

Query Consolidation: Interpreting a Set of Independent Queries Using a Multidatabase Architecture in the Reverse Direction

[Project to be known as Retroplex]

Aybar C. Acar and Amihai Motro

(submitted for publication)

Motivation

- Assume this scenario:
 - Person's *information goal* is satisfied by data stored in different databases.
 - Person submits a set of queries to these databases.
 - Then, off-line, consolidates the answers obtained in some “big answer”.
- By extracting this query set from the various logs, and analyzing it, we may be able to discover a *single interpretation* for the entire set, which (it is hoped) will correspond to the original information goal.
- This may suggest how the information should be *re-organized* to facilitate such tasks in the future.
- Recall that the original motivation for virtual databases is provide a single source for a particular task!

Analogy

- Consider a shopping center with multiple stores.
- Analysis of sales records shows that within *a small time interval*, the *same customer* purchased, in *different stores*:
 - A box of candy,
 - Gift-wrapping paper,
 - Greeting card.
- Conclusion: Customer service could be improved if all items were sold in the same store.

Another Possible Motivation

- In the analogy: What if the motivation for the individual purchases is that the customer is trying to *hide* his overall purpose?
- Hence, a possible application of query consolidation is *surveillance* and *security*:
 - A consolidated query discloses the *intentions* of the user posing the queries.

How Do Virtual Databases Fit Here?

- **Query decomposition** in virtual databases:
 - Given a global query Q , find
 - Local queries Q_1, \dots, Q_n
 - Assembly expression ESuch that $Q = E(Q_1, \dots, Q_n)$.
- We propose to adopt the same virtual database architecture, but use a process that is the *reverse* of query decomposition.
- **Query consolidation:**
 - Given local queries Q_1, \dots, Q_n , find
 - Global query Q
 - Assembly expression ESuch that the same query decomposition algorithm will decompose Q into Q_1, \dots, Q_n .
 - Therefore, $Q = E(Q_1, \dots, Q_n)$.

The Main Obstacle

- Query consolidation is a *function*:
 - Each query is mapped to a *unique* decomposition.
- But it is not *injective*:
 - There could be *multiple* global queries Q^1, \dots, Q^m ,
 - And corresponding expressions E^1, \dots, E^m ,
 - Such that Q^i will decompose to Q_1, \dots, Q_n using E^i ($I = 1, \dots, m$),

Our Approach

- We assume the independent databases to which queries are submitted had been incorporated into a virtual database system.
- We then “reverse” the query decomposition algorithm.
- Two steps could introduce multiple solutions:
 - At one point, the system must infer *equi-joins* for a set of relations.
 - At another point, it must discover the most likely *selection constraints* that would be applied to a relation.
- In each case we develop a procedure that *ranks* solutions according to their perceived likelihood.
- The final result is a *ranked* list of suggested consolidations.

The Virtual Database Architecture

- We adopt the Multiplex model:
 1. A global database scheme D .
 2. A set D_1, \dots, D_n of local database schemes and their associated instances d_1, \dots, d_n .
 3. A set $(V_1, U_1), \dots, (V_m, U_m)$ of view pairs where V_i is a view of D and each U_i is a view of some local database.
 4. Each view V_i is materialized by the corresponding view U_j .
- We assume queries and views are relational algebra expression with selections, projections and Cartesian products. These can be written:

$$Q = \pi_A \sigma_C (R_1 \times R_2 \times \dots \times R_n)$$

A Canonical Decomposition Algorithm

- A query Q is decomposed in 7 steps:
 1. Create a global relation scheme R for the Cartesian product operations in Q .
 2. Determine the views V_j that are relevant to Q (intersect with Q).
 3. Construct queries Q_i to retrieve from the corresponding local views U_j the parts that are relevant to Q .
 4. Evaluate Q_i in the local databases, obtaining answers A_i .
 5. Extend A_i with nulls, creating instances A_i^* of scheme R .
 6. Coalesce the instances A_i^* to a single instance A^* .
 7. Apply Q 's selection and projection operators, yielding an answer A to the query Q .

