Assignment 7

Hands-on

Summer 2018

***J. Packy Laverty***

***laverty@rmu.edu***

Contents

[**Introduction 4**](#_Toc508295853)

[**How to complete Hand-on Tutorial Requirements and Review Questions 4**](#_Toc508295854)

[**Oracle Error Codes Summary 5**](#_Toc508295855)

[**Documentation Requirements 5**](#_Toc508295856)

[**Required Professional Appearance of Outputted Results 5**](#_Toc508295857)

[**Improving the Professional Appearance of the SQL Statement and Results using SQL\*Plus 6**](#_Toc508295858)

[**Improving the Professional Appearance of the SQL Statement and Results using PuTTY 6**](#_Toc508295859)

[**Improving the Professional Appearance of the Results using the SQL SUBSTR() and TO-CHAR() Function and Columns Aliases (AS) 7**](#_Toc508295860)

[**SUBSSTR() Function 8**](#_Toc508295861)

[**AS (Column Alias) 9**](#_Toc508295862)

[**TO\_CHAR() Function 10**](#_Toc508295863)

[**Table Layout for the PSP Case 12**](#_Toc508295864)

[**CUSTOMER Table (PRIMARY KEY CUSTOMER NUMBER) 12**](#_Toc508295865)

[**CUSTOMER SHIPPING ADDRESS TABLE (PRIMARY KEY CUSTOMER SHIPPING ID) 12**](#_Toc508295866)

[**DRAWING TABLE (PRIMARY KEY DRAWING NUMBER) 13**](#_Toc508295867)

[**PRICE LIST TABLE (PRIMARY KEY OD + WALL + LENGTH + TYPE) 14**](#_Toc508295868)

[**OPTION TABLE (Primary Key OPTION NUMBER) 14**](#_Toc508295869)

[**DRAWING OPTION TABLE (Primary Key DRAWING NUMBER + OPTION NUMBER) 14**](#_Toc508295870)

[**TUBULAR TABLE (Primary Key Raw Material ID) 15**](#_Toc508295871)

[**SUPPLIES TABLE (Primary Key Raw Material ID) 15**](#_Toc508295872)

[**RAW MATERIAL REQUIREMENTS TABLE (Primary Key PACKLIST NUMBER + RAW MATERIAL ID 15**](#_Toc508295873)

[**RAW MATERIAL FIFO TABLE (Primary Key FIFO ID, auto sequence) 15**](#_Toc508295874)

[**RAW MATERIAL DROP INVENTORY (Primary Key DROP REFERENCE NUMBER, auto sequence) 16**](#_Toc508295875)

[**VENDOR TABLE (Primary Key VENDOR NUMBER auto sequence, auto sequence) 16**](#_Toc508295876)

[**INVENTORY VENDOR (Primary Key VENDOR NUMBER + RAW MATERIAL NUMBER 16**](#_Toc508295877)

[**LABOR OPERATION TABLE (Primary Key OPERATION NUMBER) 16**](#_Toc508295878)

[**PACKLIST TABLE (Primary key PACKLIST NUMBER) 17**](#_Toc508295879)

[**JOBCOST TABLE (Primary Key JOB COST REFERENCE NUMBER, auto sequence) 17**](#_Toc508295880)

[**EMPLOYEE TABLE (Primary Key Clock Number) 18**](#_Toc508295881)

[**TIME CARD TABLE (Primary Key TIME CARD REFERENCE NUMBER, auto sequence) 18**](#_Toc508295882)

[**2.0 PSP Level 2 Queries 19**](#_Toc508295883)

[**2.1 Apply Customer and Drawing Join with Packing List Subquery 19**](#_Toc508295884)

[**2.2 Apply Customer and Drawing Join with Packing List Subquery 19**](#_Toc508295885)

[**2.3 Apply an Inner Join between Job Cost and Packing List with CHAR() 20**](#_Toc508295886)

[**2.4 Apply an Inner Join between Job Cost and Packing List with CHAR() 20**](#_Toc508295887)

[**2.5 Apply an Inner Join between Job Cost and Packing List with CHAR() 21**](#_Toc508295888)

[**2.6 COPYING TABLES 22**](#_Toc508295889)

[**2.7 Copy Packing List Table 22**](#_Toc508295890)

[**2.8 Copy Customer List Table 22**](#_Toc508295891)

[**2.9 Insert Customers 23**](#_Toc508295892)

[**2.10 Apply UNION between Blanket Customers and Customers Tables 23**](#_Toc508295893)

[**2.11 Apply INTERSECTION between Blanket Customers and Customers 23**](#_Toc508295894)

[**2.12 Apply DIFFERENCE between Blanket Customers and Customers 23**](#_Toc508295895)

[**2.13 Apply Subquery using Drawing and Job Cost Tables 24**](#_Toc508295896)

[**2.14 Coding a Complex Subquery 24**](#_Toc508295897)

[**2.14.1 Step 1 - Code Inner Subquery – Step 1 24**](#_Toc508295898)

[**2.14.2 Step 2 - Code a Separate Outer Query 25**](#_Toc508295899)

[**2.14.3 Step 3 - Combine Outer Query with Inner Subquery 26**](#_Toc508295900)

[**2.15 Apply Inner Join between Packing List and Drawing with Job Cost Subquery 26**](#_Toc508295901)

[**3.0 Apply SQL View to PSP Case – Level 3 27**](#_Toc508295902)

[**3.1 SQL Views 28**](#_Toc508295903)

[**3.2 SQL Views to Simplify Complicated SQL statements 29**](#_Toc508295904)

[**3.3 SQL Views to Enhance Security 30**](#_Toc508295905)

[**3.4 SQL Views to Establish Conceptual Independence 31**](#_Toc508295906)

[**3.5 Restrictions on using a VIEW 32**](#_Toc508295907)

[**3.6 WITH CHECK OPTION 32**](#_Toc508295908)

[**3.7 Questions - View 33**](#_Toc508295909)

[**3.8 Apply View to Inner Join of Job Cost and Packing List 34**](#_Toc508295910)

[**3.9 Apply View to Inner Join between Customer, Drawing and Packing List 35**](#_Toc508295911)

[**3.10 Apply View to Join between Customer, Drawing, Option and Drawing Option 36**](#_Toc508295912)

[**3.11 Materialized View Concepts 37**](#_Toc508295913)

[**3.11.1 Questions - Materialized View Concepts 37**](#_Toc508295914)

[**4.0 Apply PL/SQL to the PSP Case – Level 4 38**](#_Toc508295915)

[**4.1 Duplicate Customer Name Stored Procedure 38**](#_Toc508295916)

[**4.2 Customer Add with duplicate name validation Stored Procedure 39**](#_Toc508295917)

[**4.3 Customer Drawing List Stored Procedure 39**](#_Toc508295918)

**Enter your Name Here 🡺 Puja Ghosh**

# Introduction

## How to complete Hand-on Tutorial Requirements and Review Questions

This tutorial will provide directions and demonstration examples to guide the student to perform a hands-on requirement. Using these tutorial demonstration examples the student will be required to perform a similar hands-on task. The following is an example of typical tutorial hands-on requirement.

**Use a graphical snipping tool to document your successful logon with a display of your ISPF Primary Option Menu below.**

The directions specify the use of a graphical snipping tool, such as the Window's Snipping Tool. (Directions to use the Window's Snipping Tool will be presented next.) Any graphic snipping tool may be used to demonstrate that you have successfully completed the required hands-on task.

A grey or colored box will be provided after the requirement directions. You are required to provide a graphic image, e.g., using cut-and-paste, to document that the hands-on requirement was correctly completed.

Review questions are also be provided at the end of the tutorial. The following is an example of a review question format. Since type the answer in provided grey or colored box.

1. What is the purpose of a partitioned data set? Answer:

Type in the answer to the question into the grey or colored box.

**It is recommended that you use Table of Contents at the beginning of the tutorial to review and navigate to the concept presented in the review question. Students will find that using the document FIND tool or searching GOOGLE may also be valuable for researching the review question answer.**

## Oracle Error Codes Summary

**Important!**

The following error codes are discussed at least once in this assignment.

**ORA-00904 - Error Invalid Column Name**

**ORA-00907 - Error Missing Right Parenthesis**

**ORA-00922 - Error Missing or Invalid Option**

**ORA-00942 - Error Table or View Does Not Exist**

**ORA-00955 - Name Is Already Used by Existing Object**

**ORA-00957 - Error Duplicate Column Name**

**ORA-02267 - Error Column Type Incompatible (Foreign Key Relationship)**

**ORA-02449 - Error Unique/Primary Keys in Table Referenced by Foreign Keys**

## Documentation Requirements

The following are requirements to code, execute, test and provide documentation of SQL statement

If you encounter a problem coding of a complete SQL statement that meets all requirements, then code a partial SQL statement as close to the requirement objectives as possible. Don’t leave the requirement blank. Change your Requirement header to reflect the partial SQL statement. If you are close enough I will give you credit.

**1. Using a Snipping Tool document the code of required SQL statement –** properly formatted.

**2. Using a Snipping Tool document the Results of the execution of the SQL Statement – properly formatted**

# Required Professional Appearance of Outputted Results

While you may have coded a SQL statement that meets the requirement objectives, your overall grade will be slightly decreased if your documented results are not professional in appearance. You are required to demonstrate your ability to apply SQL\*Plus line and page formatting, PuTTY formatting, and SQL formatting, e.g., SQL SUBSTR() and TO-CHAR() Function and Columns Aliases (AS).

This does not mean that you should waste excessive time formatting and not completing the assignment. While you will not be penalized for each and every improperly formatted result requirement, the bottom line is if the instructor cannot read your SQL statement and printed result you will receive NO CREDIT.

On the other hand, you may ask, "How picky will the instructor in grading for professional appearance?" Properly applying basic SQL\*Plus, Putty, and SPL formatting is important to some degree and the instructor will not "nit pick" if you have demonstrated that knowledge. The more important issue is what do you have to show a potential employer that you can?

"Design, implement, insert test data, and test a simple academic database and manufacturing cost accounting data base". (Sound like a good line to add to your resume!)

Better yet. Bind all of your final professionally prepared copies of this course assignments into a folder and take it to your internship of future job interview. It does not matter whether or not that the interview involves a database. What matters is that you can professionally apply and document any type of computer technology if given the opportunity. Don't just prepare these course assignments for a final grade, prepare them for a better and financially rewarding career opportunity.

## Improving the Professional Appearance of the SQL Statement and Results using SQL\*Plus

The SQL\*Plus default LINESIZE is 80 characters. Your output line may exceed 80 characters and then wrap to the next line this making your documentation unreadable. Increase the LINESIZE at the SQL> prompt or by adding the following in your script, e.g., SET LINESIZE 130 [this is a SQL\*Plus command, not a SQL statement, and should not end with a semicolon

Setting the LINESIZE does not affect the font size in Putty. By default SQL\*Plus display 80 characters on a line not matter what the font size is in Putty. SQL\*Plus will force the display of the result of the SELECT statement is wrap to the next line at 80 characters. Increasing the SQL\*Plus LINESIZE may still be wrapped by Putty. You can decrease the font size in Putty.

## Improving the Professional Appearance of the SQL Statement and Results using PuTTY

**Change the point size (or font) under Window/Appearance**. Click CHANGE under Font Settings. A point size of 14 that is bolded is easier to read, but you may get word wrap problems. Clicking on OK only affects the current session.



**Video - Formatting SQL\*Plus Output -** [**https://www.youtube.com/watch?v=\_46MeUAFDO4**](https://www.youtube.com/watch?v=_46MeUAFDO4)

**Video - How-to: Change the PAGESIZE and LINESIZE in SQL\*Plus -** [**https://www.youtube.com/watch?v=hSQFcRt8kqI**](https://www.youtube.com/watch?v=hSQFcRt8kqI)

## Improving the Professional Appearance of the Results using the SQL SUBSTR() and TO-CHAR() Function and Columns Aliases (AS)

Similar to the previous CREATE TABLEs, start simple and then add more features.

**Step 1.** Start by listing columns those columns that are to be displayed and FROM which table stores those columns?

Use the SQL Script method or type the following example at the SQL> prompt and execute.

**SELECT CUSTOMER\_NAME,**

**CUSTOMER\_NUMBER,**

**CUSTOMER\_TYPE,**

**ATTENTION\_NAME**

**FROM CUSTOMER;**

**SELECT JOBCOST\_REFERENCE\_NUMBER,**

**PACKLIST\_NUMBER,**

**TRANSACTION\_DATE,**

**TRANSACTION CODE,**

**JOBCOST\_UNITS \* COST\_PER\_UNIT**

**FROM JOB\_COST;**

**Using a Snipping Tool document Your Results**

Notice that each previous example coded each column on a separate line and a little indentation was used to mage the SQL code easier to read.

## SUBSSTR() Function

**Step 2.** Format the column width using the SUBTR() function. Students frequently make column names or the width of the CHAR data types to wide. For example, you may have decided to create the CUSTOMER\_NAME column as CHAR(25). Depending on the business requirements a column width of 25 characters may be acceptable or even recommended. But, if you display five columns, each with a column width of 25 characters, then the line will be a least 125 characters wide and the white space will make the output unreadable.

Therefore, it is recommended to use the SUBSTR() function to reduce the number of characters displayed, which will have no effect on the number of characters stored. see (<http://www.techonthenet.com/oracle/functions/substr.php>)

Video - Oracle Database11g tutorials 13 || SQL substr function / SQL substring function - <https://www.youtube.com/watch?v=f52uKYyYFoU>

Use the SQL Script method or type the following example at the SQL> prompt and execute.

SELECT **SUBSTR(CUSTOMER\_NAME, 1, 10),**

CUSTOMER\_NUMBER,

CUSTOMER\_TYPE,

**SUBSTR(ATTENTION\_NAME, 1, 10)**

FROM CUSTOMER;

SELECT **SUBSTR(JOBCOST\_REFERENCE\_NUMBER, 1, 6),**

**SUBSTR(PACKLIST\_NUMBER, 1, 6),**

**SUBSTR(TRANSACTION\_DATE, 1, 10),**

**SUBSTR(TRANSACTION CODE, 1, 2),**

JOBCOST\_UNITS \* COST\_PER\_UNIT

FROM JOB\_COST;

**Using a Snipping Tool document Your Results**

## AS (Column Alias)

**Step 3.** Provide a Column Alias to make the column header more meaning full. By default Oracle uses the column name as the displayed column header. When we use the SUBSTR() function we may truncate the column header. A column alias may be appropriate. A column alias may be appropriate to may a column header of a calculation more descriptive. A column alias uses the AS clause. Do not get a column alias confused with a table alias. Notice that a column alias is not permitted to have embedded spaces UNLESS it is enclosed in quotes (single or double). You may use upper or lower case letters depending on the professional appearance desired. SQL Aliases - <http://www.w3schools.com/sql/sql_alias.asp>

Using Column Alias in SELECT Statement - <http://www.geeksengine.com/database/basic-select/column-alias.php>

Use the SQL Script method or type the following example at the SQL> prompt and execute.

SELECT SUBSTR(CUSTOMER\_NAME, 1, 10) **AS CUSTOMER**,

CUSTOMER\_NUMBER ,

CUSTOMER\_TYPE **AS TYPE** ,

SUBSTR(ATTENTION\_NAME, 1, 10) **AS ATT\_NAME**

FROM CUSTOMER;

SELECT SUBSTR(JOBCOST\_REFERENCE\_NUMBER, 1, 6) **AS REF#**,

SUBSTR(PACKLIST\_NUMBER, 1, 6) **AS PL#** ,

SUBSTR(TRANSACTION\_DATE, 1, 10) **AS "TRAN DATE"** ,

SUBSTR(TRANSACTION CODE, 1, 2) **AS TC**,

JOBCOST\_UNITS \* COST\_PER\_UNIT **AS "LINE COST"**

FROM JOB\_COST;

**Using a Snipping Tool document Your Results**

## TO\_CHAR() Function

**Step 4.** Use the TO\_CHAR() function to format date and numeric values as appropriate.

TO\_CHAR Function http://www.techonthenet.com/oracle/functions/to\_char.php

TO\_CHAR http://www.java2s.com/Tutorial/Oracle/0300\_\_Conversion-Functions/0140\_\_TO\_CHAR.htm

TO\_CHAR - Convert Datetime to String - Oracle to SQL Server Migration - http://www.sqlines.com/oracle-to-sql-server/to\_char\_datetime

Video - TO\_CHAR(datetime) Function in SQL Query - <https://www.youtube.com/watch?v=00VT7Sgo5ig>

Vide - Concatenation Operator, To\_Char Currency, Money Symbol in Oracle SQL - <https://www.youtube.com/watch?v=9ePcZrzQcYQ>

Video - Oracle TO\_CHAR Function- <https://www.youtube.com/watch?v=X-uQDdmORvI>

Video - TO\_CHAR(number) Function in SQL Query- <https://www.youtube.com/watch?v=fuXkFmD1FKY>

Use the SQL Script method or type the following example at the SQL> prompt and execute.

SELECT SUBSTR(JOBCOST\_REFERENCE\_NUMBER, 1, 6) AS REF#,

SUBSTR(PACKLIST\_NUMBER, 1, 6) AS PL# ,

**TO\_CHAR(**SUBSTR(TRANSACTION\_DATE, 1, 10) **, 'DD/MM/YYYY' )** AS "TRAN DATE" ,

SUBSTR(TRANSACTION CODE, 1, 2) AS TC,

**TO\_CHAR**( (JOBCOST\_UNITS \* COST\_PER\_UNIT) **, '9999.99')** AS "LINE COST"

FROM JOB\_COST;

**Using a Snipping Tool document Your Results**

**Support Videos**

Oracle Express is used in these videos. You are not required to use Oracle Express. It does not matter if you are using the SQL\*Plus client or Oracle Express. The SQL coding concepts are the same.

