COMP3311 19T3 Database Systems

# Exercises 08
## Relational Algebra

**Notation:** in the relational schemas below, primary key attributes are shown in **bold** font, foreign key attributes are shown in *italic* font, and primary key attributes that are also foreign keys are shown in ***bold italic*** font.

Example:

```
Student(id, name, degreeCode)
Degree(code, name, requirements)
Subject(code, name, syllabus)
Marks(studentId, subjectCode, session, mark)
```

In their respective relations, the student id, the degree code and the subject code are primary keys. In the Student relation, the degree code is a foreign key. In the Marks relation, the three attributes student id, subject code and session together form the primary key; the first two (student id and subject code) are also foreign keys.

1. Relational algebra operators can be *composed*. What precisely does this mean? And why is it important?

   **Answer:**

   Composition means that the result of one operator can be used as an argument for some other operator. Since most relational algebra operators take relations as inputs and produce relations as results, they can be composed. It is important because it allows us to build arbitrarily complex relational algebra expressions to answer arbitrarily complex queries (i.e. we can build answers to complex queries from answers to simple queries).

   Note that we often find it clearer to express complex queries as a sequence of simple queries which are "assigned" to named intermediate relations, rather than as deeply nested expressions.

2. The natural join ( *R* Join *S* ) joins two tables on their common attributes. Consider a theta-join ( *R* Join[*C*] *S* ) where the join condition *C* is a conjunction of *R.A* = *S.A* for each attribute *A* appearing in the schemas of both *R* and *S* (i.e. it joins on the common attributes). What is the difference between the above natural join and theta join?

   **Answer:**

   The relation that results from the natural join has only one attribute from each pair of matching attributes. The theta-join has attributes for both, and their columns are identical. Both joins result in a relation with the same number of tuples.

3. The definition of relational algebra in lectures was based on *sets* of tuples. In practice, commercial relational database management systems deal with *bags* (or *multisets*) of tuples.

   Consider the following relation that describes a collection of PCs

   **PCs**

   | Model | Speed | RAM | Disk | Price |
   |---|---|---|---|---|
   | 1001 | 700 | 64 | 10 | 799 |
   | 1002 | 1500 | 128 | 60 | 2499 |
   | 1003 | 1000 | 128 | 20 | 1499 |
   | 1004 | 1000 | 256 | 40 | 1999 |
   | 1005 | 700 | 64 | 30 | 999 |

   Consider a projection of this relation on the processor speed attribute, i.e. `Proj[speed](PCs)`.

   a. What is the value of the projection as a set?
   b. What is the value of the projection as a bag?
   c. What is the average speed if the projection is a set?
   d. What is the average speed if the projection is a bag?
   e. Is the minimum/maximum speed different between the bag and set representation?

   **Answer:**

   a. {700, 1000, 1500}
   b. {700, 700, 1000, 1000, 1500}
   c. (700+1000+1500)/3 = 1067
   d. (700+700+1000+1000+1500)/5 = 980
   e. No. Unlike the average value, the minimum and maximum values are not affected by repetitions.

4. Consider the following two relations:

   **R**

   | A | B | C |
   |---|---|---|
   | a1 | b1 | c1 |
   | a1 | b2 | c2 |
   | a2 | b1 | c1 |

   **S**

   | B | C |
   |---|---|
   | b1 | c1 |
   | b2 | c2 |

Give the relation that results from each of the following relational algebra expressions on these relations:

a. *R* Div *S*

**Answer:**

*R* Div *S* =

| A |
|---|
| a1 |

b. *R* Div (Sel[B != b1](*S*))

**Answer:**

*Tmp1* = Sel[B != b1](*S*) =

| B | C |
|---|---|
| b2 | c2 |

*R* Div *Tmp1* =

| A |
|---|
| a1 |

c. *R* Div (Sel[B != b2](*S*))

**Answer:**

*Tmp1* = Sel[B != b2](*S*) =

| B | C |
|---|---|
| b1 | c1 |

*R* Div *Tmp1* =

| A |
|---|
| a1 |
| a2 |

d. *R* × *S*) - (Sel[R.C=S.C](*R* Join[B=B] *S*)

