

1 Key constraints

In real world, the dataset collected by an ecologist to describe measurements and observations is not normalized. I.e., two different rows in the table may have a big amount of “redundant” information to describe an object (e.g., entity).

As discussed in [?] we can use the OBOE conceptual model to capture the semantic relationships from the columns to their real meaning. However, the mapping from the columns to the OBOE model still cannot directly solve the problem of getting concise information (e.g., unique entity instances, unique observation instances, etc.)

Given Table 1(a), it records the fact that four times of dbh of some trees are taken. Very often, an ecologist may ask questions like this: Give me the average dbh for every different trees. First, we have four rows here for trees. But how many tree entity instances are here is a question.

There are several cases to consider:

- Case 1: The naive extreme way to interpret the data is that each observation is from different tree entity. Then, we have **four** tree entities. This may be too strict. People may say, well, I have some observations for the same entity.
- Case 2: The second naive extreme way is to interpret that different *spp* represent the different tree entity. That’s obvious that *piru* is different from *abba*. With this constraint, we get **two** tree entities. However, an obvious fact is also true that one *piru* tree cannot at the different plots. (It is another story if this tree was migrated from one plot to another.)
- Case 3: Following Case 2, people want to further limit that the same *spp* in the same *plt* should represent the same tree entity set.

plt	spp	dbh	plt	area	spp	dbh
A	piru	35.8	A	1.0	piru	35.8
A	piru	36.2	A	1.1	piru	36.2
B	piru	33.2	B	2.0	piru	33.2
B	abba	34	B	2.0	abba	34

(a)

(b)

Table 1: Dataset

Obviously, to answer this question, we not only need to figure out what are the observations and measurements, we also need proper constraints to help us understand whether different observations are from the same entity or not.

To tackle this problem, we proposed to use *key yes*, *distinct yes* and *identifying yes* constraints.

[FROM HP: **Describe or define** *key yes*, *distinct yes*, *identifying yes*]

Definition 1.1 (Key measurement types) *The key measurement types of an observation are the measurement types that are used to distinguish one observation/entity instance from the other.*

In most simple case, when an observation type does not have context, the key measurement types are specified with “key yes”.

Key constraints: in this report (paper), we use the term *key constraints* to refer to the general constraints of the above three types *key yes*, *distinct yes* and *identifying yes*.

2 Valid annotation constraints

Based on the semantic meaning of these key constraints, we can see that they have some correlations. Before we give any further examples to explain such correlations, we list the correlation rules among them.

- **Rule 1:** If an observation type is specified with *distinct yes*, all its measurement types are automatically marked with *key yes*.
- **Rule 2:** If the context of an observation type is specified with *identifying yes*. This observation type MUST have some key measurement types. And the context observation type also MUST have key measurement types too.

The following example shows an example why Rule 1 is needed. Given the data in Table 1(b) and the following annotation.

```

observation "o1" distinct yes
  entity "Plot"
    measurement "m1" key yes
      characteristic "EntityName"
      standard "Nominal"
    measurement "m2"
      characteristic "area"
      standard "sqft"
  ...

```

According to the annotation, if two plots have the same value for "EntityName", they represent the same plot observation. Obviously, it has problem to interpret the data in Table 1 with this annotation. E.g., the first and second rows catch information about plot with EntityName *A*. According to the annotation, they should be the same plot observation. However, the data shows that this one plot has two different areas 1.0 and 1.1, so, there is confusion here.

For this case, we can have several ways to make the annotation consistent. The first alternative to tackle this confusion is to change the annotation. We can let m_2 automatically becomes key measurement type of o_1 . Then, every different (m_1, m_2) value pair denote a different observation instance which come from the different entity instance. The second alternative is to get rid of *distinct yes* for o_1 , which means that $(A, 1.0)$ and $(A, 1.1)$ are different observation instances of the same entity instance *A*.

In our materialization algorithm, we assume that the annotations on the dataset are valid, i.e., they comply with the rules among these constraints.

