Using a Relational Database for Data Entry and Analysis

Anthony Koth
Rice University, Linguistics
amk5@rice.edu

Table of contents

1 Introduction
2 What is a relational database
3 The right tools
4 Creating the database
4.1 Think about the data as computational data
4.2 Structure and tables
4.3 Relating the tables (linking)
4.4 Populating the database
4.4.1 Appending
4.5 Creating forms
4.6 Coding
5 Using SQL
5.1 SQL Syntax primer
5.2 Basic SQL queries
5.3 Saving queries, creating tables (Views) of query results and more complex SQL queries
5.4 Queries for the whole data set exportable to statistical software
5.5 UPDATE
6 Conclusion
7 Additional resources

Keywords:* *Data Entry, Data Analysis, Relational Database

1 Introduction

If you have ever gotten lost in scribbles, rows, columns and highlighting while trying to use a spreadsheet or pen and paper for coding linguistic data, then a relational database is a tool for you. It offers a clean and manageable way to code all of your data consistently and accurately. This paper is not meant to be a complete manual for using a database or all of the complexities of the Structured Query Language (SQL) which powers its searching and counting functions. Rather, this paper serves as a primer or how-to for using a relational database for linguistic analysis. Much of the information around the internet related to databases was not straightforwardly translatable to a linguist's needs, in particular the SQL code needed for counting hierarchical data (multiple different attributes with multiple different values). I hope the following can help you start utilizing this powerful tool with fewer growing pains.

2 What is a relational database

A relational database allows users to create different **Tables** (i.e. spreadsheets) and then **Relate** (that is link) them to one another. Linking the tables allows data tokens to enter into a one-to-many relation with the various codes you assign. That is, it is possible to code as many different features, in any combination, as needed for each

specific token number, and then the program keeps track of it all. And because you have identified features as specifically related to a given token number, you can count instances accurately and consistently. To illustrate, in my research, I conducted a corpus analysis of the lexemes 'gay' and 'homosexual'. I gathered 518 tokens and coded upwards of 20 features for each token such as grammatical category, syntactic relation, website, etc. The total data table stands at just shy of 15,000 codes. Trying to code that much or count any of that consistently using a spreadsheet would be nearly impossible.

Once the database is set up, you can create **Forms** to enter the data quickly. A form is a single window that allows you to view and edit any number of data tables in one convenient location. You can then run SQL queries to find and count things. For example, if I have 518 tokens which are coded for syntactic position, I can query the database to remind me what unique syntactic positions I have coded for (e.g. subject, direct object, direct object relativizer etc.) to ensure I am consistently coding data. With a slightly different query I can count each instance of those to see how often the data occur as subjects versus direct objects etc. Change the query yet again and I can see how many instances of each occur for 'gay' and how many for 'homosexual' to see how the two lexemes behave differently in their syntactic distribution. The counts will be internally consistent because the program is tracking everything based on the unique token number. You can then export these counts as spreadsheets for use in various statistical software packages.

3 The right tools

My experiences are limited to the database program that comes built in to the OpenOffice software suite; however, many of these basics should be portable to whatever software suite you use. But, before I could even use the database, I had to clean up the data with text editors. TextWrangler is a versatile text editor for the Mac operating system which has support for regular expressions (see §7 Additional Resources). It was useful in cleaning up the articles I downloaded from the internet, allowing me to strip out the HTML, add line numbering, and find and extract the tokens themselves out of the article etc.

As I was coding, I had to add certain features in bulk to the database. After a few passes through the first 50 tokens or so, I figured out the main attributes I wanted to code for in all the tokens. I created spreadsheets that contained those attributes, combined with the remaining token numbers and imported those into the database (see §4.4.1 Appending). Five minutes in a spreadsheet saved me roughly 1 hour of retyping the same attributes. However, I spent probably one hour surfing online trying to figure out how to do the same thing in SQL. All of this is to say that there are certain tools for certain jobs. If you find yourself spending too much time trying to solve a problem, take a moment to think if you have other tools at your disposal that can handle the job better.

4 Creating the database

4.1 Think about the data as computational data

Once you have your data, take a moment to think about it as computational data, as information that you're going to ask a computer to handle for you. The computer can

only handle exact matches when it comes to searching. Typographic issues can be problematic – e.g. Homosexual vs. homosexual vs. homosexuals vs. homosexuals' vs. Homosexual's.[1] How are you going to abstract over your individual tokens to get at some kind of useable types? Thinking about your data, decide what broad factors might become useful in your categorization and organization. Do you want to be able to count things based on the title or type of the text, the speaker, the type of verb, where you got the text etc? Each of these questions offers clues as to how your data should be organized – how many tables do you need, what kind of features are you going to code for, where should these codes go etc. Having taken a moment and answered some of these questions at the outset would have saved me considerable time in terms of data consistency for searches and being able to do certain types of searches. Let's walk through some of these issues as related to my data.

You should have some kind of raw data – articles, interlinearized texts, KWIC lines from an online corpus etc. For me, the raw data was a corpus of downloaded articles (after the HTML had been stripped out). Each of these should have unique identifying information – text number / title and line number, KWIC line number etc. I used a convention of site and line number. Each line was labeled along the lines of XXX.###. I saved this raw data as a spreadsheet. Hereafter I refer to these initial spreadsheets as source spreadsheet for clarity.

From this raw data, you need to pull out the actual data you want to analyze – lexical items, verbs, phonemes, constructions etc. For me, they were the actual instances of 'gay' and 'homosexual' in the articles. Using TextWrangler I deleted all the HTML and other sentence text that was not 'gay' or 'homosexual', kept the appropriate line number where it came from, then added a unique token number (1 – 518). Save this as your token data. I then copied the raw data and the token data into separate source spreadsheets for further manipulation.[2]

Note that each of these source spreadsheets should have multiple columns already. The raw data has at minimum some kind of ID information and then the raw text. The following sample comes from an article in my corpus – note the identifying information in the SITELINE column, then the actual raw text of the article (LINETEXT).

