Chapter 6

Template-Based Provenance Querying : Towards Designing a Provenance Query Language

6.1 Introduction

While Query Provenance in SQL and NoSQL (Graph and Key-Value Pair) databases, we felt the need for a universal provenance query language (PQL) which is independent of the underlying database model in which provenance data is stored. In Chapter 3, we considered two scenarios. In the first scenario, both the data & the provenance data was stored in RDBMS. And in the second scenario, the data was in the RDBMS, but the provenance data was stored in graph database. In Chapter 4, both the data & provenance data was stored in graph database. And in Chapter 5, Key-Value pair database was used to store both the data & provenance data. While working with queries on provenance data, stored in different data models, some common characteristics could be identified. Based on these common characteristics, we have categorized provenance queries under following two categories:

1. ***Provenance queries for query results***: Justifying results of a query that tell how any result is derived (Single-Depth or Multi-Depth provenance query from destination or derived result to source) and tracing all the results derived from particular

source at any time or within any specific time duration (Single-Depth or Multi-Depth provenance query from source to destination or derived result). Following types of provenance queries are identified in this category:

(a) ***Derivability of a result (Single-Depth or Multi-Depth provenance query)***: A user may want to visualize how any query result is derived, what are its direct sources, i.e., single-depth provenance query or what are its indirect sources upto a certain depth, i.e., multi-depth provenance query. This may help in knowing trustworthiness of data, investigation purpose, audit trail etc.

(b) ***To determine the relationship from particular source to any derived result (Single-Depth or Multi-Depth provenance query)***: User may be interested in knowing about what are the different query results which are derived from certain source for different purposes like auditing, erroneous source data etc.

(c) ***To determine the relationship from particular source within some time range to any derived result (Single-Depth or Multi-Depth provenance query)***: Further, user may also want to know the query results derived from certain source, but within a specified time range, to stop error propagation because of erroneous source data in the given time span.

(d) ***To determine the complete provenance graph of a query***: A user may want to visualize complete provenance graph of a query, i.e., all sources which are contributing towards result set and how they are contributing.

(e) ***To determine the complete provenance graph of a historical query, especially in a dynamic database***: It is very interesting to know about provenance graph of a query which has been executed in the past in a dynamic database where data is constantly changing. Because source data may have been updated after execution of a particular query. It becomes very challenging to know about source of any derived result in the past in a dynamic database.

2. ***Provenance queries for historical data***: Querying provenance to know about different versions of a data object, and instance of any data object any time in the past etc. Following provenance query is identified in this category:

(a) ***To determine the different versions of a data object***: One may also be interested in knowing different versions of any data object like current version, version anytime in the past, all versions till now since the value of data object exists or versions within a given time range. This helps in knowing about all updates which have been performed on any data object.

A few models are existing in literature for provenance storage and querying. However, issue of provenance querying has not been addressed much in an application and database independent way. In DBNotes [28], an annotation based provenance model, pSQL language is proposed to query data as well as provenance information in relational database. It provides limited support for annotation querying over single values only. In pSQL language, no provenance constructs are defined to access annotations directly, thus it is not feasible to issue queries directly over generated provenance information in the form of annotations. MONDRIAN, another annotation based provenance model [41] in relational database using color annotations, proposed its own provenance query language, CQL (Color Query Language) based on color algebra. The proposed color algebra is well-suited for annotations applied by the users, but few of the propagation rules in algebra are not suitable for provenance annotations specially in join queries. Further, Trio system that supports provenance and uncertainty in databases, also proposed a query language i.e. TriQL [31]. TriQL is based on SQL to query provenance and uncertainty in databases. Lineage(R,S) predicate is used to query provenance. TriQL query is required to be translated into various SQL queries and some user defined function calls because of separate lineage tables, which degrades its performance. pSQL, CQL, and TriQL all are applications specific and suitable for relational database only.

Further, ProQL (Provenance Query Language) [90] is proposed for querying derived data in an application independent way but designed specifically for RDBMS only. Another issue is time-aware provenance models and query their provenance information. Although, capturing information about updates as provenance information is very useful in querying historical data, tracing the source of derived result whose source is modified after its generation, but most of the existing models are considered "time" as an optional parameter for provenance information. In line with provenance of updates, VQuel (Ver-

sion Query language) [126] is designed for querying different versions of data objects but it is suitable for relational database only. TriQL [31] also supports for querying versioned data object but it is also suitable for relational database only as VQuel.

Another important observation is that many organizations have multiple databases. Almost 75% of organizations use a combination of SQL and NoSQL databases (refer to Figure 1.4 in Chapter 1). It would be very convenient if users can seamlessly query provenance in SQL and NoSQL databases using an abstracted provenance query language. A query language which can abstract the underlying database models from the users.

Thus, to bridge the identified gaps above, in this work, we propose various provenance query templates to query provenance information in all possible ways, independent of underlying database and application, such as querying historical data and querying provenance to justify a query result with varying depth. These provenance query templates will contribute towards designing of an efficient provenance query language. The proposed provenance query templates provide a standardised interface to support a variety of provenance queries. The notations of these query templates are simple, unambiguous, and easy to understand. All the functionalities are well defined and implemented in such a way that an application user can express his/her own provenance queries to obtain the required provenance information without the need to understand the underlying database model. In Figure 6.1, we present the provenance query engine for executing provenance query on top of any database. In this, user issues the provenance query in the form of proposed provenance query templates as an input and retrieves the correspond-

