Annotated Relational Algebra System

Krishnan Krishnamoorthy
40089054
Department of Computer Science and Software Engineering(CSSE),
Concordia University
Montreal,Canada.

Jayaprakash
40083709
Department of Computer Science and Software Engineering(CSSE),
Concordia University
Montreal,Canada.

Ezhilmani Aakaash
40071067
Department of Computer Science and Software Engineering(CSSE),
Concordia University
Montreal,Canada.

***Abstract* —We show the Annotated Relational Algebra System that implements the work of Green et al [PODS 2007] on provenance semirings. We do the annotated relational algebra calculations for Standard database, incomplete databases, probabilistic databases, bag databases, certainty databases and provenance databases are cases of the same general algorithms involving semirings over the query that contains projection π, select σ, union ∪, and natural join (Since these are the basic operations required to represent Datalog). Our implementation operates by transparently inputting SQL queries submitted by users to perform the annotation propagation, without requiring changes to the database implementation itself. This design allows our system to be easily integrated into any application, regardless of the underlying database engine.**

I. Introduction

Curated databases, which consist of data extracted from original sources, printed articles, and other databases, are a valuable source of data for scientists. However, as curated databases aggregate information from multiple sources, the origin of the data elements can be lost. Because of this, curated databases often provide support for data annotations, which are pieces of extra information added to elements by human curators in the data extraction and collection process, containing miscellaneous data such as the original data source. However, manually recording the origin of each data element is a tedious and error-prone task. Ideally, the system would provide an automated way to track data, or the movement of a data element from its origin to its final place of use.

To help address this problem,we have developed a system that performs computation on the annotated/tagged relations in a relational database. This is a very useful tool that helps propagate annotations as the rules are propagated along. Often, scholars are in the need to work or large datasets or have to derive various results that maybe based on current or old data. A time comes when the validity and correctness of the data that is being used is under question. In other words, provenance of the data needs to be verified. Provenance is quite important as it plays a major role in deciding the quality of the data. The term annotation refers to data of data or metadata of provenance that helps with the quality and other finding of the data.

The annotated relational algebraic calculator or ARAS that is developed provides a mechanism where the annotations are tagged to the tuples in the relational databases. This is propagated along when the data transformations occur. This can help with the verification of provenance of data be it old or current. If this is done to all the data throughout the database, then scholars or researchers can verify the correctness of their work based on the result and other factors.

Annotations can also help with providing data that is usually not provided in a traditional relational database. This includes security related data that is usually verified with the aggregated data over the past results or output. This can provide really sensitive data.

II. Literature Survey

Improv improves upon the AMS default algorithm for propagating annotations by adding support for join and aggregate operations. Their system succeeds in tracking the why-provenance of each cell in the database, although we find that why-provenance may not represent exactly the information that a user desires for certain operations, such as aggregation. The performance overhead of Improv for tracking provenance is significant. [1]

An Annotation Management System for Relational Databases have described an implementation of an annotation management system where different propagation schemes can be used. Insofar, a system developed by them only supports annotations on attributes of tuples. In their current system, annotations are propagated based on where-provenance [2]

A Probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems this paper provides a clear explanation of how probability database with annotation is calculated using the independence assumptions by keeping track of the basic events that contribute to a certain tuple of a relation, and thus all probabilities are computed correctly[3]

The authors have grouped the databases into types based on their usage, data and activity. Based on them and according to the expressive powers, types of possibilistic models are suggested for each. A comparison between these models and the typical certainty model is presented with pros and cons, each in their own right. The authors have also suggested the study of casualties in them but this can be currently viewed as ‘out of scope’ in terms of our implementation.[4]

Dr. Green's paper discusses the methodology of calculating the propagated rules as the annotated tuples travel along. It places emphasis on how the algorithms involving semirings are actually similar to those of the several annotated semantics. The paper puts forth set of semantics which includes bag, incomplete and probabilistic relational database. These are discussed in depth with comparison to different methodologies of solving it.An extension of this is to the datalog is discussed but is beyond the scope of this project. [5]

In all the above paper the calculations with annotations are strikingly similar as proposed by Dr.Green. A more efficient semiring of provenance was proposed to resolve the limitations of why and where provenance.

III. Design Ideas

Initially, we had plans of following the exact same procedure that the author had suggested and showed. This meant that the polynomial has to be computed at the very beginning and multiple versions of that had to follow. So, we decided to lay back on that as there will be high calculation overhead.