SITELINE	LINETEXT
ADV.006	homosexuality, a spokeswoman for the school says. The campus's unofficial LGBT
ADV.007	student group released the video, which includes several students talking about
ADV.008	being LGBT and Mormon. BYU's official honor code states that "homosexual
behavior	
ADV.009	is inappropriate and violates the honor code. Homosexual behavior includes not

The token data has at minimum the unique token number, the line number (SITELINE) it came from and the token itself:

TOKENNUMBER	SITELINE	TOKEN	LEXITEM	WC	SITE
1	ADV.006	homosexuality	HOMOSEXUAL	HITY	ADV

1 PRO TIP: use all caps or all lowercase and avoid using spaces. This keeps you from having to remember whether something did or did not have an initial capital or a space somewhere.

2 PRO TIP: I've heard basic programming skills like Python can help in this regard. As I do not know Python, I cannot speak to how easy it is to accomplish this task in that manner.

2	ADV.008	homosexual	HOMOSEXUAL	ADJ	ADV
3	ADV.009	homosexual	HOMOSEXUAL	ADJ	ADV

Depending on how you answered the other questions above, you may want to add some columns now. As I was coding and analyzing my data, I realized I needed to standardize my spellings so that all the different types of 'homosexual' and 'gay' could be found (LEXITEM), that I wanted to be able to count by word class (WC - ADJective vs. noun etc.), by website (SITE), by political affiliation and so on. Much of this information could be read right off the token or the raw data itself. If you can figure something out based solely on the data as data, add it now; you'll be surprised how useful you'll find this. Save the coding time for things that require thinking about the data as linguistic data (clause type, grammatical role etc).

Certain features made sense to add to the token data source spreadsheet – word class, standardized spelling, website etc. Others made sense to add to the raw data – website, political affiliation etc. Had I done these at the outset, I would have saved upwards of 3 – 5 hours in coding, appending and searching for SQL codes to count things. Those features requiring linguistic analysis (animacy of subject, number of arguments etc.) get coded on the attributes table in the database.

Resist the temptation to have multiple source spreadsheets of raw data and of token data. In a previous incarnation of this study I had 6 source spreadsheets of data that I turned into 6 database tables that contained data and coding combined. In order to do any kind of searching I had to query each of those 6 tables independently which cost me time. For the present study I had 2 source spreadsheets, raw data and token data, which I turned into two of the three database tables.

4.2 Structure of the database tables

The following three-table format was what I used for my current project, and it seems quite flexible and computationally efficient for linguistic analysis.[3] Other additional tables might be useful for your analysis however.

Table 1: Three-table database structure

Table Type	Contents
DATA[4]	Raw data
TOKEN	The tokens, this is the heart of the database
ATTRIBUTES	Coding

To create these database tables, you can either use the 'Wizard' or 'Create Table' feature in 'Design View'. You'll have to define the number of **Fields**, analogous to the columns in your source spreadsheet, as well as the **Type** of data that the field can accept.

3 The naming conventions are generic and you are free to change them with one caveat: make sure you remember how you've changed them if you refer to the SQL commands below.

4 PRO TIP: I capitalized everything – tables, fields and codes.

The DATA table will contain the raw data source spreadsheet. For mine, there were four fields: SITELINE (XXX.###), SITE (XXX), LINE (###), and LINETEXT (raw data). The type of data for each field is 'Text', 'Text', 'Numeric' and 'Text' respectively. At this point, you may need to address other specifications as well. For instance, some database programs might require a definition of maximum length. My LINETEXTs of data were between 100-120 characters long so I set the **Length** to 150 to be safe. In OpenOffice, the DATA table does not require a **Primary Key**, which is the column of numbers that are used to relate two tables together; however, other database programs might.

The TOKEN table will likely vary from project to project and contains the data from the token data source spreadsheet – for my project, it was the instances of 'gay' and 'homosexual'. It contains a minimum of two fields – TOKENNUMBER and TOKEN. The TOKENNUMBER Field must be 'Numeric' and set as the Primary Key as you will use this number to Relate this table to the ATTRIBUTES table. Other Fields that might be useful include the line number and actual line of data where the token comes from (SITELINE and LINETEXT from my DATA table) for quick reference.

The ATTRIBUTES table is where the coding happens and gets stored. You should only need four fields on this table: ID[5], TOKENNUMBER, ATTRIBUTE and VALUE. The ID Field is the Primary Key and should be set to 'Integer' and 'Auto Value' (every single code you create will be given a unique ID number automatically as you enter it into the database). The TOKENNUMBER Field must be formatted exactly as the same Field in the TOKEN table for the relation to hold between them. The ATTRIBUTE and VALUE fields are where you will enter your codes – attributes such as SYN (syntactic position) and values such as the position DO, IO, OBL etc. I set the VALUE to a length of 100 to be safe. You can change this later if necessary. These attributes and values become your factors and levels should you choose to export your data for use in *R* or other statistical programs.

4.3 Relating the tables (linking)

The next step is to tell the database to link the TOKEN.TOKENNUMBER Field to the ATTRIBUTES.TOKENNUMBER Field.[6] OpenOffice places this function under 'Tools>Relationships'. The relationship tells the database that whenever you select a TOKEN.TOKENNUMBER, you want it to display information from the ATTRIBUTES table where the TOKENNUMBERs match.[7]

---

5 OpenOffice requires a Primary Key on this table since you relate this table to the DATA table even though the relation is really held by the TOKEN table's TOKENNUMBER – this just seems a quirk of the program.

