Querying Integrated Scientific Observation and Measurement Data

Author1
School of Electrical and
Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332–0250
Email: http://www.michaelshell.org/contact.html

Author2
Twentieth Century Fox
Springfield, USA
Email: homer@thesimpsons.com

Author3
Starfleet Academy
San Francisco, California 96678-2391
Telephone: (800) 555–1212
Fax: (888) 555–1212

***Abstract*—The abstract goes here.**

I. Introduction

In this work, we study the query processing over scientific observation and measurement data using OBOE model[**?**]. OBOE model is a conceptual model used to interpret observation and measurement data.

A. Background

In many scientific domains (e.g., ecology, hydrology, earth science, geology), people collect observational data. Such data record the observed value of some real world entity at some specific place and time. E.g., ecologists studying relationship between the growth pattern and the treatments often need to record the tree heights. The collected data reflect the fact the tree height of a specific tree is 30.1in on May 1, 2009 and 30.3in on May 1, 2010.

Almost all such scientific data do not follow database higher normal forms. Generally, scientists have their way in interpreting their data, but they are not ready for any normalization process. For example, Table I is a simplified but typical dataset collected by a scientist who study the growth pattern of trees. Obviously, the "plt" of the first two rows is the same place, and the "plt" of the last two rows is the same. Here, their area information is redundant. In the real application, many columns have redundant information.

code	no	ht	plt	area
piru	1	35.8	California	4.0
piru	2	36.2	California	4.0
piru	1	25.7	Oregon	3.0
Oriental poppy	1	7.8	Oregon	3.0

TABLE I
DATASET

In scientific domains where people collect observation and measurement data, there are several commonly used and widely recognized canonical concepts([**?**], [**?**]). In this paper, we refer these concepts as OM concepts. The canonical concepts include *observation*, *measurement*, *characteristic*, *standard*, *protocol* or (*procedure*). For example, **add an example to illustrate these concepts.**

Generally, every scientific dataset goes into the data repository with some metadata, e.g., Dublin core metadata[**?**], Darwin core metadata[**?**]. However, all these metadata are at the dataset level, they did not provide enough information for the data *content* inside the dataset. To better make use of the data content in the repository, more systems are embracing the ideas of having metadata on the data content. E.g., Some systems [**?**] provides a mechanism to collect the column/attribute level metadata, some system [**?**] uses *annotation* to add more semantic information to the data content. In this work, we follow the terminology in [**?**] and use the term *annotation* to distinguish the metadata on the data content from those on data objects at a coarser granularity. We propose tools for scientists to provide annotations to scientific observational data. **[**FROM HP: **Put a screen-dump of annotation.]**

The internal data structure of such scientific data repository is generally unknown to a user who wants to pose a query to such scientific data repository and get some useful information. So, it is unrealistic for a him/her to formulate a query based on the underlying data structure of the data. The easiest query they can pose to such data repository are keyword-query. Such keyword style query works well for the dataset metadata (e.g., contributor information, dataset description, etc) using the current Information Retrieval (IR) techniques. However, it is far from enough to be used to search the data content because the data content need to be interpreted with semantics. It is important in the scientific domain to have typical kinds of queries. One fact we can use is that the domain scientists know well the widely used OM concepts as discussed above. So, naturally, when searching such scientific data, they can provide the concept information to restrain their queries and to find data sets related to such OM terminologies. Given the dataset in Table I. People may ask the following observation and measurement (OM) queries.

Example 1.1: Simple and summarization queries:

- Q_1: Give me the data sets that contain species "Picea rubens" observations.
- Q_2: Give me the data sets that contain species "Picea rubens" observations in "California".
- Q_3: Give me the data sets that contain at least five distinct "Picea rubens" observations.

- Q_4: Give me the data sets that have trees with average "height" than 20.0 in "California".

