# QUERY INVERSION TO FIND DATA PROVENANCE

Md. Salah Uddin[1], Dmitry V. Alexandrov[2], Armanur Rahman[3]

[1]National Research University Higher School of Economics (NRU HSE),
Faculty of Computer Science, School of Software Engineering,
Kochnovskiy Proezd 3,
125319, Moscow, Russian Federation
[2]National Research University Higher School of Economics (NRU HSE),
Faculty of Computer Science, School of Software Engineering,
Kochnovskiy Proezd 3,
125319, Moscow, Russian Federation;
Bauman Moscow State Technical University (Bauman MSTU),
Faculty of Engineering Business and Management,
Chair of Innovative Entrepreneurship, 2-ya Baumanskaya ul. 5,
105005, Moscow, Russian Federation
[3]BJIT Limited, Level-5, Road-2/C, Block-J, Baridhara, Dhaka-1212,
Bangladesh

***ABSTRACT***

*Day by day data is increasing, and most of the data stored in a database after manual transformations and derivations. Scientists can facilitate data intensive applications to study and understand the behaviour of a complex system. In a data intensive application, a scientific model facilitates raw data products to produce new data products and that data is collected from various sources such as physical, geological, environmental, chemical and biological etc. Based on the generated output, it is important to have the ability of tracing an output data product back to its source values if that particular output seems to have an unexpected value. Data provenance helps scientists to investigate the origin of an unexpected value. In this paper our aim is to find a reason behind the unexpected value from a database using query inversion and we are going to propose some hypothesis to make an inverse query for complex aggregation function and multiple relationship (join, set operation) function.*

***KEYWORDS***

*Data Provenance, Structured Query Language (SQL), Query Processing, Query Inversion.*

## 1. INTRODUCTION

In the database system domain: Data provenance, a kind of metadata, it is called lineage or pedigree which provides description of the origins of a piece of data and the process by which it arrived in a database. At present time, in different areas, such as e-science, data-warehousing etc. are required by origin of data to avoid unexpected value. To find data provenance the author has introduced different techniques such as GIS, VDL (Virtual Data Language), DB-Notes, BF05, SPG05a, SPG05b etc. But, the relation-ship between the data and its sources is very complex and

David C. Wyld et al. (Eds) : CNSA, DMDB, AIAP - 2018
pp. 17– 31, 2018. © CS & IT-CSCP 2018 DOI : 10.5121/csit.2018.80102

difficult to identify. So we want to use a kind of data provenance technology to automatically find out from where the unexpected data users were obtained from when users see the anomalous and suspicious data. To get better result we are going to introduce query inversion technique for some complex aggregation or multiple relationship function. Uses of the property by which some derivations can be inverted to find the input data supplied to them to derive the output data. Examples include, if an output of a database query Q applied on some source data D and given tuple is T then we want to understand which tuples in D contributed to get the output tuple T. A natural approach is to generate a new query Q0, determined by Q, D and T, such that when the query Q0 is applied to D, it generates a collection of input tuples that contributed to the output tuple T. In other words, we would like to identify the provenance by inverting the original query [10].

## 2. OVERVIEW OF EXISTING APPROACHES

Representing data provenance has two major approaches: annotation and inversion. Inversion approach is used to operate on an output data to find an input data. In area of data warehouses, Cui, Widom, and Wiener [27] first introduced the problem of relational database tracing data using query inversion. A disadvantage of this approach is that it cannot be used as sub-queries of normal relational queries and only partially benefit from the query optimization of the underlying Database Management System (DBMS). Another mechanism is called *where-provenance* [1]. Mainly, we use this technique for determining where annotations are propagated from. Boris Glavic et al. [28] also introduced a mechanism that is call Provenance Extension Relational Model (PERM). The PERM prototype supports provenance computation using Structured Query Language (SQL). But the disadvantage of this technique is that it does not work for correlated sub-queries.

One of the earlier definitions was given in the context of geographic information system (GIS). In GIS, data provenance is known as lineage which explicates the relationship among events and source data in generating the data product [1]. In the context of data-base systems, data provenance provides the description of how a data product is achieved through the transformation activities from its input data [2].

Annotation systems like DB-Notes [CTV05] and MONDRIAN [GKM05] is a common approach in life sciences [4] and it enables a user to annotate data item with an arbitrary number of notes which are normally propagated when annotated data is transformed.

VDL (Virtual Data Language) provides query and data definition facilities for the Chimera system [3] and it supports relational or object-oriented databases and SQL-like transformations.

The PReServ (Provenance Recording for Services) [GMM05, GJM+06a] approach uses a central provenance management service. It uses a common interface to enable different storage systems as a provenance store.

Trio is a recursive traversing lineage algorithm to achieve complete provenance of a par-ticular output tuple which introduces a new query language TriQL [5] to deal with uncer-tainty and lineage information.

