Deadline:
Due by 11:59 pm on Wednesday, March 13, 2013

How to submit:
Submit your files for this homework using ~st10/318submit on nrs-projects, with a homework number of 5

Purpose:
To practice writing a PL/SQL trigger, and to practice a little with Java.

Important notes:
• You are expected to use standard Java naming conventions (as discussed in class) in your Java source code.
• In your Java source code, you are expected to indent the contents of all { }'s by at least 3 spaces, and each { and } should be on its own line, even with the preceding line (as seen in posted class examples).
  – also, all Java classes and methods are expected to start with a comment that at least gives its name, and a purpose statement which explicitly describes either the purpose of the class or what the method expects and what it does and/or returns.
• Remember to also follow the style guidelines and course standards given or discussed previously for the other languages used in this homework.
• Make sure that you have executed the scripts create-bks.sql and pop-bks.sql, and that the bookstore tables are successfully created and populated.

The Problems:

Problem 1
Consider PL/SQL triggers, another kind of stored PL/SQL subroutine.
You can find some examples of triggers (that are very over-commented!) at:
http://users.humboldt.edu/smtuttle/f10cis315/315ex-list.php
...in the "Week 14, Lab" section.
Here are a few key trigger-related points:
• You don't call a trigger -- instead, it is executed/triggered when the specified action occurs to the database. Consider the following trigger headers:
This trigger, named `inventory_update`, will be executed/fired BEFORE each insert of a row into a table named `orders`:

```sql
create or replace trigger inventory_update
  before insert
  on orders
  for each row
```

This trigger, named `add_prof_ct`, will be executed/fired AFTER each insert of a row into a table named `prof`:

```sql
create or replace trigger add_prof_ct
  after insert
  on prof
  for each row
```

This trigger, named `clear_advisor`, will be executed/fired BEFORE each delete of a row into a table named `prof`:

```sql
create or replace trigger clear_advisor
  before delete
  on prof
  for each row
```

- Oddly, IF you have local declarations for a trigger, you DO put `declare` after the trigger header and before those declarations:

```sql
create or replace trigger inventory_update
  before insert
  on orders
  for each row

declare  
  -- declare section can be omitted if you do not want  
  -- to declare any variables...

  amt_ordered integer;
  item_ordered integer;
  amt_in_stock integer;

begin
```

- ...but just proceed to `begin` if you don't have any local declarations:

```sql
create or replace trigger add_prof_ct
  after insert
  on prof
  for each row
begin
```

- NOTE that Oracle is very concerned about preventing potential "circular" rule firings -- so it will NOT
permit you to query the table on which the trigger is fired. I would not be allowed to query prof within add_prof_ct, for example.

• Within the body of a trigger, you can obtain the attribute values in the "new" row being inserted or updated with the syntax :new. preceding the name of the attribute whose value you want. Likewise, you can obtain the attribute values in the "old" row being deleted or updated with the syntax :old. preceding the name of the attribute whose value you want.

  – For example, the expression
    :new.prof_id
    ...would be the value of the prof_id attribute that was just inserted into prof in the trigger add_prof_ct
  – And as another example, the expression
    :old.prof_id
    ...would be the value of the prof_id attribute that was just deleted from prof in the trigger clear_advisor

Now you will write a PL/SQL trigger for the bookstore scenario.

Create a SQL script 318hw5.sql, and start it off with comments including your name, CS 318 - Homework 5, and the last-modified date.

Next, add the command to run the pop-bks.sql script each time this script is run, so that you have "fresh", original versions of these tables. (Their contents are mucked with below, so it is important that these are "reset" here.)

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 318hw5-out.txt. Then write a SQL*Plus prompt command that says problem 1.

Now, consider the bookstore's order_sum, order_detail, and order_needed tables.

When the stock of some title falls below its order_point, a row indicating that an order of this title is needed should be added to the order_needed table. (Indeed, sell_book makes sure that this happens if necessary when a book is sold.) When a row is added to the order_needed table, the current date is inserted for the date_created attribute of the new order_needed row, and the date_placed for this new row is null (because the order needed has not been placed yet).

order_sum and order_detail hold the details of an order of titles from a publisher. If I had named these tables, I probably would have named them order and order_line_item, respectively -- each row of order_sum represents an "overall" order, including such overall details of an order as the publisher that order is from, the unique order number, the date the order was placed, and the date the order is complete. And order_detail gives the details for one of the titles being ordered as part of that order -- it indicates what order is involved, what line-number of that order this represents, which title is being ordered in this line of the order, and how many of that title are involved in this order.

So, consider -- when an order is placed in response to an order_needed row, surely the order_needed table's date_placed attribute ought to be the very value in the order_sum's table's date_placed attribute for that order. Also, consider the on_order attribute of the title table -- this is
also the proper time to set this attribute to 'T' for each ordered title, also, since now such a title is indeed on-order.