**Introduction to SQL - SQL LESSON 1**

**http://www.youtube.com/watch?v=UeJKioNqe5w&feature=relmfu**

**Express Tour of SQL - PART 1 - SQL LESSON 2**

**http://www.youtube.com/watch?v=HOrpREPUs2o&feature=relmfu**

**Express Tour of SQL - PART 2 - SQL LESSON 3**

**http://www.youtube.com/watch?v=5bcLy2PT8Fk&feature=relmfu**

**Installing Oracle Express - PART 1 - SQL LESSON 4**

**http://www.youtube.com/watch?v=4EwvoCYlGmo&feature=relmfu**

**Installing Oracle Express - PART 2 - SQL LESSON 5**

**http://www.youtube.com/watch?v=0TVwQoU8vU4&feature=relmfu**

**SQL Conditions, Expressions, and Operators - PART 1 - SQL LESSON 6 - http://www.youtube.com/watch?v=lPzW1H1nbaE&feature=relmfu**

**SQL Conditions, Expressions, and Operators - PART 2 - SQL LESSON 7**

**http://www.youtube.com/watch?v=3mnc7LROq-0&feature=relmfu**

**SQL Conditions, Expressions, and Operators - PART 3 - SQL LESSON 8 -**

**http://www.youtube.com/watch?v=zUAE2Ao1m-M&feature=relmfu**

**SQL Conditions, Expressions, and Operators - PART 4 - SQL LESSON 9 - http://www.youtube.com/watch?v=RzS7eDHfGZk&feature=relmfu**

**SQL Logical Operators - PART 1 - SQL LESSON 10 -**

**http://www.youtube.com/watch?v=jhSm8tFvWSI&feature=relmfu**

**SQL Logical Operators - PART 2 - SQL LESSON 11 -**

**http://www.youtube.com/watch?v=oVF1s8WiDDk&feature=fvwrel**

**SQL Logical Operators - PART 3 - SQL LESSON 12 -**

**http://www.youtube.com/watch?v=btZ39GrWPuQ&feature=relmfu**

**SQL Multiple Logical Operators - PART 1 - SQL LESSON 13 -**

**http://www.youtube.com/watch?v=1FNlnqjW2vk&feature=relmfu**

**SQL Multiple Logical Operators - PART 2 - SQL LESSON 14 -**

**http://www.youtube.com/watch?v=gaISO0Xdpi4&feature=relmfu**

**SQL Multiple Logical Operators - PART 3 - SQL LESSON 15**

**http://www.youtube.com/watch?v=yZbml5Kga0I&feature=relmfu**

**Using Brackets in SQL to Simplify the Where Clause - PART 1 - SQL LESSON 15**

**http://www.youtube.com/watch?v=k29P\_8zSD7w&feature=relmfu**

**Using Brackets in SQL to Simplify the Where Clause - PART 2 - SQL LESSON 16**

**http://www.youtube.com/watch?v=0U5DgEV0S0Q&feature=relmfu**

**Equality and Inequality Conditions in SQL Where Clause - SQL LESSON 17**

**http://www.youtube.com/watch?v=MmzZDq3HGzw&feature=relmfu**

**Membership Conditions in SQL Where Clause (Subqueries) - Part 1 - SQL LESSON 18 -**

**http://www.youtube.com/watch?v=7MKQGXcw1aw&feature=relmfu**

**Membership Conditions in SQL Where Clause (Subqueries) - Part 2 - SQL LESSON 19 - http://www.youtube.com/watch?v=\_X1oiNqDVeE&feature=fvwrel**

**Range Conditions in SQL Where Clause (Between) - SQL LESSON 20 -**

**http://www.youtube.com/watch?v=z5YCPGzq-Ac&feature=fvwrel**

**Matching Conditions in SQL Where Clause (IN) - Part 1 - SQL LESSON 21 - http://www.youtube.com/watch?v=MIHA7ptVom0&feature=relmfu**

**Matching Conditions in SQL Where Clause (IN) - Part 2 - SQL LESSON 22 -**

**http://www.youtube.com/watch?v=bK1EEnTFQl0&feature=relmfu**

**SQL Comparison Conditions - SQL LESSON 22 B -**

**http://www.youtube.com/watch?v=iSX4Yb3\_TPY&feature=relmfu**

**Concept of a Null Value in SQL - Part 1 - SQL LESSON 23 -**

**http://www.youtube.com/watch?v=XoH-T\_0Ik9k&feature=relmfu**

**Problems with Null Value in SQL - SQL LESSON 25 -**

**http://www.youtube.com/watch?v=4cFfPaJCkxw&feature=relmfu**

**Sorting in SQL Using Order by Clause - Part 1 - SQL LESSON 27 -**

**http://www.youtube.com/watch?v=0ljgw-EFqJE&feature=relmfu**

**Sorting in SQL Using Order by Clause - Part 2 - SQL LESSON 28**

**http://www.youtube.com/watch?v=PnlgMtMh4i0&feature=channel&list=UL**

**Sorting in Descending Order in SQL Using Order By Desc Clause - SQL LESSON 29**

**http://www.youtube.com/watch?v=kBDIf6fEt-E&feature=channel&list=UL**

# Table Layout for the PSP Case

## CUSTOMER Table (PRIMARY KEY CUSTOMER NUMBER)

**CUSTOMER NAME**

**CUSTOMER NUMBER {common column}**

**CUSTOMER TYPE must be 'B','C','S' (where B means Blanket, C means Contract, S means Standard)**

**ATTENTION NAME**

**DATE ACTIVATED**

**ADDRESS 1 BILL**

**CITY BILL**

**STATE BILL**

**ZIPCODE BILL**

**COUNTRY BILL**

CUSTOMER SALES PERSON NO

PREFERRED SHIP METHOD

CUSTOMER CLASS

F O B (What Does FOB Mean in Shipping? https://www.freightquote.com/blog/what-does-fob-mean-in-freight-shipping)

BLANKET PO

DATE EXPIRE

FREIGHT FLAG

PRICE CHANGE FLAG

DATE ACTIVATED

SIC (Standard Industrial Classification - https://en.wikipedia.org/wiki/Standard\_Industrial\_Classification)

CUSTOMER PRIMARY PHONE NUMBER

CUSTOMER PRIMARY EMAIL CONTACT

**Customer Table Indexes CUSTOMER NAME**

## CUSTOMER SHIPPING ADDRESS TABLE (PRIMARY KEY CUSTOMER SHIPPING ID)

CUSTOMER SHIPPING ID

CUSTOMER NUMBER {common column}

DATE ADDED

DATE MODIFIED

ADDRESS 1 SHIP

ADDRESS 2 SHIP

CITY SHIP

STATE SHIP

ZIPCODE SHIP

COUNTRY SHIP

SHIP TYPE must be 'W', 'M','D', or 'O' where W means Warehouse, M means Manufacturing Plant, D means Drop Ship, and O means Other

Customer Shipping Table Indexes CUSTOMER NUMBER

Customer Shipping Address Table Foreign Keys CUSTOMER NUMBER (delete cascade)

## DRAWING TABLE (PRIMARY KEY DRAWING NUMBER)

**DRAWING NUMBER {common column}**

**DRAWING PREFIX (A prefix categorizes the general type of drawing, e.g., "S" Shaft, "R", Bar "B", etc.)**

**REVISION LEVEL (A revision level specifies the current version of the drawing to be used for the next customer order, which may be different from past revision levels or those on the current schedule.)**

**ORIGINAL DATE ENTERED**

**DATE LAST REVISED**

**CUSTOMER NUMBER**

**CUSTOMER PART NUMBER**

**PRIMARY RAW MATERIAL ID (Every drawing has a primary raw material used to start the packing list production process. It may be a hollow tube, a solid tube, and solid bard, plastic, type of wood, etc. All other production materials charged to the will be called supplies, not raw material.)**

**PPRIMARY RAW MATERIAL UNIT (The primary raw material unit represents how the raw material will be disbursed to the packing list, by tube, by foot, by pound, etc.**

**OD**

**WALL**

**LENGH**

**CUT LENGTH**

**GL\_NUMBER**

**Drawing Table Indexes CUSTOMER NUMBER + CUSTOMER PART NUMBER**

**PRIMARY RAW MATERIAL NUMBER**

**CUSTOMER PART NUMBER**

**Drawing Table Foreign Keys CUSTOMER NUMBER (delete cascade)**

**Note: When a Primary Key, Foreign Key, or Index specifies two columns combined by a plus-sign "+", e.g., column\_1 + column\_2, this is an indication of a composite key.**

## PRICE LIST TABLE (PRIMARY KEY OD + WALL + LENGTH + TYPE)

**OD**

**WALL**

**LENGH**

**TYPE (matches one of the categories of drawing prefixes)**

**PRICE 0 TO 5 (to store**

**PRICE 6 TO 25**

**PRICE 26 TO 50**

**PRICE 50 TO 100**

**PRICE 101 TO 250**

**PRICE OVER 250**

**Comments: This Company was originally started as a customized tubular manufacturer and expanded its product lines into other customized areas. Bob's original pricing formulas and quantity discounts was based on a raw material tube dimensions OD (outside diameter), Wall (the thickness of a tube) and Length of a tube. The type of a tube was contrast non-tube products, e.g., bar product, and howitzer shell casings. I wanted to make primary key more independent of a tubular product line but Bob would not approve it. The only area where the price table was important was to help Tom quote prices and then store this primary key in the Packing List table to document that set of prices used to quote the order.**

## OPTION TABLE (Primary Key OPTION NUMBER)

**OPTION NUMBER {common column}**

**OPTION TYPE must be % or $**

**OPTION DESCRIPITON**

**OPTION UNIT COST**

**OPTION UNIT PRICE**

**OPTION PRECENTAGE**

**Option Table Indexes OPTION DESCRIPTION**

## DRAWING OPTION TABLE (Primary Key DRAWING NUMBER + OPTION NUMBER)

**DRAWING NUMBER {common column}**

**OPTION NUMBER {common column}**

**UNITS-OF-OPTION**

**Drawing Option Table Indexes OPTION NUMBER**

**Drawing Option Table Foreign Keys DRAWING NUMBER (delete cascade)**

**RAW MATERIAL (Primary Key Raw Material ID)**

**RAW MATERIAL ID**

**RAW MATERIAL TYPE must be "T","S" or "O" ("T" means Tubular, "S" Supplies, "O" means Other)**

**MATERIAL DESCIPTION**

**UNIT TYPE**

**CURRENT UNIT COST**

Stores common information for tubular and supplies tables. This is an example of generalization and specialization

## TUBULAR TABLE (Primary Key Raw Material ID)

**RAW MATERIAL ID {common column}**

**OD**

**WALL**

**LENGH**

**CURRENT UNIT COST**

**CURRENT COST PER FOOT**

**MINIMUM DROP LENGTH**

**Tubular Inventory Table Indexes OD + WALL + LENGTH**

**Tubular Inventory Table Foreign Keys RAW MATERIAL ID (delete cascade)**

## SUPPLIES TABLE (Primary Key Raw Material ID)

**RAW MATERIAL ID {common column}**

**CURRENT UNIT COST**

**REORDER POINT**

**SAFETY STOCK**

**Tubular Inventory Table Foreign Keys RAW MATERIAL ID (delete cascade)**

## RAW MATERIAL REQUIREMENTS TABLE (Primary Key PACKLIST NUMBER + RAW MATERIAL ID

**PACKLIST NUMBER {common column}**

**RAW MATERIAL ID**

**SCHEDULED SHIP DATE**

**UNITS REQUIRED**

**Raw Material Table Indexes RAW MATERIAL ID**

**Raw Material Requirements Foreign Keys DRAWING NUMBER (delete cascade)**

(Comments: this table was used to determine future raw material requirements. One row was entered when the PL was recorded. As the raw material was used during the manufacturing process, the units required was changed. When the PL was completed this row was deleted.)

## RAW MATERIAL FIFO TABLE (Primary Key FIFO ID, auto sequence)

FIFO ID

RAW MATERIAL ID

DATE DELIVERED

UNITS ON HAND

UNIT COST

VENDOR NUMBER

MFG CODE

Raw Material Fifo Indexes RAW MATERIAL ID + DATE DELIVERED

Raw Material Fifo Table Foreign Keys RAW MATERTIAL ID (delete cascade)

## RAW MATERIAL DROP INVENTORY (Primary Key DROP REFERENCE NUMBER, auto sequence)

DROP REFERENCE NUMBER **{common column}**

RAW MATERIAL ID

DROP LENGTH

COST PER FOOT

DROP UNITS

Raw Material Drop Inventory Indexes RAW MATERIAL ID + DROP LENGTH

Raw Material Drop Inventory Foreign Keys RAW MATERTIAL ID (delete cascade)

## VENDOR TABLE (Primary Key VENDOR NUMBER auto sequence, auto sequence)

**VENDOR NUMBER {common column}**

**VENDOR NAME**

**Vendor Table Indexes VENDOR NAME**

## INVENTORY VENDOR (Primary Key VENDOR NUMBER + RAW MATERIAL NUMBER

RAW MATERIAL ID **{common column}**

VENDOR NUMBER **{common column}**

MFG CODE

LAST COST PER UNIT

Inventory Vendor Table Indexes VENDOR NUMBER

Inventory Vendor Table Foreign Keys RAW MATERTIAL ID (delete cascade)

VENDOR NUMBER (delete cascade

## LABOR OPERATION TABLE (Primary Key OPERATION NUMBER)

**OPERATION NUMBER {common column}**

**OPERATION DESCRIPTION**

**SCHEDULE CATAGORY must be 0 thru 20**

**Labor Operation Table Indexes OPTION DESCRIPTON**

**(Comments: This table was used to the type of labor operation used for labor costs in the Job Cost table. Also the type of labor operation was recorded in the Time Card. The Time Card table was used by applications to determine the job progress of a Packing List. Most labor operations were assigned a schedule category from a number 0 (no job progress) to 20 (ready for delivery). The schedule category roughly control the sequence of a packing list as it moved through production. Several labor operations could be assigned schedule category 1, i.e., cutting. If there were 20 units ordered on a Packing List and Tom saw 20 units cut on the job progress report, he would know that cutting step had been completed. It didn't matter which type of cutting was performed. Not all packing lists used all schedule category steps.)**

## PACKLIST TABLE (Primary key PACKLIST NUMBER)

**PACKLIST NUMBER {common column}**

**DRAWING NUMBER {common column}**

**REVISION LEVEL**

**CUSTOMER NUMBER {common column}**

**CUSTOMER PART NUMBER**

**CUSTOMER PURCHASE ORDER NUMBER**

**ORDER BY**

**DATE ORDERED**

**SCHEDULED SHIP DATE**

**ORGIINAL SHIP DATE**

**ACTUAL SHIPPED DATE**

**ORDER ACKNOWLEDGENT DATE**

**UNITS**

**PRICE PER UNIT**

**GROUP BY**

**QUANTITY SHIPPED**

**CUSTOMER SHIPPING ID**

**SATISFIED**

**PACKLIST Table Indexes DRAWING NUMBER**

**CUSTOMER PURCHASE ORDER NUMBER**

**CUSTOMER NUMBER + CUSTOMER PURCHASE ORDER NUMBER**

**CUSTOMER PART NUMBER**

**SCHEDULED SHIP DATE**

## JOBCOST TABLE (Primary Key JOB COST REFERENCE NUMBER, auto sequence)

**JOBCOST REFERENCE NUMBER**

**PACKLIST NUMBER {common column}**

**TRANSACTION DATE**

**TRANSACTION CODE must be "R", "S", "D", "L", "O", "C", "U", where**

**"R" means Raw Materials,**

**"S" means Supplies,**

**"D" means Drops,**

**"L" means Regular Labor**

**"O" means Overtime Labor**

**"C" means Outside Contract**

**"U" means Undefined**

**TRANSACTION CROSS REFERENCE**

**"R" will store Raw Material ID**

**"S" will store Supply ID**

**"D" will store Drop Reference Number**

**"L" will store Labor Operation**

**"O" will store Labor Operation**

**"C" will store AP Invoice Number**

**"U" means Undefined**

**CLOCK NUMBER**

**JOBCOST UNITS**

**COST PER UNIT**

**Jobcost Table Indexes PACKLIST NUMBER + TRANSACTION DATE**

**TRANSACTION CODE + TRANSACTION CROSS REFERENCE**

**TRANSACTION DATE**

**When the Job cost type is a labor type, you should have inserted one of your labor operation numbers into the cross reference number. You can insert a new Job COST row or alter a row in the Job Cost table. Likewise the cross reference number should have been a raw material number for material transactions.**

## EMPLOYEE TABLE (Primary Key Clock Number)

SOCIAL SECURITY NUMBER

CLOCK NUMBER **{common column}**

EMPLOYEE LAST NAME

EMPLOYEE FIRST NAME

EMPLOYEE MIDDLE INITIAL

HOURLY RATE

BONUS CHARGE must be "Y" or "N"

Employee Table Indexes SOCIAL SECURITY NUMBER

EMPLOYEE LAST NAME

## TIME CARD TABLE (Primary Key TIME CARD REFERENCE NUMBER, auto sequence)

TIME CARD REFERENCE NUMBER **{common column}**

CLOCK NUMBER

TIME CARD DATE

SHIFT NUMBER must be 1, 2 or 3

PACK LIST NUMBER **{common column}**

OPERATION NUMBER

HOURS must be 1 to 24

MINUTES must be 0, 15, 30 or 45

GOOD PIECES

BAD PIECES

REWORED PIECES

OVER TIME

HOURLY RATE

Time Card Table Indexes CLOCK NUMBER + TIME CARD DATE

(This index was used to determine the amount and type of work performed by an employee for that day's shift.)

TIME CARD DATE + SHIFT NUMBER

PACKLIST NUMBER, OPERATION NUMBER + TIME CARD DATE

(This index was used by job progress applications)

The employee would enter his work in a time card application program. Over time this application was modernized to include bar codes and scanning. An employee could work on several packing list per shift. It the employee worked on five packing lists during that shift then five rows would enter five lines or rows in the time card table.) The employee could update the time card at any time until the supervisor finalized the entries.

When the employee entered a row on his time card, a job cost row would automatically be entered into job cost table by the time card application. When the time card was finalized, a payroll row would automatically be entered into the payroll table (not relevant in this case.)

Since changes may be made both before and after finalizing, the update application was very sophisticated. Since the system was real-time, all previously entries in all tables must be erased or changed and the changes then recorded.

Using the schedule category of the labor operation, the time card table successfully provided detail data to watch job as it progressed through the shop. The bad and reworked pieces provided an excellent opportunity to manage quality control.