Sufficiency and Necessity Assumptions

- **Sufficiency:** The information in the local queries Q_1, \dots, Q_n is sufficient to accomplish the user's goal.
 - No outside information is used.
 - Hence, the goal can be approximated by a consolidation.
- **Necessity:** All the information in the local queries Q_1, \dots, Q_n is necessary to accomplish the user goal.
 - Nothing retrieved is discarded.

Duality

- Consider a global query that retrieves a person's age.
- Its decomposition could involve any of these (and more):
 1. Retrieve just the person's age.
 2. Retrieve the person's entire tuple, and then the expression E would project the age.
 3. Retrieve a group of persons (e.g., everybody), and then E will select the person's tuple and project the age.
- The guiding principle is to retrieve from the local database *as little as possible*:
 - Take advantage of the local database processing capabilities
 - Reduce costs of all sorts (charges by the local database, transmission).
- An *optimal decomposition* is one that minimizes the data transmitted.
- *Consolidation necessity* is similar to *decomposition optimality*:
 - Both assume that all information extracted from local databases is necessary:
 - Either to answer Q (decomposition) or to conclude Q (consolidation).

Methodology

- Reverse query decomposition with this procedure:
 1. For each local query Q_i , determine the views U_j that are relevant (intersect with Q_i).
 2. Process each answer A_i to obtain the part A_i^* that is within U_j .
 3. In the virtual database, materialize the corresponding views V_j with the answers A_i^* .
 4. Populate the relations R_k with the materialized views V_j .
 5. The global relations populated by at least one view must now be joined meaningfully.
 - If a view V_j is a join over two or more relations then a join is implied.
 - Hence, we have *clusters* with implied joins, that need to be joined.
 6. In the single relation thus obtained we keep only attributes that are in some view V_j .

Methodology (*Cont.*)

- We call the resulting relation R .
- The global (sought-after) query Q is now embedded in R .
- We now consider processing R by selection and projection.
- The necessity assumption implies that R should not be subjected to any selections based on *constants*.
 - These should have been done in local queries!
- It also implies that R should not be subjected to any *projections*.
 - These too should have been done in local queries!

Methodology (*Cont.*)

- The multiplicity of possible consolidations has two sources:
 1. The given relations can be joined in different ways.
 2. The relation R could be subjected to different *global* selections.
 - Selections that compare attributes retrieved from different local queries.
- We handle these issues consecutively:
 1. First, generate all the possible joins and rank them according to plausibility.
 2. Then for each join plan, suggest global selections, ranking them as well.
- These are explained in the “appendix”.

User Interface (mock)

The image shows a mock user interface for a 'Query Analysis' tool. It features a title bar with three colored window control buttons (red, yellow, green) and the text 'Query Analysis'. The main content is organized into sections: 'Input Queries', '3 Possible Consolidations', and a section for 'Possible Additional Comparisons'. Each query is listed with its attributes, source, and a 'Show Data' link. Consolidations are shown with a score and a 'Show Consolidation' link. Comparisons include confidence percentages and 'Apply' links.

Query Analysis

Input Queries:

Q1: {Name, Population, Capital, GDP} Source: nato.gov/members [Show Data](#)

Q2: {Name, Country, Longitude, Latitude, Pop.} Source: cityregistry.com [Show Data](#)

Q3: {Name, Country, Longitude, Latitude} Source: landmarks.org [Show Data](#)

3 Possible Consolidations:

▼ Q1(Capital) ⋈ (Name)Q2(Country) ⋈ (Country)Q3 Score: 92

Possible Additional Comparisons:

Q1.Population > 10 * Q2.Population (Conf.: **42%**) [Apply](#)

Q3.Longitude = Q2.Longitude AND Q2.Latitude = Q3.Latitude (Conf.: **15%**) [Apply](#)

[Show Consolidation](#)