3 Data materialization algorithm

The **MaterializeDB** algorithm catches the *key*, *distinct* and *identifying* constraints in the annotation during the materialization process. The input of the algorithm is the *Dataset* and the annotations $A.*$ on it. Each row in the input dataset represents the information related to one or more observations and their contexts. The input $A.*$ represents the annotation information. More specifically,

- $A.MeasType = \{\langle \underline{MeasTypeId}, ObsTypeId, CharType, StdType, ProtType, Precision, isKey \rangle\};$
- $A.ObservationType = \{\langle \underline{ObsTypeId}, EntTypeId, isDistinct \rangle\}$
Note that, we hide the "AnnotId" in this schema, which denotes the resource the annotation is on. We do not include this because the algorithm focuses on dealing with annotations on one resource. This way, we can simplify the description.
- $A.ContextType = \{\langle \underline{ObsTypeId}, ContextObsTypeId, RelType, isIdentify \rangle\}$
- $A.Map = \{\langle \underline{MeasTypeId}, ResAttribute, Cond, Val \rangle\}$

The output of the algorithm is a set of materialized tables represented in the OBOE model and denoted by $OBOE.*$. In detail,

- $OBOE.Observation = \{\langle \underline{ObsId}, ObsType, EntId \rangle\}$ keeps all the observation instances;

- $OBOE.Measurement = \{\langle \underline{MeasId}, \underline{ObsId}, \underline{MeasType}, \underline{Val} \rangle\}$ for all the measurement instances;
- $OBOE.Entity = \{\langle \underline{EntId}, \underline{EntType} \rangle\}$ for all the entity instances;
- $OBOE.Context = \{\langle \underline{ObsId}, \underline{ConextObsId}, \underline{ContextType} \rangle\}$ for all the context instances;

Algorithm 1 shows the framework of our algorithm. In this algorithm, we maintain two intermediate index structures ($EntIdx$ and $ObsIdx$) to keep track of the distinct entity and observation instances. $ObsIdx$ is the index structure maintained for the observation instances whose types are specified with *distinct yes*. This index maintains the mapping from observation type and key values to its corresponding observation instance id. Obviously, only when an observation type is specified with *distinct yes*, (i.e., we want to keep track of the same observation instances) we need to maintain their instances in this index.

The key values can be calculated in three different cases. In the first case where only one measurement of an observation type is specified with “key yes”, the key value is the value of this “key” measurement. In the second case, several measurement types are marked with “key yes”, the key value is the combined instance value of these several measurement types. In the third case, some context of this object type is marked with “identifying yes”, the key value is the combination of the instance values of its own key measurement types and the instance values of its context observation’s key measurements.

$Entidx$ is the index structure for tracking the distinct entities. In case that some measurement type(s) of an observation type is/are specified with “key yes”, if different observation instances have the same value on these key measurements, semantically, we interpret that they are of the same entity. The key value for each entity instance is computed in the same way as that for an observation instance.

The procedure of this algorithm is very straightforward. It processes the dataset in a row-wise manner. In dealing with each row, five steps are involved. The first step generates orphan measurement instances which are not connected to any observation instances. The second step groups these measurement instances according to their observation types. Then, for different observation types, the third and fourth steps materialize entity instances and observation instances respectively by either creating new ones or return existing ones. The last step assigns the context relationship among the different observation instances.

Analysis of MaterializeDB.

Time: As can be seen that this algorithm scans the original data file in a row-by-row manner without revisiting the already seen rows. So, it is linear in the size of the dataset. In addition, we use $EntIdx$ and $ObsIdx$ to facilitate the checking of unique entity and observation instances. Let m be the number of distinct keys, each checking could take $O(\log(m))$ time. So, in total, the algorithm runs in $O(n \log(m))$ where $m \ll n$ generally.

Space: $EntIdx$ and $ObsIdx$ are the intermediate structures that we use in the algorithm. Since they keep the distinct entity and observation key values. The space complexity is of $O(m)$.

Algorithm 1 MaterializeDB (*Dataset, A.**)

```
/* Dataset: [Input] in the form of a flat file */
/* A : [Input] Annotations*/

ObsIdx =  $\emptyset$ ; /* Keep index  $\langle ObsTypeId, KeyVal \rangle \rightarrow ObsId$  */
EntIdx =  $\emptyset$ ; /* Keep index  $\langle ObsTypeId, KeyVal \rangle \rightarrow EntId$  */

for (each Row  $\langle A_1, A_2, \dots, A_n \rangle \in Dataset$ ) do
  /* Step 1: Define measurement instances */
  MeasSet = CrtMeasurement(Row, A.*);

  /* Step 2: Partition the measurement instances according to observation types */
  ObsType2MeasIdx = PartMeas(MeasSet, A.*); /* ObsType2MeasIdx =  $\{ObsTypeId \rightarrow \{mi\}\}$  */

  ContextIdx =  $\emptyset$ ; /* Keep index  $ObsTypeId \rightarrow ObsId$  to materialize context */
  for (each ObsTypeId in ObsType2MeasIdx) do
    /* Step 3: Find or create the entity instance for each observation type partition */
    EntId = MaterializeEntity(ObsTypeId, ObsType2MeasIdx, EntIdx, A.*, OBOE.*);

    /* Step 4: Find or create the observation instance for each observation type partition of entity EntId */
    MaterializeObs(ObsTypeId, EntId, ObsType2MeasIdx, ObsIdx, ContextIdx, A.*, OBOE.*);
  end for

  /* Step 5: Assign the context observation instances */
  MaterializeContext(ContextIdx, A.*, OBOE.*);
end for
return OBOE;
```

