Explanation and Model Solution for Example Assignment: The Relational Algebra

COMPSCI 2DB3: Databases

Jelle Hellings Holly Koponen

Department of Computing and Software
McMaster University

Foreword

The model solution for part one ("The requested queries") consists of consists of three parts. First, there is the original query description (from the assignment). Next, there is the rational of our Relational Algebra solution. Finally, there is the resulting Relational Algebra query. The model solution for part two ("Efficiency of queries") consists of a solution with an optional rational (if there is something to explain besides the explanation that should be part of the report).

Model Solution

The requested queries

1. High-level query: 'Find all multi-day events'. Detailed description:

   > The community leader wants to include filters on the website to search based on the length of events (e.g., to distinguish between a concert and multi-day festivals).
   >
   > Write a query that returns a copy of the event table that only contains those events that are multi-day events.

   Rational:

   This query is as simple as possible: filter data from a single table. Hence, we only need to perform a selection (no joins or projections are necessary: all data is in the **event** table and we need all columns of that table).

   Solution:

   $$\sigma_{startdate<enddate}(\mathsf{event}).$$

2. High-level query: 'Find neighborhood events'. Detailed description:

   > The community leader wants to assure that all users will easily find the events in their direct neighborhood. To do so, the community leader wants to assure that users always get recommended all events that are organized in their own postal code.
   >
   > Write a query that returns pairs (*user*, *event*) in which *user* is a user identifier and *event* is an event identifier such that the user and the event share the same postal code.

Rational:

This query asks for a straightforward join between **user** and **event** based on their respective postal codes (attributes *postcode* in both tables). As *postcode* is the only shared attribute between both tables, we can use a natural join. After this join, we only keep the requested attributes: attributes *uid* (from user) and *eid* (from event). Finally, we rename these attributes to *user* and *event*, respectively, to satisfy the description.

Solution:

$$\rho_{uid \mapsto user, eid \mapsto event}(\pi_{uid,eid}(\rho_U(\mathsf{user}) \bowtie \rho_E(\mathsf{event}))).$$

Alternatives. Instead of the natural join, you can also use a conditional join:

$$\rho_{uid \mapsto user, eid \mapsto event}(\pi_{uid,eid}(\rho_U(\mathsf{user}) \bowtie_{U.postcode=E.postcode} \rho_E(\mathsf{event}))),$$

or a cross product with selection:

$$\rho_{uid \mapsto user, eid \mapsto event}(\pi_{uid,eid}(\sigma_{U.postcode=E.postcode}(\rho_U(\mathsf{user}) \times \rho_E(\mathsf{event})))).$$

3. High-level query: 'Find all unreviewed events'. Detailed description:

   The community leader wants to promote events that have not yet been reviewed, this to encourage participation in new events and encourage reviewing.
   Write a query that returns a copy of the event table that only contains those events that do not have reviews.

Rational:

We want to filter the **event** table by keeping all events that are *not* reviewed: the **review** table has all event identifiers (attribute *event*) of all events *that we do not want in the output*:

$$N := \rho_{event \mapsto eid}(\pi_{event}(\mathsf{review})).$$

Next, we compute the set of event identifiers of all events that we *do want in the output*:

$$K := \pi_{eid}(\mathsf{event}) \setminus N.$$

Finally, we take the natural join of **event** and K: this will filter the **event** table and only keep those events whose event identifier is in the set K of all events *that we do want in the output*:

$$A := K \bowtie \mathsf{event}.$$

Solution:

$$\begin{aligned} N &:= \rho_{event \mapsto eid}(\pi_{event}(\mathsf{review})), \\ K &:= \pi_{eid}(\mathsf{event}) \setminus N, \\ A &:= K \bowtie \mathsf{event}. \end{aligned}$$

4. High-level query: 'Find all optimally-annotated events'. Detailed description:

   According to the community leader, events with *exactly three* keywords have the highest engagement. To aid event organizers, the community leader wants to label these optimally-annotated events.
   Write a query that returns a list of all events (column *eid* of the **event** table) that have exactly three keywords.