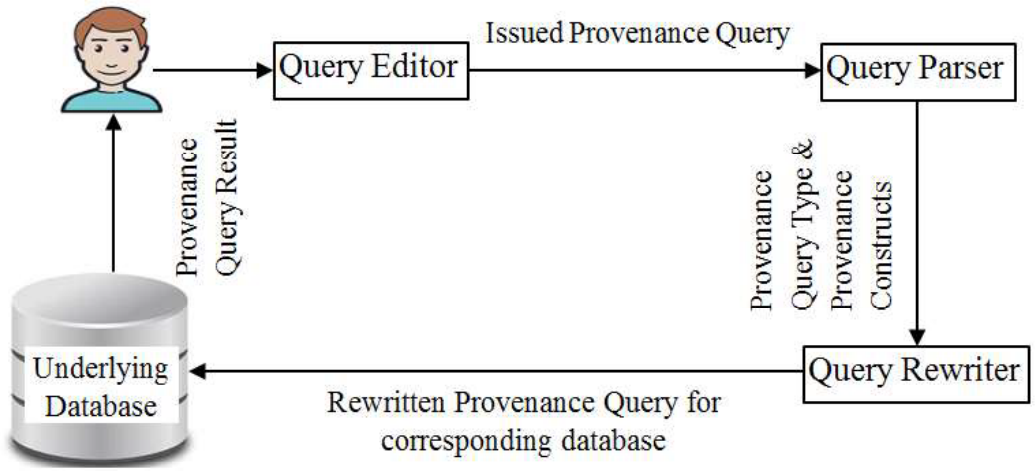

Figure 6.1: Provenance Query Engine

ing output. Initially, the user issues the provenance query via "*Query Editor*" and then issued provenance query in the form of provenance query template is passed to query parser. "*Query Parser*" parses the provenance query to obtain type of provenance query i.e. justifying query results or historical data query, and different constructs to know in detail about the provenance query i.e. forward tracing or backward tracing or depth of provenance query or time range of historical data etc. The output from query parser is then passed to query rewriter. "*Query Rewriter*" then rewrites a provenance query based on input received from query parser into a query language supported by the underlying database i.e. relational database, graph database, or key-value pair database. Finally, the rewritten provenance query executes on underlying database and returns the result back to the user.

Typically, query templates are extracted from a database query language. But, here we are doing it the other way round. The main motivation behind this is that designing and implementing a full fledged query language is a very tedious and long drawn process. Moreover, a new query language always has acceptability issue. Keeping this points in mind, a set of provenance query templates are proposed which cover a useful & practical range of provenance queries. These templates have been designed based on our experience of working with provenance queries on different database models and applications as part of the work done in Chapters 3, 4, & 5

In the previous chapters, we discussed a variety of provenance queries those are mostly executed on various database management systems such as relational, graph, and KVP databases. To design a flexible and dedicated provenance query language for efficient execution of a variety of provenance queries on a complex and very large size datasets, we categorized these provenance queries in two categories as defined in Section 6.1. Both categories of provenance queries can be combined further for expressing a wide variety of interesting queries on the provenance data.

6.2 Provenance Query Templates

In this section, we propose provenance query templates with the objective of facilitating the users to query provenance data in a database and application independent way. Before

that, we propose a vocabulary/keywords for the provenance query language.

6.2.1 Vocabulary of PQL (Provenance Query Language)

Source: A source is a database object. It could be a *cell*, *tuple*, *attribute*, *relation*, *user*, or *view* in relational database. In graph databases, it could be a *node*, *property* of a node, or a *link*. It may be a *row*, *column*, *column family*, or *keyspace* in a key-value pair (KVP) database. For example:

1. Determine the *cell* value from where the value is coming into result tuple (source - *cell* value of a tuple in relational database).

2. Determine all the *tuples* which are contributing to a specific result tuple (source - different *tuples* from different relations in relation database).

3. Determine all the *relations* which are accessed by any particular *user* (source - different *relations* and particular *user* in relational database).

4. Determine all the *tweets* posted with *hashtags* used by a particular *user* (source - *tweet* nodes, *hashtag* nodes, *link* between tweet and hashtag node, and *link* between user and tweet node in graph database).

5. Determine all the *columns* of specific *rows* in a *column family* belonging to particular *keyspace* which contributed towards a result row (source - *column*, *row*, *column family*, and *keyspace* in key-value pair database).

Result: It is the result of a query. It is a set of *tuples* in relational database, a *subgraph* or *attributes* in graph database, and *rows* in key-value pair database.

Destination: *Destination* is any *derived result* or *result*.

Attribute: Attribute is a specific *column* in relational and key-value pair database and *property/attribute* of a node in graph database.

Depth: Indicates the *depth* of a provenance query. The concept of depth is mainly related to graph database model as query can traverse up to any depth in a graph database. In the relational database model also, we can perform multi-depth queries, but we may need

to specify the depth of query in query itself so that self join with provenance table can be performed as many times. The default depth is 1. In key-value pair database, there is no concept of depth as there are no join operations. Example queries are given below:

1. Determine all the results directly or indirectly derived up to depth=2 from a particular source. (Here depth=2 indicates all the results directly derived (i.e. upto default depth=1) along with all the indirectly derived results (i.e. next level upto depth=2)

2. Determine all the tweets posted and retweeted by other users up to depth=2. (Here depth=2 indicate all the original tweets posted (i.e. upto default depth=1) along with the tweets further retweeted by other users (i.e. next level upto depth=2)

Time: It could be the current time, validity time of data, or time at which query was executed. It could also be the time interval of interest in a query.

Contributed: All sources which *contributed* towards any specific result, either directly (depth=1) or indirectly (depth >=2).

Version/Versions: All *versions* of a data object at any specific time or time interval.

Table 6.1 shows the main keywords with unique color codes. The color codes are used in later sections to emphasize their use/role in query templates.

S.No.	Keyword	Color Code
1	Source	Maroon
2	Destination	Purple
3	Result	Green
4	Attribute	Plum
5	Depth	Red
6	Contributed	Yellow
7	Version/Versions	Blue
8	valid_on	Pink
9	valid_from	Pink
10	valid_to	Pink
11	NOW	Pink

Table 6.1: Keywords with Color Code

S.No.	Query
Q1	Trace all the sources which directly contributed (Depth=1) to a specific query result.
Q2	Trace all the sources which directly or indirectly contributed (Depth=n) to a specific query result.
Q3	Trace all the results which were directly derived (Depth=1) from a particular source.
Q4	Trace all the results which were directly or indirectly derived (Depth=n) from a particular source.
Q5	Trace the direct contributions (Depth=1) of a particular source towards any result over a specific period of time.
Q6	Trace the direct or indirect (Depth=n) contributions of a particular source towards any result over a specific period of time.
Q7	Trace all the sources which directly or indirectly contributed (Depth=n) to a query executed at any particular time (historical query).
Q8	Determine all versions of a particular data object since its existence.
Q9	Determine all versions of a particular data object between any specific time period.
Q10	Determine the current version of a particular data object.
Q11	Determine the version of a particular data object at any specific time in the past.

Table 6.2: Example Provenance Queries in English

6.2.2 Example Provenance Queries in English

Table 6.2 shows possible example provenance queries of both categories viz., Provenance queries for query results, and Provenance queries for historical data.