**Answer:**

*Tmp1* = *R* Join[R.B=S.B] *S* =

| R.A | R.B | R.C | S.B | S.C |
|---|---|---|---|---|
| a1 | b1 | c1 | b1 | c1 |
| a1 | b2 | c2 | b2 | c2 |
| a2 | b1 | c1 | b1 | c1 |

*Tmp2* = Sel[R.C=S.C](*Tmp1*) =

| R.A | R.B | R.C | S.B | S.C |
|---|---|---|---|---|
| a1 | b1 | c1 | b1 | c1 |
| a1 | b2 | c2 | b2 | c2 |
| a2 | b1 | c1 | b1 | c1 |

*Tmp3* = *R* × *S* =

| R.A | R.B | R.C | S.B | S.C |
|---|---|---|---|---|
| a1 | b1 | c1 | b1 | c1 |
| a1 | b1 | c1 | b2 | c2 |
| a1 | b2 | c2 | b1 | c1 |
| a1 | b2 | c2 | b2 | c2 |
| a2 | b1 | c1 | b1 | c1 |
| a2 | b1 | c1 | b2 | c2 |

*Tmp3* - *Tmp2* =

| R.A | R.B | R.C | S.B | S.C |
|---|---|---|---|---|
| a1 | b1 | c1 | b2 | c2 |
| a1 | b2 | c2 | b1 | c1 |
| a2 | b1 | c1 | b2 | c2 |

5. Consider two relations *R1* and *R2*, where *R1* contains *N1* tuples and *R2* contains *N2* tuples, and *N1* > *N2* > 0. Give the minimum and maximum possible sizes (in tuples) for the result relation produced by each of the following relational algebra expressions. In each case state any assumptions about the schemas of *R1* and *R2* that are needed to make the expression meaningful.

   a. *R1* Union *R2*
   b. *R1* Intersect *R2*
   c. *R1* - *R2*
   d. *R1* × *R2*
   e. Sel$_{[a=5]}$(*R1*)
   f. Proj$_{[a]}$(*R1*)
   g. *R1* Div *R2*

   **Answer:**

| **Expression** | **Assumptions** | **Min** | **Max** |
|---|---|---|---|
| *R1* Union *R2* | *R1* and *R2* are union-compatible | *N1*<br>(when *R2*⊂*R1*) | *N1+N2*<br>(when *R1*∩*R2*=∅) |
| *R1* Intersect *R2* | *R1* and *R2* are union-compatible | *0*<br>(when *R1*∩*R2*=∅) | *N2*<br>*(when R2⊂R1)* |
| *R1 - R2* | *R1* and *R2* are union-compatible | *N1-N2*<br>(when *R2*∩*R1*=*R2*) | *N1*<br>(when *R1*∩*R2*=∅) |
| *R1* × *R2* | None | *N1*N2* | *N1*N2* |
| Sel$_{[a=5]}$(*R1*) | *R1* has an attribute *a* | *0*<br>(no matches) | *N1*<br>(all match) |
| Proj$_{[a]}$(*R1*) | *R1* has an attribute *a*, *N1 > 0* | *1* | *N1* |
| *R1* Div *R2* | The set of *R2*'s attributes is<br>a subset of *R1*'s attributes | *0* | *floor(N1/N2)* |

6. (Date, exercise 6.11) Let *A* and *B* be two arbitrary relations. In terms of the keys of *A* and *B*, state the keys for each of the following RA expressions. Assume in each case that *A* and *B* meet the requirements of the operation (e.g. they are union-compatible for the Union and Intersect operations).

   a. Sel$_{[cond]}$(*A*), where *cond* is any condition

   **Answer:**

   Any subset of *A* inherits all of the candidate keys for *A*

   b. Proj$_{[attrs]}$(*A*), where *attrs* is any set of atributes

   **Answer:**

   If the projection includes any candidate key *K* of *A*, then *K* is a candidate key for the projection. Otherwise, the only candidate key is the combination of all projected attributes.

   c. *A* × *B*

   **Answer:**

   Every combination of a candidate key *KA* for *A* and a candidate key *KB* for *B* is a candidate key for the product.

   d. *A* Union *B*

   **Answer:**

   Even if two relations are union-compatible, it doesn't mean that they share the same candidate keys, so in the general case, the only candidate key for a union is the set of all attributes. In the special case where *A* and *B* are selections from the same relation, the candidate keys for the union would the the same as the candidate keys for the underlying relation.

   e. *A* Intersect *B*

   **Answer:**

Same considerations as for Union.

f. *A* - *B*

**Answer:**

Every candidate key for *A* is a candidate key for the difference (could be viewed as a special kind of selection from *A*).

g. *A* Div *B*

**Answer:**

Ultimately, division is like a selection followed by projection on the original table. Similar considerations would apply as for projection.