## 3. QUERY INVERSION MECHANISM

A considerable research effort has been made by the database community to manage data provenance. Data provenance can be defined at different granularity levels such as relation or tuple. Furthermore, data provenance has been categorized based on the type of queries (e.g. why,

where, how) it can satisfy. Different techniques have been proposed to generate data provenance in the context of a database system. But we are using query inversion techniques to find out data provenance from relational and complex database system easiest and fastest way.

To find specific data from databases we used some query. But we do not know behind this query at the execution time compiler need to process some tuples to generate output. Sometimes due to those tuples we get some unexpected results. We can not find which tuple is responsible for this unexpected value. But following our query inversion technique we have become able to find the problem showing the original data (see figure 1). Example, we wanted to see the sum of salary as per job category.

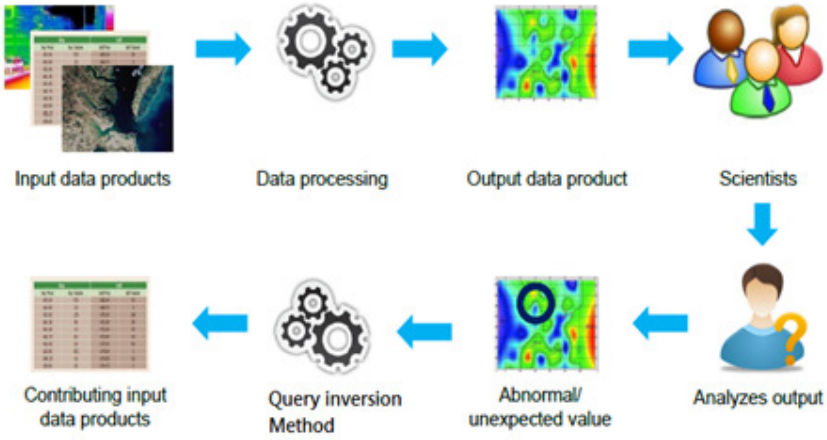


Figure 1. Data provenance technique overview.

**Code: select job,sum(salary) from emp group by job**

So we wrote above query to see the result, but it was showing an "**Invalid number**". Although our query is right but it was showing unexpected result because in our database there might be unwanted tuple. Due to that tuple this query is not working properly. Now we need to check all the values of the database to figure out the problem but at the present time the data is increasing and we have to process a huge amount of data daily. So it is not possible to check all the values and it is not time efficient. After analyzing the query we have developed an inverse query that is showing that unexpected tuple.

**Code: select job,salary from emp where job in (select job from emp group by job) (see figure 2).**

| JOB | SALARY |
|---|---|
| SALESMAN | - |
| SALESMAN | - |
| SALESMAN | - |
| SALESMAN | - |
| PRESIDENT | 20a000 |
| MANAGER | 10000 |
| MANAGER | 10000 |
| MANAGER | 15000 |
| ANALYST | - |
| ANALYST | - |

Figure 2. Showing the unexpected tuple using an inverse query.

## 3.1. Inverse Query hypothesis making technique

Inversion technique provides compact representation of provenance and this is the main advantage of this technique. But it is restricted to a certain class of multiple relationship queries and not universally applicable [26]. So in this section, our aim is to provide a hypothesis about possibility of developing of inverse query technique that can eliminate a limitation of multiple relationship queries.

When we need data manipulation we use some queries to get output but behind this output compiler need to process some tuples. After analyzing those tuples we have introduced some keys and extra tuples for our inverse query. Here we have described some complex aggregation and multiple relationship function in a generalized form of query and inverse query. We also have described the flow chart of the whole development procedure. In the flow chart we have to follow a life cycle to complete our inverse query (see figure 3, 4, 5). A Life cycle is a Black Arrow → Green Arrow → Red Arrow → Green Arrow.

### 3.1.1. Aggregation Functions

Oracle and other query languages support at least five aggregation functions such as min, max, count, sum, and avg. Aggregate functions are handled by adding parameter values to the group by list and adding the keyword condition for the aggregate column in the having clause instead of the where clause of the inverted query. For example if the general form query is:

**Code: select ΔH1, f(ΔH2) from r group by ΔH1 having <predicate>**

Then, the inverse query is as follows:

**Code: select ΔH1, ΔH2 from r where ΔH1 in (select ΔH1 from r group by ΔH1 having <predicate>**

Now, when adding keyword selections to the above keyword inverse query, any selections related to ΔH1 are added to the WHERE clause as before; however, selections relating to f(ΔH2) are added to a HAVING clause.
The relational algebraic translation of our above inverse query is:

$$\prod_{\Delta H1,\ \Delta H2,\ldots \Delta Hn}\left(\sigma_{p\wedge contains((\Delta H1,\ \Delta H2,\ldots\ \Delta Hn),k)}(r)\right) \cap \prod_{\Delta H1,\ \Delta H2,\ldots\ \Delta Hn}\left(\sigma_{p\wedge contains((\Delta H1,\ \Delta H2,\ldots\ \Delta Hn),k)>0}(r)\right)$$

Here ∏=select clause, σ=where clause for predicate, p=projection of attributes and r=table name.

**Example of complex Aggregation function:** Our generalized approach will work for all normal, complex and multiple relational aggregation functions.