We also wanted to perform manipulations at the database side, MYSQL. Although seeming to be efficient, this was not the right way to proceed. This is because, the objective of the project was to manipulate the annotations at the backend (programming side). This was also the way Dr. Green has theorized.

Finally, we had decided to perform the manipulations at the backend (Python) side. This posed a few challenges that we were able to resolve making it a bit more efficient.

IV. Technology Used

Python:-For the sole purpose of simplicity and efficiency, Python was chosen. Starting with the database connectivity to the annotation manipulation, all of it was done using Python. Pandas were used to read the data block my block from the database and SQL connector to establish database connectivity.

MySQL:-Easy to use, lightweight DB with all of the needed functionalities. No manipulation was done at the DB Side.

XAMPP:-It is a cross-platform web server container that allows the user to deploy apache server and run scripts to connect to PHPmyadmin localhost. (DB UI)

V. Annotation Calculations

Each of the below mentioned semantics have been implemented in a way that is most apt and correct. This is because correctness was favored over efficiency during the initial phase.

A. Bag Semantics

For Bags, we have randomized digits between 1 to 100 and entered as annotation value for tuple in a table and put the whole table into the database.

Annotation calculation for Bag semantics was straight forward from [5]. For projection,union we sum the values of the annotation and for join we product the annotated tuple values.

B. Provenance Semantics

For Provenance, we have appended the character 't(i)'where i is the tuple number and enter into the database.

Annotation calculation for Provenance semantics was straight forward from [5]. For projection,union we use exclusive OR symbol as ('+')the values of the annotation for common tuples and for join we AND ('X') the annotated tuple values.

Here we would like to make a note that [5] calculated the annotation should have brackets "(" which makes the calculation correct. For example,reduction of the polynomial will be certainly wrong for polynomial as t1 x t1 + t2 x t2 if it is just calculated. It should be represented as (t1 x t1) + (t2 x t2).

C. Probability Semantics

For Probability, we have assigned a value between 0.0 to 1.0 randomly for a tuple. (2 Digits) and enter them into the database.

[3],[5] we got the idea of independence calculation for probability database and implemented the same. For Union,selection $1 - ((1 - \alpha) * (1 - \beta) *\ldots..)$ and for join we have used product of annotated tuple values.

D. CERTAINTY SEMANTICS

For Certainty,we have assigned a value between 0.0 to 1.0 randomly.

For arriving at possible rules for certainty semantics we had done a lot of study on other research papers as we weren't quite sure what was being proposed in the reference paper. For union calculation we had three variables in play namely min, Max and avg. We eliminate avg because it doesn't assume any extreme values which left us with min and Max. We proceeded with Max value for our calculation.

For join we product the annotated value as the value from the result of the join will be less than that of the two annotated value .

E. Standard Semantics

For Standard we insert the tuples with value annotated as 1.

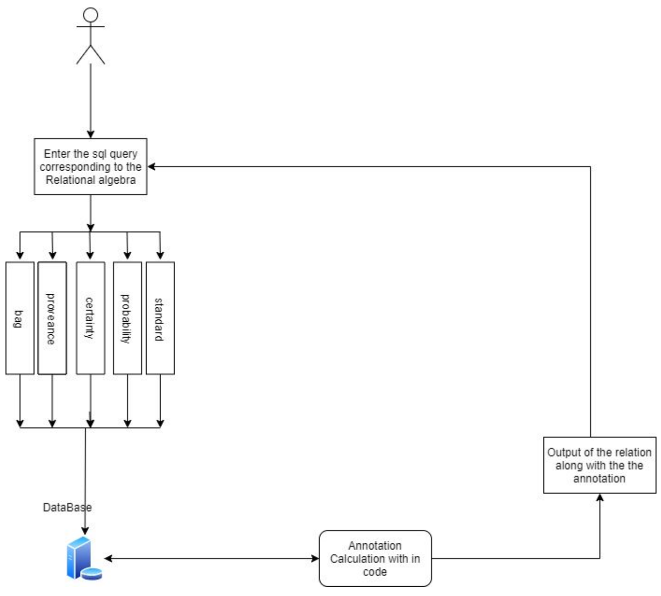
But if there are any value of zero in the database we query such that the annotations have value and run the query on them.

For union,projection we do Max of the annotated values and for join we multiply the annotated value of the tuple.

