Online Appendix to: Generalizing Database Forensics

KYRIACOS E. PAVLOU and RICHARD T. SNODGRASS, University of Arizona

This appendix presents a step-by-step discussion of the *forensic analysis protocol* that allows us to identify many types of corruption events.

A. THE FORENSIC ANALYSIS PROTOCOL

Figures 16–19 show different parts of the formulation of the forensic analysis protocol (broken across four diagrams). We will discuss Page Write Timestamp corruption (on disk), Attribute corruption (on disk), and Schema corruption (on disk) using the the four figures as a guide.

Data corruptions which affect an entire tuple will not be discussed since no tools exist right now that adequately analyze these problems, even though we can detect that corruption is present. In general, the identification of leaves for which no forensic tools exist (marked with "Pending Future Work") is part of our future work.

After the section devoted to the common starting path of the forensic protocol, each of the following sections corresponds to the leaves of the above-listed major corruption subtypes. We emphasize that this protocol assumes *a single corruption event*. A diagram depicting the taxonomy and protocol together can be accessed at `http://www.cs.arizona.edu/projects/tau/dragoon/taxonomy_protocol.pdf`.

The upper portion of the figures show portions of the UML taxonomy of corruption events (in shaded rectangles). Everything appearing below the taxonomy depicts the steps taken and the tools used in the forensic analysis protocol, in order to reach the leaves in the taxonomy.

Within the lower position of the protocol are *conditionals* in diamonds that denote questions distinguishing between variations of the same corruption subtype, *directives* in (unshaded) rectangles that indicate some computational task to be performed, and *connectors* in ellipses that connect the different parts of the protocol using unique labels. *Observables* are the results of conditionals and of the forensic analysis algorithms.

Recall that forensic analysis is a one-to-one correspondence between the observables and the elements in the taxonomy. Such a strong correspondence might not be possible to establish, either because of lack of tools (in this case the forensic analysis algorithms) or due to the nature of the problem. Hence, we characterize forensic analysis as a mapping from observables to subsets of the taxonomy, which we term *conclusion sets*. Elements belonging to the same conclusion set are indistinguishable from one another given the available tools. Conclusion sets are shown in parallelograms.

A legend in Figure 16 summarizes this useful information. The flow of analysis in the protocol is generally upward, as indicated by the arrows.

A.1. Initial Steps in Forensic Analysis Protocol

Figure 16 shows that the protocol starts at the bottom of the diagram. The first question to be asked is whether the DBA has chosen a database scheme in which records are physically clustered by their partition field. If the records are clustered then in addition to the regular validation procedure we must check the order of records during the scan. This result will be useful later on in the protocol (vid. Figure 18).

