**LAB 6**

**Chapter 12 exercises**

**Exercise 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).

A row should only be returned when the balance due is greater than zero, and the rows should be in sequence by vendor\_name.

USE ap;

CREATE OR REPLACE VIEW open\_items

AS

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

invoice\_total - payment\_total - credit\_total AS balance\_due

FROM vendors JOIN invoices

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

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

ORDER BY vendor\_name

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Exercise 2**:

Write a SELECT statement that returns all of the columns in the open\_items view that you created in exercise 1, with one row for each invoice that has a balance due of $1000 or more.

SELECT \*

FROM open\_items

WHERE balance\_due >= 1000

A screenshot of a computer

Description automatically generated

**Exercise 3**:

Create a view named open\_items\_summary that returns one summary row for each vendor that has invoices that haven’t been paid. Each row should include vendor\_name, open\_item\_count (the number of invoices with a balance due), and open\_item\_total (the total of the balance due amounts) The rows should be sorted by the open item totals in descending sequence.

CREATE OR REPLACE VIEW open\_items\_summary

AS

SELECT vendor\_name, COUNT(\*) AS open\_item\_count,

SUM(invoice\_total - credit\_total - payment\_total) AS open\_item\_total

FROM vendors JOIN invoices

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

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

GROUP BY vendor\_name

ORDER BY open\_item\_total DESC

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Exercise 4**:

Write a SELECT statement that returns just the first 5 rows from the open\_items\_summary view that you created in exercise 3.

SELECT \*

FROM open\_items\_summary

LIMIT 5

**A screenshot of a computer

Description automatically generated**

**Exercise 5**:

Create an updatable view named vendor\_address that returns the vendor\_id column and all of the address columns for each vendor.

CREATE OR REPLACE VIEW vendor\_address

AS

SELECT vendor\_id, vendor\_address1, vendor\_address2, vendor\_city, vendor\_state, vendor\_zip\_code

FROM vendors

A screenshot of a computer

Description automatically generated

**Exercise 6**:

Write an UPDATE statement that changes the address for the row with a vendor ID of 4 so the suite number (Ste 260) is stored in the vendor\_address2 column instead of the vendor\_address1 column.

UPDATE vendor\_address

SET vendor\_address1 = '1990 Westwood Blvd',

vendor\_address2 = 'Ste 260'

WHERE vendor\_id = 4

A screenshot of a computer

Description automatically generated

**Chapter 13 exercises**

**Exercise 1**:

Write a script that creates and calls a stored procedure named test. This stored procedure should declare a variable and **set it to the count of all rows** in the Invoices table that have a balance due that’s greater than or equal to $5,000. Then, the stored procedure should display a result set that displays the variable in a message like this: 3 invoices exceed $5,000.

**USE ap;**

**DROP PROCEDURE IF EXISTS test;**

**-- Change statement delimiter from semicolon to double front slash**

**DELIMITER //**

**CREATE PROCEDURE test()**

**BEGIN**

**DECLARE invoice\_count INT;**

**SELECT COUNT(\*)**

**INTO invoice\_count**

**FROM invoices**

**WHERE invoice\_total - payment\_total - credit\_total >= 5000;**

**SELECT CONCAT(invoice\_count, ' invoices exceed $5000.') AS message;**

**END//**

**-- Change statement delimiter from semicolon to double front slash**

**DELIMITER ;**

**CALL test();**

**A screenshot of a computer

Description automatically generated**

**Exercise 2**:

Write a script that creates and calls a stored procedure named test. This stored procedure should use two variables to store (1) the count of all of the invoices in the Invoices table that have a balance due and (2) the sum of the balances due for all of those invoices. If that total balance due is greater than or equal to $30,000, the stored procedure should display a result set that displays the values of both variables. Otherwise, the procedure should display a result set that displays a message like this:

**Total balance due is less than $30,000.**

**USE ap;**

**DROP PROCEDURE IF EXISTS test;**

**-- Change statement delimiter from semicolon to double front slash**

**DELIMITER //**

**CREATE PROCEDURE test()**

**BEGIN**

**DECLARE count\_balance\_due INT;**

**DECLARE total\_balance\_due DECIMAL(9,2);**

**SELECT COUNT(\*), SUM(invoice\_total - payment\_total - credit\_total)**

**INTO count\_balance\_due, total\_balance\_due**

**FROM invoices**

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

**IF total\_balance\_due >= 30000 THEN**

**SELECT count\_balance\_due AS count\_balance\_due,**

**total\_balance\_due AS total\_balance\_due;**

**ELSE**

**SELECT 'Total balance due is less than $30,000.' AS message;**

**END IF;**

**END//**

**-- Change statement delimiter from semicolon to double front slash**

**DELIMITER ;**

**CALL test();**

**A screenshot of a computer

Description automatically generated**

**Chapter 14 exercises**

**Exercise 1**:

Write a script that creates and calls a stored procedure named test. This procedure should include a set of three SQL statements coded as a transaction to reflect the following change: United Parcel Service has been purchased by Federal Express Corporation and the new company is named FedUP. Rename one of the vendors and delete the other after updating the vendor\_id column in the Invoices table.

If these statements execute successfully, commit the changes. Otherwise, roll back the changes.

**USE ap;**

**DROP PROCEDURE IF EXISTS test;**

**DELIMITER //**

**CREATE PROCEDURE test()**

**BEGIN**

**DECLARE sql\_error INT DEFAULT FALSE;**

**DECLARE CONTINUE HANDLER FOR SQLEXCEPTION**

**SET sql\_error = TRUE;**

**START TRANSACTION;**

**UPDATE invoices**

**SET vendor\_id = 123**

**WHERE vendor\_id = 122;**

**DELETE FROM vendors**

**WHERE vendor\_id = 122;**

**UPDATE vendors**

**SET vendor\_name = 'FedUP'**

**WHERE vendor\_id = 123;**

**IF sql\_error = FALSE THEN**

**COMMIT;**

**SELECT 'The transaction was committed.';**

**ELSE**

**ROLLBACK;**

**SELECT 'The transaction was rolled back.';**

**END IF;**

**END//**

**DELIMITER ;**

**CALL test();**

**A screenshot of a computer

Description automatically generated**