Rational:

We can query for all events with *exactly three keywords* in three steps. First, we find all events that have at-least three keywords:

$$T := \pi_{A.event}(\sigma_{e_3 \wedge d_3}(\rho_A(\text{keyword}) \times \rho_B(\text{keyword}) \times \rho_C(\text{keyword}))),$$

in which $e_3 := A.event = B.event \wedge B.event = C.event$ and $d_3 := A.word \neq B.word \wedge A.word \neq C.word \wedge B.word \neq C.word$. The above will not only return events (via their identifier) with exactly three keywords, but also events with *more than three keywords*. Next, we find all events that have *more than three keywords* (at-least four keywords):

$$F := \pi_{A.event}(\sigma_{e_4 \wedge d_4}(\rho_A(\text{keyword}) \times \rho_B(\text{keyword}) \times \rho_C(\text{keyword}) \times \rho_D(\text{keyword}))),$$

in which $e_4 := A.event = B.event \wedge B.event = C.event \wedge C.event = D.event$ and $d_4 := A.word \neq B.word \wedge A.word \neq C.word \wedge A.word \neq D.word \wedge B.word \neq C.word \wedge B.word \neq D.word \wedge C.word \neq D.word$. As the final step, we remove the at-least four keyword events (F) from the at-least three keyword events (T) to get all events with *exactly three keywords* and we rename the resulting column as specified in the description:

$$A := \rho_{event \mapsto eid}(T \setminus F).$$

Note that this query uses the *At-least n* and *Exact n* patterns from the slides.

Solution:

$$\begin{aligned}
T &:= \pi_{A.event}(\sigma_{e_3 \wedge d_3}(\rho_A(\text{keyword}) \times \rho_B(\text{keyword}) \times \rho_C(\text{keyword}))),\\
F &:= \pi_{A.event}(\sigma_{e_4 \wedge d_4}(\rho_A(\text{keyword}) \times \rho_B(\text{keyword}) \times \rho_C(\text{keyword}) \times \rho_D(\text{keyword}))),\\
A &:= \rho_{event \mapsto eid}(T \setminus F),
\end{aligned}$$

in which

$$\begin{aligned}
e_3 &:= A.event = B.event \wedge B.event = C.event,\\
d_3 &:= A.word \neq B.word \wedge A.word \neq C.word \wedge B.word \neq C.word,\\
e_4 &:= A.event = B.event \wedge B.event = C.event \wedge C.event = D.event,\\
d_4 &:= A.word \neq B.word \wedge A.word \neq C.word \wedge A.word \neq D.word \wedge\\
&\quad B.word \neq C.word \wedge B.word \neq D.word \wedge C.word \neq D.word.
\end{aligned}$$

5. High-level query: 'Find the most recent activity'. Detailed description:

   To drive engagement with user profiles, the community leader wants to list the most recent activities of users on their profile. In specific, the community leader wants to add the most-recent review and the review of the most-recent event to the user pages of people that reviewed events.

   Write three queries:

   (a) a query that returns pairs (*user*, *event*) in which *user* is a user identifier and *event* is the event identifier of the event for which the user wrote a review most-recently (according to *reviewdate*);

Rational:

We want, per user, to find the reviews in **review** with the highest (largest) review date (attribute *reviewdate*). As a first step, we compute the set of all reviews, per user, of reviews that *do not have the highest review date* (as there is another review by the same user with a higher review date):

$$N_r := \pi_{A.user,A.event}(\sigma_{A.user=B.user \wedge A.reviewdate<B.reviewdate}(\rho_A(\text{review}) \times \rho_B(\text{review}))).$$

Then we remove this set N of all events that have reviews: as the set N contains all reviews, per user, that do not have the highest review date, the result will be all events, per user, with the highest review date:

$$A_r := \pi_{user,event}(\text{review}) \setminus N_r.$$

Note that this query uses the *largest value* pattern from the slides.

Solution:

$$N_r := \pi_{A.user,A.event}(\sigma_{A.user=B.user \wedge A.reviewdate<B.reviewdate}(\rho_A(\text{review}) \times \rho_B(\text{review}))),$$
$$A_r := \pi_{user,event}(\text{review}) \setminus N_r.$$

(b) a query that returns pairs (*user*, *event*) in which *user* is a user identifier and *event* is the event identifier of the most-recent event (according to *enddate*) for which the user wrote a review;

Rational:

We want, per user, to find the reviews in **review** for events with the the highest (largest) end date (attribute *enddate* in **event**). This query follows the same pattern as the previous query, the main difference being that we need to combine the review and event table to get the end date of each reviewed event. Note that this query also uses the *largest value* pattern from the slides.