DOI: http://dx.doi.org/10.1145/2487259.2487264

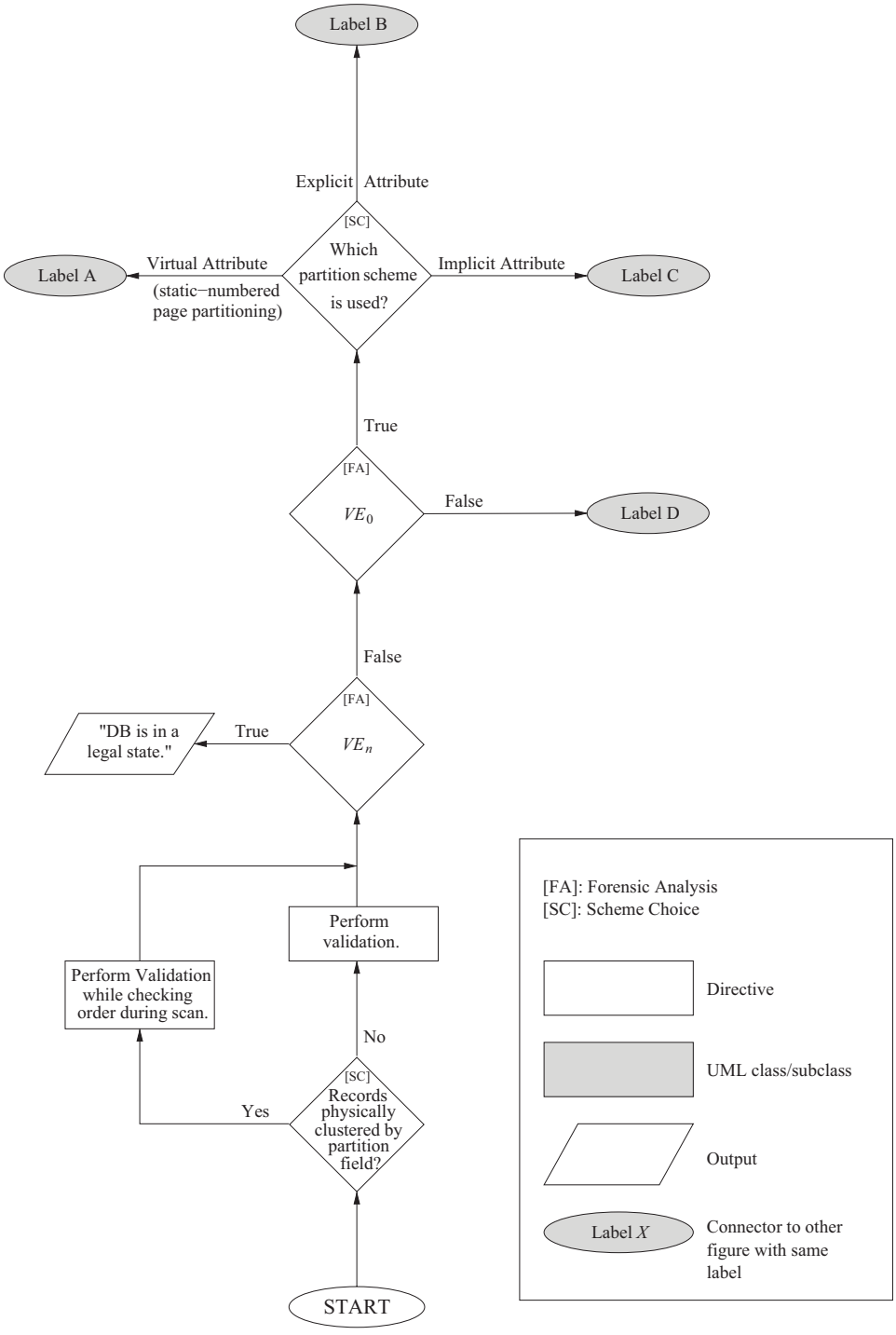

Fig. 16. Initial steps in forensic analysis protocol.

If the most recent Validation Event returns true then the total hash chain has not been compromised. This implies that the database is not corrupted and it is in a legal state. If the Validation Event returns false, meaning that the total chain has been corrupted, then we need to ascertain whether this is due to Schema corruption. This is achieved by validating the notarized hash value corresponding to the database schema. If VE_0 is false, thus revealing schema corruption, we must follow Label D into Figure 19 to identify the type of schema corruption. The discussion for this can be found in Section A.4.

If VE_0 is true then no schema corruption has occurred. Before we can proceed and execute the forensic analysis algorithm (in this case, the various types of a3D) we must choose the appropriate one based on the particular partitioning scheme used.

Labels A, B, and C lead to the discussion of use and results of the forensic analysis algorithm, for a Virtual Attribute (static-numbered pages), an Explicit Attribute (zip code), or an Implicit Attribute (commit-time) partitioning scheme, respectively. The first label connects to Figure 17 with further discussion found in Section A.2, while the latter two connect to Figure 18 with further discussion found in Section A.3.

As mentioned earlier, despite running a forensic analysis algorithm we have no way of ascertaining whether an entire tuple on disk has been corrupted, that is, we do not yet have tools that could distinguish it from other types of corruption.

