

# CS 328 - Homework 1

## Deadlines

Due by 11:59 pm on **Friday, January 29, 2016**.

## How to submit

Submit your files for this homework using `~st10/328submit` on nrs-projects, with a homework number of 1

### ***Reminder: Instructions for using the tool ~st10/328submit***

- If they are not already on nrs-projects, transfer your files to be submitted to a directory on nrs-projects.
  - You can do so by using `sftp` (secure file transfer) and connecting to `nrs-projects.humboldt.edu`, and then transferring them
- Once all of your files to be submitted are in a directory on nrs-projects, then use `ssh` to connect to `nrs-projects.humboldt.edu`.
- use `cd` to change to the directory containing the files to be submitted -- for example,  
`cd 328hw01`
- type the command:  
`~st10/328submit`

...and give the number of the homework being submitted (or whatever number you have been asked to do for lab-related files) when asked, and answer that, `y`, you do want to submit all of the files with of-interest-to-328 suffixes in the current directory. (Note that I don't mind if a few extraneous files get submitted as well -- I'd rather receive too many files than too few, and typing in all of the file names for each assignment is just too error-prone.)

- you are expected to carefully check the list of files that the tool believes have been submitted, and make sure all of the files you hoped to submit were indeed submitted! (The most common error is to try to run `~st10/328submit` while in a different directory than where your files are.)

## Purpose

To get a little familiar with the "bookstore" database we will working with semester, to write a few refresher SQL queries using this database, and to start playing a bit with PL/SQL.

## Problem 1

We will be practicing many of the concepts and languages we will be using this semester on a little bookstore database, whose tables can be found in the scripts `create-bks.sql` and `pop-bks.sql`.

Since you will be spending a lot of time using this database, you would be well-served to start getting familiar with it now.

As one means of getting familiar with this database, write each of the relations in this database in **relation structure form**.

- (Recall that we write a relation in relation-structure form by writing the name of the relation, followed by an opening parenthesis, followed by a comma-separated list of **all** of the attributes of that relation, followed by a closing parenthesis,
- being sure to write the primary-key attribute(s) in all-uppercase,
- and writing SQL-syntax foreign key clauses for each foreign key **after** the closing parenthesis.)
- ASK ME if you are not sure what I mean by this!

For example, you could write the relations `empl` and `dept` from CS 325's script `set-up-ex-tbls.sql` in relation structure form as:

```
dept(DEPT_NUM, dept_name, dept_loc)
empl(EMPL_NUM, empl_last_name, job_title, mgr, hiredate, salary,
      commission, dept_num)
      foreign key (dept_num) references dept,
      foreign key (mgr) references empl(empl_num)
```

Submit a file `rs-design-bks.txt` containing your resulting relation structures.

## Problem 2

Create copies of `create-bks.sql` and `pop-bks.sql`, and use these to create and populate these tables.

Then, write a script `sql-play.sql` that meets the following specifications:

- start it with comment(s) including CS 328 - Problem 2, your name, and the last-modified date
- start spooling to a file `sql-play-out.txt` (and make sure you `spool off` at the end of this script!)
- write a `prompt` command including your name
- precede each of the following with a `prompt` command giving the problem part being answered

### 2 part a

Write a query projecting just the names of publishers and what state each is in, displaying the results in alphabetical order of publisher name.

### 2 part b

Write a query projecting the names of titles published by Addison-Wesley and the current quantity on hand of each, displaying the results in reverse order of quantity on hand.

**2 part c**

Write a query projecting the number of titles available from each publisher, projecting the publisher's name and the number of titles available from that publisher, displaying the results in reverse order of number of titles available from that publisher, and giving the number of titles available the displayed column heading #TITLES

**2 part d**

Write a query projecting the names of titles involved in order 11010, and the quantity of each title being ordered.

**2 part e**

Write a query projecting the name of the publisher involved in the latest order placed.

Make sure you turned spooling off in `sql-play.sql`, and submit your resulting `sql-play.sql` and `sql-play-out.txt`.