SEMANTICS	UNION AND PROJECTION	JOIN
BAGS	Summation or '+'	Product or '.'
PROBABILITY	1 – ((1 – α) * (1 – β) *…..)	Product or '.'
PROVENANCE	Disjunction or '+' (t followed by a digit)	Conjunction or 'X'
CERTAINTY	Max (α , β)	Product or '.'
STANDARD	Max (α , β)	Product or '.'

Table 1 Annotation calculation

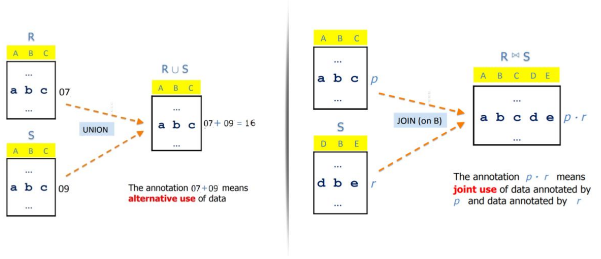
VI. System Architecture



Enter the sql query corresponding to the Relational algebra
bag
provenance
certainty
probability
standard
DataBase
Annotation Calculation with in code
Output of the relation along with the the annotation

fig 1 System Architecture

1. Initially, several brainstorming sessions were held to understand the concept behind why and how this was supposed to be implemented. Each of us had our own version of understanding and a way to go about it. Hence, the first two weeks were filled with research.
2. The next step was the selection of technologies. With a wide array of choice, choosing the most efficient or closest to being highly efficient was difficult. Java was the first choice based on commonality. Upon further inspection, 'Python' seemed to be more of an appropriate choice due to its versatility and ease of access.
3. Once the environment was setup, the next step was to implement the code. This required us to pick up a database technology and we chose, MySQL. The reason behind it was the fact that it's simple to use and each of us was aware of the usage.
4. Implementation of this project was more of a research and discuss strategy. This is because understanding of what was asked seemed more difficult than the actual coding.
5. In the backend, as mentioned before, Python was used. For the data and annotation manipulation, the Pandas framework or data structure was used. This provided a short and more efficient way to access and manipulate the required data. Since all of the manipulation was done at the backend, this seemed to be an apt choice.
6. For the SQL part, sql.connector was used to connect to MySQL and sql.execute to run the required SQL commands.
7. Once the understanding was done, each of us began to try our own at a specific semantic
8. The user will enter the sql query corresponding to the relational algebra query and select the semantic on which he chose to run the query.
9. After choosing the semantics the program will run the query one by one on the DB and provide an annotation column to the program which calculates the annotation based on the semantics and produce the results.
10. Bag was the first due to its simplicity and structure as the annotations were meant to be summed up or added. The thought of doing all of it at the database side was not pursued due to the time constraint and difficulty.



R
A B C
a b c
07
S
A B C
a b c
09
UNION
R ∪ S
A B C
a b c
07 + 09 = 16
The annotation 07 + 09 means alternative use of data
A B C
a b c
p
S
D B E
d b e
r
JOIN (on B)
R ⋈ S
A B C D E
a b c d e
p · r
The annotation p · r means joint use of data annotated by p and data annotated by r

fig 2 showing calculation of bag semantics

11. The above seen figure is an example of how the ‘Bag Semantic’ was solved for the project. This and the other remaining semantics were given a lot of thought before the solution criteria was chosen
12. Once the bare implementation of the code was successful, testing for the correctness of the code was set. This included multiple team discussions, various paper referrals and hands-on real time data check.

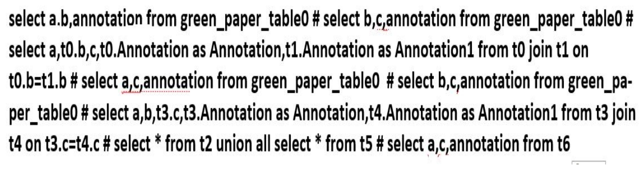
VII. Results And Discussion

A. Results

The query or the input for the implementation is no different from the usual SQL query format as no modification to the input part was made. Plans were made to get the input using pre or postfix method but was dropped due to lack of time and for the simpler access. This is because a query rewrite or parser requires extra implementation while serving not much of an upgrade. For test purposes, we have used the Dr. Green’s paper query(fig). The system runs all of the queries in a very short period with absolute completeness. The following tables were the inputs in the MySQL database stored prior to the manipulation. This was read from a file external to it.

$$q(R) \overset{\text{def}}{=} \pi_{\text{AC}}\left(\pi_{\text{AB}}R \bowtie \pi_{\text{BC}}R \cup \pi_{\text{AC}}R \bowtie \pi_{\text{BC}}R\right)$$

