# 308-612A Database Programing Principles

# Supplementary Questions and Exercises

## Relational and Domain Algebras

1. Using only the natural join and the domain algebra, write an expression which is equivalent to $R \supseteq S$ for relations $R(A, B)$ and $S(B, C)$.

2. The relation *LAYOUT(ASSY,SUBA,X,Y)* gives the two-dimensional layout of a circuit design in a three-level hierarchy, from gates up to a four-bit adder, with full-adders and a half-adder as subassemblies of the four-bit adder, and *and, nand* and *xor* gates as subassemblies of the full- and half-adder. $X$ and $Y$ are the relative positions within the assembly of each subassembly. Write Aldat code to compute the relative position for each *and, nand* and *xor* gate within the four-bit adder.

*LAYOUT*

| (*ASSY* | *SUBA* | $X$ | $Y$) |
|---|---|---|---|
| 4-adder | fulladder | 2 | 2 |
| 4-adder | fulladder | 7 | 2 |
| 4-adder | fulladder | 12 | 2 |
| 4-adder | halfadder | 17 | 2 |
| fulladder | nand | 1 | 3 |
| fulladder | nand | 1 | 1 |
| fulladder | nand | 3 | 3 |
| fulladder | xor | 2 | 3 |
| fulladder | xor | 3 | 1 |
| halfadder | and | 1 | 1 |
| halfadder | xor | 2 | 1 |

3. The relation `WORLDECON(COUNTRY, GNP, POPULATION)` gives 1965 data for gross national product and population for each country in the world. Use the domain algebra, and, if necessary, the relational algebra

   (a) to compute the per-capita GNP for each country (*i.e.*, the GNP per person);

   (b) to distinguish *Very Poor*, *Poor*, *Middle-Income* and *Rich* countries on the basis of, respectively, less than \$100 per capita, \$100–\$249 per capita, \$250–\$749 per capita and more than \$749 per capita;

   (c) to find the following ratios for each of the four categories from (b): (total GNP for category)/(total GNP for world); (total POPULATION for category)/(total POPULATION for world);

   (d) to find, assuming the Rich countries are growing economically at 3% per year, the set of poorest (measured by GNP per capita) countries whose total GNP sums to the annual increase of the Rich countries (or just below).

4. How would you represent and multiply two Boolean matrices using relations and Aldat? Give two different ways, (a) and (b). (So this question has *two* parts.)

$$(AB)_{ik} = \mathbf{OR}_j(A_{ij}\mathbf{and}B_{jk})$$

$$\text{e.g., } \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

5. A relation RECIPE(DISH, CATEGORY, INGREDIENTS) gives all the ingredients in a dish, and the category of the dish. For exampl, the dish *Coupe canadienne* belongs to the category *dessert* and has ingredients *vanilla ice cream, maple syrup, orange juice*, and *butter*: it thus occupies four tuples in RECIPE.

   Give Aldat code (a) to find all dishes with four ingredients or less, and (b) to find all dishes that require all the ingredients in their categories.

6. A closed region in a two-dimensional diagram can be represented as a sequence of points, *Chain*(*seq, X, Y*), or as a set of edges, *Edges*($X_1, Y_1, X_2, Y_2$). Write relational and/or domain algebra to make the following conversions.

   (a) Given *Chain*, derive *Edges*.

   (b) Given *Edges* and assuming the region is a triangle, derive *Chain*.

   For bonus marks, do not assume the region is a triangle. (But do not work on this problem at the expense of the other questions.)

7. Given the relations *Accounts*(*customer, acctId*) and *Loans*(*customer, loanId*), write relational and/or domain algebra so the bank can find all customers who have more loans than accounts. (*customer* is not the key of either of these relations. *acctId* is numerical with "–"s. *loanId* is numerical with a leading letter.)

8. How would you implement the $\sigma$-join, **sup**, using only **ijoin** and counting? Show the code for a simple example.

9. Given the relations *RunOn*(*Train, Track*) and *LongTrack*(*Track*), write relational algebra and domain algebra code to answer the following queries.

   (a) Find trains that do not run on any long tracks.

   (b) Find trains that do not run on all long tracks.

   (c) Find the total number of different trains.

   (d) Find the number of trains for each track.

   (e) Find the average number of trains per track. (Do this in two different ways.)

