Relational Data Model

CS236 - Discrete Structures
Instructor: Brett Decker
Fall 2021

Relational Data Model

A *relation* can be used to represent data. The relation defines a table, and each tuple is a row in the table. This model is called the relational data model. Here is an example with the relation R, a binary relation on sets A and B:

$A = \{a, b, c, d\}$
$B = \{a, c, e, g\}$
$R = \{(a, a), (a, c), (b, c), (b, g), (c, e)\}$

Let's create the relational table for R:

A	**B**
a	a
a	c
b	c
b	g
c	e

We say that A, B is the schema of the relation (this should seem irksomely familiar to Datalog schemes). There are five rows, or instances, of the relation $(a, a), (a, c), (b, c), (b, g), (c, e)$ (think about Datalog facts). Let's look at a Datalog program and see how it can be represented in a relational data model:

```
Schemes:
   myScheme(A, B)
Facts:
   myScheme(‘a’, ‘a’).
   myScheme(‘a’, ‘c’).
   myScheme(‘b’, ‘c’).
   myScheme(‘b’, ‘g’).
   myScheme(‘c’, ‘e’).
Rules:
   myScheme(‘c’, X) :- myScheme(‘a’, X), myScheme(‘b’, X).
Queries:
   myScheme(A, ‘c’)?
   myScheme(X, X)?
```

This Datalog program defines a scheme, or schema of a relation, `mySchememe` that has two columns: `A` and `B`. This correlates to two sets A and B. The Datalog program defines all the facts, or instances of the relation. Thus the information contained in the Datalog schemes and facts can be represented in a relational table (the exact same as we saw above):

`mySchememe` =

A	**B**
a	a
a	c
b	c
b	g
c	e

The only difference is that we now have a label (or name) for the table. The facts also define the domains for the set specified by the schema columns. $A = \{a, b, c\}$. $B = \{a, c, e, g\}$. Note that these domains are smaller than what we specified originally. That's because Datalog cannot reason about values in the domain not specified in the Datalog facts. This is an important difference. Before we can dive into Datalog rules and queries, we need to introduce operators and operations that can be used with relational tables.

Unary Operators

There are three unary operators that we will be studying: select, project, and rename. Each of these operators operates on a relational table. Each operation returns a new relational table, thus preserving the original. While it is not necessary to put parentheses around the relation being operated on by a unary operator, this document will for clarity.

Select

The select operator is denoted with the Greek lowercase-letter sigma, σ. The select operator can be thought of as "choosing rows." For example, $\sigma_{A=b}(\texttt{mySchememe})$ results in a new relational table that only contains the rows from `mySchememe` where column `A` has the value `b`. Thus:

$\sigma_{A=b}(\texttt{mySchememe}) =$

A	**B**
b	c
b	g

The result is that certain rows were "selected" from the original relational table. In order for this operator to work, there are constraints on the subscript of σ with the labels used. All labels specified must be in the schema (the column labels). Otherwise this operation is undefined, e.g. $\sigma_{W=g}(\texttt{mySchememe})$ is undefined. There is no constraint on the values – if the value is not in the domain then the result is just an empty table.

Project

The project operator is denoted with the Greek lowercase-letter pi, π. The project operator can be thought of as "choosing columns." For example, $\pi_A(\texttt{myScheme})$ results in a new relational table that only contains the specified column, `A`, from `myScheme`. Thus:

$\pi_A(\texttt{myScheme}) =$

A
a
b
c

The result is that one column was "projected" from the original relational table. Note that we never store duplicate rows. With projection, the number of rows in the final table may not be the same as that in the original table. This is possible because the "uniqueness" of rows is reduced with less columns, resulting in possibly more duplicates, and therefore fewer final rows in our table (as seen in our example above). In order for this operator to work, there are constraints on the subscript of π with the labels used. All labels specified must be in the schema (the column labels). Otherwise this operation is undefined, e.g. $\pi_E(\texttt{myScheme})$ is undefined. More than one label can be specified on which to perform the project. Consider $\pi_{AB}(\texttt{myScheme})$:

$\pi_{AB}(\texttt{myScheme}) =$

A	**B**
a	a
a	c
b	c
b	g
c	e

The resulting table is the same as the original because we projected all columns. Projection also allows us to reorder columns in a table as shown below:

$\pi_{BA}(\texttt{myScheme}) =$

B	**A**
a	a
c	a
c	b
g	b
e	c

Rename

The rename operator is denoted with the Greek lowercase-letter rho, ρ. The labels of our columns in relational tables are called attributes. The rename operator can be thought of as "changing attributes." For example, $\rho_{A \leftarrow Z}(\texttt{myScheme})$ results in a new relational table where the column `A` from `myScheme` is renamed to `Z`. Thus:

$\rho_{A \leftarrow Z}($`myScheme`$) =$

Z	**B**
a	a
a	c
b	c
b	g
c	e

In order for this operator to work, there are constraints on the subscript of ρ with the labels used, as in $A \leftarrow Z$. The label on the right-hand side of the arrow must not be already used in the schema. The label on the left-hand side of the arrow must be in the schema. Otherwise the operation is undefined. This operator seems to not be very helpful, but it will be for answering Datalog queries.

