Evaluating Rules

Add rule evaluation to the query interpreter from the last project. The major steps of the interpreter are:

1. Process the schemes (same as the last project)
2. Process the facts (same as the last project)
3. Evaluate the rules (new code)
4. Evaluate the queries (same as the last project)

In the order they are given in the input file, evaluate each rule using relational algebra operations as follows:

1. **Evaluate the predicates on the right-hand side of the rule:**

   For each predicate on the right-hand side of a rule, evaluate the predicate in the same way you evaluated the queries in the last project (using select, project, and rename operations). Each predicate should produce a single relation as an intermediate result. If there are n predicates on the right-hand side of a rule, there should be n intermediate results.

2. **Join the relations that result:**

   If there are two or more predicates on the right-hand side of a rule, join the intermediate results to form the single result for Step 2. Thus, if p1, p2, and p3 are the intermediate results from Step 1, join them (p1 |x| p2 |x| p3) into a single relation.

   If there is a single predicate on the right hand side of the rule, use the single intermediate result from Step 1 as the result for Step 2.

3. **Project the columns that appear in the head predicate:**

   The predicates in the body of a rule may have variables that are not used in the head of the rule. Use a project operation on the result from Step 2 to remove the columns that don't appear in the head of the rule and to reorder the columns to match the order in the head.

4. **Reorder the columns of the relation to produce the order in the head predicate:**

   The variables in the head predicate may appear in a different order than the columns of the relation produced in the previous step

5. **Rename the relation to make it union-compatible:**

   Rename the relation that results from Step 3 to make it union compatible with the relation that matches the head of the rule. Rename each attribute in the result from Step 3 to the attribute name found in the corresponding position in the relation that matches the head of the rule.

6. **Union with the relation in the database:**

   Union the result from Step 4 with the relation in the database whose name matches the name of the head of the rule.

```
Schemes:
snap(S,N,A,P)
  csg(C,S,G)
  cn(C,N)
  ncg(N,C,G)
Facts:
  snap('12345','C. Brown','12 Apple St.','555-
1234').
  snap('22222','P. Patty','56 Grape Blvd','555-
9999').
  snap('33333','Snoopy','12 Apple St.','555-
1234').
  csg('CS101','12345','A').
  csg('CS101','22222','B').
  csg('CS101','33333','C').
  csg('EE200','12345','B+').
  csg('EE200','22222','B').
Rules:
   cn(c,n) :- snap(S,n,A,P),csg(c,S,G)
   ncg(n,c,g) :- snap(S,n,A,P),csg(c,S,g)
```

Here are the relations after populating the relational database:

snap			
S	N	A	P
'12345'	'C. Brown'	'12 Apple St.'	'555-1234'
'22222'	'P. Patty'	'56 Grape Blvd'	'555-9999'
'33333'	'Snoopy'	'12 Apple St.'	'555-1234'

csg		
C	S	G
'CS101'	'123435'	'A'
'CS101'	'22222'	'B'
'CS101'	'33333'	'C'
'EE200'	'12345'	'B+'
'EE200'	'22222'	'B'

cn	
C	N

ncg		
N	C	G

1. **Evaluate the predicates on the right-hand side of the rule:**

For each predicate on the right-hand side of a rule, evaluate the predicate in the same way you evaluated the queries in the last project (using select, project, and rename operations). Each predicate should produce a single relation as an intermediate result. If there are n predicates on the right-hand side of a rule, there should be n intermediate results.

Here are the *intermediate relations* for the right-hand side of the first rule in the datalog program

Rules:

```
cn(c,n) :- snap(S,n,A,P),csg(c,S,G)
```

snap			
S	n	A	P
'12345'	'C. Brown'	'12 Apple St.'	'555-1234'
'22222'	'P. Patty'	'56 Grape Blvd'	'555-9999'
'33333'	'Snoopy'	'12 Apple St.'	'555-1234'

csg		
c	S	G
'CS101'	'123435'	'A'
'CS101'	'22222'	'B'
'CS101'	'33333'	'C'
'EE200'	'12345'	'B+'
'EE200'	'22222'	'B'

2. **Join the relations that result:**

    If there are two or more predicates on the right-hand side of a rule, join the intermediate results to form the single result for Step 2. Thus, if p1, p2, and p3 are the intermediate results from Step 1, join them (p1 |x| p2 |x| p3) into a single relation.

    If there is a single predicate on the right hand side of the rule, use the single intermediate result from Step 1 as the result for Step 2.

