Causality in Databases*

Alexandra Meliou[1], Wolfgang Gatterbauer[1], Joseph Y. Halpern[2], Christoph Koch[2], Katherine F. Moore[1], and Dan Suciu[1]

[1]University of Washington
{ameli,gatter,kfm,suciu}@cs.washington.edu

[2]Cornell University
{halpern,koch}@cs.cornell.edu

Abstract

Provenance is often used to validate data, by verifying its origin and explaining its derivation. When searching for "causes" of tuples in the query results or in general observations, the analysis of lineage becomes an essential tool for providing such justifications. However, lineage can quickly grow very large, limiting its immediate use for providing intuitive explanations to the user. The formal notion of causality *is a more refined concept that identifies causes for observations based on user-defined criteria, and that assigns to them gradual degrees of responsibility based on their respective contributions. In this paper, we initiate a discussion on causality in databases, give some simple definitions, and motivate this formalism through a number of example applications.*

1 Introduction

The notion of causality and causation is a topic in philosophy, studied and argued over by philosophers over the centuries. A formal, mathematical study of causation and responsibility has been initiated recently by the work of Pearl [Pea00] and Halpern and Pearl [HP05], who have distilled the generally accepted aspects of causality into a rigorous definition. This work has been influential in the Computer Science research community and has already lead to applications in model checking and verification [BBDC$^+$09, CHK08, CGY08].

The goal of this paper is to initiate a discussion on causality in databases. In Sec. 2, we give a minimalistic definition of query causality that still captures the key concepts in the HP definition: (1) counterfactual v.s. actual cause, (2) the notion of a contingency set, (3) Chockler and Halpern's notion of responsibility [CH04]. We also show its relation to provenance, also refered to as *lineage* in this paper. Then in Sec. 3, we show that our definition is general enough to be applied to a broad class of database-related applications.

2 Definitions

Causality in databases aims to answer the following question: *given a query over a database instance and a particular output of the query, which tuple(s) in the instance caused that output to the query?* An important

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*This work was partially supported by NSF III 0911036.

contribution of Halpern and Pearl's definition of causality is its reliance on *structural equations*, which express the notion of *intervention*: what happens to the outcome if we change the state of the input. As a starting point, we propose to define causality in databases by using a single structural equation: the lineage of that answer to the query. We first define causality without referring to lineage, then relate the two notions.

2.1 Causality

Following Halpern and Pearl (HP from now on), we partition the tuples in the database into *exogenous* and *endogenous* tuples. The endogenous tuples are those that are considered candidates for causes. Exogenous tuples are deemed not to be possible causes: they define the context determined by external factors. This partition depends on the particular application and is defined either by the system or by the user. Let $D^{\mathrm{n}} \subseteq D$ denote the *endogenous tuples*; the *exogenous tuples* are then $D^{\mathrm{x}} = D - D^{\mathrm{n}}$. For each relation name R_i, we write R_i^{n} and R_i^{x} to denote the endogenous and exogenous tuples in R_i, respectively. Thus, $R_i = R_i^{\mathrm{n}} \cup R_i^{\mathrm{x}}$.

Our definition of causality simplifies the HP definition while still retaining its two key elements, *counterfactual causes* and *contingencies*.

Definition 1 (DB Causality): Let $t \in D^{\mathrm{n}}$ be an endogenous tuple, and r a possible answer to query q.

- t is called a *counterfactual cause* for r in D if $D \models q(r)$ and $D - \{t\} \not\models q(r)$.
- $t \in D$ is called an *actual cause* for r if there exists a set $\Gamma \subseteq D^{\mathrm{n}}$, called a *contingency* for t, such that t is a counterfactual cause for r in $D - \Gamma$.

A tuple t is a counterfactual cause if its removal from the database also removes r from the query result. The tuple is an actual cause if one can find a contingency for which the tuple becomes a counterfactual cause: more precisely, one has to find a set of tuples Γ such that, after removing Γ from the database we bring the database to a state where removing/inserting t causes r to switch between an answer and a non-answer. The set Γ is called a contingency for t. Obviously, every counterfactual cause is also an actual cause, by taking $\Gamma = \emptyset$.

