Lenses: An On-Demand Approach to ETL

Ying Yang†, Niccolò Meneghetti†, Ronny Fehling◦,
Zhen Hua Liu◦, Oliver Kennedy†
† SUNY Buffalo, ◦ Oracle
{yyang25, niccolom, okennedy}@buffalo.edu
{ronny.fehling, zhen.liu}@oracle.com

ABSTRACT

Three mentalities have emerged in analytics. One view holds that reliable analytics is impossible without high-quality data, and relies on heavy-duty ETL processes and upfront data curation to provide it. The second view takes a more ad-hoc approach, collecting data into a data lake, and placing responsibility for data quality on the analyst querying it. A third, on-demand approach has emerged over the past decade in the form of numerous systems like Paygo or HLog, which allow for incremental curation of the data and help analysts to make principled trade-offs between data quality and effort. Though quite useful in isolation, these systems target only specific quality problems (e.g., Paygo targets only schema matching and entity resolution). In this paper, we explore the design of a general, extensible infrastructure for on-demand curation that is based on probabilistic query processing. We illustrate its generality through examples and show how such an infrastructure can be used to gracefully make existing ETL workflows "on-demand". Finally, we present a user interface for On-Demand ETL and address ensuing challenges, including that of efficiently ranking potential data curation tasks. Our experimental results show that On-Demand ETL is feasible and that our greedy ranking strategy for curation tasks, called CPI, is effective.

1. INTRODUCTION

Effective analytics depends on analysts having access to accurate, reliable, high-quality information. One school of thought on data quality manifests as Extract-Transform-Load (ETL) processes that attempt to shield analysts from any uncertainty, by cleaning all data thoroughly up-front. The cleansed data is usually represented in a new or transformed way as tables in a data warehouse. Those tables, typically in form of star schemas, as well as the transformation and parsing logic, all have to be designed up front, an arduous and time-consuming task. Only after loading the parsed and transformed data into the data warehouse can the analyst query the data to do any actual analysis. We collectively refer to this selective parsing, transformation and loading into a new structure as data curation.

Proceedings of the VLDB Endowment, Vol. 8, No. 12

EXAMPLE 1. *Alice is an analyst at the HappyBuy retail store, and is developing a promotional strategy based on public opinion ratings for its products gathered by two data collection companies. A thorough analysis of the data requires substantial data-cleaning effort from Alice: As shown in Figure 1, the rating companies schemas are incompatible, and HappyBuy's own product data is incomplete. However, Alice's preliminary analysis is purely exploratory, and she is hesitant to invest the full effort required to curate this data.*

The upfront costs of curation have lead many to instead inline curation tasks into the analytical process, so that only immediately relevant curation tasks are performed.

EXAMPLE 2. *Alice realizes that she only needs two specific attributes for her analysis:* `category` *and* `rating`*. She considers manually constructing a task-specific data set containing a sanitized version of only these two columns.*

This deferred approach is more lightweight, but encourages analysts to develop brittle one-off data cleansing solutions, incurring significant duplication of effort or organizational overheads. A third approach, initially explored as part of Paygo [25], instead curates data incrementally in response to specific query requirements. This form of *on-demand curation* results in a sanitized data set that is based on a principled trade-off between the quality desired from the data set and the human effort invested in curating it. Paygo specifically targets two curation tasks: schema matching and entity resolution, and other systems have since appeared for schema matching [2], as well as other tasks like information extraction [10], and inference [41, 42].

A typical ETL pipeline often involves many distinct curation tasks, requiring that multiple on-demand data curation systems be used in tandem. However, the data representations and quality metrics used by these systems are optimized for very specific use-cases, making composition difficult. In this paper, we explore and address the challenges of composing specialized on-demand curation techniques into a general-purpose workflow. The result is **a unified model for on-demand curation called On-Demand ETL** that bridges the gap between these systems and allows them to be gracefully incorporated into existing ETL and analytics workflows. This unified model builds around ordinary SQL, retaining compatibility with existing standards for ETL design, data analysis, and database management.