**Code: select sum(e.sal), count(t.deptno) from emp e, dept t**

Now if we want to make an inverse query following our algorithm (see figure 3) for complex relationship, first of all we need to put *select* key then all attributes, *from* keyword and table name. After that according our algorithm, we need to put *where* keyword and first attribute without function attribute value, but here we can see that without function attribute there is no attribute, so we do not need to check the rest of the query after a table name. Our final inverse query will be as following:

**Code: select e.sal, t.deptno from emp e, dept t**

Following our generalized formula, it is possible to find data provenance for all simple, complex and multiple relationship aggregation functions. But for some correlated sub-queries our generalized formula does not work and it is the only limitation.

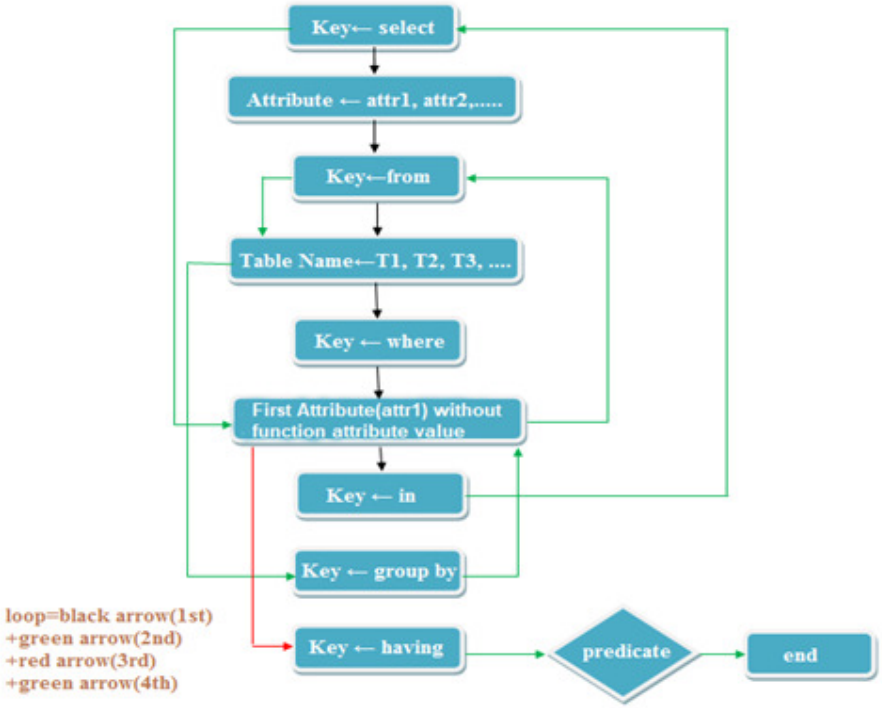


Figure 3. Inverse Query making procedure for aggregation functions.

### 3.1.2. Join Operation

The problem of join ordering is very restricted and at the same time it is a very complex one because it combines two or more tables in a relational database. At the compilation time if it has found any abnormal tuple it can not produce a result. So it is very difficult to find out the problem after searching multiple tables. For every tuple in the left input an output tuple must be produced for every tuple in the right input. A join operation can be implemented much more efficiently.
The approaches to handle the join operations are very similar to the ones for handling the aggregation function queries that are described in detail in the above section (3.1.1) but the difference is here we need to create a relationship between two or multiple tables. For example if the general form query is:

**Code: select ΔH1, f(ΔH2) from r1 natural join r2 group by ΔH1 having <predicate>**

Then, the inverse query is as follows:

**Code: select ΔH1, ΔH2 from r1 natural join r2 where ΔH1 in (select ΔH1 from r1 natural join r2 group by ΔH1 having <predicate>**

In this case we have to be more careful about the tuples in relationship to the different tables.
The relational algebraic translation of our above inverse query is:

$\prod_{\Delta H1,\ \Delta H2,\ldots \Delta Hn}( \sigma_{p\text{^}contains((\Delta H1,\ \Delta H2,\ldots\ \Delta Hn),k)}(r \bowtie r)) \cap \prod_{\Delta H1,\ \Delta H2,\ldots\ \Delta Hn}( \sigma_{p\text{^}contains((\Delta H1,\ \Delta H2,\ldots\ \Delta Hn),k)>0}\ (r \bowtie r))$

Here $\prod$=select clause, $\sigma$=where clause for predicate, p=projection of attributes, r=table name and $\bowtie$=join operation name.