6 PRO TIP: This is the standard notation in SQL for TABLE.FIELD and is used often in SQL commands.

7 To my knowledge it is not possible to relate TOKEN.SITELINE to DATA.SITELINE. I thought perhaps when creating the Form, this might be a useful relationship so that when selecting a new TOKEN.TOKENNUMBER it would jump to the SITELINE in the DATA table, but I could not make that happen. Certain databases may offer this as a **Foreign Key** option, at least as my reading on the web suggests it would be called.

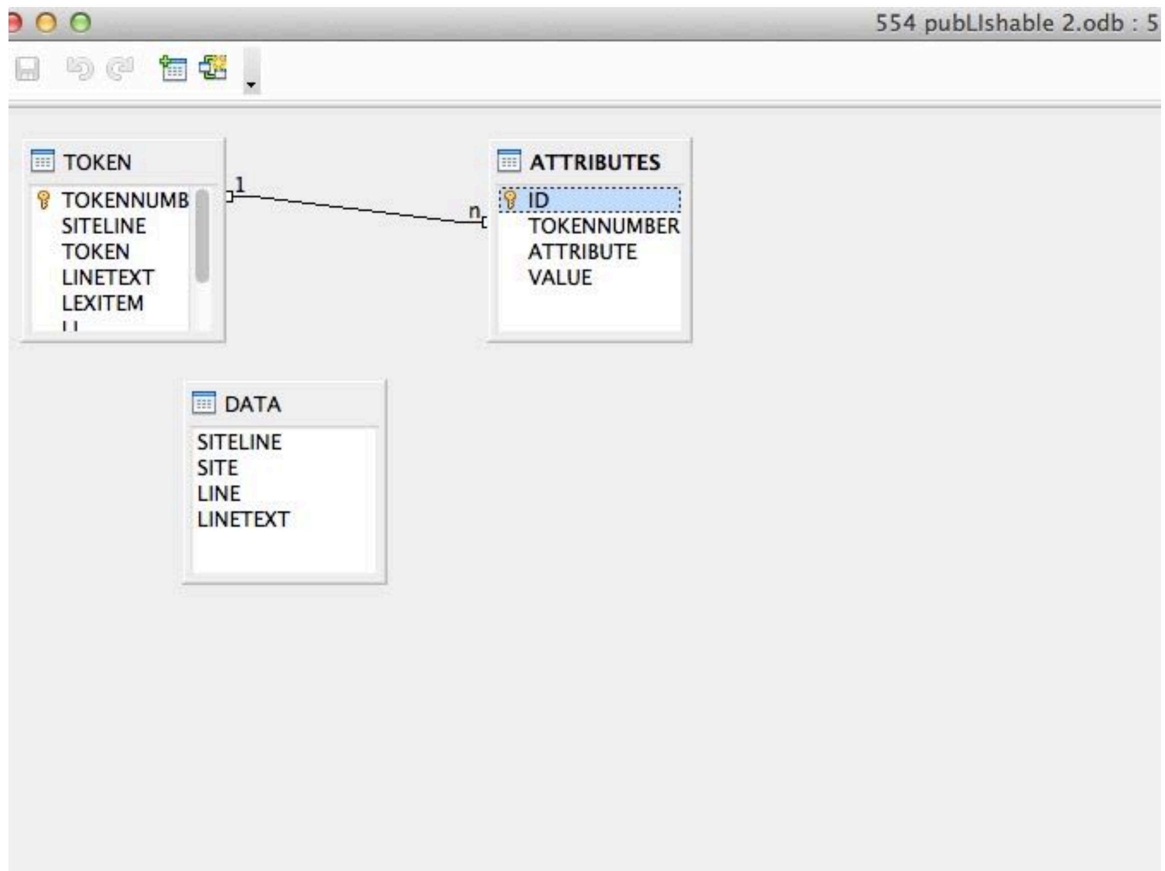

Figure 1. Table relationship output in OpenOffice.

Figure 1 shows the schematic view of the table relationship output by OpenOffice. The three tables are represented in the boxes: TOKEN, ATTRIBUTES and DATA with their corresponding Fields. The key icon shows which Field is the Primary Key. The line linking TOKEN.TOKENNUMBER to ATTRIBUTES.TOKENNUMBER represents the relationship where each single TOKEN.TOKENNUMBER (1) is related to any number of possible ATTRIBUTES.TOKENNUMBER (n) indicating you can code multiple ATTRIBUTEs and VALUEs (see Figure 3 below).

4.4 Populating the database

To populate the database, copy and paste the data from your source spreadsheets to the appropriate database Tables. Make sure your source spreadsheet columns match the layout of your database Table Fields. For OpenOffice, highlight and copy the cells on the source spreadsheet, then right click on the appropriate table in the 'Tables' menu and select paste – do not open the table from the database menu. When you copy and paste the data, a dialog will open up offering some options. Unselect 'Use first line as column names' as you've already defined this in the Table creation. The 'Next' dialog will allow you to ensure the data line up properly with your database Table Fields. You may have to insert blank columns in your source spreadsheet to get things to line up properly. Adding data like this is also called **Appending**. See Figure

2 below for a demonstration of lining up source spreadsheet columns to database table Fields (described in detail in §4.4.1).