Database Schema for provenance queries in relational database:

For provenance framework in relational database, we have used TPC-H Benchmark schema (as given in Chapter 3). For ready reference, the schema is reproduced here in Figure 6.2. In our corresponding Zero-Information Loss Relational Database (ZILRDB), every updatable column in TPC-H Benchmark schema is modelled as a nested table comprising of column value, valid_from, and valid_to fields representing the validity time of column value. Captured provenance information for further visualization is stored in provenance table and query table in relational database and graph database as well. Schema for provenance table (provtbl1) and query table (querytabletpch) in relational database are as follows:

provtbl1 *(Resultid, Provenance)*

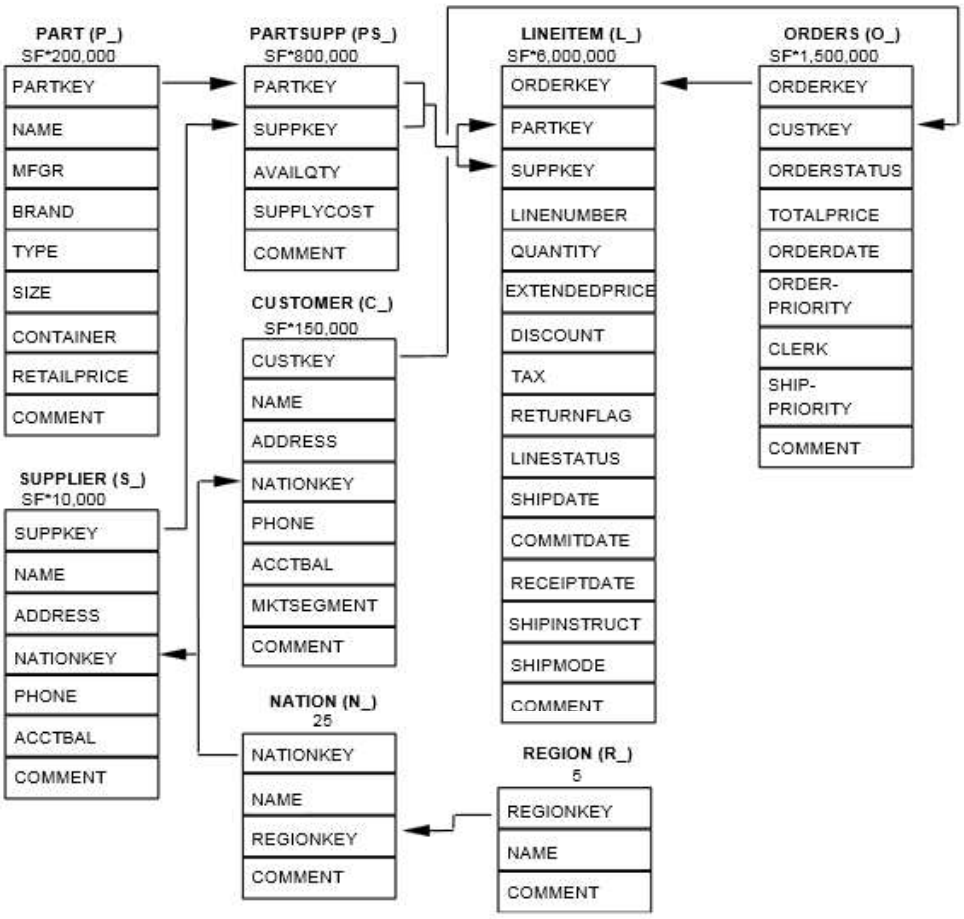

Figure 6.2: Relational Database Schema

querytabletpch *(QID, Query, Username, Validon)*

The corresponding graph structure contains Source tuple nodes, Result tuple nodes, Query nodes, and Operator nodes. Source tuple nodes are annotated with label "*table name*", and have two properties viz. "*tupleid*" and "*table name*". Result tuple nodes in provenance graph are annotated with label "*resultuple*", and have two properties viz. "*resultid*" as in relational database and "*query execution time*" that gives validity time of source tuples. Query nodes in graph are also annotated with label "*querytable*", and have following properties viz. "*QID*", "*query*", "*user*", and "*time*". Edges from source to result tuple via operator node signify the sources which are contributing in the generation of this result tuple and also explains how they are contributing. And edges from query node to result tuple nodes signify that these are the tuples which are generated from this query.

Database Schema for provenance queries in graph database:

