CSCI 189 Assignment 5 Problem Set

This assignment provides an opportunity to learn about relations and their applications to database design, information retrieval, and Web searching.

Summary of Concepts in Section 4.1 and 4.3

The following table will be used again in the discussion below:

OurClass

Name	*Class*	*Home*	*Calc*	*Prog*	*Career*	*Subject*	*Xwords*
Chris	2007	MA	no	yes	computers	cs	yes
Evan	2007	NC	hs	yes	unk	cs	no
Jarrett	2008	MD	yes	yes	computers	cs	yes
Jason	2010	VA	no	no	pilot	aeronautics	yes
Joe	2008	MA	yes	yes	banking	econ	yes
John	2008	NY	yes	yes	unk	cs	yes
Karina	2007	NY	yes	no	unk	anthro	no
Kate	2006	ME	hs	no	unk	english	yes
Matt	2009	MA	hs	no	business	unk	yes
Nolan	2008	CA	yes	yes	computers	cs	yes

Relations

A *binary relation* ρ on a set S is a subset of $S \times S$. A binary relation ρ on sets S and T is a subset of $S \times T$. An n-ary relation ρ on sets $S_1, S_2, \ldots, S_n$ is a subset of $S_1 \times S_2 \times \ldots \times S_n$. We think of ρ itself as a *property* that all members of the relation share.

E.g., the relation $\rho_1 = \{(0,0),(1,1),(2,4),(3,9),\ldots\}$ describes the set of pairs (x, y) in $N \times N$ which share the property $y = x^2$.

E.g., the relation $\rho_2 = \{(chris, cs),(evan, cs),(jarrett, cs),(joe, econ),(john, cs),(nolan, cs)\}$ describes the set of pairs (x, y) in $Names \times Subject$ which share the property that x's probable major subject is computer science or economics (i.e., $y = cs \vee y = econ$).

A relation ρ is *one-to-one* if $\forall(x, y) \in \rho$, x appears only once and y appears only once. For example, ρ_1 is one-to-one, but ρ_2 is not. A relation ρ is *one-to-many* (or *many-to-one*) if $\exists(x, y) \in \rho$ such that x (or y) appears somewhere else in ρ. For example, ρ_2 is many-to-one, since two different people have the same favorite subject, but it is not one-to-many.

A relation is *many-to-many* if it is both one-to many and many-to-one. For example, $\rho_3 = \{(2007, NY),(2008, NY),(2008, MD)\}$ is many-to-many. That is, there are more than one home

states for members of the class of 2008, and there are more than one classes represented by the same home state (NY).

A relation ρ on a set S is:

reflexive if $\forall x(x \in S \rightarrow (x, x) \in \rho)$.

symmetric if $\forall x \forall y(x, y \in S \wedge (x, y) \in \rho \rightarrow (y, x) \in \rho)$.

transitive if $\forall x \forall y \forall z(x, y, z \in S \wedge (x, y), (y, z) \in \rho \rightarrow (x, z) \in \rho)$.

antisymmetric if $\forall x \forall y(x, y \in S \wedge (x, y) \in \rho \wedge (y, x) \in \rho \rightarrow x = y)$.

A relation that is reflexive, symmetric, and transitive is called an *equivalence relation.*

For example, suppose $S = Names$ and $\rho_4 = \{(x, y) \in Names \mid$ x and y are in the same graduating class$\}$. Thus, $\rho_5 = \{(chris, chris), (chris, evan), (chris, karina), (evan, chris), (evan, evan), (evan, karina), \ldots\}$. This is an equivalence relation, since 1) each person is in the same graduating class as him/herself, 2) if x is in the same class as as y then y must be in the same class as x, and 3) if x is in the same class as y and y is in the same class as z, then surely x must be in the same class as z.