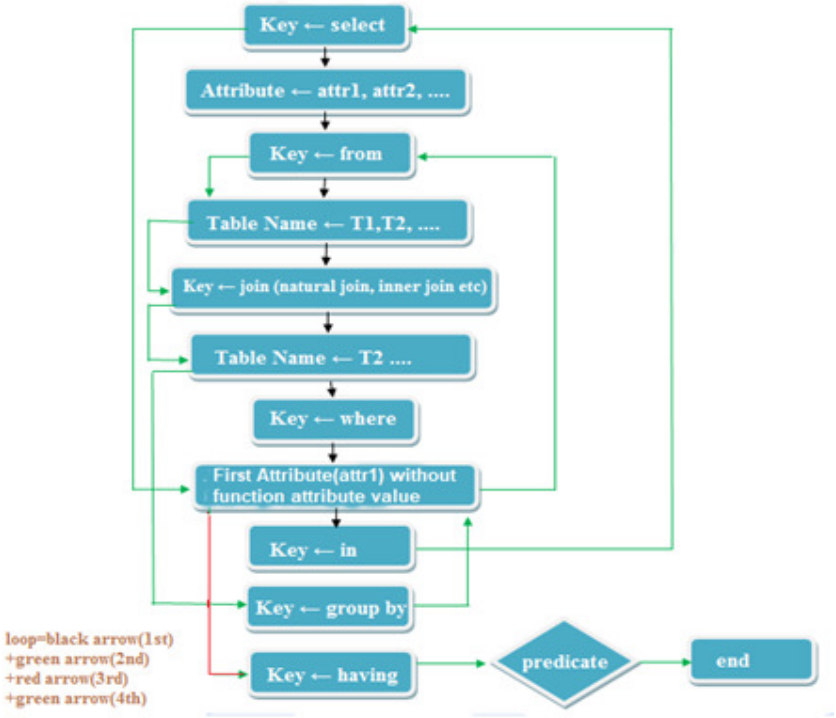


Figure 4. Inverse Query making procedure for join operation.

**Example of complex Join operation:** Our generalized approach will work for all normal, complex and multiple relational join operations.

**Code: select count(cus.cust_first_name), sum(ord.order_total), sum(pro.quantity) from demo_customers cus natural join demo_orders ord natural join demo_order_items pro**

Now if we want to make an inverse query following our algorithm (see figure 4) for multiple relationship, first of all we need to put *select* key then all attributes, *from* keyword and table name with *join* keyword. After that according our algorithm, we need to put *where* keyword and first attribute without function attribute value, but here we can see that without function attribute there is no attribute, so we do not need to check the rest of query after a table name. Our final inverse query will be as following:

**Code: select cus.cust_first_name,ord.order_total,pro.quantity from demo_customers cus natural join demo_orders ord natural join demo_order_items pro**

Following our generalized formula it is possible to find data provenance for all simple, complex and multiple relationship join operations. But for some correlated sub-queries our generalized formula does not work and it is the only limitation.

### 3.1.3. Set Operation

Set operation allows to be combined the results of multiple queries into a single result. Queries containing set operators are called compound queries. It includes union, intersect, minus operation in a database.

#### 3.1.3.1 Intersect

A Set Intersection can be handled by individually inverting a query with respect to each keyword and then taking a join of the inverted queries on the common parameters. Let's consider a general query:

**Code: select ΔH1 from r where <predicate1> intersect select ΔH1 from r where <predicate2>**

Then, the inverse query is as follows:

**Code: select distinct ΔH1, ΔH2, ΔH3….. from r where <predicate1> and ΔH1 in (select ΔH1 from r where <predicate2>**

The relational algebraic translation of our above inverse query is:

$\prod_{\Delta H1,\ \Delta H2,\ldots \Delta Hn}( \sigma_{p\text{^}contains((\Delta H1,\ \Delta H2,\ldots\ \Delta Hn)\ \cap k>0))}( r \bowtie r)) \cap \prod_{\Delta H1,\ \Delta H2,\ldots\ \Delta Hn}( \sigma_{p\text{^}contains((\Delta H1,\ \Delta H2,\ldots\ \Delta Hn),k)>0} (r \bowtie r))$

#### 3.1.3.2 Union

Handling the Union clause is complex mainly due to these reasons:

1. Each subquery involved in the Union may contain some of the keywords.

2. Each subquery in the Union may contain only a subset of the overall query parameters.

The approaches to handle the Union clause are very similar to the ones for handling the intersect queries and are described in detail in section (3.1.3.3). Let's consider a general query:

**Code: select ΔH1 from r where <predicate1> union select ΔH1 from r where <predicate2>**

Then, the inverse query is as follows:

**Code: select distinct ΔH1, ΔH2, ΔH3….. from r where <predicate1> or ΔH1 in (select ΔH1 from r where <predicate2>**

The relational algebraic translation of our above inverse query is:

$\prod_{\Delta H1,\ \Delta H2,\ldots \Delta Hn}( \sigma_{p\text{^}contains((\Delta H1,\ \Delta H2,\ldots\ \Delta Hn)\ \cup k>0))}( r \bowtie r)) \cap \prod_{\Delta H1,\ \Delta H2,\ldots\ \Delta Hn}( \sigma_{p\text{^}contains((\Delta H1,\ \Delta H2,\ldots\ \Delta Hn),k)>0} (r \bowtie r))$

#### 3.1.3.3 Minus/Except

The Minus/Except clause/operator is used to combine two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement. This means except/minus returns only rows, which are not available in the second SELECT statement.