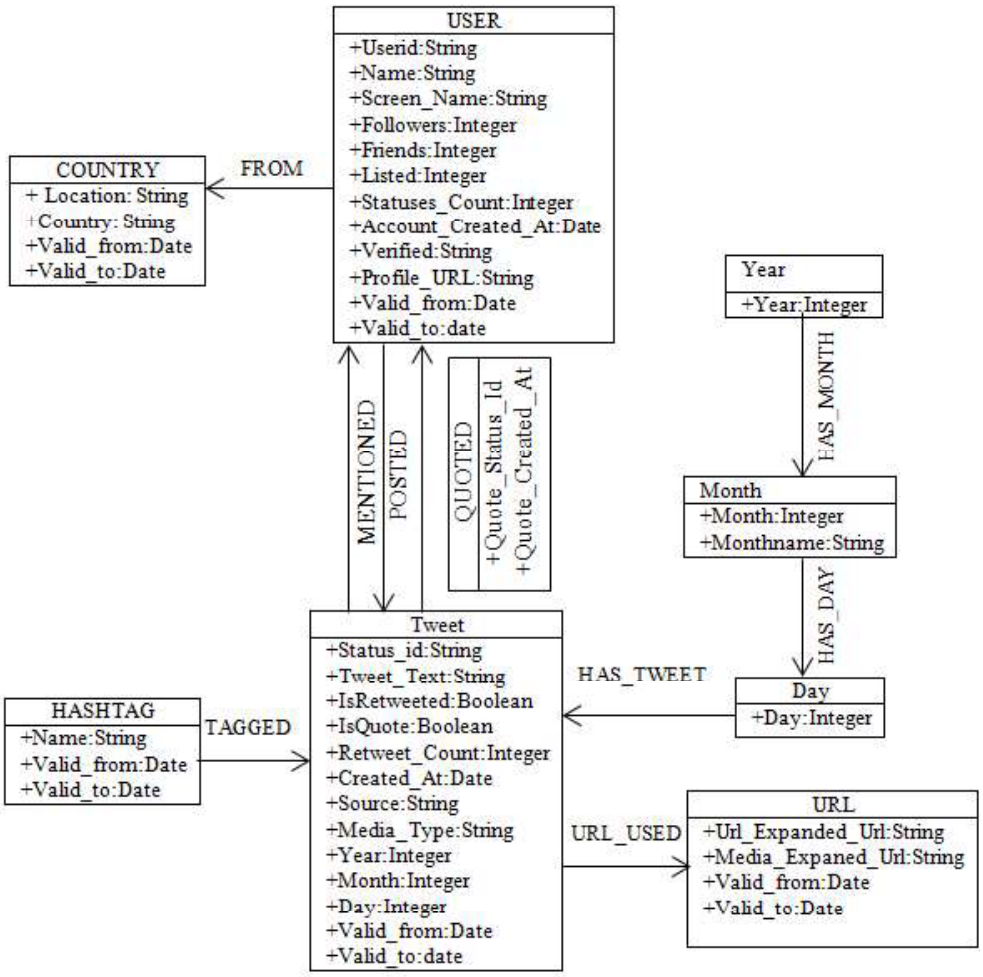

Figure 6.3: Graph Database Schema

For designing provenance framework in graph database, our Zero-Information Loss Graph Database (ZILGDB) contains various labelled nodes (USER, COUNTRY, TWEET, HASHTAG, URL, Day, Month, Year), properties of nodes, and relationships (FROM, MENTIONED, POSTED, QUOTED, TAGGED, URL_USED, HAS_TWEET, HAS_DAY, HAS_MONTH) between nodes, as shown in Figure 6.3

Our provenance graph contains nodes and relationships from source graph along with result tuple nodes (labelled as 'QUERYTUPLE') and query nodes (labelled as 'QUERY'). QUERY node is linked with QUERYTUPLE node via 'TUPLE' relationship. Further, each result tuple node is associated with all source nodes contributed towards it via 'provenance' relationship.

Database Schema for provenance queries in key-value pair database:

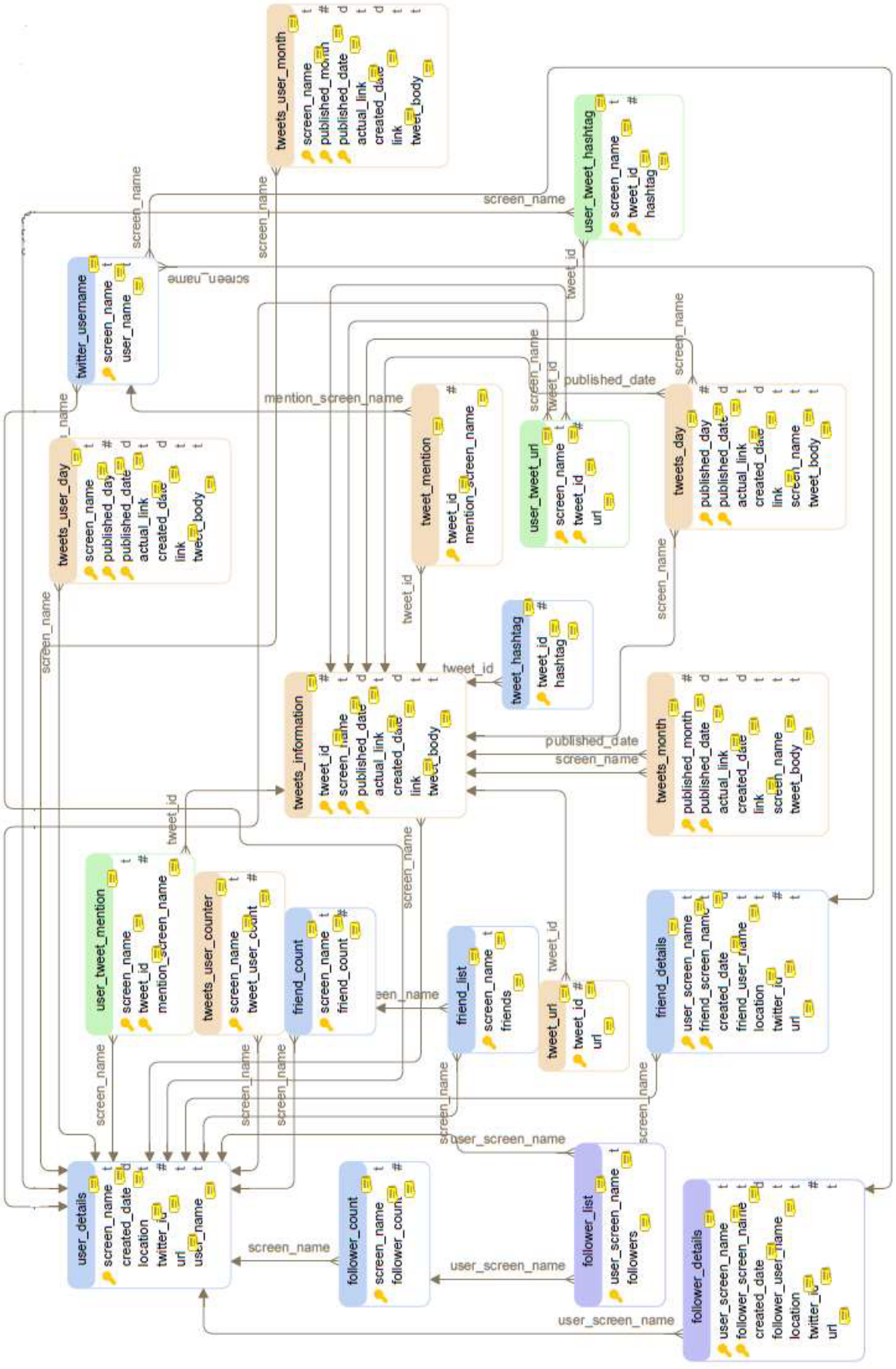

Figure 6.4: Key-Value Pair Database Schema

Our Zero-Information Loss Key-Value Pair Database (ZILKVD) contains a keyspace named "NewTwitter_Keyspace" that consists of 20 column families. The various column names of these column families with their row keys are shown in Figure 6.4.

Further, every row is added with valid_from and valid_to column as current date/time which indicates validity time of data whenever any insert or delete operation occurs.

