Select Project Rename – Examples from Project 3

Michael A. Goodrich

October 12, 2020

1 Introduction

Project 3 uses relational algebra to "answer" queries in a Datalog program. Only facts will be used to evaluate the queries, with rules added in the next project. The purpose of this tutorial is to use examples from the project to help you understand the concepts from the book and from class.

Consider the Datalog program shown below, which is a simplified version of *in62.txt* from the set of test files.

```
Schemes:
   snap(S,N,A,P)
   csg(C,S,G)
   cp(C,Q)
   cdh(C,D,H)
   cr(C,R)

Facts:
   snap('12345','C. Brown','12 Apple St.','555-1234').
   snap('67890','L. Van Pelt','34 Pear Ave.','555-5678').
   snap('22222','P. Patty','56 Grape Blvd.','555-9999').
   snap('33333','Snoopy','12 Apple St.','555-1234').
   csg('CS101','12345','A').
   csg('CS101','67890','B').
   csg('EE200','12345','C').
   csg('EE200','22222','B+').
   csg('EE200','33333','B').
   csg('CS101','33333','A-').
   csg('PH100','67890','C+').
   cp('CS101','CS100').
   cp('EE200','EE005').
   cp('EE200','CS100').
   cp('CS120','CS101').
   cp('CS121','CS120').
   cp('CS205','CS101').
   cp('CS206','CS121').
   cp('CS206','CS205').
```

```
   cdh('CS101','Tu','10AM').
   cdh('EE200','M','10AM').
   cdh('EE200','W','1PM').
   cdh('EE200','F','10AM').
   cdh('PH100','Th','11AM').
   cr('CS101','Turing Aud.').
   cr('EE200','25 Ohm Hall').
   cr('PH100','Newton Lab.').

Rules:

Queries:
   snap(Id,'Snoopy',A,P)?
   csg(Course,'33333',Grade)?
   cp(Course,'CS100')?
   cdh('EE200',Day,Hour)?
   cr('CS101',Room)?
```

Let's walk through the steps required for the project, using the steps to illustrate definitions from the textbook and from class.

1.1 Relations and Representations

The textbook defines a *relation* as a subset of the Cartesian product over a collection of sets. A relation has a *name*, a *scheme*, and a *set of tuples*. The *scheme* is the list of attributes associated with the relation.

In Datalog, the `Schemes:` portion of the code declares the name of the relation and the list of attributes. For example, in `snap(S,N,A,P)` the name of the relation is *snap* and the scheme is the list of attributes *(S,N,A,P)*. I like to think of each of the attributes as the name of the set used in the cross product from which the relation is derived, so $snap \subseteq S \times N \times A \times P$.

We can represent the relation using a data structure that encodes the useful figure from the Wikipedia article on the *Relational Data Model*, found at this link `https://en.wikipedia.org/wiki/Relational_model` and shown in Table 1. For this tutorial, we'll give the name of the relation in the caption. The relation in

$\mathbf{A_1}$	$\cdots$	$\mathbf{A_n}$

Table 1: `EmptyRelation`

Table 1 is empty, meaning that there are no tuples in the relation. Thus, I called the relation `EmptyRelation`. The *scheme* is the list of attributes, which are shown in the first row of the table (in bold font and separated from the rest of the table by double lines). For `EmptyRelation`, the scheme is $(A_1, A_2, \ldots, A_n)$. All of this is given in the `Schemes:` portion of the Datalog program. The tuples, which correspond to the rows of the table (at least the rows below the scheme) are given in the `Facts:` poritino of the Datalog program.

Given the information about representing a relation using a table, we can create tables for each of the relations in the Datalog program above. Table 2 has four rows of tuples, one for each of the `snap` facts.

S	N	A	P
'12345'	'C. Brown'	'12 Apple St.'	'555-1234'
'67890'	'L. Van Pelt'	'34 Pear Ave.'	'555-5678'
'22222'	'P. Patty'	'56 Grape Blvd.'	'555-9999'
'33333'	'Snoopy'	'12 Apple St.'	'555-1234'

Table 2: `snap`

We can construct a similar table for `csg`.