[FROM HP: **Challenges to answer such queries: uniqueness]**

[FROM HP: **Current data integration effort]**

[FROM HP: **Our effort in querying observational data: introduce OBOE]**

[FROM HP: **Need re-organization. The description in the following several sections may be moved to here later.]**

B. Contribution and paper organization

Contributions of this work:

- We formulate a canonical set of queries over observation and measurement scientific data repository. Such formal queries can formalize most OM concepts related searches.
- We propose three methods to evaluate such OM queries. **[**FROM HP: **Elaborate the three methods.]**

This paper is organized as follows. Section II reviews works that are related to this research. Section III formalizes the data model and the queries that people are interested to ask.

II. Related work

[FROM HP: **This will come after the real problem definition and the solution. The description in the following several sections may be moved to here later.]**

III. Data model, annotation and query

In this section, we first illustrate the data model. Then, we formalize the queries that scientists in this domain tend to ask.

A. Data model

When a scientist contributes data into an integrated data repository, a widely accepted way is to convert each dataset to a data table[**?**] or treated as a separate object entity[**?**]. **(HP: is it really widely used? any other system uses this way?. Add more citations here.)** In using this method, the database contains metadata of each dataset and the definition of the data table (e.g., attribute/column name, attribute type, etc).

In our work, we focus on querying this kind of data repository (or databases). To formalize the scenario, we use D to denote the set of data tables in the data repository and d to refer to a specific data table. Each data table d contains metadata about the attribute definition $Attr_d$. Sometimes, we also use *column* to refer to an attribute. Given one data table d, an attribute $attr_i \in Attr_d$ or column index i, $d[attr_i]$ or $d[i]$ represents the set of values for the attribute $attr_i$ or for the i-th column.

B. Annotation

We use A to denote the annotation of one dataset. Internally, we keep the following information for the annotation.

For a data table d with annotation A, the system keeps the corresponding information between them in AD tables. Besides it, we also have four main concepts to describe: observation type (OT), measurement type (MT), and context type (CT), and the mapping (Map) from the measurement type to resource attributes. So, the annotation contains the following information.

- $AD = \{(d_{id}, A_{id}, d_{meta})\}$ to keep the information of data sets d_{id}-s and their related annotations A_{id}. Here, d_{meta} is an abstract attribute for all the other metadata of d_{id}.
- $OT = \{(\underline{A_{id}, ot_{id}}, et, isDistinct)\}$ to describe an observation type. It denotes on which entity type (object in the real world) the observation is made. Very often, more than one observation can be made on one entity. Thus,*isDistinct* is used to denote whether the same value of key measurements of an obbservation types can uniquely identify one observation or not.
- $MT = \{(\underline{A_{id}, mt_{id}}, ot_{id}, isKey, Cha, \cdots)\}$. Generally, MT contains information about characteristic (Cha), Standard, Protocol and Precision. We do not include them here to make the description clearer. Here, *isKey* is used to denote whether one measurement type is the key measurement for the observation type OT_{id} or not.
- $CT = \{(\underline{A_{id}, ct_{id}}, ot_{id}, cot_{id}, Rel, isIdentify)\}$ where cot represents the context observation type.
- $Map = \{(\underline{A_{id}, mt_{id}, attr_i, mapCond, mapVal}, Val\}$. This structure denotes the correspondences between an measurement type (mt_{id} and an attribute $attr_i$ in an dataset d_{id} for a given condition $mapCond$.

We use the following example to illustrate the concepts in the annotation.