Our simplified definition of causality is analogous to the definition for causality in Boolean circuits used in [CHK08]. For monotone queries, it can be shown that this simplified definition agrees with the full HP definition. For example, one requirement that we dropped is the following restriction on the contingency Γ: for any $\Gamma' \subseteq \Gamma$, it is required that $D - \Gamma' \models q(r)$. In other words, if we keep the candidate cause t in the database, then any subset of the contingency should not change the output to the query. If the query is monotone then this condition is automatically satisfied by our simple definition (as $D - \Gamma \models q(r)$ implies $D - \Gamma' \models q(r)$). However, for non-monotone queries, our simple definition differs from the HP definition and may be inadequate. For example, consider a non-monotone query over a database containing three tuples t_1, t_2, t. The query evaluates to true if both t_1 and t_2 are in the database, or if t is in the database and neither of t_1, t_2 are; in other words, its lineage is $(X_{t_1} \wedge X_{t_2}) \vee (\neg X_{t_1} \wedge \neg X_{t_2} \wedge X_t)$ (see Sec. 2.2). In our simplified definition, t is an actual cause with contingency $\Gamma = \{t_1, t_2\}$ (as removing t_1, t_2 from the database makes t counterfactual). Under the full HP definition, this contingency is not allowed because the query becomes false by just removing t_1 from the database. For a preliminary attempt towards a more general adaptation of the HP definition in a database context, see [MGMS09].

2.2 Relating Causality and Lineage

We can now relate causality and lineage. We define lineage as a Boolean expression, specifically based on the Boolean c-tables representation [GT06, GKT07]. Associate a Boolean variable X_t to each tuple t in the database instance D. The *lineage* of an answer r to a query q is defined as the unique (up to logical equivalence) Boolean expression $\varphi^{q,r,D}$ such that, for every truth assignment θ of X_t, $\varphi^{q,r,D}[\theta] = \texttt{true}$ iff the query q returns the answer r when restricted to the subset $D_0 = \{t \mid \theta(X_t) = \texttt{true}\} \subseteq D$. For example, the lineage of the answer c to the query $q(x) :- R(x, y), S(y)$ might be $(X_{R(c,b)} \wedge X_{S(b)}) \vee (X_{R(c,f)} \wedge X_{S(f)})$, which says that c will be an answer to q on any subset of the database that contains the tuples $R(c, b)$ and $S(b)$, or the tuples $R(c, f)$ and $S(f)$.

The lineage expression gives us a simple tool for computing all causes for an answer r to a monotone query q. Write the lineage $\varphi^{q,r,D}$ in DNF, then substitute all Boolean variables corresponding to exogenous variables with `true`; call the result ψ. A *minterm* of ψ is a minimal set of tuples T such that $\bigwedge_{t\in T} X_t \Rightarrow \psi$. Then every tuple that occurs in a minterm of ψ is a cause for the answer r.

Proposition 1 (Lineage Minterms): Let $\varphi^{q,r,D}$ be the lineage expression for an answer r to a monotone query q over a database instance D. Let $\psi^{q,r,D}$ be obtained from $\varphi^{q,r,D}$ by replacing all Boolean variables for exogenous tuples with `true`. Then tuple t is an actual cause of the answer r to the query q on D iff there exists a minterm in $\psi^{q,r,D}$ that contains t.

For example, if $\psi = (X_{t_1} \wedge X_{t_2} \wedge X_{t_3}) \vee (X_{t_1} \wedge X_{t_3} \wedge X_{t_4}) \vee (X_{t_1} \wedge X_{t_2} \wedge X_{t_3} \wedge X_{t_5})$, then there are two minterms, $\{t_1, t_2, t_3\}$ and $\{t_1, t_3, t_4\}$, and the actual causes are t_1, t_2, t_3, t_4; the tuple t_5 is not a cause because the third term in ψ is redundant. In the case where all tuples are endogenous, Proposition 1 eliminates terms that do not appear in the *minimal witness basis* of the answer [BKT01], and therefore the set of actual causes is equivalent to the union of all sets in the minimal witness basis.

2.3 Degree of Responsibility

To measure the degree to which a tuple is considered a cause, Chockler and Halpern [CH04] introduced the notion of *degree of responsibility*, or simply *responsibility*. We present the intuition with a simple example: Assume 11 voters who can vote for either candidate A or candidate B. In an 11-0 outcome, each voter is a cause for A's win, but each one's responsibility is less than in a 6-5 outcome. The following definition, an adaptation from Chockler and Halpern [CH04], captures this intuition.