Evaluating Datalog Queries

Now that we have the operators select, project, and rename we can evaluate all queries. Consider again the queries of the Datalog program shown previously:

```
Queries:
   myScheme(A, ‘c’)?
   myScheme(X, X)?
```

Let's start with the first query: `myScheme(A, ‘c’)`. The query can be described in English: what values are in column `A` where the second column has the value `c`? How do we solve for this? There is a systematic approach. We select on the table when there is a specific value we care about, as in the 'c'. We project on the table when there are specific columns we care about, as in the `A`. We also need to do it in that order. We perform all selections before any projections. Using our unary operators, we can now evaluate the query by evaluating this expression: $\pi_A(\sigma_{B=c}($`myScheme`$))$. Let's evaluate one piece at a time.

$\sigma_{B=c}($`myScheme`$) =$

A	**B**
a	c
b	c

$\pi_A(\sigma_{B=c}($`myScheme`$)) =$

A
a
b

Thus the answer to our query is that the values left in column `A` are `a, b`. Let's consider the second query, because it is slightly more complicated: `myScheme(X, X)`. This query is stating the following: what are the values of `X` such that the table contains the same value in the first two columns, `A` and `B`, respectively? Note that this query does not use the same variable name in the columns as the relational table. That means we will need to use the rename operator. The rename will occur as the last step. Also, note the difference in that we need to select rows where the two values in each column are the same. This query can be represented by the following expression: $\rho_{A \leftarrow X}(\pi_A(\sigma_{A=B}($`myScheme`$)))$. Note that the projection could be on either column, since they will both hold the same value. Let's evaluate this expression:

$\sigma_{A=B}($`myScheme`$) =$

A	**B**
a	a

$\pi_A(\sigma_{A=B}($`myScheme`$)) =$

A
a

$\rho_{A \leftarrow X}(\pi_A(\sigma_{A=B}(\texttt{myScheme}))) =$

X
a

Thus the result is that the query is satisfiable with the answer as `X = a`. Note that not all queries will be satisfiable. The result may sometimes be there is no possible answer.

Binary Operators

There are five binary operators that we will be studying: union, intersection, difference, cross product, and natural join. Each of these operators operates on two relational tables. Each operation returns a new relational table, thus leaving the original two tables unmodified.

Union

The union operator is denoted, as usual, by $\cup$. The union of two relational tables is the same as the union of two relations. Here is an example:

$T_1 =$

A	**B**
a	1
b	2
b	3
c	6

$T_2 =$

A	**B**
a	2
a	4
b	3
c	5
c	6

$T_1 \cup T_2 =$

A	**B**
a	1
a	2
a	4
b	2
b	3
c	5
c	6

Note that, just as with union on relations, we do not keep duplicate rows (tuples). Unlike union on relations, the schema of each relational table must be the same. The following union fails, as the tables are not compatible.

$T_1 =$

A	**B**
a	1
b	2
b	3
c	6

$T_2 =$

C	**D**
a	2
a	4
b	3
c	5
c	6

$T_1 \cup T_2 =$ *undefined*, no result

Intersection

The intersection operator is denoted, as usual, by $\cap$. The intersection of two relational tables is the same as the intersection of two relations. Here is an example:

$T_1 =$

A	**B**
a	1
b	2
b	3
c	6

$T_2 =$

A	**B**
a	2
a	4
b	3
c	5
c	6

$T_1 \cap T_2 =$

A	**B**
b	3
c	6

Note that, just as with intersection on relations, we only keep the rows (tuples) found in both tables. Similar to union, the intersection of two tables is only possible if the schema are identical. The following intersection fails, as the tables are not compatible.

$T_1 =$

A	**B**
a	1
b	2
b	3
c	6

$T_2 =$

C	**D**
a	2
a	4
b	3
c	5
c	6

$T_1 \cap T_2 =$ *undefined*, no result

Difference

The difference operator is denoted, as usual, by $-$. The difference of two relational tables is the same as the difference of two relations, except that with tables the schemas must be the same (as we saw with union and intersection). Here is an example:

$T_1 =$

A	**B**
a	1
b	2
b	3
c	6

$T_2 =$

A	**B**
a	2
a	4
b	3
c	5
c	6

$T_1 - T_2 =$

A	**B**
a	1
b	2

Note that, just as with intersection on relations, we only keep the rows (tuples) in the table on the left-hand side that are not also in the table on the right-hand side of the operator. The following difference fails, as the tables are not compatible—the schemas do not match.