Example 3.1: $OT = \{(A_1, OT_1,$ tree, false),$(A_1, OT_2,$ GeoSpot, true) $\}$ denotes that annotation A_1 are for two observation types. One type OT_1 is on real world entity "tree" and the values of its key measurements do not uniquely identify a distinct observation. The other type OT_2 is for real world entity "GeoSpot" (i.e., geospatial location) and its key measurement types can identify it's unique observations. **[**FROM HP: **Introduce key measurements before this.]**

MT =$\{(A_1, MT_1, OT_1,$ Species, true), $(A_1, MT_2, OT_1,$ SpecNo, true), $(A_1, MT_3, OT_1,$ Height, false), $(A_1, MT_4, OT_2,$ Plot_State, true), $(A_1, MT_5, OT_2,$ Plot_Area, false)$\}$ represents that OT_1 (for "tree") has three measurement types ($MT_1, MT_2,$ MT_3) for characteristics "Species", "SpecNo", and "Height". The fist two measurement types together form the key measurement type for OT_1. OT_2 (for "GeoSpot") has one key measurement type MT_4 for characteristic "Plot_State" and another measurement type MT_5 for characteristic "Plot_Area".

CT = $\{(A_1$,OT_1, OT_2, within, true)$\}$. It shows that observation of a *tree* is within the context of a geo-spatial location. And this context is used to identify the uniqueness of the observation. E.g., a tree with species name "piru" and species no 1 in California is different from a tree with the same species name "piru" at Oregon.

Map = $\{(A_1$, MT_1, "code", "eq 'piru"', "Picea rubens"), (A_1, MT_2,"no", null,), (A_1, MT_3, "plt", null,), (A_1, MT_4,"area", null,$\}$. The first mapping rules maps "code" attribute to the measurement type MT_1 (for "Species" characteristic) and change the value to "Picea rubens" if the code is “piru”. The meaning of the other mapping rules are very obvious here, so we skip the details.

C. *Observation and measurement (OM) query*

Above we defined the data model and the annotation. In this sub-section, we formalize the queries that scientists are interested in. We denote the queries on the observational data as *Observation and measurement (OM) query*.

As discussed in Section I, it is generally impossible for a scientist to formulate an OM query using the internal data structure of a dataset because such data structures differ from contributors to contributors. More realistically, they can pose queries (Example 1.1) related to key OM concepts, e.g., *observation, measurement,characteristic, and standard*, and these OM concepts satisfy given keyword and context conditions.

Definition 3.1 (Basic OM query): A basic observation and measurement query is defined as

$$Q ::= \texttt{OMconcept} : \langle \texttt{cond} \rangle \}$$

Here,

- `OMconcept` refers to one of the major OM terminologies.
- $\langle$`cond`$\rangle$ is in the form of `f([distinct entity.]measurement) op value` where `op` is a basic operator from $\{=, \neq, >, <\}$ and `f` is an aggregation function from $\{sum, avg, count, min, max\}$. `distinct` and `entity` are optional.

[FROM HP: **The symbol fonts are a little bit messy...need to distinguish between the keywords and variables.]**

Definition 3.2 (Result of OM query): The result of a query Q is a set of data objects (e.g., data tables) $\{d | d \in D \wedge d\ satisfies\ \langle \texttt{cond}\ \rangle\}$. For each of such result d, we use $d\ s.t.\ Q$ to denote that an data object d satisfies the query.

Based on the different $\langle$`cond`$\rangle$ definitions, “$d\ \ satisfies\ \ \langle \texttt{cond} \rangle$” is translated differently. E.g., if $\langle$`cond`$\rangle$ is defined that the value of “*area*” need to be smaller than 3.0, then, “$d\ satisfies$ `cond`” is translated to “$dv < 3.0 | dv \in d[area]$”.

Definition 3.1 can formulate queries on one specific observation and measurement. In real application, we need to consider the context relationship. **[**FROM HP: **E.g., etc.]** So, we generalize the basic OM query to contextualized OM query as follows.

Definition 3.3 (Contextualized OM query): A contextualized OM query is defined as:

$$CQ ::= Q_i \wedge \texttt{context}(Q_i.\texttt{OMconcept}, Q_j.\texttt{OMconcept}) \wedge Q_j$$

Here, Q_i, Q_j are basic queries.

For example, the queries in Example 1.1 can be formalized as the following formal OM queries.