Let's consider a general query:

**Code: select ΔH1 from r where <predicate1> minus select ΔH1 from r where <predicate2>**

Then, the inverse query is as follows:

**Code: select distinct ΔH1, ΔH2, ΔH3….. from r where <predicate1> and ΔH1 not in (select ΔH1 from r where <predicate2>**
The relational algebraic translation of our above inverse query is:

$\prod_{\Delta H1,\ \Delta H2,\ldots \Delta Hn}( \sigma_{p\text{^}contains((\Delta H1,\ \Delta H2,\ldots\ \Delta Hn)\ \cap k>0))}( r \bowtie r)) \neg\cap \prod_{\Delta H1,\ \Delta H2,\ldots\ \Delta Hn}( \sigma_{p\text{^}contains((\Delta H1,\ \Delta H2,\ldots\ \Delta Hn),k)>0}\ (r \bowtie r))$

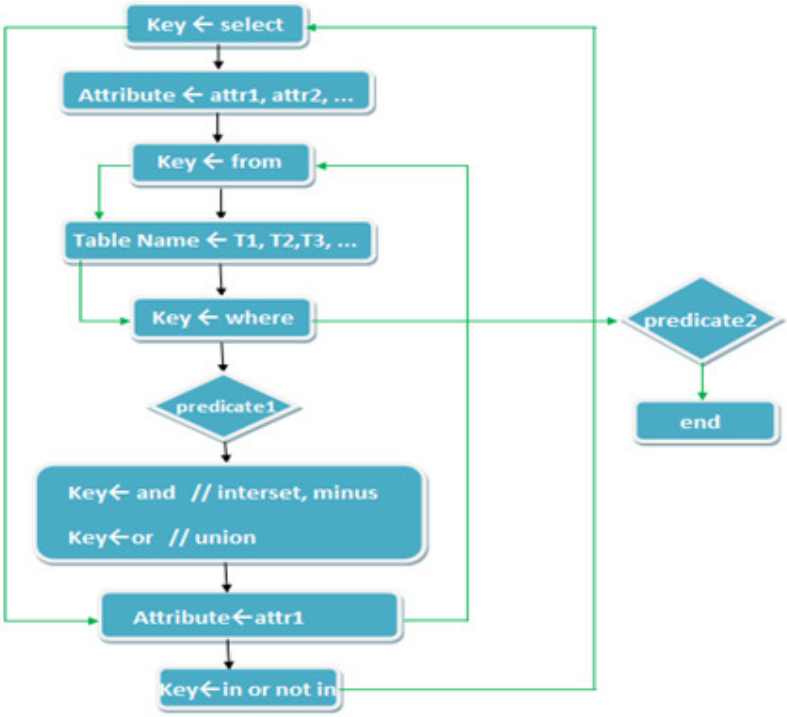


Figure 5. Inverse Query making procedure for set operation.

**Example of complex Set operation:** Our generalized approach will work for all normal, complex and multiple relational set operations.

**Code: select course_id from section where semester='Fall' and year=2009 intersect select course_id from section where semester='Spring' and year=2010 union select course_id from teaches where semester='Fall' and year=2009**

From above query we can see that there are two set operations: intersect and union. So following our algorithm (see figure 5), the inverse query will be:

**Code: select course_id,semester,year from section where semester='Fall' and year=2009 and course_id in (select course_id from section where semester='Spring' and year=2010) or course_id in ( select course_id from teaches where semester='Fall' and year=2009)**

Following our generalized formula it is possible to find data provenance for all simple, complex and multiple relationship set operations. But for some correlated sub-queries our generalized formula does not work and it is the only limitation.

### 3.2. Prototype Development Algorithm:

First of all we have separated all keys, tables, attributes and predicates from the main query then we have checked the query to find it,s aggregation, set operation or join operation. Finally our prototype dynamically set those values following the sequence of the previous flow chart (see figure 3, 4, 5).

**Algorithm 1** Find out attributes, keys, tables and predicates

```
input ← Query input
output ← Display result of attributes, keys, tables and predicates related input query
attribute_count ← 0
key_count ← 0
table_count ← 0
tokens ← split input query
for i = 0,i < tokens.length,i++
if tokens[i] == select or from or where or group or having or union or join
then
keys[key_count] ← tokens[k]
key_count++
Output ← allkeys.
end loop
if tokens(i) = select then
for j = i + 1,j >= 0,j++
    attributes[attribute_count] ← tokens[j]
attribute_count++
    if tokens(j) = from then
        i ← j − 1.
attribute_count−−
        Output ← allattributes.
end loop
        if tokens(i) = from then
for j = i + 1,j >= 0,j++
            tables[table_count] ← tokens[j]
table_count++
            if tokens(j) = where or group then
                i ← j − 1.
table_count−−
                Output ← alltables.
                if tokens(i) = where or having then
for j = i + 1,j >= 0,j++
predicate[0]+=tokens[j]
predicate[0]+=" "
                    if j == tokens.length-1 then
                        i ← j − 1.
                        Output ← allpredicates.
end loop
```

Figure 6. Algorithm for separating keys, attributes, tables and predicate.