$T_1 =$

A	**B**
a	1
b	2
b	3
c	6

$T_2 =$

C	**D**
a	2
a	4
b	3
c	5
c	6

$T_1 - T_2 =$ *undefined*, no result

Cross Product

We defined a relation as a subset of the Cartesian product of sets, e.g. $R = A \times B$. The cross product uses the same product symbol, $\times$, but is performed on two relations. It can be thought of in the same way, that is, that it creates all possible combinations. Consider the following example:

$T_1 =$

A	B
a	1
b	2

$T_2 =$

C	D
a	1
c	5
c	6

$T_1 \times T_2 =$

A	B	C	D
a	1	a	1
a	1	c	5
a	1	c	6
b	2	a	1
b	2	c	5
b	2	c	6

Note that the schema must be completely different. If there is a column found in both schema the cross product is undefined. If the cross product is desired in this situation, the columns in common must be renamed to unique labels; then the cross product can be performed.

$T_1 =$

A	B
a	1
b	2

$T_2 =$

A	B
a	1
c	5
c	6

$T_1 \times T_2 =$ *undefined*, must use $T_1 \times \rho_{A \leftarrow C, B \leftarrow D}(T_2)$

Natural Join

The natural join (sometimes referred to only as join) is the operation performed that joins two tables on any common columns. The natural join is denoted as $|\times|$, or $\bowtie$. When two tables have the exact same schema, the result of the natural join is the same as the intersection. When two tables have unique schema, the result of the natural join is the same as the cross product. In the examples below the matching columns will be highlighted in blue with the matching rows highlighted in yellow.

$T_1 =$

A	B
a	1
b	2

$T_2 =$

A	B
a	1
c	5
c	6

$T_1 |\times| T_2 =$

A	B
a	1

$T_1 =$

A	B
a	1
b	2

$T_2 =$

C	D
a	1
c	5
c	6

$T_1 |\times| T_2 =$

A	B	C	D
a	1	a	1
a	1	c	5
a	1	c	6
b	2	a	1
b	2	c	5
b	2	c	6

So what does the natural join do when there are only some common columns of the schema? It essentially performs the cross product and then selects and projects out the common columns. Here is an example:

$T_1 =$

A	B
a	1
b	2

$T_2 =$

A	D
a	1
c	5
c	6

$T_1 |\times| T_2 =$

A	B	D
a	1	1

Note that we were able to join on one, common column, `A`. We find the rows in both tables that have matching values in column `A` and then join those rows. Let's see an example of joining on two columns:

$T_1 =$

A	B	C
a	1	x
a	2	x
b	2	y
c	5	y

$T_2 =$

A	B	D
a	2	1
c	5	3
c	5	6

$T_1 |\times| T_2 =$

A	B	C	D
a	2	x	1
c	5	y	3
c	5	y	6

Note that the ordering of the columns in the resulting table does not have to be in any particular order (the example is in alphabetical order for clarity). Also, the ordering of common columns in T_1 and T_2 is irrelevant – we would get the same result if T_1's schema was `B`, `A`, `c`, instead of `A`, `B`, `C`. Therefore:

$T_1 =$

B	A	C
1	a	x
2	a	x
2	b	y
5	c	y

$T_2 =$

A	B	D
a	2	1
c	5	3
c	5	6

$T_1 |\times| T_2 =$

A	B	C	D
a	2	x	1
c	5	y	3
c	5	y	6

Evaluating Rules

In order to evaluate all Datalog rules with our relational tables, we will need to use the natural join operation. Consider again the rule from our Datalog program above:

```
mySCheme('c', X) :- myScheme('a', X), myScheme('b', X).
```

This rule can be thought of logically that $\forall$ `x [myScheme('a', X)` $\wedge$ `myScheme('b', X)` $\rightarrow$ `myScheme('c', X)]`. The first step is to evaluate each predicate on the right-hand side of the implication (the *left-hand* side of the colon-dash token) just as we evaluate queries to create intermediate relational tables (the results of the predicate evaluation). The second step is to join all intermediate relational tables that correspond to the predicates. This final table now contains new facts. It is important to understand that once new facts have been created there is a possibility for the rules to generate even more facts (based on the new facts just generated). That means we need to iterate the two step process again. We need to keep iterating until no new facts are generated. An algorithm that continues to run until it converges on the final answer is called a fixed-point algorithm. Our evaluation of rules using the relational data model is a fixed-point algorithm (this entire process is outlined in extensive detail in the Project 4 document by Dr. Goodrich).

Order of Operations

All unary operators are performed before binary operators. Also, the unary operators are performed right to left (inside to outside). Thus for the expression $\rho_{A \leftarrow Z} A \,|\!\times\!|\, B$ the unary operation on A is performed first. The join is performed next. If we want to perform a unary operation on the result of a join we need parentheses: $\pi_Z(A \,|\!\times\!|\, B)$. Now the join on A and B is performed first. We then project the column Z (assuming it is a valid column) from the result of the join.

Conclusion

Relational data models are the basis for modern databases. You will implement the Datalog interpreter by storing schemes and facts (Project 3), and then by generating new facts from rules (Project 4). In both projects you will evaluate the queries (some queries will fail without the generation of new facts). Make sure you understand those sections of this document before starting the projects. See the book, Section 9.2.1-9.2.5* for further examples and details (note: the book uses different symbols for the operations described in this document).

**Discrete Mathematics and Its Applications*, by Kenneth H. Rosen.