Hi everybody,

George wanted us to come up with a list of concrete research questions in order to test the feasibility of the query language. Upon discussing potential questions, it became evident for us that the language would benefit very strongly if at least some basic statistical procedures would be implemented.
For descriptive statistics, we would need easy ways to produce output for
- Frequencies
- Arithmetic means
- Standard deviations
- Pearson correlations

For inferential statistics, the following tests would be useful (in descending order of usefulness)
- t-Tests
- Analysis of variance
- multiple regression
- mediator analysis

Alternatively, outputs that can be read by SPSS, would be very welcome.

That being said, we tried to come up with a number of questions that are a) interesting from a psychological point of view; b) differ in complexity; c) require both descriptive and inferential statistics; and d) touch on all three main data sets.

1. Questions comparing threads
Background of the first pair of queries is the question of whether readers will like reading better threads.

1.1 Simple relation between interactivity and rating
For instance, when readers rate a thread positively, they could do it because of the topic (it's a fascinating or important topic), but they could also do it because of the quality of the discussion itself.
A potential marker of a good discussion quality is the interactivity. Thus, the first step of the query should compute an interactivity variable for each thread, defined as the percentage of posts in the thread that contain quotes.

To get this, one would have to check for each post [spiegelPostUrl] in a given thread [spiegelPostThreadUrl] whether it contains a quote [spiegelQuotePostUrl] or not (coded binary), sum up the binary values over the entire thread, and divide it by the number of posts in the thread [spiegelThreadReplies]. The interactivity variable would be named [spiegelThreadInteractivity] and saved as new variable in the thread database.

For the analysis of the first research question we would further have to exclude all threads that did not receive a rating [spiegelThreadRating].

Finally, we would need a Pearson correlation coefficient (plus significance). This would allow us to see whether interactivity [spiegelThreadInteractivity] is positively correlated with the average rating of the thread [spiegelThreadRating].

So far our example for a comparatively "simple" query.

1.2 Complex relation between interactivity, author role, and rating

We also came up with a more complex query, based on the same idea as above. For instance, is the hypothesized relation between interactivity and rating in some way dependent on the contributors? The concrete assumption would be that interactivity boosts ratings even more if the interactivity is related to (or caused by) supposedly experienced users.
Registering the experience of a user would need a re-coding of the variable [spiegelUserRole] into the variable [spiegelUserExp1] with “Erfahrener Benutzer” coded as 1, “Administrator” coded as -1, and all other entries (“Nutzer”, “Neuer Nutzer”, no entry) coded as 0.

In the next step, the posts of each thread are taken into account. If a post contains a quote [spiegelQuoteUrl?], a running tally *i* should be incremented by 1. If the post that’s quoted [spiegelPostUrl] was created by an author [spiegelPostAuthorUserUrl] that is categorized as “Erfahrener Benutzer” [spiegelUserRole], another tally *j* is incremented by 1. In order to exclude replies to the sysop, variable *i* is decremented if the author role is “Administrator”. The number we are finally looking for will be the ratio of variables *i* and *j* for each thread. It indicates how many of the replies in a thread are replies to an experienced user. This rate, termed [spiegelThreadInteractivityExp1] would be statistically independent from the overall rate of replies [spiegelThreadInteractivity].

In an ideal case, we could then do a multiple regression with the overall interactivity [spiegelThreadInteractivity] and the specific interactivity with regard to experienced users [spiegelThreadInteractivityExp1] as predictors, and the rating of the thread [spiegelThreadRating] as criterion.

2. Questions regarding posts

Here we came up with two different queries.

2.1 Determining “time until quoted”

The first query is to find out the average time that passes between a post and an eventual quote of the post, as we believe that there is a temporal window where the likelihood of a post being noted (quoted) is relatively high. In order to arrive at such a parameter, all threads must be opened sequentially. If a post contains a quote [spiegelQuotePostUrl], the difference between the time of the quote [spiegelPostTime] and the time of the original message [spiegelPostTime] will be captured and stored along with the original message [spiegelQuoteUrl]. An exception to this rule would be if the original message [spiegelQuoteUrl] was from the sysop [spiegelUserName], as their initial thread openers are functionally different from a typical user post. Therefore, quotes of the thread-opening post should be ignored.

