Relational Databases

Reading / References

- Concise Guide to Databases: A Practical Introduction[1] chapter 4
- Introduction to SQL: Mastering the Relational Database Language[2]
- MySQL[3]

Practice questions on the reading

-

Notes

Basic concepts of Relational Databases

Essential in the structure of relational databases is the notion of a table. A **table** contains specific columns, each holding values of a specified type. The table is filled with rows, containing a value for each column. These rows are often called **tuples**, and the table is often called a **relation**. The columns are often refered to as **attributes**. For example the relation student might look something like this:

login	first	last
st1	Joe	Somebody
st2	Peter	Other
st3	Mary	Somebody

This relation has three attributes, login, first, and last, all containing strings. It also has 3 tuples/records corresponding to three students.

Every table/relation must also contain a **primary key**. A primary key is an attribute or combination of attributes that uniquely identify each row/record/tuple. In the example above, the login attribute itself can serve as a primary key. If we were discussing storing courses, a combination of the department prefix and course number can act as the primary key for the course table.

Many tables may also contain **foreign keys**. These are attributes of a table that point to the primary keys in another table. For instance the course table will contain a foreign key identifying the department that a course is from. Foreign Keys is what links tables together.

[1] http://learning.acm.org/books/book_detail.cfm?id=2560109&type=24
[2] http://learning.acm.org/books/book_detail.cfm?id=1208031&type=safari
[3] http://learning.acm.org/books/book_detail.cfm?id=2484635&type=safari

Normalization

An important process in database design is that of normalization. *Normalization** is a process that aims to rewrite a database in such a way as to minimize the duplication of data values. This often leads to a reduced storage space, at the cost of creating possibly more tables. This is done in multiple steps, via the use of so called "normal forms".

As an example, consider a list of enrollment of students in classes. As a first approximation, we might have a long table that looks like this:

login	first	last	courseNo	title	term	year	tags
st1	Joe	Somebody	MAT121	Calculus 1	Fall	2016	online, afr
st2	Peter	Other	MAT121	Calculus 1	Winter	2016	afr
st3	Mary	Somebody	MAT121	Calculus 1	Winter	2016	afr
st1	Joe	Somebody	MAT122	Calculus 2	Winter	2016	online, afr
st2	Peter	Other	CS220	Intro to CS	Fall	2016	afr
st1	Joe	Somebody	CS220	Intro to CS	Fall	2017	afr

You could imagine this list going on and on for a while. There is a certain degree of redundancy, and normalization aims to reconsile that. It is typically expressed as dependence between the attributes.

First Normal Form First Normal Form essentially dictates that we cannot have multiple values in one "cell". In our example above we have violated that principle with the presence of the "tags" attribute, for which there are on occasion two tags for the same item. In order to be in First Normal Form, we cannot allow this and must instead have different rows, one for each of the possible values. So the table would instead look like this:

login	first	last	courseNo	title	term	year	tag
st1	Joe	Somebody	MAT121	Calculus 1	Fall	2016	online
st1	Joe	Somebody	MAT121	Calculus 1	Fall	2016	afr
st2	Peter	Other	MAT121	Calculus 1	Winter	2016	afr
st3	Mary	Somebody	MAT121	Calculus 1	Winter	2016	afr
st1	Joe	Somebody	MAT122	Calculus 2	Winter	2016	online
st1	Joe	Somebody	MAT122	Calculus 2	Winter	2016	afr
st2	Peter	Other	CS220	Intro to CS	Fall	2016	afr
st1	Joe	Somebody	CS220	Intro to CS	Fall	2017	afr

This seemingly increases the amount of duplicate entries, but it is an essential first step.

Before we move on to discuss normalization further, we need to discuss the idea of functional dependencies and keys:

Functional dependencies and keys We say that an attribute B is **functionally dependent** on another set of attributes A1, A2, . . . , if for any two tuples that have the same value on each of the attributes A1, A2, . . . it is also the case that the tuples have the same value for B. In other words, if two rows agree on the A attributes, they would also have to agree on the B attribute.

Looking at our example above, we can see that "term" does not functionally depend on "login", as we have rows that have the same login but different terms. On the other hand, both "first" and "last" are functionally dependent on "login", as each time "login" has a certain value this also determines the value for "first" and "last".

