Assignment 4- Advance DML

Sep 2021

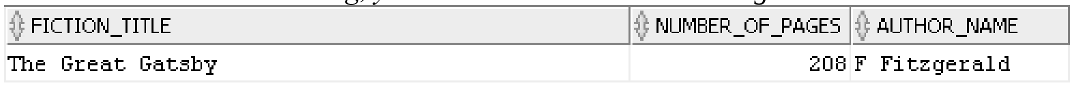
Instructions:

- **Do your own work:** This is an individual assignment and you must do your own work and create your own SQL statements. If you are caught cheating on this, you will receive a zero on this assignment and be reported to the Dean of Students. Also, this homework prepares you for the exam coming up so doing the work now will help you learn and do well on when it counts more.
- **<u>What to turn in</u>**
    - Clearly separate your code for each question. Save your code into one SQL file with the naming format: LastName_FirstName_UTEid. Please make sure the lastname and firstname you use matches what is in Canvas.
    - Submit your .sql file on Canvas before the deadline. Late submissions receive 50% off. No submissions will be accepted 24 hours after the deadline.
    - Only submissions in .sql or .txt format will be accepted. All other file formats will be given a 0.
- The SQL problems below will be based on the DDL script that is posted on the Canvas instructions. Download that script and run it before you start.

Joins and multi-table SELECT problems:

1. Write a query that returns the *first_name, last name, email,* and *accrued_fees* of all patrons from the "Northeast Central" branch. Filter out any patrons with no accrued fees. Sort results to show the patrons with highest fees first. *Hint on how to write this query and others in the future. Build step by step. Start by writing a query that joins Location to Patron and pulls all columns. Then add a WHERE to filter down to patrons from the right branch with accrued fees. Then last step is to pick the appropriate columns in your SELECT and add in the Sort (i.e. Order by).*

2. Write a query that joins the appropriate tables in order to show the title of a book, number of pages, and the author's full name. Only include titles that are in 'Book' format (i.e. 'B') and ignore others like Audiobooks or DVD. Also only pull books that are of the 'Fiction' genre (i.e. FIC). I expected something like the following but with more rows. Sort results first by the author_name <u>using the column alias</u> and then by title. *FYI: If I don't specify that the sort is in a certain order or in descending, you can assume it's an ascending order sort.*



FICTION_TITLE	NUMBER_OF_PAGES	AUTHOR_NAME
The Great Gatsby	208	F Fitzgerald

…………………………………………………………………………………………………

…………………………………………………………………………………………………

3. Write a query that pulls *Title, Format, Genre,* and *ISBN* of all the titles that currently do not have holds. While there are ways to do this using a subquery, I'd like you to use an outer join to accomplish this to show you understand how to properly use them. Sort results by genre and title ascending.

4. Write a query that generates results to show the title, publisher, number of pages, and genre of all titles that are either a book or e-book format (i.e. 'B', 'E'). Please also include an additional column called "Reading_Level" that indicates the reading level of the book which can be 'College', 'High School', or 'Middle School'. The reading level is determined simply by the number of pages.
    - If a book is more than 700 pages, its level is 'College'
    - If a book is more than 250 page but less than 700, its level is 'High School'
    - If a book is less than 250 pages, its level is 'Middle School'

    Sort the final result set by *Reading_Level* and *Title*. Results should look something like this but with more records.

READING_LEVEL	TITLE	PUBLISHER	NUMBER_OF_PAGES	GENRE
College	A Game of Thrones	Bantam	704	FAN
…	…	…	…	…
High-School	1491: New Revelations of the Americas	Vintage	541	HIS
…	…	…	…	…
Middle-School	Into the Wild	Anchor Books	240	BIO

Summary Problems:

5. Write a query that pulls patrons and their associated checkouts. Filter records down to just the checkouts that are not flagged as “Late” (i.e. ignore is late_flag is ‘Y’). Summarize the data to return just two columns named as follows:
   Zip – the zip column from patron
   Average_Accrued_Fees – the average of accrued_fees for each zip. Round the results to nearest 2 decimals.
   Order results by average_accrued_fee from highest to lowest

6. Find all the titles that have more than 1 author and then return the ***title***, ***genre***, and ***author_count*** for that book. Sort your results by genre descending and the title ascending. You’ll need to find the count of authors for each title and then filter out titles that has author counts of 1.

