STARS: a System To Aid Reuse in Systematic Reviewing

Alex Pagan

1. Introduction

Systems that interleave computation with human interaction are pervasive. These systems, called *interactive systems*, need human input to function. An interactive system is useless without a “human-in-the-loop.” Therefore, we can think of human interaction as a type of resource that incurs some amount of cost. Indeed, many human-performed tasks have high latency, require monetary compensation, or introduce error (e.g. data inconsistency) that must be later corrected. Thus, we would be remiss to ignore the cost of human labor when considering these systems’ effectiveness.

This thesis explores the expense of human-in-the-loop data management tasks in the context of constructing systematic reviews of clinical trials. A *systematic review* is an assessment of medical evidence relating to a specific research question. The purpose of a systematic review is to reach conclusions about the effectiveness of a treatment. This is done by conducting an exhaustive literature search about that treatment, then analysing the quality and concordance of all empirical evidence returned by that search. While systematic reviews help justify the decisions that medical professionals make on a daily basis, they are very expensive to create. This expense is due to the amount of time and effort required from skilled medical professionals to conduct good systematic reviews.

In response to this problem, we propose a tool called STARS, or a **S**ystem **T**o **A**id **R**euse in **S**ystematic Reviewing. The design of STARS was guided by opportunities for *data reuse* found in the systematic review process. STARS aims to reduce the aggregate expense of conducting systematic reviews by eliminating duplication of effort wherever possible.

1.1 Data reuse in interactive systems

By *data reuse*, I refer to the practice of trading a data creation task for an equivalent data retrieval task within an interactive system. To make this clear, suppose that a task T is performed by a user, resulting in data. This data is stored by the system. Later, some other user requires the data generated by performing T. Instead of doing T again, the user retrieves data generated earlier. This is data reuse.

In general, data reuse is effective when the resources required for data retrieval are on average less than those required for data creation. A canonical computational example of data reuse is memoization of function calls. Memoization saves the results of a function call, then associates those results in memory with the function's arguments. This way, results of subproblems can be looked up from memory instead of wasting CPU cycles on redundant computation.

By *duplication of effort*, I refer to cases in which a task is independently performed by different parties to the same effect. The reuse potential, or *reusability,* of a data object is a function of its creation cost and the probability that its creation causes duplication of effort. Objects with high creation cost and high probability of duplication of effort tend to be highly reusable.

1.2 Overview

STARS is a data management system. STARS manages bibliographic data about clinical trials, as well as user interactions, user annotations, and domain-specific artifacts. STARS also implements SR-SQL, a set of extensions to the common SQL query language. SR-SQL is designed to facilitate queries about systematic review data, thereby making reuse of those data plausible.

For this thesis, I first present an extensive overview of the aims and methods used in Evidence Based Medicine and Systematic Reviewing. This is required in order to understand the opportunities for reuse exhibited by the system and to make sense of its primary use cases. I then discuss opportunities for reuse in the review process, the design of the STARS data model, and the design and functionality of SR-SQL. Following this, I present an overview of STARS's architecture and implementation. Finally, I provide a review of related work and a discussion of possible future work.

2. Evidence-based medicine and systematic reviews

2.1 Aims of Evidence-based medicine

A fundamental principle of health care is that the treatments that clinicians perform are safe and effective. This principle is typified by evidence-based medicine (EBM), which encourages healthcare professionals and institutions to make use of the growing body of empirical evidence (e.g. randomized-controlled clinical trials) to support decisions about individual patients.

While EBM is acknowledged as a powerful means for improving healthcare, it requires that clinicians to stay current with all medical research in their domain. This is clearly infeasible - while it has been estimated that a medical professional needs to read about twenty articles per day to follow their field [1], many only have a few hours per week to devote to literature review [7].

2.2 Systematic reviews

One way to make sense of medical evidence is for groups of physicians to write and publish *systematic reviews*. A systematic review is exhaustive assessment of medical evidence relating to a specific research question. The review question generally addresses the effectiveness of a drug or treatment. The personnel who conduct systematic reviews realize that medical evidence is not uniformly good. For example, some clinical trials might lack proper randomization, be subject to bias, or fail to reflect other modern technology or methodology. As a corollary, results of trials can conflict one another. Systematic reviews attempt to resolve conflicting conclusions by analyzing all available evidence in a standardized manner so as to find sources of discrepancies, like bias.

Systematic reviews use methodology established by organizations organizations within the medical community, like the Cochrane Collaboration [6], and are primarily created and consumed by healthcare professionals. Systematic reviews are often created as a part of a group, rather than by a single author. While systematic reviews make medical literature more accessible, they are very expensive to create. Systematic reviews generally take 6-12 months to write [10], and as mentioned above, their authors are generally physicians, whose time is expensive.

2.3 The Systematic Review Process

The systematic review process is composed of several high-level steps. These are not discrete, but they do fall into rough sequential order. First, reviewers formulate a precise research question. Next, they search for citations of relevant clinical trials using bibliographic databases and other means. Following this, they screen collected trials for relevance. Once they have obtained a set of relevant trials, they extract trial data and determine potential sources of bias. The reviewers then analyze data presented by the relevant clinical trials. Finally, reviewers interpret their findings and document their results. After the systematic review is published, authors maintain their review. I will now explain each step.

2.3.1. Formulate a precise, answerable research question

The research question determines the scope of the systematic review. A “well-formed” systematic review question has four parts: *population*, *intervention*, *comparison*, and *outcome*. These parts are called PICO.

- The *population* describes the patient demographic. In practice, this often means a group of persons (sometimes qualified by some attribute, like age) that have a health problem.

For example, a valid description of a population is “older women with common cold symptoms.”

- An *intervention* is an action performed by a healthcare professional to improve health problems found in the population. Following the example given above, a valid intervention is “200mg of echinacea taken once daily.”
- A *comparison* describes an action whose effect is compared with that of the intervention. While population, intervention, and outcome are required for a well-formed systematic review question, comparison is not always necessary. For example, it is possible for only one treatment to exist for a medical problem. An example comparison is “placebo taken once daily.”
- An *outcome* is a measurable effect resulting from the application of an intervention to the population. An example outcome is “relief from congestion.”

A well-formed PICO question based on the foregoing example is: “In older females with common cold symptoms (population), does 200mg of echinacea taken once daily (intervention) provide better congestion relief (outcome) than placebo? (comparison)”

The PICO question, and other details about the design of the systematic review, are recorded and published in a document called the systematic review *protocol.* The protocol is an *a* record of the systematic review question and the anticipated methods for addressing that question. Protocols are written *a priori* (i.e., before examining evidence) to ensure that the design of the systematic review is unbiased.

2.3.2. Search for applicable citations

After forming a systematic review question, reviewers must find all relevant evidence. A study contains relevant evidence if there is a close match between the study’s research question and the review’s PICO question. Finding relevant studies is often done by issuing keyword searches over bibliographic databases, like PubMed [12]. While large databases are common sources of trials, it is considered good practice for reviewers to also search a number of sources, including print journals, foreign language sources, and indexes of unpublished (“gray”) literature [10]. This wider search is conducted to minimize bias that could be introduced by restricting sources of trials.

One way of thinking about the initial literature search is as a binary classification test. Citations are classified as “relevant” or “not relevant” on the basis of inclusion or exclusion in search results. This stage of the process must be conducted carefully, as systematic reviews are supposed to be exhaustive. This is, if a reviewer excludes a relevant study (i.e. the test creates a false negative), the review’s credibility is compromised. An irrelevant trial that is marked “relevant” (i.e. a false positive), costs only a few minutes of reviewer time during citation screening. Therefore the cost of classifying trials is imbalanced: cost of false negatives is much greater than that of false positives [3].

Given this cost imbalance, it follows that searches ought to err on the side of over-inclusion in order to maximize sensitivity, which is the proportion of correctly-classified, relevant results. However, if searches are optimized for sensitivity rather than accuracy, (i.e. the proportion of results that are correctly classified) then only a small proportion of search results are likely to be relevant. For example, the Tufts EBM group reports approximately 10% search accuracy [3].

When developing a search strategy, reviewers aim to find an appropriate balance between accuracy and sensitivity that minimizes the cost of the search. In practice, this means that searches are developed iteratively. Since no efficient method is known for determining sensitivity and accuracy of a search, refinement strategies are highly heuristic. For example, to estimate whether a search is appropriately sensitive, reviewers might pick a small number of prominent studies that they know *a priori* ought to be returned by the search. If those studies are not returned, then it suggests that that search should be be made broader. For estimations of accuracy, reviewers might select a maximal size M for the search results, where M is the largest number of citations that the group can screen in a timely manner. If n is the number of actually relevant trials (n is unknown, but it is generally estimated to be small, e.g. tens of studies) then n/M is the lowest tolerable accuracy; any search that returns more than M trials will take too long to screen and should be made more specific by, e.g., adding keywords. It should be noted that the best known resource for constructing effective search strategies is the expertise of domain specialists.