C	S	G
'CS101'	'12345'	'A'
'CS101'	'67890'	'B'
'EE200'	'12345'	'C'
'EE200'	'22222'	'B+'
'EE200'	'33333'	'B'
'CS101'	'33333'	'A-'
'PH100'	'67890'	'C+'

Table 3: `csg`

In the lab, you will implement a relational database using classes. The key to getting the class organization designed well is the following statement from the lab specs: "A relational database is a collection of relations. A relation has a name, a scheme, and a set of tuples. A scheme is a list of attribute names. A tuple is a list of attribute values." The class structure follows from the spec: "Write classes that implement a simple relational database. Your design must include at least the following classes: Database, Relation, Scheme, and Tuple."

1.2 Relational Algebra

A relational algebra consists of a collection of relations and a set of operations that can be performed on those relations. The book talks about several operations, and we can divide the operations into two categories: *set operations* and *relational operations*. Relations are sets of tuples, so all of the set operations that we've already discussed can be used, including union, intersection, Cartesian product, and set difference. I'll post some slides with examples of each of these.

The relation-specific operations are more interesting because they form the basis of answering relational database queries. In this tutorial, we'll consider four: select, project, rename, and join. The first three are necessary for project 3.

To help motivate the relation-specific operations, consider the algorithm identified in the project specs:

1. Get the Relation from the Database with the same name as the predicate name in the query.
2. Use one or more **select** operations to select the tuples from the Relation that match the query. Iterate over the parameters of the query: If the parameter is a constant, select the tuples from the Relation that have the same value as the constant in the same position as the constant. If the parameter is a

variable and the same variable name appears later in the query, select the tuples from the Relation that have the same value in both positions where the variable name appears.

3. After selecting the matching tuples, use the **project** operation to keep only the columns from the Relation that correspond to the positions of the variables in the query. Make sure that each variable name appears only once in the resulting relation. If the same name appears more than once, keep the first column where the name appears and remove any later columns where the same name appears. (This makes a difference when there are other columns in between the ones with the same name.)

4. After projecting, use the **rename** operation to rename the scheme of the Relation to the names of the variables found in the query.

Essentially, the algorithm iteratively selects, projects, and renames. Let's practice, with the intent of matching the definition of selection from the textbook with the practice in the algorithm above.

Consider the query `snap(Id,'Snoopy',A,P)?`

1.2.1 Select

According to the textbook,

> Let R be an n-ary relation and C and condition that element sin R must satisfy. Then the *selection operator* s_C maps the n-ary relation R to the n-ary relation of all n-tuples from R that satisfy the condition C.

Simply put, the selection operator creates a new relation, let's call it $\texttt{Select}_C(R)$ from an existing relation R by *eliminating tuples* from R that don't satisfy the condition. Selection drops tuples from an existing relation. It takes as input an existing relation R and a condition C, and it outputs the new relation $\texttt{Select}_C(R)$.

Let's illustrate by considering the first query `snap(Id,'Snoopy',A,P)?`. Step 1 in the algorithm above says to find the relation from the database with the same name as the predicate in the query. Thus, we'll deal with the `snap` relation from Table 2.

Step 2 says to use the select operator. We do this by iterating over the The inputs are (a) the `snap` relation from Table 2 and (b) the condition that the second attribute is the constant `'Snoopy'`. The resulting tuple, which is the output of the select operator, is give in Table 4. Only one tuple satisfied the condition, so the

S	N	A	P
'33333'	'Snoopy'	'12 Apple St.'	'555-1234'

Table 4: $\texttt{Select}_{\texttt{N=Snoopy}}\texttt{(snap)}$

new relation only has one tuple in the table.

1.2.2 Projection

According to the textbook,

> The *projection* $P_{i_1,i_2,\ldots,i_m}$, where $i_a < i_2 \ldots i_m$, maps the n-tuple $(a_1, a_2, \ldots, a_n)$ to the m-tuple $(a_{i_1}, a_{i_2}, \ldots, a_{i_m})$, where $m \leq n$.

Simply put, the projection operator creates a new relation, let's call it $\texttt{Project}_{\texttt{KeepAttributes}}(R)$ from an existing relation R by *eliminating attributes* that don't match the list of attributes. Projection drops attributes from an existing relation. It takes as input an existing relation R and a list of attributes to keep `KeepAttributes`, and it outputs a new relation $\texttt{Project}_{\texttt{KeepAttributes}}(R)$.

Step 3 in the above algorithm says to use the project operation says to use projection to eliminate those variable names (which we would call "attribute names" to match how we defined a relation) that correspond to the variables in the query. The query has three variables, `Id`, `A`, and `P`, and the query has one constant `'Snoopy'`. We will eliminate the constant attribute. Table 5 shows the new relation, the one created from

S	A	P
'33333'	'12 Apple St.'	'555-1234'

Table 5: $\texttt{Project}_{S,A,P}\big(\texttt{Select}_{\texttt{N=Snoopy}}\texttt{(snap)}\big)$

the Table 4 by eliminating the N attribute and keeping the S, A, and P attributes. Note that the input relation, the stuff between the big parentheses, is the selection relation from Step 2.