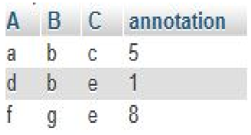
our input to the system would be Q1)



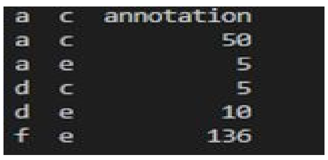
```
select a.b,annotation from green_paper_table0 # select b,c,annotation from green_paper_table0 #
select a,t0.b,c,t0.Annotation as Annotation,t1.Annotation as Annotation1 from t0 join t1 on
t0.b=t1.b # select a,c,annotation from green_paper_table0  # select b,c,annotation from green_pa-
per_table0 # select a,b,t3.c,t3.Annotation as Annotation,t4.Annotation as Annotation1 from t3 join
t4 on t3.c=t4.c # select * from t2 union all select * from t5 # select a,c,annotation from t6
```

Even though the query is big it was never a constraints mentioned to us in the problem as we were given freedom to provide the input to our wish but more on this part will be described on the difficulties/challenges faced and the future work.

The result of running the query is as follows:



A	B	C	annotation
a	b	c	5
d	b	e	1
f	g	e	8



```
a   c   annotation
a   c           50
a   e            5
d   c            5
d   e           10
f   e          136
```

More output is on Link :

https://drive.google.com/open?id=1xSlnj8lf60IJ7CBdPAj5X4LFN3_b2RZW

Q2) was to run on the production_procuct test csv provided to us

select product_id, product_name, brand_id,annotation from production_product1 # select brand_id, category_id, model_year, list_price,annotation from production_product1 # select product_id, product_name,t0.brand_id, category_id, model_year, list_price,t0.Annotation as Annotation,t1.Annotation as Annotation1 from t0 join t1 on t0.brand_id=t1.brand_id # select product_id, product_name,annotation from production_product1 # select product_id,brand_id, category_id, model_year, list_price,annotation from production_product1 # select t4.product_id, product_name,brand_id, category_id, model_year, list_price,t3.Annotation as Annotation,t4.Annotation as Annotation1 from t3 join t4 on t3.product_id=t4.product_id # select * from t2 union all select * from t5 # select product_id, product_name, brand_id,annotation from t6

Q3) was to run on the production_stocks test csv provided to us

select store_id,product_id,annotation from production_stocks1 # select product_id, quantity,annotation from production_stocks1 # select store_id,t0.product_id,quantity,t0.Annotation as Annotation,t1.Annotation as Annotation1 from t0 join t1 on t0.product_id=t1.product_id # select store_id,quantity,annotation from production_stocks1 # select product_id,quantity,annotation from production_stocks1 # select store_id,product_id,t3.quantity,t3.Annotation as Annotation,t4.Annotation as Annotation1 from t3 join t4 on t3.quantity=t4.quantity # select * from t2 union all select * from t5 # select product_id,quantity,annotation from t6

Q4) we ran the query on sales_customers.csv provided

select first_name,last_name,phone,email,annotation from sales_customer1 # select first_name,street,city,state,zip_code,annotation from sales_customer1 # select last_name,phone,email,street,city,state,zip_code,t0.first_name,t0.Annotation as Annotation,t1.Annotation as Annotation1 from t0 join t1 on t0.first_name=t1.first_name # select first_name,last_name,phone,email,city, annotation from sales_customer1 # select street,city,state,zip_code,annotation from sales_customer1 # select first_name,last_name,phone,email,street,state,zip_code,t3.city,t3.Annotation as Annotation,t4.Annotation as Annotation1 from t3 join t4 on t3.city=t4.city # select * from t2 union all select * from t5 # select first_name,last_name,city,state,annotation from t6

Q5) We ran the query on sales.order_items csv provided

select order_id,item_id,product_id,annotation from order_items1 # select product_id,quantity,list_price,discount,annotation from order_items1 # select order_id,item_id,t0.product_id,quantity,list_price,discount,t0.Annotation as Annotation,t1.Annotation as Annotation1 from t0 join t1 on t0.product_id=t1.product_id # select order_id,item_id,product_id,quantity,annotation from order_items1# select quantity,list_price,discount, annotation from order_items1 # select order_id,item_id,product_id,list_price,discount,t3.quantity,t3.Annotation as Annotation,t4.Annotation as Annotation1 from t3 join t4 on t3.quantity=t4.quantity # select * from t2 union all select * from t5 # select order_id,customer_id,order_status,staff_id,annotation from t6

Record Size	Tuplese Generated	Execution time
3(Q1)	5	5
322(Q2)	12133	12
940(Q3)	901	7
1446(Q4)	14298	16
4723(Q5)	>1000000	88

Table. 2. Performance on Query for the project.