Algorithm 2 CrtMeasurement (*Row, A.**)

```
/* Create new orphan measurement instances */
MeasSet =  $\emptyset$ ; /* Keep the set of new measurement instances */
for (each  $m = \langle MeasTypeId, ResAttribute, Cond, Val \rangle \in A.Map$ ) do
  if  $((m.ResAttribute! = Row.A_i.Attrname) \text{ OR } (Row.A_i \text{ does not satisfy } m.Cond))$  continue;
   $mi_{id} = \text{GetNewMeasId}(OBOE.Measurement)$ ;
  if  $(m.Val! = NULL)$  MeasVal =  $m.Val$ ;
  else MeasVal =  $Row.A_i.Val$ ;
  Create a measurement instance  $\langle mi_{id}, null, MeasTypeId, MeasVal \rangle$  and add it to MeasSet;
end for
return MeasSet;
```

Example 3.1 (Example with “key yes” and “distinct yes”, without “identifying yes”) *Take the*

Algorithm 3 PartMeas (*MeasSet, A.**)

```
/* Partition measurement instances according to their observation types */
ObsType2MeasIdx =  $\emptyset$  /* Keep index for  $ObsTypeId \rightarrow \{mi\}$  */
for (each  $mi \in MeasSet$ ) do
  ObsTypeId = GetObsTypeId ( $A.MeasType, mi.MeasTypeId$ );
  Update ObsType2MeasIdx by changing the item  $ObsTypeId \rightarrow \{mi\}$ ;
end for
return ObsType2MeasIdx;
```

Algorithm 4 MaterializeEntity(*ObsTypeId*, *ObsType2MeasIdx*, *EntIdx*, *A.**, *OBOE.**)

```
KeyVal = GetObsTypeKeys (ObsTypeId, ObsType2MeasIdx);
HasKey = false;
if (ObsTypeId has key measurements OR is specified with distinct yes) HasKey = true;
EntType = GetObsEntityType (A.ObservationType, ObsTypeId);
CrtNewEntInst = true;
if (HasKey==true) then
  EntId = GetEntId(ObsTypeId, KeyVal, EntIdx);
  if (EntId! = NULL) CrtNewEntInst = false;
end if
if (CrtNewEntInst == true) then
  EntId = CrtEntId(EntType);
  Create an entity instance  $ei = \langle EntId, EntType \rangle$  and put  $ei$  to  $OBOE.Entity$ ;
  if (HasKey==true) EntIdx = EntIdx  $\cup \{ \langle ObsTypeId, KeyVal \rangle \rightarrow EntId \}$ ;
end if
return EntId;
```

Algorithm 5 MaterializeObs(*ObsTypeId*, *EntId*, *ObsType2MeasIdx*, *ObsIdx*, *ContextIdx*, *A.**, *OBOE.**)

```
KeyVal = GetObsTypeKeys (ObsTypeId, ObsType2MeasIdx);
IsObsDistinct = CheckIfObsDistinct(A.ObservationType, ObsTypeId);
CrtNewObsInst = true;
if (IsObsDistinct==true) then
  ObsId = GetObsId(ObsTypeId, KeyVal, ObsIdx);
  if(ObsId! = NULL) CrtNewObsInst = false;
end if
if (CrtNewObsInst == true) then
  Create an observation instance  $oi = \langle ObsId, EntId \rangle$  and put  $oi$  to  $OBOE.Observation$ ;
  if (IsObsDistinct==true) ObsIdx = ObsIdx  $\cup \{ \langle ObsTypeId, KeyVal \rangle \rightarrow ObsId \}$ ;
end if

/* Maintain the measurement instances for this observation instance */
miSet = GetMeasInst(ObsType2MeasIdx, ObsTypeId);
if (ObsId is a new one) then
  Set the  $obsId$  to each  $mi \in miSet$  so that  $mi$ -s are not orphans;
  Put all the  $mi \in miSet$  to  $OBOE.Measurement$ ;
else
  Discard all the  $mi \in miSet$ ;
end if
ContextIdx = ContextIdx  $\cup \{ ObsTypeId \rightarrow ObsId \}$ ; /* ContextIdx is also output*/
```

data in Table 2¹ as an example to explain the algorithm.

