Exercises for *Temporal Data and The Relational Model*

These exercises were not devised for CS319. They were devised for the book on which this part of CS319 is based. Nevertheless, most of them seem suitable for CS319 students. Some of them obviously take too long to be usable in an exam but you might still benefit from having a go at them.

Part 1

Given these **Tutorial D** relation variable definitions for a database called "Students and Courses":

```
VAR STUDENT REAL RELATION { S# S#, SNAME CHAR, REGISTERED DATE }
   KEY { S# } ;
```
Predicate: "Student S#, named SNAME, registered with the university on date REGISTERED."

```
VAR course REAL RELATION { C# C#, CTITLE CHAR, AVAILABLE DATE } ;
   KEY { C# } ;
```
Predicate: "Course C#, titled CTITLE, has been available at the university since date AVAILABLE."

```
VAR ENROLMENT REAL RELATION { S# S#, C# C#, ENROLLED DATE }
   KEY { S#, C# }
   FOREIGN KEY { S# } REFERENCES STUDENT
   FOREIGN KEY { C# } REFERENCES COURSE ;
```
Predicate: "Student S# enrolled on course C# on date ENROLLED."

Selector operators S# and C#, each having a single parameter of type CHAR, are available for types S# and C#, respectively. Thus, `S# ( 'S1' )` yields the student number S1, for example.

1. Write a **Tutorial D** CONSTRAINT expression, using IS_EMPTY, to the effect that no student can be enrolled on a course prior to (a) that course becoming available and (b) that student's registration.

2. Write a **Tutorial D** relational expression whose result shows the student number and name of every student who is enrolled on all the courses that student S2 is enrolled on.

3. Consider the following **Tutorial D** expression:

```
( ( STUDENT WHERE S# = S# ( 'S1' ) ) JOIN COURSE ) WHERE AVAILABLE >
      REGISTERED
```

   (a) List the candidate keys of the resulting relation.

   (b) Express the meaning of the expression in the form of a predicate.

Chapter 5

Consider the following revised and extended version of the Students and Courses database, noting the change of key for STUDENT:

```
VAR STUDENT REAL RELATION { S# S#, SNAME CHAR, REGISTERED DATE }
   KEY { S#, REGISTERED }
```
Predicate: "Student S#, named SNAME, registered with the university on date REGISTERED."

```
VAR UNREG_STUDENT REAL RELATION { S# S#, UNREGISTERED DATE }
   KEY { S#, UNREGISTERED } ;
```
Predicate: "Student S# left the university on date UNREGISTERED."

```
VAR CANCELLED_COURSE REAL RELATION { C# C#, CANCELLED DATE }
   KEY { C# }
   FOREIGN KEY { C# } REFERENCES COURSE ;
```
Predicate: "Course C# ceased to be available on date UNAVAILABLE."

```
VAR COMPLETED_COURSE REAL RELATION { S# S#, C# C#, COMPLETED DATE,
   GRADE GRADE }
   KEY { S#, C# }
   FOREIGN KEY { C# } REFERENCES COURSE ;
```
Predicate: "Student S# completed course C# on date COMPLETED, achieving grade GRADE."

Grades are represented by integers in the range 1 to 5. Selector operator GRADE, having a single parameter of type INTEGER, is available for type GRADE. Thus, `GRADE ( 2 )` yields the grade 2, for example.

1. What additional constraints (expressed in natural language) might be required, assuming that this database constitutes a record of the relevant part of a typical university's business?

2. The following relvar declaration is added (with the intention of replacing the relvar COMPLETED_COURSE):

```
VAR STUDIED REAL RELATION { S# S#, C# C#, DURING INTERVAL_DATE,
                            GRADE GRADE }
   KEY { S#, C# }
   FOREIGN KEY { C# } REFERENCES COURSE ;
```
Predicate: "Student S# studied course C# during the period ENROLLED, achieving grade GRADE."

   Write a **Tutorial D** relation expression, referencing relvars ENROLMENT and COMPLETED_COURSE, whose result is consistent with the predicate given for studied (and can therefore usefully be assigned to that relvar).

3. Write a **Tutorial D** declaration for a relvar, COURSE_AVAILABILITY, that combines relvars COURSE and CANCELLED_COURSE analogously to the way STUDIED combines ENROLMENT and COMPLETED_COURSE. Include at least one KEY clause in the declaration and any appropriate FOREIGN KEY clauses.

Chapter 6

1. Let *i* be a value of type INTERVAL_INTEGER. Write an expression denoting the interval resulting from extending *i* by its own length in both directions. Under what circumstances will evaluation of this expression fail at run-time?

2. Write an expression denoting the interval representing the middle third of a given interval, *i*. You may assume that `COUNT(i)` is divisible by 3.

3. Let *i1, i2* and *i3* be intervals such that there is a single interval, *i4,* consisting of every point that is in either *i1, i2* or *i3*. Write an expression, using operators defined in this Chapter, that when evaluated yields *i4*. (Beware of the trap!)

4. Given the relvar `STUDIED` as defined in Exercise 2 for Chapter 5, write a **Tutorial D** relational expression whose result shows for every grade the average length of study for all students who achieved that grade. (We assume for this exercise that the university's courses are "self-study" ones, completed in students' own time.)

5. Write a **Tutorial D** relational expression, referencing relvars `STUDENT` and `ENROLMENT`, that when evaluated yields the relation pairing the student number of each student that has enrolled on at least two courses with the interval from that student's registration date to the date of their second enrolment. Don't forget that several enrolments might occur on the same date.