Definition 2 (DB Responsibility): Let r be an answer to a query q, and let t be a cause. With Γ ranging over all possible contingencies for t, the *responsibility* of t for the answer r is $\rho_t = (1 + \min_\Gamma |\Gamma|)^{-1}$

Thus, the responsibility of t is a function of the minimal number of tuples that we need to remove from the real database D before t becomes counterfactual. The tuple t is a counterfactual cause iff $\rho_t = 1$. If t is not a cause then, by convention, we take $\rho_t = 0$. Intuitively, the larger the contingency, the less counterfactual a variable is, and thus its responsibility is lower. In the case of the 6-5 vote, each vote for A is critical, so it has degree of responsibility 1. On the other hand, in the case of the 11-0 vote, each vote has degree of responsibility $1/6$; 5 votes have to be changed before one becomes critical. Some first results on the complexity of computing causality and responsibility for the case of conjunctive queries can be found in [MGMS11].

Constraints. The definitions of causality and responsibility should be revisited in the presence of known constraints over the data. Take for example $q(x) :- R(x, y, z), S(x, y, u)$, with the constraint $R(x, y) \subseteq S(x, y)$. Under this constraint, q is equivalent to $q'(x) :- R(x, y, z)$, and yet computing responsibility over q and q' would yield different results. Constraints are beyond the scope of this paper, but important to consider in future work.

3 Applications of Causality and Responsibility in Databases

3.1 Explaining Unexpected Answers

We start with the most basic application: explaining unexpected answers to a query. Consider the IMDB movie database, available from `www.imdb.org`. It consists of the six tables illustrated in Fig. 1. Tim Burton is an Oscar-nominated director well known for fantasy movies involving dark, Gothic themes. Examples of his work include "Edward Scissorhands", "Beetlejuice", and the recent "Alice in Wonderland". A user wishing to learn more about Burton's movies queries the IMDB dataset to find out about the genres of movies that he has directed; the query issued, shown in Fig. 1, retrieves all genres of movies directed by Burton. The query's answer, ran on the actual database and shown in Fig. 1, includes some expected genres, such as Fantasy and Horror, and

Database Schema

Director(*did*, *firstName*, *lastName*)
Actor(*aid*, *firstName*, *lastName*)
Movie(*mid*, *name*, *year*, *rank*)
Movie_Directors(*did*, *mid*)
Genre(*mid*, *genre*)
Casts(*aid*, *mid*, *role*)

Query

```
select    distinct g.genre
from      Director d, Movie_Directors md,
          Movie m, Genre g
where     d.lastName like 'Burton'
          and g. mid=m.mid
          and m. mid=md.mid
          and md. did=d.did
order by  g.genre
```

Query Answers

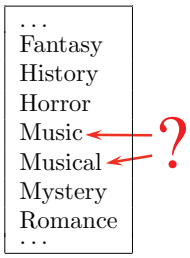


Figure 1: Example constructed with real data from the IMDB dataset. The user issues the query above to identify the movie genres that Burton has directed, and is surprised to see the categories `Music` and `Musical` listed. (Note that a movie may be classified under several genres.)

some genres that seem more suspicious, such as Music and Musical. Are the latter errors in the data or the query, or are they interesting discoveries? In other words, what are the causes of Musical and Music appearing in the query's answer?

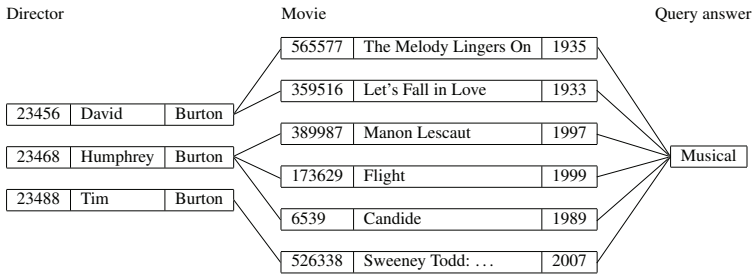
Tracing back the lineage of each answer to the query is a first step towards explaining them, but it is not sufficient. In combination, the lineage of these two answers consists of a total of 137 base tuples, and this, we argue, is overwhelming to a user. Representing this lineage as a list of minterms (44 in this case), still requires further analysis by the user to determine the frequency of a tuple's appearance in the lineage, as well as its participation in minterms of possibly different sizes, to determine its actual significance to the result. It also turns out that all 137 tuples are actual causes (by Prop. 1), so causality does not provide us with any more information than lineage in this case. However, ranking by responsibility gives us much more interesting information, as we now show.