For Row(2007, 1, piru, 35.8)

- Step 1 creates four measurement instances: $\langle mi_1, null, Year, 2007 \rangle$, $\langle mi_2, null, DBH, 35.8 \rangle$,

¹The detailed annotation is in page 6 of Shawn's powerpoint file.

yr	spec	spp	dbh
2007	1	piru	35.8
2008	1	piru	36.2
2008	2	abba	33.2

Table 2: Dataset 1

Algorithm 6 `MaterializeContext`(*ContextIdx*, *A.**, *OBOE.**)

```
for (ObsTypeId → ObsId ∈ ContextIdx) do
  ContextObsTypeId, Rel = GetContextObsTypeRel(A.ContextType, ObsTypeId);
  if (ContextObsTypeId! = NULL) then
    ContextObsId = GetContextObsId(ContextIdx, ContextObsTypeId);
    Create a context instance ci = (ObsId, ContextObsId, Rel);
    Put ci to OBOE.Context;
  end if
end for
```

$\langle mi_3, null, TaxonomicTypeName, Picea\ rubens \rangle$, $\langle mi_4, null, EntityName, 1 \rangle$,
and returns $MeasSet = \{mi_1, mi_2, mi_3, mi_4\}$; ².

- Step 2 returns $ObsType2MeasIdx = \{o_1 \rightarrow \{mi_1\}, o_2 \rightarrow \{mi_2, mi_3, mi_4\}\}$.
- Step 3-4: for observation types o_1 and o_2 , materialize entity and observation instance
 - for o_1 (with associated instance mi_1 of type m_1)
 - * Since m_1 is specified as key, get the $KeyVal = 2007$;
 - * No entity with this key exists in *EntIdx*, create an entity $\langle ei_1, TemporalRange \rangle$; Now, $EntIdx = \{\langle o_1, 2007 \rangle \rightarrow ei_1\}$.
 - * Since o_1 is specified as distinct, need to make sure we do not create redundant observation instances. No entry with the key $\langle o_1, 2007 \rangle$ exists in *ObsIdx*, so, create an observation instance oi_1 , which is of entity ei_1 and represented as $\langle oi_1, ei_1 \rangle$.
Now, $ObsIdx = \{\langle o_1, 2007 \rangle \rightarrow oi_1\}$
 - * Connect mi_1 to oi_1 ;
 - When deal with o_2 ,
 - * $KeyVal = 1$.
 - * Create an entity instance $\langle ei_2, Tree \rangle$; $EntIdx = \{\langle o_1, 2007 \rangle \rightarrow ei_1, \langle o_2, 1 \rangle \rightarrow ei_2\}$.
 - * Create an observation instance $\langle oi_2, ei_2 \rangle$. No need to update *ObsIdx* because o_2 is not identified as distinct.
 - * Connect mi_2, mi_3 and mi_4 to oi_2 ;
- Step 5 assigns the context relationship between oi_1 and oi_2 ;

For Row (2008, 1, piru, 36.2)

- Step 1 creates measurement instances $\langle mi_5, null, Year, 2008 \rangle$, $\langle mi_6, null, DBH, 36.2 \rangle$,
 $\langle mi_7, null, TaxonomicTypeName, Picea\ rubens \rangle$, $\langle mi_8, null, EntityName, 1 \rangle$
and returns $MeasSet = \{mi_5, mi_6, mi_7, mi_8\}$;
- Step 2 gets $ObsType2MeasIdx = \{o_1 \rightarrow \{mi_5\}, o_2 \rightarrow \{mi_6, mi_7, mi_8\}\}$
- Step 3-4: for observation types o_1 and o_2 materialize entity and observation instance
 - for o_1
 - * $KeyVal = 2008$;
 - * Create an entity instance $\langle ei_3, TemporalRange \rangle$;
 $EntIdx = \{\langle o_1, 2007 \rangle \rightarrow ei_1, \langle o_2, 1 \rangle \rightarrow ei_2, \langle o_1, 2008 \rangle \rightarrow ei_3\}$.