   A single row example row could look something like this:

TITLE	GENRE	AUTHOR_COUNT
The President Is Missing	MYS	2

   Not done yet! Update the query to add in a filter to only show titles that have authors that include the letters ‘PhD’ in their last_name. This should only return titles that have at least 2 or more authors that are PhDs.

Subquery Problems:

7. Write a SELECT statement that answers this question: Which titles have a higher than average number of pages? i.e. Just find the books that have a number of pages greater than the average number of pages for all books.
   Return the title, publisher, number_of_pages, genre for each title that fit this criteria.
   Sort the results by the genre ascending and then by number_of_pages decsending.

8. Here’s a Real-World Business Scenario! Lots of times you may to look at a list of values and figure out which values are not in another list. So in this situation, the Library Member Outreach team would like to know all the Patron’s that currently do not have a phone on file with us. Write a SELECT statement that returns the first_name, last_name, and email from the *Patron* table for each patron that has a Patron record but does not have a phone records on *Patron_Phone*. Sort final results by the last_name. *Hint: Use a subquery that pulls a list of patron_ids from Patron_Phone that you then use as part of an IN clause for a separate query from Patron.*

DataTypes & Function Problems:

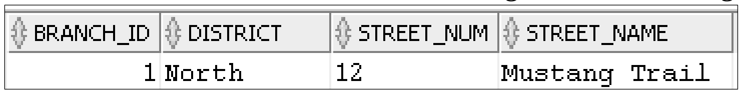
9. Please write a checkout status report for patrons that have any checkouts. The report should format the data to be more user-friendly and clear for volunteers to understand. It should return the following four columns:

   - PATRON – a combination of the patron’s first and last name
   - CHECKOUT_DUE_BACK – Concatenate text and the appropriate columns to return a record for each checkout and when it’s due back. See examples below
   - RETURN_STATUS – If the checkout has not been returned (i.e. date_in is null) show “Not returned yet”, otherwise show “Returned”.
   - FEES – Concatenate the text and the patron’s accrued fees while formatting fees as a currency to match examples below
   - Sort by *retruned_status and due_back_date.*

PATRON	**CHECKOUT_DUE_BACK**	**RETURN_STATUS**	**FEES**
Roda Devil	Checkout 16 due back on 11-FEB-21	Not returned yet	Accrued fee total is: $1.10
Laure Chatin	Checkout 6 due back on 11-FEB-21	Not returned yet	Accrued fee total is: $.20
…	…	…	…
…	…	…	…
Joline Gloyens	Checkout 3 due back on 04-FEB-21	Returned	Accrued fee total is: $.00
Oren Darycott	Checkout 15 due back on 11-FEB-21	Returned	Accrued fee total is: $6.50

NOTE: This is just an example design. Your results could produce different values and number of rows

10. Write a query that uses the appropriate string parsing functions to return the branch_id, a "District" column based on the first word in the branch_name, and also divides the address into two columns called street number and street name. Street number is the first numbers on the address and everything that follows the first space is considered the street name. Results should look something like the following but for all branches



BRANCH_ID	DISTRICT	STREET_NUM	STREET_NAME
1	North	12	Mustang Trail

11. Write a single SELECT that returns the same results that the UNION query did on a #4 above but this time use a CASE statement to dynamically change the reading_level column. Make sure your new query returns the same results as the union query.

12. Write a SELECT statement that pulls all books and their number of checkouts. Include a column called row_number that uses row_number function to create a row number. That means, do not use the rownum pseudo colomn that only works in the WHERE clause. Include all books, even ones that have no checkouts. Results should show the following columns for all titles.
    - Popularity_Rank – Use dense_rank based on the count of distinct checkouts.
    - Title
    - Number_of_Checkouts – Count the distinct checkout_ids attributed to this title if there are any

    Note: Since the library system is new at Guillory, they don't have a lot of checkouts across all the times, so it's quite possible that some many books will have the same number of checkouts in cases where the titles have only been checked out once. That's okay. As times goes one and more checkouts occur, this report will become more helpful in identifying popular and unpopular titles.

    Once you have your query running, as a final step sort results by number_of_checks descending then title ascending.

    After you complete this query, make it an in-line subquery and select * from it like a table but only return row_number 58 like so to reveal one of Clint's favorite books

Row_Number	Title	Number_of_Checkouts
58	Man Called Ove	0