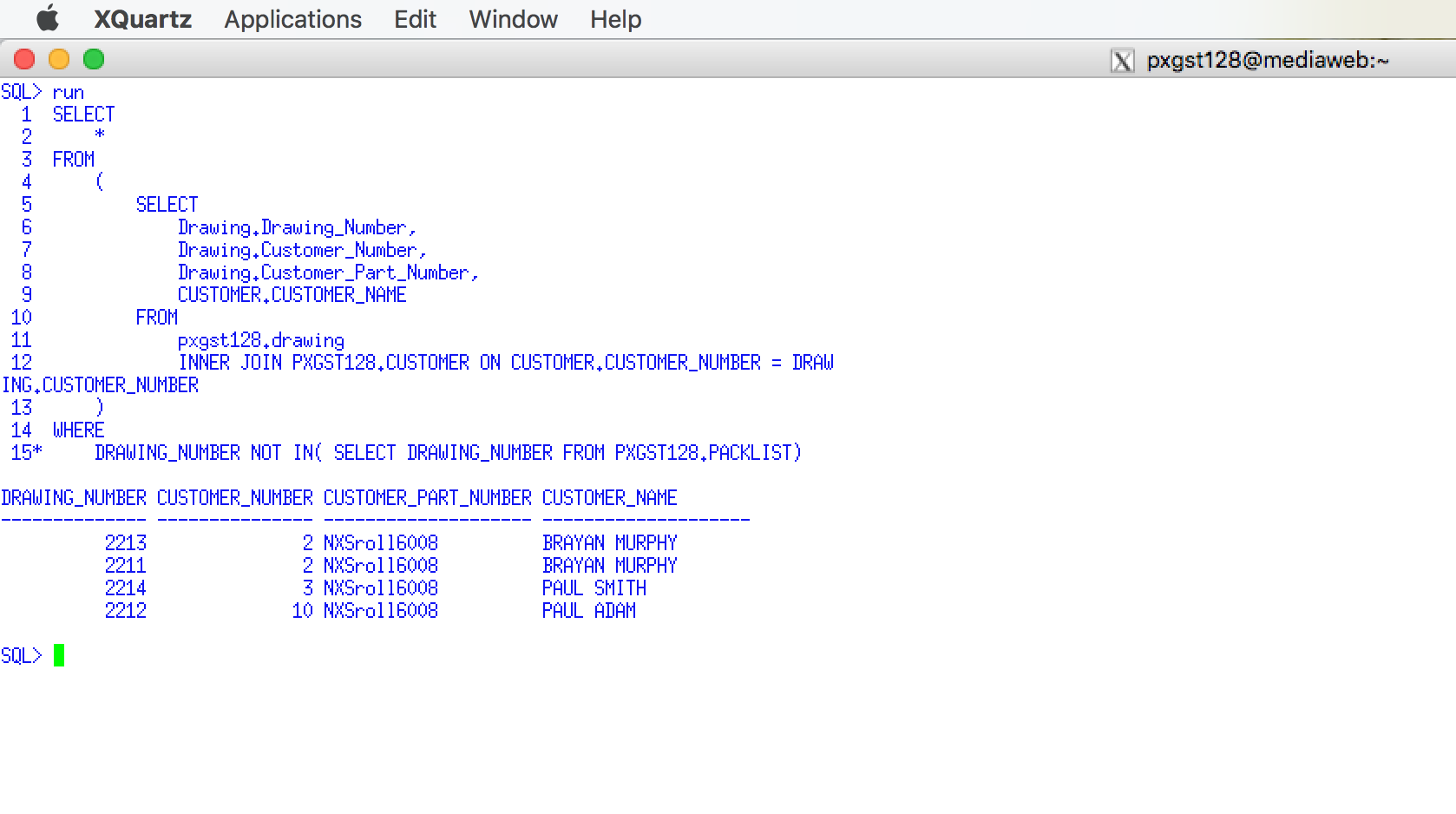
# 2.0 PSP Level 2 Queries

## 2.1 Apply Customer and Drawing Join with Packing List Subquery

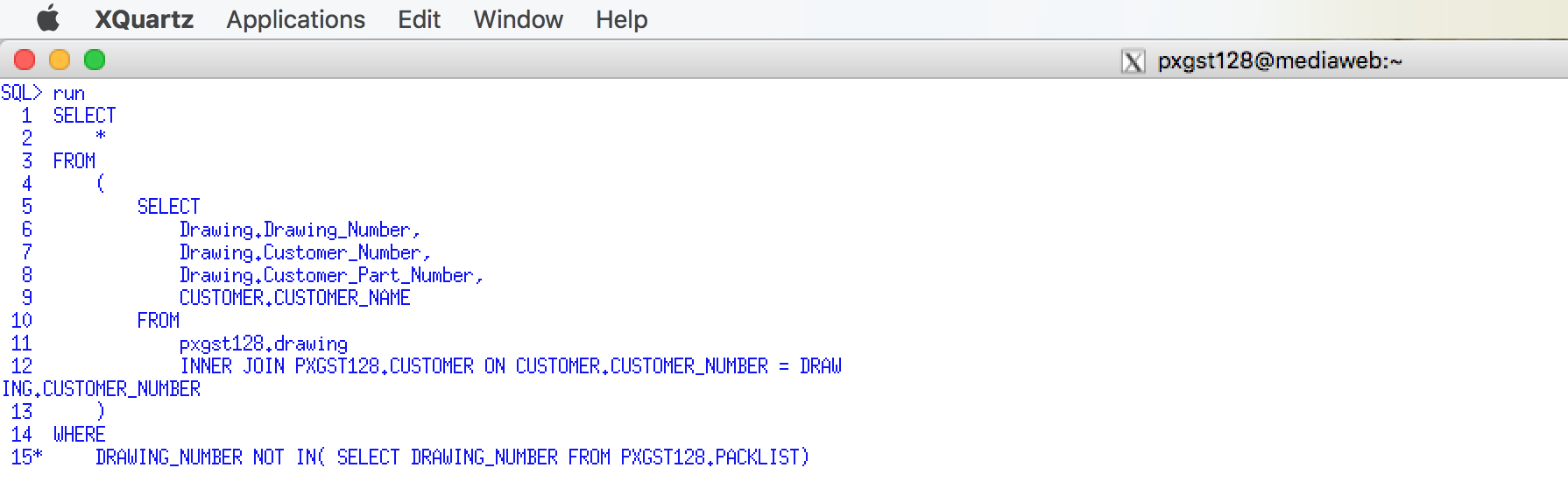
Display those drawing numbers, customer names, and customer part numbers for those drawings that are NOT in the packing list files in drawing number order. If the drawing is not in the Packing List table it has not been ordered yet. One should see at least one drawing that has not be order yet. Hint: This is a subquery using the NOT IN operator.

**Use a graphical snipping tool to document an image of these requirements below.**

Use a Snippit to document your SQL output =>



Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>



SELECT

\*

FROM

(

SELECT

Drawing.Drawing\_Number,

Drawing.Customer\_Number,

Drawing.Customer\_Part\_Number,

CUSTOMER.CUSTOMER\_NAME

FROM

pxgst128.drawing

INNER JOIN PXGST128.CUSTOMER ON CUSTOMER.CUSTOMER\_NUMBER = DRAWING.CUSTOMER\_NUMBER

)

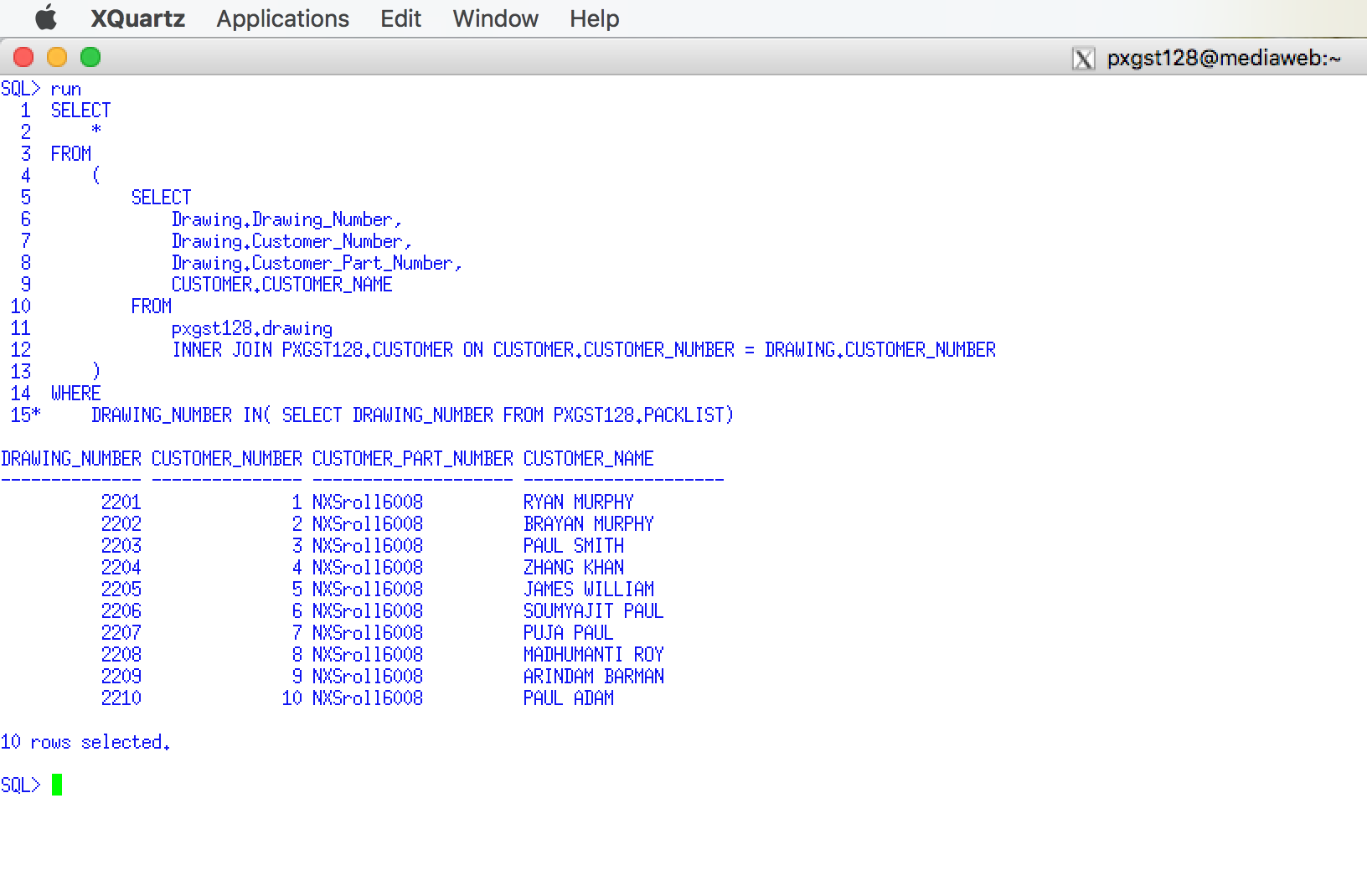
WHERE

DRAWING\_NUMBER NOT IN( SELECT DRAWING\_NUMBER FROM PXGST128.PACKLIST);

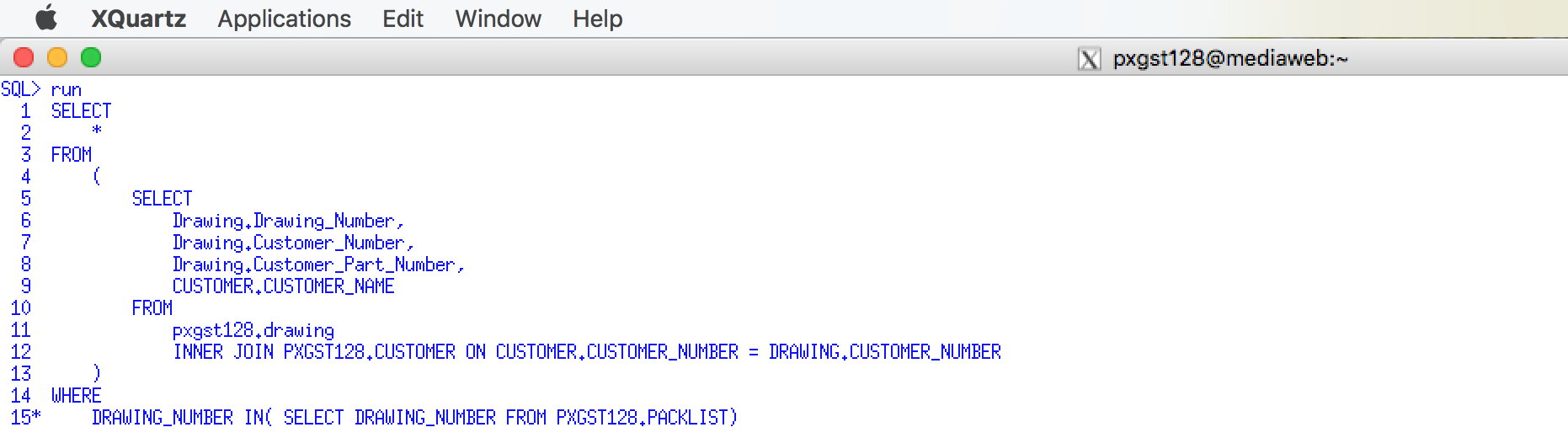
## 2.2 Apply Customer and Drawing Join with Packing List Subquery

Display those drawing numbers, customer names, and customer part numbers for those drawings that ARE in the packing list files in drawing number order. Hint: This is a subquery using the IN operator

Use a Snippit to document your SQL output =>



Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>



SELECT

\*

FROM

(

SELECT

Drawing.Drawing\_Number,

Drawing.Customer\_Number,

Drawing.Customer\_Part\_Number,

CUSTOMER.CUSTOMER\_NAME

FROM

pxgst128.drawing

INNER JOIN PXGST128.CUSTOMER ON CUSTOMER.CUSTOMER\_NUMBER = DRAWING.CUSTOMER\_NUMBER

)

WHERE

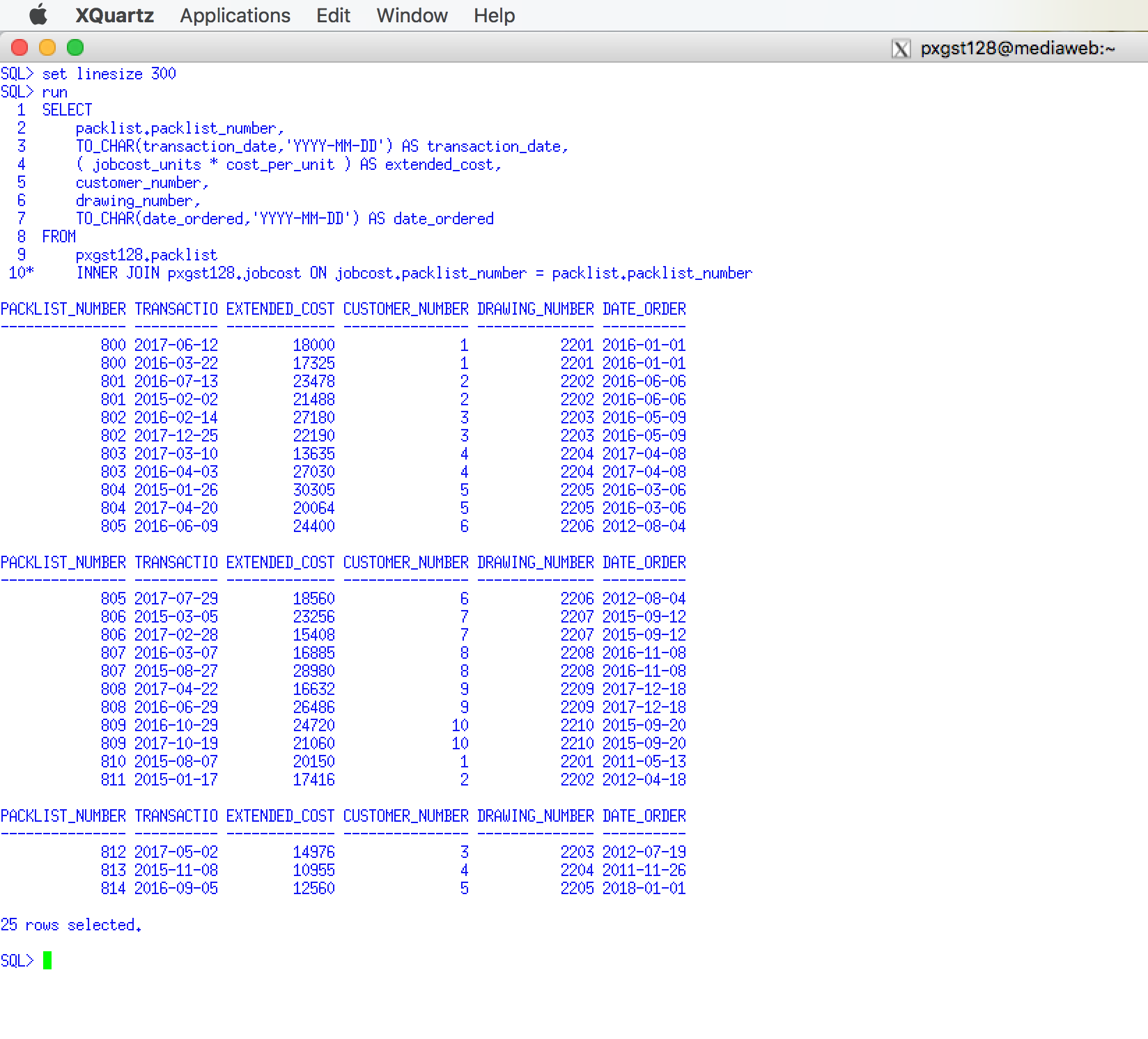
DRAWING\_NUMBER IN( SELECT DRAWING\_NUMBER FROM PXGST128.PACKLIST);

**Use a graphical snipping tool to document an image of these requirements below.**

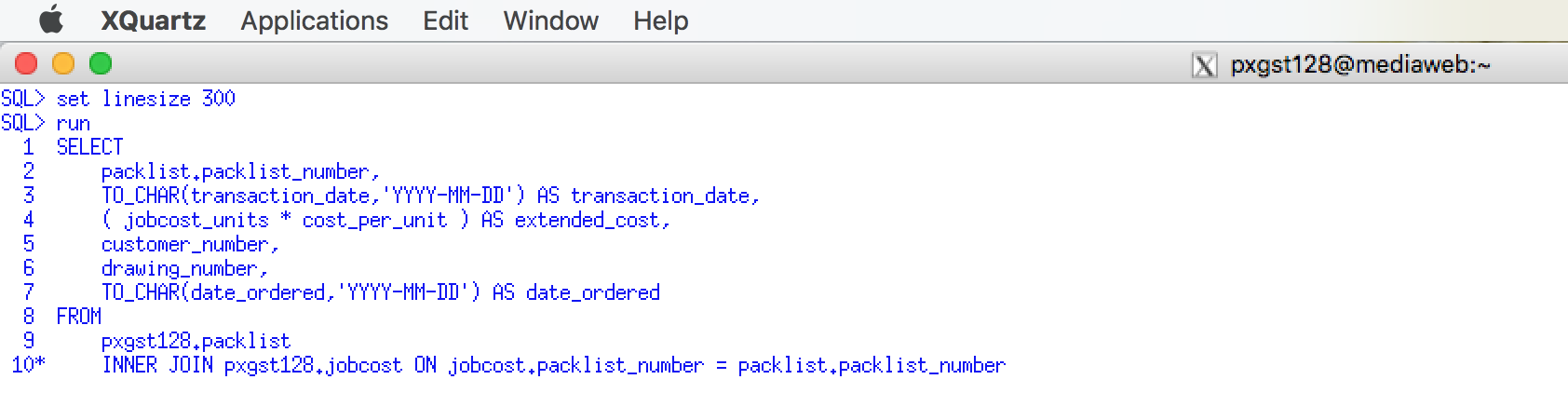
## 2.3 Apply an Inner Join between Job Cost and Packing List with CHAR()

Code a join between the job cost table and packing list table that will display the Packing list number, Transaction date, extended cost (unit cost times transaction units), and any other three different columns from the Packing list table. Use the CHAR() function to format the date.