## 4. EXPERIMENTAL RESULT

We have developed a prototype that is providing an inverse query for any given query. After that we have checked this inverse query is right or wrong in Oracle Database XE 11.2. We have written a query to get total summation of salary and bonus information the specific department name category field.

**Query: select dept_name,sum(bonus+salary) from instructor group by dept_name**
**Output: Invalid Number**

After that, now we want to know how to get the above results which tuples are working and which one is responsible for an unexpected result. Then we use our inverse query:

**Inverse Query: select dept_name,bonus,salary from instructor where dept_name in(select dept_name from instructor group by dept_name)**

| DEPT_NAME | BONUS | SALARY |
|---|---|---|
| Elec. Eng. | 10000 | 80000 |
| Physics | 10000 | 87000 |
| Physics | 25000 | 95000 |
| Comp. Sci. | 10000 | 92000 |
| Comp. Sci. | /2450 | 75000 |
| Comp. Sci. | 20000 | 65000 |
| Finance | 10000 | 80000 |
| Finance | 25000 | 90000 |
| Biology | 10000 | 72000 |
| Music | 3000 | 40000 |
| More than 10 rows available. Increase rows selector to view more rows. | | |

Figure 7. Output of inverse query for above aggregation function.

From our inverse query output **(see figure 7)** we can see that there is a tuple which contains an unexpected value. Due to this value our main query provides an unexpected result.
We tested for set operation and join query, we got accurate result to find data provenience.

**Query: select course_id from section where semester='Fall' and year=2009 intersect select course_id from section where semester='Spring' and year=2010 union select course_id from teaches where semester='Fall' and year=2009 minus select course_id from takes where semester='Fall' and year=2010 (see figure 8).**

| COURSE_ID |
|---|
| CS-101 |
| CS-347 |
| PHY-101 |

Figure 8. Output of above intersect query.

**Inverse query: select course_id,semester,year from section where semester='Fall' and year=2009 and course_id in (select course_id from section where semester='Spring' and year=2010) or course_id in ( select course_id from teaches where semester='Fall' and year=2009) and course_id not in (select course_id from takes where semester='Fall' and year=2010) (see figure 9).**

| COURSE_ID | SEMESTER | YEAR |
|---|---|---|
| CS-101 | Fall | 2009 |
| CS-101 | Spring | 2010 |
| CS-347 | Fall | 2009 |
| PHY-101 | Fall | 2009 |

Figure 9. Output of above intersect inverse query.

**Query: select cust_first_name,sum(order_total),sum(quantity) from demo_customers natural join demo_orders natural join demo_order_items group by cust_first_name (see figure 10).**

| CUST_FIRST_NAME | SUM(ORDER_TOTAL) | SUM(QUANTITY) |
|---|---|---|
| Eugene | 8280 | 34 |
| John | 23800 | 27 |
| William | 10390 | 27 |
| Edward | 12395 | 27 |
| Fiorello | 5450 | 16 |
| Albert | 4750 | 12 |
| Edward "Butch" | 4240 | 9 |

Figure 10. Output of above join query.

**Inverse Query: select cust_first_name,order_total,quantity from demo_customers natural join demo_orders natural join demo_order_items where cust_first_name in(select cust_first_name from demo_customers natural join demo_orders natural join demo_order_items group by cust_first_name) (see figure 11).**

| CUST_FIRST_NAME | ORDER_TOTAL | QUANTITY |
|---|---|---|
| Eugene | 1890 | 10 |
| Eugene | 1890 | 8 |
| Eugene | 1890 | 5 |
| John | 2380 | 3 |
| John | 2380 | 3 |
| John | 2380 | 3 |
| John | 2380 | 3 |
| John | 2380 | 3 |
| John | 2380 | 2 |
| John | 2380 | 2 |
| More than 10 rows available. Increase rows selector to view more rows. | | |

10 rows returned in 0.00 seconds Download

Figure 11. Output of above join inverse query.

Finally, we have become able to find the data provenance **(see figure 2, 7, 8, 9, 10, 11)** using our inverse query. Now if any user gets an abnormal or unexpected value, or they want to check behind their query, which tuples work, they can easily check by creating an inverse query.

## 4.1. Performance Evaluation:

To check performance of our new algorithm all experiments are performed on Intel core i3 machine with 4 GB ram and the size of our test database 10MB, 100MB and 500MB. For testing we have used three types of SQL query such as normal (Q1), complex (Q2) and multiple relationship (Q3) of four tables. To get execution time for each query we wrote “set statistics time

on" before our query and at the end of our query we also added "set statistics time off". We evaluated execution time of each query for aggregation function (see table 1), join operation (see table 2) and set operation (see table 3). Analyzing the experimental results of aggregation function (see table 1), we can see that in most of cases there is a little time execution difference between the main query and the inverse one. If we compare small (10MB) and large (500MB) datasets, the execution time of our inverse queries is not increasing too much.