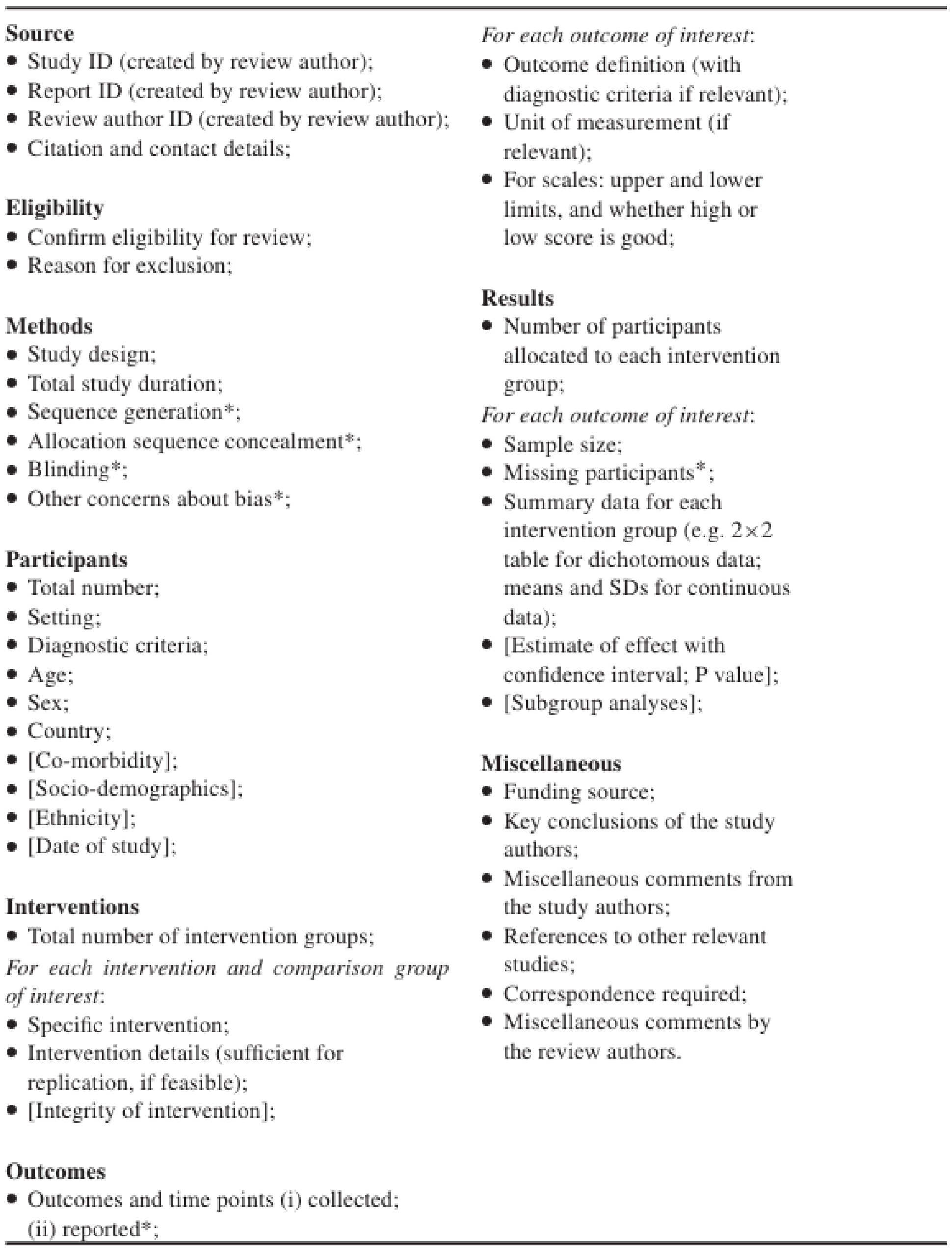
2.3.3. Screen citations for relevance

After collecting possibly relevant citations, reviewers screen citations by hand. This is done by reading abstracts of every trial returned in the previous search step. Since the number of results to be screened is typically large (e.g., thousands of studies) this step is very time-intensive; even though the cost of examining each study is small, the aggregate cost of screening citations is prohibitive. Additionally, the Cochrane Collaboration recommends that studies are screened independently by two or more researchers. While this reduces the risk of bias, it multiplies the cost. [10]

As mentioned before, inclusion criteria for citations are determined largely by the PICO question. For example, a study could be excluded because it addresses a different population or intervention than desired. In addition to the PICO question, study design is a factor in trial screening. For example, it may not make sense to compare an observational study (e.g. longitudinal study) to a controlled study (e.g. randomized-controlled trial) because of differences in intervention allocation. Thus, it may be desirable to limit results to one type of study.

2.3.4. Extract trial data

Trial data is information about a study. This information includes (but is not limited to) details about methods, participants, setting, context, interventions, outcomes, results, publications, and investigators. The Cochrane Handbook recommends that the following data be considered for extraction:



Source
- Study ID (created by review author);
- Report ID (created by review author);
- Review author ID (created by review author);
- Citation and contact details;

Eligibility
- Confirm eligibility for review;
- Reason for exclusion;

Methods
- Study design;
- Total study duration;
- Sequence generation*;
- Allocation sequence concealment*;
- Blinding*;
- Other concerns about bias*;

Participants
- Total number;
- Setting;
- Diagnostic criteria;
- Age;
- Sex;
- Country;
- [Co-morbidity];
- [Socio-demographics];
- [Ethnicity];
- [Date of study];

Interventions
- Total number of intervention groups;

For each intervention and comparison group of interest:
- Specific intervention;
- Intervention details (sufficient for replication, if feasible);
- [Integrity of intervention];

Outcomes
- Outcomes and time points (i) collected; (ii) reported*;

For each outcome of interest:
- Outcome definition (with diagnostic criteria if relevant);
- Unit of measurement (if relevant);
- For scales: upper and lower limits, and whether high or low score is good;

Results
- Number of participants allocated to each intervention group;

For each outcome of interest:
- Sample size;
- Missing participants*;
- Summary data for each intervention group (e.g. 2×2 table for dichotomous data; means and SDs for continuous data);
- [Estimate of effect with confidence interval; P value];
- [Subgroup analyses];

Miscellaneous
- Funding source;
- Key conclusions of the study authors;
- Miscellaneous comments from the study authors;
- References to other relevant studies;
- Correspondence required;
- Miscellaneous comments by the review authors.

Figure 1: Information that the Cochrane Collaboration recommends for inclusion in systematic review. Bracketed attributes are relevant to only some reviews. Starred elements are described in coming sections, which addresses sources of bias. [10]

This information is extracted from full-text trials by reviewers, often using a *data collection form*. A data collection form is a standardized form that used within a systematic review group to collect data for a specific question. These forms are designed to target information needed to answer the systematic review question, and ignore data irrelevant to the question. Data collected through these forms is generally stored using software like Microsoft Excel and Microsoft Access.

As in the citation screening step, the Cochrane Handbook suggests that more than one person should extract data for each trial to minimize bias present in selecting relevant data. Conflicts between extracted data can be later resolved between group members.

2.3.5. Assess studies for bias and quality

After all relevant studies have been selected and their data extracted, it is important for systematic reviewers to determine whether the experiments described by those studies are reliable. This is done through analysis of *bias*, i.e. error in experimental procedure such that the experiment, if repeated, yields incorrect results in the average case. Determining whether a study is biased requires reading each study in detail, extracting information about study design, and noting whether proper experimental design principles were followed. Examples of these principles include:

- Sequence generation

  Randomization is an important factor in good experimental design. This is true especially in the case of randomized-controlled trials, which depend on randomization of tasks like intervention allocation. If (purportedly) random sequence generation is actually nonrandom, an experiment is said to exhibit selection bias.

- Blinding

  Blinding is the practice of concealing information about an experiment's design from study participants or personnel. Blinding is done in order to prevent conscious or unconscious bias caused by knowing sensitive information (e.g. intervention allocation) that may affect outcomes or outcome measurements.

- Complete reporting of outcome data

  A study should report on all outcomes, regardless of their statistical significance. The tendencies to report and publish only statistically significant findings are called reporting and publishing bias, respectively.

Various ways of numerically rating bias exist. One such method is the Jadad rating, which ranks trials on a scale of 1 to 5 using criteria like those presented above.

In cases where experimental data is not published with a study, it may be necessary to request this data from the study authors. Completeness of result data is important for accurate assessment of bias.

2.3.6. Analysis and meta-analysis

In order to make conclusions about the relative *effects* of interventions, reviewers conduct analyses of trial data. In SR terminology, *effect* is the difference in outcome between groups that receive different interventions. In order to make conclusions about intervention effect, reviewers analyze direction of effect (i.e. whether the outcome was improved or not), magnitude of effect (by how much did the outcome improve or degrade), and the consistency of effect across studies. Reviewers also use data about study methodology to determine the strength of the evidence supporting the observed effect.

Analyses can be qualitative or quantitative. Qualitative analysis may include a structured, narrative summary of a study’s characteristics and conclusions. Quantitative analysis often takes the form of statistical meta-analysis, which pools results of trials using statistical methods.

The choice of analytical method depends on the quality and design of included studies. Because meta-analyses combine measures from many trials, it is important to ensure that the combination of those measures is actually meaningful. For example, if outcomes are measured in different ways, then simply pooling outcome measurements by computing a weighted average may not be meaningful. Similarly, quantitative analyses are likely to be biased if the constituent trials are also biased. When conducting statistical meta-analysis, reviewers must be sure that experimental heterogeneity and bias are adequately accounted for in their chosen model.

