CS252 Fundamentals of Relational Databases — Solutions to Worksheet 3
Working with a Database in *Rel*

1. *No questions asked.*

2. Execute a *Rel* `VAR` statement for each of `S, P` and `SP`.

```
VAR S BASE RELATION { S# CHAR, Sname CHAR,
                      Status INTEGER, City CHAR}
           KEY { S# } ;
VAR P BASE RELATION { P# CHAR, Pname CHAR,
                      Colour CHAR, Weight RATIONAL,
                      City CHAR}
           KEY { P# } ;
VAR SP BASE RELATION {S# CHAR, P# CHAR, Qty INTEGER}
            KEY { S#, P# } ;
```

"Populate" (as they say) each relvar with the values shown in Date's tables. There are several ways of achieving this. Choose whichever you prefer from the following:

a.
```
VAR S BASE RELATION { S# CHAR, Sname CHAR,
                      Status INTEGER, City CHAR}
           KEY { S# }
           INIT ( RELATION {
                    TUPLE { S# 'S1', Sname 'Smith',
                            City 'London',
                            Status 20 },
                    TUPLE { S# 'S2', etc. } }
                ) ;
```

b.
```
S := RELATION {
       TUPLE { S# 'S1', Sname 'Smith',
               City 'London', Status 20 },
       TUPLE { S# 'S2', etc. } } ;
```

c.
```
INSERT S RELATION {
           TUPLE { S# 'S1', Sname 'Smith',
                   City 'London', Status 20 } } ;
```

and repeat for each tuple to be inserted.

3. What, then, is the predicate for the expression `S JOIN SP JOIN P`?

Supplier `S#` is named `Sname`, has status `Status` and is located in city `City` and part `P#` is named `Pname`, is coloured `Colour`, weighs `Weight` and is located in city `City` and Supplier `S#` ships part `P#` in quantities of `Qty`.

What do you expect to be the result of that expression?

Rel gives this (with Enhanced not checked):

```
RELATION {S# CHAR, Sname CHAR, Status INTEGER, City CHAR, P# CHAR, Qty
INTEGER, Pname CHAR, Colour CHAR, Weight RATIONAL}
{ TUPLE {S# "S1", Sname "Smith", Status 20, City "London", P# "P1",
        Qty 300, Pname "Nut", Colour "Red", Weight 12.0},
    TUPLE {S# "S1", Sname "Smith", Status 20, City "London", P# "P4",
           Qty 200, Pname "Screw", Colour "Red", Weight 14.0},
    TUPLE {S# "S4", Sname "Clark", Status 20, City "London", P# "P4",
           Qty 300, Pname "Screw", Colour "Red", Weight 14.0},
    TUPLE {S# "S1", Sname "Smith", Status 20, City "London", P# "P6",
           Qty 100, Pname "Cog", Colour "Red", Weight 19.0},
    TUPLE {S# "S3", Sname "Blake", Status 30, City "Paris", P# "P2",
           Qty 200, Pname "Bolt", Colour "Green", Weight 17.0},
    TUPLE {S# "S2", Sname "Jones", Status 10, City "Paris", P# "P2",
           Qty 400, Pname "Bolt", Colour "Green", Weight 17.0}
   }
```

In tabular form (i.e., with Enhanced checked):

S#	**Sname**	**Status**	**City**	**P#**	**Pname**	**Colour**	**Weight**	**Qty**
S1	Smith	20	London	P1	Nut	Red	12.0	300
S1	Smith	20	London	P4	Screw	Red	14.0	200
S2	Jones	10	Paris	P2	Bolt	Green	17.0	400
S3	Blake	30	Paris	P2	Bolt	Green	17.0	200
S4	Clark	20	London	P4	Screw	Red	14.0	300
S1	Smith	20	London	P6	Cog	Red	19.0	100

What is its degree? 9

Does *Rel* give the result you expected? Explain what you see.

S and P both have an attribute named `City`, so this is a common attribute for matching purposes, as well as `S#` and `P#`. Note the two appearances of `City` in the predicate for the join. They must both stand for the same city.

4. Attempt to insert a tuple into `SP` with supplier number S1, part number P1 and quantity 100. Explain the result of your attempt.