Solution:

$$C = \sigma_{\text{review}.event=\text{event}.eid}(\text{review} \times \text{event}),$$
$$N_e := \pi_{A.user,A.event}(\sigma_{A.user=B.user \wedge A.enddate<B.enddate}(\rho_A(C) \times \rho_B(C))),$$
$$A_e := \pi_{user,event}(\text{review}) \setminus N_e.$$

(c) a query that returns triples (*user*, *lreview*, *levent*) in which *user* is a user identifier, *lreview* is the event identifier of the event for which the user wrote a review most-recently, and *levent* is the event identifier of the most-recent event for which the user wrote a review.

Rational:

We simply join the answers from the previous two queries together based on their *user* attributes. Before we perform the join, we rename the event attribute from A_r to *lreview* and the event attribute from A_e to *levent* in accordance with the description. After this rename, we can perform a natural join, as only the attribute *user* is common among the two joined queries.

Solution:

$$\rho_{event \mapsto lreview}(A_r) \bowtie \rho_{event \mapsto levent}(A_e).$$

6. High-level query: 'Find postal code experts'. Detailed description:

   The event website is community-driven and will only succeed with enthusiastic participation of its users. To further encourage such participation, the community leader wants to hand out *postal code badges* to all users that reviewed all events organized in their postal code.

   Write a query that returns all users (column *uid* from the **user** table) that have reviewed all events that are organized in their postal code.

   Rational:

   To find all users that have reviewed all events organized in their postal code, we will look for users that did not review all events in their postal code. To do so, we first pair each user with *all events* in their region:

   $$P := \pi_{uid,eid}(\sigma_{\mathsf{user}.postcode=\mathsf{event}.postcode}(\mathsf{user} \times \mathsf{event})).$$

   To find all events (organized in their postal code) a user *did not* review, we take P and remove all events the user did review:

   $$N := P \setminus \rho_{user \to uid, event \to eid}(\pi_{user,event}(\mathsf{review})).$$

   If we project N on *uid*, then we end up with a set S of all user identifiers of users that *did not review all events in their postal code*. To obtain the user identifiers of all users that have reviewed all events that are organized in their postal code, we simply remove the set S from all user identifiers:

   $$A := \pi_{uid}(\mathsf{user}) \setminus \pi_{uid}(N).$$

   Note that this query does not use the *division* pattern from the slides exactly, but the approach is similar. The division pattern would fit exactly for queries such as "return all users that reviewed all events", in which case we compare the reviews of each user with the *same* set of all events. The asked query did not compare the reviews of each users with the *same* set of all events, but with a different set of events per user.

   Solution:

   $$\begin{aligned} P &:= \pi_{uid,eid}(\sigma_{\mathsf{user}.postcode=\mathsf{event}.postcode}(\mathsf{user} \times \mathsf{event})), \\ N &:= P \setminus \rho_{user \to uid, event \to eid}(\pi_{user,event}(\mathsf{review})), \\ A &:= \pi_{uid}(\mathsf{user}) \setminus \pi_{uid}(N). \end{aligned}$$

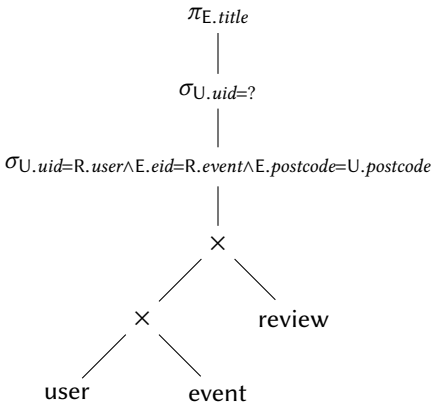
Efficiency of queries

The consultant that originally advised our local community leader is worried about whether some of the complex queries can be implemented efficiently. The consultant has specific concerns about two such queries. To aid the consultant, we will be providing estimates of the complexity of these two queries in terms of the size of intermediate query evaluation results. To do so, the consultant provided the following rough estimates of the involved data: there are 1000 rows in **user**, 5000 rows in **event**, 25000 rows in **keyword**, 5000 rows in **region**, and 25000 rows in **review**. Furthermore, the consultant estimated that all users write an equal amount of reviews (25 reviews per user), that each event has an equal amount of reviews (5 reviews per event), that each event has an equal amount of keywords (5 keywords per event), that there are 50 distinct regions in **region**, and that there are 1250 distinct postal codes in **region** (each associated with 4 regions),