Chapter 7

1. Let `MOD3` be the type whose values are the integers 0, 1 and 2. Consider the type `RELATION { DURING INTERVAL_MOD3 }`. How many relations *xr* of this type satisfy the condition `EXPAND ( xr ) = xr`? List every relation *cr* of this type satisfying the condition `COLLAPSE ( cr ) = cr`.

2. Write a **Tutorial D** relational expression, referencing relvars `STUDENT, UNREG_STUDENT` and `STUDIED` as defined for the Exercises of Chapter 5, that when evaluated yields the unary relation showing intervals, no two of which overlap or abut, throughout which student S1 was registered but studying no courses. You may assume that student S1 has completed all the courses on which she enrolled.

Chapter 8

1. Consider the type `RELATION { A1 INTERVAL_T, A2 INTERVAL_T }`. Assuming *T* to be a type consisting of integers in the range 1 to *n,* find the smallest value for *n* such that relations *r1* and *r2* of the given type exist that satisfy the following conditions:

   (a) `r1 ≠ r2`

   (b) `UNPACK r1 ON ( A1, A2 ) = UNPACK r2 ON ( A1, A2 )`

   (c) `r1 ≠ PACK r1 ON ( A1, A2 )`

   (d) `r2 ≠ PACK r2 ON ( A1, A2 )`

   (e) `COUNT ( r1 ) = COUNT ( r2 )`

   (f) There does not exist a relation *r3* such that `UNPACK r1 ON ( A1, A2 ) = UNPACK r3 ON ( A1, A2 ) AND COUNT ( r3 ) < COUNT ( r1 )`.

   [Warning: it took HD several hours to arrive at a wrong solution, then several more to arrive at the right one!]

Chapter 9

1. Give a **Tutorial D** relational expression, referencing relvars `STUDENT` and `UNREG_STUDENT` as defined in the Exercise for Chapter 5, that when evaluated yields the entire student registration history of the university. The heading of this result should be `{ S# S#, SNAME CHAR, REGDURING INTERVAL_DATE }`. The value of `END(REGDURING)` for current registrations should be that given by `LAST_DATE()`.

Chapter 10

1. Consider the following revised version of the Students and Courses database:

```
VAR CURRENT_STUDENT REAL RELATION ( S# S#, SNAME CHAR,
                                    REGISTERED DATE }
   KEY { S#, REGISTERED }
```
Predicate: "Current student `S#`, named `SNAME`, registered with the university on date `REGISTERED`."

```
VAR STUDENT_HISTORY REAL RELATION { S# S#, SNAME CHAR,
                                    REG_DURING INTERVAL_DATE }
   KEY { S#, REG_DURING } ;
```
Predicate: "Student `S#` was registered with the university throughout the period `REG_DURING`."

```
VAR CURRENT_COURSE REAL RELATION { C# C#, CTITLE CHAR,
                                   AVAILABLE DATE }
   KEY { C# };
```
Predicate: "Course `C#`, titled `CTITLE`, has been available at the university since date `AVAILABLE`."

```
VAR OLD_COURSE REAL RELATION { C# C#, CTITLE CHAR,
                               AVAILABLE_DURING INTERVAL_DATE }
   KEY { C# }
```
Predicate: "Course `C#`, titled `CTITLE`, was available throughout the period `AVAILABLE_DURING`."

```
VAR ENROLMENT REAL RELATION { S# S#, C# C#, ENROLLED DATE }
   KEY { S#, C# }
   FOREIGN KEY { S# } REFERENCES CURRENT_STUDENT
   FOREIGN KEY { C# } REFERENCES CURRENT_COURSE ;
```
Predicate: "Student `S#` enrolled on course `C#` on date `ENROLLED`."

```
VAR COMPLETED_COURSE REAL RELATION { S# S#, C# C#,
                                     STUDIED_DURING INTERVAL_DATE,
                                     GRADE GRADE }
   KEY { S#, C# }
```
Predicate: "Student `S#` studied course `C#` throughout the period `STUDIED_DURING`, achieving grade `GRADE`."

For each relvar, state whether or not it is in 6NF. If it is not in 6NF, identify any problems that might be solved by decomposing it into two or more 6NF relvars.

2. Give a **Tutorial D** expression that when evaluated yields a relation showing, for each student, the number of course completed during each period of registration for that student. What database design changes might you consider as a result of this exercise?

3. Assume that for each course there are zero or more offerings, each taking place over a given period of time. Some offerings have taken place in the past; others are currently taking place but have a scheduled completion date; others are scheduled to start and end at some time in the future.

   When students enrol for courses, they have to specify which offering they are enrolling for. Each offering has a specified quota, which the number of students enrolled for that offering must not exceed.

   Give an appropriate **Tutorial D** declaration and predicate for the relvar `COURSE_OFFERING`, reflecting the given requirements.

Chapter 11

1. Consider the revised Students and Courses database used in the Exercises for Chapter 10, including the following declaration for the relvar `COURSE_OFFERING` asked for in Exercise 3:

```
VAR COURSE_OFFERING REAL RELATION { C# C#,
                                    O# POSINT,
                                    QUOTA POSINT,
                                    OFFERED_DURING INTERVAL_DATE }
   KEY { C#, O# }
```

   Predicate: "The O#-th offering of Course `C#` took place or is scheduled to take place during the period `DURING`."

   Where `POSINT` is a type whose values are the integers greater than zero.

   Revise the database definition again to include such `PACKED ON` constraints, `WHEN … THEN` constraints and U_key constraints as you think necessary.

Chapter 12

1. Consider the revised Students and Courses database resulting from Exercise 1 of Chapter 11. What `SINCE_FOR` and `HISTORY_IN` constraints, if any, would you add?