1.2.3 Rename

The textbook doesn't include a definition of the rename operation, but we can make one up:

> The *rename operator* maps the relation R with a scheme consisting of the following attributes $(A_1, A_2, \ldots, A_{i-1}, A_i, A_{i+1}, \ldots, A_n)$ to a new relation $\rho_{A_i \leftarrow B_i}(R)$ with a scheme consisting of the following attributes, $(A_1, A_2, \ldots, A_{i-1}, B_i, A_{i+1}, \ldots, A_n)$.

Simply put, the rename operator creates a new relation, let's call it $\texttt{Rename}_{A \leftarrow B}(R)$ from an existing relation R by *changing one attribute* from A to B. Renaming changes what we call one attribute. It takes as input an attribute that we want to rename, the new name for that attribute, and a relation, and it returns the new relation $\texttt{Rename}_{A \leftarrow B}(R)$.

Step 4 from the algorithm above asks us to use the rename operator to create a relation whose attributes match the variable names in the query. Table 5 has the first attribute named S but the query has the first variable named `Id`. We need to rename S. This creates the new relation shown in Table 6 The name of this

Id	A	P
'33333'	'12 Apple St.'	'555-1234'

Table 6: $\texttt{Rename}_{S \leftarrow \texttt{Id}}\Big(\texttt{Project}_{S,A,P}\big(\texttt{Select}_{\texttt{N=Snoopy}}\texttt{(snap)}\big)\Big)$

relation is really awkward now so we'll need a better way to create names in Project 3, but the idea is simple. First, Table 4 creates a new relation by selecting tuples for which $N =$ `Snoopy`. Second, Table 5 eliminates the attribute that matched with the constant. Third, Table 6 changed the attribute name from S to `Id`.

1.2.4 Output for the Query

The output for the first query in test case *in62.txt* is given by:

```
snap(Id,‘Snoopy’,A,P)? Yes(1)
  Id=‘33333’, A=‘12 Apple St.’, P=‘555-1234’
```

The output repeats the query, `snap(Id,‘Snoopy’,A,P)?` followed by whether there are any tuples in the relation that satisfies the query, `Yes`, and the number of tuples that satisfy the query, `(1)`. For each tuple that satisfies the query, the next lines output the name of the attribute in the relation and the corresponding value in the tuple. Viewed another way, the line

```
  Id=‘33333’, A=‘12 Apple St.’, P=‘555-1234’
```

outputs the attribute and value columns, one tuple at a time.

2 A Second Example

Consider the second query, `csg(Course,’33333’,Grade)?`. See if I did the select, project, and rename operations correctly to respond to the query.

2.1 Select

C	S	G
‘EE200’	‘33333’	‘B’
‘CS101’	‘33333’	‘A-’

Table 7: $\texttt{Select}_{\texttt{S}=\texttt{33333}}(\texttt{csg})$

2.2 Project

C	G
‘EE200’	‘B’
‘CS101’	‘A-’

Table 8: $\texttt{Project}_{C,G}\big(\texttt{Select}_{\texttt{S}=\texttt{33333}}(\texttt{csg})\big)$

2.3 Rename

Course	G
‘EE200’	‘B’
‘CS101’	‘A-’

Table 9: $\texttt{Rename}_{C\leftarrow\texttt{Course}}\Big(\texttt{Project}_{C,G}\big(\texttt{Select}_{\texttt{S}=\texttt{33333}}(\texttt{csg})\big)\Big)$

Course	Grade
‘EE200’	‘B’
‘CS101’	‘A-’

Table 10: $\texttt{Rename}_{G \leftarrow \texttt{Grade}}\Big[\ \texttt{Rename}_{C \leftarrow \texttt{Course}}\Big(\texttt{Project}_{C,G}\big(\texttt{Select}_{\texttt{S}=33333}(\texttt{csg})\big)\Big)\Big]$

2.4 Output

```
csg(Course,‘33333’,Grade)? Yes(2)
  Course=‘EE200’, Grade=‘B’
  Course=‘CS101’, Grade=‘A-’
```

Naturally, I’d need to sort the output so that it matches the order required in the project specs.