CMSC 461 HW 1 Answers - The Relational Algebra and RDB

1 The Suppliers and Parts DB

1. The CITYs of Suppliers (S).
`project S over city;`

2. The PNAMEs of Parts (P).
`project P over pname;`

3. The PNAMEs of Parts that are Red.
`project (restrict P with color='Red') over pname;`

4. The SNUMs and SNAMEs of Suppliers located in "Paris".
`project (restrict S with city='Paris') over snum,sname;`

5. The SNUMs and PNUMs of Suppliers and Parts that are in the same CITY.
`project (join S with P on city) over snum,pnum;`

6. The SNAMEs of Suppliers that supply at least one Part.
`project (join S with SP on snum) over sname;`

7. The STATUSs of Suppliers that supply some Part in QTY greater than 250.
`project (join S with (restrict SP with qty>250) on snum) over status;`

8. The PNUMs of Parts that are supplied by more than one Supplier.
```
project (restrict (join (rename snum to snum1 in SP) with SP on pnum) with snum<>snum1)
over pnum;
```

9. The PNUMs and WEIGHTs of Parts that can be obtained from Suppliers in London.
```
project (restrict (join (join S with SP on snum) with P on pnum) with city='London') over
pnum,weight;
```

10. The SNUMs of Suppliers who supply all parts.
`project (project SP over snum,pnum / project P over pnum) over snum;`

11. All pairs of SNUMs such that the two Suppliers are located in the same CITY. Avoid listing suppliers as being located in the same city as themselves, and listing duplicate tuples (*e.g.* (Blake,Jones) and (Jones,Blake)).
```
project (restrict (rename snum to snum1 in (join S with S on city)) with (snum<>snum1)
and (snum<snum1)) over snum1,snum;
```
or
```
join s with s on city;
rename snum:2 to snum2 in it;
project (restrict it with snum>snum2) over snum, snum2;
```

12. The total quantity of part "P2" supplied by suppliers.
```
let tquan(x)=qty+x;
reduce (restrict SP with pnum='p2',tquan,0);
```

13. The average quantity of part "P2" supplied by suppliers.
```
let add1(x)=1+x;
reduce (restrict SP with pnum='p2',tquan,0)
it div reduce (restrict SP with pnum='p2',add1,0);
```

2 The DBB DB

1. The name of each drinker who frequents a bar which serves all of beers he likes.

   This one is a bit tricky. In the following solution, $t1 - t2$ is a set of tuples (drinker,bar,beer) such that the drinker frequents the bar, the drinker likes the beer, and the bar does **not** serve the beer. So, $t3$ represents the pairs of drinkers and bars in which the drinker frequents the bar but the bar fails to serve *some* beer the drinker likes. Thus, if we take these away from the frequents relation, we have our answer.

   ```
   let t1= join frequents with likes on drinker;
   let t2= join frequents with serves on bar;
   let t3 = project (t1-t2) over drinker,bar;
   project (frequents - t3) over drinker;
   ```

2. The name of each drinker who frequents a bar which serves at least one of the beers he likes.

   Note that we have to do a join over two attributes for this problem. TRIPLES is a relation over drinkers, bars and beers such that it contains a tuple (A,B,C) if drinker A frequents bar B, Bar B serves beer C, and person A likes beer C.

   ```
   let TRIPLES = (join (join likes with serves on beer)
                       with frequents
                       on drinker,bar)
   project TRIPLES over drinker;
   ```

3. The name of each drinker who frequents a bar which serves none of the beers he likes.

   We can make good use of the TRIPLES relation to answer this one.

   ```
   project (frequents  - (project TRIPLES over drinker,bar))
           over drinker;
   ```

4. The name of each drinker who frequents at least one bar; but none of the bars he frequents serves a beer that he likes.

   And again, the TRIPLES relation comes in handy.

   ```
   (project frequents over drinker) - (project TRIPLES over drinker);
   ```

3 The Family DB

1. grandparent(Gparent,Gchild)

   This is easiest to understand if you visualize it. Draw the family relation twice. In one copy rename the *child* attribute to x and in the other rename the *parent* attribute to x. Now think about joining the two copies overs X.

   ```
   join (rename child to x in family) with
        (rename parent to x in family)
        over x;
   project it over parent,child;
   let grandparent = rename parent,child to gparent,gchild in it;
   ```

2. greatgrand(Ggparent,Ggchild)

   Once you've understood the first problem, this is an easy extension to it.

   ```
   join (rename gchild to x in grandparent) with
        (rename parent to x in family)
        over x;
   project it over gparent,child;
   let greatgrand= rename gparent,child to ggparent,ggchild in it;
   ```

3. Generalize this to find the relation *ancestor(Anc,Person)*, where *Anc* is an ancestor of *Person*.

   A non general solution would be to start with the parent relation and then compute grandparent, greatgrandparent, etc. for some number of generations. The results can then be unioned together (with +) after having the attribute names suitably renamed to `anc` and `person`. This is the only way to do this kind of problem in most relational systems.

   A general solution can be obtained in rdb which depends on rdb's ability to write recursive functions. In the solution given below, the *gen* function takes an *ancestor*(*anc*, *person*) relation and tries to "extend" it one more generation by joining it with the base *family* relation and returning the union with the original relation after appropriate attribute renaming. The *anc* function recursively applies the *gen* function until it reaches a "fix point", i.e., no more changes are found.

   ```
   let gen(g)= g + project (join (rename anc to x in g) with
                                 (rename person to x in g)
                             on x)
                           over anc,person;

   let anc(g)= if g=gen(g) then g else anc(gen(g));

   let initial=rename parent,child to anc,person in family;

   let ancestor=anc(initial);
   ```