7. Consider the following relational schema:

```
Suppliers(sid, sname, address)
Parts(pid, pname, colour)
Catalog(supplier, part, cost)
```

Assume that the ids are integers, that `cost` is a real number, that all other attributes are strings, that the ***supplier*** field is a foreign key containing a supplier id, and that the ***part*** field is a foreign key containing a part id. Write a relational algebra expression to answer each of the following queries:

a. Find the *names* of suppliers who supply some red part.

**Answer:**

```
RedPartIds = Proj[pid](Sel[colour='red'](Parts))
RedPartSupplierIds = Rename[sid](Proj[supplier](RedPartIds Join Catalog))
Answer = Proj[sname](RedPartSupplierIds Join Suppliers)
```

b. Find the *sids* of suppliers who supply some red or green part.

**Answer:**

```
RedGreenPartIds = Proj[pid](Sel[colour='red' OR colour='green'](Parts))
Answer = Proj[sid](RedGreenPartIds Join Catalog)
```

c. Find the *sids* of suppliers who supply some red part or whose address is 221 Packer Street.

**Answer:**

```
RedPartIds = Rename[RedPartIds(part)](Proj[pid](Sel[colour='red'](Parts)))
RedPartSupplierIds = Rename[RedPartSupplierIds(sid)](Proj[supplier](RedPartIds Join Catalog))
PackerStSupplierIds = Proj[sid](Sel[address='221 Packer Street'](Suppliers))
Answer = RedPartSupplierIds Union PackerStSupplierIds
```

d. Find the *sids* of suppliers who supply some red part and some green part.

**Answer:**

```
RedPartIds = Proj[pid](Sel[colour='red'](Parts))
RedPartSupplierIds = Proj[sid](RedPartIds Join Catalog)
GreenPartIds = Proj[pid](Sel[colour='green'](Parts))
GreenPartSupplierIds = Proj[sid](GreenPartIds Join Catalog)
Answer = RedPartSupplierIds   Intersect   GreenPartSupplierIds
```

e. Find the *sids* of suppliers who supply every part.

**Answer:**

```
AllPartIds = Proj[pid](Parts)
PartSuppliers = Proj[sid,pid](Catalog)
Answer = PartSuppliers Div AllPartIds
```

f. Find the *sids* of suppliers who supply every red part.

**Answer:**

```
RedPartIds = Proj[pid](Sel[colour='red'](Parts))
PartSuppliers = Proj[sid,pid](Catalog)
Answer = PartSuppliers Div RedPartIds
```

g. Find the *sids* of suppliers who supply every red or green part.

**Answer:**

```
RedGreenPartIds = Proj[pid](Sel[colour='red' OR colour='green'](Parts))
Answer = PartSuppliers Div RedGreenPartIds
```

h. Find the *sids* of suppliers who supply every red part or supply every green part.

**Answer:**

```
RedPartIds = Proj[pid](Sel[colour='red'](Parts))
GreenPartIds = Proj[pid](Sel[colour='green'](Parts))
PartSuppliers = Proj[sid,pid](Catalog)
AllRedPartSuppliers = PartSuppliers Div RedPartIds
AllGreenPartSuppliers = PartSuppliers Div GreenPartIds
Answer = AllRedPartSuppliers Union AllGreenPartSuppliers
```

i. Find the *pids* of parts that are supplied by at least two different suppliers.

**Answer:**

```
C1 = Catalog
C2 = Catalog
SupplierPartPairs = Sel[C1.sid!=C2.sid](C1 Join[pid] C2)
Answer = Proj[C1.pid](SupplierPartPairs)
```

j. Find pairs of *sids* such that the supplier with the first *sid* charges more for some part than the supplier with the second sid.