Table 1. The Execution time for each query of Aggregation function.

| **Query** | **10MB** | | **100MB** | | **500MB** | |
|---|---|---|---|---|---|---|
| | **Query** | **Inverse query** | **Query** | **Inverse query** | **Query** | **Inverse query** |
| Q1 | 35 ms | 36 ms | 69 ms | 76 ms | 77 ms | 94 ms |
| Q2 | 66 ms | 69 ms | 149 ms | 157 ms | 158 ms | 169 ms |
| Q3 | 19 ms | 321 ms | 29 ms | 575 ms | 31 ms | 581 ms |

Table 2. The Execution time for each query of Join operation.

| **Query** | **10MB** | | **100MB** | | **500MB** | |
|---|---|---|---|---|---|---|
| | **Query** | **Inverse query** | **Query** | **Inverse query** | **Query** | **Inverse query** |
| Q1 | 49 ms | 70559 ms | 69 ms | 109513 ms | 77 ms | 250124 ms |
| Q2 | 52 ms | 88015 ms | 81 ms | 191092 ms | 98 ms | 398029 ms |
| Q3 | 55 ms | 90939 ms | 92 ms | 220306 ms | 123 ms | 502196 ms |

Table 3. The Execution time for each query of Set operation.

| **Query** | **10MB** | | **100MB** | | **500MB** | |
|---|---|---|---|---|---|---|
| | **Query** | **Inverse query** | **Query** | **Inverse query** | **Query** | **Inverse query** |
| Q1 | 3 ms | 9598 ms | 8 ms | 15598 ms | 11 ms | 18400 ms |
| Q2 | 2 ms | 8598 ms | 9 ms | 15101 ms | 12 ms | 18139 ms |
| Q3 | 3 ms | 9321 ms | 14 ms | 17004 ms | 19 ms | 21409 ms |

Execution time changes rather high from main query to inverse query for join and set operations. And if we compare the execution time of our inverse query for small (10MB) and large (500MB) datasets, the difference will be higher due to in this case a processor needs to process a huge dataset to provide the results.

## 5. CONCLUSION AND FUTURE WORK

Nowadays, provenance of data products is a widely-studied topic that attracts much attention of researchers. In this paper the main focus was to provide a guideline to find data provenance for unexpected values. To solve this problem we used an inverse query mechanism. Therefore, we showed that can easily find data provenance with the use of inverse queries. We proposed the

generalized forms that work for all types of normal, complex and multiple relationship queries except nested query. We presented also the execution times of our inverse queries for small and large datasets. Finally, we found that the execution time is not high for large datasets comparing to small datasets, and in most of the cases our solution is rather fast. Thus this technique can be used in different applications. For example, we generate business reports using different applications. If we suspect any errors in these reports, we have to check the related source datasets manually. Now this problem has been solved by using the proposed technique, as we can easily find the possible error in rather efficient way.

Since the proposed technique does not work for sub-queries, but only works for normal, complex and multiple relationship functions, the sub-queries go in focus of future research. Thus we plan to solve the sub-queries problem by improving the technique and develop a web prototype so that any user could generate the inverse query related his or her main query.

## ACKNOWLEDGEMENTS

We would like to thank all the colleagues at the School of Software Engineering of NRU HSE for their feedback and useful recommendations that contributed to bringing this paper to its final form.

## REFERENCES

[1] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In Proceedings of the International Conference on Database Theory, pages 316–330, 2001. (Cited on pages 2 and 21.)

[2] D. P. Lanter. Design of a lineage-based meta-data base for GIS.Cartography and Geographic Information Scence, 18(4):255–261, 1991.(Cited on pages 2 and 29.)

[3] Ian T. Foster, Jens-S. V¨ockler, Michael Wilde, and Yong Zhao. Chimera: A Vir-tual Data System for Representing, Querying, and Automating Data Derivation. In SSDBM'02: Proceedings of the 14th International Conference on Scientific and Statistical Database Management, pages 37–46, Washington, DC, USA, 2002. IEEE Computer Society.

[4] Laura Chiticariu, Wang-Chiew Tan, and Gaurav Vijayvargiya. DBNotes: a post-it system for relational databases based on provenance. In SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data, pages 942–944, New York, NY, USA, 2005. ACM Press.

[5] O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In Proceedings of the International Conference on Very Large Data Bases, pages 953–964, 2006. (Cited on page 22.)

[6] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. Nabar,T. Sugihara, and J. Widom. Trio: A system for data, uncertainty, and lineage. In Proceedings of the International Conference on Very Large DataBases, pages 1151–1154, 2006. (Cited on page 22.)

[7] M. K. Anand, S. Bowers, T. McPhillips, and B. Ludäscher. Efficient provenance storage over nested data collections. In Proceedings of the International Confe-rence on Extending Database Technology: Advances in Database Technology, pages 958–969. ACM, 2009. (Cited on page 20.)

[8] S. Davidson, , S. C. Boulakia, A. Eyal, B. Ludäscher, T. M. McPhillips, S. Bowers, M. K. Anand, and J. Freire. Provenance in scientific workflow systems. IEEE Data Engineering Bulletin, 30(4):44–50, 2007.(Cited on pages 15 and 17.)