A.2. Corruption of Page Write Timestamps on Disk

Figure 17 provides the portion of the protocol required for the identification of corrupted Page Write Timestamps when these are stored on disk as per the static-numbered page-based partition scheme. Even though this is a type of corruption which occurs under a Virtual Attribute partitioning scheme it does not affect the partition attribute itself, namely the page number. In fact, in the case of static-numbered page-based partition scheme, the page number is not explicitly stored anywhere and therefore cannot be corrupted. The algorithm can determine the page number just by looking at what page the tuples resides in. A genuine Virtual Attribute corruption can occur in the non-static-numbered-page partitioning scheme but currently there are no tools to analyze such a corruption (vid. Section A.3 for more details).

After ascertaining that static-numbered page partition scheme is used (vid. Figure 16) we follow Label A into Figure 17. The Page-based a3D Forensic Analysis Algorithm is run, which could yield one or two resulting corruption regions. If a single corruption region is returned then we conclude that non-partition attribute has been corrupted. If two corruption regions are returned then a Page Write Timestamp residing on a page on disk has been corrupted. Changing the timestamp will cause the associated data to be hashed in the wrong order during the validation scan. This causes the creation of two corruption regions: one corresponds to the "missing" page (i.e., the position where the page ought to have been hashed) and the other to the "new" page (i.e., the position where the page is actually hashed). If the timestamp is changed to later time the page will be hashed at a later stage in the validation scan, while if it is changed to an earlier time the page will be hash at an earlier stage. The former corruption is termed postdating and the latter backdating. Currently, we have no tools to distinguishing between the two types of corruption. (Physical clustering cannot help us here because we cannot cluster the pages by page write time since the same page—albeit with different contents—is written out to disk multiple times.) We are forced to resort to the backup tapes which will allow us to determine the direction of corruption. (Even though there might be ways to find the direction of timestamp corruption without making use of the backup tapes.) Discovering the original correct timestamp and comparing it with the corrupted one will allow us to decide if the timestamp has been Backdated or Postdated.