**Answer:**

```
C1 = Catalog
C2 = Catalog
SupplierPartPairs = Sel[C1.sid!=C2.sid AND C1.cost>C2.cost](C1 Join[pid] C2)
Answer = Proj[C1.sid,C2.sid](SupplierPartPairs)
```

k. Find the *pids* of the most expensive part(s) supplied by suppliers named "Yosemite Sham".

**Answer:**

```
R1 = Proj[sid,pid,cost](Sel[name='Yosemite Sham'](Suppliers Join Catalog))
R2 = R1
R3 = Rename[1->sid,2->pid,3->cost](Sel[R2.cost<R1.cost](R1 × R2))
Answer = Proj[pid](R2 Minus Proj[sid,pid,cost](R3))
```

l. Find the *pids* of parts supplied by every supplier at a price less than 200 dollars (if any supplier either does not supply the part or charges more than 200 dollars for it, the part should not be selected).

**Answer:**

```
CheapParts = Proj[part,supplier](Sel[cost<200](Catalog))
AllSuppliers = Rename[AllSuppliers(supplier)](Proj[sid](Suppliers))
Answer = Proj[part,supplier](CheapParts Div AllSuppliers)
```

8. Using the Suppliers-Parts-Catalog schema from the previous question, state what the following relational algebra expressions compute:

a.
```
Proj[sname](
        Proj[sid](Sel[colour='red'](Parts))
        Join
        Sel[cost<100](Catalog)
        Join
        Suppliers
)
```

**Answer:**

Finds the names of all suppliers who supply red parts that cost less than $100.

b.
```
Proj[sname](
        Proj[sid](
                Sel[colour='red'](Parts)
                Join
                Sel[cost<100](Catalog)
                Join
                Suppliers
        )
)
```

**Answer:**

Produces nothing, because there is no `sname` field left to job after the `Proj[sid]`.

c.

```
Proj[sname](
        Sel[colour='red'](Parts)
        Join
        Sel[cost<100](Catalog)
        Join
        Suppliers
)
Intersect
Proj[sname](
        Sel[colour='green'](Parts)
        Join
        Sel[cost<100](Catalog)
        Join
        Suppliers
)
```

**Answer:**

Finds the names of all suppliers who supply both red and green parts that cost less than $100.

d.

```
Proj[sid](
        Sel[colour='red'](Parts)
        Join
        Sel[cost<100](Catalog)
        Join
        Suppliers
)
Intersect
Proj[sid](
        Sel[colour='green'](Parts)
        Join
        Sel[cost<100](Catalog)
        Join
        Suppliers
)
```

**Answer:**

Finds the sids of all suppliers who supply both red and green parts that cost less than $100.

e.

```
Proj[sid,sname](
        Proj[sid,sname](
                Sel[colour='red'](Parts)
                Join
                Sel[cost<100](Catalog)
                Join
                Suppliers
        )
        Intersect
        Proj[sid,sname](
                Sel[colour='green'](Parts)
                Join
                Sel[cost<100](Catalog)
                Join
                Suppliers
        )
)
```

**Answer:**

Finds the names and sids of all suppliers who supply both red and green parts that cost less than $100. The outermost projection is completely redundant.

9. Consider the following relational schema containing flight, aircraft and pilot information for an airline:

```
Flights(flno, from, to, distance, departs, arrives)
Aircraft(aid, aname, cruisingRange)
Certified(employee, aircraft)
Employees(eid, ename, salary)
```

The `Flights` relation describes a particular timetabled fight from a source (city) to a destination (city) at a particular time each week. Note that this schema doesn't model which specific aircraft makes the flight. The `Aircraft` relation describes individual planes, with the `aname` field containing values like `'Boeing 747'`, `'Airbus A300'`, etc. The `Certified` relation indicates which pilots (who are employees) are qualified to fly which aircraft. The ***`employee`*** field contains the *eid* of the pilot, while the ***`aircraft`*** field contains Finally, the `Employees` relation contains information about all of the employees of the airline, including the pilots.

Write, where possible, a relational algebra expression to answer each of the following queries. If it is not possible to express any query in relational algebra, suggest what extra mechanisms might make it possible.

a. Find the *ids* of pilots certified for 'Boeing 747' aircraft.

**Answer:**

Approach: collect together information about aicraft and certification and project out just the employee ids for tuples that involve 747's.

```
Answer = Proj[eid](Sel[aname='Boeing 747'](Aircraft Join Certified))
```

b. Find the *names* of pilots certified for 'Boeing 747' aircraft.