- Q_1 = $Observation : \langle Species =' \ Picea\ rubens' \rangle$
- Q_2 = $Observation_1 : \langle Species =' \ Picea\ rubens' \rangle$ $\wedge IN(Observation_1, Observation_2)$ $\wedge Observation_2 : \langle Meas_{unknown} = `California' \rangle$
- Q_3 = $Observation : \langle Species =' \ Picea\ rubens'$ $\wedge count(distinct\ Species) \geq 5 \rangle$
- Q_4 = $Observation_1 : \langle avg(distinct\ tree.height) \geq 20.0 \rangle \wedge IN(Observation_1, Observation_2)$ $\wedge Observation_2 : \langle State =' \ California' \rangle$

IV. Querying annotated and integrated OM data

In this section, we propose several methods that can be used to answer OM queries over scientific observational data with annotations.

A very direct way to search such data is to extract all the content in the data cells and index them. This way the system may help answer very simple queries with the help of some thesaurus. For example, for Q_1 in Example 1.1, as long as we can translate the abbreviation “piru” in the dataset to “Picea rubens” based on some predefined thesaurus, we can find the results. However, even for this kind of simple query, if we want to find observation of “California poppy”, an approximate match may return the dataset in Table I as a result because it contains both *California* and *Poppy*. If the search problem caused by the compound words can be alleviated using some existing techniques [**?**], it is still almost impossible to directly use such IR-style system to answer a query in the form of Q_3 and Q_4. This naive method cannot be directly used answer the desired queries. It is because it fails to catch the semantics of the context and uniqueness of observations in the data. In what follows, we utilize the annotation information to get more semantic support, and provide new strategies to answer the queries.

As we analyzed in Section II, there are two extremes in querying integrated data. One is to rewrite the original query to a series of new queries over the data; the other is to materialize the data to a consistent data model and then answer the query over the materialized database. In what follows we work from these two directions to leverage the semantic information in answering such queries.

A. *Query rewriting*

As described in Section III, an OM query is represented using OM concept terminologies such as observation, measurement, characteristic, etc. To answer such queries from the original data model (i.e., dataset metadata and translated data tables), we need to *rewrite* the OM query over the real data tables. In what follows, we first sketch the re-writing process. Then, we detail the procedure in using the data model and structures. Finally, we include the process for the more complicated cases (e.g., with distinct, aggregate).

Roughly speaking, the query rewriting consists of two steps:

- From the given query with entity, characteristic information, search the database metadata and annotation information to find the relevant data tables and attributes that need to be used to answer the given query.
- Translate the given OM query to queries over the relevant data tables. To do this translation, we decompose an OM query to have several basic component operators.

– Basic OM query with one entity type. While the characteristic condition may be conjunctive or disjunctive, it may also has aggregation. E.g., $A(Cha1 > 25.0\&\&Count(Cha) > 3, Height > 2.0)$
– Context OM chain refer to a set of queries having "HasContext" relationship. Obviously, these queries have different entity condition. E.g., $Ent1(...) \rightarrow Ent2(), Ent1() \rightarrow Ent3()$
– Disjunctive OM query with multiple entity types. $Ent1(...), Ent2(), Ent3(...)$.

[FROM HP: **todo:put a figure here to illustrate the steps]**.

The first step is to map the OM concepts in the query to the real data structure. With the annotation structures OT, CT, and MT, any given OM concepts can be linked to a measurement type mt_{id}. Then, the annotation structure Map can be used to bridge OM concepts and data sets. Utilizing Map, we can find the attributes $attr_i$ and the annotation A_{id}). E.g., when the given concept is *measurement*, we can directly get its related attribute names and annotation id A_{id}. In case that the given OM concept is a non-measurement concept, we can use other annotation structures to figure out the measurement types. E.g., if the OM concept is *characteristic* "Species", we can use MT to get the measurement type id mt_{id}. After we get A_{id}, we can quickly find the dataset id d_{id} using AD metadata table (Section III-B). Then, we get the relevant data tables and the needed attributes.

