1. **Using the following relations:**

product ( pno, pname, man)

pno – product number

pname – product name

man – manufacturer identification number

manufacturer ( manno, mname, tickerno )

manno – manufacturer identification number

mname – manufacturer name

tickerno – stock market identification code

**For each of the following, give the relational algebra (6 points each), tuple relational calculus (5 points each) and domain relational calculus (3 points each) notion for each**

* 1. **Find the pno for all products.** (I am assuming we want only the pno attribute)
     1. **Relational Algebra**

∏pno (product)

* + 1. **Tuple Relational Calculus**

{t | ∃ s ∈ product(t[pno] = s[pno])}

* + 1. **Domain Relational Calculus**

{ < pno> | ∃ pname, man ( <pno, pname, man> ∈ product)}

* 1. **Find all tuples in product where pname = “eraser”** 
     1. **Relational Algebra**

σpname=’eraser’(product)

* + 1. **Tuple Relational Calculus**

{t | t ∈ product(t[pname] = ‘eraser’)}

* + 1. **Domain Relational Calculus**

{<pno, pname, product> | <pno, pname, product> ∈ product ∧ product = ‘eraser’|

* 1. **Find the pno and tickerno for all products**
     1. **Relational Algebra**

∏pno, tickerno(σman = manno(product x manufacturer))

* + 1. **Tuple Relational Calculus**

{t | ∃ s ∈ product(t[pno] = s[pno]) ∧

∃ u ∈ manufacturer(s[man] = u[manno]) ∧

u[tickerno] = t[tickerno]}

* + 1. **Domain Relational Calculus**

{<pno, tickerno> | ∃ pname, man (<pno, pname, man> ∈ product)

∧ ∃ mname (<man, mname, tickerno> ∈ manufacturer)}

I used full attribute names in my DRC to improve readability to myself. In order to join relations in the DRC in part c, I renamed the variable corresponding to “manno” as “man” in the manufacturer expression to match the name in the product expression.

1. **Write the SQL code for division. Try to do it without using the EXCEPT clause. If you cannot figure it out, I will accept the EXCEPT clause with a slight deduction in points. There are no particular tables for this assignment so you make up table and entity names if necessary for your code. (8 points)**

Remember you can write r ÷ s as

temp1← ∏R-S (r )

temp2 ← ∏R-S ((temp1 x s ) – ∏R-S,S (r ))

result = temp1 – temp2

Made up scenario: A series of different tests are run on a number of different models of TVs. A test can pass or fail. r ÷ s in this case returns tv models that fail every test. First, I write out the relational algebra to build a scenario to work from.

relation 1: t(T) = tv(manufacturer, model, test, result)

r 🡨 ∏model, test(σresult=’fail’(tv))

s 🡨 ∏test(tv)

r(model, test) ÷ s(test) = <models of tv that fail every test>

**SQL:**

Relations r and s could already exist, or we could create views of the relation above using the select statements below. The values that are contained in r and s are described using the select statements below.

**r:**

SELECT model, test

FROM tv

WHERE result = ‘fail’

**s:**

SELECT DISTINCT test

FROM tv

**Division r ÷ s Without “EXCEPT”:**

SELECT DISTINCT model

FROM r

WHERE model not in (SELECT DISTINCT total.model

FROM (SELECT DISTINCT r.model, s.test

FROM r, s) total FULL OUTER JOIN r on total.model = r.model

AND total.test = r.test

WHERE r.model is Null)  
I am taking a cartesian product of the two tables, outer joining this with ‘r’, and finding all the null values. The nulls exist in tuples that would be found in “temp2” in the example. I select rows from r that do not correspond to these null values. This last “select where not in” mimics the behavior of both instantiating temp1 and subtracting temp2 in the example.