Product

id	name	brand	category	ROWID
P123	Apple 6s, White	?	phone	R1
P124	Apple 5s, Black	?	phone	R2
P125	Samsung Note2	Samsung	phone	R3
P2345	Sony to inches	?	?	R4
P34234	Dell, Intel 4 core	Dell	laptop	R5
P34235	HP, AMD 2 core	HP	laptop	R6

Ratings1

pid	...	rating	review_ct	ROWID
P123	...	4.5	50	R7
P2345	...	?	245	R8
P124	...	4	100	R9

Ratings2

pid	...	evaluation	num_ratings	ROWID
P125	...	3	121	R10
P34234	...	5	5	R11
P34235	...	4.5	4	R12

Figure 1: Incomplete example relations, annotated with implicit per-row lineage markers (ROWID).

Representing Incomplete Data. On-demand curation permits trade-offs between data quality, and the effort needed to obtain high-quality data. This requires a representation for the quality loss incurred by only partially curating data. Existing on-demand curation systems use specialized, task-specific representations. In Section 2 we describe an existing representation for incomplete information called PC-Tables [17, 18, 23], and show how it can be leveraged by On-Demand ETL.

Expressing Composition. If the output of a curation technique is non-deterministic, then for closure, it must accept non-deterministic input as well. In Section 3, we define a model for non-deterministic operators called lenses that capture the semantics of on-demand data curation processes. We illustrate the generality of this model through examples, and show that it is closed over PC-Tables.

Backwards Compatibility. For On-Demand ETL to be practical, it must be compatible with traditional data management systems and ETL pipelines. In Section 4, we develop a practical implementation of PC-Tables [23] called Virtual C-Tables that can be safely embedded into a classical, deterministic database system or ETL workflow.

Presenting Data Quality. In Section 5, we discuss how to present the quality loss incurred by incomplete curation to end-users. We show how lightweight summaries can be used to alert an analyst to specific problems that affect their analysis, and how On-Demand ETL computes a variety of quality measures for query results.

Feedback. Section 6 highlights how lenses act as a form of provenance, linking uncertainty in query outputs to the lenses that created them. These links allow for lens-defined curation tasks that improve the quality of query results. We introduce a concept called *the cost of perfect information* (CPI) that relates the value of a curation task that improves a result's quality, to the cost of performing the task, allowing curation tasks to be ranked according to their net value to the analyst.

Experimental Results. Finally, in Section 7, we present experimental results that demonstrate the feasibility of On-Demand ETL and provide several insights about its use.

Concretely, this paper's contributions include: (1) A *composable* model for expressing data curation tasks based on probabilistic components called lenses, (2) A practical implementation of PC-Tables called Virtual C-Tables that can be deployed into classical databases without needing support for labeled nulls, (3) A family of heuristics called CPI used to prioritize curation tasks that improve result quality at a cost, and (4) Experimental results that illustrate the feasibility of On-Demand ETL and the effectiveness of CPI.

$$
\begin{aligned}
e &:= \mathbb{R} \mid Column \mid \texttt{if } \phi \texttt{ then } e \texttt{ else } e \\
&\quad \mid e \ \{+,-,\times,\div\} \ e \mid Var(id[,e[,e[,\ldots]]]) \\
\phi &:= e \ \{=,\neq,<,\leq,>,\geq\} \ e \mid \phi \ \{\wedge,\vee\} \ \phi \mid \top \mid \bot \\
&\quad \mid e \texttt{ is null} \mid \neg\phi
\end{aligned}
$$

Figure 2: Grammars for boolean expressions ϕ and numerical expressions e including support for VG-Functions $Var(\ldots)$.

2. BACKGROUND AND RELATED WORK

A deterministic database is a finite collection of relation instances $\{R_1, \ldots, R_k\}$ over a schema $\mathcal{S} = \{\mathcal{S}_1, \ldots, \mathcal{S}_k\}$. According to the "possible worlds" semantics [37] a probabilistic database $\mathcal{D}$ consists of a pair $(\mathbf{W}, P)$, where $\mathbf{W}$ is a large collection of deterministic databases, the so called possible worlds, all sharing the same schema $\mathcal{S}$, and P is a probability measure over $\mathbf{W}$. Roughly speaking, $\mathcal{D}$ is a database whose schema is known but whose internal state is uncertain, and $\mathbf{W}$ simply enumerates all its plausible states. We denote by R the set of all tuples that appear in some possible world (often called possible tuples). Each element of R is an *outcome* for the probability space $(\mathbf{W}, P)$. The *confidence* of a possible tuple t is simply the probability that it will appear in the database $\mathcal{D}$, i.e. its marginal probability