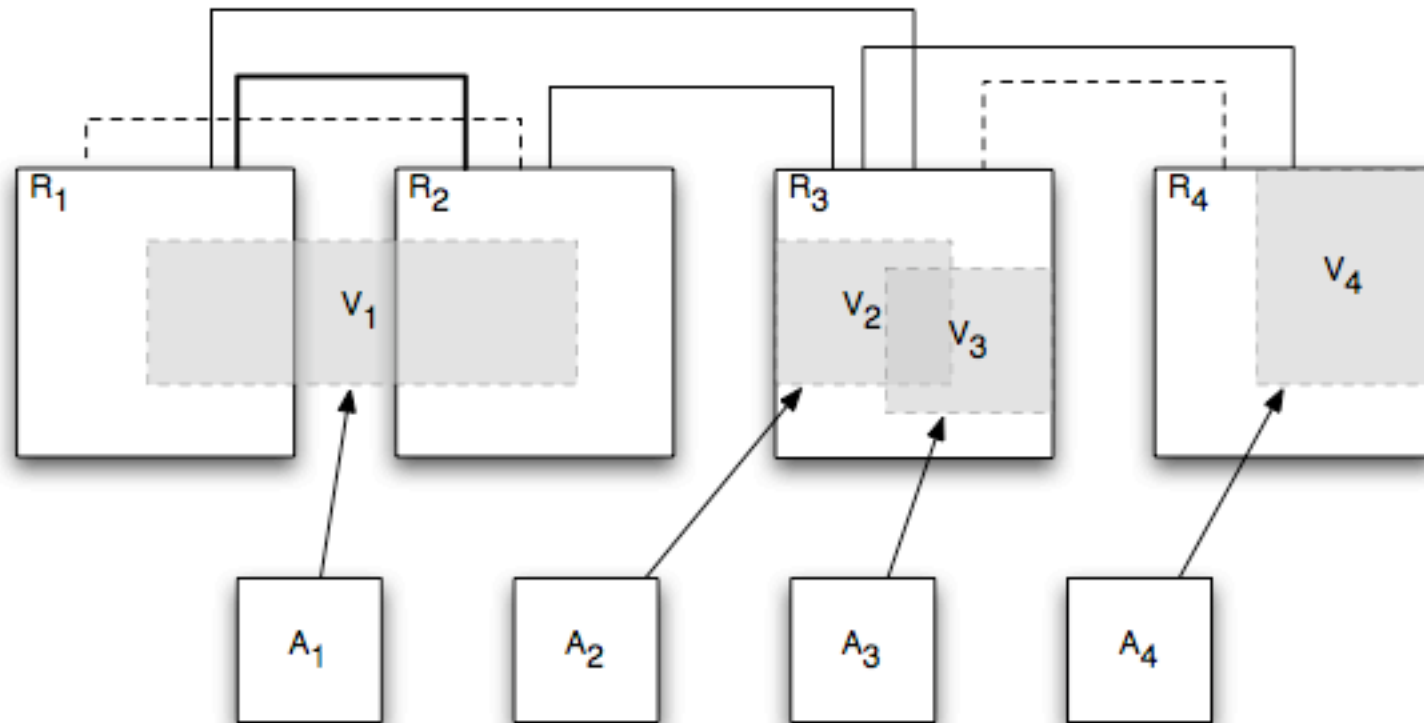
▶ Q1(Name) ⋈ (Country)Q2(Country) ⋈ (Country)Q3 Score: 84

▶ Q1(Name) ⋈ (Country)Q2(Name) ⋈ (Country)Q3 Score: 78

Conclusion

- We described an entirely new problem: *query consolidation*.
- This problem exhibits attractive duality with the well-known problem of *query decomposition*.
- We assumed the independent databases have been integrated into a virtual database system (and a global scheme exists).
 - A more challenging situation is when a global scheme does not exist.
 - The extensions of the given queries must be analyzed to infer their semantic associations (scheme-matching!).
- Other research issues:
 1. Cull from independent logs a set of queries that constitute a single task.
 2. Relax the assumptions of sufficiency and necessity
 - i.e., the information in the queries Q_1, \dots, Q_n is neither sound nor complete.
 3. Prefer *precise-but-complex* precise consolidations or *simple-but-rough*?
 4. Discover interpretations in *real-time*.

Appendix



- Alternative Join Path
- - - - Discarded Join Path
- Locked Join Path

Appendix