   Rel gives this:

```
INSERT SP RELATION { TUPLE { S# 'S1', P# 'P1',  Qty 200 } } ;
ERROR: Inserting tuple would violate uniqueness constraint of KEY
 {S#, P#}
Line 1, column 62 near '100'
```

   The declaration of relvar SP includes the specification `KEY { S#, P# }`, which means the same as, for example:

```
CONSTRAINT SPkey COUNT(SP{S#,P#})=COUNT(SP);
```

   In other words, the cardinality of `SP` must always be the same as that of its projection over `S#` and `P#`. In other words, for any given combination of `S#` and `P#` values, there must be at most one tuple in `SP`. Successful insertion of `TUPLE { S# 'S1', P# 'P1',  Qty 200 }` would have resulted in `SP` containing two tuples with `S# = 'S1'` and `P# = 'P1'`, thus violating the constraint.

5. Get *Rel* to evaluate each of the following expressions. For each one, write down the corresponding predicate and also give an informal interpretation of the query in the style of those given in Exercise 5 below.

   a. `SP WHERE P# = 'P2'`

Supplier **`S#`** ships part **`P#`** in quantities of **`Qty`** and **P#** is P2.

Note that although we have fixed the value for P#, we haven't actually substituted P2 for P# in the predicate!

Get shipments of part P2.

b. `S { ALL BUT Status }`

There exists a status `Status` such that supplier **`S#`** is named **`Sname`**, has status `Status` and is located in city **`City`**.

Get all information about suppliers, apart from their status.

c. `SP { S#, Qty }`

There exists a part number `P#` such that Supplier **`S#`** ships part `P#` in quantities of **`Qty`**.

For each supplier, get the various quantities used for shipments.

d. `S MATCHING ( SP WHERE P# = 'P2' )`

Supplier **`S#`** is named **`Sname`**, has status **`Status`** and is located in city **`City`** and there exists a quantity `Qty` such that **`S#`** ships part P2 in quantities of `Qty`.

In these predicates the parameters (free variables) are shown in bold to distinguish them from the bound variable `Qty`. `Qty` is bound by use of the quantifier, "there exists". You can use the more formal mathematical notation for existential quantification if you prefer (warning: the symbol at the beginning of the next line, a backwards capital E, was obtained using a Windows font called Math c—it might not show up correctly on your computer if you do not have that font):

∃ `Qty` (Supplier **`S#`** is named **`Sname`**, has status **`Status`** and is located in city **`City`** and **`S#`** ships part P2 in quantities of `Qty`)

Get suppliers who supply part P2.

e. `P NOT MATCHING ( SP WHERE S# = 'S2' )`

Part **`P#`** is named **`Pname`**, is coloured **`Colour`**, weighs **`Weight`** and is located in city **`City`** and there does not exist a quantity `Qty` such that supplier S2 ships **`P#`** in quantities of `Qty`.

Get parts that supplier S2 cannot supply.

f. `S { City } UNION P { City }`

There exist a supplier number `S#,` a name `Sname`, and a status `Status` such that Supplier `S#` is named `Sname`, has status `Status` and is located in city **`City`**, or there exist a part number `P#,` a name `Pname`, a colour `Colour`, and a weight `Weight` such that part `P#` is named `Pname`, is coloured `Colour`, weighs `Weight` and is located in city **`City`**.

or, if you prefer, *(warning: the next few lines include certain mathematical symbols—backwards capital E and the v-shaped symbol for logical "or"—that obtained using a Windows font called Match c—they might not show up correctly on your computer if you do not have that font):*

∃S#∃Sname∃Status (supplier S# is named Sname, has status Status and is located in city **City**) ∨ ∃P#∃Pname∃Colour∃Weight (part P# is named Pname, is coloured Colour, weighs Weight and is located in city **City**)

Note the use of ∨ to signify disjunction ("or").

Get cities where either a supplier or a part is located.

```
g. S { City } MINUS P { City }
```

There exist a supplier number S#, a name Sname, and a status Status such that Supplier S# is named Sname, has status Status and is located in city **City**, and there do not exist a part number P#, a name Pname, a colour Colour, and a weight Weight such that part P# is named Pname, is coloured Colour, weighs Weight and is located in city **City**.

Get cities where a supplier is located but no part is located.

```
h. S { S#, City } COMPOSE P { P#, City }
```

There exist a city City, a name Sname, a status Status, a name Pname, a colour Colour, and a weight Weight such that Supplier **S#** is named Sname, has status Status and is located in city City, and part **P#** is named Pname, is coloured Colour, weighs Weight and is located in city City.

Sometimes it seems impossible to write an informal interpretation without having recourse to the kind of variable symbols we use in predicates. Hence:

Get <S#, P#> pairs such that supplier S# and part P# are located in the same city.

```
i. ( S RENAME ( City AS SC ) ) { SC } JOIN
        ( P RENAME ( City AS PC ) ) { PC }
```

There exist a supplier number S#, a name Sname, a status Status, a part number P#, a name Pname, a colour Colour, and a weight Weight such that Supplier S# is named Sname, has status Status and is located in city **SC**, and part P# is named Pname, is coloured Colour, weighs Weight and is located in city **PC**.