Along with the column families mentioned in Figure 6.4, captured provenance information is stored in 3 column families viz. query_table (contains information about all queries executed), select_provenance (contains provenance information for select, aggregate queries), and update_provenance (contains provenance information for update queries) as follows:

query_table *(queryid, query, time)*

select_provenance *(queryid, resulttupleid, provenance_paths, query, time)*

update_provenance *(queryid, column_type, new_value, old_value, old_value_writetime, provenance_paths, query, rowkey, time)*

6.2.3 Provenance Query Template Design

Now we are proposing the two classes of provenance query templates as discussed in earlier section. First, we will write example provenance queries in all three databases considered viz., RDBMS, GDBMS, and KVPDB, and then we will give the corresponding provenance query template.

1. ***Provenance queries for query results***

   (a) ***How is any result derived? (Default Single-Depth (Depth=1) Provenance Query)***

   We may want to visualize the different ways a result can be derived—including all the sources. This is like the provenance graph projection, containing all the direct sources from which the result of interest is derivable, as well as their derivations.

   Example 1: Trace all the sources which directly contributed (Depth=1) to a specific query result.

   Example Provenance Query in Target Databases:

RDBMS: Trace all the sources directly contributing towards result tuple with id 'q1t1'.

(**source** - tuples of multiple relations and **result** - result tuple with id 'q1t1'.)

SQL Provenance Query: select provenance, validon, query, resultid from provtbl1 pt, querytabletpch qt where resultid = 'q1t1' and qt.qid= 'q1';

Result of above provenance query in RDBMS is shown in Figure 6.5. Here * denotes join operation and + denotes OR operation i.e. multiple derivations of result tuple q1t1.

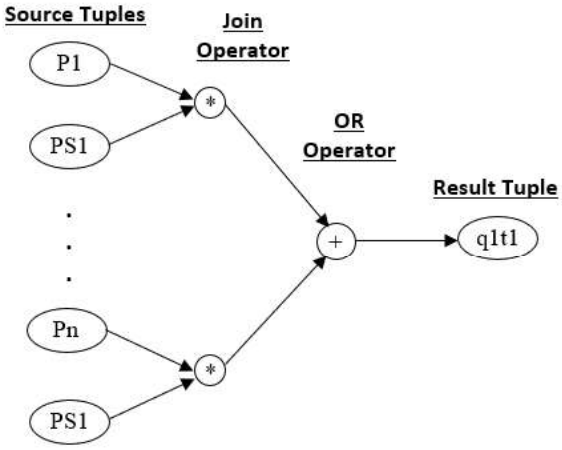

Figure 6.5: Provenance Graph in RDBMS

GDBMS: Trace all the sources directly contributing towards the result node with id 'q1t1'.

(**source** - nodes of source graph and **result** - result node with id 'q1t1'.)

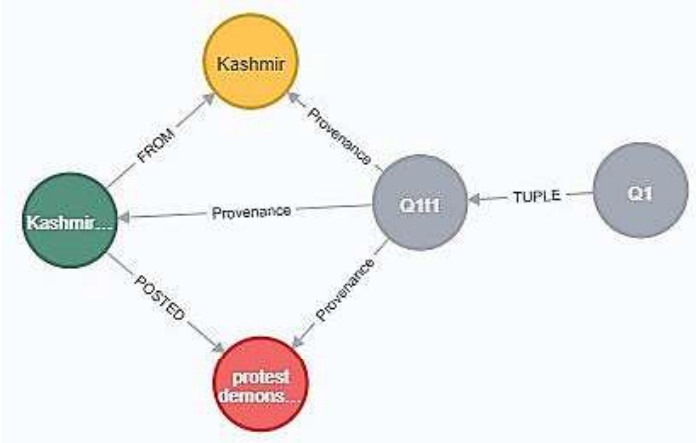

Figure 6.6: Provenance Graph in GDBMS

Cypher Provenance Query: MATCH (n:QUERY)-[t:TUPLE]->(n1:QUERYTUPLE)-[p:Provenance]->(a) where n.qid='q1' and n1.qtid = 'q1t1' return n,t,n1,p,a.

Result of above provenance query in GDBMS is shown in Figure 6.6, which shows three nodes of source graph i.e. User (Green Node) named "Kashmir-Cause_" from location "Kashmir" (Yellow Node) has posted the tweet (Red Node) have contributed to desired result tuple.

KVPDB: Trace all the sources directly contributing towards result row with id 'q1t1'.

(**source** - rows of column family and **result** - result row with id 'q1t1'.)

CQL Provenance Query: select provenance_paths, query, time from select_provenance where resulttupleid = 'q1t1';

Figure 6.7: Provenance Graph in KVPDB

Result of above provenance query is shown in Figure 6.7, which shows the keyspace, column family, all rows and corresponding column contributed to generate desired result.

Based on the above provenance query in different databases for explaining the derivability of a result with single depth, we propose the following provenance query template:

Provenance Query Template 1:

TRACE

Sources (S), **Query (Q)**, **Time (T)**

**Where Destination="$D"** and **Query_Id**="$QID"

RETURN S, Q, T, D, QID

The above provenance query template returns all sources which are directly contributing to given result with result id (D) along with how they are contributing. Result of provenance query also includes query (Q), time of execution (T) of query producing result (D) to retrieve the value of sources at that particular time as database is updatable.

(b) ***Knowing the sources of a derived result upto certain depth? (Multi-Depth Provenance Query)***

We may want to visualize the provenance of a query result with its direct as well as indirect sources to track complete history of derived result. Thus, we need to trace provenance of result upto required depth.

Example 2: Trace all the sources which directly or indirectly contributed (Depth = n) to a specific query result.

Example Provenance Query in Target Databases:

RDBMS: Trace all the sources which directly or indirectly contributed (Depth=2) to result tuple with id 'q1t2'.

(**source** - tuples of multiple relations and **result** - result tuple with id 'q1t2'.)

Provenance Query: select provenance from provtbl1 pt where resultid = 'q1t2' depth 2;

Here, direct source i.e. provenance polynomial can be retrieved from provtbl1. Provenance polynomial is then parsed for all source tuple ids at first level (depth 1). Afterwards, the query is automatically written to get source of all the tuple id's retrieved at depth 1 via self table join.

Rewritten SQL Provenance Query: (select provenance from provtbl1 where resultid = 'q1t2') union (select pt2.provenance from provtbl1 pt1, provtbl1 pt2 where pt1.resultid = 'q1t2' and pt1.provenance like '%' || pt2.resultid || ' %');

GDBMS: Trace all the sources which directly or indirectly contributed (Depth=2) to result node with id 'q1t2' and how they have contributed.

