An in-situ, one-shot algorithm for updating score tables

Kjell Post
Irsta Fotostudio
kjell@irstafoto.se

Abstract

In competitions, a two-dimensional table is often used to record scores given by a judge for a certain participant. When the structure of the score table needs to be updated (adding/removing judges, adding/removing participants) we like the keep the existing scores and grow the table to accomodate new entries for the new judges and participants. This memorandum presents an algorithms for creating new entries for a given set of added/removed judges and added/removed participants without having to copy the existing entries.

1 Updating score tables

Consider a score table $S(i, k)$ that stores the score s_{ik} for participant i, given by judge k. Such a table can be represented either as a 2D matrix or as a set of tuples (p_i, j_k, s_{ik}). The tuple representation is the obvious representation for an SQL database.

For instance, consider the score table in fig. 1 (left) where scores have already been given. Now, let's assume that we need to retire judge j_4 and add judges j_5 and j_6. In addition, we would like to remove participant p_3 and add participant p_4. This would result in the table shown in fig. 1 (right).

	j_1	j_2	j_3	j_4
p_1	72	86	85	78
p_2	66	68	68	69
p_3	92	90	91	95

	j_1	j_2	j_3	j_5	j_6
p_1	72	86	85		
p_2	66	68	68		
p_4					

Figure 1: A sample score table (left), and after removing judge j_4, adding judges j_5, and j_6, removing participant p_3 and adding participant p_4 (right).

The algorithm presented in the next section describes how to add and remove tuples to accomodate a given set of changes in judges and participants. Rather than creating a new table and copying the current scores, the algorithm described here creates and deletes only the tuples necessary. Furthermore, the algorithm can handle all four kinds of requests in one go.

2 The algorithm

In the most general case we may be asked to

- Remove a set of participants: *participantsOFF*.
- Add a set of participants: *participantsON*.
- Exclude a set of judges: *judgesOFF*.

- Use a new set of judges: *judgesON*.

(The astute reader may notice the word "exclude", rather than "remove", the reason being that in a typical application, the set *judgesOFF* may contain judges not currently amongst the present ones. Likewise, *judgesON* will be the new set of judges, which includes the current ones. This has to do with the user interface for this application.)

We may illustrate these changes as in fig. 2.

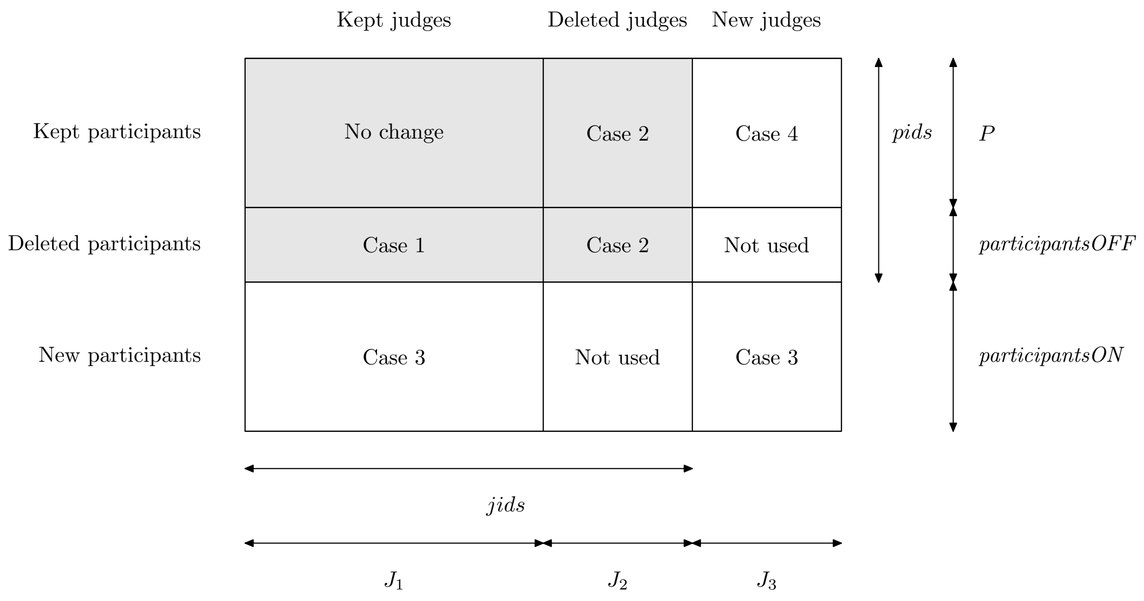

Figure 2: Changes after adding/removing judges and participants. The grey area is the original scoretable.

We will proceed in four steps, corresponding to the cases shown in fig. 2. But first some notation to help us along:

Let *pids* be the set of current participants.
Let *jids* be the set of current judges.

In other words, all the tuples (p_i, j_k, s_{ik}), where $i \in \mathit{pids}, k \in \mathit{jids}$, is the set of scores before the changes take place.

Let $J_1 = \mathit{jids} - \mathit{judgesOFF}$ be the set of original judges, minus the deleted judges.
Let $J_2 = \mathit{jids} \cap \mathit{judgesOFF}$ be the set of original judges removed from this event.
Let $J_3 = \mathit{judgesON} - \mathit{jids} - \mathit{judgesOFF}$ be the set of newly introduced judges.
Let $P = \mathit{pids} - \mathit{participantsOFF}$ be the set of current participants minus deleted participants.

The four steps then, are:

1. Delete all (p_i, j_k, s_{ik}) where $p_i \in \mathit{participantsOFF}, j_k \in J_1$.
2. Delete all (p_i, j_k, s_{ik}) where $p_i \in \mathit{pids}, j_k \in J_2$.
3. Add (p_i, j_k, NULL) where $p_i \in \mathit{participantsON}, j_k \in \mathit{judgesON}$.
4. Add (p_i, j_k, NULL) where $p_i \in P, j_k \in J_3$.

The reader should be able to convince himself, using the diagram in fig. 2, that these steps adds and removes the necessary tuples to reflect the new score table.