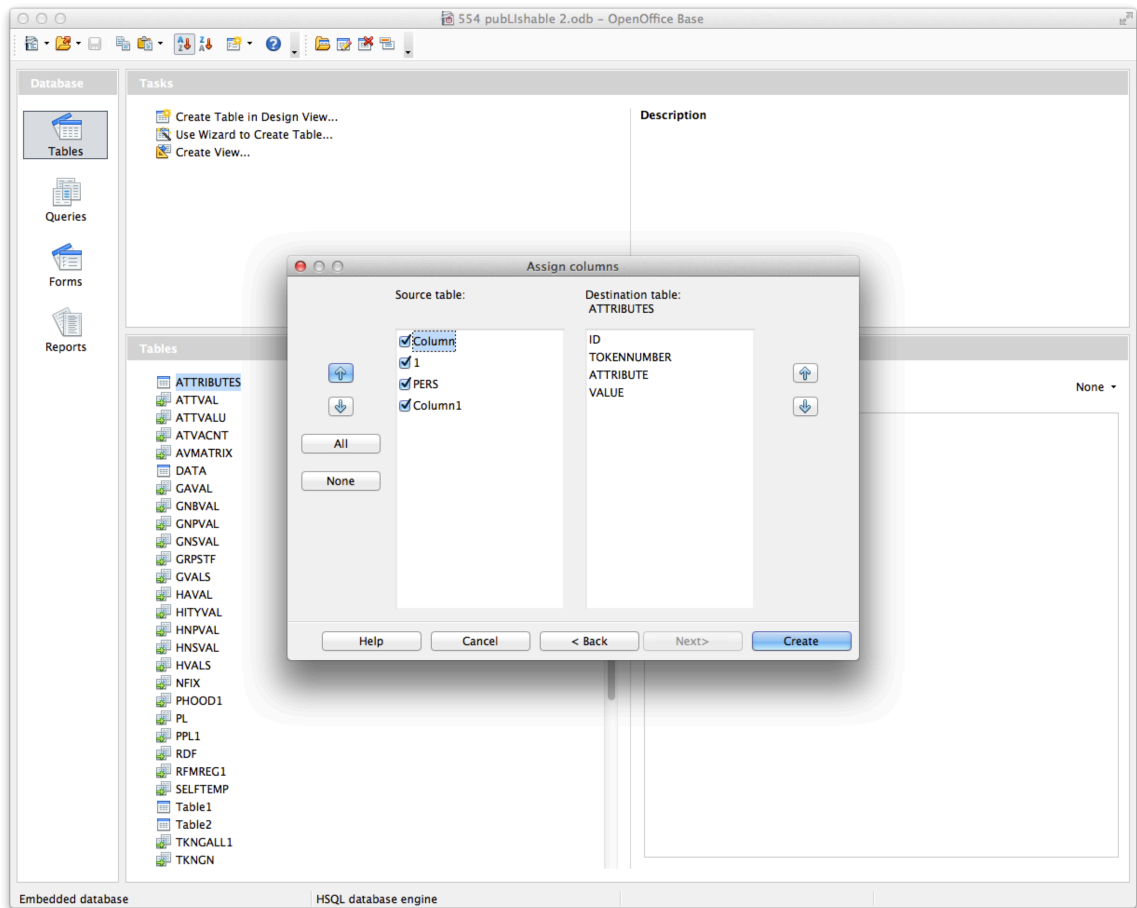

Figure 2. Lining up source spreadsheet columns.

4.4.1 Appending

Appending data simply adds new rows of data to the table and is a useful feature. Figure 2 shows an instance of Appending which functions exactly as in the manner as adding all of your data to the database. The Table menu is in the background window and the dialog which appears after hitting ‘Next’ is in the foreground window. Note how the data from the “Source table” has an empty column represented by Column. The 1 and TOKENNUMBER Fields being aligned shows you that your data line up from the source spreadsheet into the database Table. When hitting ‘Create’ here, this data is added to the bottom of the ATTRIBUTES table and due to the auto-valueing feature of the ID column, a unique number is added to each entry. The final Field (Column 1) remains empty and awaits a new code.

As you go along coding you might find an ATTRIBUTE you want to code for all your tokens. All you have to do is append that ATTRIBUTE with a series of TOKENNUMBERS you haven't coded for yet into the ATTRIBUTES table and now you'll remember to code that ATTRIBUTE everywhere since it will pop up in every

TOKEN. I found I needed to add an ATTRIBUTE called PERS (person) to all my TOKENNUMBERS. In a blank spreadsheet I made an empty column (for the ID), a column of numbers 1 – 518 (TOKENNUMBERS) and a column of PERSs (ATTRIBUTE) next to that. I copied the columns with 518 rows and Appended them to the ATTRIBUTES table. The auto value feature of the ID column automatically assigned each new entry an ID number. Every time I went to a token, I had a VALUE Field ready to fill with an ATTRIBUTE PERS (see Figure 3 below).

4.5 Creating forms

Now that you have your three Tables (two with data, and one waiting for codes) and your relationship(s) that hold between them, you can create a Form (or two) to help with coding. A Form lets you pick what TABLES and FIELDS you want to view and edit in a single window. My preference is to use the OpenOffice 'Form Wizard' to create Forms; the 'Design View' seemed a bit of a hassle.

I wanted my Form to give me access to all three Tables. I wanted to see the raw data from the DATA table (the lower left window); I needed the TOKEN Table to pick which token to code for (the upper left window); the ATTRIBUTES Table had to be available to enter the codes (the right window). However, as only a single relationship holds in my database (due to OpenOffice's quirk mentioned above), I had to hack a workaround involving the creation of two Forms. The first Form to create is one for the DATA table. Follow the prompts to select the DATA table and the Fields you want. For me they were SITELINE and LINETEXT – I wanted to know where in the data I was, and to be able to read the surrounding context. I used the default 'Data Sheet' option under the 'Arrange controls' menu – this presents the data in a scrolling block. Feel free to try out the other options though as they may suit your data better. Follow the remaining prompts to finish the Form.