(**source** - nodes of source graph and **result** - result node with id 'q1t2'.)

Cypher Provenance Query: MATCH (n:QUERY)-[t:TUPLE]->(n1:QUERYTUPLE)-[p:provenance*1..2]->(a) where n.qid='Q1' and n1.qtid = 'Q1t2' return a, n1, p.

KVPDB: Currently, mutli-depth provenance query is not supported by proposed framework.

Based on the above provenance query for explaining derivability of a particular result with multi-depth, we propose the following provenance query template:

Provenance Query Template 2:

TRACE

Sources (S), **Query (Q)**, **Time (T)**

**Where Destination="$D"** and **Query_Id**="$QID"

With Depth="$n"

RETURN S, Q, T, D, QID

The above provenance query template returns all sources which are directly or indirectly contributing to a given result with result id (D) and depth of provenance query as n, along with how they are contributing. Result of provenance query also includes query (Q), time of execution (T) of query producing result (D) to retrieve the value of the sources at that particular time as database is updatable.

(c) ***How and where any source has contributed to derive some results? (Provenance query to track all derived results from a particular source)***

It is a kind of relationships between source and derived result. User may also be interested in restricting the set of derivations to those certain sources e.g., if that source is known to be authoritative or error in particular source (to explain the error propagation from source).

Example 3: Trace all the results which were directly derived (Depth=1) from a particular source.

Example Provenance Query in Target Databases:

RDBMS: Trace all the result tuples directly derived from the source tuple P1 of part relation.

(**source** - tuple P1 of "part" relation.)

SQL Provenance Query: select resultid, query, validon from provtbl1 pt, querytabletpch qt where SUBSTR(pt.resultid, 0, INSTR(pt.resultid, 't')-1) like qt.qid and provenance like '%P1%';

GDBMS: Trace all the result nodes derived from source node labelled "COUNTRY" with location attribute value as "Jammu and Kashmir".

(**source** - node of source graph labelled as "COUNTRY" with value of location attribute "Jammu and Kashmir".)

Cypher Provenance Query: MATCH (n:QUERY)-[t:TUPLE]->(n1:QUERYTUPLE)-[p:Provenance]->(co:COUNTRY{location: "Jammu and Kashmir"}) RETURN n, t, n1, p, co;

KVPDB: Trace all the result rows derived from source "tweets_user_month".

(**source** - Column Family named "tweets_user_month".)

CQL Provenance Query: select query, resulttupleid, time from select_provenance where provenance_paths LIKE '%tweets_user_family%' allow filtering;

Based on the above provenance query, we propose the following provenance query template:

Provenance Query Template 3:

TRACE

Destinations (D), Query (Q), Time (T)

Where Source="$S"

RETURN D, Q, T, S

The above provenance query template returns all results (D) directly derived from given source (S) along with the query statement and its time of execution.

Example 4: Trace all the results which were directly or indirectly derived (Depth=n) from a particular source.

Example Provenance Query in Target Databases:

RDBMS: Trace all the result tuples directly or indirectly derived (depth=2) from source tuple P1 of part relation.

(**source** - tuple P1 is of "part" relation.)

Provenance Query: select resultid from provtbl1 pt where provenance like '%P1%' depth 2;

Rewritten SQL Provenance Query: (select resultid from provtbl1 where provenance like '%P1%') union (select pt2.resultid from provtbl1 pt1, probtbl1 pt2 where pt1.provenance like '%P1%' and pt2.provenance like '%' || pt1.resultid || ' %');

GDBMS: Trace all the result nodes directly or indirectly derived (depth=2) from source node labelled "COUNTRY" with location attribute value as "Jammu and Kashmir".

(**source** - node of source graph labelled as "COUNTRY" with value of location attribute "Jammu and Kashmir".)

Cypher Provenance Query: MATCH (co:COUNTRY{location: "Jammu and Kashmir"}) <-[p:provenance*1..2]<-(n1: QUERYTUPLE) with co, p, n1 MATCH (n: QUERY)-[t:TUPLE]->(n1) return n, t, n1, p, co;

KVPDB: Currently, mutli-depth provenance query is not supported by proposed framework.

Based on the above provenance query, we propose the following provenance query template:

Provenance Query Template 4:

TRACE

Destinations (D), Query (Q), Time (T)

Where Source="$S"

With Depth="$n"

RETURN D, Q, T, S

The above provenance query template returns all results (D) directly or indirectly derived from a given source (S) upto a given depth (n) along with the query statement and its time of execution.

(d) ***Where any particular source, with updates within a specific duration, has contributed to some derived results? (Provenance tracing from source (with updates in specific duration in past) to derive results of historical queries)***

A kind of provenance for historical queries to check for authoritativeness, trustworthiness, auditing or error propagation.

Example 5: Trace the direct contributions (Depth=1) of a particular source towards any result over a specific time period.

Example Provenance Query in Target Databases:

RDBMS: Trace all the result tuples derived from source tuple P1 of part relation between '10/05/2015' to '15/05/2015'.

(**source** - tuple P1 of "part" relation between 10/05/2015 to 15/05/2015.)

SQL Provenance Query: select provenance, validon, query, resultid from provtbl1 pt, querytabletpch qt where SUBSTR(pt.resultid, 0, INSTR(pt.resultid, 't')-1) like qt.qid and provenance like '%P1%' and (qt.validon>=10/05/2015 and qt.validon<=15/05/2015);

GDBMS: Trace all the result nodes derived from source node labelled "USER" with screen_name "Anubhav" between '19/10/2019' to '21/10/2019'.

(**source** - node of source graph labelled as "USER" with value of screen_name

attribute "Anubhav" between 19/10/2019 to 21/10/2019.)

Cypher Provenance Query: MATCH (n:QUERY)-[t:TUPLE]->(n1:QUERYTUPLE)-[p:Provenance]->(u:USER) where u.screen_name= "Anubhav" and (n.time>= date('2019-19-10') and n.time<= date('2019-21-10')) return n, t, n1, p, u.

KVPDB: Trace all the result rows derived from source "tweets_user_month" between '18/10/2019' to '20/10/2019'.

(**source** - Column Family named "tweets_user_month" between 18/10/2019 to 20/10/2019.)