$$P(t \in \mathcal{D}) = \sum_{W_i \in \mathbf{W} | t \in W_i} P(W_i)$$

The goal of probabilistic databases [1,8,16,22,24,27,34,36] is to support the execution of deterministic queries like regular, deterministic databases do. Let's denote by Q an arbitrary deterministic query (i.e., a query expressible in classical bag-relational algebra) and by $sch(Q)$ the schema defined by it, which consists of a single relation. The application of Q to $\mathcal{D}$, denoted by $Q(\mathcal{D})$, generates a new probability space $(\mathbf{W}', P')$ where $\mathbf{W}' = \{Q(W_i) \mid W_i \in \mathbf{W}\}$ and

$$P'(t \in Q(\mathcal{D})) = \sum_{W_i \in \mathbf{W} | t \in Q(W_i)} P(W_i)$$

A probabilistic query processing (PQP) system is supposed to answer a deterministic query Q by listing all its possible answers and annotating each tuple with its marginal probability, or by computing expectations for aggregate values. These tasks are difficult in practice, mainly for two reasons: (i) $\mathbf{W}$ is usually too large to be enumerated explicitly, and (ii) computing marginals is provably #P-hard in the general case. For example, if our schema contains a single relation and our set of possible worlds contains all subsets of a given set of 100 tuples, then we have 2^{100} distinct possible worlds where each possible tuple appears in half.

One way to make probabilistic query processing efficient is to encode $\mathbf{W}$ and P with a compact, factorized representation. In this paper we adopt a generalized form of C-Tables [23, 27] to represent $\mathbf{W}$, and PC-Tables [17, 18] to represent the pair $(\mathbf{W}, P)$. A C-Table [23] is a relation instance where each tuple is annotated with a lineage formula ϕ, a propositional formula over an alphabet of variable symbols Σ. The formula ϕ is often called a *local condition* and

the symbols in Σ are referred to as *labeled nulls*, or just variables. Intuitively, for each assignment to the variables in Σ we obtain a possible relation containing all the tuples whose formula ϕ is satisfied. For example:

Product					
	pid	**name**	**brand**	**category**	ϕ
t_1	P123	Apple 6s, White	Apple	phone	$x_1 = 1$
t_2	P123	Apple 6s, White	Cupertino	phone	$x_1 = 2$
t_3	P125	Samsung Note2	Samsung	phone	$\top$

The above C-Table defines a set of three possible worlds, $\{t_1, t_3\}$, $\{t_2, t_3\}$, and $\{t_3\}$, i.e. one world for each possible assignment to the variables in the one-symbol alphabet $\Sigma = \{x_1\}$. Notice that no possible world can have both t_1 and t_2 at the same time. C-Tables are closed w.r.t. positive relational algebra [23] : if $\mathbf{W}$ is representable by a C-Table and Q is a positive query then $\mathbf{W}' = \{Q(W_i) \mid W_i \in \mathbf{W}\}$ is representable by another C-Table.

Following the approach of PIP [27], in this paper we adopt VG-RA (variable-generating relational algebra), a generalization of positive bag-relation algebra with extended projection, that uses a simplified form of VG-functions [24]. In VG-RA, VG-functions (i) dynamically introduce new Skolem symbols in Σ, that are guaranteed to be unique and deterministically derived by the function's parameters, and (ii) associate the new symbols with probability distributions. Hence, VG-RA can be used to define new C-Tables. Primitive-valued expressions in VG-RA (i.e., projection expressions and selection predicates) use the grammar summarized in Figure 2. The primary addition of this grammar is the VG-Function term: $Var(\ldots)$.

From PIP, we also inherit a slightly generalized form of C-Tables. Our C-Tables differ from the canonical ones in the following: (i) Variables in Σ are allowed to range over continuous domains, (ii) attribute-level uncertainty is encoded by replacing missing values with VG-RA *expressions* (not just functions) that act as Skolem terms and (iii) these VG-RA expressions allow basic arithmetic operations. The previous example is equivalent to the PIP-style C-Table:

Product			
pid	**name**	**brand**	**category**
P123	Apple 6s, White	$Var('X', \texttt{R1})$	phone
P125	Samsung Note2	Samsung	phone