Let D_{cand} be the set of relevant (or candidate) data tables and $d_{a_1} \cdots d_{a_m}$ are the related attributes for table $d \in D_{cand}$.

The second step translates the original OM query to SQL queries over the data tables. From now on, we assume that we know the attribute already for all the needed characteristics.

To translate the given query: Figure 1 shows the algorithm framework to deal with one OM query. **[**FROM HP: **More description comes later.]**

In particular, for each candidate data table d, the translation process works in the following rules. For an OMBasic query with one entity type. We need to do the

- The SELECT clause always get the distinct d_{id} and the key attributes for this entity type;
- The FROM clause has table $d \in D_{cand}$;
- The WHERE clause contains all the non-aggregational conditions $f(\cdot)$ in the OM query;
- When there is distinct constraint on entity or observation, the GROUP BY clause contains of all the key attributes corresponding to the of the entity. we detail all process of getting the key measurement types of an observation later.
- The aggregation requirement in $f(\cdot)$ is translated to HAVING clause.

Based on these rules, an OM query can be translated to the following SQL scripts.

For non-aggregate conditions,

```
Algorithm Execute(OMQ)
Result := ∅;
//From the given query, get the DNF of context queries.
contextChainDNF := GroupBasicQuery(OMQ);

//Execute every DNF int the context chain set and union their results
for each context chain oneContextQuery in contextChainDNF
    result_CNF := ∅;
    //Execute every basic query in this CNF query and intersect their results
    for each basic query OM_b in ContextQuery
        result_b := ExecuteBasicOM(M_b);
        result_CNF := result_CNF ∩ result_b;
    Result := Result ∪ result_CNF
Return Result;

Algorithm GroupBasicQuery(OMQ);
contextChainSet := ∅;

//Put the basic queries in context into context chains
for (each basic context OM_b → COM_b in OMQ)
    added := false;
    for (each set oneContextChain ∈ contextChainDNF)
        if ((OM_b ∈ oneContextChain) or OM_b ∈ oneContextChain))
            oneContextChain := oneContextChain ∪ OM_b ∪ COM_b
            added :=true;
            break;
    if(added==false)
        Create a set newContextChain with two elements OM_b and COM_b;
        Add newContextChain to contextChainSet;

//Put the basic queries without context into context chains
for (each OM_b not in context of OMQ)
    Create a set newContextChain with one element OM_b
    Add newContextChain to contextChainSet;
return contextChainSet;
```

Fig. 1. Algorithm to execute an OM query

```
SELECT DISTINCT d_id, key attribute list
FROM d
WHERE non − aggregate conditions)
```

For each aggregate condition,

```
SELECT DISTINCT d_id, key attribute list
FROM d
WHERE key attribute_values IN(
        non − aggregate SQL)
[GROUP BY distinct requirement]
[HAVING aggregation condition];
```

When there are more than one aggregation condition, the above SQL is formed for each aggregation condition and their results are intersected.

[FROM HP: **Output: record id or key attribute values.]** If the output is record id, we further need to

```
SELECT record_id
FROM d
WHERE (key measurement value IN(
       SELECT DISTINCT key attribute list
       FROM (results from aggregationfunctions)
```

The above describes the process for the OM query. In real application, people ask queries with distinct observation or entity constraints. Also, people ask queries about different observatitons using context. The following elaborate the details in processing these two cases.

When a query with distinct constraints, e.g., find data sets with *distinct* observations or entities. To deal with this case, we need to be able to figure out what distinguishes one observation from another. Here, one observation type's key measurement types can play this role well. When an observation type is denoted with *distinct yes* and some of its measurement types are denoted with *key yes*, the values of the measurements on the key measurement types can uniquely distinguish one observation from the other. To implement this, in SQL, we can perform "group by" operations on the key measurement types of the observations.