CQL Provenance Query: select query, resulttupleid, time from select_provenance where provenance_paths LIKE '%tweets_user_family%' and (time>=date(2019-10-18) and time<=date(2019-10-20)) allow filtering;

Based on the above provenance query, we propose the following provenance query template:

Provenance Query Template 5:

TRACE

Destinations (D), **Query (Q)**

Where Source="$S"**, **valid_from>="$T1" and valid_to<="$T2"

RETURN D, Q, S

The above provenance query template returns all results (D) directly derived from a given source (S) within a specific time duration along with the query statement that generates these results.

Example 6: Trace the direct or indirect (Depth=n) contributions of a particular source towards any result over a specific period of time.

Example Provenance Query in Target Databases:

RDBMS: Trace all the result tuples directly or indirectly derived (depth=2) from source tuple P1 of part relation between '10/05/2015' to '15/05/2015'.

(**source** - tuple P1 is of "part" relation between 10/05/2015 to 15/05/2015.)

Provenance Query: select provenance, resultid from provtbl1 pt, querytablet-pch qt where SUBSTR(pt.resultid, 0, INSTR(pt.resultid, 't')-1) like qt.qid and

provenance like '%P1%' and (qt.validon>= 10/05/2015 and qt.validon<=15/05/2015) depth 2;

Rewritten SQL Provenance Query: (select provenance, validon, query, resultid from provtbl1 pt, querytabletpch qt where SUBSTR(pt.resultid, 0, INSTR(pt.resultid, 't')-1) like qt.qid and provenance like '%P1%' and (qt.validon >=10/05/2015 and qt.validon<=15/05/2015)) union (select pt2.resultid, pt1.provenance, pt1.validon, pt1.query from provtbl1 pt2, (select provenance, validon, query, resultid from provtbl1 pt, querytabletpch qt where SUBSTR(pt.resultid, 0, INSTR(pt.resultid, 't')-1) like qt.qid and provenance like '%P1%' and (qt. validon>=10/05/2015 and qt.validon<=15/05/2015)) pt1) where pt2.provenance like '%' || pt1.resultid || ' %');

GDBMS: Trace all the result nodes directly or indirectly derived (depth=2) from source node labelled "USER" with screen_name "Anubhav" between '19/10/2019' to '21/10/2019'.

(**source** - node of source graph labelled as "USER" with value of screen_name attribute "Anubhav" between 19/10/2019 to 21/10/2019.)

Cypher Provenance Query: MATCH (u:USER{screen_name: "Abubhav"}) <-[p:provenance*1..2]<-(n1: QUERYTUPLE) with u, p, n1 MATCH (n:QUERY)-[t:TUPLE]->(n1) where (n.time>= date('2019-19-10') and n.time<= date('2019-21-10')) return n,t, n1, p, u.

KVPDB: Currently, mutli-depth provenance query is not supported by proposed framework.

Based on the above provenance query, we propose the following provenance query template:

Provenance Query Template 6:

TRACE

Destinations (D), Query (Q)

Where Source="$S", valid_from>="$T1" and valid_to<="$T2"

With Depth="$n"

RETURN D, Q, S

The above provenance query template returns all results (D) directly or indirectly derived from a given source (S) within specific time duration upto a given depth (n) along with the query statement that generates these results.

(e) ***Determine all the sources contributed to query executed at a particular time*** It is a kind of provenance tracing from derived results to source tuples for historical queries.

Example 7: Trace all the sources which directly or indirectly contributed (Depth=n) to a query executed at any particular time (historical query).

Example Provenance Query in Target Databases:

RDBMS: Trace all the source tuples contributed to result tuples of Query q1 on '10/05/2015'.

SQL Provenance Query: select provenance, validon, query, resultid from provtbl1 pt, querytabletpch qt where pt.resultid like 'q1%' and qt.qid= 'q1' and qt.validon = "10/05/2015";

GDBMS: Trace all the source nodes contributed to result nodes of Query q6 on '19/10/2019'.

Cypher Provenance Query: MATCH (n:QUERY)-[t:TUPLE]->(n1:QUERYTUPLE)-[p:provenance]->(a) where n.qid = "q6" and (n.time = date('2019-19-10')) return n, t,n1, p, a.

KVPDB: Trace all the source rows contributed to result rows of query q3 on '16/12/2019'.

CQL Provenance Query: select query, resulttupleid, provenance_paths, time from select_provenance where queryid= "q3" and (time=date(2019-12-16)) allow filtering;

Based on the above provenance query, we propose the following provenance query template:

Provenance Query Template 7:

TRACE

Source (S), **Destinations (D)**, **Query (Q)**

Where Query_Id="$QID" and Time="$T"

RETURN S, D, Q

The above provenance query template returns sources (S) of all the results (D) of a historical query with given queryid (QID) and Time (T).

2. *Provenance queries for querying historical data*

We have proposed extended query constructs in the three database models to query on historical data as already explained in Chapters 3, 4 and 5. All the queries are automatically rewritten before executing on the corresponding database.

Example 8: Determine all versions of a particular data object since its existence.

Example Provenance Query in Target Databases:

RDBMS: Trace all versions of address cell of supplier with supplier key = 1.

(**source** - address cell values of supplier with supplier key=1 in the supplier relation.)

SQL Provenance Query: select all s_address from supplier where supplier_key=1 validon NOW;

GDBMS: Trace all versions of location attribute of user named 'Anubhav'.

(**source** - location property of node "COUNTRY" linked with node "USER" whose name property is 'Anubhav'.)

Cypher provenance Query: MATCH all (u:USER)-[:FROM]->(c:COUNTRY) where u.screen_name= 'Anubhav' return c.Location validon NOW.

KVPDB: Trace all versions of location column of user named 'MemeBaaaz'.

(**source** - location column of user with user name = 'MemeBaaaz' in user_details column family.)

CQL Provenance Query: select all location from user_details where screen_name= 'MemeBaaaz' validon NOW;

Based on the above provenance query, we propose the following provenance query template:

Provenance Query Template 8:

RETRIEVE

All Versions (V)

Of Attribute ($A)

Where Source="$X", validon NOW

RETURN V

The above provenance query template retrieves all versions of given Attribute (A) for a given source (X) till now since it exists.

Example 9: Determine all versions of a particular data object between any specific period of time.

Example Provenance Query in Target Databases:

RDBMS: Trace all versions of address cell of supplier with supplier key = 1 till '15/04/2016'.