²For all the instances, the measurement characteristic is set to represent Measurement Type

- * Create an observation instance $\langle oi_3, ei_3 \rangle$;
 $ObsIdx = \{\langle o_1, 2007 \rangle \rightarrow oi_1, \langle o_2, 1 \rangle \rightarrow oi_2, \langle o_1, 2008 \rangle \rightarrow oi_3\}$
- * Connect mi_5 to oi_3 ;
- When deal with o_2 ,
 - * $KeyVal = 1$.
 - * **item $\langle o_2, 1 \rangle \rightarrow ei_2$ is already in $EntIdx$, so get the entity id ei_2 . No need to create an entity.**
 - * Since o_2 is not specified with distinct yet, we *NEED* to create an observation $\langle oi_4, ei_2 \rangle$. No need to update $ObsIdx$.
 - * Connect mi_6, mi_7, mi_8 to oi_4 ;

For **ROW (2008, 2, abba, 33.2)**

- For o_1 's measurement 2008,
 - Since $\langle o_1, 2008 \rangle \rightarrow ei_3$ already exists in $EntIdx$, **no need to create a new entity.**
 - Since o_1 is specified with distinct yes, and $\langle o_1, 2008 \rangle \rightarrow oi_3$ already exists in $ObsIdx$, **no need to create a new OBSERVATION** and no need to put the measurement instance for 2008 into OBOE model.

plt	spp	dbh
A	piru	35.8
A	piru	36.2
B	piru	33.2

Table 3: Dataset 2

Example 3.2 (Example with identifying) Let us use the data in Table 3 as an example.³ Before we go through the algorithm step by step, we first note that o_1 and o_2 have key measurements m_1 and m_2 respectively. So, we need to maintain the distinct entity instances for both of these two observation types. In addition, o_1 is specified with distinct yes while o_2 is not. So, we need to maintain the distinct observation instances for o_1 but not for o_2 .

For the **first row**,

- The first step generates three measurement instances $MeasSet = \{\langle mi_1, null, EntityName, A \rangle, \langle mi_2, null, TaxonomicTypeName, Picea\ rubens \rangle, \langle mi_3, null, DBH, 35.8 \rangle\}$.
- The second step gets $ObsType2MeasIdx = \{\{o_1 \rightarrow \{mi_1\}, o_2 \rightarrow \{mi_2, mi_3\}\}$.
- For each observation type, create entity and observation instances.
 - For o_1 , the key value is A. Since there is no such a key in $EntIdx$, we create an entity ei_1 of type Plot.
 $EntIdx = \{\langle o_1, A \rangle \rightarrow ei_1\}$.
 We create an observation instance oi_1 whose entity is ei_1 .
 $ObsIdx = \{\langle o_1, A \rangle \rightarrow oi_1\}$.
 Connect the measurement instance mi_1 to observation instances oi_1 ,

³The detailed annotation information is at page 8 in Shawn's powerpoint file.

- For o_2 , the key value is $(A, Picea\ rubens)$ since it has context o_1 with “identifying yes”.
We create an entity instance ei_2 of type *Tree*.
 $EntIdx = \{\langle o_1, A \rangle \rightarrow ei_1, \langle o_2, (A, Picea\ rubens) \rangle \rightarrow ei_2\}$.
We create an observation instance oi_2 whose entity is ei_2 .
Connect the measurement instances mi_2 and mi_3 to observation instance oi_2 .

- The last step for this row is to connect the observations using context relationship. For this instance, we connect oi_1 to oi_2 with context “Within”.

For the **second row**,

- The first step defines three measurement instances $MeasSet = \{\langle mi_4, null, EntityName, A \rangle, \langle mi_5, null, TaxonomicTypeName, Picea\ rubens \rangle, \langle mi_6, null, DBH, 36.2 \rangle\}$.
- The second step gets $ObsType2MeasIdx = \{\{o_1 \rightarrow \{mi_4\}, o_2 \rightarrow \{mi_5, mi_6\}\}$.
- For each observation type, create entity and observation instances.
 - For o_1 , the key value is $\langle o_1, A \rangle$, $EntIdx$ already has an item for it with entity instance ei_1 . No need to create a new instance for it.
To create observation instance, since o_1 is specified with “distinct yes” and the key value is $\langle o_1, A \rangle$, which corresponds to an existing observatin instance oi_1 . So, we do not need to create a new observation for it.
When we try to connect the measurement instance mi_4 to observation instance, we realize that we did not create a new observation instance for type o_1 . So its related measurement instance mi_4 can be discarded.
 - For o_2 , the new key value is $\langle o_2, (A, Picea\ rubens) \rangle$, which corresponds to ei_2 in $EntIdx$, so no need to create a new instance for it either.
To create observation instances, since no “distinct yes” is specified, we create a new observation instance oi_3 for it. Then, we conctnct the measurement instances $\{m_5, m_6\}$ to observation instance oi_3 .

When we process the **third row**, we have a new key value $\langle o_1, B \rangle$ for o_1 , thus we create a new entity instance for it. For o_2 , we have new key value $\langle o_2, (B, Picea\ rubens) \rangle$ and create a new entity instance for it. Similarly, we need to create new observation instances for both type.