AutoJoin: Generalizing an Example into an EDM query

Jack Mostow and Bao Hong (Lucas) Tan
mostow@cs.cmu.edu, btan@andrew.cmu.edu
Project LISTEN, School of Computer Science, Carnegie Mellon University

Abstract. This paper describes an implemented method to generalize an example tutor interaction into a query to retrieve similarly related sets of events. It infers `WHERE` clauses to equate repeated values unlikely to match by accident.

The Session Browser [1] shown in **Figure 1** is an EDM tool to view data retrieved by querying a database of events logged by a tutor. It displays retrieved events in a context tree of enclosing events, with a 1-line summary of the database record for each event.

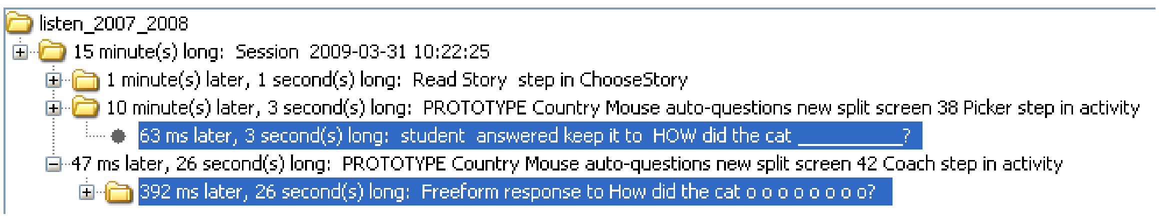

Figure 1: Event context tree highlighting two events selected by the user to AutoJoin

This brief example occurred in an activity to teach children to ask themselves questions about the text they read. The first highlighted event summarizes a child's multiple-choice response to a prompt to fill in the rest of a question about the text. The second event describes the child's spoken response to a prompt to speak the completed question aloud.

Often we want a query to retrieve examples similar to a current example. Complex queries are hard to construct, so we developed AutoJoin to generate them automatically. AutoJoin generalizes the two highlighted events into a query that finds "similar" cases, in this case a multiple choice step immediately followed by a free-form response step:

```
SELECT * FROM
  multiple_choice_question mcq,
  sentence_encounter se
WHERE mcq.activity_directory = se.activity_directory
    /* '..\data\stories\PROTOTYPE Country Mouse auto-questions new
    split screen' */
AND mcq.end_time = se.step_start_time /* '20090331103359' (03/31/2009
    10:33:59 AM) */
AND mcq.machine_name = se.machine_name /* 'LISTEN07-211' */
AND mcq.story_encounter_start_time = se.story_encounter_start_time
    /* '20090331102336' (03/31/2009 10:23:36 AM) */
AND mcq.user_id = se.user_id /* 'mKJ9-5-2001-07-23' */;
```

(The example here is simple for brevity; AutoJoin can generalize from more events too.)

AutoJoin constructs a query as a join of the tables where the selected events were logged. It abbreviates each table by its initials, e.g. `multiple_choice_question` as `mcq` and infers **`WHERE`** clauses by equating field values that show up more than once in those events. The comments, in green, show which values it abstracted into variables; they help the user understand the query and fix over-generalizations easily by uncommenting.

AutoJoin assumes that (1) unlikely matches matter, but (2) their specific values do not, so it (1) infers that the variables must be equal, but (2) abstracts away their specific values. When (1) is wrong, it under-generalizes; when (2) is wrong, it over-generalizes. To avoid under-generalizing, it uses heuristics to prevent meaningless matches. It gauges the likelihood of accidental match from how often the matching value occurs in each field. For instance, NULL values are frequent in many fields due to various reasons, one of which is its use as a no-value indicator. Thus NULL values in two such fields are likely to match by accident. In contrast, data types such as strings and dates have a large range of possible values. Assuming that they are unlikely to match by accident saves time by not bothering to estimate the frequency of the matching value in the two table columns.

Computing how often a given value occurs in a table column takes time for a large table, so AutoJoin estimates it from a sample of 400 rows. This sample is fast to retrieve but may be unrepresentative, especially if the column is sorted. Sampling blocks of 20 rows at 20 randomly chosen offsets is more reliable, but slow enough to be worth caching.

Another heuristic to avoid under-generalizing classifies certain columns as "non-cross-match" and compares them only with columns of the same name in other tables. For example, the table `story_encounter` has non-cross-match column `story_count`, and `sentence_encounter` has non-cross-match columns `sentence_count` and `word_count`. Since `story_count`, `sentence_count`, and `word_count` count different types of things, we assume it does not make sense to compare them. In contrast, comparing `start_time` and `end_time` from different tables does make sense.

The "non-cross-match" heuristic eliminates additional spurious matches, but relies on the naming convention it exploits, and on the user's knowledge of which fields make sense to compare. In contrast, estimating the frequency of matching values does not depend on column names or knowledge of the database schema, so it's a more general method.

We have identified the EDM task of generalizing from a single example of tutorial interaction, described AutoJoin's simple but powerful heuristics, and reported their implementation in the Session Browser. By systematically generating clauses we might omit or mistype, it helps us build complex joins much faster than by hand. It's limited in what it can notice, and it sometimes generalizes accidental matches, so we check its output, but checking is faster than writing queries. AutoJoin may also apply to many non-EDM domains. It seems too simple to be novel, but we have not found it elsewhere.

Acknowledgements: This work was supported by the Institute of Education Sciences, U.S. Department of Education, through Grant R305B070458, R305A080157, and R305A080628 to Carnegie Mellon University. The opinions expressed are those of the authors and do not necessarily represent the views of the Institute or the U.S. Department of Education. We thank Session Browser users for testing AutoJoin, and the educators, students, and LISTENers who helped generate, collect, and analyze our data.

References (also see Publications page at www.cs.cmu.edu/~listen)

[1] Mostow, J., J. Beck, A. Cuneo, E. Gouvea, C. Heiner, and O. Juarez. Lessons from Project LISTEN's Session Browser. In C. Romero, et al., Editors, *Handbook of Educational Data Mining*. Taylor & Francis Group, in press.