Here is the *intermediate relation* for the right-hand side of the first rule in the datalog program after the natural join

snap(S,n,A,P) |x| csg(c,S,G)					
S	n	A	P	c	G
'12345'	'C. Brown'	'12 Apple St.'	'555-1234'	'CS101'	'A'
'12345'	'C. Brown'	'12 Apple St.'	'555-1234'	'EE200'	'B+'
'22222'	'P. Patty'	'56 Grape Blvd'	'555-9999'	'CS101'	'B'
'22222'	'P. Patty'	'56 Grape Blvd'	'555-9999'	'EE200'	'B'
'33333'	'Snoopy'	'12 Apple St.'	'555-1234'	'CS101'	'C'

3. **Project the columns that appear in the head predicate:**

    The predicates in the body of a rule may have variables that are not used in the head of the rule. The variables in the head may also appear in a different order than those in the body. Use a project operation on the result from Step 2 to remove the columns that don't appear in the head of the rule and to reorder the columns to match the order in the head.

Here is the *intermediate relation* for the right-hand side of the first rule after the projection:

```
cn(c,n) :- snap(S,n,A,P),csg(c,S,G).
```

π_{cn}[snap(S,n,A,P) |x| csg(c,S,G)]	
n	c
'C. Brown'	'CS101'
'C. Brown'	'EE200'
'P. Patty'	'CS101'
'P. Patty'	'EE200'
'Snoopy'	'CS101'

4. **Reorder the columns to match the order of the head predicate:**

The predicates in the head of a rule may appear in a different order than the order in the relation produced by the projection of the predicates n in the body. Reorder the relation produced by the projection to match the order of the head predicate.

Here is the *intermediate relation* for the right-hand side of the first rule after the projection:

```
cn(c,n) :- snap(S,n,A,P),csg(c,S,G).
```

$\tau_{nc} \left[\pi_{cn}[\text{snap}(S,n,A,P) \ |\times| \ \text{csg}(c,S,G)] \right]$	
c	n
'CS101'	'C. Brown'
'EE200'	'C. Brown'
'CS101'	'P. Patty'
'EE200'	'P. Patty'
'CS101'	'Snoopy'

5. **Rename the relation to make it union-compatible:**

Rename the relation that results from Step 3 to make it union compatible with the relation that matches the head of the rule. Rename each attribute in the result from Step 3 to the attribute name found in the corresponding position in the relation that matches the head of the rule.

The attribute names for the cn relation already in the database are C and N. Here is the *intermediate relation* for the right-hand side of the first rule after renaming to match the relation in the database that matches the head of the rule.

$\rho_{c \leftarrow C, n \leftarrow N} \left[\tau_{nc} \left[\pi_{cn}[\text{snap}(S,n,A,P) \ |\times| \ \text{csg}(c,S,G)] \right] \right]$	
C	N
'CS101'	'C. Brown'
'EE200'	'C. Brown'
'CS101'	'P. Patty'
'EE200'	'P. Patty'
'CS101'	'Snoopy'

6. **Union with the relation in the database:**

Union the result from Step 4 with the relation in the database whose name matches the name of the head of the rule.

$$cn \leftarrow cn \cup \rho_{c \leftarrow C, n \leftarrow N} \left[\tau_{nc} \left[\pi_{cn}[snap(S,n,A,P) |\times| csg(c,S,G)] \right] \right]$$

Before union:

cn	
C	N

After union:

cn	
C	N
'CS101'	'C. Brown'
'EE200'	'C. Brown'
'CS101'	'P. Patty'
'EE200'	'P. Patty'
'CS101'	'Snoopy'

The database has now been modified to include what was there before for snap, csg, and ncg, but the cn relation has elements. All of the relations created in steps 1-4 are intermediate and temporary relations; the union operator adds new tuples to the existing cn relation, so step 5 is the only step that modifies the database.

Evaluate the predicates on the right-hand side of the rule:

For each predicate on the right-hand side of a rule, evaluate the predicate in the same way you evaluated the queries in the last project (using select, project, and rename operations). Each predicate should produce a single relation as an intermediate result. If there are n predicates on the right-hand side of a rule, there should be n intermediate results.

Here are the *intermediate relations* for the right-hand side of the second rule in the datalog program

Rules:
```
ncg(n,c,g) :- snap(S,n,A,P),csg(c,S,g)
```

snap			
S	n	A	P
'12345'	'C. Brown'	'12 Apple St.'	'555-1234'
'22222'	'P. Patty'	'56 Grape Blvd'	'555-9999'
'33333'	'Snoopy'	'12 Apple St.'	'555-1234'

csg		
c	S	g
'CS101'	'123435'	'A'
'CS101'	'22222'	'B'
'CS101'	'33333'	'C'
'EE200'	'12345'	'B+'
'EE200'	'22222'	'B'

2. **Join the relations that result:**

If there are two or more predicates on the right-hand side of a rule, join the intermediate results to form the single result for Step 2. Thus, if p1, p2, and p3 are the intermediate results from Step 1, join them (p1 |x| p2 |x| p3) into a single relation.

