Project - Package Queries:
Efficient and scalable computation of high-order constraints

1. Project Description
Traditional database queries follow a simple model: they define constraints that each tuple in the result must satisfy. This model is computationally efficient, as the database system can evaluate the query conditions on each tuple individually. However, many practical, real-world problems require a collection of result tuples to satisfy constraints collectively, rather than individually. This paper presents package queries, a new query model that extends traditional database queries to handle complex constraints and preferences over answer sets.

2. Reading Material
Read paper from the beginning up to (and including) the entire Section 4 to understand the two algorithms: Direct and SketchRefine. Read the beginning of Section 6, Section 6.1, Section 6.2.1 (Query performance as dataset size increases) and 6.2.2 (Effect of varying partition size threshold) to understand the experimental evaluation methodology and the main scalability results. The rest of the paper is optional.

3. Algorithms to Implement
Implement the Direct method (Section 3.2) and SketchRefine with Greedy Backtracking (Algorithm 1 and Algorithm 2).

You can make the following simplifications:

- No need to implement the full PaQL parser. Your program can take as input five things:
  1. a table name,
  2. “MIN” or “MAX” for the objective,
  3. an attribute name $A_o$ for the objective function or None,
  4. a list of pairs of the form $(L_k, U_k)$, for each attribute $A_k$ of the table for which there is a constraint, and
  5. a pair $(L_c, U_c)$. The objective will be

    - MINIMIZE (or MAXIMIZE) $\text{SUM}(A_o)$ if $A_o$ is not None, or
    - MINIMIXE/MAXIMIZE COUNT(*) otherwise.

The meaning of $L_k$ and $U_k$ is that the package query will include a constraint $\text{SUM}(A_k)$ BETWEEN $L_k$ and $U_k$. $L_k$ or $U_k$ can be set to None to indicate a one-sided constraint. For example (None, $U_k$) indicates the constraint $\text{SUM}(A_k) \leq U_k$. All constraints are in conjunction (AND).
The meaning of \((L_c, U_c)\) is that the package query will include a “count constraint” of the form \(\text{COUNT}(\ast) \text{ BETWEEN } L_c \text{ AND } U_c\). Again, either \(L_c\), \(U_c\), or both can be None.

- Your implementation does not need to support the WHERE clause in your PaQL queries. Therefore, your implementation of Direct can skip the first step (Base relations).
- You can use any ILP solver of your choice. One easy solution is to use Python’s PuLP (https://pypi.org/project/PuLP/) with the default solver.
- Simplify the partitioning procedure: (1) You can do partitioning in memory (that is, you can bring the whole table from the DB into your program and do partitioning with your code). You can either implement partitioning yourself, or (better yet) use an existing library. (2) Your partitioning can be implemented with an existing clustering procedure, e.g., k-means.
- Simplify the creation of representative tuples: a representative can simply be the centroid of its group.

Important implementation aspects:

- Make sure your partitioning procedure puts together “similar” tuples.
- Always set a time limit for the execution of the ILP solver (e.g., 1 hour). If the time limit elapses, you can decide whether (1) you don’t report any result for that run or (2) you get the current best solution (if any) from the solver. Your final report should clearly state which option you picked.

Suggestions:

- For very large problems, the ILP solver may crash. Make sure that if the ILP solver crashes, your code does not crash. You can achieve that by running the ILP solver in a different process.
- Once you have a working implementation for Direct, reuse it inside of SketchRefine to solve the smaller problems.
- For some queries, in order to achieve feasibility, it may be necessary to implement “Hybrid Sketch”. Hybrid Sketch is explained in more detail in the Appendix.

4. Results to Produce

Dataset: TPCH table, provided by the course staff.
Package Queries: Package queries provided by the course staff.

Results to produce:

- **Fig. 7**: Run both Direct and SketchRefine on increasing data sizes. Compare running time and, when both produce results, show the approximation ratio of SketchRefine compared to Direct.
- **Fig 8**: Run SketchRefine with different partitioning settings (x-axis), by varying either the partition size limit (like in the paper) or the number of partitions (depending on your particular implementation of the partitioning procedure). Compare running time and approximation ratio of SketchRefine with Direct.
5. Ideas for Extension (20%)

This project also requires an extension. We offer a few ideas below. Please feel free to choose one idea to explore in your project. Students are also welcome to propose their own idea.

1) Compare the performance of SketchRefine using two different partitioning methodologies. For example, k-means clustering and k-dimensional quad trees (like in the paper). Is one better than the other?
2) Implement the PaQL parser and test it with a variety of package queries.
3) Play with the solver’s internal settings to make it run faster. For example, increase the “optimality gap”. If you run Direct with these new settings, how does it compare with SketchRefine that uses the old settings? And if you also run SketchRefine with the new settings, how do these algorithm all compare now?