2.3.7. Interpret findings and present results

The final step of the review process is to summarize the review's methodology and findings and synthesize a conclusion to the review question. An important aspect of the interpretation step is a discussion of the methodological limitations of the trials and reviews discussed. Addressing the limitations of the evidence presented allows the reader to objectively determine how confident she should be in the findings and conclusions of the review.

2.3.8. Maintain studies

After a review is published, authors are responsible for continually revising the review to reflect new evidence, e.g. newly published trials.

2.3.9 Time requirements for creating systematic reviews

Month	**Activity**
1-2	Preparation of protocol.
3-8	Searches for published and unpublished studies.
2-3	Pilot test of study eligibility criteria.
3-8	Inclusion assessments.
3	Pilot test of ‘Risk of bias’ assessment.
3-10	Validity assessments.
3	Pilot test of data collection.
3-10	Data collection.
3-10	Data entry.
5-11	Follow up of missing information .
8-10	Analysis.
1-11	Preparation of review report.
12-	Keeping the review up-to-date.

Figure 2: A general timeline for conducting systematic reviews, provided by the Cochrane collaboration. [10]

2.4 Opportunities for reuse

Systematic reviews are expensive to make and maintain. Most of this expense comes from human effort spent in selecting and analysing clinical trials, and much of this effort produces data as a result. Throughout the authorship process, data is created, collected, and extracted. This data, which I refer to as *systematic review data* (SR data), is stored locally by a review team. There are few widely-used method for distributing systematic review data between review groups.

The design and implementation of STARS is based on the hypothesis that if reviewers were given a system to store and retrieve SR data, then that system could facilitate reuse after accumulating data. A complementary hypothesis is that data reuse in systematic reviewing could reduce of the cost of creating systematic reviews. In order to build a system to test these hypotheses, I ask three questions: First, which SR data are generated? Next, which SR-data are most reusable? Finally, how should SR data be managed? In the rest of this section, I address the first two questions. The last question is the primary topic of the next section (3).

2.4.1 Which SR-data are generated?

I structure my description of SR data using three categories: *novel data*, *collected data*, and *extracted data*. Novel data have user-created content and structure. Collected data are imported from one structured source into another with little change in logical organization. Extracted data are imported from one source to another, but have undergone changes in either logical organization or content, e.g. through truncation.

Examples of novel SR data include PICO questions, search strategies, lists of relevant trials, tables, annotations (perhaps including assessment of bias), statistical meta-analysis, and unstructured prose writing. Indeed, much (if not most) SR data is novel. Examples of collected SR data include possibly relevant trial citations and experimental data obtained from investigators. Extracted data includes trial data.

Step of Systematic Review:	**Data Generated:**
Forming PICO Question	PICO question, systematic review protocol
Literature search	Search strategies, lists of possibly relevant study citations
Citation screen	Lists of *actually* relevant trials (per reviewer),

	annotations explaining rationale for leaving out excluded trials
Data extraction and entry	Data extraction forms, extracted study data, missing study data retrieved by investigators, annotations about study quality
Assessing bias	Annotations about study bias (e.g. Jadad rating), tables indicating risk of bias, narrative assessment evaluating sources of bias
Analysis	Data prepared for statistical software, results of statistical meta-analysis, narrative analysis
Interpretations	Summary of findings table, risk of bias tables, other SR prose

Figure 3: Examples of data generated at each step of a systematic review

2.4.2 Which SR data are good candidates for reuse?

Following the definition of reusable data given earlier, reusable SR-data should be expensive to create and not unique to one systematic review. Many of the SR data mentioned above satisfy both criteria, but some do not.

Given that creating novel data requires user decision-making, virtually all novel data are expensive to create, thus satisfying the first criterion. Some novel data, however, are for all intents and purposes unique to a systematic review. For example, some data are generated by processes that must be performed for every systematic review. An example of such a process is creating PICO questions. There are also novel data that are synthesized from work done earlier in the review, like unstructured prose and statistical meta-analysis. In this case, there is no obvious way to reuse these data in whole or part; reusing any part of the synthesized data implies that all of the base data (i.e. data from which the synthesis is derived) are the same. The probability, however, of *all* base data being the same between two different systematic reviews is low. For this reason, it does not make sense to try to reuse these data.

Collected data are generally good candidates for reuse. For example, lists of potentially relevant trials are collected from many different sources - electronic databases, grey literature registries, print journals, etc. Collating results from these searches takes a non-trivial amount of time, especially when data-entry is required for citations that cannot be automatically exported from search engines. Similarly, experimental data retrieved from reviewers has high reuse-potential; latency for this data is generally high, and the experimental results are potentially valid for any reviewer who chooses to include the trial in question.

Finally, extracted data also make good candidates for reuse. As described above, trial data is extracted by reading the full text of a trial and entering relevant information into a form. This is an expensive task, requiring human judgement and data-entry. While not all trial data extracted from a trial are relevant to every systematic review that includes that trial, there is certainly overlap. This opportunity has been explored by systems like SRDR [18] and ClinicalTrials.gov [5].

2.4.1 Specific opportunities for reuse

I will now present two realistic scenarios that demonstrate the utility of data reuse in systematic reviewing. The first is the case of revising out-of-date systematic reviews. The second is the case of auditing systematic reviews for methodological soundness.

Revising an old systematic review

Suppose that a physician treats many patients with type 2 diabetes. A new drug to treat type 2 diabetes, Exenatide, has been recently approved by the FDA. The physician wants to find and evaluate evidence supporting its effectiveness for patients she treats, who currently use daily insulin to regulate levels of blood glucose. The physicians and her colleagues decide to conduct a systematic review to answer this question. The PICO question (which I call Q) is:

"In adults 18 and older with type 2 diabetes (*population*), how well does exenatide taken at interval *x* with dosage *y* (*intervention*) regulate glucose (*outcome*) as compared to insulin (*comparison*)?"

After forming her question, the review group searches for systematic reviews that answer questions similar to Q; she does not want to create a new systematic review if another review adequately answers her question.

She finds a review that addresses a very similar question, Q'. This review happens to be more than one year old and has not been maintained properly. This means that no new evidence have been included in the systematic review since its publication. I call this systematic review S_{old}.

What choices do the review group have? It can use the old systematic review to justify choices of treatments and thus save the time and effort of building a new review. This approach should be avoided if possible, since S_{old} is not exhaustive. Alternatively, the review group can build a new systematic review from scratch. While this approach better guarantees that the needs of patients are addressed, it is expensive. S_{old} and S_{new}, if performed properly, should follow the same methodology, i.e. that which has been established by the Cochrane Collaboration. Therefore the following hold:

1. The search strategies for S_{old} and S_{new} should be very similar, if not identical.

2. If trial selection is performed correctly in S_{old}, then all of its trials should also be included in S_{new}.

3. The data extracted and collected (e.g. trial data) for those trials should also be very similar.

Since search strategies and selected trials are (or at least ought to be) similar for S_{old}

and S_{new}, these data can be used to help generate search strategies for S_{new}. This observation follows from the fact that S_{old}'s searches are composed of combinations of keywords likely to yield relevant trials. Therefore, S_{old}'s search terms can be used as a starting point for constructing S_{new}'s search strategy. Since S_{new} might be constructed under different assumptions than S_{old} (e.g., S_{new}'s review group has fewer reviewers to screen trials), the strategies borrowed from S_{old} may have to be modified to assure effectiveness. This is not necessarily problematic, since SR-data obtained from S_{old} can be used to tune S_{olds}'s search strategy. For example, the list of relevant trials from S_{old} can be used to heuristically adjust the sensitivity of search strategies as described in Section 2.3.2. This observation follows from the fact that if S_{old}'s trials are actually relevant to S_{new}, then each of S_{old}'s trials should be returned by some search strategy in S_{new}.

Since there is significant overlap between trials in S_{old} and S_{new}, SR-data directly relating to trials can be reused between reviews, assuming that the extraction process is sound. This means that reviewers can forego some of the effort needed to extract trial data. Similarly, if a systematic review requires that some experimental data be retrieved from clinical trial investigators and that data is managed by the system, then reviewers are no longer required to wait for investigators to reply to requests for data. The trial screening process presents another possible opportunity for extracted data reuse. Once a set of candidate trials has been established, it may be useful to issue structured queries over extracted data that are not expressible using a regular keyword search. An example of such a query might be: “Give me all trials with populations greater than one hundred that declare no conflict of interest.”

Auditing systematic reviews

The opportunities described in the previous example depend on the assumption that S_{old} was performed correctly. This is not always the case. Just like clinical trials, systematic reviews vary in methodological rigor. While systematic reviews report on methodology, detailed analysis is required in order to ensure that the review is unbiased. One technique for determining validity of systematic reviews is meta-analysis of the reviews themselves. These meta-analyses also present opportunities for data reuse, particularly the ability to critically examine and annotate intermediate data generated by the review process.

The fact that systematic reviews vary in quality is also important to systematic review data reuse. That is, while reusing SR data can save reviewer time, sometimes work must be done to determine whether potentially reusable data is any good. Note that this work can generate reusable data - user feedback, like annotations and ratings, can be used to make quick judgements about SR data. For example, if the extracted trial data is skewed by the design of its extraction form, a reviewer can leave an annotation indicating the bias. Annotations can also be used to correct inconsistent values in extracted trial data, or to even add missing data.