Use a Snippit to document your SQL output =>



Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>



SELECT

packlist.packlist\_number,

TO\_CHAR(transaction\_date,'YYYY-MM-DD') AS transaction\_date,

( jobcost\_units \* cost\_per\_unit ) AS extended\_cost,

customer\_number,

drawing\_number,

TO\_CHAR(date\_ordered,'YYYY-MM-DD') AS date\_ordered

FROM

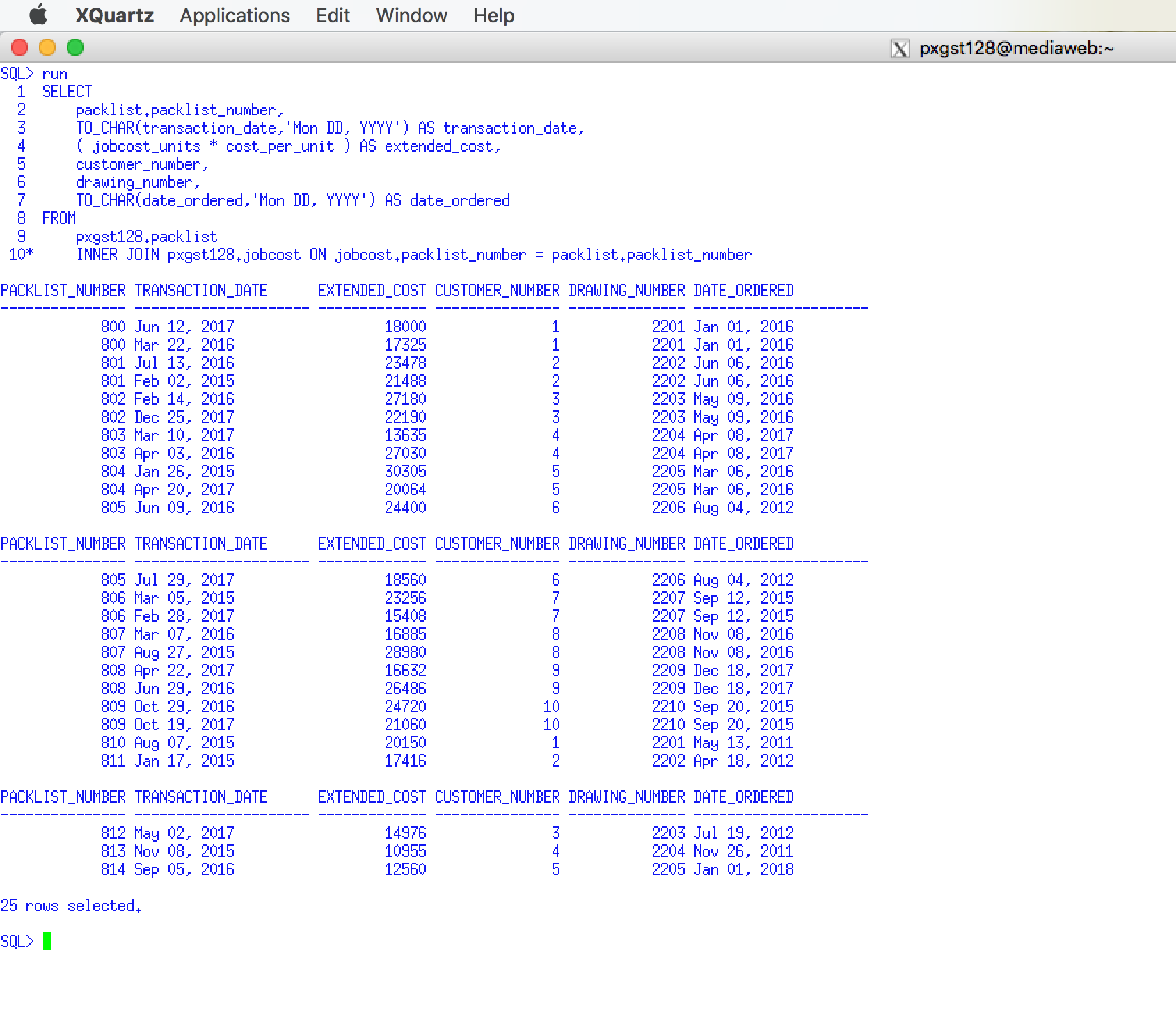
pxgst128.packlist

INNER JOIN pxgst128.jobcost ON jobcost.packlist\_number = packlist.packlist\_number;

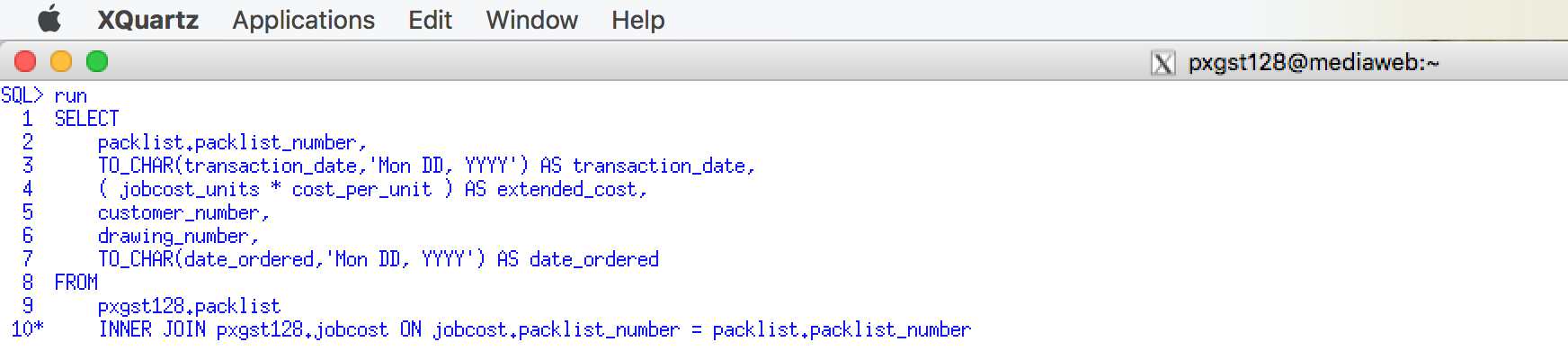
## 2.4 Apply an Inner Join between Job Cost and Packing List with CHAR()

This is a repeat and modification of the previous requirement. Code a join between the job cost table and packing list table create that will display the Packing list number, Transaction date, extended cost (unit cost times transaction units), and any other three different columns from the Packing list table. ***Use a different CHAR() function to format the date.***

Use a Snippit to document your SQL output =>



Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>



SELECT

packlist.packlist\_number,

TO\_CHAR(transaction\_date,'Mon DD, YYYY') AS transaction\_date,

( jobcost\_units \* cost\_per\_unit ) AS extended\_cost,

customer\_number,

drawing\_number,

TO\_CHAR(date\_ordered,'Mon DD, YYYY') AS date\_ordered

FROM

pxgst128.packlist

INNER JOIN pxgst128.jobcost ON jobcost.packlist\_number = packlist.packlist\_number;

## 2.5 Apply an Inner Join between Job Cost and Packing List with CHAR()

This is a repeat and modification of the previous requirement Using a join between the job cost table and packing list table that will display the Packing list number, Transaction date, extended cost (unit cost times transaction units), and any other three different columns from the Packing list table. ***Use a different CHAR() function to format the time.*** You must have inserted the time data to make this work.

Use a Snippit to document your SQL output =>

Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>

## 2.6 COPYING TABLES

There is a frequent need to copy data from one table into a second table. Suppose that one wants to copy student information from the STUDENT table into CIS\_STUDENT table for CIS students. One select statement will create the table and copy the data from the old table to the new table.

**Example 62 CREATE TABLE CIS\_STUDENT**

**AS**

**(SELECT SNO, SNAME, SMAJOR FROM STUDENT**

**WHERE SMAJOR = ‘CIS’);**

One can also create a new empty table by following the following approach. Since 1 will never equal 2 no data will be copied.

**Example 63 CREATE TABLE CIS\_STUDENT**

**AS**

**(SELECT SNO, SNAME, SMAJOR FROM STUDENT**

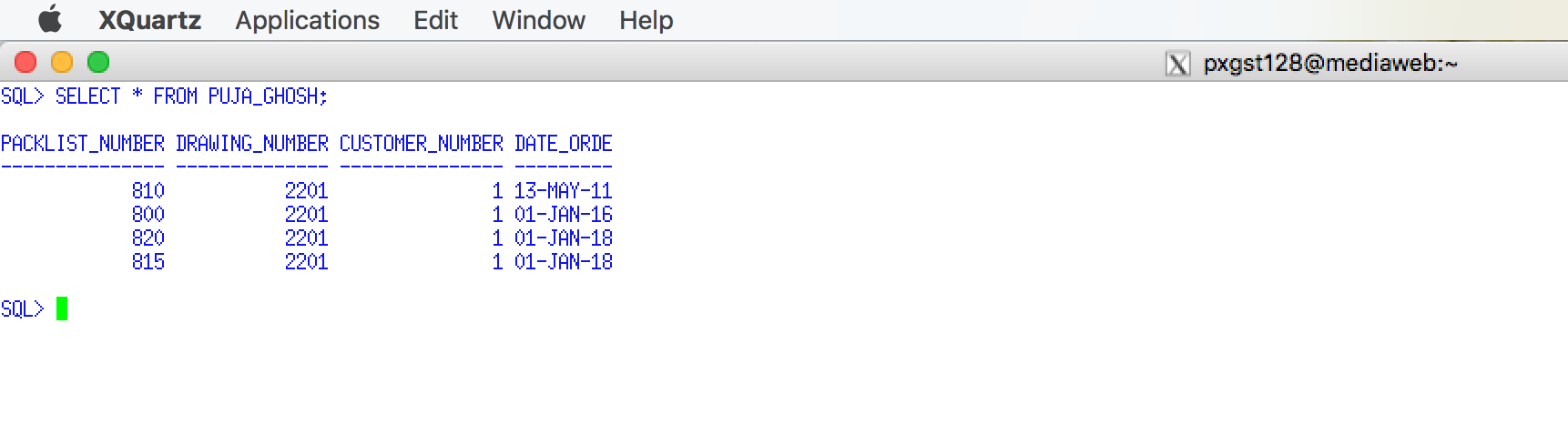
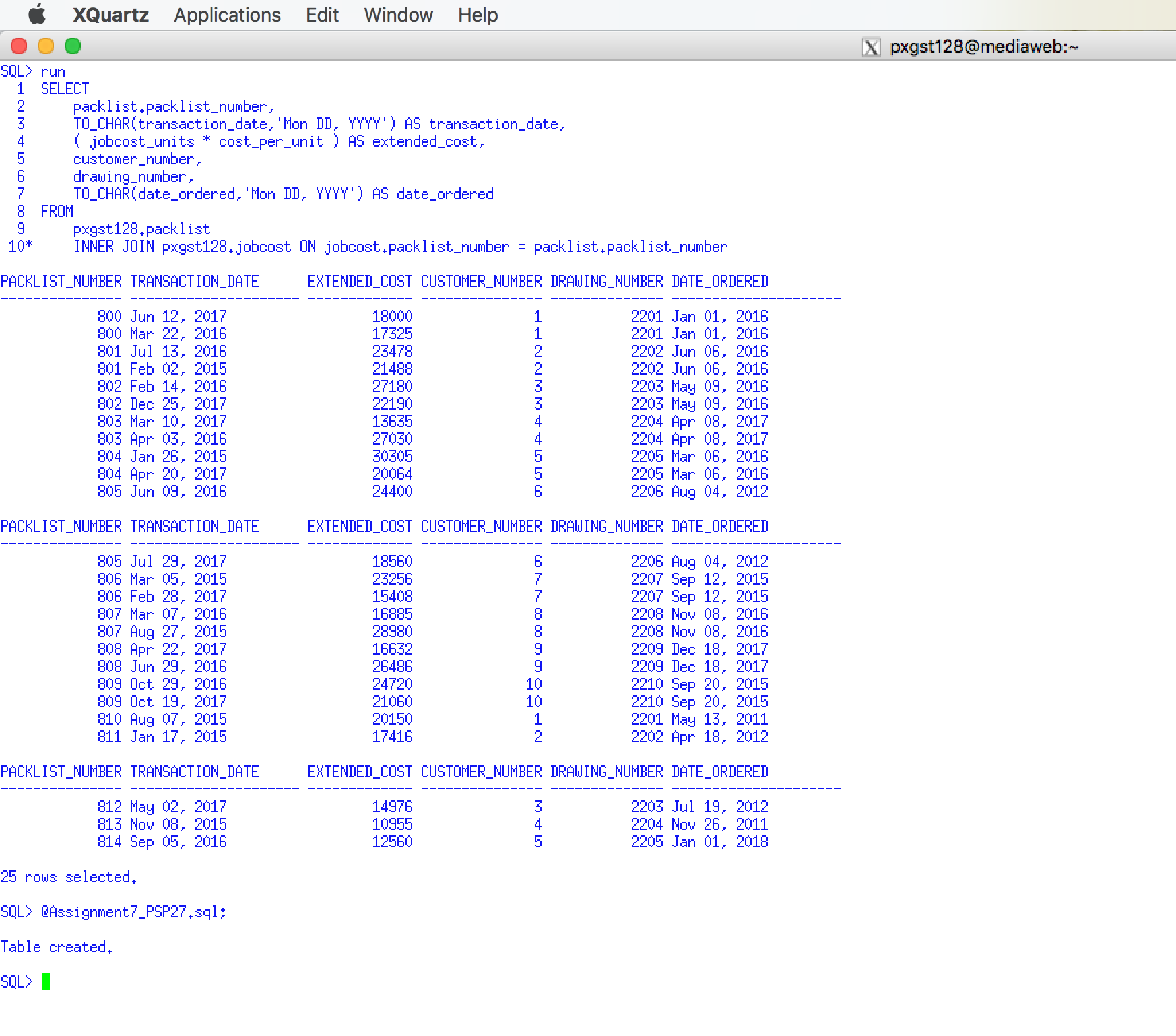
**WHERE 1 = 2);**

## 2.7 Copy Packing List Table

Code a CREATE TABLE .. AS to copy the original packing list table to a new table named using YOUR-NAME**, only for those packing list numbers who are for one customer number who four or more packing lists for any customer drawing. If you do not have a four records for a single customer, insert additional packing list to satisfy this requirement. (See Example 62. )**

**Use a graphical snipping tool to document an image of these requirements below.**

Use a Snippit to document your SQL output =>

Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>

CREATE TABLE PUJA\_GHOSH

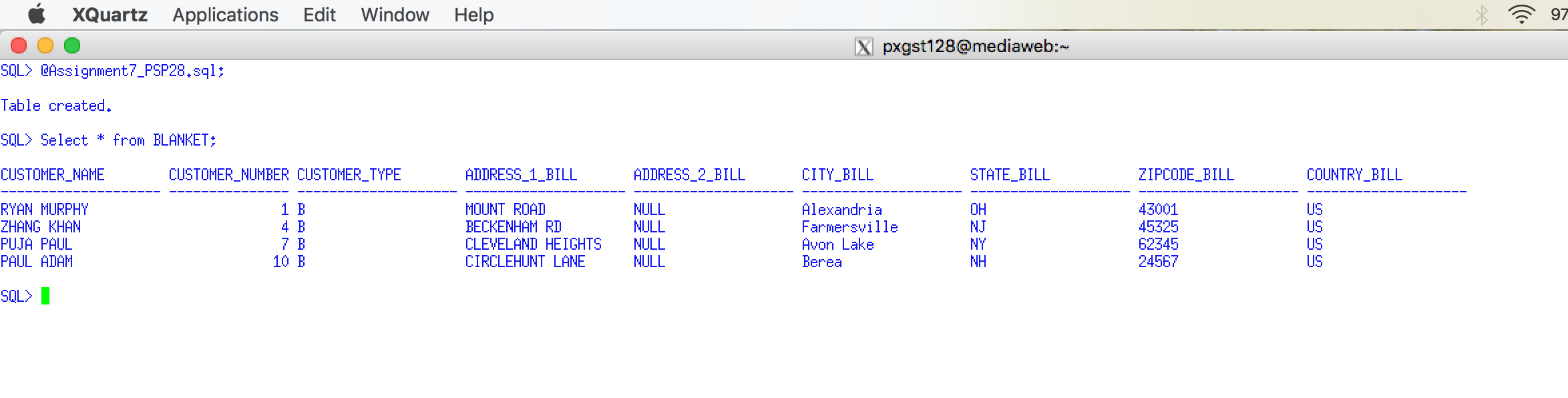
AS (SELECT PACKLIST\_NUMBER,DRAWING\_NUMBER,CUSTOMER\_NUMBER,DATE\_ORDERED FROM PXGST128.PACKLIST

WHERE CUSTOMER\_NUMBER = 1);

## 2.8 Copy Customer List Table

Code Using a Create Table as to a new table names BLANKET which only copies blanket customer's rows.