Consider the case of the answer Musical, whose lineage is much smaller than Music and shown in Fig. 2a. [1] We can see immediately the causes of the suspicious answer. Although Tim Burton *did* direct one musical, "Sweeney Todd" (surprise!), there are two other directors with last name Burton (oops!) who directed several musicals. What we would like from a system is to list "Sweeney Todd", David Burton, and Humphrey Burton as top causes for Musical: the first is a surprising finding, the last two point to a bug in the query. We don't want a system to list a movie like "The Melody Lingers On" as a top explanation, because that doesn't really explain directly the surprising answer Musical. Ranking by responsibility achieves precisely this behavior, as shown in Fig. 2b. The movie "Sweeney Todd" has a minimum contingency of size two, consisting of the directors David and Humphrey Burton, because by removing these two directors from the database, "Sweeney Todd" becomes counterfactual. In addition, each of the three directors with last name Burton also has a contingency of size two, consisting of the other two directors. Thus, these four tuples (the movie and the three directors) have the same responsibility, $1/3$, and are listed as the top explanations for why Musical is returned by the query. On the other hand, the smallest contingency for the movie "Manon Lescaut" has size four,[2] hence its responsibility is 1/5. The answer Music has a much larger lineage consisting of 118 tuples; ranking them by responsibility proves to be a key technique in identifying interesting explanations. In this case Tim Burton and his movie "Corpse Bride" appear highest in the ranking, along with three more directors who share the same last name: Al, Tom, and Humphrey.

Discussion. Our example illustrates a general phenomenon. Tuples with high responsibility tend to be interesting explanations to query answers; tuples with low responsibility tend to be uninteresting. The reason for why a tuple is interesting may vary: it may be a surprising fact in the data, it may point to a bug in the query, or it may represent an error in the data. Our simple definition of causality does not distinguish between these explanations, but only ranks them by the degree of interestingness to the user. A challenging research direction

[1] The lineage has 21 tuples. Tuples from the `Movie_Directors` and `Genre` relations are depicted in Fig. 2a as edges between the `Director` and `Movie` tuples and the `Movie` and answer tuples, respectively. It is natural to consider the latter exogenous; if they were endogenous, they would have the same responsibility as the movie they correspond to.

[2] {`Movie`("Candide"), `Movie`("Flight"), `Director`(David Burton), `Director`(Tim Burton)}.



(a) Lineage from `Directors` and `Movies` of the Musical tuple

Answer tuple	ρ_t
Movie(526338, "Sweeney Todd", 2007)	0.33
Director(23456, David, Burton)	0.33
Director(23468, Humphrey, Burton)	0.33
Director(23488, Tim, Burton)	0.33
Movie(359516, "Let's Fall in Love", 1933)	0.25
Movie(565577, "The Melody Lingers On", 1935)	0.25
Movie(6539, "Candide", 1989)	0.20
Movie(173629, "Flight", 1999)	0.20
Movie(389987, "Manon Lescaut", 1997)	0.20

(b) Responsibility rankings for Musical

Figure 2: Ranking causes in the IMDB example. Displayed are the results for the Musical tuple, assuming that all the tuples in the relations `Directors` and `Movies` are endogenous.

would be to refine these notions in order to adapt them to specific purposes.

3.2 Diagnosing Network Failures

With the basic intuition in mind, we will illustrate now a quite different application of causality. Consider a computer network consisting of a number of servers, with physical links between the servers stored in a table `Link(srv1,srv2)`. At any given time, a server may be active or down, and `Active(Srv)` is the table of servers that are assumed to be active. A network administrator, call her Alice, is concerned in maintaining the network's connectivity, which can be expressed by the following Datalog program:

```
Connected(x,y) :- Link(x,y),Active(x),Active(y)
Connected(x,y) :- Connected(x,z),Link(z,y),Active(y)
Connected(x,y) :- Connected(y,x)
```

The last rule expresses the fact that the graph is undirected. If a server is down and Alice knows that, she will take immediate corrective action, by rebooting the server or replacing the physical machine. Thus, under normal operation, all servers are up and the network is connected.