3. A System to Aid Reuse in SR Formulation

STARS provides features for managing reusable systematic review data. In this section, I introduce these features and the data model upon which they are built. I begin by indicating the types and logical organization of the data that STARS stores. I then discuss the design of SR-SQL, a query language for accessing STARS's data model. Following this, I discuss how the STARS data model can be implemented using a relational database and how SR-SQL can be automatically translated to standard SQL. Finally, I discuss STARS's high-level architecture.

3.1. Design goals

Before I discuss STARS at length, I will provide an overview of some of the principles which helped shape its design and implementation.

STARS is useful insofar as it aids management and reuse of systematic review data. As shown in the previous section, there are a large number of data that a system can potentially store to aid reuse. The cost of implementing each type of data is significant. Hence the following design principle: choose to manage data that jointly minimize implementation time and maximize reuse potential.

STARS aims to provide an interface to reusable data that is simultaneously simple and expressive. This is because we anticipate the primary workload of STARS to be *ad hoc* exploratory queries issued by medical professionals. Since such users may not have extensive database knowledge, it follows that just as the *types* of data that STARS manages must be chosen judiciously, their representation must also be carefully selected. Specifically: the logical organization of SR-data should reflect the user's mental model of the data and not be overly complex.

Since many systematic review data are implicitly related to clinical trials, (e.g. extracted study data, assessment of bias, etc.) STARS must include some minimal model for representing bibliographic trial data. Design of bibliographic databases for clinical trials has been studied extensively. Thus, our aim was not to create a system that implements every feature needed for a user-facing citation database. Rather, we designed STARS to represent a set of trial attributes that uniquely identify trials, can be imported from established bibliographic databases, and can be extended and described by user-generated SR-Data.

We use the relational model as our main point of reference in developing the data model for STARS. One goal of STARS was to determine the extent to which the relational model succeeds or fails in representing systematic review data.

3.2. Logical data organization

3.2.1 Trials

As mentioned above, managing data about clinical trials is a fundamental task. Many bibliographic databases have been created to index clinical trials. Among the most prominent are Pubmed and ClinicalTrials.gov, both of which are administered by the National Library of Medicine division of the NIH. Pubmed contains approximately 21 million citations of medical literature and collates data from many different sources, including the MEDLINE database and print journals [12]. ClinicalTrials.gov is a trial registry comprising approximately 124,500 citations of clinical trials [5]. ClinicalTrials.gov is unique in the fact that it stores trial results as well as extracted trial data and bibliographic information. Investigators of certain types of trials evaluating treatments under FDA regulation are legally required to register their trials with ClinicalTrials.gov [17]. Both ClinicalTrails.gov and Pubmed provide interfaces for exporting the data that they store: Pubmed through a web API, and ClinicalTrials.gov through a tool that allows users to export trial data as XML.

ClinicalTrials.gov was chosen as the source of trials early in the design process because it provides rich, exportable, semi-structured data about each trial in the registry. In future iterations of STARS, it will be necessary to import trial data from any potential source of clinical trial data. This follows from the fact that systems to support systematic reviews should collate data from many different sources, as is the case in systematic reviews themselves.

ClinicalTrials.gov has a document-based data model that is implemented using XML. This provides a flexible schema that maps well to the nested structure of trials. The attributes that I included for trial data are a subset of those stored by ClinicalTrials.gov. The schema that I use is flat rather than hierarchical, and values of attributes are automatically extracted from XML data made available on ClinicalTrials.gov. The ability to import trial data helps mitigate the difficulty of bootstrapping a system that stores primarily human-generated data. This goal is accomplished by providing utility even before users have contributed anything. For the purposes of our prototype, trial data are immutable but extensible. The data are shown in the following schema:

```
     (Trial ID, Title, Summary, Description, Status, Phase, Study
Type, Study Design, Condition, Intervention, Study Population,
Sampling method, Study start date, Study completion date)[1]
```

3.2.2 Systematic review data

Logically, a *systematic review* is a data object associated with an author and a review question. Each systematic review is also associated with a set of clinical trials. A clinical trial is included in a systematic review's trial set if the review's author determines that the trial it is

[1] These are only a small subset of the data recorded by ClinicalTrials.gov. The data stored in ClinicalTrials.gov represent only *some* of the possible data stored by a trial. In general, it is difficult (or impossible) to determine a schema for trial data *a priori*, meaning that the representation must be approximate

relevant to the review question.

Artifacts are structured objects that are generated during SR formulation. Systematic review artifacts can include extracted trial data, results of questionnaires assessing trial quality and bias (e.g. Jadad scale), and results of statistical meta-analysis. Artifacts are domain-specific, and therefore can represent and needed domain information. At this stage, the types of artifacts available to users are fixed by the system's implementation. In STARS, I focus on a novel type of artifact called an *outcome matrix*, which categorizes trials hierarchically according to predicates on extracted features.

Annotations are user-defined metadata about data objects. The user that creates a particular annotation is the *owner* of that annotation. Entities that can be annotated are called *annotatables*. Annotations are themselves annotatable. The simplest (i.e. least structured) type of annotation is a text comment. Annotations can also be used to add missing trial data. More complicated annotations include numeric ratings, which are potentially multi-part. In STARS, I represent these sorts of annotations using a simple key-value relation. Syntax is provided in SR-SQL to facilitate simple access to these annotations.

Interactions record actions made by users in the system. Interactions indicate the time at which the action was performed, the identity of the user performing the action, the type of action performed, and the systematic review affected by the action. For example, a user's searches can be modeled as a series of interactions. Creating an artifact can also be considered a type of interaction.

3.2.3 Artifacts: Outcome matrices

In STARS, I focus on storing and querying artifacts called *outcome matrices*, which are a class of data structures initially described by the Evidence Based Medicine group at Tufts.

An outcome matrix is a hierarchical data structure that is used to classify trials. Outcome matrices are hierarchical in that they are organized into ordered levels, where at each level the outcome matrix sorts trials by user-defined criteria. For example, one level might sort trials by comparisons of different drugs, whereas another might sort trials by the outcome that they measure. At each level, trials are placed into “buckets” based on whether they meet those criteria. For example, a trial will be placed in the bucket for ‘Drug A’ and ‘Drug B’ if a trial compares outcomes for those two interventions. The trials in each bucket are then subdivided by all categorizations possible in the level below, and so on. The ordering of levels, selection of categories, and placement of trials in buckets are user-defined.

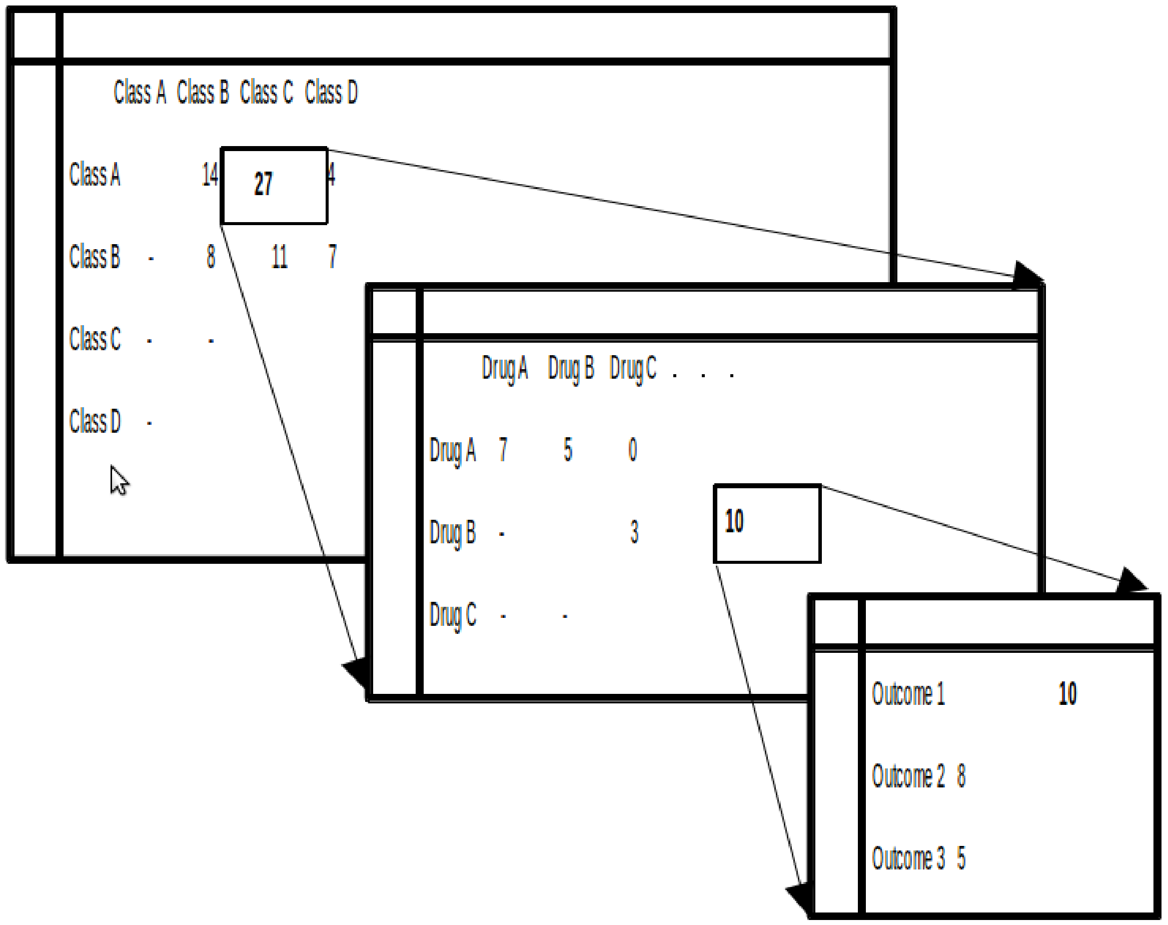


Figure 5: This is a visual representation of an outcome matrix, provided by the Tufts EBM group. Each box with labeled axes represents a *level*. The labels on axes are *categories*. The elements indexed by two categories represent trial sets. Trial sets are indicated by a number that represents their cardinalities. Note that some levels have two axes, but some only have one.

