

## Chapter 8

### I n t e g r i t y

```
a. TYPE CITY POSSREP ( CHAR )
   CONSTRAINT THE_CITY ( CITY ) = 'London'
      OR THE_CITY ( CITY ) = 'Paris'
      OR THE_CITY ( CITY ) = 'Rome'
      OR THE_CITY ( CITY ) = 'Athens'
      OR THE_CITY ( CITY ) = 'Oslo'
      OR THE_CITY ( CITY ) = 'Stockholm'
      OR THE_CITY ( CITY ) = 'Madrid'
      OR THE_CITY ( CITY ) = 'Amsterdam' ;
```

An obvious shorthand would be:

```
a. TYPE CITY POSSREP ( CHAR )
   CONSTRAINT THE_CITY ( CITY ) IN { 'London' , 'Paris'      ,
                                     'Rome'      , 'Athens'      ,
                                     'Oslo'      , 'Stockholm'  ,
                                     'Madrid'   , 'Amsterdam' } ;

b. TYPE S# POSSREP ( CHAR ) CONSTRAINT
   SUBSTR ( THE_S# ( S# ), 1, 1 ) = 'S' AND
   CAST_AS_INTEGER ( SUBSTR ( THE_S# ( S# ), 2 ) ≥ 0 AND
   CAST_AS_INTEGER ( SUBSTR ( THE_S# ( S# ), 2 ) ≤ 9999 ;
```

We assume here that the substring operator SUBSTR and the explicit conversion operator CAST\_AS\_INTEGER are both available.

```
c. CONSTRAINT C IS_EMPTY ( P WHERE WEIGHT ≥ WEIGHT ( 50.0 ) ) ;

d. CONSTRAINT D COUNT ( J ) = COUNT ( J { CITY } ) ;

e. CONSTRAINT E COUNT ( S WHERE CITY = 'Athens' ) ≤ 1 ;

f. CONSTRAINT F
   IS_EMPTY ( ( EXTEND SPJ ADD 2 * AVG ( SPJ, QTY )
              AS X ) WHERE QTY > X ) ;

g. CONSTRAINT G
   IS_EMPTY ( ( S WHERE STATUS = MIN ( S { STATUS } ) ) JOIN
              ( S WHERE STATUS = MAX ( S { STATUS } ) ) ) ;
```

Actually, the terms "highest status supplier" and "lowest status supplier" are not well-defined, since status values are

not unique. We have interpreted the requirement to be that if  $S_x$  and  $S_y$  are any suppliers with "highest status" and "lowest status," respectively, then  $S_x$  and  $S_y$  must not be colocated. Note that the constraint will necessarily be violated if the "highest" and "lowest" status are equal! -- in particular, it will be violated if there is just one supplier.

- h. CONSTRAINT H IS\_EMPTY ( J { CITY } MINUS S { CITY } ) ;
- i. CONSTRAINT I IS\_EMPTY ( J WHERE NOT ( TUPLE { CITY CITY } IN ( J { J# } JOIN SPJ JOIN S ) { CITY } ) ) ;
- j. CONSTRAINT J NOT ( IS\_EMPTY ( P WHERE COLOR = COLOR ( 'Red' ) ) ) ;

This constraint will be violated if there are no parts at all. A better formulation would be:

```
CONSTRAINT J IS_EMPTY ( P ) OR NOT ( IS_EMPTY
    ( P WHERE COLOR = COLOR ( 'Red' ) ) ) ;
```

- k. CONSTRAINT K IF NOT ( IS\_EMPTY ( S ) )
 THEN AVG ( S, STATUS ) > 18
 END IF ;

The IF test here is to avoid the error that would otherwise occur if the system tried to check the constraint when there were no suppliers at all.

- l. CONSTRAINT L IS\_EMPTY
 ( ( S WHERE CITY = 'London' ) { S# } MINUS
 ( SPJ WHERE P# = P# ( 'P2' ) ) { S# } ) ;
- m. CONSTRAINT M IS\_EMPTY ( P ) OR NOT
 ( IS\_EMPTY ( P WHERE WEIGHT < WEIGHT ( 50.0 ) ) ) ;
- n. CONSTRAINT N
 COUNT ( ( ( S WHERE CITY = 'London' ) JOIN SPJ ) { P# } ) >
 COUNT ( ( ( S WHERE CITY = 'Paris' ) JOIN SPJ ) { P# } ) ;
- o. CONSTRAINT O
 SUM ( ( ( S WHERE CITY = 'London' ) JOIN SPJ ), QTY ) >
 SUM ( ( ( S WHERE CITY = 'Paris' ) JOIN SPJ ), QTY ) ;
- p. CONSTRAINT P IS\_EMPTY
 ( ( SPJ JOIN ( SPJ' RENAME QTY AS QTY' ) )
 WHERE QTY > 0.5 \* QTY' ) ;
- q. CONSTRAINT Q IS\_EMPTY (
 ( S JOIN ( S' WHERE CITY = 'Athens' ) ) WHERE

```

        CITY =/ 'Athens' AND CITY =/ 'London' AND CITY = 'Paris' )
        AND IS_EMPTY (
( S JOIN ( S' WHERE CITY = 'London' ) ) WHERE
        CITY =/ 'London' AND CITY = 'Paris' ) ;

```

**8.2** The first two are type constraints, of course. Of the others, constraints C, D, E, F, G, J, K, M, P, and Q are relvar constraints, the rest are database constraints.

### 8.3

- a. CITY selector invocation.
- b. S# selector invocation.
- c. INSERT on P, UPDATE on part WEIGHT.
- d. INSERT on J, UPDATE on project CITY.
- e. INSERT on S, UPDATE on supplier CITY.
- f. INSERT or DELETE on SPJ, UPDATE on shipment QTY.
- g. INSERT or DELETE on S, UPDATE on supplier STATUS.
- h. INSERT on J, DELETE on S, UPDATE on supplier or project CITY.
- i. INSERT on J, DELETE on or SPJ, UPDATE on supplier or project CITY or shipment S# or J#.
- j. INSERT or DELETE on P, UPDATE on part CITY.
- k. INSERT or DELETE on S, UPDATE on supplier STATUS.
- l. INSERT on S, DELETE on SPJ, UPDATE on supplier CITY or shipment S# or P#.
- m. INSERT or DELETE on P, UPDATE on part WEIGHT.
- n. INSERT or DELETE on S or SPJ, UPDATE on supplier CITY or shipment S# or P#.
- o. INSERT or DELETE on S or SPJ, UPDATE on supplier CITY or shipment S# or P# or QTY.
- p. UPDATE on shipment QTY.
- q. UPDATE on supplier CITY.

**8.11** The referential diagram is shown in the figure below. Note that the database involves a referential cycle (there is a referential path from each of the two relvars to itself). Apart from this consideration, the database definition is essentially straightforward. We omit the details.