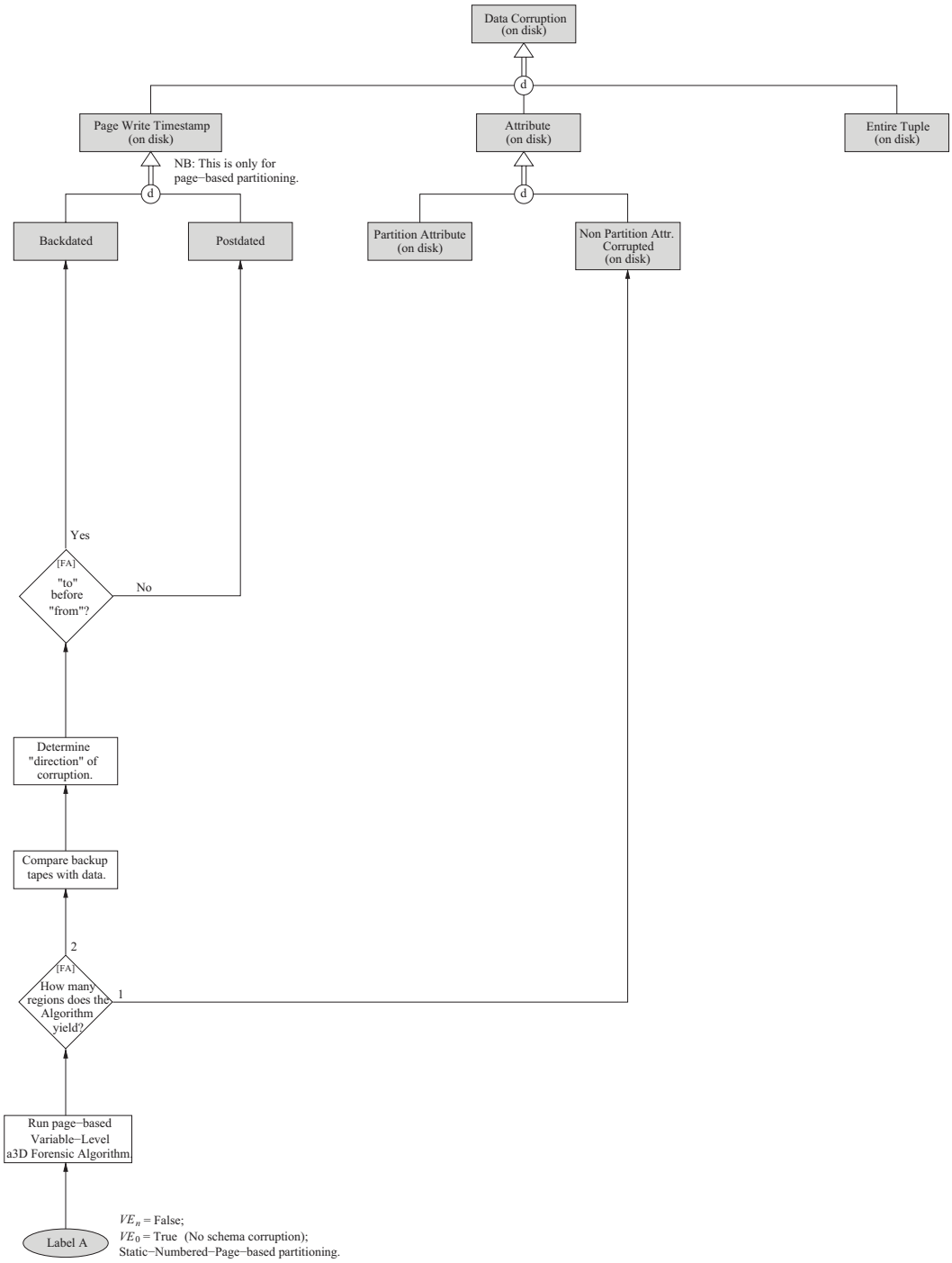

Fig. 17. Forensic analysis protocol to identify Page Write Timestamp corruption (on disk).

A.3. Corruption of Partition Attribute on Disk

Figure 18 shows the portion of the forensic analysis protocol which allows for the identification of the corruption event that affects partition attributes on disk; the attribute happens to be one that the data are partitioned on. This leads to three possibilities depending on the nature of the partition attribute, each with a different forensic analysis technique applicable: Virtual attribute-based (non-static-numbered page-based), Explicit Attribute-based (what previous sections called simply attribute-based), and Implicit attribute (commit-time-based).

Paged-based data partitioning with non-statically numbered pages presents an interesting challenge. For example, in the case of τBerkeleyDB [Snodgrass et al. 2004], the records are stored in the leaf nodes of a B+-tree index. The B+-tree invariant dictates that leaves must be at least half full, a fact which causes page splits. This results in records migrating from the page they were originally stored in and consequently in the page number of a specific record changing over time. An effective forensic analysis algorithm for non-static-numbered page partitioning scheme has yet to be designed.

Following Label C into Figure 18 we first run Commit-time-based a3D Forensic Analysis Algorithm. We know we are partitioning on an Implicit Attribute (e.g., commit-time timestamps) so then next step is to check how many corruption regions the forensic analysis algorithm created. If it returned only one then we can safely say that the corruption affected an attribute the data are not partitioned on. Elsewhere [Pavlou and Snodgrass 2008] this was termed a *data-only corruption* and could be either *retroactive* or *introactive*.

If two corruption regions are produced by the algorithm then this implies that an implicit attribute (timestamp) has been corrupted since a single corruption of a non-partition attribute cannot yield two corruption regions. Now we must distinguish between a corruption which results in a backdated timestamp and one which results in a postdated timestamp. To achieve this we can impose a spatial organization on the database by exploiting the physical order on the data. Data can be stored in contiguous segments of the disk in order of commit time or of any attribute whose values/granules are naturally ordered, for instance, age, transaction ID, page number, primary key, or according to any order imposed *ad hoc*. This is essentially a heap where new records are appended at the end.