When the given OM query has context, we need to consider two cases. In the first case the context is not *identifying* to its observation types, i.e., the context value does not affect the uniqueness of one observation, this can be processed as the basic way. When the context is denoted with *identifying yes*, it means the identity of one observation also depend on the contextual observation. In this case, we need to figure out the all the key measurement types of one observation, then the "GROUP BY" operation should be on all these key measurement types. [FROM HP: **Put an example here.]** In this case, there is no need to change the SQL scripts. the only change are the relevant attribute set $d_{a_1} \cdots d_{a_m}$, which corresponds to not only the measurement types that directly related to the observation, but also the measurement types that are in the identifying context chain of the observation.

[FROM HP: **Put a specific routine to get key measurements of observation and/or entity.]**

Example 4.1: TODO: refine this example Take Q_1 $(Observation : \langle Species =' \ Picea \ rubens' \rangle)$ as example. The `OMconcept` is "observation" and `cond` is "*species* = '*Picea rubens*'";

From the annotation MT, we can find the measurement type m_3 with the characteristic *species*. From the mapping Map, we further find that the attribute *spp* that the measurement type m_3 is mapped to.

The second step is to find the data sets which really have this value. For this one, we find the data table, and do a selection on the table content.

```
SELECT DISTINCT d_id
FROM d
WHERE spp =' piru'
```

Example 4.2: Take Q_3 $(Observation :$ $\langle Species =' \ Picea \ rubens' \wedge count(distinct \ Species) \geq 5 \rangle)$ as an example.

[FROM HP: **Refine**
For the first step, we find the data tables that contain the needed concepts. In each table, we need to figure out the distinct observations. From m_i $obstype_i$, all the other measurements that are of $obstype_i$. Group by the key measurements of $obstype_i$ if distinct yes. If this observation type does not have any context, then the key measurements are all the measurements directly defined under this observatioin type. Otherwise, i.e., this observation type has context, then the key measurements are all the measurements in the context chain.]

```
SELECT DISTINCT d_id
FROM d
WHERE spp =' piru'
GROUP BY d_id, spp
HAVING count(d_id, spp) ≥ 5
```

Cost analysis: From the above description, we can tell that the computation cost is to search the different data tables. This way, for each candidate data table, we need to send an SQL query to the server for evaluation purpose. If the number of candidate data tables are small. This method should work well. However, when we have a lot of candidate data tables, the efficiency may be affected. In what follows, we introduce another strategy, which logically merge the data content in different tables and perform queries ober the materialized database.

B. Querying materialized database

The previous section shows the method to rewrite a given query. However, as we analyzed, the computation cost may be very high when there are many candidate data tables. In this section, we propose another strategy to answer the given OM query by making use of the existing optimization strategies of current DBMS.

In this strategy, we do not directly work on the original data table D. On the other hand, we *materialize* the original datasets to centralized concept tables with instances of entities, observations, measurements and contexts. Such concept tables are called core OBOE tables, and denoted as $OBOE.*$. Then, the query process works on the materialized databases (i.e., the conceptual data tables).

In what follows, we first describe the structure of the core conceptual tables and present our algorithm to materialize the data into such concept instance tables in Section IV-B1. Our algorithm considers all the distinct and context constraints. Then, Section IV-B2 provides the detailed process of performing queries over materialized database.

1) Materialize data: In particular, we have the following concept instance tables.

- $OI = \{(\underline{oi_{id}}, ot_{id}, ei_{id})\}$ keeps all the observation instances where ot_{id} is used to link to the annotation information in OT;
- $MI = \{(\underline{mi_{id}}, oi_{id}, mt_{id}, mVal)\}$ for all the measurement instances where mt_{id} is used to link to the annotation information in MT;
- $CI = \{(\underline{oi_{id}, coi_{id}, ct_{id}})\}$ for all the context instances where ct_{id} is used to link to the annotation information in CT.

