**Use ap database to complete Exercises 1-8 at the end of Chapter 7 in your book.**

**\*Screen clip your query and results below. DO NOT screen print all when the result sets has many rows … just the first few and last few rows including the row count. (12.5 points each)**

1a. Write a SELECT statement that returns the same result set as this SELECT statement, but don’t use a join. Instead, use a subquery in a WHERE clause that uses the IN keyword.

SELECT DISTINCT vendor\_name

FROM vendors JOIN invoices

ON vendors.vendor\_id = invoices.vendor\_id

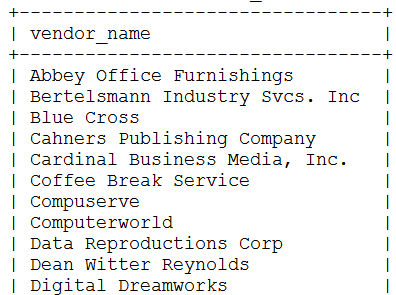
ORDER BY vendor\_name;

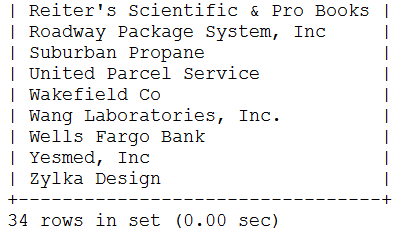
**SELECT vendor\_name**

**FROM vendors**

**WHERE vendor\_id IN (SELECT DISTINCT vendor\_id FROM invoices)**

**ORDER BY vendor\_name;**





1b. For the SELECT statement in Q1, what is the goal of that statement?

List all vendors who have invoices.

2. Write a 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.

**SELECT invoice\_number, invoice\_total**

**FROM invoices**

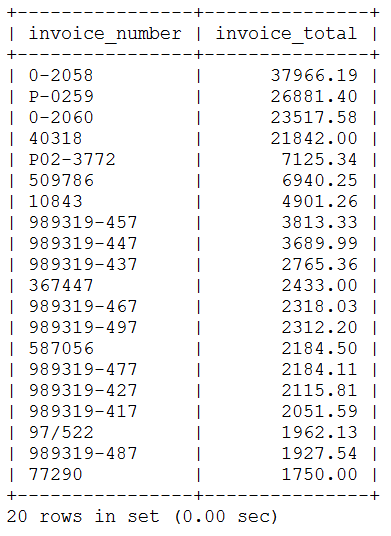
**WHERE payment\_total >**

**(SELECT AVG(payment\_total)**

**FROM invoices**

**WHERE payment\_total > 0)**

**ORDER BY invoice\_total DESC;**



3. Write a select statement that returns two columns from the general\_ledger\_accounts table: account\_number and account\_description.

Return one row for each account number that has never been assigned to any line item in the invoice\_line\_items table. To do that, use a subquery introduced with the NOT EXISTS operator. This should return 54 rows.

Sort the results by the account\_number column.

**SELECT account\_number, account\_description**

**FROM general\_ledger\_accounts gl**

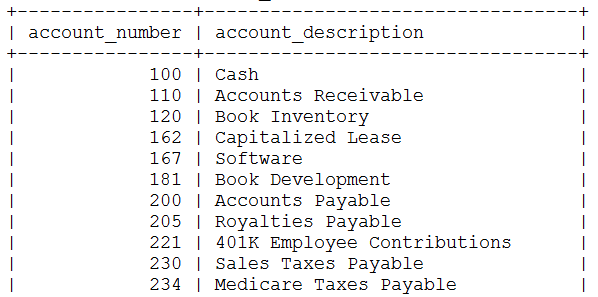
**WHERE NOT EXISTS**

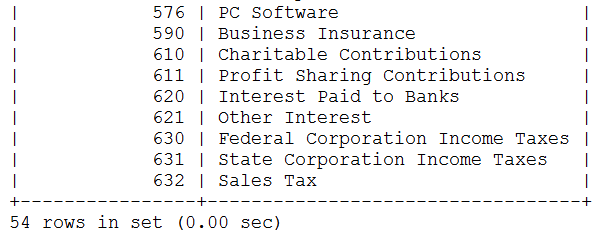
**(SELECT \***

**FROM invoice\_line\_items**

**WHERE account\_number = gl.account\_number)**

**ORDER BY account\_number;**





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 tha one line item in the invoice\_line\_items table. *Hint: Us a subquery that tests for invoice\_sequence > 1.* This should return 6 rows.

**SELECT vendor\_name, i.invoice\_id, invoice\_sequence, line\_item\_amount**

**FROM vendors v JOIN invoices i**

**ON v.vendor\_id = i.vendor\_id**

**JOIN invoice\_line\_items li**

**ON i.invoice\_id = li.invoice\_id**

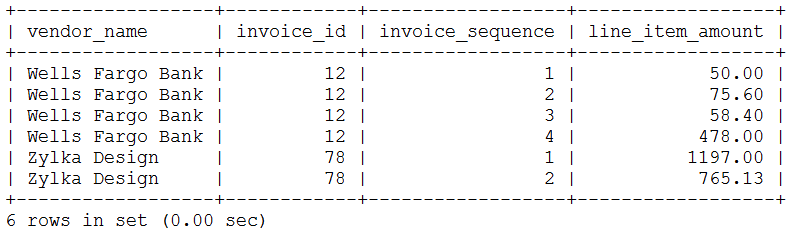
**WHERE i.invoice\_id IN**

**(SELECT DISTINCT invoice\_id**

**FROM invoice\_line\_items**

**WHERE invoice\_sequence > 1)**

**ORDER BY vendor\_name, i.invoice\_id, invoice\_sequence;**

****

5. Write a SELECT statement that returns two columns: vendor\_id and the largest unpaid invoice for each vendor. To do this, you can group the result set by the vendor\_id column. This should return 7 rows.