Let us consider a Backdating CE. Backdating is *in effect* the removal of a tuple from its original location on the disk and its subsequent introduction into a different location. Changing only the timestamp, with no physical movement of the tuple, will render the tuple to be out of order, something that is evidence of corruption. Therefore, to answer the "to" before "from" conditional we use the result of checking the order of records during the Validation scan (cf. initial steps of Figure 16). This is feasible because changing only the timestamp, with no physical movement of the tuple, will render the tuple to be out of order with respect to the imposed physical order, something which is evidence of corruption. A tuple with an earlier timestamp than the timestamps of the proceeding tuples is evidence of backdating. Conversely, a tuple with a later timestamp than the timestamps of the succeeding tuples is evidence of postdating.

If the data are not physically clustered then we must compare the current data with the data in the backup tapes to determine the "direction" of corruption. If the new value of the changed implicit attribute (timestamp) refers to a time before the original value then we the corruption resulted in a backdated timestamp. Otherwise, we have identified a postdated timestamp.

The path starting at Label B, to identify a corruption under an explicit attribute partitioning scheme, is very similar to that dealing with implicit attribute corruption. The only two differences are first, the need to identify which of the total chains

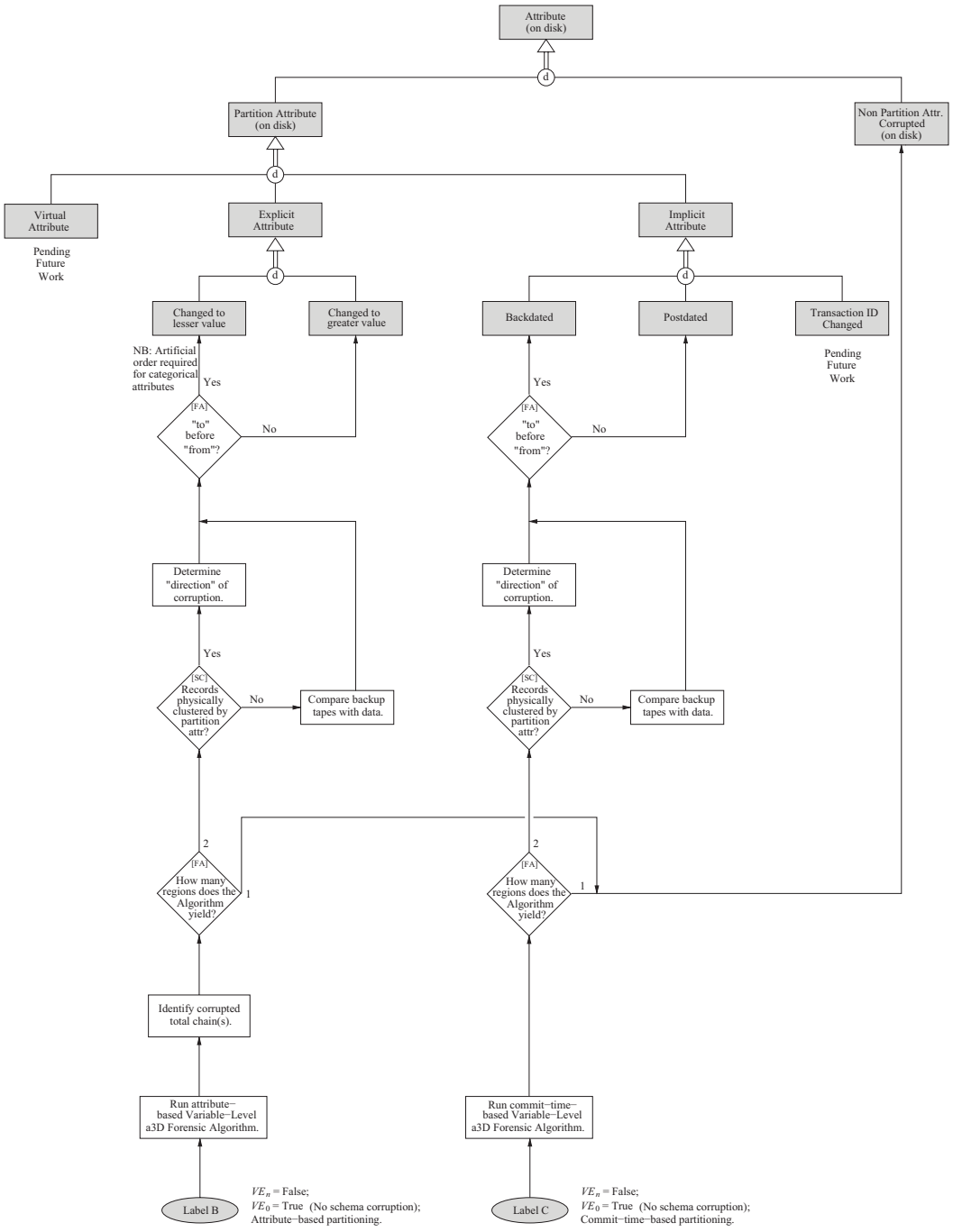

Fig. 18. Forensic analysis protocol to identify Attribute Corruption (on disk).

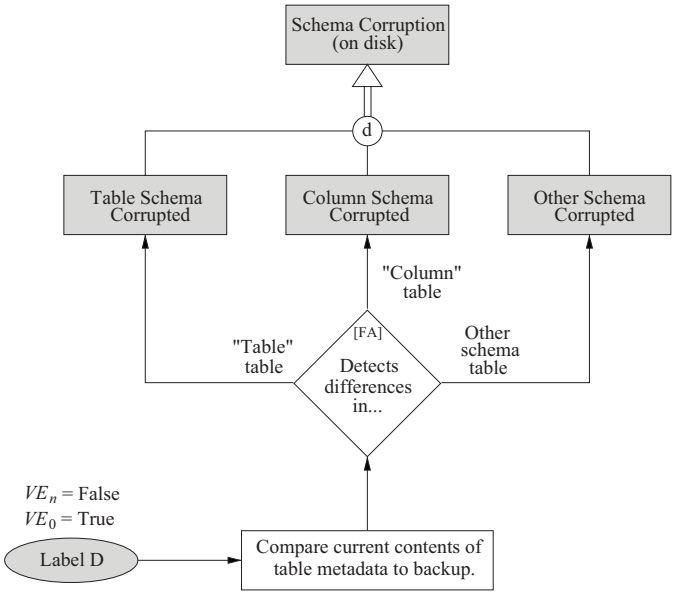

Fig. 19. Forensic analysis to identify Schema Corruption (on disk).