   Example data for *RunOn* might start

| *Train* | *Track* |
|---|---|
| 1 | A |
| 1 | B |
| 2 | A |

10. A bus timetable lists routes, each being a cycle of stops. For instance, four routes might be: route 1 is the cycle A B C A; route 2 is D E B F B E D; route 3 is D G H D; and route 4 is H E A E H.

    Read *all* of the rest of the question before commencing.

    (a) Design a relational representation for such a bus timetable. Show the above example as data, adding reasonable relative times for each route (starting with time 0 at the first stop). What are the functional dependences and keys?

    (b) Requests come in simultaneously from three travellers who want to know how to go from G to C, from B to D, and from E to G, respectively. Use the relational and domain algebras to answer these requests, minimizing route changes, and giving the bus route(s) and the stops at which the passengers must change routes. (Omit the first request, G to C, if it is taking you too long.)

    (c) Your solution need not be general, but you should comment on the issues arising in solving the problem for any number of changes. *Hint.* Do it step-by-step: start with solving a request with no changes, then go to one change, then two, etc.

(d) In the case of more than one solution to a request, how would you pick the one taking least time? (Assume equal delays for all bus changes.)

Wherever you do relational algebra, show the result relations, based on your sample data.

11. A software analysis program generates a relation,

*FunctionStats*(*UserFunc, UsedFunc, LineNo*)

which records for each function (*UserFunc*) in the program all the functions it calls (*UsedFunc*) and the line of code (*LineNo*) where each is invoked. (The same *UsedFunc* may be called several times by a *UserFunc*, but we assume that this does not happen more than once per line.)

Write expressions and statements

(a) to find how many different *UsedFunc*s are called directly by each *UserFunc*;

(b) to find *UserFunc*s that call `sqrt` and at least two other *UsedFunc*s;

(c) to find how many times each *UserFunc* calls each *UsedFunc* directly; and

(d) to find how many times each *UserFunc* calls each *UsedFunc* indirectly.

12. The software analysis program generates another relation,

*Signatures*(*Function, FormalParam, FPType, FPPos*)

which records for each function (*Function*) in the program all of its formal parameters and their types. *FPPos* gives the position of the parameter in the parameter list of the function.

One function is *similar* to another if they have the same "bag" of formal parameter types. (A "bag", or "multiset", is a set with duplicates considered.) Thus, (a:**intg**, b:**real**, c:**intg**) is similar to (u: **real**, b:**intg**, q:**intg**) but not to (x:**intg**, y:**real**) or to (a:**intg**, b:**intg**, c:**intg**).

Write expressions and statements to find pairs of similar functions.

13. A relation, $PTS(A, B)$, describes pairs of points on a map which are directly connected together somehow, for instance by roads. $A$ and $B$ are attributes of type integer. (So there is no problem about round-off error in comparisons.) Use the relational and domain algebras to formulate the following queries.

(a) For each $A$, how many $B$s are connected to it?

(b) Find all $A$s connecting to over 1% of all $B$s.

(c) Find a relation, $PAIR(C, D)$, giving pairs of directly connected points, *either A to B, or B to A*.

(d) From *PAIR*, find pairs of *indirectly* connected points.

(What are the names of the last two operations?)

14. Given the relations *Orders*(*Store, Part, Qty*) and *Products*(*Part, Price*), write Aldat code to find the total cost to each store which has ordered all parts. Assume the functional dependences *Store, Part* $\rightarrow$ *Qty* and *Part* $\rightarrow$ *Price*.

Invent some data to check your code.

15. Consider relations that have *weights* associated with each tuple (which could be interpreted as the probability that the tuple is true, or the certainty with which it is known, etc.). A weight has a value between `0` and `1`. The weight of any absent tuple is always taken to be `0`. Ordinary relations are the special case where all tuples have weight `1`.

Weights combine in the following ways 3

| | | |
|---|---|---|
| *and*, $\cap, \wedge$: | tuple 1 *and* tuple 2 | $w_1 \times w_2$ |
| *or*, $\cup, \vee$: | tuple 1 *or* tuple 2 | $1 - (1 - w_1) \times (1 - w_2)$ |
| *not*, $\sim, \neg$: | *not* tuple 1 | $1 - w_1$ |

Furthermore, universal quantification, $\forall$, is the conjunction (*and*) of all the associated weights, and existential quantification, $\exists$ is the disjunction (*or*) of them all.

For relations $R(W, X, w_R)$, $S(Y, Z, w_S)$, and $T(Y, Z, w_T)$, show how to compute the following in Aldat.