There are always some **trivial** functional dependencies. For example an attribute A3 is always functionally dependent on a set of attributes "A1, A2, A3" that contains it. Any attribute is naturally dependent on any set of attributes containing it. We call these trivial dependencies. All other dependencies are called **non-trivial**.

We say that a set of attributes is a **key** for the relation/table, if any attribute in the table is functionally dependent on that set of attributes. In other words, the values at those attributes completely determine all other values.

An extreme example of a "key" is the set of all attributes. If we know all the values in a row, then we can obviously determine all values in a row.

In our example table above, a key would be the set of "login, courseNo, term, year, tag". Knowing these 5 values would determine the rest. It is also in a certain sense a "minimal" key; we could not remove any of the attributes from the set and still expect the rest to determine everything. A key that is "minimal" is called a **candidate key**.

What we earlier called a **primary key** is essentially a candidate key that we have designated as special.

Attributes that are in some candidate key are called **prime**, and other attributes are called **non-prime**. In the example above "last" is non-prime, as it would never be a part of a candidate key, at least not if we allow the possibility of two students with the same name: Any candidate key will need to include the "login" attribute in order to distinguish between the different students, and in that case the "last" attribute becomes redundant. Similarly, "first" and "title" are non-prime attributes. All other attributes are prime.

Boyce-Codd Normal Form We say that a relation/table is in BCNF, if whenever we have a non-trivial functional dependency then the set of attributes that determine the other attribute must be a key.

For instance the fact that there is a functional dependency between "login" and "first" in the table above means that that relation is not in BCNF form, as "login" is not itself a key (though it is part of a key). The idea is that since "login" alone determines some other attributes, then it should form a separate table along with those attributes.

There is a process that allows us to **decompose** any relation into Boyce-Codd Normal Form. It goes roughly like this:

- Start with a functional dependency that violates BCNF: So there is a set "A1, A2, . . . " of attributes that does not form a key, and they determine some attribute B1.
- Determine any other attributes that functionally depend on the "A1, A2, . . . ". Let's called these B2, B3, etc.
- Break our relation in two: One that contains the As and the Bs only. And another that contains the As, none of the Bs, and also any other attributes in the original table.
- Repeat the process with these tables.

Let's apply this idea to our table. We see that login functionally determines first. It also functionally determines last, and none of the other attributes. We will therefore break it into two tables, one containing login, first and last, and another containing login and the rest. And thus the student table is born.

login	first	last
st1	Joe	Somebody
st2	Peter	Other
st3	Mary	Somebody

login	courseNo	title	term	year	tag
st1	MAT121	Calculus 1	Fall	2016	online
st1	MAT121	Calculus 1	Fall	2016	afr
st2	MAT121	Calculus 1	Winter	2016	afr
st3	MAT121	Calculus 1	Winter	2016	afr
st1	MAT122	Calculus 2	Winter	2016	online
st1	MAT122	Calculus 2	Winter	2016	afr
st2	CS220	Intro to CS	Fall	2016	afr
st1	CS220	Intro to CS	Fall	2017	afr

Now we continue this process, and notice that courseNo functionally determines title. We see if it functionally determines anything else, but it does not. We therefore split the course number and title apart, and this way the course table is born:

login	first	last
st1	Joe	Somebody
st2	Peter	Other
st3	Mary	Somebody

courseNo	title
MAT121	Calculus 1
MAT122	Calculus 2

courseNo	title
CS220	Intro to CS

login	courseNo	term	year	tag
st1	MAT121	Fall	2016	online
st1	MAT121	Fall	2016	afr
st2	MAT121	Winter	2016	afr
st3	MAT121	Winter	2016	afr
st1	MAT122	Winter	2016	online
st1	MAT122	Winter	2016	afr
st2	CS220	Fall	2016	afr
st1	CS220	Fall	2017	afr

Third Normal Form The "second" normal form is not much in use today, we therefore won't discuss it. We will also not discuss third normal form as it is fairly close to BCNF. You are encouraged to explore the topic on your own.

Multivalued Dependencies and the Fourth Normal Form TODO