If there is a single predicate on the right hand side of the rule, use the single intermediate result from Step 1 as the result for Step 2.

Here is the *intermediate relation* for the right-hand side of the first rule in the datalog program after the natural join

snap(S,n,A,P) |x| csg(c,S,g)					
S	n	A	P	c	g
'12345'	'C. Brown'	'12 Apple St.'	'555-1234'	'CS101'	'A'
'12345'	'C. Brown'	'12 Apple St.'	'555-1234'	'EE200'	'B+'
'22222'	'P. Patty'	'56 Grape Blvd'	'555-9999'	'CS101'	'B'
'22222'	'P. Patty'	'56 Grape Blvd'	'555-9999'	'EE200'	'B'
'33333'	'Snoopy'	'12 Apple St.'	'555-1234'	'CS101'	'C'

3. **Project the columns that appear in the head predicate:**

The predicates in the body of a rule may have variables that are not used in the head of the rule. The variables in the head may also appear in a different order than those in the body. Use a project operation on the result from Step 2 to remove the columns that don't appear in the head of the rule and to reorder the columns to match the order in the head.

Here is the *intermediate relation* for the right-hand side of the first rule after the projection:

```
Rules:
    ncg(n,c,g) :- snap(S,n,A,P),csg(c,S,g)
```

π_{ncg}[snap(S,n,A,P) |x| csg(c,S,g)]		
n	c	g
'C. Brown'	'CS101'	'A'
'C. Brown'	'EE200'	'B+'
'P. Patty'	'CS101'	'B'
'P. Patty'	'EE200'	'B'
'Snoopy'	'CS101'	'C'

Reorder the columns to match the order of the head predicate:

The predicates in the head of a rule may appear in a different order than the order in the relation produced by the projection of the predicates n in the body. Reorder the relation produced by the projection to match the order of the head predicate.

Here is the *intermediate relation* for the right-hand side of the first rule after the projection:

```
cn(c,n) :- snap(S,n,A,P),csg(c,S,G).
```

$\tau_{ncg}\left[\pi_{ncg}[\text{snap(S,n,A,P)} \; |\times| \; \text{csg(c,S,g)}]\right]$		
n	c	g
'C. Brown'	'CS101'	'A'
'C. Brown'	'EE200'	'B+'
'P. Patty'	'CS101'	'B'
'P. Patty'	'EE200'	'B'
'Snoopy'	'CS101'	'C'

Nothing changed because the order was already correct

5. **Rename the relation to make it union-compatible:**

Rename the relation that results from Step 3 to make it union compatible with the relation that matches the head of the rule. Rename each attribute in the result from Step 3 to the attribute name found in the corresponding position in the relation that matches the head of the rule.

The attribute names for the cn relation already in the database are N, C, and G. Here is the *intermediate relation* for the right-hand side of the first rule after renaming to match the relation in the database that matches the head of the rule.

$\rho_{n\leftarrow N, \; c\leftarrow C, \; g\leftarrow G,}\left[\tau_{ncg}\left[\pi_{ncg}[\text{snap(S,n,A,P)} \; |\times| \; \text{csg(c,S,G)}]\right]\right]$		
N	C	G
'C. Brown'	'CS101'	'A'
'C. Brown'	'EE200'	'B+'
'P. Patty'	'CS101'	'B'
'P. Patty'	'EE200'	'B'
'Snoopy'	'CS101'	'C'

6. **Union with the relation in the database:**

   Union the result from Step 4 with the relation in the database whose name matches the name of the head of the rule.

$$ncg \leftarrow ncg \cup \rho_{n\leftarrow N,\, c\leftarrow C,\, g\leftarrow G,}\ \left[\tau_{ncg}\left[\pi_{ncg}[snap(S,n,A,P) \bowtie csg(c,S,G)]\right]\right]$$

Before union:

ncg		
N	C	G

After union:

ncg		
N	C	G
'C. Brown'	'CS101'	'A'
'C. Brown'	'EE200'	'B+'
'P. Patty'	'CS101'	'B'
'P. Patty'	'EE200'	'B'
'Snoopy'	'CS101'	'C'

The database has now been modified to include what was there before for snap and csg, but the cn and ncg relations now have elements. All of the relations created in steps 1-5 are intermediate and temporary relations; the union operator adds new tuples to the existing cn and ncg relations, so step 6 is the only step that modifies the database.