**Answer:**

Approach: collect together information about pilots and what planes they are certified on and project out just the name (since we need the pilot's name this time, we need to include `Employees`).

```
Answer = Proj[ename](Sel[aname='Boeing 747'](Aircraft Join Certified Join Employees))
```

c. Find the *ids* of all aircraft that can be used on non-stop flights from New York to Los Angeles.

**Answer:**

Approach: get all of the flights from New York to Los Angeles (to get the distance), join these with aircraft information and select only those combinations where the plane is capable of cruising that far. If you were fussy, you could also include flights from Los Angeles to New York with a disjunction in the inner `Sel` (in case the distance was different for some reason e.g. slightly different route).

```
Answer =
        Proj[aid](
                Aircraft
                Join[cruisingRange>distance]
                Sel[from='New York' AND to='Los Angeles'](Flights)
        )
```

d. Identify the flights that can be piloted by a pilot whose salary is more than $100,000.

**Answer:**

Approach: collect together information about which planes can make which flights and pilot certification, then select just those for which the pilot is certified and where the pilot earns more than $100000.

```
Temp   =
        Proj[flno,aid](
                Flights
                Join[distance<cruisingRange]
                Aircraft
        )
Answer =
        Proj[flno](
                Sel[salary>100000](
                        Temp Join Certified Join Employees
                )
        )
```

e. Find the names of pilots who can operate planes with a range greater than 3000 miles, but are not certified on 'Boeing 747' aircraft.

**Answer:**

Approach: get a list of employee ids who are certified for aircraft with a crusing range more than 3000 miles, then subtract the employee ids for those who *are* certified on a 747, leaving just those who are *not* certified on a 747; to get their names, join with the employees relation and project out the names.

```
LongRangeAircraftCert = Proj[eid](Sel[cruisingRange>3000](Aircraft Join Certfied))
Boeing747Cert = Proj[eid](Sel[aname='Boeing 747'](Aircraft Join Certfied))
Answer =
        Proj[ename](
                Employees Join (LongRangeAircraftCert Minus Boeing747Cert)
        )
```

f. Find the total amount paid to employees as salaries.

**Answer:**

```
Answer = Sum[salary](Employees)
```

g. Find the *ids* of employees who make the highest salary.

**Answer:**

Approach: find employees who do not have the highest salary (via the join), and then subtract them from the set of all employees, thus leaving just the highest paid employees.

```
E1 = Employees
E2 = Employees
Employees = Proj[eid](Employees)
LowerPaidEmployees = Proj[E2.eid](E1 Join[E1.salary>E2.salary] E2)
Answer = Employees Minus LowerPaidEmployees
```

h. Find the *ids* of employees who make the second highest salary.

**Answer:**

Approach: use the idea from the previous question; first find highest paid employees, then remove them from list of employees, then find highest paid employees in what's left; these will be the second highest-paid employees.

```
E1 = Employees
E2 = Employees
HighestPaid =
        Proj[eid](Employees)
        Minus
        Proj[E2.eid](E1 Join[E1.salary>E2.salary] E2)
NotHighestPaid =
        Proj[eid](Employees)  Minus  HighestPaid
E3 = NotHighestPaid
E4 = NotHighestPaid
SecondHighestPaid =
        NotHighestPaid
        Minus
        Proj[E4.eid](E3 Join[E3.salary>E4.salary] E4)
```

i. Find the *ids* of employees who are certified for the largest number of aircraft.

**Answer:**

Approach: Group by employee and count their certifications; then find max certifications; then select employees who have that many certifications

```
R1 = GroupBy[employee,Count[employee]](Certified)
R2 = Rename[1->employee,2->ncertified](R1)
R3 = Max[ncertified](R2)
Answer = Sel[ncertified=R3](R1)
```

j. Find the *ids* of employees who are certified for exactly three aircraft.