[FROM HP: **TODO: refine the writing in MaterializeDB and move them here.]**

2) Querying materialized database: Once the data are materialized using the annotation constraints, we can perform the OM query over them.

[FROM HP: **TODO:Formalize the translation of operators in the OM query]**.

For a non-aggregate condition,

```
SELECT DISTINCT did, record_id, eid
       characteristic, mvalue
FROM mi, mt, oi, ei
[WHERE table join condition,
         AND selection condition]
INTERSECT other measurement cond
```

For an aggregate condition,

```
SELECT DISTINCT did, record_id, eid
FROM oi
WHERE (did, eid) IN (
       SELECT did, eid
       FROM non-aggregate sql
       WHERE characteristic condition
       GROUP BY did, eid
       HAVING aggregation condition;
```

When the given query is a basic OM query, i.e., there is not context in the query. Then, we can form a template query as the following;

```
SELECT [DISTINCT] AD.d_id
FROM AD, MI as mi, MT as mt[, other tables]
[WHERE table join condition,
        AND selection condition],
[GROUP BY distinct requirement ]
[HAVING aggregation condition ];
```

Fig. 2. SQL template to answer OM using materialized DB

In particular,

$$table\ join\ condition = (mi.mt_{id} = mt.mt_{id})\ AND\ (mt.A_{id} = AD.A_{id})$$

For the very basic query (e.g., Q_1 in Example 1.1), the SQL generally need only the first three clauses. If the query has some restriction on the aggregated result, the `GROUP BY` and `HAVING` clauses to deal with it.

We use Q_1 and Q_3 to illustrate the query on the materialized database. can be answered using the above template. Example 4.1 shows how to deal with Q_1 using the query rewriting method.

Example 4.3: Using the materialized database, we just need to search the MI table with the condition that *mVal="Picea rubens"*.

```
SELECT DISTINCT AD.d_id
FROM AD, MI as mi, MT as mt
WHERE mt.Cha =' Species'
       AND mi.mVal = 'Picea rubens'
       AND mi.mt_id = mt.mt_id
       AND mi.A_id = AD.A_id;
```

For Q_3, which asks for data sets that contain at least five distinct "Picea rubens" observations. The SQL query can be written as:

```
SELECT distinct AD.d_id
FROM AD, MI as mi, MT as mt
WHERE mt.Cha =' Species'
       AND mi.mVal = 'Picea rubens'
       AND mi.mt_id = mt.mt_id
       AND mt.A_id = AD.A_id
GROUP BY d_id, oi_id
HAVING COUNT(*)>5;
```

For this kind of query, the computation cost is the selection cost of the measurement instance (MI) table. Compared with the query rewriting strategy, which need to pose the query over multiple data tables, this query should answer the queries more efficiently.

Figure 2 shows the SQL template to process the basic OM query without context. In case the OM query has context, we need to add the context into consideration.

1) The first step is similar to that in the basic OM query processing. Besides this, we need to figure out what are the concept obserbationi types of the needed observations.
2) Form an SQL for the observation and every of its context observations.
3) Intersect all the SQLs from Step 2).

```
SELECT [DISTINCT] AD.d_id
FROM AD, CI as ci, MI as mi, MT as mt
    MI as cmi, MT as cmt[, other tables]
[WHERE table join condition,
        AND selection condition],
[GROUP BY distinct requirement ]
[HAVING aggregation condition ];
```

Fig. 3. SQL template to answer contextualized OM using materialized DB

In this case, the table join condition is much more complex since we need to use the context instance table ci to connect the observation and context observation table. In particular,

$$\begin{aligned} &table\ \ join\ \ condition = \\ &\quad (mi.oi_{id} = ci.oi_{id})\ \text{AND}\ (cmi.oi_{id} = ci.coi_{id})\ \text{AND} \\ &\quad (mi.mt_{id} = mt.mt_{id})\ \text{AND}\ (cmi.mt_{id} = cmt.mt_{id})\ \text{AND} \\ &\quad (mt.A_{id} = AD.A_{id}) \end{aligned}$$

Execution cost analysis: From the join condition, we can see that the expensive cost comes from the join operation between the measurement instance, context measurement instance, and context instance table. As we analyzed in Section IV-B1, the two tables with growing space are measurement instance table and context instance table.