The first query the consultant worries about returns the titles of events a specified user (parameter ?) reviewed in their own postal code. For this query, the consultant already wrote the following relational algebra query:

$$\pi_{\mathsf{E}.title}(\sigma_{\mathsf{U}.uid=?}(\sigma_{\mathsf{U}.uid=\mathsf{R}.user\wedge\mathsf{E}.eid=\mathsf{R}.event\wedge\mathsf{E}.postcode=\mathsf{U}.postcode}(\rho_U(\mathsf{user})\times\rho_E(\mathsf{event})\times\rho_R(\mathsf{review})))).$$

The consultant expects that this query will be evaluated via the following query execution plan:



To gain insight into the performance of this query, the consultant asked us to figure out how costly this evaluation is and to improve on this plan. To do so, proceed in the following steps:

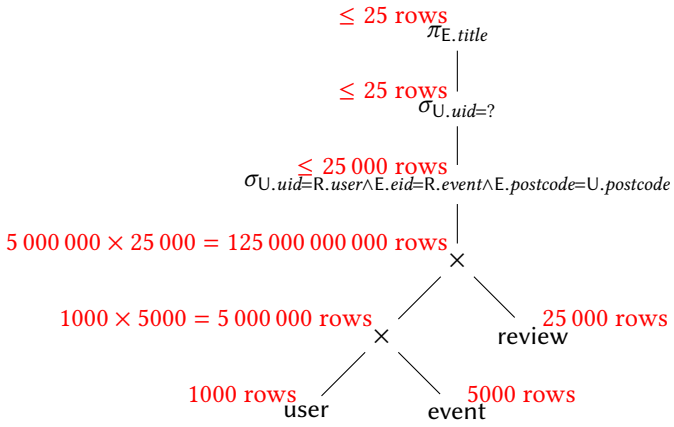
7. Estimate the size of the output of all intermediate steps in the query execution plan provided by the consultant.

Solution:

The estimates for the joins are straightforward, as all joins are cross products.

For the first selection ($\sigma_{\mathsf{U}.uid=\mathsf{R}.user\wedge\mathsf{E}.eid=\mathsf{R}.event\wedge\mathsf{E}.postcode=\mathsf{U}.postcode}$), we can use the fact that each user wrote 25 reviews. Furthermore, each review is for one event. Hence, the selection $\mathsf{U}.uid = \mathsf{R}.user \wedge \mathsf{E}.eid = \mathsf{R}.event$ will keep 25 rows per user (for a total of 25000 rows). We don't know anything about the geographical distribution of events and users (or the ratio between events in the postal code of users versus other events). Hence, we cannot use the condition $\mathsf{E}.postcode = \mathsf{U}.postcode$ to further reduce our estimate.

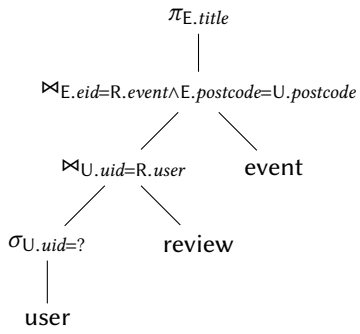
For the second selection ($\sigma_{\mathsf{U}.uid=?}$), we can use the fact that we keep only one user. Hence, we will keep at-most 25 rows. The final projection will keep these at-most 25 rows (unless there are events with duplicate titles, in which case we end up with fewer rows).



8. Provide a *good* query execution plan for the above relational algebra query. Explain why your plan is good.

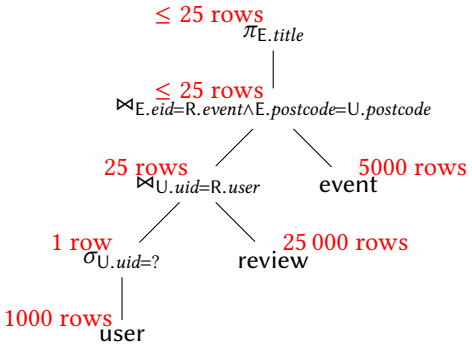
   Solution:

   We push down the selections as far as possible. Furthermore, we do not join **user** and **event** directly (as this is a Cartesian product due to the lack of common attributes to join on), but join **user** first with **review** and then join the result with **event**, this to make sure we only link up the **user** and **event** via their common review(s). We end up with the following query execution plan:



9. Estimate the size of the output of all intermediate steps in your query execution plan. Explain each step in your estimation.

   Solution:



   For the first selection ($\sigma_{U.uid=?}$), we can use the fact that we only keep one user (with a unique *uid*). Hence, this selection keeps 1 row.