How might we ensure that these updates are made, if necessary, to the proper rows in the order_needed and title tables?

We said very early in the semester that triggers can be used to enhance database integrity. And, indeed, we can use a trigger here for just that purpose.

What action should trigger a corresponding action? Not an insertion into order_sum -- that's overall information for the order, not the individual title for which an order is needed. But it might be handy, after each insert into order_detail, to:

- see IF there is an order_needed for that line item's title that has a null value for date_placed -- if there is a pending order_needed row for that line item's title -- that could now be updated to be the date that the corresponding order was placed;
- change the on_order attribute for that line item's title to 'T', since it is now on-order.

Important additional information:
- DON'T assume that date is the current date -- someone might be entering in this order information the next day, for example, or on Monday after a Friday order.
- Note that an order for some title might be placed even though there isn't an "open" order_needed row for it -- the bookstore manager may choose to simply order more of a title for strategic reasons. So no row in order_needed would be updated in that case, although that title's on_order attribute should still be set to 'T'.

Within your 318hw5.sql, design and implement this PL/SQL trigger order_maint. Follow your trigger with the following testing code:

```
prompt ***********************
prompt demo order_maint
prompt ***********************
prompt

commit;

-- put in some "fake" old order_needed rows for '0805367829'
-- to make sure these AREN'T changed by the trigger
-- (only pending order_needed rows for a title should
-- be changed by an order, you see... 8-)

insert into order_needed
values

insert into order_needed
values
(1001, '0805367829', 10, '07-May-2010', '10-May-2010');
```
var results_code number;
exec :results_code := sell_book('0805367829', 11);

prompt ==============================================================
prompt title is not yet on order, although order is needed
prompt (and can see 2 fake "older" order_needed rows for this title)
prompt ==============================================================
prompt

select isbn, on_order
from title
where isbn = '0805367829';

select *
from order_needed
where isbn = '0805367829';

prompt ==============================================================
prompt simulate an order being placed for this title tomorrow
prompt ==============================================================
prompt

insert into order_sum(ord_no, pub_no, date_placed)
values
(11016, 147, sysdate+1);

insert into order_detail
values
(11016, 1, '0805367829', 10, 0);

prompt ==============================================================
prompt after order of this title, is this title now shown as
prompt on_order?
prompt ==============================================================
prompt

select isbn, on_order
from title
where isbn = '0805367829';

prompt ==============================================================
prompt ...and is JUST the LATEST order_needed date_placed now
prompt tomorrow?
prompt ==============================================================
prompt

prompt
You may add additional testing calls if you would like. Your files 318hw5.sql and 318hw5-out.txt are now ready to submit.

**Problem 2**

Now, for some Java. We'll be able to do a lot more once we actually discuss JDBC, but let's get some Java warmup, at least.

Fun fact: note that you can exit from a Java method "early" (say, because of inappropriate user input...) by using the statement `System.exit(0);`

Write a Java command-line application `TellLength` that:

• expects a single command-line argument

• prints to the screen the length of that command-line argument (when it is treated as a `String`, which fortunately all command-line arguments are read-in as...)

• if NO command-line arguments are given? Complain in a message to the screen.

• if MORE than one command-line argument is given? Cheerfully ignore all but the first one.

Your resulting `TellLength.java` is now ready to submit.

**Problem 3**

Write a command-line Java application `GetPubBooks.java` that will be a bit of an empty shell right now, but that we will add more functionality to later. In the meantime, make sure that it:

• expects 1 or more command-line arguments, names of publishers;

• if it doesn't get at least one command-line argument, it should complain in a descriptive message to the screen and exit;

• otherwise, for each command-line argument, it prints a message to the screen, on its own line, noting that you would eventually list this publisher's books here (including the name of the publisher in the message).

Your resulting `GetPubBooks.java` is now ready to submit.

**Problem 4**

As a little more Java warmup, write a command-line Java application `SellBook.java` that will be another bit of an empty shell right now, but that we will also add more functionality to later. In the meantime, make sure that it:
• expects exactly two command-line arguments, the ISBN to be sold and the quantity of that book to be sold.

• If it doesn't get exactly two command-line arguments, it should complain in a descriptive message to the screen and exit.

• if its second argument cannot be parsed as an integer, it should complain in a different descriptive message to the screen and exit.
  
  – Hint: what method will throw an exception -- that one could catch and handle in this way -- if given a string that cannot be parsed as an int?

• if that second argument/quantity is less than 0, it should complain in yet-another different descriptive message to the screen and exit.

• but if all is well, print a message to the screen noting that two appropriate command-line arguments were given, including those command-line arguments in your message. (Later, you'll actually call the PL/SQL stored function `sell_book` at this point...)

Your resulting `SellBook.java` is now ready to submit.