERNATIVE option for TESTING `get_pub`:

- You may use Homework 4 - Problem 7's PL/SQL stored procedure **`print_test`** for your tests,

  first including a **`prompt`** command to state that you are testing **`get_pub`**, and then, for each of your tests:
  - using a first argument to **`print_test`** that is a **string** containing a boolean expression containing an example call to **`get_pub`** and what it should return,
  - and using a second argument to **`print_test`** that is that boolean expression.
- For example,

  (recalling that you print a single quote in SQL by putting two single quotes,

  and that when you want to **extend** a SQL*Plus command to a **next** line, you must end the **first** line with a **DASH**):

```
exec print_test('get_pub(''9780262534802'') = ''The MIT Press''',-
                get_pub('9780262534802') = 'The MIT Press')
```

Make sure you turned spooling off at the end of **`328hw5.sql`**, and, each time you submit your work thus far, submit your latest **`328hw5.sql`** and its corresponding **`328hw5-out.txt`**.