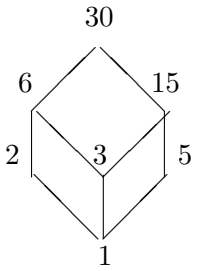
A *partition* of S is a collection of nonempty disjoint subsets $S_1, S_2, \ldots, S_n$ such that $S_1 \cup S_2 \cup \ldots S_n = S$. If ρ is an equivalence relation on S, then $[x]$ denotes the set of all members of S to which x is related, and is called the *equivalence class* of x. That is, $[x] = \{y \mid y \in S \wedge (x, y) \in \rho\}$. An equivalence relation ρ on S defines a *partition* of S, and conversely.

For example, since ρ_4 is an equivalence relation, we can find the equivalence class for, say, *chris*, as: $[chris] = \{chris, evan, karina\}$, since these two are in the same graduating class and nobody else is. Similarly, the equivalence class $[jarrett] = \{jarrett, joe, john, nolan\}$, and so forth.

A *partial ordering* is a binary relation on a set S that is reflexive, antisymmetric, and transitive. For example, $\leq$ is a partial ordering on Z (the integers), since 1) every integer is less than or equal to itself, 2) if $x \leq y$ and $y \leq z$ then $x \leq z$, and 3) if $x \leq y$ and $y \leq x$ then $x = y$. But $<$ is not a partial ordering on the integers, since it is not reflexive (no integer is less than itself!). Similarly, $\subseteq$ is a partial ordering on a set, while $\subset$ is not.

A *Hasse diagram* of a partial ordering is a graphical representation in which each pair (x, y) is connected by a line and x is placed below y, as shown:

Below is a Hasse diagram for the partial ordering $\rho_6 = \{(x, y) \mid x, y \in \{1, 2, 3, 5, 6, 15, 30\}$ and x divides y$\}$.



To construct such a diagram, we can start with the largest element y in the set and place it at the top. Then write down, just below y, every element x in the set for which 1) there's a pair (x, y) in the relation, and 2) there is no $z > x$ whose pair (x, z) is in the relation. Now connect each such x with y. Repeat this step again at the next lower level in the diagram, and continue until all the elements in the set appear exactly once. For example, the above diagram would have been constructed beginning with 30, and then connecting 6 and 15 to it (but no others, since the rest divide either 6 or 15 or both).

For a counterexample, consider $\rho_5 = \{(x, y) \in Names \times Names \mid$ x graduates no later than y$\}$. This is reflexive, since everyone graduates no later than him/herself. It is also transitive, since if x graduates no later than y and y graduates no later than z, then surely x graduates no later than z. However, this relation is not a partial ordering, since it is not antisymmetric. That is, the fact that x graduates no later than y and y graduates no later than x doesn't force x and y to be the same person. For example, the pair $(joe, john)$ is in ρ_5 as is the pair $(john, joe)$, but they are different. Some of the members of this relation are: $\{(joe, joe), (john, john), (john, joe), (joe, matt), (joe, jason), \ldots\}$

Relational Databases

In the relational database model, a *table* is a set of n-tuples (rows), and each row is called a *tuple*. Each column in the table contains values of a certain *attribute*, and the number n of attributes (columns) is th *degree* of the relation. The table at the beginning of this handout is such a table, with attributes $A_1 = Name$, $A_2 = Class$, etc.

A *relational database* is a subset of $D_1 \times D_2 \times \ldots \times D_n$, where each D_i is the domain from which attribute A_i takes its values. So a relational database is consistent with the idea of an "n-ary relation" on multiple sets.

The *primary key* of a database is a (set of) attribute(s) that can be used to uniquely identify each tuple (row). E.g., the attribute *Name* is a primary key of our database, since everyone in the class has a different name.

There are three major operations that can be performed on relational databases D, E, and F: **restrict**, **project**, and **join**. Here is a definition of each one, with examples.