From now on, without explicitly mentioning PIP, we will assume all C-Tables support the generalizations discussed above. It has been shown that C-Tables are closed w.r.t VG-RA [23, 27]. The semantics for VG-RA query evaluation $[[\cdot]]_{CT}$ over C-Tables [22, 23, 27] are summarized in Figure 3. These semantic rules make extensive use of the *lazy evaluation* operator $[[\cdot]]_{lazy}$, which uses a *partial* binding of $Column$ or $Var(\ldots)$ atoms to corresponding expressions. Lazy evaluation applies the partial binding and then reduces every sub-tree in the expression that can be deterministically evaluated. Non deterministic sub-trees are left intact. Any tuple attribute appearing in a C-Table can be encoded as an abstract syntax tree for a partially evaluated expression that assigns it a value. This is the basis for evaluating projection operators, where every expression e_i in the projection's target list is lazily evaluated. Column bindings are given by each tuple in the source relation. The local condition ϕ is preserved intact through the projection. Selection is evaluated by combining the selection predicate ϕ with each tuple's existing local condition. As an optimization, tuples for which ϕ deterministically evaluates to false ($\bot$) are preemptively discarded.

A PC-Table [17, 18] is a C-Table augmented with a probability measure P over the possible assignments to the variables in Σ. Since each assignment to the variables in Σ generates a possible world, a PC-Table induces a probability measure over $\mathbf{W}$. Hence, it can be used to encode a probabilistic database $(\mathbf{W}, P)$. PC-Tables are the foundation for several PQP systems, including MayBMS [22], Orchestra [16] and PIP [27]. Green et al. [17] observed that PC-Tables generalize other models like Trio [1]. The relationship between PC-Tables and VG-RA is discussed in further detail in Section 3.

2.1 Other Related Work

There are numerous tools for on-demand data curation, each targeting *specific* challenges like schema matching [2, 25], de-duplication [25], information extraction [9, 10], information integration [5], or ontology construction [4, 19]. On-Demand ETL generalizes these, providing a tool for on-demand data curation that these solutions can be plugged into. On-demand curation can also be thought of as a highly-targeted form of crowd-sourced databases [32], which leverage the power of humans to perform complex tasks.

The problem of incomplete data arises frequently in distributed systems, where node failures are common. Existing solutions based on uncertainty [7, 15, 30, 40] can similarly be expressed in On-Demand ETL to provide more fine-grained analyses of result quality over partial data than these approaches provide natively.

Prioritizing curation tasks, as addressed in Section 6, is quite closely related to Stochastic Boolean Function Evaluation, or SBFE, where the goal is to determine the value of a given Boolean formula by paying a price to discover the exact value of uncertain boolean variables. The problem is hard in its general form; exact solutions and heuristics have been proposed for several classes of functions [26, 39]. More recently Deshpande et al. [11] designed a 3-approximation algorithm for linear threshold formulas, while Allen et al. [3] developed exact and approximate solutions for monotone k-term DNF formulas.

3. LENSES

A lens is a data processing component that is evaluated as part of a normal ETL pipeline. Unlike a typical ETL processing stage that produces a single, deterministic output, a lens instead produces a PC-Table $(\mathbf{W}, P)$, which defines the set of possible outputs, and a probability measure that approximates the likelihood that any given possible output accurately models the real world. In effect, a lens gives structure to uncertainty about how an ETL process should interpret its input data.

Asking ETL designers to specify this structure manually for the entire ETL process is impractical. Lenses allow this structure to be specified as a composition of individual *simple* transformations, constraints, or target properties that take the place of normal operators in the ETL pipeline. However, composition requires closure. In this section, we define a closed framework for lens specification, and illustrate its generality through three example lenses.

3.1 The Lens Framework

A lens instance is defined over a query $Q(D)$, and is in turn responsible for constructing a PC-Table $(\mathbf{W}, P)$. A lens defines $\mathbf{W}$ as a C-Table through a VG-RA expression

Expression	Evaluates To
$[[\pi_{a_i \leftarrow e_i}(R)]]_{CT}$	$\{ \langle a_i : [[e_i(t)]]_{lazy}, \phi : t.\phi \rangle \mid t \in [[R]]_{CT} \}$
$[[\sigma_{\psi}(R)]]_{CT}$	$\{ \langle a_i : t.a_i, \phi : [[t.\phi \wedge \psi(t)]]_{lazy} \rangle \mid t \in [[R]]_{CT} \wedge ([[t.\phi \wedge \psi(t)]]_{lazy} \not\equiv \bot) \}$
$[[R \times S]]_{CT}$	$\{ \langle a_i : t_1.a_i, a_j : t_2.a_j, \phi : t_1.\phi \wedge t_2.\phi \rangle \mid t_1 \in [[R]]_{CT} \wedge t_2 \in [[S]]_{CT} \}$
$[[R \uplus S]]_{CT}$	$\{ \langle a_i : t.a_i, \phi : t.\phi \rangle \mid t \in ([[R]]_{CT} \uplus [[S]]_{CT}) \}$