**Answer:**

Approach: find pilots who are certified for at least three aircraft; then find pilots who are certified for at least four aircraft; then subtract the first set from the latter and you'll pilots who are certified for exactly three aircraft; the "counting" is built into the join condition.

```
R1 = R2 = R3 = R4 = Certified
CertifiedForAtLeast3 =
        Proj[eid](
                Sel[R1.eid=R2.eid=R3.eid AND R1.aid!=R2.aid!=R3.aid](
                        R1 × R2 × R3
                )
        )
CertifiedForAtLeast4 =
        Proj[eid](
                Sel[R1.eid=R2.eid=R3.eid=R4.eid AND R1.aid!=R2.aid!=R3.aid!=R4.aid](
                        R1 × R2 × R3 × R4
                )
        )
Answer = CertifiedForAtLeast3 Minus CertifiedForAtLeast4
```

The above is a particularly inefficient (and obscure) solution. A better approach would be to solve it in a manner similar to the previous question.

```
R1 = GroupBy[employee,Count[employee]](Certified)
R2 = Rename[1->employee,2->ncertified](R1)
Answer = Proj[employee](Sel[ncertified=3](R2))
```

k. Find if there is any non-stop or single-stop way to travel by air from Sydney to New York.

**Answer:**

Approach: find any non-stop flights from Sydney to New York; find any single-stop flights from Sydney to New York; take the union of these two sets.

```
F1 = F2 = Flights
SydToNYNonStop = Proj[flno](Sel[from='Sydney' AND to='New York'](Flights))
SydToNYSingleStop = Proj[flno](Sel[F1.from='Sydney' AND F2.to='New York'](F1 Join[F1.to=F2.from] F2))
Answer = SydToNYNonStop Union SydToNYSingleStop
```

l. Is there a sequence of flights from Sydney to Timbuktu (somewhere in the middle of Africa)? Each flight in the sequence is required to depart from the destination of the previous flight; the first flight must leave Sydney, and the last flight must arrive in Timbuktu, but there is no restriction on the number of intermediate flights.

**Answer:**

This is a generalisation of the previous question to routes with an arbitrary number of stops. This cannot be expressed in standard relational algebra. It requires some kind of iterative or recursive querying mechanism before it will work. If we're willing to put an upper bound on the number of intermediate stops, we can generalise the previous query to do it.

10. (Date, exercises 6.13-6.40) Consider the following classic relational schema for a Parts-Suppliers-Jobs application:

```
Supplier(sid, sname, status, scity)
Part(pid, pname, colour, weight, pcity)
Job(jid, jname, jcity)
SPJ(supplier, part, job, qty)
```

The SPJ relation represents a shipment of parts from a supplier for a particular job and indicates how many parts are in the shipment; the ***supplier*** field contains the **sid** of the supplier, the ***part*** field contains the **pid** of the part, the ***job*** field contains the **jid** of the job.

Write a relational algebra expression (possibly expressed as a sequence of relational assignments) to answer each of the queries below. If a query says "get all XXX" it means get all instances in a given database, not all possible instances based on the domains.

a. Get full details of all jobs.

**Answer:**

```
Answer = Jobs
```

b. Get full details of all jobs in London.

**Answer:**

```
Answer = Sel[city=London](Job)
```

c. Get supplier ids for all suppliers who supply job J1

**Answer:**

```
Answer = Proj[supplier](Sel[job=J1](SPJ))
```

d. Get all shipments where the quantity is in the range 300 to 750 inclusive.

**Answer:**

```
Answer = Sel[qty>=300 && qty<=750](SPJ)
```

e. Get all (part,colour) and (part,city) combinations.

**Answer:**

```
Answer Proj[colour,city](Part)
```

f. Get all (supplier,part,job) triples such that the supplier, part and job are all co-located.

**Answer:**

```
Answer = Proj[sid,pid,jid](Supplier Join Part Join Job)
```

g. Get all (supplier,part,job) triples such that the supplier, part and job are not all co-located.

**Answer:**

```
TheLot = (Supplier × Part × Job)
DiffCities = Sel[scity!=pcity || pcity!=jcity || scity!=jcity](TheLot)
Answer = Proj[sid,pid,jid](DiffCities)
```

h. Get all (supplier,part,job) triples such that no two of the supplier, part and job are co-located.