Use a Snippit to document your SQL output =>



Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>

CREATE TABLE BLANKET

AS (SELECT CUSTOMER\_NAME,CUSTOMER\_NUMBER,CUSTOMER\_TYPE,ADDRESS\_1\_BILL,

ADDRESS\_2\_BILL,CITY\_BILL,STATE\_BILL,ZIPCODE\_BILL,COUNTRY\_BILL FROM PXGST128.CUSTOMER

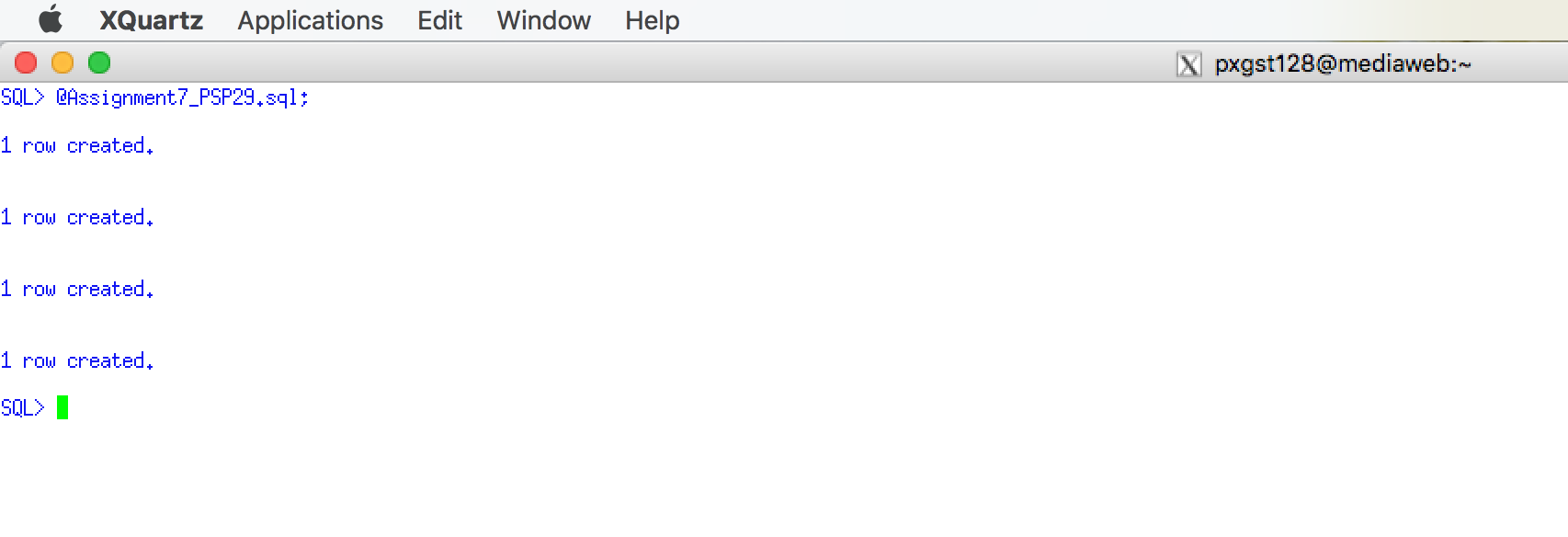
WHERE CUSTOMER\_TYPE = 'B');

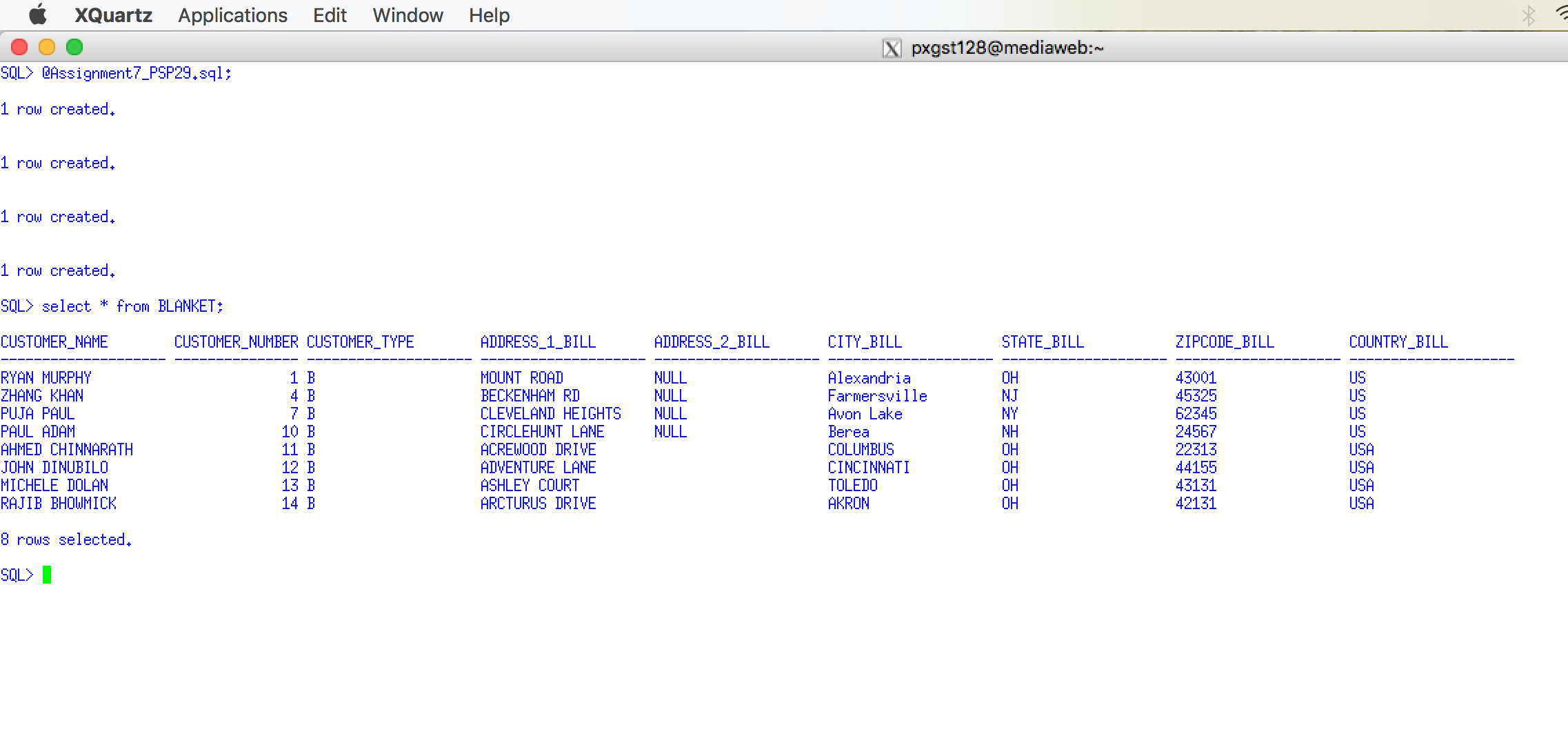
**Use a graphical snipping tool to document an image of these requirements below.**

## 2.9 Insert Customers

Add four new customers to the blanket table who are not in the original customer table.

Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>





INSERT INTO "PXGST128"."BLANKET" (CUSTOMER\_NAME, CUSTOMER\_NUMBER, CUSTOMER\_TYPE, ADDRESS\_1\_BILL, CITY\_BILL, STATE\_BILL, ZIPCODE\_BILL, COUNTRY\_BILL) VALUES ('AHMED CHINNARATH', '11', 'B', 'ACREWOOD DRIVE', 'COLUMBUS', 'OH', '22313', 'USA');

INSERT INTO "PXGST128"."BLANKET" (CUSTOMER\_NAME, CUSTOMER\_NUMBER, CUSTOMER\_TYPE, ADDRESS\_1\_BILL, CITY\_BILL, STATE\_BILL, ZIPCODE\_BILL, COUNTRY\_BILL) VALUES ('JOHN DINUBILO', '12', 'B', 'ADVENTURE LANE', 'CINCINNATI', 'OH', '44155', 'USA');

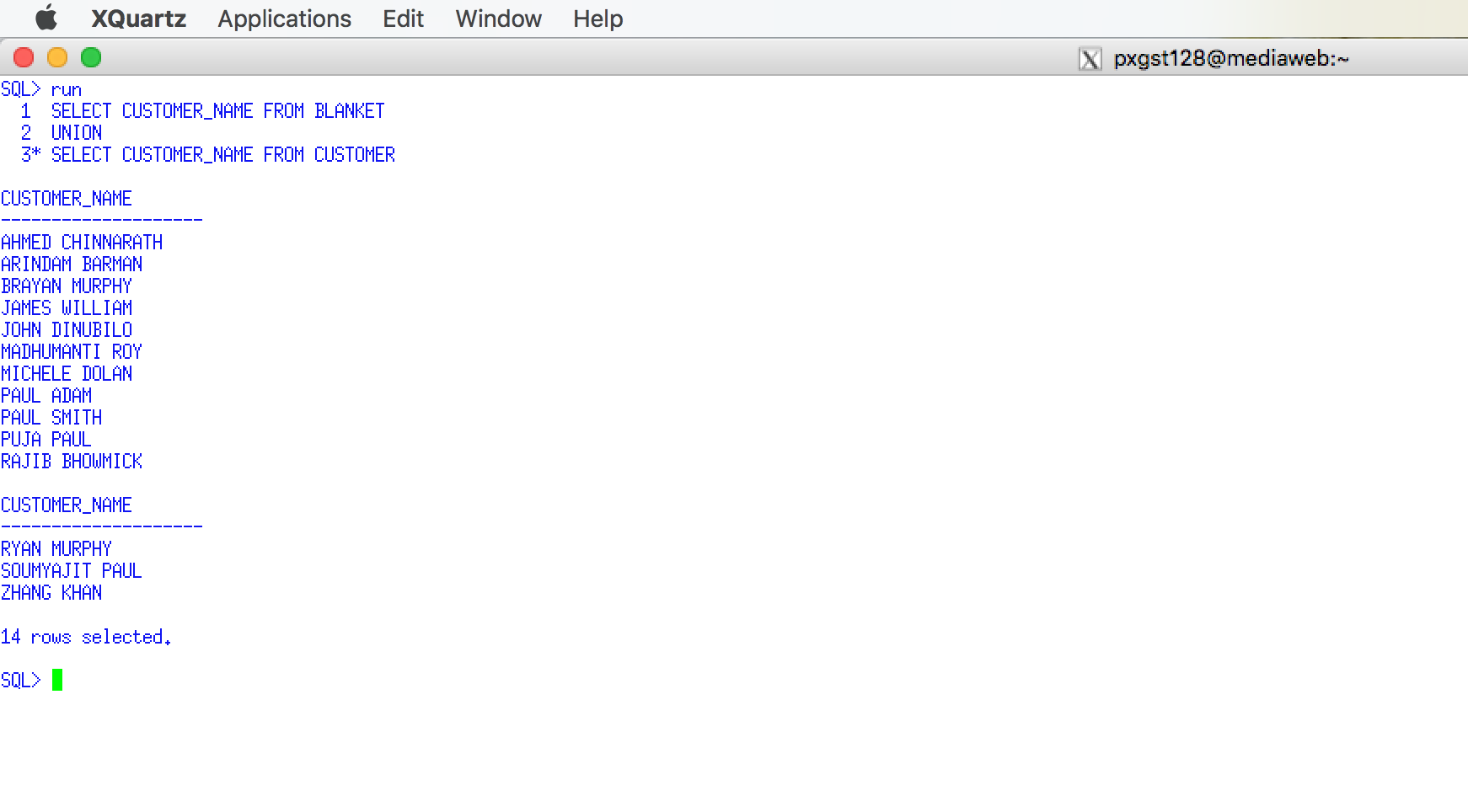
INSERT INTO "PXGST128"."BLANKET" (CUSTOMER\_NAME, CUSTOMER\_NUMBER, CUSTOMER\_TYPE, ADDRESS\_1\_BILL, CITY\_BILL, STATE\_BILL, ZIPCODE\_BILL, COUNTRY\_BILL) VALUES ('MICHELE DOLAN', '13', 'B', 'ASHLEY COURT', 'TOLEDO', 'OH', '43131', 'USA');

INSERT INTO "PXGST128"."BLANKET" (CUSTOMER\_NAME, CUSTOMER\_NUMBER, CUSTOMER\_TYPE, ADDRESS\_1\_BILL, CITY\_BILL, STATE\_BILL, ZIPCODE\_BILL, COUNTRY\_BILL) VALUES ('RAJIB BHOWMICK', '14', 'B', 'ARCTURUS DRIVE', 'AKRON', 'OH', '42131', 'USA');

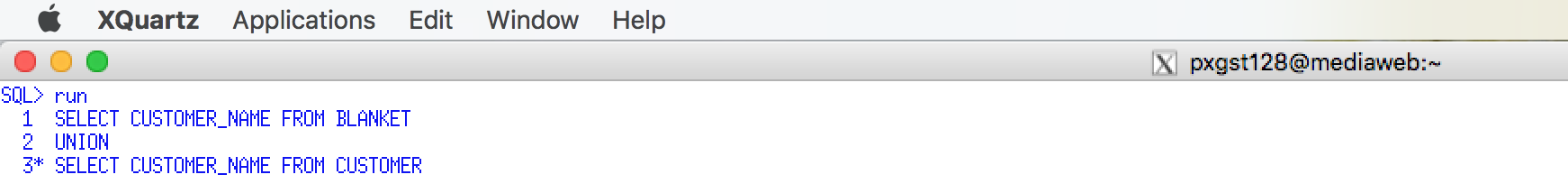
## 2.10 Apply UNION between Blanket Customers and Customers Tables

Code a UNION of these Blanket and Customer displaying only the customer name.

Use a Snippit to document your SQL output =>



Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>



SELECT CUSTOMER\_NAME FROM BLANKET

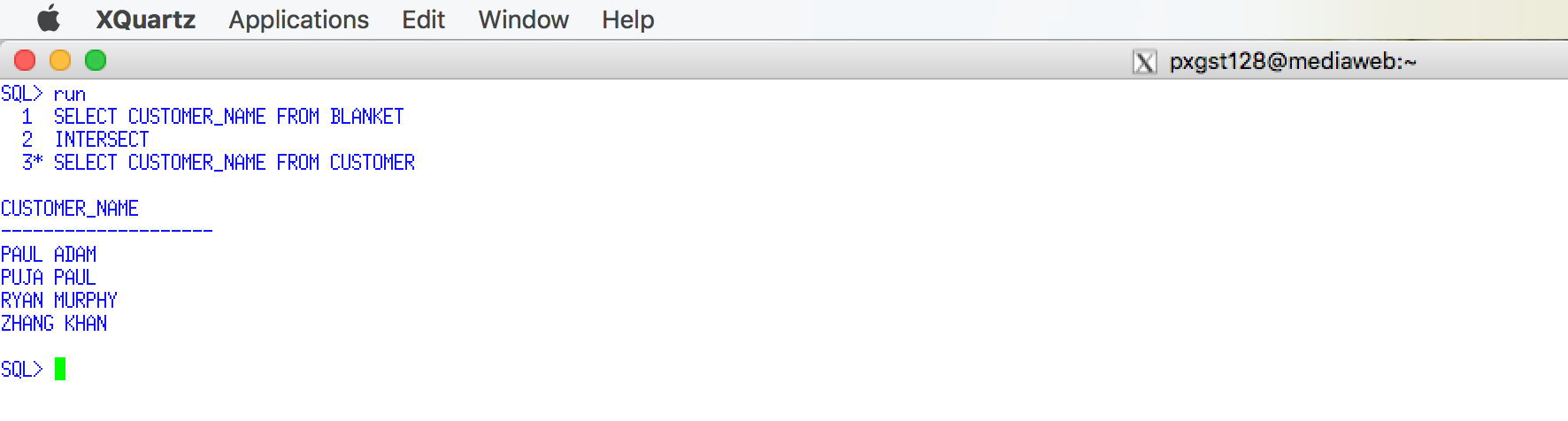
UNION

SELECT CUSTOMER\_NAME FROM CUSTOMER;

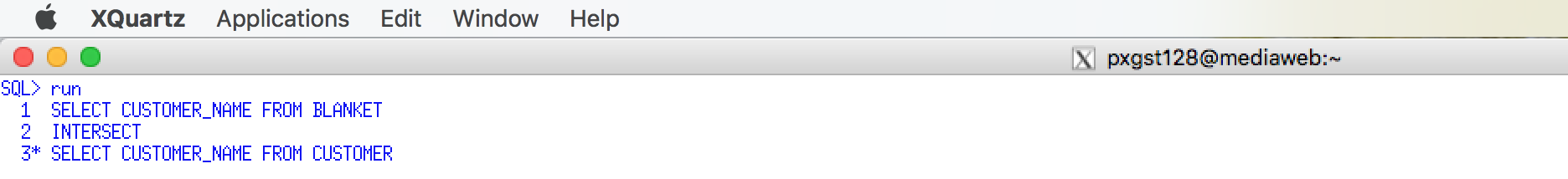
## 2.11 Apply INTERSECTION between Blanket Customers and Customers

Code an Intersection of Blanket and Customer displaying only the customer name.