In order to speak more precisely about outcome matrices, I provide a formal definition. A *k*-level outcome matrix is a 4-tuple $< T, C, E, F >$, where T is a set of trials, C is a set of categories, E is a set of elements, and F is a function from E to subsets of T. C is sorted into a sequence of disjoint subsets $L_1, ..., L_k$. These are called the *levels* of the outcome matrix. An outcome matrix *element* is a sequence $S_1, ..., S_m$ such that each $S_i \in L_i \times L_i$, where $0 < i \leq k$. Each OM element is ordered as follows: The first member of the sequence is in $L_1 \times L_1$, and for every S_i and S_j, if Si directly succeeds S_j, then $S_i \in L_p \times L_p$ and $S_j \in L_o \times L_o$, where $p = o + 1$. If an element A is an initial subsequence of B, then B is *descended from* A. If the difference of their lengths is 1, then A is the *parent o*f B. F obeys the property that if B is descended from A, then $F(B) \subseteq F(A)$. If a trial is a member of some $F(A)$ where $A \in E$, then that trial is described by each pair of categories in A.

Queries over outcome matrices that facilitate reuse

Outcome matrices present an organized view of various trial features in a systematic review. An outcome matrix's structure and contents are largely user-defined and can therefore be designed to describe whatever trial criteria the review authors consider relevant. Using the outcome matrix described by Figure 4 as an example, I present sample questions that demonstrate the expressive power of outcome matrices in STARS:

1. How many trials compare drugs in class A and in class C?

This question might arise if a doctor wishes to determine the effectiveness of a drug in class A as compared to one in class C. To do so, the doctor can simply ask this question about all outcome matrices, then choose to read the review whose outcome matrix indicates largest number of trials that compare A and C. This way, the doctor selects the review containing evidence most relevant to her needs.

2. For outcome matrix M, which classes are compared with class A, and how many trials fall under each comparison?

A doctor can ask this question to determine the distribution of interventions within the systematic review containing outcome matrix M. This allows the doctor to see whether, e.g. comparisons of classes of drugs are represented equally by the systematic review. This is a relevant concern since included trials may skew towards a particular comparison of classes. By omitting the requirement that only M is considered, this question can also be used to determine which other classes of drugs are compared with class A. This could be useful for determining other possible interventions or comparisons.

3. Of trials that compare Class A and Class B, what outcomes do they report on?

This sort of question can be used by reviewers to conduct an analysis of outcome reporting bias in trials included by a systematic review. For example, if one group of trials reports on a certain outcome and another group fails to report on that outcome, it suggests that the second group might fail to report on that outcome because its results were not found to be statistically significant.

Language Support for Outcome Matrices

STARS provides a set of language features in SR-SQL that allow users to store and query outcome matrices. While an outcome matrix is strictly not a relation, SR-SQL's features allow outcome matrices to be included in SQL-style queries.

Collection of Outcome Matrix Elements

The first language feature that SR-SQL introduces to support this type of functionality is a named, unordered set of outcome matrix elements, called *OMs*. This collection can be included in the FROM clause of a SQL-style SELECT query, meaning that attributes of the OMs collection can be associated with tables stored in a relational database. One way of performing this association is through joins. The members of the collection OMs are, logically speaking, OM *elements*. These elements are related as described above; each element is characterized by one or two category labels on a given level. Each element may have a parent in a preceding level and children in a succeeding level. In order to establish relationships between OM elements, each element is identified by a unique ID. The language interface provided by SR-SQL provides a concise syntax for expressing relationships between outcome matrix elements that conceals the details of how OMs are implemented.

Consider the following query. Expressed in English, this query asks the following: “Find the number of outcome matrix elements comparing Drug A and Drug B.” Recall that an outcome matrix element compares two categories (e.g., Drug A and Drug B) if the *trials* residing in the OM element’s trial set compare those categories. When expressed in SR-SQL, this query is the following:

```
SELECT COUNT(O.ID) FROM OMs AS O[‘Drug A’, ‘Drug B’];
```

This shows how the outcome matrix abstraction is exposed to users. The outcome matrix collection, OMs, is included in the FROM clause as if it were a table. Each element returned from OMs is bound to an *alias*, O, which can be referenced in any other part of the query.

SR-SQL add syntax for qualifying which OM elements (from OMs) can be bound to O. For example, when O is parameterized as above, only elements comparing Drug A and Drug B will be selected from the collection.

Determining trial set membership

Outcome matrices are useful because they provide a structured way of organizing trials by category. This organization is performed by sorting trials into *trial sets* of outcome matrix elements, which are themselves described by a combination of categories. In order to allow for queries over outcome matrix trial sets, SR-SQL associates each outcome matrix element (e.g. the members of the OMs collection) with a trial set. This trial set is accessed through an OM element's `trial_set` attribute. The following query, Q2, illustrates how the `trial_set` attribute is used. In English, Q2 reads: "Find all outcome matrix elements that contain the trial 'NCT12345' in its trial set." In SR-SQL, the same query is expressed as follows:

```
SELECT O.ID FROM OMs AS O WHERE 'NCT12345' IN O.trial_set;
```

Logically, the trial set is an attribute of each OM element; it is referenced in much the same way as the ID of each outcome matrix element. Note the semantics of the "IN" operator, which forms a predicate like the following:

```
     … <value> IN <OM>.trial_set …
```

A predicate of this form evaluates to **True** if `<value>` is a valid trial ID stored in the trial set of the specified outcome matrix element.

Specifying category names with variables

As shown above, SR-SQL provides basic language features for selection on the categories of OM elements, and queries about OM elements' trial sets. The next set of features -

namely, aliasing of categories - facilitates a set more expressive queries.

In order to motivate the idea that category aliasing is desirable, consider the following: Suppose that a user knows the name of one category (C), and wants to find all other categories with which C is compared. For example, a review author might want to see the number of trials in the trial set for each outcome matrix element that compares C with some other category. In English, such a query might be stated as follows: find all interventions X that are compared with ‘insulin’, and the number of trials that compare ‘insulin’ and X.

The following SR-SQL query demonstrates the ability to bind category names to variables:

```
SELECT +intervention_B, count(T)
FROM OMs AS O[‘insulin’, intervention_B], trials as T
WHERE T.trial_ID in O.trial_set
GROUP BY +intervention_B;
```

Here, the category alias, called `intervention_B`, is declared at the same time as the outcome matrix alias O. The category name ‘insulin’ is specified as usual. Note how the “+” operator is used; when “+” is applied to the category alias, the alias is dereferenced to the name of a category of a specific outcome matrix element. This means that it is possible to include a (possibly unknown) category name in the SELECT, WHERE, GROUP_BY, or HAVING clauses of a SR-SQL query.

PARENT OF operator

Outcome matrices provide the ability to hierarchically organize trials, using different “levels” of categories. Logically, the closer a level is to root, the broader the categorizations that occur at that level. A query author can obtain more specific results by querying deeper into the structure of an outcome matrix. The following query exemplifies this sort of behavior: “For trials that compare Class A and Class B, on which outcomes do they report?" Recall that in the example outcome matrix given in Figure 4, Class is a *top level* category, whereas Outcome is a *bottom level* category. In order to express this query, it must be possible to specify parent-child relationships between outcome matrix elements. SR-SQL provides the PARENT OF operator to specify parent-child relationships between outcome matrix elements. Hence:

```
SELECT +outcome
FROM OMs AS O1['Class A', 'Class B'],
     OMs AS O2,
     OMs AS O3[outcome]
WHERE O1 PARENT OF 02 AND O2 PARENT OF O3;
```

In this case, the PARENT OF operator allows query-writers to specify a "path" from OM elements that categorize trials by Class to those that sort trials by Outcome.

Inserting outcome matrix data

The ability to write queries about outcome matrices depends on the fact that outcome matrix data actually exists in the database. This means that there must be some syntax for inserting outcome matrices. Determining an appropriate syntax turns out to be a fairly difficult task, given that outcome matrices are dense with information. That is, each outcome matrix encodes information about categories, levels, outcome matrix elements, and trial sets, and most of this information is user-defined.

In response to the complexity of outcome matrix insertion, SR-SQL proposes relatively simple syntax. Suppose that a user wishes to describe a small outcome matrix. On its first level, Drug A is compared against Drugs B and C. On its next level, the trials are sorted by the outcomes on which they report, Outcomes one, two, and three. This SR-SQL statement inserts such an outcome matrix into the OMs collection:

```
INSERT INTO OMs SELECT {
     [ ‘Drug A’, ‘Drug B’ ] = (‘NCT012’) ++
     {
          [‘Outcome one’] = (‘NCT012’)
     },
     [ ‘Drug A’, ‘Drug C’] = (‘NCT123’, ‘NCT456’) ++
     {
          [‘Outcome two’] = (‘NCT456’),
          [‘Outcome three’] = (‘NCT123’, ‘NCT456’)
     }
};
```

Each expression of the form:

```
     … [ ‘…‘,‘…‘ ] = ( ‘T1’, … ,‘Tn’ )…
```

defines an OM element with trials $T_1,\ldots,T_n$ in its trial set. This is called an OM initialization. The “++” operator after an OM initialization signifies addition of a lower level. Braces (e.g., `… { … } …`) encapsulate OM initializations in lower levels.