(if multiple are present) are the affected ones and second, the need to have an artificial order imposed on the partitioning attribute (if it is categorical) when implementing the physical clustering of tuples on disk.

Finally, note that for Virtual Attribute corruption to occur (and indeed, corruption of any kind of attribute) the attribute values must themselves be susceptible to corruption. This is true in the case of non-static-numbered-paged-based partitioning of the data where the page number associated with a specific tuple changes with time. Therefore, the page number where the tuple was *originally* stored in must be maintained in some way. This renders these maintained original page numbers susceptible to corruption. However, this is not an issue when the Virtual attribute we are partitioning on is static-numbered-pages. In this case the page numbers cannot be corrupted since they are not explicitly maintained in any way. This is because at each point in time we know the page number associated with a specific tuple by virtue of looking at what page that tuple resides in.

A.4. Forensic Analysis of Schema Corruption

One of the issues left unaddressed from earlier work in forensic analysis was how to deal with schema corruption. The initial idea was that before the database went online the schema of the database would be hashed and notarized (NE_0). If during forensic analysis the validation of NE_0 produced a false result then we would know that the schema of the database has been compromised.

As shown in Figure 19, we can deal with schema corruption by comparing the current contents of the metadata table to the backup. Depending on the detected differences we can identify a corruption to the Table or Column tables, or any other piece of metadata.