At any given time, `Active` contains all servers that are believed by Alice to be active. Suppose that, at some point, Alice observes that a server `A` is no longer connected to a server `B`. She can learn this from failed processes, or by examining logs, or by receiving a system alert. Some servers in the network are down, and this caused `A` and `B` to become disconnected. Alice's challenge is to identify which servers are down. Thus, the content of `Active` is no longer up to date: if we run the query `Connected`, it still returns the answer `A,B`, but this clearly disagrees with the actual observation because `Active` is not up to date. Here we use causality and responsibility to help diagnose the real system, instead of the database. Suppose that we declare all tuples in `Link` to be exogenous; only tuples in `Active` are endogenous. In other words, we assume that links do not fail, only servers. Then the lineage of the answer `Connected(A,B)` consists of all active servers on all simple paths connecting `A` to `B`, and each server `C` on a simple path from `A` to `B` is an actual cause. Ranking these servers by responsibility will help Alice diagnose the network. Indeed, a minimal contingency for a server `C` to be a cause is a set of servers Γ such that $\Gamma \cup \{\texttt{C}\}$ is a minimal cut disconnecting `A` from `B`. Thus, servers with a high responsibility are a more likely explanation for the observed disconnection in the network, while servers with low responsibility are less likely explanations. By ranking servers in decreasing order of their responsibility the system assists the network administrator with valuable information, allowing her to prioritize her diagnosis work.

Extensions. In a well-studied problem called *network reliability* [Col87], we are given a network where each edge (or node) has a given probability of failing, and are asked to compute the probability that the network remains connected. The network diagnosis example is clearly related to network reliability, and this connection suggests the following extension of causality and responsibility. Suppose that for each tuple in the database we are given a score (or weight): then it makes sense to define the responsibility of a tuple t as a function of the mini-

mum weight of a contingency Γ for t. Thus, we no longer consider only the cardinality of Γ, but take into account the weights of the tuples in Γ. In the case of network reliability, the weight of a node s can be the probability that s fails, that is, s is active with probability $1-p(s)$, and the probability of a contingency Γ is $\prod_{s\in\Gamma} p(s) \prod_{s\notin\Gamma}(1-p(s))$. Define the responsibility of a tuple t to be the maximum probability of all contingencies.

Halpern [Hal08] has recently discussed a related refinement of causality, by combining it with default reasoning. In that setting, the weight of a world represents how normal that world is. $\mathcal{K}(D_1) \leq \mathcal{K}(D_2)$ means that D_1 is more "normal" than D_2, where $\mathcal{K}$, is a given ranking function; equivalently, D_2 is more "exceptional" than D_1). The definition of causality is then modified by requiring the contingency Γ to lead to a more "normal" world, that is, $\mathcal{K}(D-\Gamma) \leq \mathcal{K}(D)$. Notice that this is different from changing the definition of responsibility, because it restricts the set of allowable contingencies rather than modifying their numerical value. The two extensions are motivated by different applications. In the case of network reliability, *every* contingency containing at least one failure is more exceptional than the normal state (with no failures), so restricting contingencies does not make a difference here.

3.3 Handling Aggregate Queries

Causality poses additional challenges when the query contains aggregates. Consider a simple aggregate query: `select sum(A) from R`. Here every tuple in `R` that has `R.A`$\neq$`0` is counterfactual, because its addition/removal changes the sum, but obviously not all tuples in `R` are equal contributors to the sum. This highlights the limitations of our simple definition of causality. It would be interesting to extend the definition of causality and responsibility to quantify the contribution of each input tuple to the aggregate, which may be challenging when the query combines aggregates with joins.

However, even our simple definition of causality captures some interesting aspect of aggregate queries. Consider the question "why is the sum greater than 500?". This can be phrased as a causality question for the boolean query Q: `select 'true' from R having sum(A) > 500`, to which our notion of causality and responsibility applies. Assume the following instance of $R.A$: $\{450, 150, 75, 25\}$. In this case, 450 is a counterfactual tuple for Q (removing it makes Q false); 150 is an actual cause with contingency $\{75\}$, while 25 is not an actual cause at all. The responsibilities are $\rho_{450} = 1$, $\rho_{150} = \rho_{75} = 1/2$, and $\rho_{25} = 0$ (for details see [MGMS09]).

Typical monitoring queries contain several joins and aggregates, and generate alerts based on whether an aggregate reached a specified value, that is, `having count(*) > N`. Causality aims to answer the question "why did the number of events (or errors, or failures) exceed N?". We note here that, while causality and responsibility are captured by our definition, the computational challenges may be much higher in the case of queries with aggregates, because the lineage may have a size that is exponential in N, and therefore it is not possible to compute explicitly. New algorithms are likely to be needed in order to efficiently compute or approximate the responsibility of tuples in queries with aggregates.