Demo link :

https://drive.google.com/open?id=13I9ZIIrNC526fm2uAagkJlIoMgBxpiVJ

B. Performance And Discussion

The table 2 that portrays the performance and timings of our project implementation.

Points to Note:

1. Fast and Efficient/Correctness
2. Scalable to even large record size.
3. Coding strategy pattern

1. Fast and Efficient

Initial code was providing exponential result as shown in fig(4). As we increased the number of tuple we couldn't run the program. for example when we run query on test data set of 4723 records it was running for 20 minutes to complete the whole process.
we tried to run in main memory for all the queries but when tuples generated more than 100000(Q6) we got memory exception since having previous table stored in the main memory as well.

so we went with bulk insertion of records into the database and we found that by using *sqlalchemy* we were able to do bulk insert and performance was increased linearly fig(3) than exponential.

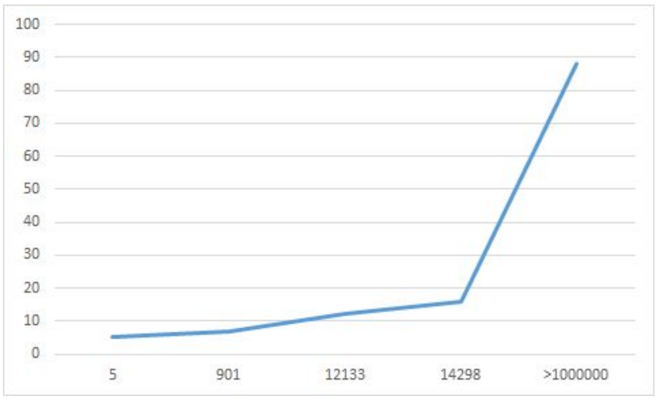


fig 3 shows the effective execution of query Q5 after optimization with x-axis as tuples generated and y-axis as execution time in seconds

2. Scalable to even large record size.

Due to our methodology of bulk insertion and a light weight database connectivity, scaling this implementation to a desirable large input should be of no trouble. This is because, we all of the operation is handled at the core part (backend).

3. Coding strategy pattern

With manipulations being done on the tables that are created based on the previous ones and temporarily stored, adding another operation or semantic is similar to adding another strategy.

This is because our style of coding was similar to a strategy pattern where any operation or semantic is an additional method or function call with the needed portion built right into it.

An operation such as ‘Set Difference’ can be easily implemented by the same way as bag or any other. By retrieving the tables from the DB and comparing it at the code side and storing the results.

Eg, A = {1,2,3,4}

B = {2,4}

A-B can be added in the current implementation by creating a method that compares the required columns alone. A panda data frame can act as a carrier and once the 2 blocks of data is retrieved, a simple comparison of those present in A and not in B can be easily accomplished.

VIII. Related Work

In the case of [5], a more 'compare and contrast' approach was used to display the pros and cons of few approaches. This includes the Maybe tables, C-Tables and Event tables. He seemed to be leaning towards the 'polynomial provenance' approach where a polynomial is formed which signifies the annotation propagation. Despite being a good approach from a theoretical standpoint, when it comes to the actually hands-on implementation, causes a lot of overhead as the polynomial needs to be calculated every single time to calculate the semantics annotations.

[1]Improv improves upon the AMS default algorithm for propagating annotations by adding support for join and aggregate operations. Although seeming to be efficient, this was not the right way to proceed. This is because, the objective of the project was to manipulate the annotations at the backend (programming side).

[4] With the authors suggesting a more model-type approach, the extension of c-tables and maybe tables have been proposed but with least interest. The possibilistic model that suggests the use of independent events or 1-P(A) produces proper results that are complete. A deeper step into this involves cleaning of the data before querying or propagation. This however is currently not implemented due to its complexity.

IX. Challenges Faced

There were many challenges faced by us during the development. We would like to list the most important things

A. SQL Parser/ Multiple Queries

we have partially implemented the SQL parser with the reference[6] but it was implemented in SQLlite instead of MariaDB and due to time constraints, the port from MariaDB to SQLLite was not successful.If we would have implemented initially with SQLlite we could have used radb library which would have taken care for the parser.