3.2.4 Annotations

Annotations are user-defined metadata associated with a given data object. In STARS, annotations serve two main purposes. First, annotations can be used to extend trial data. The ability to extend trial data is needed since the schema for trial data is not known *a priori*. That is, it is always possible for trial data to be incomplete.Second, annotations provide a useful way to indicate the quality and consistency of data stored in the repository.

While a wide range of data can be managed in ways that address these goals, STARS aims to provide three of the most important: free-form text comments, additions of missing data, and numeric ratings. To do so, STARS models annotations as a collection of key-value pairs. The annotation *key* indicates how the annotation should be interpreted (e.g. as a comment, extension, or rating). The annotation *value* holds the content of the annotation. Each pair is related to an annotatable data object and the user who created the annotation. Examples of annotatable data objects include outcome matrices, systematic reviews, and annotations themselves.

Sample queries on annotations that facilitate reuse:

1. Find the minimal Jadad rating for trials in systematic review S's trial set.

Suppose that S uses Jadad ratings to assess trial quality. This query can then be used to quickly find the quality cutoff designated by a systematic review. Since the data model for trials does not include Jadad rating by default, this query must be answered by joining the trials in S's trial set with annotations that have 'jadad rating' (or similar) as key.

2. Find all comments for trials with Jadad rating equal to '5'.

This query can be used to find discrepancies in user feedback about trials. For example, if the trials have a Jadad rating '5' (the maximal rating), then they should not have negative feedback in the comments section. Signs of conflict indicate that the annotations themselves should be reviewed.

Language support for annotations

SR-SQL provides an interface for extending the schema of a table with annotations. For example, suppose that a systematic review author wishes to record the Jadad score that she calculated for a trial, and that there is no attribute representing Jadad score in the schema for that trial. SR-SQL allows users to insert an annotation recording this extension, then query on the annotation as if it were an attribute of the annotatable table. Consider this query: find all comments for trials with Jadad rating equal to ‘5’. Using SR-SQL, this query can be expressed as follows:

```
SELECT title from trials AS T where jadad_rating = ‘5’;
```

Note that this can return a non-empty result set if `jadad_rating` is an attribute of the `trials` relation, or if there exists some annotation that has the key “jadad_rating”. In the former case, the literal value ‘5’ is compared to the row’s `jadad_rating` attribute. In the latter case, ‘5’ is compared to the value of an annotation associated with a row in the `trials` table.

3.2.5 Interactions

Interactions record user actions in interactive systems. STARS manages two types of interactions: records of searches over the repository and records of artifact creation. Interactions are recorded to make analysis of the review process easier. By the storing the temporal ordering of steps in the review process, inferences can be made about the nature and quality of the final product.

In STARS, an interaction comprises several important pieces of information. First, an interaction must be associated with a systematic review. Second, an interaction must record some indicator of when the interaction occurred in the systematic review process. This could be represented by a serial counter or a time stamp. It is also necessary to record the identity of the user who created the interaction. Using these basic components, it is possible to record both interactions recording searches and those recording the creation of outcome matrices.

Sample query that facilitates reuse:

What is the sequence of searches leading to a given outcome matrix?

The data returned by this question have several possible uses. For example, suppose that a user is auditing a systematic review for methodological rigor. Given the sequence of searches issued before the creation of an outcome matrix, it is possible to “play back” the series of searches by re-issuing them over their databases of origin. If the search results have grown significantly since the original query was issued, then it suggests that the systematic review needs to be updated in order to determine whether the new evidence is relevant to the question posed. This process is one of the steps indicated by the Cochrane Handbook for conducting updates of systematic reviews.

Another possible use of data returned by this query is to use search strategies successfully employed in the past to automatically generate suggestions of queries to reviewers searching on similar topics. While this is an interesting topic, we primarily focus on the first use case.

3.3 Representing SR-Data with Relations

3.3.1 The relational model

The relational data model is a common way of representing structured data. In this model, collections of related attributes (called tables) are used to represent data. The way in which these collections and attributes are logically organized is called the *schema*.

The relational model provides many benefits: it is conceptually simple, it allows system designers to specify consistent views of data, and it allows users to declaratively select data for retrieval and modification. This model, however, requires a level of structuredness that can be limiting when the schema of the stored data is not known *a priori*. Similarly, representing certain types of data - like hierarchical data - can be very complex. In STARS, we initially represented trial and SR data using the relational model. We found that this model, while conceptually simple, led to highly complex queries. It was in response to this complexity what we implemented the query language SR-SQL, which allows for queries that are simultaneously simple and expressive. While SR-SQL provides language features to ease the complexity caused by using a relational model for SR data, it may be useful to evaluate whether other models (e.g. the document model) may be better suited to storing SR data.

3.3.2 Representing Outcome Matrices using Relations

In STARS, outcome matrices are represented using a set of tables. While defining the schema of these tables is simple, accessing outcome matrices using traditional SQL grows complex as queries become sophisticated. The relations that collectively specify an outcome matrix in the data repository are the following:

The `outcome_matrices` relation:

```
      (annotatable_ID, parent_ID, root_parent_ID, categoryX,
categoryY, trial_bitmap, systematic_review_ID, ...)
```

The `outcome_matrix_ID` is an integer that uniquely identifies outcome matrix elements. All elements of an outcome matrix have an row in this table. An extra row is also generated as a $0th$ level OM element that serves a root element for the outcome matrix. This is is the element to which all other elements are associated. It provides a unique identifier for the outcome matrix *as a whole*, rather than for just one outcome matrix element. The `parent_ID` and `root_parent_ID` attributes are foreign key references to the `outcome_matrix_ID` attribute of other outcome matrix tuples; `root_parent_ID` refers to the ID of the extra, $0th$ level OM element, and `parent_ID` refers to the element the level directly above. `CategoryX` and `CategoryY` are foreign key references to tuples in the Categories relation. These represent categories that characterize members of an OM element's trial set. For elements that use only one category, e.g. elements that describe outcomes, CategoryY is set to `NULL`. The attribute `trial_bitmap` is a bit string specifying which trials from the systematic review's trial set are included in the element's own trial set. The attribute `systematic_review_ID` is a foreign key associating each outcome matrix element to the systematic review to which it belongs.[2]

The `categories` relation:

`(category_ID, outcome_matrix_ID, category_name, category_description)`

Each row in the Categories table is identified by an integer primary key called `category_ID`. Each category is also related to a root outcome matrix row by the foreign key `outcome_matrix_ID`. The `category_name` and `category_description` are what their names imply: text fields for a category's name and description.

The `trial_set` relation:

`(element_ID, systematic_review_ID, trial_ID)`

Trial sets are represented by a many-to-one relationship between the trial set table, which identifies set members by the foreign key `trial_ID`, and the systematic review relation, referenced by the foreign key `systematic_review_ID`. The primary key `element_ID` totally orders the trial set, allowing it to be masked by the bit string stored in each outcome matrix element.

[2] The schema for the outcome matrix table also includes attributes that allow it to be treated as an annotatable and as an interaction. These sets of attributes will be explained in the forthcoming discussions of annotations and interactions.

Translating SR-SQL to SQL: Outcome matrices

1. Collections of outcome matrix elements

I will now show how queries that reference the OMs collection are mapped to SQL queries referring to relations shown above. Recall this query from the previous discussion of SR-SQL:

```
SELECT COUNT(O.ID) FROM OMs AS O[‘Drug A’, ‘Drug B’];
```

When SR-SQL is translated to SQL, O is actually aliased to the `outcome_matrices` table. This table is then automatically joined to as many as two rows in the `categories` table. The translation process also adds a condition to the WHERE clause specifying that these categories have names equal to ‘Drug A’ and ‘Drug B’. The resulting query returns the ID’s of all outcome matrix elements categorized by the labels ‘Drug A’ and ‘Drug B’, and is shown equivalently in standard SQL:

```
SELECT COUNT(O.annotatable_ID) FROM outcome_matrices O
     JOIN categories c1 ON ( O.CategoryX = c1.category_ID AND
     O.root_parent_ID = c1.outcome_matrix_ID )
     JOIN categories c2 ON ( O.CategoryY = c2.category_ID AND
     O.root_parent_ID = c2.outcome_matrix_ID)
WHERE c1.category_name = ‘Drug A’ AND c2.category_name = ‘Drug B’;
```

2. Trial set membership

An example of a query incorporating predicates on trial set membership is the following:

```
SELECT O.ID FROM OMs O WHERE ‘NCT12345’ IN O.trial_set;
```

Recall that an outcome matrix’s trial set is represented by a bitstring that can be used to mask the set of trials included in a systematic review. This query simply looks for every outcome matrix element such that the set generated by masking the systematic review’s trial set with its bitstring still contains a the trial identified by `‘NCT12345’.`