3.4 View-Conditioned Causality

In all previous examples we have assumed that there is a single query q, and a single answer r to that query, and have defined causality for a fixed q and r. In practice one often has multiple queries and/or multiple answers. Fix a query q and an answer r, and consider k additional queries, called views, $v_1, \ldots, v_k$, with answers $b_1, \ldots, b_k$. The answers to the views are normal and expected, but the answer to the query is unexpected. We want to explain the cause for the answer r to q, conditioned on the answers to the views: we call this *View-Conditioned causality*.

We define view-conditioned causality by making the following small change to Definition 1: we require a contingency Γ to be such that $D - \Gamma - \{t\} \models v_i(b_i)$, for all $i = 1, k$. This is related to the view side-effect problem studied by Buneman et al. [BKT02], which asks for changes to the database that have minimal side-effects on some given views. We illustrate the usefulness of this notion on the examples considered so far:

- Suppose that all answers in Fig. 1 are normal except Music and Musical. With the revised definition, we consider as contingencies only those sets of tuples whose removal does not affect the other query answers.
- Consider a monitoring query that is run periodically on a data stream: for example the query may count the number of dropped IP packets per minute in a computer network. The query returns a stream of answers, $a_1, a_2, \ldots, a_{t-1}, a_t$. Suppose that the last answer a_t is abnormal: there is a spike in the number of dropped packets. The natural question to ask is: what is the cause for the spike a_t given that the previous answers $a_1, a_2, \ldots, a_{t-1}$ were normal? This is captured by view-conditioned causality, where $a_1, \ldots, a_{t-1}$ are the views and a_t is the query.
- Consider a distributed query execution environment, such as Pig over Hadoop [GNC$^+$09, ORS$^+$08]. Complex Pig (or Map-reduce) jobs take hours to run, and their performance depends on a large number of parameters, such as the number of servers, the size of the data, number of reduce tasks, size of the data chunks, etc. Suppose that the same job is run repeatedly, with slightly different parameter settings, and results in running times $a_1, a_2, \ldots, a_n$. Suddenly, the last running time a_n is abnormally high. The interesting question is: what caused a_n to be abnormally high, given that $a_1, a_2, \ldots, a_{n-1}$ were normal?
- Assume the network administrator (Alice) has more information about the actual state of the network: she knows that `C` and `D` are still connected, and so are `E` and `F`, but that `A,B` are disconnected and so is the pair `M,N`. View-conditioned causality allows her to ask the question: why are certain pairs of servers disconnected while others remain connected?

Discussion. Conditioning on views seems to be a key ingredient to causality in databases. We can think of exogenous tuples as those we want to take as given; view-conditioning allows us to to extend this intuition by treating tuples in the view as given.

3.5 Preemption

A major effort in the original HP definition of causality consisted in capturing correctly the notion of preemption, which had been discussed in philosophy and turned out to be difficult to capture in a mathematical formalism. Preemption occurs when one possible cause is known to rule out another possible cause. The standard example of preemption given in [HP05] and elsewhere is when Alice and Bob both throw a rock at a bottle: both aim accurately, but Alice throws first, and therefore her throw hits the bottle and is the cause the bottle breaking, not Bob's. Here preemption is due to temporal ordering, where events that happened earlier preempt events that happened later. For another example, suppose one wants to trace the source of an infection with a computer virus: servers that were infected earlier preempt (as potential causes) servers that were infected later. Non-temporal criteria may also be reasons for preemption. For example, suppose that a burst in stock purchases caused a certain stock to jump 5 points in a few minutes: when searching for the causes of the sudden increase, transactions with high volume should preempt transactions of low volume, as their contribution to the change is higher.

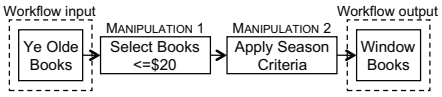
The HP definition approach deals with preemption by choosing the appropriate structural equations. If both Alice and Bob throw a rock at a bottle and we take the equation for bottle breaks (BB) to be $BB = A \vee B$ (where A is "Alice throws" and B is "Bob throws"), then both A and B are actual causes, because the set $\{B\}$ is a contingency for A and vice versa. This equation treats A and B symmetrically; it does not capture the fact that Alice's throw hit the bottle before Bob's did. It is beyond the scope of this paper to review Halpern and Pearl's full definition of causality and responsibility and explain how it addresses preemption. The HP treatment of preemption seems to require the more complicated full HP definition. A major challenge is that of finding a definition more in the spirit of the one used in this paper which handles preemption well.