(**source** - address cell values of supplier with supplier key=1 in the supplier relation.)

SQL Provenance Query: select all s_address from supplier where supplier_key=1 validon '15-Apr-2015';

GDBMS: Trace all versions of location attribute of user named 'Anubhav' till '20/11/2019'.

(**source** - location property of node "COUNTRY" linked with node "USER" whose name property is 'Anubhav'.)

Cypher provenance Query: MATCH all (u:USER)-[:FROM]->(c:COUNTRY) where u.screen_name= 'Anubhav' return c.Location validon date('2019-11-20').

KVPDB: Trace all versions of location column of user named 'MemeBaaaz' till '23/10/2019'.

(**source** - location column of user with user name = 'MemeBaaaz' in user_details column family.)

CQL Provenance Query: select all location from user_details where screen_name= 'MemeBaaaz' validon 2019-10-23;

Based on the above provenance query, we propose the following provenance query template:

Provenance Query Template 9:

RETRIEVE

All Versions (V)

Of Attribute ($A)

Where Source="$X", [valid_from>= "$T1"] and valid_to<="$T2"/NOW

RETURN V

Here [...] is optional. In case time T1 is supplied then all versions of Attribute A since its existence in database till time T2 or NOW will be retrieved.

The above provenance query template retrieves all versions of given Attribute (A) for a given source (X) till now or given time T2 since it exists or from time T1.

Example 10: Determine the current version of a particular data object.

Example Provenance Query in Target Databases:

RDBMS: Trace the current version of address cell of supplier with supplier key = 1.

(**source** - address cell values of supplier with supplier key=1 in the supplier relation.)

SQL Provenance Query: select instance s_address from supplier where supplier_key =1 validon NOW;

GDBMS: Trace the current version of location attribute of user named 'Anubhav'.

(**source** - location property of node "COUNTRY" linked with node "USER" whose name property is 'Anubhav'.)

Cypher provenance Query: MATCH instance (u:USER)-[:FROM]->(c:COUNTRY) where u.screen_name= 'Anubhav' return c.Location validon NOW.

KVPDB: Trace the current version of location column of user named 'MemeBaaaz'.

(**source** - location column of user with user name = 'MemeBaaaz' in user_details column family.)

CQL Provenance Query: select instance location from user_details where screen_name= 'MemeBaaaz' validon NOW;

Based on the above provenance query, we propose the following provenance query template:

Provenance Query Template 10:

RETRIEVE

Instance Version (V)

Of Attribute ($A)

Where Source="$X", validon NOW

RETURN V

The above provenance query template retrieves the current version of given Attribute (A) for a given source (X).

Example 11: Determine the version of a particular data object at any specific time in the past.

Example Provenance Query in Target Databases:

RDBMS: Trace the version of address cell of supplier with supplier key = 1 on '15/04/2016'.

(**source** - address cell values of supplier with supplier key=1 in the supplier relation.)

SQL Provenance Query: select instance s_address from supplier where supplier_key =1 validon '15-Apr-2015';

GDBMS: Trace the version of location attribute of user named 'Anubhav' on '20/11/2019'.

(**source** - location property of node "COUNTRY" linked with node "USER" whose name property is 'Anubhav'.)

Cypher provenance Query: MATCH instance (u:USER)-[:FROM]->(c:COUNTRY) where u.screen_name= 'Anubhav' return c.Location validon date('2019-11-20').

KVPDB: Trace the version of location column of user named 'MemeBaaaz' on '23/10/ 2019'.

(**source** - location column of user with user name = 'MemeBaaaz' in user_details column family.)

CQL Provenance Query: select instance location from user_details where screen_name= 'MemeBaaaz' validon 2019-10-23;

Based on the above provenance query, we propose the following provenance query template:

Provenance Query Template 11:

RETRIEVE

Instance Version (V)

Of Attribute ($A)

Where Source="$X", validon "$T"

RETURN V

The above provenance query template retrieves historical version of given Attribute (A) for a given source (X) at given time (T).

In summary, all the proposed provenance query templates related to provenance queries for justifying query results and provenance queries for querying historical data are shown in Table 6.3.

6.3 Conclusions and Future Work

In this chapter, we proposed various provenance query templates for querying provenance information in a way which is independent of underlying database and application. The main motivation behind proposing the templates is to facilitate users to pose useful & common provenance queries using the templates. The proposed templates will guide the development of a full-fledged PQL which can be taken as a follow up work of the thesis.

S. No.	Provenance Query Template
T1	**TRACE** **Sources (S), Query (Q), Time (T)** **Where Destination="$D"** and **Query_Id**="$QID" **RETURN S, Q, T, D, QID**
T2	**TRACE** **Sources (S), Query (Q), Time (T)** **Where Destination="$D"** and **Query_Id**="$QID" **With** Depth="$n" **RETURN S, Q, T, D, QID**
T3	**TRACE** **Destinations (D), Query (Q), Time (T)** **Where Source="$S"** **RETURN D, Q, T, S**
T4	**TRACE** **Destinations (D), Query (Q), Time (T)** **Where Source="$S"** **With** Depth="$n" **RETURN D, Q, T, S**
T5	**TRACE** **Destinations (D), Query (Q)** **Where Source="$S", valid_from>="$T1" and valid_to<="$T2"** **RETURN D, Q, S**
T6	**TRACE** **Destinations (D), Query (Q)** **Where Source="$S", valid_from>="$T1" and valid_to<="$T2"** **With** Depth="$n" **RETURN D, Q, S**
T7	**TRACE** **Source (S), Destinations (D), Query (Q)** **Where** Query_Id="$QID" and Time="$T" **RETURN S, D, Q**
T8	**RETRIEVE** **All Versions (V)** Of Attribute ($A) **Where** Source="$X", validon NOW **RETURN V**
T9	**RETRIEVE** **All Versions (V)** Of Attribute ($A) **Where** Source="$X", [valid_from>= "$T1"] and valid_to<="$T2"/NOW **RETURN V**
T10	**RETRIEVE** **Instance Version (V)** Of Attribute ($A) **Where** Source="$X", validon NOW **RETURN V**
T11	**RETRIEVE** **Instance Version (V)** Of Attribute ($A) **Where** Source="$X", validon "$T" **RETURN V**

Table 6.3: Provenance Query Templates

Blue Color-Templates for Justifying Query Results.
Yellow Color-Templates for Querying Historical Data.