(a) The union, $ST(Y, Z, w_{ST})$, of $S$ and $T$.

(b) The natural join, $RS(W, X, Y, Z, w_{RS})$, of $R$ and $S$ on $X$ and $Y$.

(c) The projection, $RX(X, w_{RX})$, of $R$ on $X$.

E.g., union

| $S(Y$ | $Z$ | $w_S)$ |
|---|---|---|
| a | b | 0.8 |
| c | d | 0.5 |

| $T(Y$ | $Z$ | $w_T)$ |
|---|---|---|
| a | b | 0.7 |
| e | f | 0.9 |

| $ST(Y$ | $Z$ | $w_{ST})$ |
|---|---|---|
| a | b | 0.94 |
| c | d | 0.5 |
| e | f | 0.9 |

16. In a relation $R(X, Y)$, the *probability* $p_x$ of a value, $x$ of attribute $X$ is the fraction $nx/n$, where $nx$ is the number of times $x$ appears in $R$, and $n$ is the number of tuples in $R$.

    The *entropy* of $X$ in $R$ is the sum over all values, $x$: $\sum_x p_x \ln p_x$.

    Write Aldat code, with at most *one* expression of the relational algebra, to calculate the entropy of $X$ in $R$, and with a second expression which calculates the entropy of $Y$ in $R$.

17. A polygon, in a large map or design, can be represented in various ways. For instance

    - a sequence of points
    - a set of edges
    - a set of triangles

    (where, in a sequence, order is important and duplicates allowed, while, in a set, order is unimportant and duplicates are not permitted).

    Answer the following questions in order: they depend on each other.

    (a) Devise relational representations for each of the above three, showing keys and functional dependences, and giving sufficient sample data (but no more) to show that other dependences do not hold. Draw, roughly, the polygons corresponding to your data.

    (b) Write relational and domain algebra statements for two algorithms, each mapping one of the above representations to one other, so that all three representations are connected. (So, you might give an algorithm mapping point sequences to edge sets, and an algorithm mapping triangle sets to edge sets.)

    (c) What is the worst-case complexity of each of these algorithms? (Read the next question before answering.)

    (d) What is the worst case cost of your relational implementations? (Assume some specific implementation of relations and relational operators in answering this question, and describe the implementation briefly.)

18. The relation *Evolution*(*Eat, Eaten*) describes what eats what in the food chain. For example, rabbits eat lettuce, foxes eat rabbits, humans eat lettuce, humans eat rabbits, for instance. In three different ways, write Aldat code to find the omnivores, i.e., species that eat all foods:

    (a) using $\sigma$-joins;

    (b) using QT-selectors only;

    (c) using neither of the above.

19. The relation of the last question is extended by the attribute *How*, which describes how *Eat* eats the *Eaten*. For example, humans eat lettuce in salads, humans eat lettuce in sandwiches, etc. How do the three answers above change, if extra projections are not allowed? (This is a one-part question.)

20. A value-added tax (GST) taxes items each time they are sold. For this question, suppose the tax rate is 1%. Write Aldat code to determine the effective tax rate on manufactured goods.

    *For example*, an integrated circuit costs \$1 and so is taxed at \$0.01. Five of them make up a memory board, which sells at \$10, and so is taxed at \$0.10. Five memory boards make up a computer for \$100 (tax \$1). Five computers go into a control system (\$1000, tax \$10). Supposing no other taxable components, the integrated circuit has provided \$0.01 in tax, the memory board \$0.15, the computer \$1.75, and the control system \$18.75. So the effective rates are 1%, 1.5%, 1.75%, and 1.875% respectively.

21. Give Aldat expressions to answer the following queries on the five relations shown below.

    (a) List the names of students whose CGPA $> 3.0$ and who work in the grounds crew. (N.B. 'A' counts 4 towards the CGPA, 'B' counts 3, .. 'F' counts 0.)

    (b) List the names of students whose CGPA $> 3.0$ or who work in the grounds crew.

    (c) Find each course offering taught to fewer than 4 students. List course numbers and semesters.

    (d) List names of students who have no jobs.

    (e) List names of students who have received all possible grades. (A, B, C, D, F)

    (f) List names of students who have received all possible grades in a single semester.

    (g) List names of students who have taken all courses Prof. Black teaches.