3.6 Why-not Causality for non-answers

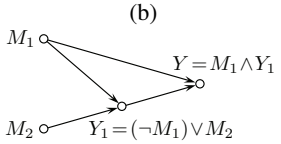
Explanation of query answers naturally extends to the absence of expected results. Recent work has focused on the *why-not* question, that is, *why a certain tuple is not in the result set*. The problem has been addressed from

Author	Title	Price	Publisher
	Epic of Gilgamesh	$150	Hesperus
Euripides	Medea	$16	Free Press
Homer	Iliad	$18	Penguin
Homer	Odyssey	$49	Vintage
Hrotsvit	Basilius	$20	Harper
Longfellow	Wreck of the Hesperus	$89	Penguin
Shakespeare	Coriolanus	$70	Penguin
Sophocles	Antigone	$48	Free Press
Virgil	Aeneid	$92	Vintage

(a)



(b)



(c)

Figure 3: Example book store database from [CJ09] (a), representation of the workflow (b), and the corresponding causal network (c).

both the perspectives of data [HCDN08], which is our focus, and query revisions [CJ09, TC10]. Causality unifies explanations for answers and non-answers under a common framework. We demonstrate how causality, with the notion of preemption discussed in Sec. 3.5, can model an example presented by Chapman and Jagadish in [CJ09].

Example 1 (Book Shopper (Ex. 1) [CJ09]): A shopper knows that all "window display books" at Ye Olde Booke Shoppe are around $20, and wishes to make a cheap purchase. She issues the query: Show me all window display books. Suppose the result from this query is (Euripides, Medea). Why is (Hrotsvit, Basilius) not in the result set? Is it not a book in the store? Does it cost more than $20? Is there a bug in the query-database interface such that the query was not correctly translated?

Chapman and Jagadish consider a discrete component of a workflow, called *manipulation*, as an explanation of a why-not query. The workflow describing the query of the example is shown in Fig. 3b. Roughly, a manipulation is considered *picky* for a non-result if it prunes the tuple. For example, manipulation 1 of Fig. 3b is picky for "Odyssey", as it costs more than $20. Equivalently, a manipulation is *frontier picky* for a set of non-results if it is the last in the workflow to reject tuples from the set. In this framework, the cause of a non-answer will be a frontier picky manipulation.

In Example 1, tuple $t =$(Hrotsvit, Basilius) passes the price test, but is rejected by manipulation 2 as it does not satisfy the seasonal criteria. The causal network representing this example is presented in Fig. 3c. Input nodes model the events and describe the pickiness of manipulations: M_1 holds if manipulation 1 is *not* potentially picky with respect to t, and M_2 holds if manipulation 2 is *not* potentially picky with respect to t. At the end, the tuple appears only if neither manipulation is picky: $M_1 \wedge M_2$. Node Y_1 encodes the precedence of the manipulations in the workflow. A tuple will be stopped at point Y_1 of the workflow if M_2 is picky but M_1 was not, that is, if $M_1 \wedge \neg M_2$ holds. Thus, it will pass this point if $Y_1 = \neg(M_1 \wedge \neg M_2) = \neg M_1 \vee M_2$ holds. A tuple is reported if $Y = M_1 \wedge Y_1$ holds.

Applying the HP framework in the context of the example where M_2 is picky but M_1 is not ($M_1 \wedge \neg M_2$), correctly yields that M_2 is the only cause (counterfactual): $\Gamma = \emptyset$. If both manipulations were potentially picky ($\neg M_1 \wedge \neg M_2$), the HP definition again correctly picks M_1 as the only cause with contingency $\Gamma = \{M_2\}$. Hence, even though M_2 is potentially picky, it is preempted by M_1. This is in agreement with the why-not framework that selects as explanation the last manipulation in the workflow that finally rejected the tuple.

This example demonstrates not only that causality can model why-not questions, but also that it can handle as causes any element that may be considered contributory to an event (in this case workflow manipulations). However, when focusing on tuples, we can again use a simple definition along the lines of the one presented in Sec. 2, this time tweaked to specify causes and contingencies for missing tuples. This requires some minor changes to Definition 1. Now the real database consists entirely of exogenous tuples D^{x}. In addition, we are given a set of potentially missing tuples, whose absence from the database caused r to be a non-answer: these form the endogenous tuples D^{n}, and we denote $D = D^{\mathrm{x}} \cup D^{\mathrm{n}}$. We do not discuss here how to compute the set

D^{n}; we assume that D^{n} is given (one possibility is to compute it using techniques from [HCDN08]). Definition 1 is then modified as follows:

Definition 3 (Why-not Causality): Let $t \in D^{\mathrm{n}}$ be an endogenous tuple, and r a missing answer for q.