**Answer:**

```
TheLot = (Supplier × Part × Job)
DiffCities = Sel[scity!=pcity && pcity!=jcity && scity!=jcity](TheLot)
Answer = Proj[sid,pid,jid](DiffCities)
```

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

**Answer:**

```
Answer = Proj[part](SPJ Join Select[scity=London](Supplier))
```

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

**Answer:**

```
LondonJobParts = Proj[part,job](SPJ Join Select[scity=London](Supplier))
LondonJobs = Select[jcity=London](Jobs)
Answer = Proj[part](LondonJobParts Join LondonJobs)
```

k. Get all pairs of city names such that a supplier in the first city supplies parts to a job in the second city.

**Answer:**

```
Answer = Proj[scity,jcity](Supplier Join SPJ Join Job)
```

l. Get the total number of projects supplied by supplier S1.

**Answer:**

```
Answer = Count(Proj[job](Sel[supplier=S1](SPJ)))
```

m. Get the total quantity of part P1 supplied by supplier S1.

**Answer:**

```
Answer = Sum(Proj[qty](Sel[supplier=S1 && part=P1](SPJ)))
```

n. For each part being supplied to a project, get the part number, the project number, and the corresponding total quantity.

**Answer:**

```
Answer = GroupBy[part]Sum[qty](SPJ)
```

Note: The `GroupBy` partitions the SPJ relation into groups based on part number; the `Sum` adds up the quanitities within each group and gives a single tuple with the part number and the total quantity.

o. Get part numbers supplied to some job in an average quantity of more than 320.

**Answer:**

```
PartUsage = GroupBy[part]Avg[qty](SPJ)
Answer = Proj[part]Sel[avg>320](PartUsage))
```

p. Get job names for jobs supplied by supplier S1.

**Answer:**

```
S1Shipments = Sel[supplier=S1](SPJ)
Answer = Proj[jname](Jobs Join S1Shipments)
```

q. Get colours of parts supplied by supplier S1.

**Answer:**

```
S1Shipments = Sel[supplier=S1](SPJ)
Answer = Proj[colour](Part Join S1Shipments)
```

r. Get part numbers for parts supplied to any job in London.

**Answer:**

```
LondonJobs = Sel[jcity=London](Job)
Answer = Proj[part](LondonJobs Join SPJ)
```

s. Get job numbers for jobs using at least one part available from supplier S1.

**Answer:**

```
S1Parts = Proj[part](Sel[supplier=S1](SPJ))
S1PartJobs = (SPJ Join S1Parts)
Answer = Proj[job]S1PartJobs
```

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

**Answer:**

```
statusOfS1 = Proj[status](Sel[sid=S1](Supplier))
Answer = Proj[sid](Sel[status<statusOfS1](Supplier))
```

Note: this assumes that you can treat a relation with a single row and column (statusOfS1) as a scalar value for use in comparisons.

u. Get job numbers for jobs supplied entirely by supplier S1.

**Answer:**

```
NotAllS1Jobs = Proj[job](Sel[supplier=S1](SPJ))
AllJobs = Proj[jid](Job)
Answer =  AllJobs - NotAllS1Jobs
```

v. Get supplier numbers for suppliers who supply the same part to all jobs.

**Answer:**

```
SPJTriples = Proj[supplier,part,job](SPJ)
AllJobNums = Proj[jid](Job)
Answer = Proj[supplier](SPJTriples Div AllJobNums)
```

w. Get all cities in which at least one supplier, part or job is located.

**Answer:**

```
SupplierCities = Proj[scity](Supplier)
PartCities = Proj[pcity](Part)
JobCities = Proj[jcity](Job)
Answer = SupplierCities U PartCities U JobCities
```

x. Get all pairs of supplier numbers ($S_x$,$S_y$) such that $S_x$ and $S_y$ supply exactly the same set of parts each.

**Answer:**

```
SupplierA = Rename[SupplierA(sidA,sname,status,scity)](Supplier)
SupplierB = Rename[SupplierB(sidB,sname,status,scity)](Supplier)
AllPairs = (SupplierA × SupplierB)
Answer = Sel[Proj[job](Sel[supplier=sidA](SPJ)) = Proj[job](Sel[supplier=sidA](SPJ)) && sidA<sidB](AllPairs)
```

Note: This solution assumes that we can compare sets as part of a select operation. It takes pairs of supplier numbers (sidA,sidB), finds the set of jobs supplied by each, and then checks that these sets are equal. The `sidA<sidB` term simply avoids having trivial pairs like (S1,S1) and also avoids having both (S1,S2) and (S2,S1).