Use a Snippit to document your SQL output =>



Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>



SELECT CUSTOMER\_NAME FROM BLANKET

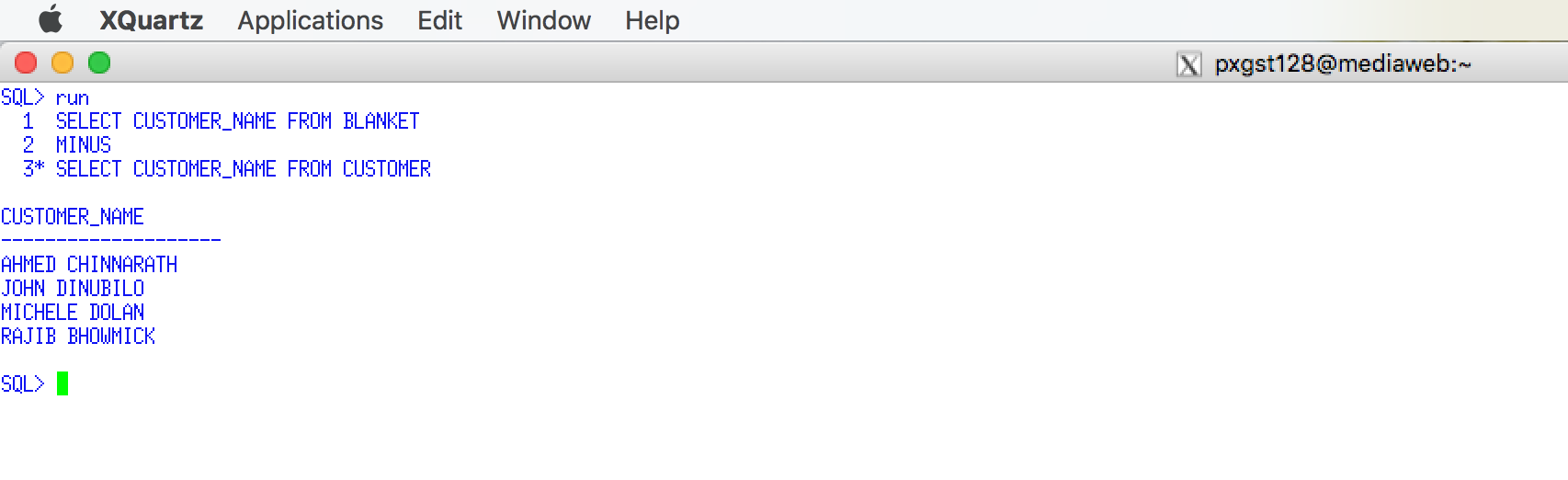
INTERSECT

SELECT CUSTOMER\_NAME FROM CUSTOMER;

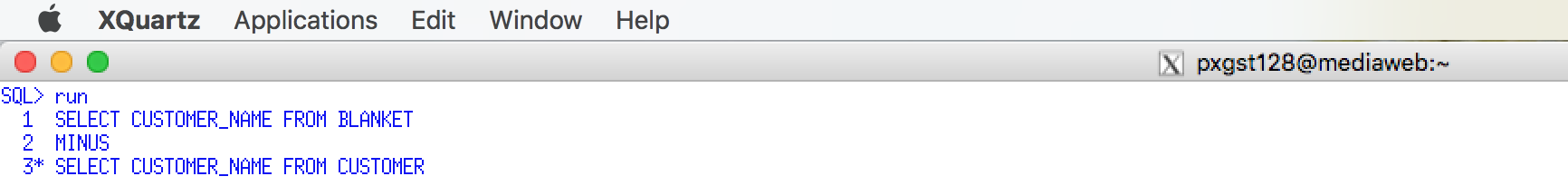
## 2.12 Apply DIFFERENCE between Blanket Customers and Customers

Display the Difference of Blanket and Customer displaying only the customer name.

Use a Snippit to document your SQL output =>



Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>



SELECT CUSTOMER\_NAME FROM BLANKET

MINUS

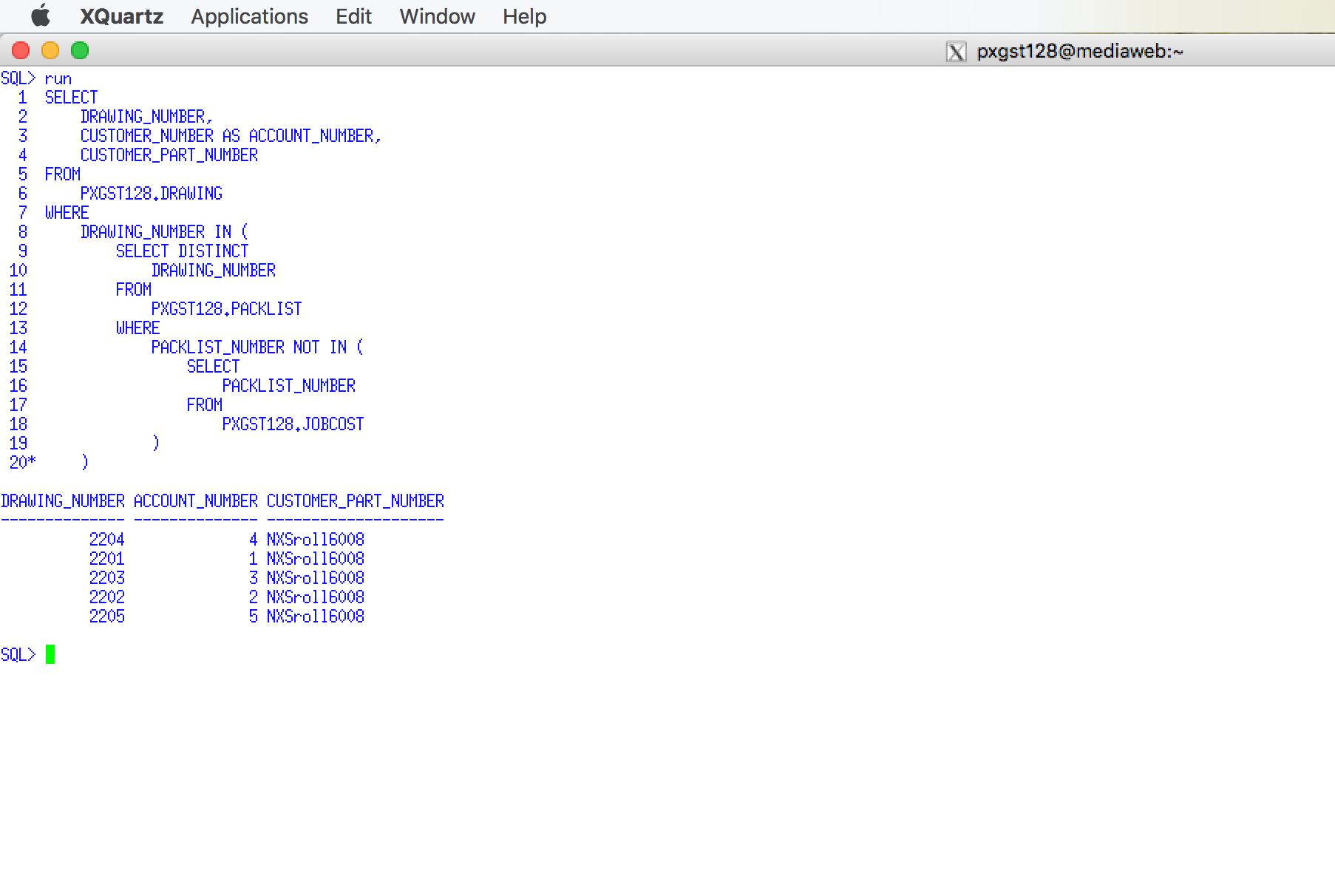
SELECT CUSTOMER\_NAME FROM CUSTOMER;

**Use a graphical snipping tool to document an image of these requirements below.**

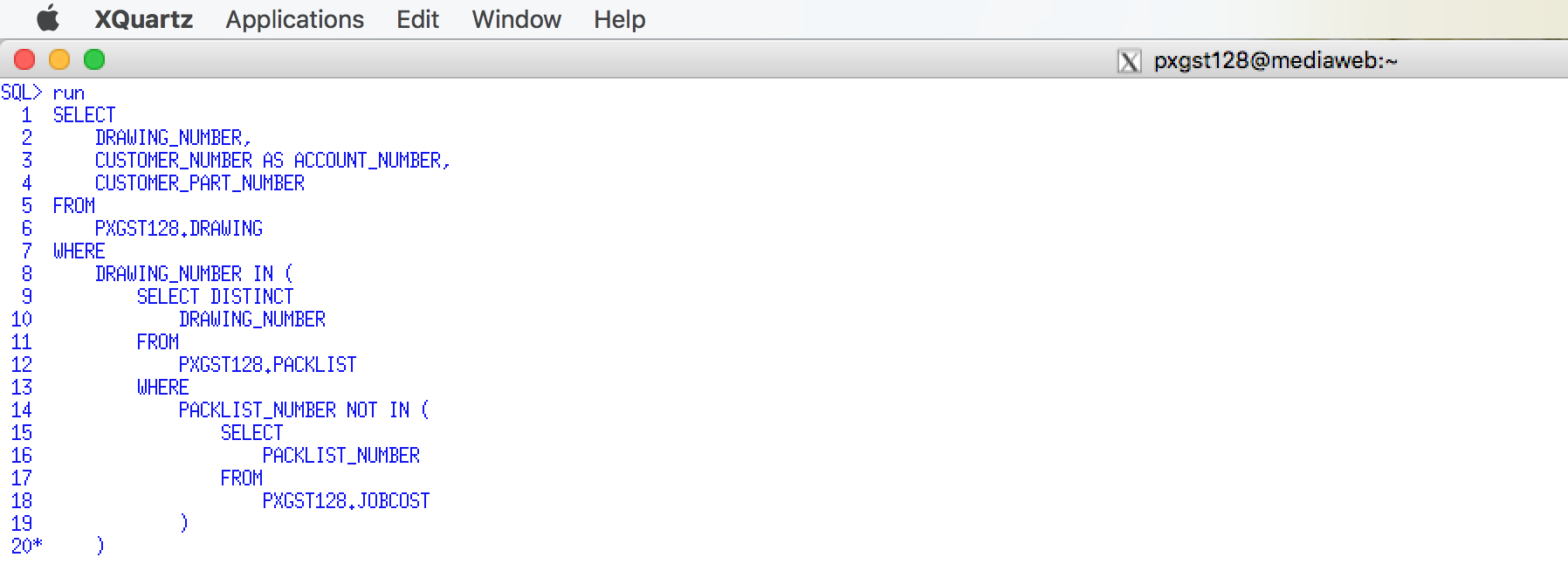
## 2.13 Apply Subquery using Drawing and Job Cost Tables

Display those Drawing Number, Account Number and Customer Part No that does not have any job costs. (Hint: use a subquery that uses a distinct retrieval).

Use a Snippit to document your SQL output =>



Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>



SELECT

DRAWING\_NUMBER,

CUSTOMER\_NUMBER AS ACCOUNT\_NUMBER,

CUSTOMER\_PART\_NUMBER

FROM

PXGST128.DRAWING

WHERE

DRAWING\_NUMBER IN (

SELECT DISTINCT

DRAWING\_NUMBER

FROM

PXGST128.PACKLIST

WHERE

PACKLIST\_NUMBER NOT IN (

SELECT

PACKLIST\_NUMBER

FROM

PXGST128.JOBCOST

)

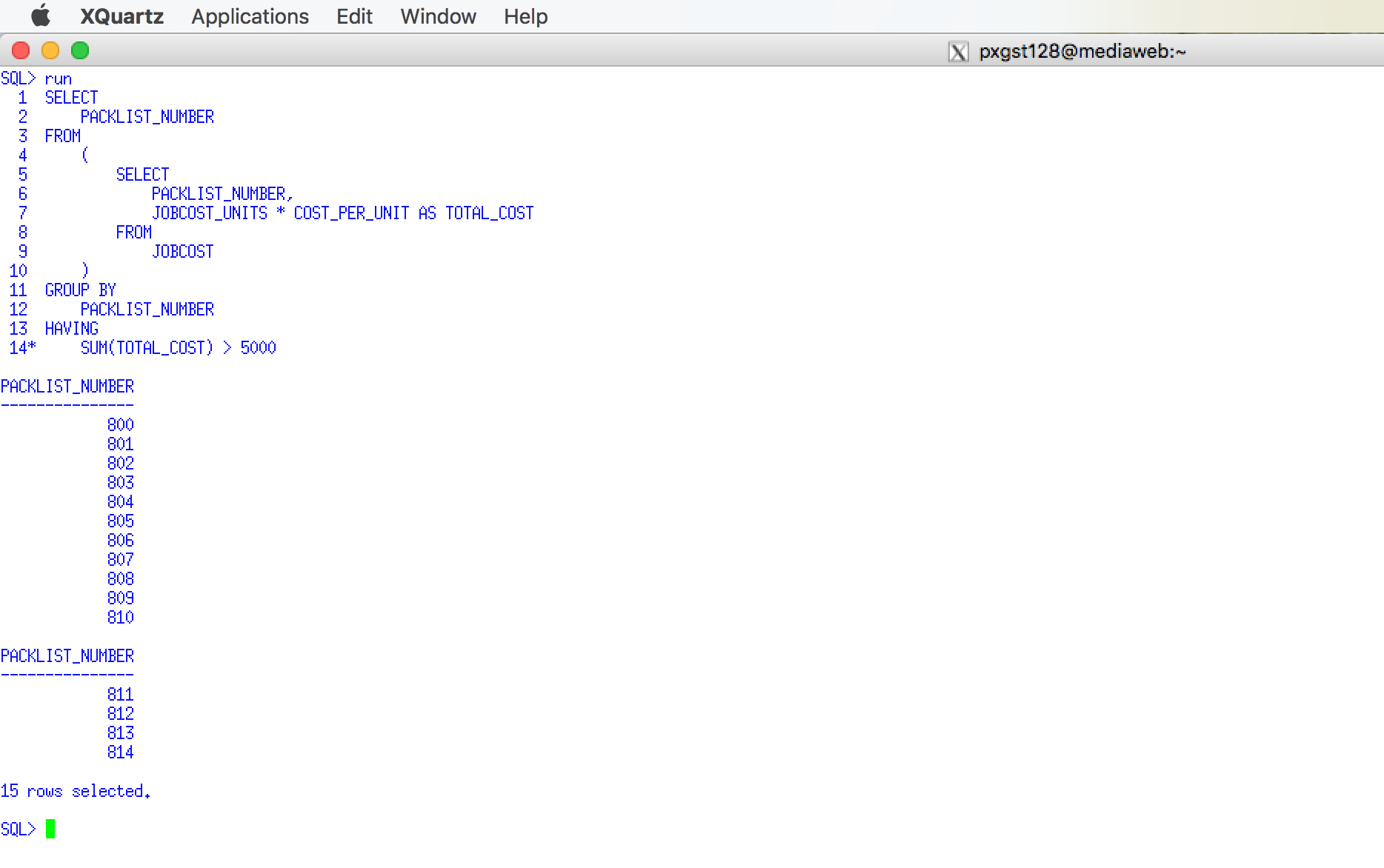
);

## 2.14 Coding a Complex Subquery

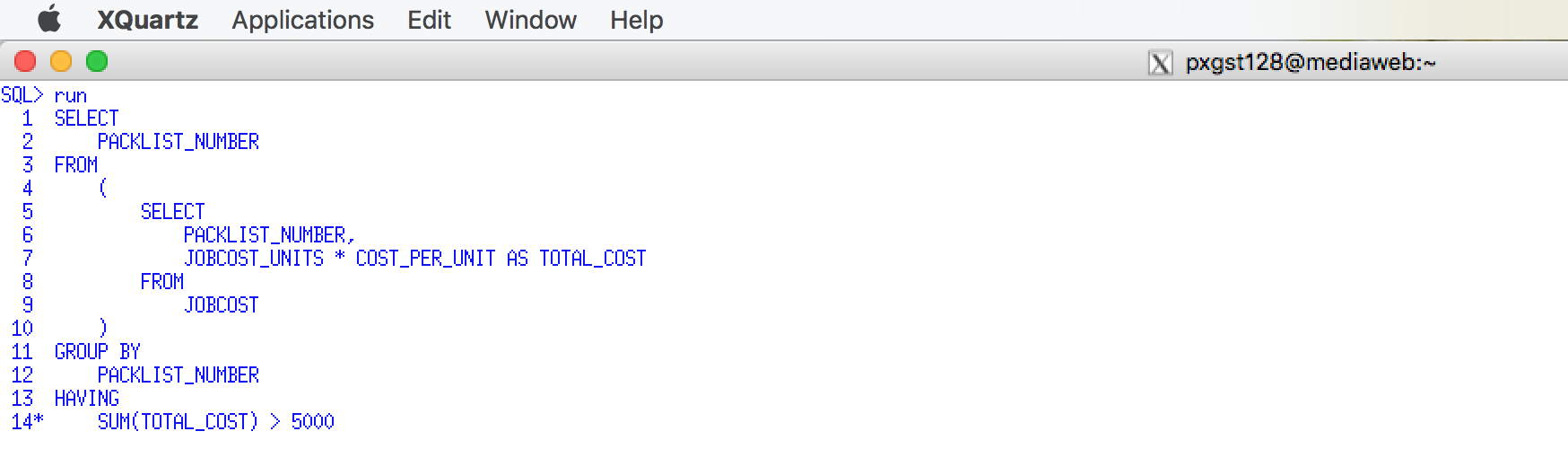
### 2.14.1 Step 1 - Code Inner Subquery – Step 1

Display those Packing List Numbers, which had a total job cost greater than the $5000. Hint: Calculate the total cost using a mathematical expression and using the HAVING clause and display the PL#s. The result should be a logical list of packing list number which had job cost greater than $5000. If you have no PL display you may have to add additional job costs or your code is incorrect.