Embedding multiples queries in itself was a difficult task as simple queries may not always be the choice of the application user. Hence, we had to work on splitting the multiple queries and temporarily storing them until one was completed.

B. Calculation of different semantics

Despite an implementation that handles multiple queries, the quest did not stop there as to now the input query needed to be processed. Because if right order isn't followed, then the end result might be undesirable.

C. Simplify the polynomial

The simplification of the polynomial is one of the difficulties we encountered.At the early stage of the project, we're trying to implement the simplification algorithms, however, We have to call our process of simplification, which is a bit overwhelming. Since due to time constraints and speed is one the requirement we went with speed and not with polynomial reduction.

D. OPTIMIZATION

Initial code was providing exponential result as shown in fig(4). As we increased the number of tuple we couldn't run the program. for example when we run query on test data set of 4723 records it was running for 20 minutes to complete the whole process.Then we checked few easy and bulk insertion of records into the database and we found that by using *sqlalchemy* we were able to do bulk insert and performance was increased linearly as in fig (3) than exponential.

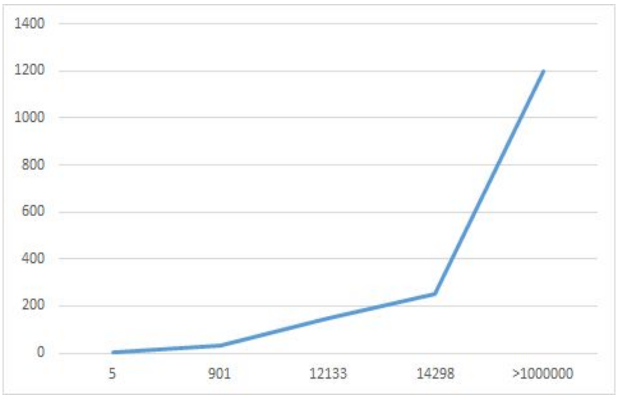


Fig. 3. Initial execution of query Q5 without optimization as on table 2. with x-axis as tuples generated and y-axis as execution time in seconds

X. CONCLUSION

Over the course of period, we have successfully implemented an annotated relational algebraic calculator that works on the required set of semantics and produces a desirable output that is complete and sound. Despite the lack of a sophisticated parser, the methodology adopted by us on the backend (Python) has provided us with the needed workarounds along with higher efficiency and a faster computation and less overhead compared to the computation of polynomial of the provenance and then substituting for each semantics. As mentioned already, this application has the ability to be extended for other semantics without causing any internals to go haywire due to its adapter pattern. Hence, a near-perfect implementation that checks all the boxes along with a few extras.

XI. Contribution

Apart from the below table, research of the posted papers and few external papers were done in union and discussed later on for knowledge transfer during brainstorming sessions and regular meetings.

Name	Contribution

Krishnan	Certainty ,Provenance Semantic,optimization,Report
Jayaprakash	Standard, Probability Semantic ,optimization,Report
Ezhilmani Aakaash	Bag Semantic,optimization,Report

Table 3

XII. Future Work

1. One of the major parts that we were partially working on was sql parser. This part of the implementation required extra time beyond that of the available. Hence, we strongly feel that if extra time was provided, these could have been added to refine the implementation.we also have a reference of how it can be implemented in[6]
2. other major part was polynomial reduction This part of the implementation required extra time beyond that of the available. Hence, we strongly feel that if extra time was provided, these could have been added to refine the implementation.
3. The current implementation of the code does not support other forms of data structure such as hierarchical XML type. With XML being used in a lot of fields, this could come in handy if implemented properly.
4. Support of negation for the union functionality could be quite the addition for the already complete implementation.
5. Performance and storage specs can be investigated on other platforms and architecture.

Acknowledgements We thank professor N.Shiri and TA Naghmeh and Raheel for helping us during the initial implementation of this system and for several helpful suggestions.

References

[1] https://courses.cs.washington.edu/courses/cse544/11wi/projects/hornyack_hunt.pdf.

[2] http://www.vldb.org/conf/2004/RS23P1.PDF.

[3] **N. Fuhr and T. R¨olleke**, A probabilistic relational algebra for the integration of information retrieval and database systems. TOIS, 14(1), 1997.

[4] **Olivier Pivert and Henri Prade** , Handling Uncertainty in Relational Databases with Possibility Theory - A Survey of Different Modelings

[5] https://users.encs.concordia.ca/~c65912/notes/Readings/green.pdf.

[6] https://users.cs.duke.edu/~junyang/radb/