Example: When we are given contextualized query, e.g., Q_2, which asks for the datasets that contain species "Picea rubens" observations in "California".

In the first step, we get the requirement is on Characteristic "Species" with value ($mVal$) "Picea rubens" and observation has context *in* another observation for "State" of value "California".

In the first step, we need to find the context type id tmp_ct_id that satisfy the context relationship.

SELECT $ct.ct_{id} as tmp\_ct\_id$
FROM OT as ot, OT as cot, CT as ct
WHERE $ct.ot_{id} = ot.ot_{id}$ AND
$ct.cot_{id} = cot.ot_{id}$ AND
$ct.Rel =' IN'$ AND $ot.et =' tree'$;

Then, the next step is to find the distinct data sets.

SELECT DISTINCT $AD.d_{id}$
FROM $AD, CI\ as\ ci, MI\ as\ mi, MT\ as\ mt,$
$MI\ as\ cmi, MT\ as\ cmt,$
WHERE $(mi.oi_{id} = ci.oi_{id})$ AND $(cmi.oi_{id} = ci.coi_{id})$ AND
$(mi.mt_{id} = mt.mt_{id})$ AND $(cmi.mt_{id} = cmt.mt_{id})$ AND
$(mt.A_{id} = AD.A_{id})$
AND $mi.mVal = `Picearubens'$
AND $cmi.mVal =' California'$
AND $mt.Cha =' Species'$
AND $cmt.Cha = `State'$
AND $ci.ct_{id} = tmp\_ct\_id)$

Now let's think about the query Q_4 with aggregation and context requirement. $Q_4 = Observation_1 : \langle avg(distinct\ tree.height) \geq 20.0\rangle \wedge IN(Observation_1, Observation_2) \wedge Observation_2 : \langle State =' California'\rangle$

Also, in the first step, we discover the context type id tmp_ct_id that satisfy the context relationship.

SELECT DISTINCT $AD.d_{id}$
FROM $AD, CI\ as\ ci, MI\ as\ mi, MI\ as\ cmi,$
$MT\ as\ mt, MT\ as\ cmt,$
WHERE $(mi.oi_{id} = ci.oi_{id})$ AND $(cmi.oi_{id} = ci.coi_{id})$ AND
$(mi.mt_{id} = mt.mt_{id})$ AND $(cmi.mt_{id} = cmt.mt_{id})$ AND
$(mt.A_{id} = AD.A_{id})$
AND $mt.Cha = `height'$
AND $cmt.Cha = `State'$
AND $cmi.mVal = `California'$
AND $ci.ct_{id} = tmp\_ct\_id)$
GROUP BY $AD.d_{id}, oi_{id}$
HAVING $avg(mi.mVal) > 5.0$;

Note that, in the above example, there is only one context. In the real application, there may be multiple context observations, or even context chains. In this case, we need to get *all* the context and get the intersection of the results.

C. Query partially materialized database

As we analyzed in the first two sub-sections, the query-rewriting suffers the multiple queries posed on different candidate dataset. The querying of materialized database suffers from the multiple self-join operation of the measurement instance (MI) table, which has the most amount of information.

In this section, we propose to materialize only the measurement instance and also put the measurement instance and context measurement instances together in the same table to avoid the expensive computation cost.

[FROM HP: **Let's see how this can work.]**

V. EXPERIMENTS

Synthetic data generator.

Algorithm to materialize DB. Report the time and space.

Synthetic query generator.

Test algorithm to perform query over materiazed DB. Test algorithm to perform query-rewriting over materiazed DB.

Test algorithm for half-materialized data.

VI. CONCLUSION