Get pairs of cities such that a supplier is located in one, a part in the other.

6. Write **Tutorial D** expressions for the following queries and get *Rel* to evaluate them:

a. Get all shipments.

```
SP
```

b. Get supplier numbers for suppliers who supply part P1.

```
S MATCHING ( SP WHERE P# = 'P1' ) { S# }
```

c. Get suppliers with status in the range 15 to 25 inclusive.

```
S WHERE Status > 14 AND Status < 26
```

d. Get part numbers for parts supplied by a supplier in London.

```
( SP JOIN ( S WHERE City = 'London' ) ) { P# }
```

e. Get part numbers for parts not supplied by any supplier in London.

```
P { P# } NOT MATCHING
  ( SP JOIN ( S WHERE City = 'London' ) )
```

f. Get city names for cities in which at least two suppliers are located.

```
( ( S {S#, City} RENAME(S# AS S#1)
  JOIN
  S {S#, City} RENAME(S# AS S#2) )
  WHERE S#1 < S#2 ) {City}
```

or you can use `SUMMARIZE`:

```
( ( SUMMARIZE S PER ( S { City } )
                ADD ( COUNT() AS No_of_Supps ) )
WHERE No_of_Supps > 1 ) { City }
```

g. Get all pairs of part numbers such that some supplier supplies both of the indicated parts.

```
( ( SP { S#, P# } RENAME ( P# AS Px ) )
  JOIN
  ( SP { S#, P# } RENAME ( P# AS Py ) )
) { Px, Py } WHERE Px < Py
```

The final restriction is optional. It assumes that only pairs of distinct part numbers are required, and that we do not want the result to include both `TUPLE { PX` *`px`*`, PY` *`py`* `}` and `TUPLE { PX` *`py`*`, PY` *`px`* `}` for any (*px*, *py*).

h. Get the total number of parts supplied by supplier S1.

```
COUNT ( SP WHERE S# = 'S1' )
```

But that expression yields a scalar. To obtain the result in the form of a relation, e.g.:

```
RELATION { TUPLE { Parts_from_S1 COUNT ( SP WHERE S# =
'S1' ) } }
```

or:

```
SUMMARIZE ( SP WHERE S# = 'S1' )
               PER ( TABLE_DEE )
               ADD ( COUNT() AS Parts_from_S1 )
```

i. Get supplier numbers for suppliers with a status lower than that of supplier S1.

```
( ( ( ( S WHERE S# = 'S1' ) { Status }
    RENAME ( Status AS S1_Status ) )
  JOIN S ) WHERE Status < S1_Status ) { S# }
```

j. Get supplier numbers for suppliers whose city is first in the alphabetic list of such cities.

```
( S WHERE City = MIN ( S, City ) ) { S# }
```

k. Get part numbers for parts supplied by all suppliers in London.

```
WITH ( S WHERE City = 'London' ) AS LS ,
     ( SP RENAME ( P# AS X ) )   AS T  :
( P WHERE (
   ( T WHERE X = P# ) { S# } ) <= ( LS { S# } ) )
{ P# }
```

Note the use of relation comparison (<= is Rel*'s notation for* ***Tutorial D****'s ⊆, "is a subset of"). Use of* `WITH` *is optional, of course. You might come up with a different* *solution, but does it address the possibility of there being no suppliers at all in London?*

l. Get supplier-number/part-number pairs such that the indicated supplier does not supply the indicated part.

```
        ( S { S# } JOIN P { P# } ) NOT MATCHING SP
```

m. Get all pairs of supplier numbers, S*x* and S*y* say, such that S*x* and S*y* supply exactly the same set of parts each.

The following solution, using relation comparison, appeals directly to the exercise's "the same set":

```
WITH
  ( S RENAME ( S# AS SX ) ) AS RX ,
  ( S RENAME ( S# AS SY ) ) AS RY :

( ( RX JOIN RY ) WHERE
                ( ( SP WHERE S# = SX ) { P# } )
                = ( ( SP WHERE S# = SY ) { P# } )
) { SX, SY } WHERE SX < SY
```

Note the importance of referencing S and not SP in the definitions of RX and RY. If we reference SP we might miss suppliers who supply no parts at all.

The final restriction is optional—see part g.

End of Solutions