- t is called a *counterfactual cause* for the non-answer r in D^{x} if $D^{\mathrm{x}} \not\models q(r)$ and $D^{\mathrm{x}} \cup \{t\} \models q(r)$
- $t \in D$ is called an *actual cause* for the non-answer r if there exists a set $\Gamma \subseteq D^{\mathrm{n}}$ such that t is a counterfactual cause for the non-answer of r in $D^{\mathrm{x}} \cup \Gamma$.

4 Conclusions and Future Work

This paper initiates a discussion of causality in databases. It gives a simple definition of causality in database queries, and illustrates it with several applications, including explanations, validation, data correction, and system debugging. Currently, such tasks can be performed by manually exploring provenance information. *Causality* allows to refine the analysis of provenance by including user and application specific constraints, and provides meaningful ranking metrics. Future work needs to specialize this definition to more concrete applications, and to study the computational challenges associated with computing cause and responsibility.

References

[BBDC+09] Ilan Beer, Shoham Ben-David, Hana Chockler, Avigail Orni, and Richard J. Trefler. Explaining counterexamples using causality. In *21st Annual Conference on Computer Aided Verification (CAV '09)*, pages 94–108, 2009.

[BKT01] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where: A characterization of data provenance. In *ICDT*, pages 316–330, 2001.

[BKT02] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. On propagation of deletions and annotations through views. In *PODS*, pages 150–158, 2002.

[CGY08] Hana Chockler, Orna Grumberg, and Avi Yadgar. Efficient automatic STE refinement using responsibility. In *Proc. 14th Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 233–248, 2008.

[CH04] Hana Chockler and Joseph Y. Halpern. Responsibility and blame: A structural-model approach. *J. Artif. Intell. Res. (JAIR)*, 22:93–115, 2004.

[CHK08] Hana Chockler, Joseph Y. Halpern, and Orna Kupferman. What causes a system to satisfy a specification? *ACM Trans. Comput. Log.*, 9(3), 2008.

[CJ09] Adriane Chapman and H. V. Jagadish. Why not? In *SIGMOD*, pages 523–534, 2009.

[Col87] Charles J. Colbourn. *The combinatorics of network reliability*. Oxford University Press, New York, 1987.

[GKT07] Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, pages 271–280, 2007.

[GNC+09] Alan Gates, Olga Natkovich, Shubham Chopra, Pradeep Kamath, Shravan Narayanam, Christopher Olston, Benjamin Reed, Santhosh Srinivasan, and Utkarsh Srivastava. Building a highlevel dataflow system on top of MapReduce: The pig experience. *PVLDB*, 2(2):1414–1425, 2009.

[GT06] Todd J. Green and Val Tannen. Models for incomplete and probabilistic information. *IEEE Data Eng. Bull.*, 29(1):17–24, 2006.

[Hal08] Joseph Y. Halpern. Defaults and normality in causal structures. In *KR*, pages 198–208, 2008.

[HCDN08] Jiansheng Huang, Ting Chen, AnHai Doan, and Jeffrey F. Naughton. On the provenance of non-answers to queries over extracted data. *PVLDB*, 1(1):736–747, 2008.

[HP05] Joseph Y. Halpern and Judea Pearl. Causes and explanations: A structural-model approach. Part I: Causes. *Brit. J. Phil. Sci.*, 56:843–887, 2005. (Conference version in *UAI*, pages 194-202, 2001).

[MGMS09] Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. Why so? or Why no? Functional causality for explaining query answers. *CoRR*, abs/0912.5340, 2009.

[MGMS11] Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. The causality and responsibility of query answers and non-answers. In *PVLDB*, 2011. (to appear, see `http://db.cs.washington.edu/causality/`).

[ORS+08] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In *SIGMOD*, pages 1099–1110, 2008.

[Pea00] Judea Pearl. *Causality: models, reasoning, and inference*. Cambridge University Press, Cambridge, U.K., 2000.

[TC10] Quoc Trung Tran and Chee-Yong Chan. How to conquer why-not questions. In *SIGMOD*, pages 15–26, 2010.