    (h) List names of professors whose advisees collectively work on all jobs.

| Student( | SID | SName) | StuJob( | SID | Job) | Course( | CNo | Prof) |
|---|---|---|---|---|---|---|---|---|
| | 111 | Nancy | | 111 | Grounds | | CS100 | Adams |
| | 222 | John | | 333 | TA | | CS110 | Adams |
| | 333 | Ann | | 333 | RA | | CS450 | Black |
| | 444 | Mark | | 444 | RA | | CS550 | Black |
| | 555 | Jill | | 555 | Grounds | | CS560 | Adams |
| | | | | 555 | TA | | | |

| Advise( | Prof | SID) |
|---|---|---|
| | Adams | 111 |
| | Adams | 222 |
| | Black | 333 |
| | Black | 444 |
| | Black | 555 |

| StuCour( | SID | GD | CNo | Semester) |
|---|---|---|---|---|
| | 111 | F | CS100 | Spring94 |
| | 111 | B | CS100 | Summer94 |
| | 111 | A | CS100 | Fall94 |
| | 111 | D | CS110 | Fall94 |
| | 111 | C | CS110 | Winter95 |
| | 222 | B | CS100 | Fall94 |
| | 333 | A | CS450 | Fall94 |
| | 333 | A | CS550 | Winter95 |
| | 444 | A | CS100 | Fall94 |
| | 444 | A | CS450 | Winter95 |
| | 555 | A | CS100 | Fall94 |
| | 555 | B | CS450 | Winter95 |

22. An extended Horn clause representation adds *confidence factors* to the rules and to the facts, as follows.

| *HORN*(*Rconf* | *R#* | *Ante* | *Concl*) | *FACTS*(*Fconf* | *Concl*) |
|---|---|---|---|---|---|
| ... | ... | ... | ... | ... | ... |
| 0.7 | 2 | bird | duck | 0.8 | bird |
| 0.7 | 2 | swims | duck | 0.6 | swims |
| 0.7 | 2 | brown | duck | 0.9 | brown |

When the rule is fired by the facts , its confidence factor, *Rconf* (which is the same for every tuple of the rule), is multiplied by the *minimum* of the confidence factors, *Fconf*, of the facts that match the antecedents.

Read all of the following questions before starting any. Do each part in a way that you can use in the next part.

(a) Write Aldat code to fire a single rule without taking the confidence factors into account.

(b) Write Aldat code which fires all rules for which *Facts* has matching antecedents, and computes the resulting confidence factor. (Thus, "duck", with a confidence factor of 0.42 would be the result of the above single rule.) This code fires all rules at one level, but does not go on to make the consequent deductions.

(c) Write Aldat code for an inference engine which makes all possible deductions by generating new facts, then using these to fire more rules.

23. A tree can be represented as a binary relation, *Tree2*(*Sr, Jr*), or as a quaternary relation, *Tree4*(*self, parent, sibling, child*), as shown by the following example.

| *Tree2*(*Sr* | *Jr*) | *Tree4*(*self* | *parent* | *sibling* | *child*) |
|---|---|---|---|---|---|
| CARS | FORD | 0 | 5 | -1 | 2 |
| CARS | GM | 1 | -1 | -1 | 4 |
| FORD | FAIRLANE | 2 | 0 | -1 | -1 |
| FORD | MUSTANG | 3 | 4 | 6 | -1 |
| GM | BUICK | 4 | 1 | -1 | 3 |
| BUICK | CENTURY | 5 | 1 | -1 | 0 |
| | | 6 | 4 | -1 | -1 |

The connection between the strings and the integers in these two representations is that the integers are synonyms for the strings and that the numerical order of the integers gives the alphabetical order of the strings, and vice-versa: (`BUICK, 0`), (`CARS, 1`), (`CENTURY, 2`), (`FAIRLANE, 3`), (`FORD, 4`), (`GM 5`), (`MUSTANG, 6`).

The point of this question is to generate *Tree4* from *Tree2*. Note that it has four parts, equally marked. Steps b, c, d may be done in any order.

(a) Put the two attributes of *Tree2* together and generate an attribute which is the position of the string in sorted order.

(b) Generate attributes *self, parent, sibling, child* for the internal nodes of the tree.

(c) Generate attributes *self, parent, sibling, child* for the root node(s) of the tree.

(d) Generate attributes *self, parent, sibling, child* for the leaf nodes of the tree.

And, of course, put them all together to generate *Tree4*.

Hint: draw *Tree4* with corresponding strings replacing the integers, in order to understand exactly what its attributes mean, particularly *child* and *sibling*.