   For the first natural join ($\bowtie_{U.uid=R.user}$), we use the fact that each user wrote 25 reviews. Hence, the join will produce 25 rows.

   For the second natural join ($\bowtie_{E.eid=R.event \wedge E.postcode=U.postcode}$), we use the fact that each review relates to one event, resulting in 25 rows that are considered during the join. We don't know anything about the geographical distribution of events and users (or the ratio between events in the postal code of users versus other events). Hence, we cannot use the condition E.*postcode* = U.*postcode* to further reduce our estimate, and we will keep at-most 25 rows.

   The final projection ($\pi_{E.title}$) will keep these at-most 25 rows (unless there are events with duplicate titles, in which case we end up with fewer rows).

10. Finally, also provide an SQL query equivalent to the original relational algebra query.

Solution:

```
SELECT E.title
FROM user U, event E, review R
WHERE U.uid = R.user AND E.eid = R.event AND
        E.postcode = U.postcode AND U.uid = ?;
```

The second query the consultant worries about is the following SQL query that obtains the *keywords* of events a specified user (parameter $?_1$) has reviewed in a specified region (parameter $?_2$):

```
SELECT DISTINCT K.word
FROM user U, review Rv, keyword K
WHERE U.uid = ?1 AND
        Rv.user = U.uid AND
        Rv.event = K.event AND
        K.event IN (SELECT E.eid
                    FROM event E, region Rg
                    WHERE E.postcode = Rg.postcode AND
                            Rg.name = ?2);
```

Unfortunately, the consultant was not convinced whether this query is optimal and could be executed efficiently. To gain some insight into the performance of this query, the consultant asked us to figure out how this query could be evaluated in practice. To do so, proceed in the following steps:

11. Translate this SQL query into a relational algebra query (that uses only relation names, selections, projections, renamings, cross products, and conditional joins). You are allowed to simplify the query if you see ways to do so.

    Rational:

    The table **user** is not used: the user identifier U.*uid* must be the same as Rv.*user*. Hence, there is no point in involving the **user** table in the query. Next, we can translate the SQL query straightforwardly into the following *bad* relational algebra query:

    $$\pi_{\mathsf{K}.word}(\sigma_{\mathsf{Rv}.user=?_1 \wedge \mathsf{Rg}.name=?_2}(\sigma_{\mathsf{Rv}.event=\mathsf{K}.event \wedge \mathsf{K}.event=\mathsf{E}.eid \wedge \mathsf{E}.postcode=\mathsf{Rg}.postcode}(\rho_{\mathsf{Rv}}(\mathsf{review}) \times \rho_{\mathsf{K}}(\mathsf{keyword}) \times \rho_{\mathsf{E}}(\mathsf{event}) \times \rho_{\mathsf{Rg}}(\mathsf{region})))).$$

    To turn this query into a *good* query, we push down the selections as far as possible. We also rearrange the joins to *only* look up keywords after we found all relevant events (and filtered away all non-relevant events) and to only join tables with common attributes (to prevent Cartesian products). As the first step of doing so, we filter the **review** table based on the provided user identifier and join each resulting review with its event:

    $$X := \rho_{\mathsf{Rv}}(\sigma_{user=?_1}(\mathsf{review})) \bowtie_{\mathsf{Rv}.event=\mathsf{E}.eid} \rho_{\mathsf{E}}(\mathsf{event})$$

    With the relevant reviews and their events in hand, we can join each event with the regions in which that event is organized. As we are only interested in a single region, we first filter the **region** table based on the provided region name:

    $$Y := X \bowtie_{\mathsf{E}.postcode=\mathsf{Rg}.postcode} \rho_{\mathsf{Rg}}(\sigma_{name=?_2}(\mathsf{region}))$$

    We note that the above join could be done with a *left semi-join*: we only need the event information from the **event** table. Finally, we join the resulting events with the **keyword** table to obtain their keywords and output these keywords:

    $$A := \pi_{\mathsf{K}.word}(Y \bowtie_{\mathsf{Rv}.event=\mathsf{E}.eid} \rho_{\mathsf{K}}(\mathsf{keyword})).$$

As we only need the keywords from the **keyword** table after the join, this last join could be done with a *right semi-join*.