Figure 3: Evaluation semantics for positive bag-relational algebra over C-Tables

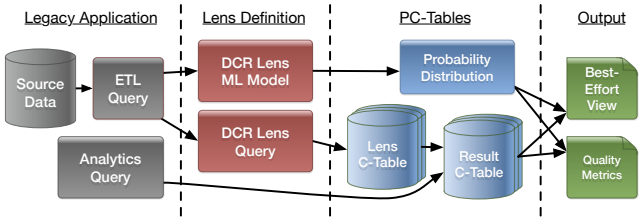

Figure 4: Example of the domain constraint repair lens applied in a legacy application. The output layer is discussed in Section 5.

$\mathcal{F}_{lens}(Q(D))$. Independently, the lens constructs P as a joint probability distribution over every variable introduced by $\mathcal{F}_{lens}$, by defining a sampling process in the style of classical VG-functions [24], or supplementing it with additional metadata to create a PIP-style grey-box [27]. An example of the complete process for the Domain Constraint Lens defined below is illustrated in Figure 4. These semantics are closed over PC-Tables. If $Q(D)$ is non-deterministic — that is, the lens' input is defined by a PC-Table $(Q(D), P_Q)$ — the lens' semantics are virtually unchanged. VG-RA is closed over C-Tables, so $\mathcal{F}_{lens}(Q(D))$ simply defines a new C-Table. Defining P as an extension of P_Q with distributions for all variables newly introduced by $\mathcal{F}_{lens}$ provides closure for the probability measure, a topic we will return to in Section 3.3.

3.2 Lens Examples

We first illustrate the generality of the lens framework through three example lenses: domain constraint repair, schema matching, and archival. To construct a lens over query Q, the user writes:

```
CREATE LENS <lens_name> AS Q
  USING <lens_type>(<lens_arguments>);
```

Domain Constraint Repair. A domain constraint repair lens enforces attribute-level constraints such as `NOT NULL`. Under the assumption that constraint violations are a consequence of data-entry errors or missing values, domain constraint violations can be repaired by finding a legitimate replacement for each invalid value. Obtaining reliable replacement values typically requires detailed domain knowledge. However, in an on-demand setting, approximations are sufficient. The domain constraint repair lens uses educated guesses about data domains (e.g., uniform distributions over allowable values) and machine learning models (e.g., a naive Bayes classifier trained over $Q(D)$) to approximate domain knowledge. With $sch(Q) = \{\langle a_1, t_1 \rangle, \ldots, \langle a_n, t_n \rangle\}$ denoting the attributes a_i of $Q(D)$ and their type t_i, a domain constraint repair lens definition has the form:

$$\texttt{... USING DOMAIN\_REPAIR(}a_1\ t_1\texttt{, ..., }a_n\ t_n\texttt{)}$$

The C-Table for the lens' output is constructed by the query $\mathcal{F}_{lens} = \pi_{\{\ldots, a_i \leftarrow V_i, \ldots\}}$, where each V_i is defined as:

$$\texttt{if } t_i \models a_i \texttt{ then } a_i \texttt{ else } Var(Name_i, \texttt{ROWID}))$$

id	name	brand	category
P123	Apple 6s, White	$Var('X', \texttt{R1})$	phone
P124	Apple 5s, Black	$Var('X', \texttt{R2})$	phone
P125	Samsung Note2	Samsung	phone
P2345	Sony 60 inches	$Var('X', \texttt{R4})$	$Var('Y', \texttt{R4})$
P34234	Dell, Intel 4 core	Dell	laptop
P34235	HP, AMD 2 core	HP	laptop

Figure 5: The C-Table for `SaneProduct`

In this expression, $t_i \models a_i$ if a_i satisfies the type constraints of t_i, and each $Name_i$ is a freshly allocated variable name. Independently, P is defined by training a classifier or similar model for each attribute on the output of Q.