Use a Snippit to document your SQL output =>



Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>



SELECT

PACKLIST\_NUMBER

FROM

(

SELECT

PACKLIST\_NUMBER,

JOBCOST\_UNITS \* COST\_PER\_UNIT AS TOTAL\_COST

FROM

JOBCOST

)

GROUP BY

PACKLIST\_NUMBER

HAVING

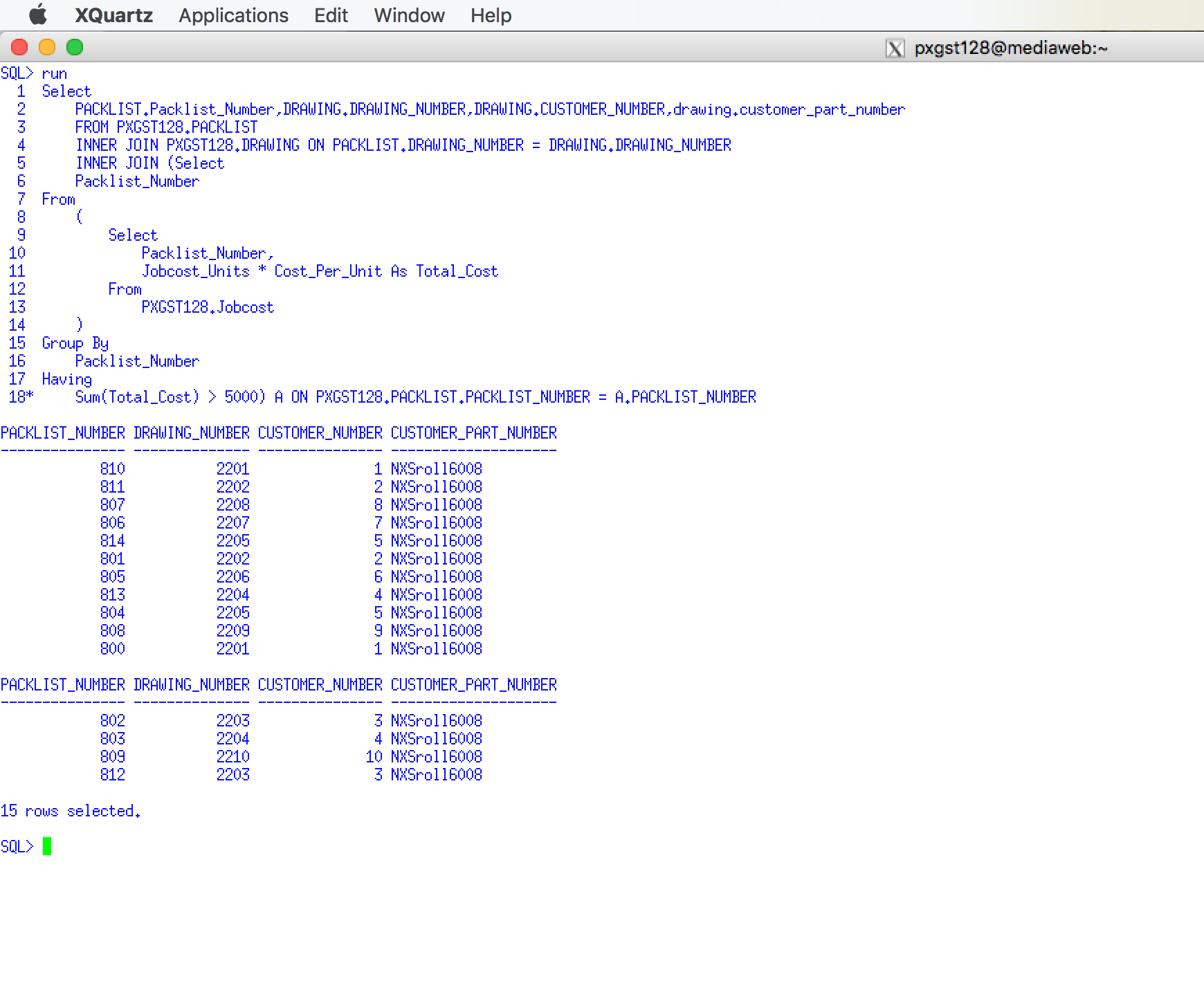
SUM(TOTAL\_COST) > 5000;

### 2.14.2 Step 2 - Code a Separate Outer Query

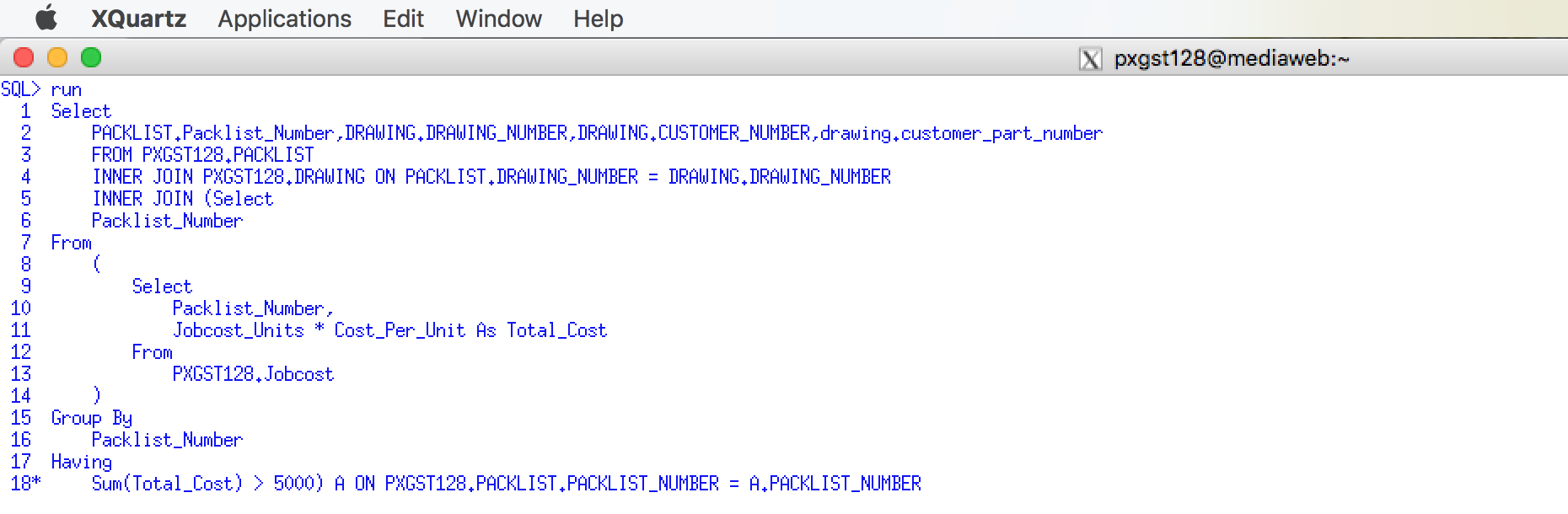
This example will add the outer query for the previous requirement. You must have completed the previous requirement to display some PL#s who have total job costs greater than $5,000. If not then insert job cost data to make some PLs have a total job cost > $5000

Code a join between Packing List and Drawing tables and thee Packing List, Drawing Number, Account Number and Customer Part No. This will be the outer query for next step.

Use a Snippit to document your SQL output =>



Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>



SELECT

PACKLIST.PACKLIST\_NUMBER,DRAWING.DRAWING\_NUMBER,DRAWING.CUSTOMER\_NUMBER,DRAWING.CUSTOMER\_PART\_NUMBER

FROM PXGST128.PACKLIST

INNER JOIN PXGST128.DRAWING ON PACKLIST.DRAWING\_NUMBER = DRAWING.DRAWING\_NUMBER

INNER JOIN (SELECT

PACKLIST\_NUMBER

FROM

(

SELECT

PACKLIST\_NUMBER,

JOBCOST\_UNITS \* COST\_PER\_UNIT AS TOTAL\_COST

FROM

PXGST128.JOBCOST

)

GROUP BY

PACKLIST\_NUMBER

HAVING

SUM(TOTAL\_COST) > 5000) A ON PXGST128.PACKLIST.PACKLIST\_NUMBER = A.PACKLIST\_NUMBER;

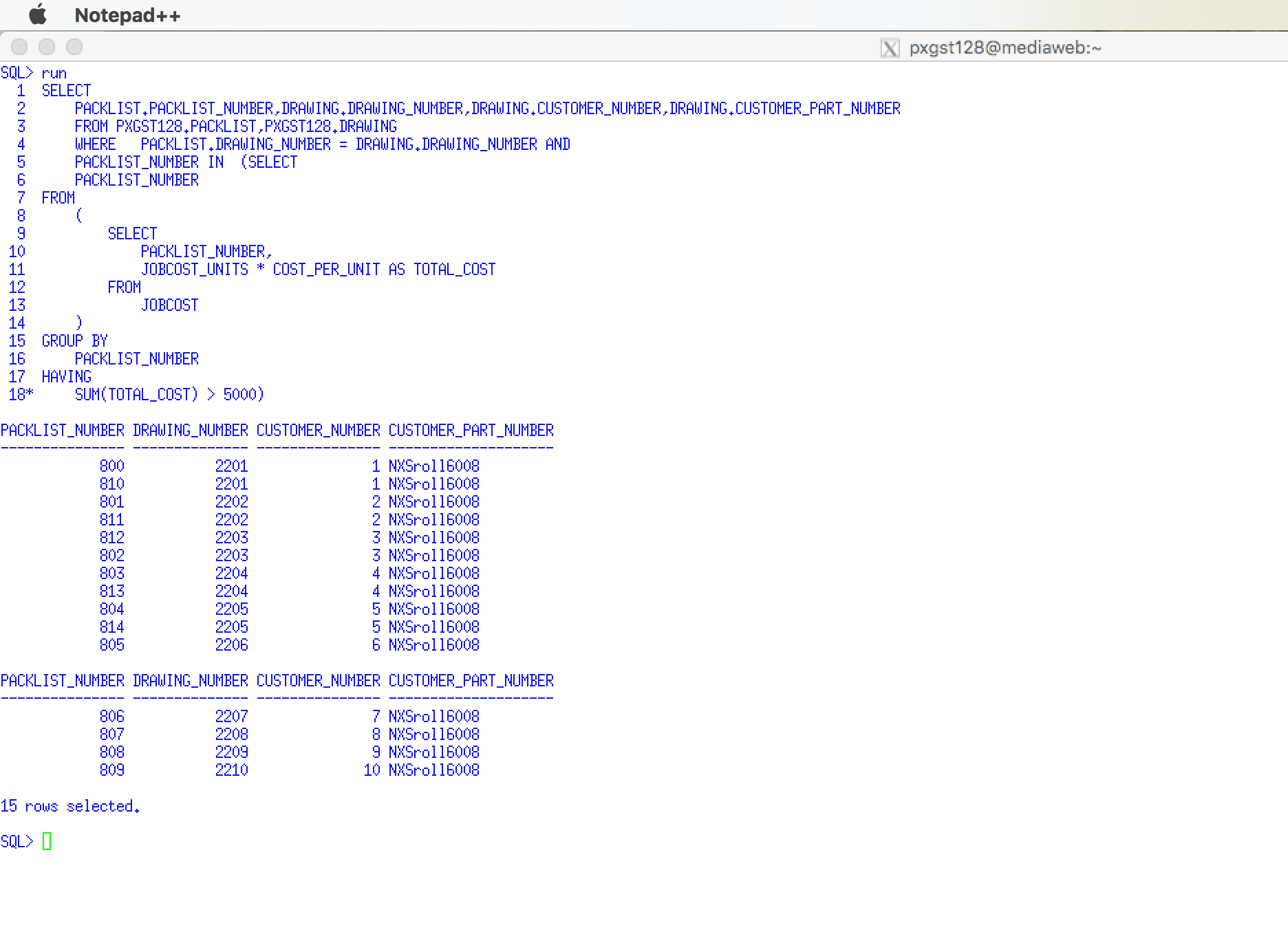
**Use a graphical snipping tool to document an image of these requirements below.**

### 2.14.3 Step 3 - Combine Outer Query with Inner Subquery

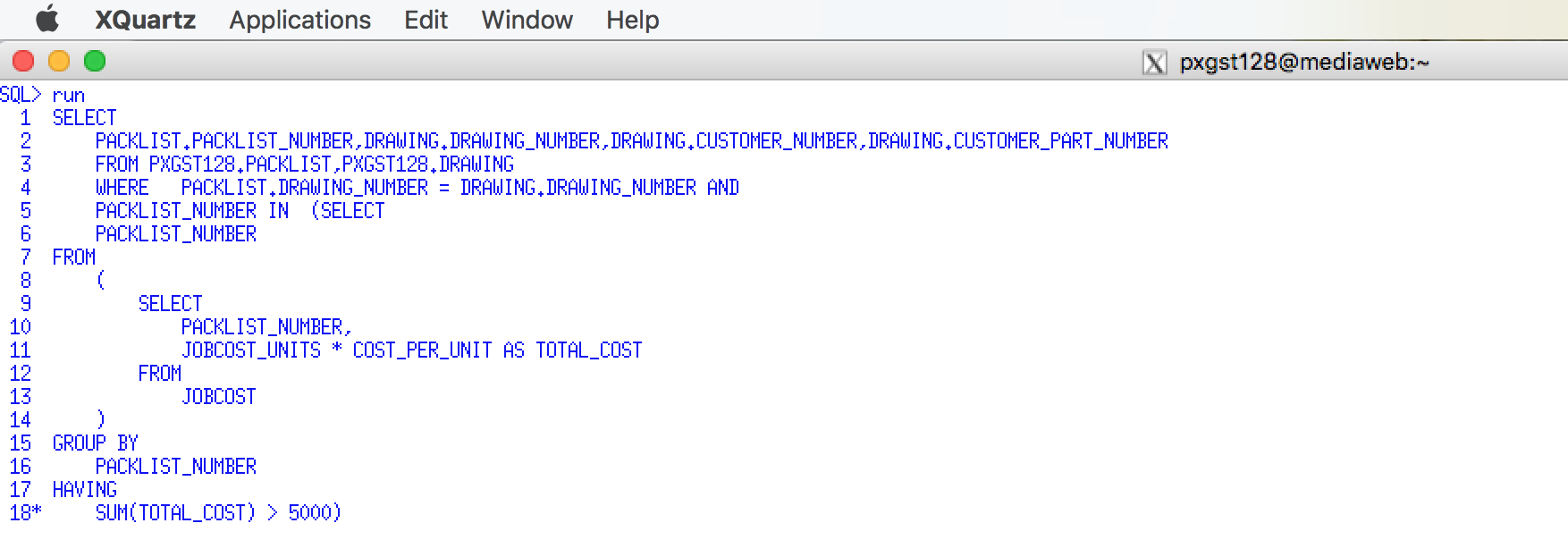
This Requirement will combine the Requirement Y the subquery and Requirement Y2 the outer query. You must have Requirement Y display some PL#s who have total job costs greater than $5,000. If not then insert job cost data to make some PLs have a total job cost > $5000

This requirement will display those Packing List, Drawing Number, Account Number and Customer Part No using a join between Packing List and Drawing and then using a subquery with the IN operator select those PLs in the subquery set whose total cost is greater than the $5000. Hint: Use a subquery to calculate the total cost and using the HAVING clause to select only those PLs

Use a Snippit to document your SQL output =>



Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>



SELECT

PACKLIST.PACKLIST\_NUMBER,DRAWING.DRAWING\_NUMBER,DRAWING.CUSTOMER\_NUMBER,DRAWING.CUSTOMER\_PART\_NUMBER

FROM PXGST128.PACKLIST,PXGST128.DRAWING

WHERE PACKLIST.DRAWING\_NUMBER = DRAWING.DRAWING\_NUMBER AND

PACKLIST\_NUMBER IN (SELECT

PACKLIST\_NUMBER

FROM

(

SELECT

PACKLIST\_NUMBER,

JOBCOST\_UNITS \* COST\_PER\_UNIT AS TOTAL\_COST

FROM

JOBCOST

)

GROUP BY

PACKLIST\_NUMBER

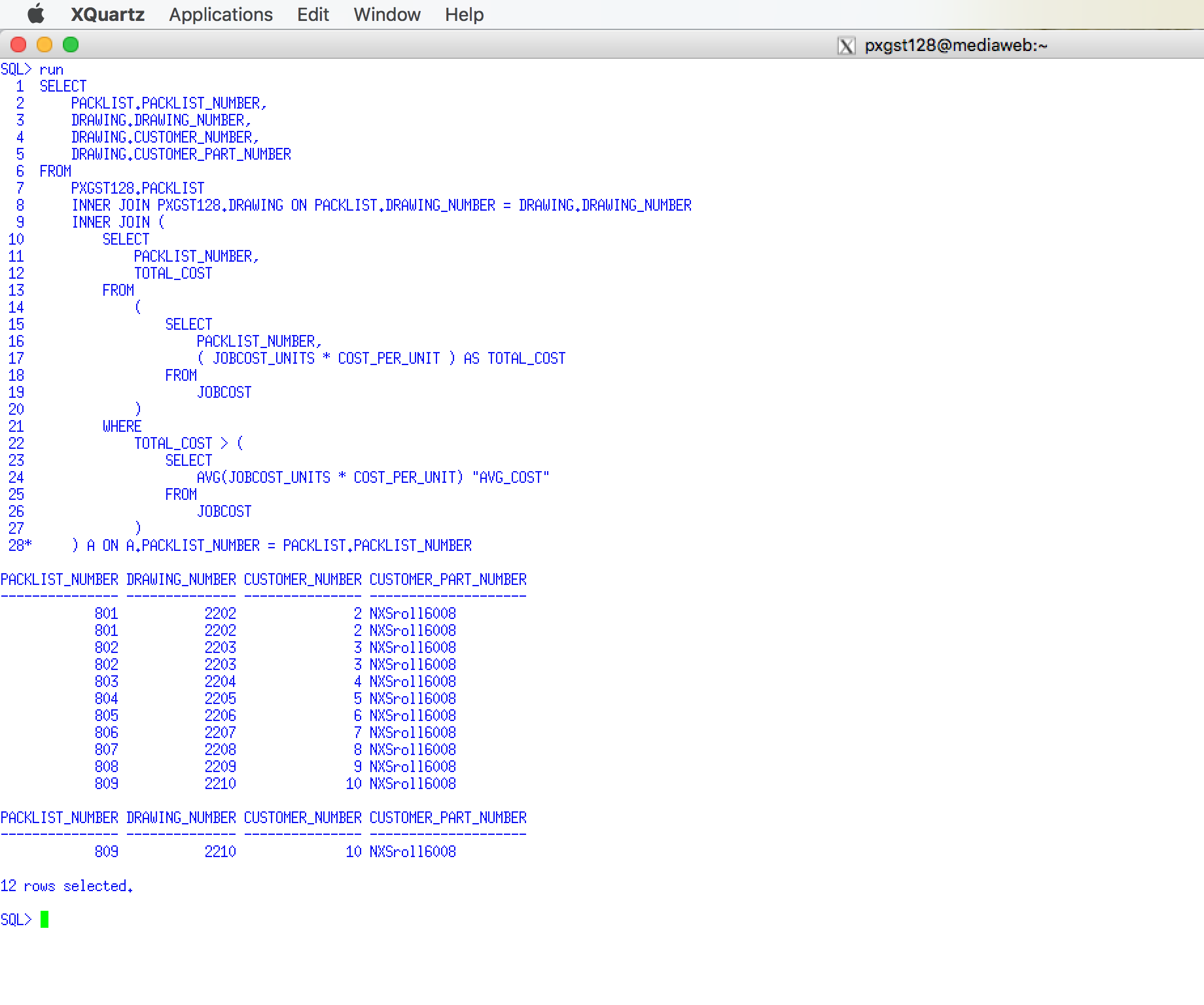
HAVING

SUM(TOTAL\_COST) > 5000) ;

## 2.15 Apply Inner Join between Packing List and Drawing with Job Cost Subquery

Display those Packing List, Drawing Number, Account Number and Customer Part No whose total cost is greater than the average total cost for all packing lists.