The **restrict** operation creates a new relation E from a relation D by selecting only those rows that meet a certain "condition," which is a logic expression on the attributes of D. It is written as: **restrict** D **where** condition **giving** E

For example, if we wanted to create a new relation "Juniors" from our class (call the whole table "OurClass"), we would write:

restrict OurClass **where** $Class = 2007$ **giving** Juniors

which would create the following table:

Juniors

Name	*Class*	*Home*	*Calc*	*Prog*	*Career*	*Subject*	*Xwords*
Chris	2007	MA	no	yes	computers	cs	yes
Evan	2007	NC	hs	yes	unk	cs	no
Karina	2007	NY	yes	no	unk	anthro	no

The **project** operation creates a new relation E from a relation D by including only certain attributes and discarding the rest, and eliminating duplicates. It is written as:

project D **over** (attributes) **giving** E

For example, if we wanted to define the relation OurBackgrounds (that is, all the combinations of programming and calculus backgrounds represented by different members of our class), we would write:

project OurClass **over** (Prog, Calc) **giving** OurBackgrounds

This would yield the following table:

OurBackgrounds

Prog	*Calc*
no	yes
hs	yes
yes	yes
no	no
yes	no
hs	no

The **join** operation is useful when there are two or more relations, say D and E that share the same value for a particular attribute, creating a new relation F. It is written as:

join D **and** E **over** (attribute) **giving** F

For example, suppose our class database had three tables rather than the one shown on the first page. That is, suppose one table called *Roster*, contains a list of Name, Class, and Home; one called *Backgrounds* contains a list of Name, Calc, Prog, and XWords; and one called *Interests* contains a list of Name, Career, and Subject. That is, the information for our class is distributed among the following three tables rather than one:

Roster

Name	*Class*	*Home*
Chris	2007	MA
Evan	2007	NC
Jarrett	2008	MD
Jason	2010	VA
Joe	2008	MA
John	2008	NY
Karina	2007	NY
Kate	2006	ME
Matt	2009	MA
Nolan	2008	CA

Backgrounds

Name	*Calc*	*Prog*	*Xwords*
Chris	no	yes	yes
Evan	hs	yes	no
Jarrett	yes	yes yes	
Jason	no	no	yes
Joe	yes	yes	yes
John	yes	yes	yes
Karina	yes	no	no
Kate	hs	no	yes
Matt	hs	no	yes
Nolan	yes	yes	yes

Interests

Name	*Career*	*Subject*
Chris	computers	cs
Evan	unk	cs
Jarrett	computers	cs
Jason	pilot	aeronautics
Joe	banking	econ
John	unk	cs
Karina	unk	anthro
Kate	unk	english
Matt	business	unk
Nolan	computers	cs

Now if we want to make a single table listing only the *Name* and *Class* of all students in our class who like crosswords, we would need to combine two different tables to derive the information we need. This is the purpose of the `join` operation. To do a join, the two tables would have to share a common attribute, such as *Name* in our case, which would be the basis for joining them (Notice the use of **restrict** here to select only rows for people who like crosswords).

join Roster **and** (**restrict** Backgrounds **where** XWords = 'yes') **over** (Name)

To complete this activity, we need to **project** that result so that it shows only the Names and Classes of those who do like crosswords by saying:

project (**join** Roster **and** (**restrict** Backgrounds **where** XWords = 'yes') **over** (Name)) **over** (Name, Class) **giving** LikesXWords

This creates a new relation that looks like this:

LikesXWords

Name	*Class*
Chris	2007
Jarrett	2008
Jason	2010
Joe	2008
John	2008
Kate	2006
Matt	2009
Nolan	2008

Using SQL to Access Relational Databases

SQL (for "Structured Query Language") is a programming language that allows people to retrieve information from relational databases such as this one. Its syntax is a bit awkward, but it allows various kinds of information to be retrieved. Here is the beginning of an SQL session using the "OurClass" database described on the first page of this document.

```
mysql> select * from OurClass;
+---------+-------+------+------+------+-----------+--------------+--------+
| Name    | Class | Home | Calc | Prog | Career    | Subject      | Xwords |
+---------+-------+------+------+------+-----------+--------------+--------+
| Chris   | 2007  | MA   | no   | yes  | computers | cs           | yes    |
| Evan    | 2007  | NC   | hs   | yes  | unk       | cs           | no     |
| Jarrett | 2008  | MD   | yes  | yes  | computers | cs           | yes    |
| Jason   | 2010  | VA   | no   | no   | pilot     | aeronautics  | yes    |
| Joe     | 2008  | MA   | yes  | yes  | banking   | econ         | yes    |
| John    | 2008  | NY   | yes  | yes  | unk       | cs           | yes    |
| Karina  | 2007  | NY   | yes  | no   | unk       | anthro       | no     |
| Kate    | 2006  | ME   | hs   | no   | unk       | english      | yes    |
| Matt    | 2009  | MA   | hs   | no   | business  | unk          | yes    |
| Nolan   | 2008  | CA   | yes  | yes  | computers | cs           | yes    |
+---------+-------+------+------+------+-----------+--------------+--------+
10 rows in set (0.00 sec)
```