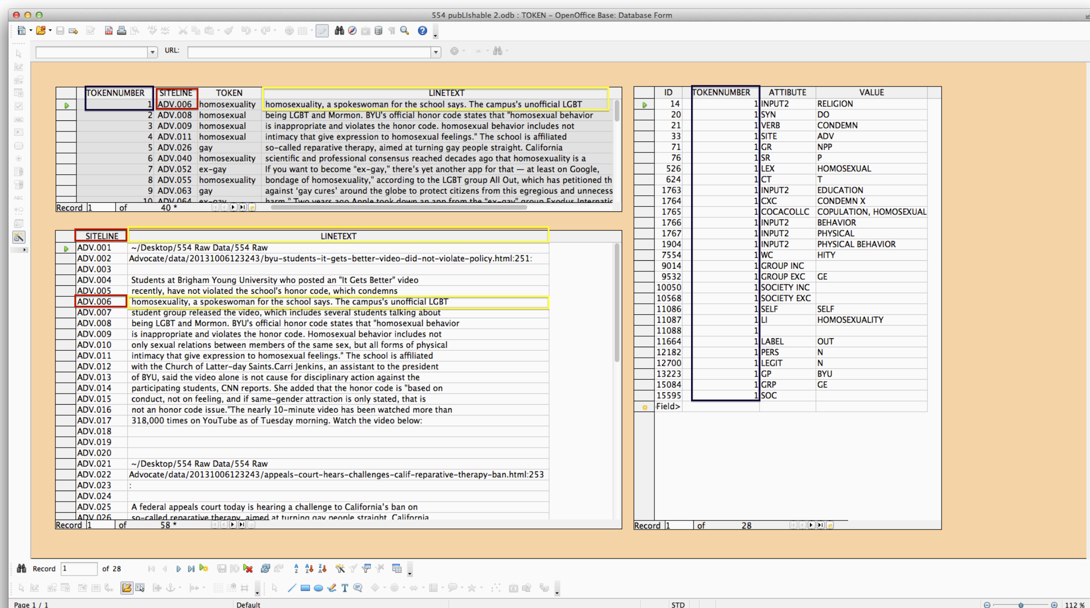

Figure 3. Creating Forms.

The Form you need for your coding is this next one. Create a Form for the TOKEN table as the main form. Select the appropriate Fields you want to have access to. For me these were TOKENNUMBER, SITELINE, TOKEN and LINETEXT. The 'Next' menu offers you the option to add a Subform. Choose the ATTRIBUTES table and select the appropriate Fields. For me they were ID, TOKENNUMBER, ATTRIBUTE and VALUE. Again I selected the 'Data sheet' option under 'Arrange controls'. Follow the prompts to finish the table.

The workaround: right click and edit both Forms. Copy the block of DATA from the first Form and paste it into the second Form. Arrange them as you see fit – I would recommend making the ATTRIBUTES and DATA blocks the largest so you can see your context and all of the codes you enter per token. Save it once you're happy. Do a standard double click on the second Form to open it. If all has worked well, you'll have three blocks in a single window to work with as in Figure 3. The DATA and TOKEN blocks should be showing your data and you should be able to scroll up and down. The ATTRIBUTES block should have an empty row ready for data entry [indicated by the * and Field> cells at the bottom of the right window] – note that the ID Field is ready to auto-value and the TOKENNUMBER Field will match the TOKENNUMBER Field from the TOKEN block once you click into that window. Select a new TOKENNUMBER from the TOKEN block and the ATTRIBUTES block should change to that number.

I should note that there are also ways to create Form Fields that offer drop down menus or check boxes for point and click options. Those will each usually require their own Field in one of your tables though. Again, depending on your data, you may find those options useful.

4.6 Coding

You are now ready to code. As you enter data into your ATTRIBUTES block, the ATTRIBUTES table is automatically updated and saved. Referring back to Figure 3, note how as you select TOKENNUMBER.1 in the top left window, you will automatically switch to that TOKENNUMBER in the ATTRIBUTES window (on the right) [highlighted in black]. As you can see I have added multiple codes to this particular token. As I stated before, each ID is unique – these numbers indicate how over time I had: changed or deleted codes (hence ID 14 as the very first code in my database – the others were deleted at some point); skipped around during coding; or Appended things (note ID 12182 which we Appended above).

Some general tips to consider at this point:

- Use either ALL CAPS or not, avoid spaces and make sure to spell things exactly the same – consistency is the key
- Code as much as you think you'll need, even excessively so – touch the data as few times as possible
- However, be willing to go back and change codes frequently – as you go along, your coding will change (usually for the better)
- Try to keep your ATTIRBUTES and VALUES unique: early on I was coding

a token's semantic role (SR) Agent with a value of A and its grammatical role (GR) if the subject of a transitive clause also with a value of A. Depending on how you have things counted later, you can wind up double counting those A's. So, I had to change all those GR.As to GR.NPAs (see §5.5 UPDATE).

Before moving on to the next section, code a bit of your data so that you'll have something to query.

5 Using SQL

The most time-intensive aspect of learning to use a relational database seems to be getting the hang of coding in SQL, the programming language used to SELECT and COUNT things. There are plenty of websites that can help you get the hang of SQL (see §7 Additional resources). I have spent numerous hours learning how to code different search types; make sure to factor in some time in your work flow for this as well – while I offer some handy searches below, the nature of your project might necessitate different ones.

Note that there are different types of SQL which offer different features. Some sites make note of which database engine the code works for. OpenOffice runs on an HSQL database engine – make sure to use the SQL type as a tag when searching the internet. Even then, don't be surprised if you enter a code from a website that does not work. Two main reasons for this that I have noticed are unsupported features (OpenOffice can't handle PIVOT tables for example) and a slightly different syntax which might require reformulation.

There are two areas in OpenOffice for using SQL. The 'Queries' menu offers options for creating local or search queries – these allow you to view and COUNT your data. The other is under 'Tools>SQL' and allows you to run SQL commands that can make changes to your entire database. USE THIS AREA WITH CAUTION or you can obliterate hours of work (see §5.5 UPDATE).