This query can be shown equivalently in standard SQL:

```
SELECT O.annotatable_ID
FROM OMs AS O, trial_set AS T
WHERE
     T.systematic_review_ID = O.systematic_review_ID AND
     T.trial_ID = ‘NCT12345’ AND
     in_trial_set(T.element_ID, O.trial_bitmap);[3]
```

3. Specifying category names with variables

The example given for such a query is the following:

```
SELECT +intervention_B, count(T)
FROM OMs AS O[‘insulin’, intervention_B], trials as T
WHERE T.trial_ID in O.trial_set
GROUP BY +intervention_B;
```

Recall that values instantiated in the statement that defines the alias to the outcome matrix collection are mapped to the `category_name` attribute of the Categories relation. Thus, when this query is translated into an equivalent SQL query, all instances of the dereferenced variable `+intervention_B` are translated to equivalent references to Categories.category_name. Hence the SQL translation:

```
SELECT DISTINCT c2.category_name, count(T)
FROM outcome_matrices O, trials as T, trial_set as ts
     JOIN categories c1 ON ( O.CategoryX = c1.category_ID AND
     O.root_parent_ID = c1.outcome_matrix_ID )
     JOIN categories c2 ON ( O.CategoryY = c2.category_ID AND
```

[3] `in_trial_set( <element_ID> , <trial_bitmap> )` is a stored procedure that determines whether the *n*th position of the trial bitmap is set to 1, where *n* is equal to the position of the trial assigned by the serial key `element_ID`.

```
      O.root_parent_ID = c2.outcome_matrix_ID)
WHERE
      c1.category_name = ‘insulin’ AND
      ts.systematic_review_ID = O.systematic_review_ID AND
      ts.trial_ID = T.trial_ID AND
      in_trial_set(T.element_ID, O.trial_bitmap);
```

This translation also illustrates some of the complexity that can be caused by *composing* language features. While the SR-SQL expressed above is still comprehensible, the equivalent SQL query is perhaps too large to be constructed in an ad-hoc, exploratory setting.

4. The PARENT OF operator

The PARENT OF operator determines whether two outcome matrix elements are directly related. The translation of the PARENT OF operator is very simple: it simply takes any where-clause expression of the form … `O1 PARENT OF O2` … , where O1 and O2 are outcome matrix aliases, and converts the expression to `O1.annotatable_ID = O2.parent_id`. This allows for encapsulation of the underlying relational outcome matrix implementation.

5. Inserting outcome matrix data

While STARS does indeed process outcome matrix insertion, this is not an instance of translation. This is due to the fact that while an outcome matrix insertion logically comprises only one statement, it is composed of many interrelated operations in the underlying implementation. For example, one insertion must be performed for every outcome matrix element defined by the user. The system must also generate bit strings for each outcome matrix element’s trial set. While these operations could be performed by PL/SQL, in STARS they are instead performed during the process of parsing the INSERT statement.

3.3.3 Representing Annotations using relations

As described before, annotations are represented using a very simple key-value model. Annotations are associated with data objects. Those objects that can be annotated are called *annotatables*. Before I discuss how annotations are implemented, I will first discuss what it means to be annotatable.

In STARS, an object is annotatable if it implements the annotatable interface. In this context, I refer to an *interface* as a set of attributes that must be implemented (i.e. included) in a relation in order to enable a set of actions. In the case of the annotatable interface, the relevant action is that of associating annotations with the annotatable object. The interface is defined as follows:

```
    ( annotatable_ID, user_ID )
```

That is, any annotatable relation should contain a primary key, called the `annotatable_ID`, and a foreign key referencing the `user_ID` of the annotatable's owner. As mentioned before, annotations are themselves annotatable and thus implement this interface. Other annotatable relations include the `systematic_review` relation, the `outcome_matrix` relation, and certain relations representing interactions.

Annotations are represented by the `annotations` table:

```
    ( annotatable_ID, root_annotatable_ID, parent_ID, owner, key,
value )
```

Note that the `annotatable_ID` and `owner` attribute are supplied by the annotatable interface. The `parent_ID` is a foreign key referencing the annotatable described by a given annotation.

Translating SR-SQL to SQL: Annotations

While annotations have simpler structures than outcome matrices, writing queries about them can grow complex. This is due to the fact that a join is required for each annotation referenced in the query. For example, if a user wishes to find all comments for trials with a Jadad rating of '4', the user would have to issue the follow SQL query:

```
SELECT a1.comment
FROM trials as T, annotations as a1, annotations as a2
WHERE a1.key = 'comments'
     AND a2.key = 'jadad rating'
     AND a2.value = '4'
     AND T.annotatable_ID = a1.parent_ID
     AND T.annotatable_ID = a2.parent_ID;
```

This query requires joining three tables to answer a relatively simple question, meaning that it is also not well-suited to *ad hoc* data exploration.

SR-SQL addresses this problem by providing syntax for seamlessly integrating references to annotations into SQL-style queries. For example, the prior query could be easily expressed in SR-SQL:

```
SELECT T.comment FROM trials AS T WHERE T.jadad_rating = '4';
```

The translation is performed as follows: for each attribute referenced in the query that does *not* correspond to an actual attribute of the annotatable relation (e.g., `comment,` `jadad_rating`), replace the reference to that attribute with the "value" attribute of a joined instance of the `annotations` table, while specifying that the "key" is equal to the name of the non-existent attribute.

3.3.5 Representing Interactions using relations

Interactions, like annotatables, are represented by an interface. All relations that implement that interface must include the following set of attributes:

```
( creation_time, step_number, systematic_review_ID )
```

The `creation_time` attribute is a timestamp that is set when an interaction row is created. The step_number attribute is a serial integer that records the interaction's position within the systematic review. The `systematic_review_ID` attribute is a foreign key reference that ties the interaction to a given systematic review record.

In STARS, we manage records indicating user-issued searches and the creation of outcome matrices. As such, the `searches` table and `outcome_matrices` table both implement the interaction interface. The schema of the `searches` table is defined below:

```
    (annotatable_ID, query_string, result_cardinality,
creation_time, step_number, systematic_review_ID, owner)
```

Attributes included in the `annotatable` and `annotations` interfaces have the same meaning as demonstrated above. The `query_string` attribute stores the query issued over the trial repository and the `result_cardinality` attribute records the number of results returned by the search.

3.4 System architecture

STARS is constructed of four basic components: the user interface, the database, the interaction manager, and the translation layer. The dependencies between these components are illustrated in Figure 5.

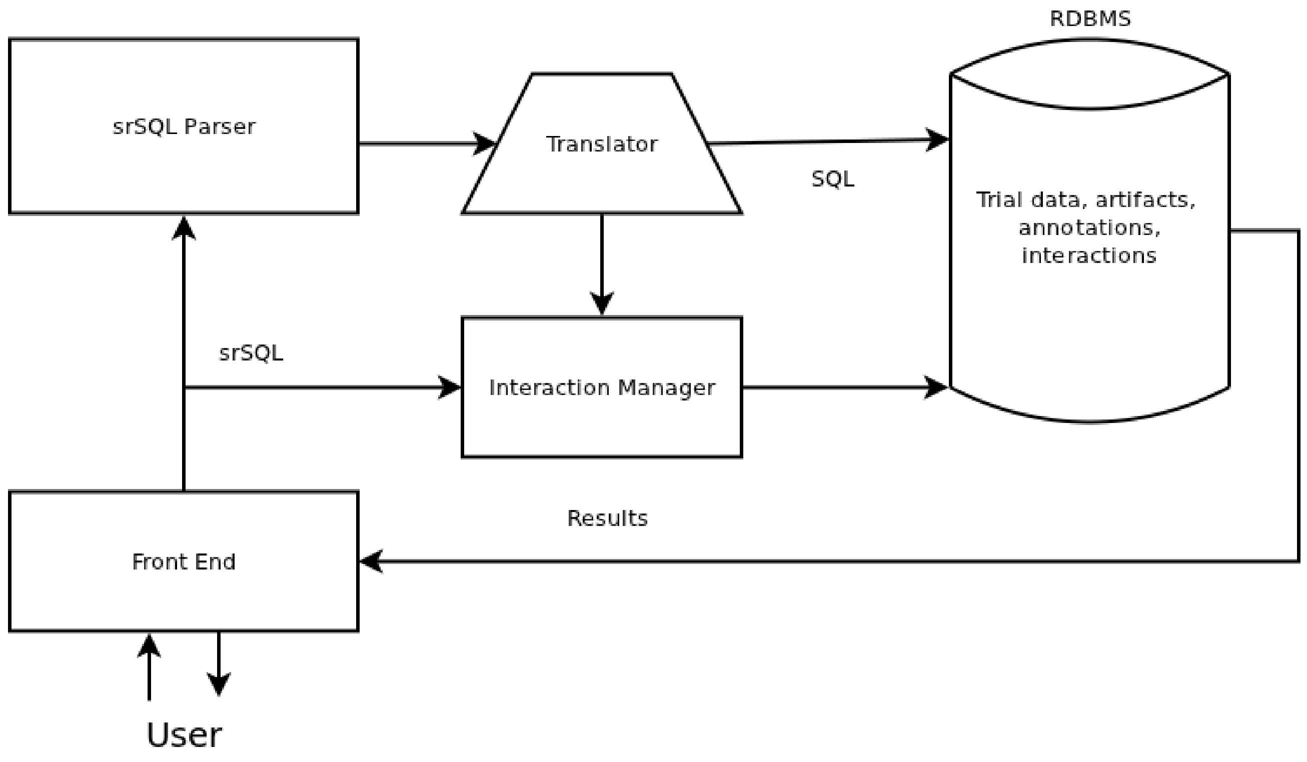


Figure 5: The dependencies between the architectural components of STARS.