(a) Design a relational representation for an array such as the following, to be used to answer the query in the next question.

| | $B$ | | |
|---|---|---|---|
| $A$ | 1 | 2 | 3 |
| 1 | 7 | 0 | 4 |
| 2 | 6 | 5 | 0 |
| 3 | 0 | 8 | 0 |

(b) Write Aldat statements to calculate the "prefix sum" of all the elements in the array: the prefix sum of any element is the sum of all elements in the rectangle above it and to its left, including itself and the (1,1) element.
For the above example, the result would be

| | $B$ | | |
|---|---|---|---|
| $A$ | 1 | 2 | 3 |
| 1 | 7 | 7 | 11 |
| 2 | 13 | 18 | 22 |
| 3 | 13 | 26 | 30 |

Hint: it can be done in two steps.

(c) Say how you would generalize this to an array of not two but any number of dimensions. (Showing the code for three dimensions would help.)

(d) Discuss the running cost of this general case, for, say, $d$ dimensions.

24. In a hippie society, both men and women may have multiple partners, but the children are dependent on their biological parents.

(a) Write Aldat code to find all man($M$)-woman($W$) pairs such that each has at least two partners.

(b) Write Aldat code to find all man($M$)-woman($W$) pairs such that each has exactly the same number of partners.

(c) Write Aldat code which records the marriage of `z` and `a` in *Hippie* (retaining hippie social conventions). Write code which gives a new child, `6`, to `y` and `c`.

*Hippie*

| ($M$ | $W$ | $C$) |
|---|---|---|
| x | a | 1 |
| x | a | 2 |
| x | b | 3 |
| y | b | 4 |
| y | c | $\mathcal{DC}$ |
| z | d | 5 |

25. Limiting your use of relational algebra to a single statement containing only projections, create a *Summary* such as that shown of the physics *Experiment* in which particles of different types are detected to nanosecond accuracy (shown measured in units of microseconds). The summary must report for the latest microsecond interval on the counts of each type during that microsecond and on all the counts up to the start of the interval.

| *Experiment*(*Time* | *Type*) | *Summary*(*MicroSec* | *Type* | *Count*) |
|---|---|---|---|---|
| 135.063 | electron | 137 | electron | 1 |
| 135.079 | electron | 137 | photon | 1 |
| 136.102 | photon | prev | electron | 2 |
| 137.003 | electron | prev | photon | 1 |
| 137.011 | photon | | | |

You may suppose that your code is to be run exactly once, so that you cannot optimize by re-using previous results.

26. Re-write your code from question 25 supposing that it is to be run *every* microsecond and must avoid recalculating `prev` from scratch each time. Assume *Summary* already exists for the previous interval. Reuse any domain algebra you can from question 25, but write separate relational algebra statements for each selection and join. (Write only the code needed to evaluate *Summary* once, but in such a way that it could easily be incorporated into the repeated calculation, assuming fast enough hardware and compiler.)

27. The following relation represents a text as a hierarchy of sequences of words.

| *Text* (*paraseq* | *sentseq* | *word* | *wseq*) |
|---|---|---|---|
| 1 | 1 | The | 1 |
| 1 | 1 | quick | 2 |
| : | : | : | : |
| 1 | 2 | Blowsy | 1 |
| 1 | 2 | night-frumps | 2 |
| : | : | : | : |
| 2 | 1 | letter | 4 |
| 2 | 1 | tests | 5 |

Write general code:
a) to give the absolute word sequence, *wabs*, so that the text could be represented as the projection on *word* and *wabs* without losing word order; to find the *wabs* value, and the corresponding word, for the first and last word of each sentence; and
b) to convert *Text* into the nested relation

*nestText*(*paraseq*,*Paras*(*sentseq*,*Sents*(*word*,*wseq*)))

3. Write code to show violations of functional dependence in a relation. For example, applied to $R(A, B)$, shown, your code might extract the
28. (`2,y`) tuple when testing for the functional dependence $A \rightarrow B$. (It will be sufficient for you to write code to test for $A \rightarrow B$, but for any relation, $R(A, B, C, ..)$.)

| $R$($A$ | $B$) |
|---|---|
| 1 | x |
| 2 | x |
| 2 | y |
| 3 | y |

The author gratefully acknowledges support from the taxpayers of Québec and of Canada who have paid his salary and research grants while this work was developed at McGill University, and from his students (who built the implementations and investigated the data structures and algorithms) and their funding agencies.