5.1 SQL syntax primer

The general form of a SQL command is:

> SELECT some FIELDs and do something with them,
> FROM a TABLE
> WHERE certain criteria are met (this is optional)

TABLEs and FIELDs are almost always enclosed in " " while data in those Fields (your codes) are enclosed in ' '. Note that these are not the “smart” double and single quotation marks – if you copy the commands from a document that uses smart quotes, you'll have to replace them with the natural ones.

Only one TABLE can be in a FROM or WHERE clause. To use two tables you have to do some kind of **JOIN** as in 6 where you map the values in "TABLE1.FIELD" onto the values in "TABLE2.FIELD" or to do a **Subquery** as in 6 and 7 (see §5.3).

5.2 Basic SQL queries

In the 'Queries' menu you have several options. I prefer to 'Create Query in SQL View' which opens up a blank query window. There is a wizard to help create queries, but I never got the knack of that. Besides, there are some things you may not be able to do with the wizard that you can do by inputting the SQL manually. And not all programs might have a wizard that works quite the same way. Spending the time learning SQL will definitely pay off in the long run.

Enter the following code and hit the run button. For OpenOffice, there is a 'Run SQL Command Directly' option (the SQL with a checkmark above it) which you may need to select. When unselected you are running the query through OpenOffice's front end – this allows you to edit the results directly in the query window – which can be useful for quick edits. However, some advanced commands will not function properly in this manner and you'll have to run the SQL command directly – avoiding OpenOffice's front end.

(1) What attributes have I coded so far?

```
SELECT DISTINCT ("ATTRIBUTE")
FROM "ATTRIBUTES"
```

DISTINCT searches out the unique codes in ATTRIBUTE rather than all the codes – the difference being I have 134 distinct ATTRIBUTE codes (e.g. LEX from Figure 3 – ID 526) which have been used more than 14000 times (note the four instances of INPUT2 on Figure 3).

(2) What VALUEs have I coded in a given ATTRIBUTE?

```
SELECT DISTINCT ("VALUE")
FROM "ATTRIBUTES"
WHERE "ATTRIBUTE" = 'X'
```

This returns the distinct VALUEs in the 'X' ATTRIBUTE from the ATTRIBUTES table. Note the use of ' ' for the ATTRIBUTE code being sought. For the SYN attribute I have 19 codes such as DO, APP (appositive), PSD (possessed predicate) etc.

(3) How many of those VALUEs do I have?[8]

SELECT DISTINCT "VALUE",	selects the DISTINCT codes in VALUE [note the comma here]
COUNT("VALUE") AS "COUNTOF"	**AS** labels the next column "COUNTOF" which COUNTs each instance of a "VALUE" [note the lack of comma here]
FROM "ATTRIBUTES"	from the "ATTRIBUTES" table
WHERE "ATTRIBUTE" = 'X'	but only those where 'X' is the ATTRIBUTE

8 PRO TIP: The comma is easy to overlook in SQL. It usually occurs between what become columns in the query result, but not after the final column before the FROM clause, hence its absence in 1 and 2.

GROUP BY "VALUE"	**GROUP BY** is usually required when counting things, it matches the pieces together
ORDER BY "VALUE" DESC	please display data by the "VALUE" column in **DESC**ending **ORDER** (optional)

These results come up as a table and tell me I have DO 118, APP 6, PSD 1 etc.

(4) What are the COUNTs of all my ATTRIBUTE and VALUE combinations?

```
SELECT "ATTRIBUTE", "VALUE", COUNT(*) AS "COUNTOF"
FROM "ATTRIBUTES"
GROUP BY "ATTRIBUTE", "VALUE"
ORDER BY "COUNTOF" DESC
```

This selects the unique combinations of ATTRIBUTE and VALUE and counts them. The **(*)** says to count the total instances of each ATTRIBUTE.VALUE combination that has been SELECTed and GROUPed together.

Sometimes you just need to know what TOKENNUMBERs meet certain criteria, perhaps so you can double check how they were coded overall by looking back at each of them on your Form.

(5) Which tokens are coded as X?

```
SELECT "TOKENNUMBER"
FROM "TOKEN"
WHERE "LI" = 'gay' OR "LI" = 'gays'
```

Note how the WHERE conditions are coupled with **OR** since they are both pointing to VALUEs in the same Field ("LI"). You can use **AND** when the values are from different columns as in the WHERE clauses of the Subqueries in 7. Note that this query looks at a Field FROM the TOKEN table, whereas most others have looked at ATTRIBUTES.

5.3 Saving queries, creating tables of query results (Views) and more complex SQL queries

You can save queries so you don't have to keep writing them over and over again. It's a good idea to save the complex ones. Some of the quicker ones may be best to enter over and over again; this helps you keep up on how SQL works, and unclutters the query menu so you're not having to hunt all the time for the right one. Make sure to give the ones you save informative names so you're not always editing or running them to see what they do. Note that each Query, Table and View has to have a unique name.

You may have noticed that the above queries all refer to one of the Tables created at the outset. Sometimes though, you may have to refer to the results from a query you've run in another query. To do this you turn the first query you need to look at into a Table (or in OpenOffice terms, right click the saved query in the Query menu and 'Create **View**'). I created two queries, one for gay lexemes (called PGAY) and one for homosexual lexemes (PHOMO) which counted their unique combinations of

ATTRIBUTE and VALUE (and which also returns a percentage of lexemes that are coded as such). I saved each query, then created them as Views.