By the way, how does the structure of the database work when there are multiple quotes for a post? Let’s take an example where we have three quotes and three time differences in total. For instance, post p1 is quoted once, then the time difference dt1 could be stored along with p1. However, if a post p2 is quoted twice, we have two time differences dt1 and dt2. Are dt1 and dt2 treated as different variables? And how do we get an average of all three time differences (one for p1 and two for p2)?

In any case, in order to find an answer to the time window question, we would need the average time differences for all posts. Unquoted posts should neither receive a zero nor infinity for the time difference variable(s), but should have an empty value.

Another question about the database: Let's assume that the query yields a result of 12 hours average "time until quoted". Is it possible, to save this value somewhere (after all, it is not a variable, but a single parameter)? And could later analyses then refer to this parameter, e.g. by querying how many posts were written within the average "time until quoted" of 12 hours?

2.2 Relative post position and its influence on quoting probability

Our second query with regard to posts refers to the display structure of Spiegel posts. We believe that a post that is displayed at the top of a page is more likely to being quoted than a post that appears on the bottom of a page. It should be relatively easy to answer this question, as Spiegel online always displays threads in pages of 10 posts.
In order to answer this question, an initial step would take the Post-URL [spiegelPostUrl] as input.
A typical URL looks like this:
http://forum.spiegel.de/shopost.php?p=5838410&postcount=218
As we can get the relative position of a post on a page from the "postcount" part of each URL, we would need the last byte of the URL string (that would be "8" in the example above), and maybe convert the text string "8" into the number 8. An additional rule would be that the text string "0" should be converted to 10, representing the last post on the bottom of a page. The resulting variable should be named, e.g. [spiegelPostPagePosition].

The second variable for this analysis would be the average number of replies (quotes) for a post. In order to arrive at this variable, we need a method very similar to the one described in query 2.1, i.e. whenever a quote is found, the value of the original post will be changed (in this case, incremented by 1). The default value should not be "empty" as in the 2.1 query, but should be 0. As a result, we know for each post how often it was quoted [spiegelPostQuoteFreq]. In order to get unbiased results, this variable should be empty for the thread-opening post.

Finally, we need the Pearson correlation coefficient between [spiegelPostPagePosition] and [spigelPostQuoteFreq].

3. Questions about Users

As first example queries with regard to users we would like to know if new users are quoted more often than experienced users (or maybe vice versa?)

3.1 Quoting probability based on user category

A first step would require opening all posts and look for quotes (as in queries 2.1 and 2.2). However, the running tallies would not be stored along with the posts, but along with the authors of the posts [spiegelPostAuthorUserUrl]. The tallies would be stored in the new variable [spiegelUserQuoteFreq].

Once this is accomplished, all author entries [spiegelUserUrl] will be opened, and the number of quotes [spiegelUserQuoteFreq] is divided by the number of posts [spiegelUserPosts], arriving at the probability that a user's posts are quoted [spiegelUserQuoteProb]. In case that you have extracted all users of SPIEGEL online (including all lurkers) we would have lots of cases with a division by zero that should be discarded for the analysis.

Finally, different running tallies would be recorded for each user category ("Erfahrener Benutzer", "Neuer Nutzer" etc.).

3.2 Quoting probability based on post count

This query would use the results and variables of query 3.1, but the final analysis would not be based on running tallies for user categories, but would involve the Pearson correlation between the quoting probability [spiegelUserQuoteProb] and the post count [spiegelUserPosts].

(A refined version of this second query would not be based on the actual post count numbers, but on an estimated post count at the time of posting, using the variables for posts per day [spiegelUserPostsPerDay], and the time difference between the post date [spiegelPostTime] and the registration date [spiegelUserRegistered]).