AM05 Data Mgmt – Lab 3

This lab asks you to complete some SQL queries that include joins, aggregations and views.

SQL Queries

These queries are based on the ap database used in the hands on exercises for AM05 Session 1. The questions are the same as in Murach's MySQL (3rd Edition). I have indicated the original chapter and question numbering so that you can easily find the solutions in the zip file you should have downloaded from the murach.com website. See the material for the hands on exercises for the first class session to find out where to download the zip file if you have not already done so. These solutions can be found in the mysql>ex_solutions>{ch04,ch06,ch07,ch12} directories.

1. [Ch 4 q 4] Write an SELECT statement that returns these five columns

vendor_name	The vendor_name column from the Vendors table
invoice_date	The invoice_date column from the Invoices table
invoice_number	The invoice_number column from the Invoices table
li_sequence	The invoice_sequence column from the Invoice_Line_Items table
li_amount	The line_item_amount column from the Invoice_Line_Items tabel

   Use aliases for the tables. This should return 118 rows.

   Sort the final result set by vendor_name, invoice_date, invoice_number, and invoice_sequence.

2. [Ch 4 q 5] Write an SELECT statement that returns three columns

vendor_id	The vendor_id column from the Vendors table
vendor_name	The vendor_name column from the Vendors table
contact_name	A concatenation of the vendor_contact_first_name and vendor_contact_last_name with a space between

   Return one row for each vendor whose contact has the same last name as another vendor's contact. This should return 2 rows. *Hint: Use a self-join to check that the vendor_id columns are not equal but the vendor_contact_last_name columns are equal*.

   Sort the final result set by vendor_contact_last_name.

3. [Ch 4 q 6] Write an SELECT statement that returns these three columns

account_number	The account_number column from the General_Ledger_Accounts table
account_description	The account_description column from the General_Ledger_Accounts table
invoice_id	The invoice_id column from the Invoice_Line_Items table

   Return one row for each account number that has never been used. This should return 54 rows. *Hint: Use an outer joint and only return rows where the invoice_id column contains a null value.*

   Remove the invoice_id column from the SELECT clause.

   Sort the final result set by account_number column.

4. [Ch 6 q 4] Write an SELECT statement that returns one row for each general ledger account number that contains three columns:

   The account_description column from the General_Ledger_Accounts table

   The count of the items in the Invoice_Line_Items table that have the same account_number

   The sum of the line_item_amount columns in the Invoice_Line_Items table that have the same account number

   Return only htose rows where the count of line times is greater than 1. This should return 10 rows.

   Group the result set by the account_description column.

   Sort the result set in descending sequence by the sum of the line item amounts.

5. [Ch 6 q 5] Modify the solution to exercise 4 so it returns only invoices dated in the second quarter of 2018 (April 1, 2018 to June 30, 2018). This should still return 10 rows but with some different line item counts for each vendor. *Hint: Join to the Invoices table to code a search condition based on invoice_date.*

6. [Ch 6 q 8] Write an SELECT statement that answers this question: What are the last payment date and total amount due for each vendor with each terms id? Return these columns:

   The terms_id column from the Invoices table

   The vendor_id column from the Invoices table

   The last payment date for each combination of terms id and vendor id in the Invoices table

   The sum of the balance due (invoice_total – payment_total – credit_total) for each combination of terms id and vendor id in the Invoices table

   Use the WITH ROLLUP operator to include rows that give a summary for each terms id as well as a row that gives the grand total. This should return 40 rows.

   Use the IF and GROUPING functions to replace the null values in the terms_id and vendor_id columns with literal values if they're for summary rows.

7. [Ch 7 q 2] Write an SELECT statement that answers this question: Which invoices have a payment total that's greater than the average payment total for all invoices with a payment total greater than 0?

   Return the invoice_number and invoice_total columns for each invoice. This should return 20 rows.

   Sort the results by the invoice_total column in the descending order.

8. [Ch 7 q 4] Write a SELECT statement that returns four columns: vendor_name, invoice_id, invoice_sequence, and line_item_amount.

   Return a row for each line item of each invoice that has more than one line item in the Invoice_Line_Items table. *Hint: Use a subquery that tests for invoice_sequence > 1*. This should return 6 rows.

   Sort the results by the vendor_name, invoice_id, and invoice_sequence columns.

9. [Ch 7 q 7] Use a correlated subquery to return one row per vendor, representing the vendor's oldest invoice (the one with the earliest date). Each row should include these four columns: vendor_name, invoice_number, invoice_date, and invoice_total. This should return 34 rows.

   Sort the results by the vendor-name column.

10. [Ch 7 q 8] Rewrite exercise 9 so it gets the same result but uses an inline view instead of a correlated subquery.

11. [Not avail] Rewrite exercise 9 so it gets the same result but uses a common table expression (CTE) instead of a correlated subquery.

12. [Ch 12 q 1] Create a view named open_items that shows the invoices that haven't been paid.

    This view should return four columns from the Vendors and Invoices tables:

    vendor_name, invoice_number, invoice_total, and balance_due (invoice_total – payment_total – credit total)

13. [Ch 12 q 2] Write a SELECT statement that returns all of the columns in the open_items view that you created in the previous exercise, with one row for each invoice that has a balance due of $1,000 or more.