(6) JOIN used and a percent calculation, created as View PGAY

SELECT "ATTRIBUTE","VALUE", COUNT (*) AS "COUNT",	[note the commas]
(COUNT("VALUE")*100.0 / (SELECT COUNT ("TOKENNUMBER") FROM "TOKEN" WHERE "LEXITEM"='GAY')) AS "GAY"	This calculates the % column using a Subquery in ( ) as the denominator [note the lack of a comma]
FROM "ATTRIBUTES"	
JOIN "TOKEN" ON "TOKEN"."TOKENNUMBER" = "ATTRIBUTES"."TOKENNUMBER" WHERE "LEXITEM" ='GAY'	This **JOIN** restricts the ATTRIBUTES.TOKENNUMBERs counted to those matching TOKEN.TOKENNUMBERs which meet the "LEXITEM"='GAY' criteria
GROUP BY "ATTRIBUTE","VALUE" ORDER BY "COUNT" DESC, "ATTRIBUTE" ASC, "VALUE" ASC	GROUPing and ORDERing by multiple features are set off by commas

These two new Views can then be queried just like Tables and in 7 are used in Subqueries – the embedded SELECTs set off in ( ) – before the final FROM clause. The way this next query is written returns the counts of any possible combination of ATTRIBUTE and VALUE. As there are instances where combinations are not shared between the two lexemes, we have to select the total possible combinations from ATTRIBUTES (COUNTing them for good measure) then match the COUNT columns from the PGAY and PHOMO Views.

(7) Subquery

```
SELECT DISTINCT "ATTRIBUTE", "VALUE", COUNT(*) AS "COUNT",
(SELECT "COUNT" FROM "PGAY" WHERE "PGAY"."ATTRIBUTE" =
"ATTRIBUTES"."ATTRIBUTE" AND "PGAY"."VALUE" = "ATTRIBUTES"."VALUE") AS
"PGAY",
(SELECT "COUNT" FROM "PHOMO" WHERE "PHOMO"."ATTRIBUTE" =
"ATTRIBUTES"."ATTRIBUTE" AND "PHOMO"."VALUE" = "ATTRIBUTES"."VALUE") AS
"PHOMO"
FROM "ATTRIBUTES"
GROUP BY "ATTRIBUTE", "VALUE"
ORDER BY "COUNT" DESC, "VALUE" DESC
```

The SELECTs in parentheses are the Subqueries and their WHERE clauses tell how to match them up with the main Query. These WHERE clauses in the Subqueries are a kind of quick and dirty JOIN. JOIN is more powerful with other useful functions (thus not discussable at present) and is worth looking into for its applicability to your project.

5.4 Queries for the whole data set exportable to statistical software

The next two queries are ways to look at all of your data in one table. This is useful

when exporting your data to statistical software. Both of these queries will take a bit of computational time to run since they're looking at your entire database multiple times. Seriously, we're talking 3 – 5 minutes or more depending on how big the database is and how many Fields Subqueries you are running. This first query returns the actual VALUEs of each ATTRIBUTE by TOKENNUMBER.

(8) Data by ATTRIBUTE.VALUE

```
SELECT DISTINCT "TOKENNUMBER",
(SELECT "VALUE" FROM "ATTRIBUTES" WHERE "ATTRIBUTES"."ATTRIBUTE" = 'LEX'
AND "ATTRIBUTES"."TOKENNUMBER" = "TOKEN"."TOKENNUMBER") AS "LEX",
(SELECT "VALUE" FROM "ATTRIBUTES" WHERE "ATTRIBUTES"."ATTRIBUTE" = 'SITE'
AND "ATTRIBUTES"."TOKENNUMBER" = "TOKEN"."TOKENNUMBER") AS "SITE"
...
FROM "TOKEN"
ORDER BY "TOKENNUMBER" ASC
```

This calls up all the DISTINCT TOKEN.TOKENNUMBERS then creates a column e.g. LEX and copies the VALUE from ATTRIBUTES WHERE the criteria is met and the TOKENNUMBERs match. You'll need to add Subqueries for all the ATTRIBUTES you want to include. I had roughly 40 of these Subqueries stacked inside the main Query.

If your data need to be in the form of 1s and 0s, the query needs to be a **SUM CASE WHEN**. This requires breaking every VALUE into its own column then offers up a 1 for "Yes, this TOKENNUMBER is ATTRIBUTE X VALUE Y" or 0 for "No".

(9) Data as 1s or 0s by VALUE

```
SELECT DISTINCT "TOKENNUMBER",
SUM (CASE WHEN "ATTRIBUTE" ='LEX' AND "VALUE" ='GAY' THEN 1 ELSE 0 END) AS
"GAY",
SUM (CASE WHEN "ATTRIBUTE" ='LEX' AND "VALUE" ='HOMOSEXUAL' THEN 1 ELSE 0
END) AS "HOMOSEXUAL"
...
FROM "ATTRIBUTES"
GROUP BY "TOKENNUMBER"
ORDER BY "TOKENNUMBER" ASC
```

This calls up every DISTINCT ATTRIBUTES.TOKENNUMBER, then creates a column GAY and enters a 1 in the CASE WHEN the criteria are met ELSE it puts a 0. The SUM is required, summing the returns – if SUM is missing you get only 0s. This might seem odd, but it can help you catch double coding by returning a total greater than 1. For example, I accidentally coded some tokens twice for SYNtactic position and found this out when I saw some 2s floating around. This happened in two ways - conjoined clauses needed both codes, and truly accidental double coding. For the conjoined clauses I tried to remember to use SYN and SYN1 to distinguish the two, but sometimes forgot the '1'. Accidental double coding is just that - when you code the same TOKENNUMBER with the same thing. In cases of double coding,

the Subqueries described in 8 will give you an error message that something is wrong in the whole query but not which specific Subquery is at fault or the exact TOKENNUMBER. This query in 9, however, does let you know what is wrong so you can fix it.