The general form of an SQL statement, where D and E again represent relations (SQL calls them "tables"), is shown below:

```
CREATE TABLE E
   SELECT attributes FROM D
   WHERE condition
      AND/OR condition  ... ;
```

Here, the CREATE line is like the "giving" clause in the **restrict**, **project** and **join** operations discussed above. Moreover, the `attributes` in the SELECT line are like the **project** operation,

and the `conditions` in the WHERE and AND/OR lines are like the **restrict** operation.

For example, the new relation *Sophomores* defined earlier can be created using the following SQL commands:

```
mysql> create table Juniors
    -> select * from OurClass where Class = 2007;
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> select * from Juniors;
+--------+-------+------+------+------+-----------+------------+--------+
| Name   | Class | Home | Calc | Prog | Career    | Subject    | Xwords |
+--------+-------+------+------+------+-----------+------------+--------+
| Chris  | 2007  | MA   | no   | yes  | computers | cs         | yes    |
| Evan   | 2007  | NC   | hs   | yes  | unk       | cs         | no     |
| Karina | 2007  | NY   | yes  | no   | unk       | anthro     | no     |
+--------+-------+------+------+------+-----------+------------+--------+
3 rows in set (0.01 sec)
```

Note here that the asterisk (*) means "select all columns." To select particular columns, their headings are listed in place of the asterisk. For example, the relation *OurBackgrounds* can be generated by selecting the columns *Calc* and *Prog* in the following way:

```
mysql> create table OurBackgrounds
    ->   select distinct Prog, Calc from OurClass;
Query OK, 5 rows affected (0.02 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select * from OurBackgrounds;
+------+------+
| Prog | Calc |
+------+------+
| no   | yes  |
| hs   | yes  |
| yes  | yes  |
| no   | no   |
| yes  | no   |
| hs   | no   |
+------+------+
6 rows in set (0.01 sec)
```

The word `distinct` here specifies that duplicate pairs, like two instances of `(yes, no)`, not appear in the table.

Finally, the new relation *LikesXWords* can be generated using the SQL `natural join` command, which looks like this:

```
mysql> create table LikesXwords
    ->  select Roster.Name, Roster.class from Roster natural join Backgrounds
    ->     where Xwords = 'yes';
+---------+-------+
| Name    | class |
+---------+-------+
| Chris   | 2007  |
| Jarrett | 2008  |
| Jason   | 2010  |
| Joe     | 2008  |
| John    | 2008  |
| Kate    | 2006  |
| Matt    | 2009  |
| Nolan   | 2008  |
+--------+-------+
8 rows in set (0.00 sec)
```

If you would like to experiment with SQL, you can download and install it on your computer. The free version used here is called `mysql` and it can be obtained from the Web site `http://www.mysql.com/`.

The file from which the above examples are drawn is called `OurClass.sql` and it can be directly downloaded from this Web location.

Problems to be handed in

You are welcome to work in groups of 2 or 3 to complete this assignment. Each group member should contribute a fair share of the work, and the group should turn in one set of answers (listing the names of group members at the top).

Section 4.1 (p 262) Exercises 2ac, 6ac, 8ac, 9ace, 16bd, 23a, 25c, 36, 37.

Section 4.3 (p 287) Exercises 5, 7, 10, 13, 16, 18 (use only relational algebra and SQL for 16 and 18, but not relational calculus), 19.

Extra credit (optional) Do exercise 30 on page 266.