**Problem 3**

Consider your trigger `inventory_update` from your script `trigger1.sql` from the Week 1 Lab. You might not have gotten your copy debugged by the end of lab -- here is a working version from the 3:00 pm lab:

(Note: I am deliberately NOT posting `trigger1.sql` yet, because I want you to modify your version, not simply paste in a posted version.)

```
-- CS 328 - Week 1 Lab
-- last modified: 2016-01-22
-- your name

-- this trigger only permits orders to be inserted
--     if their quantity is reasonable,
--     and if they are, update the inventory accordingly

create or replace trigger inventory_update
    before insert
    on orders
    for each row
declare
    amt_ordered integer;
    item_ordered inventory.item_num%TYPE;
    amt_in_stock integer;
begin
    -- set some local variable for convenience

    amt_ordered := :new.order_quantity;
    item_ordered := :new.item_num;
```

```
select item_quantity
into amt_in_stock
from inventory
where item_num = item_ordered;

-- prevent this insert if the order is not for at least 1 item

if amt_ordered <= 0 then

    raise_application_error( -20600,
        'Order number ' || :new.order_num ||
        ' cannot be placed, because order quantity of ' ||
        amt_ordered || ' must be at least 1');

end if;

-- allow this insertion and update inventory
-- IF I have enough in stock for this order

if (amt_in_stock >= amt_ordered) then

    update inventory
    set     item_quantity = item_quantity - amt_ordered
    where  item_num = item_ordered;

    dbms_output.put_line('yay!!! updated inventory!');

else

    raise_application_error( -20601,
        'insufficient stock on stand - order rejected');

end if;

end;
/
show errors
```

### 3 part a

COPY your file `trigger1.sql` into your `328hw01` directory. If it works, proceed to 3 part b -- otherwise, using the above, modify your version of `inventory_update` until it does work.

### 3 part b

For a little PL/SQL practice...

First: declare a local variable `curr_order_num`, and write an assignment statement setting it to the

order number in the `insert` statement triggering this trigger.

Second: now use `curr_order_num` whenever referring to this order number in the remainder of the trigger.

Third: notice how, when there is not enough inventory on hand to fill an order, we currently say "insufficient stock on hand - order rejected".

Nope, the client wants this error message to also include:

- the order number of the order rejected
- the item number in this rejected order
- the quantity in the rejected order, and
- the current quantity on hand of the item number in the rejected order
- ...in as clear an error statement as you can craft.

Make these changes, and make sure your resulting trigger still works.

### 3 part c

Now you'll demonstrate that your trigger works. Make a script `trigger-test.sql` meeting the following specifications:

- start it with comment(s) including CS 328 - Problem 3 part c, your name, and the last-modified date
- start spooling to a file `trigger-test-out.txt` (and make sure you `spool off` at the end of this script!)
- write a prompt command including your name
- include the command:  

```
set serveroutput on
```
- write a prompt command saying here are the current contents of `inventory` and `orders`, followed by queries displaying the current contents of each.
- write a prompt command indicating that a SUCCESSFUL insert should follow,  
write a SQL `insert` into `orders` that SHOULD be allowed,  
write a query displaying the contents of `inventory`, and  
write a query displaying the contents of `orders`
  - these should show that the order was inserted, and that the ordered item's quantity was reduced
- write a prompt command indicating that a LESS-THAN-0 order insertion attempt should follow,  
write a SQL `insert` into `orders` with a negative order quantity,  
write a query displaying the contents of `inventory`, (which should be unchanged),  
write a query displaying the contents of `orders`, (which should be unchanged)

- write a `prompt` command indicating that an ORDERING-TOO-MUCH order insertion attempt should follow,

write a SQL `insert` into `orders` with a quantity that is greater than the current quantity on hand of the item being ordered,

write a query displaying the contents of `inventory`, (which should be unchanged),

write a query displaying the contents of `orders`, (which should be unchanged)

- add any additional tests that you would like (preceding them with `prompt` commands letting me know what you are testing)
- turn off spooling

Submit your resulting `trigger1.sql`, `trigger-test.sql` and `trigger-test-out.txt`.