COMP3311 22T1   # Week 10   Database Systems
Relational Algebra,
Transaction Processing

Relational Algebra

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	$N1$ (when $R2 \subset R1$)	$N1+N2$ (when $R1 \cap R2 = \varnothing$)
R1 Intersect *R2*	*R1* and *R2* are union-compatible	0 (when $R1 \cap R2 = \varnothing$)	$N2$ *(when $R2 \subset R1$)*
R1 - R2	*R1* and *R2* are union-compatible	$N1-N2$ (when $R2 \cap R1 = R2$)	$N1$ (when $R1 \cap R2 = \varnothing$)
R1 × R2	None	$N1*N2$	$N1*N2$
Sel[a=5](*R1*)	*R1* has an attribute *a*	0 (no matches)	$N1$ (all match)
Proj[a](*R1*)	*R1* has an attribute *a*, $N1 > 0$	1	$N1$
R1 Div *R2*	The set of *R2*'s attributes is 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:

Since *A* Intersect *B* is a subset of *A*, it has *A*'s key.

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](RedPartI
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
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[co
RedPartSupplierIds = Rename[RedPartSupplierIds(sid)]
PackerStSupplierIds = Proj[sid](Sel[address='221 Packer Stre
Answer = RedPartSupplierIds Union PackerStSupplierId
```

 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 Cat
Answer = RedPartSupplierIds   Intersect   GreenPartS
```

 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
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 GreenPartI
Answer = AllRedPartSuppliers Union AllGreenPartSuppli
```

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](C
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 J
R2 = R1
R3 = Rename[1->sid,2->pid,3->cost](Sel[R2.cost<R1.cost](R1 × R
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[si
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 Ce
```

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
```

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'](Flig
        )
```

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 Emp]
                )
        )
```

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](A
Boeing747Cert = Proj[eid](Sel[aname='Boeing 747'](Aircraft
Answer =
        Proj[ename](
                Employees Join (LongRangeAircraftCer1
```

```
                    )
```

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

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
SydToNYSingleStop = Proj[flno](Sel[F1.from='Sydney' AND F2.to
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.

Transaction Processing

10. [based on Ramakrishnan, ex.17.1]
    Give a brief definition for each of the following terms:

    a. transaction
    b. serializable schedule
    c. conflict-serializable schedule
    d. view-serializable schedule

Answer:

a. *transaction*
   An execution of a user program that performs some action that is treated as atomic according to the semantics of some database application. The DBMS sees the transaction as a sequence of actions that can include read and write operations on the database, as well as computations.

b. *serializable schedule*
   A schedule over a set of transactions that produces a result that is the same as some serial execution of the transactions.

c. *conflict-serializable schedule*
   A schedule is conflict-serializable if it is conflict-equivalent to some serial schedule. Two schedules are conflict-equivalent if they involve the same set of actions and they order every pair of conflicting actions in the same way.

d. *view-serializable schedule*
   A schedule is view-serializable if it is view-equivalent to some serial schedule. Two schedules are view-equivalent if they satisfy:

   - the initial value of any object is read by the same transaction in both schedules, and

- the final value of any object is written by the same transaction in both schedules, and
- any shared object is written-then-read by the same pair of transactions in both schedules.

11. Draw the precedence graph for the following schedule (where C means "commit"):

```
T1:      R(A) W(Z)                C
T2:                R(B) W(Y)          C
T3: W(A)                     W(B)     C
```

Answer:

It has an edge from T3 to T1 (because of A) and an edge from T2 to T3 because of B.

This gives: T2 --> T3 --> T1

12. [based on Ramakrishnan, ex.17.2]
Consider the following incomplete schedule *S*:

```
T1: R(X) R(Y) W(X)            W(X)
T2:                R(Y)            R(Y)
T3:                     W(Y)
```

a. Determine (by using a precedence graph) whether the schedule is conflict-serializable

b. Modify *S* to create a complete schedule that is conflict-serializable

Answer:

a. *Determine (by using a precedence graph) whether the schedule is serializable*

The precedence graph has an edge, from *T1* to *T3*, because of the conflict between T1:R(Y) and T3:W(Y). It also has an edge, from *T2* to *T3*, because of the conflict between the first T2:R(Y) and T3:W(Y). It also has an edge, from *T3* to *T2*, because of the conflict between T3:W(Y) and the second T2:R(Y). There is a cycle, so it is not conflict-serializable.

b. *Modify S to create a complete schedule that is conflict-serializable*

Trick question. It is not possible. Since the precedence graph is cyclic, we know that it's not conflict-serializable.

If we are allowed to add abort actions (which was not mentioned in the question), we could simply abort either T2 or T3 and the schedule would become conflict-serializable.

13. [based on Ramakrishnan, ex.17.3]

For each of the following schedules, state whether it is conflict-serializable and/or view-serializable. If you cannot decide whether a schedule belongs to either class, explain briefly. The actions are listed in the order they are scheduled, and prefixed with the transaction name.

 a. `T1:R(X) T2:R(X) T1:W(X) T2:W(X)`

 b. `T1:W(X) T2:R(Y) T1:R(Y) T2:R(X)`

 c. `T1:R(X) T2:R(Y) T3:W(X) T2:R(X) T1:R(Y)`

 d. `T1:R(X) T1:R(Y) T1:W(X) T2:R(Y) T3:W(Y) T1:W(X) T2:R(Y)`

 e. `T1:R(X) T2:W(X) T1:W(X) T3:W(X)`

Answer:

The methods used to determine these solutions:

- for conflict-serializablility, draw precedence graph and look for cycles
- for view-serializablility, apply the definition from lecture notes.

You can short-circuit the view serializability check. As soon as you know that the schedule is conflict-serializable, it must also be view serializable.

Solutions:

 a. not conflict-serializable, not view-serializable

 b. conflict-serializable, view-serializable

 c. conflict-serializable, view-serializable

 d. not conflict-serializable, not view-serializable

 e. not conflict-serializable, view-serializable (view equivalent to the serial schedule T1, T2, T3)

14. Is the following schedule serializable? Show your working.

```
T1:               R(X)W(X)W(Z)         R(Y)W(Y)
T2: R(Y)W(Y)R(Y)              W(Y)R(X)         W(X)R(V)W(V)
```

Answer:

When we talk about serializability and don't specifically say what kind, we usually mean conflict-serializable. As above, the "working" for this question involves constructing a precedence graph, based on conflicting operations, and looking for cycles.

In this case there's a conflict between $T1:R(X)$ and $T2:W(X)$, giving a graph edge from $T1$ to $T2$. There's also a conflict between $T2:R(Y)$ and

`T1:W(Y)`, giving a graph edge from `T2` to `T1`. This means the graph has a cycle, so the schedule is not serializable.