[9] U. Park and J. Heidemann. Provenance in sensornet republishing. In Provenance and Annotation of Data and Processes, volume 5272 of LNCS, pages 280–292. Springer, 2008. (Cited on pages 26, 28, 37, 38, 111, 122,and 264.)

[10] M. R. Huq, P. M. G. Apers, and A. Wombacher. An Inference-based Framework to Manage Data Provenance in Geoscience Applications. Accepted in IEEE Transactions on Geoscience and Remote Sensing, Earlyaccess article DOI: 10.1109/TGRS.2013.2247769, IEEE Geoscience and Remote Sensing Society, 2013.

[11] C. Beeri, A. Eyal, S. Kamenkovich, and T. Milo. Querying business processes. In VLDB, 2006.

[12] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In VLDB, 2006.

[13] Grigoris Karvounarakis, Zachary G. Ives and Val Tannen: Querying Data Prove-nance. SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.

[14] Boris Glavic, Klaus Dittrich: Data Provenance: A Categorization of Existing Ap-proaches. SNF Swiss National Science Foundation: NFS SESAM

[15] Qi Yang. Computation of chain queries in distributed database systems.In Proc. of the ACM SIGMOD Conf. on Management of Data, pages 348-355

[16] L. Becker and R. H. G uting. Rule-based optimization and query processing in an extensible geometric database system. ACM Trans. on Database Systems (to appear)

[17] P. Bernstein, E. Wong, C. Reeve, and J. Rothnie. Query processing in a system for distributed databases (sdd-1). ACM Trans. on Database Systems, 6(4):603-625

[18] Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Chris Hayworth, Shubha Nabar, Tomoe Sugihara, and JenniferWidom. An Introduction to ULDBs and the Trio System. IEEE Data Engineering Bulletin, 29(1):5–16, 2006.

[19] Yingwei Cui, Jennifer Widom, and Janet L. Wiener. Tracing the lineage of view data in a warehousing environment. ACM Trans. Database Syst., 25(2):179–227, 2000.

[20] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. A survey of data prove-nance in e-science. SIGMOD Rec., 34(3):31–36, 2005.

[21] Paul Groth, Sheng Jiang, Simon Miles, Steve Munroe, Victor Tan, Sofia Tsasa-kou, and Luc Moreau. An Architecture for Provenance Systems — Executive Summary.Technical report, University of Southampton, February 2006.

[22] Ian T. Foster, Jens-S. V¨ockler, Michael Wilde, and Yong Zhao. Chimera: A Vir-tual Data System for Representing, Querying, and Automating Data Derivation. In SSDBM '02: Proceedings of the 14th International Conference on Scientific and Statistical Database Management, pages 37–46, Washington, DC, USA, 2002. IEEE Computer Society.

[23] Dennis P. Groth. Information Provenance and the Knowledge Rediscovery Prob-lem. In IV, pages 345–351. IEEE Computer Society, 2004.

[24] P. Yue, Z. Sun, J. Gong, L. Di, and X. Lu. A provenance framework for web geo-processing workflows. In Proceedings of the IEEE International Geoscience and Remote Sensing Symposium, pages 3811–3814. IEEE, 2011. (Cited on page 29.)

[25] M. R. Huq, A. Wombacher, A. Mileo. Data Provenance Inference in Logic Pro-gramming: Reducing Effort of Instance-driven Debugging.Technical Report TR-CTIT-13-11, Centre for Telematics and Information Technology, University of Twente, 2013.

[26] Bhagwat, L. Chiticariu, W. C. Tan, and G.Vijayvargiya, "An Annotation Management System for Relational Databases," in VLDB, 2004, pp. 900-911.

[27] Y. Cui, J. Widom, and J. Wiener. Tracing the Lineage of View Data in a Warehousing Environment. ACM Transactions on Database Systems (TODS), 25(2):179–227, 2000.

[28] Perm: Processing Provenance and Data on the same Data Model through Query Rewriting –Boris Glavic, Gustavo Alonso — 2009 — In ICDE '09: Proceedings of the 25th International Conference on Data Engineering

## AUTHORS

**Md Salah Uddin** is Master's student of the program "System and Software Engineering" at NRU HSE, Moscow, Russian Federation.
Research interests: Cloud Computing and Security, Big Data, Solid State Drivers, Databases and Neural networks.

**Dmitry V. Alexandrov** is Professor of the School of Software Engineering at NRU HSE, Moscow, Russian Federation; Professor of the Chair of Innovative Entrepreneurship at Bauman MSTU, Moscow, Russian Federation.
Research interests: Artificial Intelligence, Multi Agent Systems, Data Analysis, Databases, Mobile Applications Development

**Armanur Rahman** is Junior Software Engineer at BJIT Limited, Dhaka, Bangladesh. He has awarded his bachelor degree at East West University, Bangladesh.
Research interests: Data Mining, Databases and Neural networks.