Use a Snippit to document your SQL output =>



Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>



SELECT

PACKLIST.PACKLIST\_NUMBER,

DRAWING.DRAWING\_NUMBER,

DRAWING.CUSTOMER\_NUMBER,

DRAWING.CUSTOMER\_PART\_NUMBER

FROM

PXGST128.PACKLIST

INNER JOIN PXGST128.DRAWING ON PACKLIST.DRAWING\_NUMBER = DRAWING.DRAWING\_NUMBER

INNER JOIN (

SELECT

PACKLIST\_NUMBER,

TOTAL\_COST

FROM

(

SELECT

PACKLIST\_NUMBER,

( JOBCOST\_UNITS \* COST\_PER\_UNIT ) AS TOTAL\_COST

FROM

JOBCOST

)

WHERE

TOTAL\_COST > (

SELECT

AVG(JOBCOST\_UNITS \* COST\_PER\_UNIT) "AVG\_COST"

FROM

JOBCOST

)

) A ON A.PACKLIST\_NUMBER = PACKLIST.PACKLIST\_NUMBER;

# 3.0 Apply SQL View to PSP Case – Level 3

**Important**

The results should be easy to read and grade. The following is some guidelines that may help you minimize some formatting.

SQL String Functions - http://webspace.cs.odu.edu/~ibl/450/common/sqlstring.html

To\_Date Function - <http://www.techonthenet.com/oracle/functions/to_date.php>

Oracle TO\_DATE Function – SQL Syntax Examples - <http://www.mandsconsulting.com/oracle-to_date-function-sql-syntax-examples>

Oracle TO\_CHAR Function – SQL Syntax Examples (Most With Dates, TO\_DATE) - <http://www.mandsconsulting.com/oracle-to_char-function-sql-syntax-examples-most-with-dates-to_date>

Oracle Built in Functions - <http://beginner-sql-tutorial.com/oracle-functions.htm>

a. **I cannot grade wrapped text.** By default SQLPlus has a default line size of 80 characters. If your results are greater than 80 characters wide it will wrap to the next line. While you may correct the problem in Word, a better solution is change SQLplus's linesize, e.g., set linesize 150

b. A problem with the set linesize is that the text will be display past the right margin of the Putty screen. One solution would be to change Putty's font size. You may also change Word's font size or even switch to landscape.

c. The substr function, e.g., see <http://www.techonthenet.com/oracle/functions/substr.php>, can help by displaying a smaller column width than is stored.

For example Select substr(Cname ,1, 4) from customer; would only display the first four characters of the customer name.

d. The to-char function, e.g., see <http://www.techonthenet.com/oracle/functions/to_char.php>, can better display dates and times. For example,

Select to\_char(date\_shipped, 'Month DD, YYYY') from invoice; would display July 09, 2012

## 3.1 SQL Views

A view is a logical representation of another table or combination of tables. A view derives its data from the tables on which it is based. These tables are called base tables. Base tables might in turn be actual tables or might be views themselves. All operations performed on a view actually affect the base table of the view. You can use views in almost the same way as tables. You can query, update, insert into, and delete from views, just as you can standard tables.

Views can provide a different representation (such as subsets or supersets) of the data that resides within other tables and views. Views are very powerful because they allow you to tailor the presentation of data to different types of users.

A view is an external record. It is the technique by which DB2 and Oracle implements conceptual independence. ACCESS does not support the concept of a VIEW directly because it provides the users with the Query facility.

***There are two steps involved with a view: 1) creating a view, 2) using or executing a view.***

To create a view, you must meet the following requirements:

* To create a view in your schema, you must have the CREATE VIEW privilege. To create a view in another user's schema, you must have the CREATE ANY VIEW system privilege. You can acquire these privileges explicitly or through a role.
* The owner of the view (whether it is you or another user) must have been explicitly granted privileges to access all objects referenced in the view definition. The owner cannot have obtained these privileges through roles. Also, the functionality of the view depends on the privileges of the view owner
* If the owner of the view intends to grant access to the view to other users, the owner must have received the object privileges to the base objects with the GRANT OPTION or the system privileges with the ADMIN OPTION.

Views, Materialized Views and Synonyms - http://ss64.com/ora/syntax-views.html

What is the difference between Views and Materialized Views in Oracle? http://stackoverflow.com/questions/93539/what-is-the-difference-between-views-and-materialized-views-in-oracle

**Video How to create a View in Oracle SQL** [**https://www.youtube.com/watch?v=mDL7W5jbj-0**](https://www.youtube.com/watch?v=mDL7W5jbj-0)

**Video - Materialized Views -** [**https://www.youtube.com/watch?v=QuBD5DAvxPM**](https://www.youtube.com/watch?v=QuBD5DAvxPM)

## 3.2 SQL Views to Simplify Complicated SQL statements

Suppose that users believe that the previous INNER JOIN example merging four separate tables was too difficult to remember. We could create a view to simplify the process. For example,

Example 84

ANSI and Oracle SQL

To create the VIEW

**CREATE VIEW SCHEDULE (CNO, CNAME, SNO, SNAME, FNAME)**

**AS**

**SELECT COURSE.CNO, CNAME, STUDENT.SNO, SNAME, FACULTY.FNAME**

**FROM COURSE\_SCHEDULE, COURSE**

**WHERE COURSE\_SCHEDULE.CNO = COURSE.CNO AND**

**COURSE\_SCHEDULE.SNO = STUDENT.SNO AND**

**COURSE\_FNO = FACULTY.FNO**

To execute the VIEW

**SELECT \* FROM SCHEDULE**

**ORDER BY SNO**

or

**SELECT \* FROM SCHEDULE**

**WHERE BY SNAME = ‘GEORGE’**

As you can see from the previous example, once a view has been created it can be processed similar to any other SQL base table. However, you cannot create an "ORDERed view". You must use the ORDER clause when you execute the view.

## 3.3 SQL Views to Enhance Security

Suppose you want the students to be able to see the faculty's names, but you don't want to disclose sensitive information, i.e., FPAY.

Example 85

ANSI and Oracle SQL

Create the VIEW

**CREATE VIEW FACULTY\_INFO (FID, FNAME, FDEP)**

**AS**

**SELECT FID, FNAME, FDEP**

**FROM FACULTY**

Executing the VIEW

**SELECT \* FROM FACULTY\_INFO**

**Or**

**SELECT \* FACULTY\_INFO WHERE FDEP = ‘CIS’**

By GRANTing a student SELECT privileges to the view FACULTY\_INFO and not the base table FACULTY, students can only have access only to the information available in the view.

## 3.4 SQL Views to Establish Conceptual Independence

As a rule for the thumb, you should always create a view for all base tables and your never program or use base tables. If you execute SQL commands processing a view, your will never be affected by changes in the base table.

For example, Create the VIEW

**CREATE VIEW FACULTY\_VIEW (FID, FNAME, FDEP, FPAY)**

**AS**

**SELECT FID, FNAME, FDEP, FPAY**

**FROM FACULTY**

Now, you should only use the FACULTY\_VIEW. In the DB2 system Catalogue, the system tables, the table SYSVIEWS promotes conceptual independence: For example:

**VIEW NAME VIEW COLUMN BASE TABLE NAME BASE TABLE COLUMN**

**FACULTY\_VIEW FID FACULTY FID**

**FACULTY\_VIEW FNAME FACULTY FNAME**

**FACULTY\_VIEW FDEP FACULTY FDEP**

**FACULTY\_VIEW FPAY FACULTY FPAY**

Suppose that you wish to use FACULTY\_VIEW(FPAY). At the time of execution DB2 and Oracle looks up the location of base table and real column name, i.e., FACULTY(FDEP). Now suppose that the data base administrator alters the column name of FACULTY(FDEP) to FACULTY(SALARY) as SYSVIEWS is updated to the following:

**VIEW NAME VIEW COLUMN BASE TABLE NAME BASE TABLE COLUMN**

**FACULTY\_VIEW FID FACULTY FID**

**FACULTY\_VIEW FNAME FACULTY FNAME**

**FACULTY\_VIEW FDEP FACULTY FDEP**

**FACULTY\_VIEW FPAY FACULTY SALARY <‑‑‑**

It doesn't matter to SQL users processing the view FACULTY\_VIEW because the FACULTY\_VIEW(FPAY) remains unchanged. A view column, i.e., FACULTY\_VIEW(FPAY) inherits the data type of the base table column, i.e., FACULTY(SALARY).

## 3.5 Restrictions on using a VIEW

1) You cannot update a view that joins more than one table, i.e., more than one table specified in the FROM clause.

2) You cannot update a view that SELECTS arithmetic expressions, aggregate functions, or uses the DISTINCT clause.

3) You cannot update a view that contains a nested SELECT, a GROUP BY, or a HAVING clause.

## 3.6 WITH CHECK OPTION

Suppose that we create the following view for the chairman of the CIS department. While the view will prevent the CIS chairman from SELECTING non‑CIS faculty members, the CIS chairman could INSERT, UPDATE or DELETE non\_CIS faculty members. The WITH CHECK OPTION limits the scope of any changes to that are within the scope of the VIEW.

**CREATE VIEW FACULTY\_CIS (FID, FNAME, FDEP, FPAY)**

**AS**

**SELECT FID, FNAME, FDEP, FPAY**

**FROM FACULTY**

**WHERE FDEP = ‘CIS’**

**WITH CHECK OPTION**

## 3.7 Questions - View

1. What is a View? Answer =>

A view is a logical represenatation of another table or combination of tables. A view derives its data from the table to which is it based.

2. List three reasons why one would code a view. Answer =>

complex quiries that need to executed often can be saved in a view. Hence by calling the view name, query can be executed.

View is useful when query uses multiple tables. If tables are changed , only the view definition needs to be changed and not the code where view is used.

View are virtual and occupy no space.

3. List three restrictions on creating or using a VIEW. Answer =>

view can not be updated when join one or more tables.

It can not be updated that selects arithmetic expression.

4. Why should consider the use of creating a VIEW using the WITH CHECK OPTION. Answer =>

WITH CHECK OPTION clause can be given for an updatable view to prevent inserts to rows for which the where clause in the select statement is not true. It also prevents updates to rows for which the WHERE clause is true but the update would cause it to be not true. In other words, it prevents visible rows from being updated to non visible rows.

**Use a graphical snipping tool to document an image of these requirements below.**

## 3.8 Apply View to Inner Join of Job Cost and Packing List

Create a View to simplify the execution of representing join requirement for the following SELECT requirements

Labor costs by Packing List Number for Customer's Part Number

Join the Job Cost and Packing List table, and display the Packing List Number, Customer Number, Drawing Number and the total (SUM) cost (transaction units multiplied by unit cost) for only labor over time grouped for each packing list number ordered by packing list number. Only the totals should be printed, not the detail records.

**Code the new View and execute it.**

Use a Snippit to document your SQL output =>

Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>

**Execute the view.**

Use a Snippit to document your SQL output =>

Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>

**Use a graphical snipping tool to document an image of these requirements below.**

## 3.9 Apply View to Inner Join between Customer, Drawing and Packing List

Create a View representing the following join requirements

Join the Customer, Drawing, and Packing List tables. Display the Customer Name, Customer Number, Drawing Number, Customer Part Number, Packing List Number, Planned Ship Date, order by Account Number and Drawing Number. Code the View and execute it. Use the WITH CHECK OPTION

Use a Snippit to document your SQL output =>

Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>

**Execute the view.**

Use a Snippit to document your SQL output =>

Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>

**Use a graphical snipping tool to document an image of these requirements below.**

## 3.10 Apply View to Join between Customer, Drawing, Option and Drawing Option

Create the view that will simplify the

Join the Customer, Drawing, Option and Drawing Option tables. Display the Customer Name, Customer Number, Drawing Number, Customer Part Number, Option Number, Option Description, and a Dollar or Percentage option each Customer drawing order by Account Number and Drawing Number. You need to display at least two drawing with some drawing options. Code the View and execute it.

**Code the View and execute it.**

Use a Snippit to document your SQL output =>

Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>

**Execute the view.**

Use a Snippit to document your SQL output =>

Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>

## 3.11 Materialized View Concepts

Oracle uses materialized views (also known as snapshots in prior releases) to replicate data to non-master sites in a replication environment and to cache expensive queries in a data warehouse environment. DB2 supports the same concept, except they use the terminology of Materialized Query Tables (MQTs).

Views evaluate the data in the tables underlying the view definition at the time the view is queried. It is a logical view of your tables, with no data stored anywhere else. The upside of a view is that it will always return the latest data to you. The downside of a view is that its performance depends on how good a select statement the view is based on. If the select statement used by the view joins many tables, or uses joins based on non-indexed columns, the view could perform poorly.

Materialized views or query tables are similar to regular views, in that they are a logical view of your data (based on a select statement), however, the underlying query result set has been saved to a table. The upside of this is that when you query a materialized view, you are querying a table, which may also be indexed. In addition, because all the joins have been resolved at materialized view refresh time, you pay the price of the join once (or as often as you refresh your materialized view), rather than each time you select from the materialized view. In addition, with query rewrite enabled, Oracle can optimize a query that selects from the source of your materialized view in such a way that it instead reads from your materialized view. In situations where you create materialized views as forms of aggregate tables, or as copies of frequently executed queries, this can greatly speed up the response time of your end user application. The downside though is that the data you get back from the materialized view is only as up to date as the last time the materialized view has been refreshed.

Materialized views or query tables can be set to refresh manually, on a set schedule, or based on the database detecting a change in data from one of the underlying tables. Materialized views can be incrementally updated by combining them with materialized view logs, which act as change data capture sources on the underlying tables.

Materialized views or query tables are most often used in data warehousing / business intelligence applications where querying large fact tables with thousands of millions of rows would result in query response times that resulted in an unusable application. When you need performance on data that don't need to be up to date to the very second, materialized views are better, but your data will be older than in a standard view.

### 3.11.1 Questions - Materialized View Concepts

1. What is a Materialized View? Answer =>

A materialized view in oracle is a database object that contains the results of a query. They are local copies of data located remotely, or are used to create summary tables based on aggregations of a table’s data. Materialised views, which store data based on the remote tables are also, known as sanpshots.

2. List several reasons why one would prefer a Materialized View as compared to a traditional View. Answer =>

materialized view helps to reduce netwoek loads to distribute your corporate database to regional sites. Instead of the entire company accessing a single database server. User load is distributed across multiple database servers.

Deployment templates allow you to precreate a materialized view environment locally. Deployment templates can be used to quickly and easily deploy materialized view environments to support sales force automation and other mass deployment environments.

Materialzed view allow to replicate data based on column- and row-level setting, while multimater replication requires replication of the entire table.

Materialized views do not require a dedicated network connection.

3. Where are Materialized Views most often used?. Answer =>

Materialized views or query tables are most often used in data warehousing or business intelligence applications. In this kind of querying large fact tables with thousands of millions of rows would result in query response times that resulted in an unusable application.

# 4.0 Apply PL/SQL to the PSP Case – Level 4

Most of the PL/SQL procedures are slight variations of what has been demonstrated in the Assignment 7 Theory on PL/SQL, Triggers, etc. in Blackboard.

## 4.1 Duplicate Customer Name Stored Procedure

Code and Create a Stored PL/SQL Procedure that inputs a Customer name and then checks to see if a Customer Name already exists in the Customer Table. This could mean that the customer may be a duplicate customer record and that it already exists.

Use the LIKE clause that your validation searches for any match and not an exact match. If a duplicate does exist return the Customer Name of the duplicate with an error message

Execute and use a Snippit to document your SQL output =>

Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>

## 4.2 Customer Add with duplicate name validation Stored Procedure

Code and Create a Stored PL/SQL Procedure that inserts a new customer into the Customer Table by inputting the CUSTOMER NAME, CUSTOMER NUMBER {common column} and CUSTOMER TYPE

Before the new record is inserted the previous Duplicate Customer N should be called.

If a duplicate/similar customer name exists and error message should be displayed using DMBS\_OUTPUT.PUT\_LINE listing the duplicate name.

A successful add message should be displayed using DMBS\_OUTPUT.PUT\_LINE.

Appropriate Exception handling should be used. Execute the procedure and document the results.

Execute and use a Snippit to document your SQL output =>

Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>

## 4.3 Customer Drawing List Stored Procedure

Code and Create a Stored Procedure that will use a cursor that will list the Appropriate Customer Information and drawing information (a join is required) using DMBS\_OUTPUT.PUT\_LINE.

Appropriate Exception Handling should be used.

Execute and use a Snippit to document your SQL output =>

Use a Snipping tool document your SQL code, which also displays your schema name for each table name =>