The user interface captures user interactions, i.e. SR-SQL queries, and relays them to the translation layer. In the translation layer, those queries are parsed and then translated into standard SQL. These interactions are logged by the interaction manager. Finally, the SQL query is issued over the database and results are returned to the user.

STARS's user interface consists of a simple REPL-style shell, built on top of the psql command-line tool for accessing PostgreSQL in Unix-like environments. While more a more advanced and intuitive user interface is imperative for a user-facing system, a shell-style interface is appropriate for testing and interacting with SR-SQL's language features, especially since the focus of this thesis is data management, rather than interface design.

The translation layer consists of a parser and a query rewriter. The user's SR-SQL input is parsed and converted into an abstract syntax tree, or AST. The AST is then passed as input to the rewriter, which makes a number of transformations. In the final phase of the rewriting process, the parser generates a SQL query from the result.

The database is a standard instance of PostgreSQL, and the interaction manager consists of a method that records interactions in the database at each step of the Read-Evaluate-Print Loop.

3.5 System implementation

STARS is primarily implemented using Java. I used ANTLR [16], a parser generation tool, to specify and generate the parser and rewriters. Parsing SR-SQL is done in four stages. In the first stage, SR-SQL is tokenized, parsed, and incrementally converted into an AST representation. Here, the names of symbols that refer to tables are recorded. These symbols include tables name, aliases, and artificial table names (e.g. OMs). In the second stage, references to variables and artificial table names are resolved to actual attributes. This is done by issuing queries over the PostgreSQL database to first determine the attributes that are actually defined in the database, then retrieve appropriate information about the SR data tables in order to generate joins. In this stage, tables to be added to the FROM clause and propositions to be added to the WHERE clause are buffered. In the third stage, the buffered additions to the FROM and WHERE clauses are injected into the AST. In the final stage, the AST is converted back to standard SQL using the StringTemplate engine. This was done by mapping each part of the AST to a template that extracts token values and generates the appropriate part of the SQL query. These parts are composed recursively by ANTLR as the AST is parsed to form the final query.

4. Related work

Various computational methods have been developed for optimizing or reusing certain parts of the systematic review process. I will provide a brief survey of these methods, which I sort into two categories: search and information retrieval, and data management. Whereas STARS aims to facilitate both data storage *and* data retrieval, these methods are appropriate for one task, but not the other.

RevMan [14] and RevBase [13] are examples of data storage projects that fail to adequately address data retrieval. RevMan is a software suite designed to aid the preparation of systematic reviews. While it does provide limited data management functionality, it is designed to prepare a document for publication rather than store SR data for reuse. RevBase is closer to the mark, as it provides data management functionality for each step of the systematic review process. It does not, however, provide any mechanism for *querying* over systematic review data, and thus does not provide for data reuse.

The Tufts EBM group has been involved in the creation of a Systematic Review Data Repository (SRDR) [18]. The primary purpose of SRDR is to provide a central locale for storing extracted trial information and data extraction forms, as well as annotations assessing the quality of the data stored. While this does present clear opportunities for reuse, as stored data is accessible from *other* systematic review groups, the system is limited to storing only a few types of reusable data and does not provide an expressive interface for querying SR data. Similar concerns apply to ClinicalTrials.gov, which stores a extracted trial data uploaded by investigators; ClinicalTrials.gov provides a simple search form rather than an expressive query language.

CiteSeer [4] and CoBib [2] provide community-managed bibliographic data. These efforts, however, focus on academic research literature rather than medical literature. As a result, they have no mechanism for creating artifacts or other types of domain-specific reusable data, and are not suited for the domain-specific tasks presented by systematic review data management.

Other projects adequately aid searching systematic review data but neglect other data management tasks. There has been much work done into developing search methods for clinical trials [15][16], but little work into managing and searching for reusable systematic review data.

Abstrackr [3] , which was developed at Tufts in collaboration with its EBM group, uses active machine learning to optimize the citation screening process. Reviewers are presented with a series of abstracts, which they label according to relevance. Abstrackr uses this feedback to train machine learning algorithms that separate irrelevant citations from trials that might be relevant. Abstrackr does provide for a sort of data reuse, as it allows review authors to reuse past *experience* to determine relevant trials. It does not, however, present an interface for searching over or manipulating that data. Abstrackr might be useful as a means for data entry for STARS. That is, in addition to providing its primary function as a citation screening tools, Abstrackr could piggyback functionality that enters annotations into STARS's database.

The CQMS project at University of Washington has similar aims to STARS regarding management of interaction history. A major aim of the project is to use user experience and expertise to help non-expert users build sophisticated queries. The SnipSuggest system [11], for example, uses previously issued queries to predict the data retrieval needs of users during data exploration. This is done by presenting the user with possible autocompletions of SQL queries. Application of similar methods to STARS could provide interesting ways of analyzing of interaction histories and also mitigate some of the difficulty introduced by utilization of a SQL-like query language.

5. Conclusions and Future work

In this thesis, I present STARS, a system to aid reuse of systematic review data. The main intellectual contributions of STARS are a detailed analysis of reuse possibilities in the systematic reviewing process and the introduction of the SR-SQL query language, which provides interesting ways of querying over systematic review data.

One avenue for future exploration is the question of which data model best represents systematic review data. As we have seen, the relational model does not map well to the systematic review domain; a more natural choice might have been hierarchically-structured document data. Therefore re-implementing SR-SQL using a different model might provide valuable insights.

Given the vast variety of reusable data presented by the systematic review process, STARS currently facilitates only a small fraction of possible data reuse scenarios. Ways to expand the power and scope of the system might include the specification of a data definition language for artifacts that automatically generates a representation of the artifact and query language extensions for manipulating that data. This would allow the user to create data structures that map well to his or her needs.

Providing an intuitive user interface is essential to the success of a system like STARS, which aims to be used in some capacity by non-experts. Thus, research into ways of best combining the precision and clarity of a query languages with the intuitiveness of visualization techniques may help improve the chances of the medical community adopting a system like STARS.

Finally, it is worthwhile to consider ways in which interaction data can be used to improve the process of generating searches over trial databases. For example, it may be possible to develop techniques for sampling query results so that it is possible to search for queries by elements that appear in their results. Such a technique might be used to develop higher-quality searches given limited information about trials that out to appear in search results.

References

1. B. Shea, J. Grimshaw, G. Wells, M. Boer, N. Andersson, C. Hamel, A. Porter, P. Tugwell, D. Moher, and L. Bouter. Development of AMSTAR: a measurement tool to assess the methodological quality of systematic reviews. BMC Medical Research Methodology (2007) 7:10.

2. B. Trushkowsky, K. Campbell, J. Forbes. An architecture for a collaborative bibliographic database. In Proc. of the TAPIA '07 Conference on Diversity in Computing (2007)

3. B. Wallace, K. Small, C. Brodley, and T. Trikalinos, Active Learning for Biomedical Citation Screening. In Proc. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), 2010.

4. CiteSeer: http://citeseerx.ist.psu.edu/index

5. ClinicalTrials: http://www.clinicaltrials.gov/

6. Cochrane Collaboration: http://www.cochrane.org/

7. D. Sackett, W. Rosenberg, J. Gray, R. Haynes, W. Richardson. Evidence based medicine: what it is and what it isn't. BMJ (1996), 312:71.

8. E.K. Lee, H. Lee, and A. Quarshie. SEACOIN – An Investigative Tool for Biomedical Informatics Researchers. AMIA Annu Symp Proc. (2011): 750–759.

9. Elamin M.B., Flynn D.N., Bassler D., Briel M., Alonso-Coello P., Karanicolas P.J., Guyatt G.H., (...), Montori V.M., Choice of data extraction tools for systematic reviews depends on resources and review complexity, Journal of Clinical Epidemiology, (2009), 62 (5), pp. 506-510.

10. J.P.T. Higgins, S. Green, Eds. Cochrane Handbook for Systematic Reviews of Interventions Version 5.0.2 [updated September 2009]. The Cochrane Collaboration, 2009. Available from www.cochrane-handbook.org.

11. N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu. SnipSuggest: Context-Aware Autocompletion for SQL. PVLDB Vol 4 Number 1 - VLDB 2011

12. PubMed: http://www.ncbi.nlm.nih.gov/pubmed/

13. RevBase: http://www.revbase.org/html/about

14. RevMan: http://ims.cochrane.org/revman

15. R. I. Dogan, G. C. Murray, A. Neveol, and Z. Lu. Understanding PubMed user search behavior through log analysis. *Database.* (2009)

16. T.J. Parr, and R.W. Quong. ANTLR: A predicated-*LL(k)* parser generator. Software Practice and Experience, (1994).

17. U.S. Public Law 110-85 : http://prsinfo.clinicaltrials.gov/fdaaa.html

18. S. Ip, N. Hadar, S. Keefe, C. Parkin, R. Iovin, E.M. Balk, and J. Lau. A Web-based archive of systematic review data. *Systematic Reviews* 2012, 1:15