Once you have these queries the way you like them, create them as Views, then right click on the View (in the 'Table' menu) to copy and paste them out into a blank spreadsheet (sadly yes, it took a while to figure something this easy out). Note that if you need both types of data you'll have to run these as separate queries and create them as separate Views since they look at different Tables.

5.5 UPDATE

This last command is how you can make a broad change to an entire table. USE THIS WITH CAUTION or else you can obliterate hours of work by overwriting data. I usually make a copy of my database first, then run the command on one copy to make sure it does what I want it to do – if it doesn't, I still have the back up. This is not a query, so at least for OpenOffice you have to run this command from a different area in the program (specifically 'Tools>SQL').

As an example, say you've coded some VALUE under a given ATTRIBUTE but you want to divorce it into its own ATTRIBUTE. I used a generic INPUT2 attribute for a lot of things but then needed to separate some of them out to count them in Subqueries in 8 above. Because each token had multiple INPUT2s I couldn't just use that as a Subquery or I would get errors. So I used UPDATE to change various INPUT2s to other labels with:

(10) Rename an ATTRIBUTE for the whole data set

```
UPDATE "ATTRIBUTES" SET "ATTRIBUTE"='ACTIVISM' WHERE "ATTRIBUTE"='INPUT2'
AND "VALUE"='ACTIVISM'
```

This finds every instance in ATTRIBUTES WHERE an INPUT2 ATTRIBUTE with an ACTIVISM VALUE and **UPDATE**s it by **SET**ting that ATTRIBUTE to ACTIVISM instead.

You can also UPDATE to a completely different ATTRIBUTE. I had started coding a series of features that I thought were separate, but really should have been under a single ATTRIBUTE for the purposes of statistical analysis. I ran the following two commands to collapse them together under a new ATTRIBUTE "GRP". Note that I had Appended this new GRP ATTRIBUTE to the ATTRIBUTES table first (see §4.4.1 Appending).

(11) Collapse two ATTRIBUTEs into one

```
UPDATE "ATTRIBUTES" SET "VALUE"='I' WHERE "ATTRIBUTE" = 'GRP' AND
"ATTRIBUTE"='GROUP INC' AND "VALUE"='I'
UPDATE "ATTRIBUTES" SET "VALUE"='E' WHERE "ATTRIBUTE" = 'GRP' AND
"ATTRIBUTE"='GROUP EXC' AND "VALUE"='E'
```

This finds every GRP in ATTRIBUTES where there exists another ATTRIBUTE GROUP INC that has a VALUE I and UPDATEs it by SETting the VALUE to I for

that GRP. Then it does the same with a VALUE E where the second set of conditions held.

Lastly, you can update a column on one table with data from a column on another table. This is different from Appending where data is added at the bottom of a table as new rows. Here, in essence, you are adding some data to all the rows that currently exist. As I was coding I realized I wanted to have access to the grammatical category (GC) of a token on the TOKEN table. First I queried up all the TOKENNUMBERs and their GCs from ATTRIBUTES and created a View called GC3.[9] Then I edited the DATA Table to have a GC Field (right click the Table and edit, SQL not required). Then I ran the following:

(12) Add information from one column to a column on another table

```
UPDATE "TOKEN"
SET "GC" = (SELECT "GC" FROM "GC3" WHERE "TOKENNUMBER" =
"TOKEN"."TOKENNUMBER")
```

This SETs the GC column on TOKEN with the value in the GC column of GC3 WHERE the TOKENNUMBER condition holds.

6 Conclusion

I hope the above introduction to relational databases proves useful in your coding practices. I know that prior to using a relational database, I often forgot to take a few moments to think of the data as computational data and how I would be manipulating or counting it. I would rush headlong into the coding and fix any shortcomings later on. I thought that this is how it was done. Having been through several growing pains in learning this tool, I now have a better appreciation of linguistic data as computational data.

I would also remind you that different tools have different strengths and weaknesses. Just because you now have a shiny new tool doesn't mean you can put away all of the others. Think about how you need to manipulate your data and whether you have a tool that seems more apt to the task. Text editors, spreadsheets, regular expressions, Python and relational databases all have their uses in analysis. Learn how to use each one for the most benefit – and pay it forward by writing how-tos for the rest of us.

7 Additional resources

Software:

OpenOffice (http://www.openoffice.org/)

Free office suite with relational database included.

LibreOffice (http://www.libreoffice.org/)

Free office suite with relational database included. This is a second programming fork of OpenOffice which is updated as its own program. It was started when the

9 SELECT DISTINCT "TOKENNUMBER", "VALUE" FROM "ATTRIBUTES" WHERE "ATTRIBUTE"='GC' ORDER BY "TOKENNUMBER" ASC

OpenOffice suite changed hands of developers. I have only recently started looking into it, and can make no comments as to how it functions in relation to OpenOffice's database.

TextWrangler (http://www.barebones.com/products/textwrangler/)

This is a very useful text editor available for Mac OS only. It offers support for regular expressions. It also has many useful features such as line numbering, hard wrapping, adding and changing line break types and more.

SQL sites:

StackExchange (http://stackexchange.com/)

A good website for searching out code snippets that have been posted to other peoples' questions. The code doesn't always work outright, or have you to think about how the author is presenting an answer in linguistic rather than business or accounting terms, but I have figured out many things from my searches here.

SQL Tutorial (http://www.w3schools.com/sql/defalt.asp)

Good basic introductions to various searches and terminology.

Other:

Regular Expressions (RegEx) (http://www.regular-expressions.info/)

Regular expressions are a form of advanced searching. I used this to find the beginning sequence of an HTML string and an ending sequence (often with 100s of lines in between) in order to extract only the useful portions of my data (the article text) and get rid of all the unneeded HTML code sections.