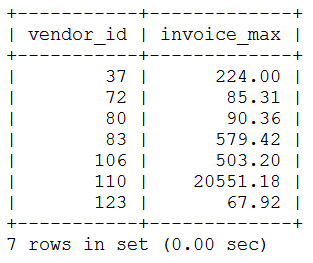
Write a second SELECT statement that uses the first SELECT statement in it’s FROM clause. The main query should return a single value that represents the sum of the largest unpaid invoices for each vendor. This should return 1 row.

**SELECT vendor\_id, MAX(invoice\_total) AS invoice\_max**

**FROM invoices**

**WHERE invoice\_total - credit\_total - payment\_total > 0**

**GROUP BY vendor\_id;**

****

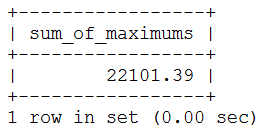
**SELECT SUM(invoice\_max) AS sum\_of\_maximums**

**FROM (SELECT vendor\_id, MAX(invoice\_total) AS invoice\_max**

**FROM invoices**

**WHERE invoice\_total - credit\_total - payment\_total > 0**

**GROUP BY vendor\_id) t;**

****

6. Write a SELECT statement that returns the name, city, and state of each vendor that’s located in a unique city and state. In other words, don’t include vendors that have a city and state in common with another vendor. This should return 38 rows.

Sort the results by the vendor\_state and vendor\_sity columns.

**SELECT vendor\_name, vendor\_city, vendor\_state**

**FROM vendors**

**WHERE CONCAT(vendor\_state, vendor\_city) NOT IN**

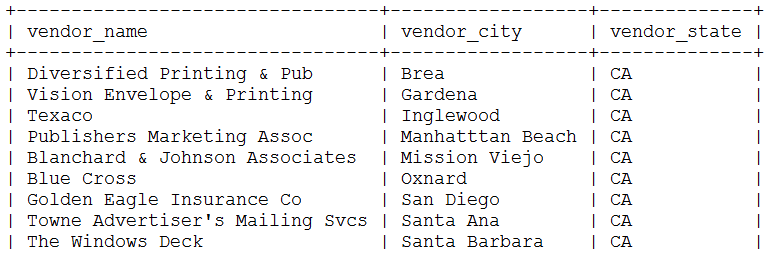
**(SELECT CONCAT(vendor\_state, vendor\_city) as vendor\_city\_state**

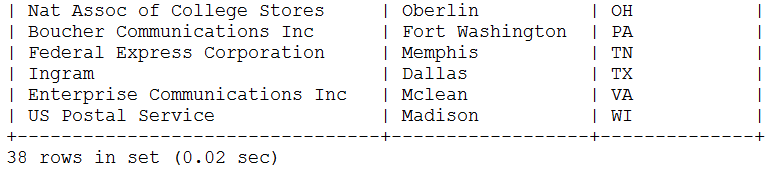
**FROM vendors**

**GROUP BY vendor\_city\_state**

**HAVING COUNT(\*) > 1)**

**ORDER BY vendor\_state, vendor\_city;**





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.

**SELECT vendor\_name, invoice\_number, invoice\_date, invoice\_total**

**FROM invoices i JOIN vendors v**

**ON i.vendor\_id = v.vendor\_id**

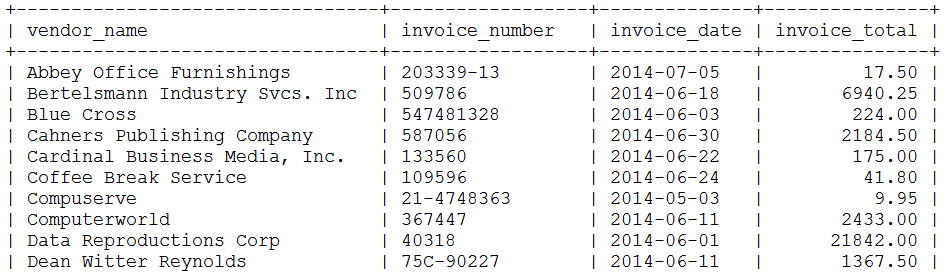
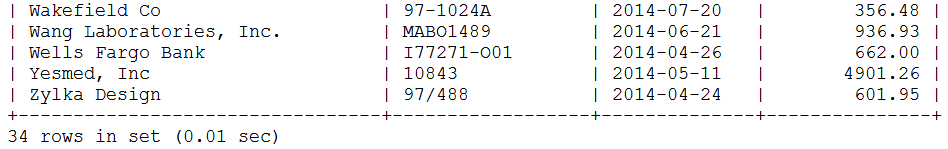
**WHERE invoice\_date =**

**(SELECT MIN(invoice\_date)**

**FROM invoices**

**WHERE vendor\_id = i.vendor\_id)**

**ORDER BY vendor\_name;**



8. Rewrite Q7 so it gets the same result but uses an inline view instead of a correlated subquery.

**SELECT vendor\_name, invoice\_number,**

**invoice\_date, invoice\_total**

**FROM invoices i**

**JOIN**

**( SELECT vendor\_id, MIN(invoice\_date) AS oldest\_invoice\_date**

**FROM invoices**

**GROUP BY vendor\_id ) oi**

**ON i.vendor\_id = oi.vendor\_id AND**

**i.invoice\_date = oi.oldest\_invoice\_date**

**JOIN vendors v**

**ON i.vendor\_id = v.vendor\_id**

**ORDER BY vendor\_name;**