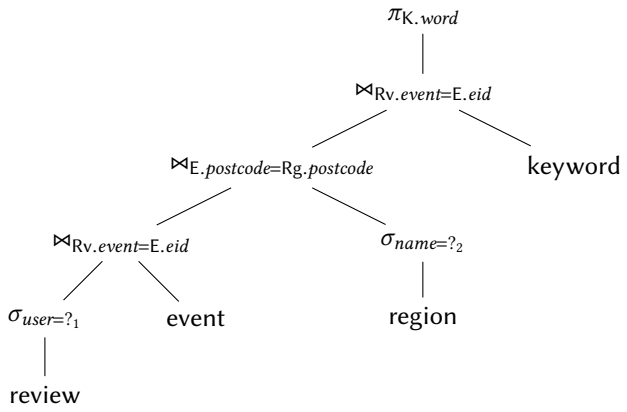
Solution:

$$X := \rho_{\mathsf{Rv}}(\sigma_{user=?_1}(\mathsf{review})) \bowtie_{\mathsf{Rv}.event=\mathsf{E}.eid} \rho_{\mathsf{E}}(\mathsf{event}),$$
$$Y := X \bowtie_{\mathsf{E}.postcode=\mathsf{Rg}.postcode} \rho_{\mathsf{Rg}}(\sigma_{name=?_2}(\mathsf{region})),$$
$$A := \pi_{\mathsf{K}.word}(Y \bowtie_{\mathsf{Rv}.event=\mathsf{E}.eid} \rho_{\mathsf{K}}(\mathsf{keyword})).$$

12. Provide a *good* query execution plan for your relational algebra query. Explain why your plan is good.

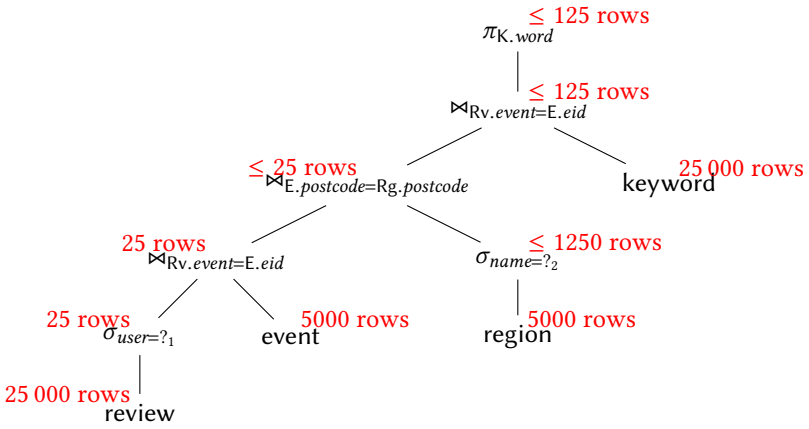
Solution:

The *good* query we provided already pushed down the selections as far as possible and dictated a join order in which all joins join tables via their common attributes and that allowed us to filter tables early during query evaluation. Hence, the query ends up up with the following query execution plan:



13. Estimate the size of the output of all intermediate steps in your query execution plan. Explain each step in your estimation.

Solution:



For the first selection ($\sigma_{user=?_1}$), we can use the fact that we only keep one user and the fact that each user wrote 25 reviews. Hence, the selection will keep 25 rows.

For the first natural join ($\bowtie_{\mathsf{Rv}.event=\mathsf{E}.eid}$), we note that each review pertains to exactly one event. Hence, the join will produce 25 rows.

For the second selection ($\sigma_{name=?_2}$), we can use the fact that there are only 1250 postal codes and, hence, that the selected region can have at-most 1250 postal codes. As we do not know how postal codes are distributed over the regions, we cannot further reduce our estimate.

For the second natural join ($\bowtie_{\mathsf{E}.postcode=\mathsf{Rg}.postcode}$), we note that each review pertains to exactly one event and that each event pertains to exactly one postal code. Hence, in the worst-case, we keep all 25 review-event pairs previously obtained. As we do not know which postal codes are kept during our selection of region, some events previously obtained can be eliminated, however. Hence, the join will produce at-most 25 rows.

For the third natural join ($\bowtie_{\mathsf{E}.postcode=\mathsf{Rg}.postcode}$), we note that each event has 5 keywords. As all previously obtained events must be distinct (as all reviews are by the same user), the join will produce at-most 125 rows.

The final projection ($\pi_{\mathsf{K}.word}$) will keep these at-most 125 rows (unless there are duplicate keywords, in which case we end up with fewer rows).