Example 3. *Returning to Example 1, Alice creates a lens to handle missing values in the Product table:*

```
CREATE LENS SaneProduct AS SELECT * FROM Product
  USING DOMAIN_REPAIR( category string NOT NULL,
                       brand    string NOT NULL );
```

From Alice's perspective, the lens `SaneProduct` *behaves as a standard database view. However, the content of the lens is guaranteed to satisfy the domain constraints on* `category` *and* `brand`*. `NULL` values in these columns are replaced according to a classifier built over the output of the query over* `Product`*. Figure 5 shows the C-Table for this lens.*

Schema Matching. A schema matching lens creates a mapping from the source data's schema to a user-defined target schema. This is especially important for non-relational data like JSON objects or web tables, which may not have well defined schemas [21, 25]. Given a destination schema $\{\langle b_1, t_1 \rangle, \ldots, \langle b_m, t_m \rangle\}$ and a threshold ω, a schema matching lens definition has the form:

$$\texttt{... USING SCHEMA\_MATCHING(}b_1\ t_1\texttt{, ..., }b_m\ t_m\texttt{, }\omega\texttt{)}$$

The schema matching lens defines a fresh boolean variable $Var(Name_{i,j})$ for every pair a_i, b_j, where $\langle a_i, t_i \rangle \in sch(Q)$. The probability of $Var(Name_{i,j})$ corresponds to the probability of a match between a_i and b_j. $\mathcal{F}_{lens}$ takes the form: $\pi_{\{\ldots, b_j \leftarrow V_j, \ldots\}}$, where V_j enumerates possible matches for b_j:

$$\texttt{if } Var(Name_{1,j}) \texttt{ then } a_1 \texttt{ else}$$
$$\vdots$$
$$\texttt{if } Var(Name_{n,j}) \texttt{ then } a_n \texttt{ else NULL}$$

As an optimization, matches for type-incompatible pairs of attributes are skipped. Additionally, the lens discards matches where the likelihood of a schema-level match falls below a user-defined threshold (ω).

Example 4. *Alice next turns to the ratings data sets, which have incompatible schemas. She creates a lens and a joint view:*

```
CREATE LENS MatchedRatings2 AS SELECT * FROM Ratings2
  USING SCHEMA_MATCHING( pid string, ..., rating float,
                         review_ct float, NO LIMIT );
CREATE VIEW AllRatings AS SELECT * FROM MatchedRatings2
                    UNION SELECT * FROM Ratings1;
```

The resulting C-Table for `MatchedRatings2` *is shown in Figure 6. From Alice's perspective,* `AllRatings` *behaves as a normal view combining* `Ratings1` *and* `Ratings2`*. Behind the scenes, the attributes of* `Ratings2` *are quietly matched against those of* `Ratings1`*. In this example, only* `evaluation` *and* `num_ratings` *are type compatible match candidates, and other match cases are dropped.*

Numerous options are available for constructing P, including domain-based schemes or complex ontology-based matching. However, even a simple matching scheme can be sufficient for On-Demand ETL. We approximate the probability of a match between two attributes by a normalized edit distance between the two attribute names. As we show in Section 7 (Figure 11), even this simple matcher can produce suitable results.

Archival. An archival lens captures the potential for errors arising from OLAP queries being posed over stale data [29], like queries run in between periodic OLTP to OLAP bulk data copies. The lens takes a list of pairs $\langle T, R \rangle$, where R is a reference to a relation in an OLTP database, and T is the period with which R is locally archived.

$$\ldots \texttt{USING ARCHIVAL}(\langle T_1, R_1 \rangle, \ldots, \langle T_m, R_m \rangle)$$

This lens probabilistically discards rows from its output that are no-longer valid according to the lens query $\mathcal{F}_{lens} = \sigma_{Var(Name,\texttt{ROWID})}$, where $Name$ is a freshly allocated identifier. In the background, the lens periodically polls for samples drawn from each R_j to estimate the volatility of each relation referenced by Q. Denote by ν_j the probability of a tuple in R_j being invalidated at some point during the period T_j. P is defined independently for each row as a binomial distribution with probability $\prod_{\{j|R_j \in Q\}} \nu_j$.

3.3 Composing Lenses

EXAMPLE 5. *When Alice examines* `AllRatings`*, she suddenly realizes that the data in* `Ratings1` *is missing* `rating` *information. She creates a domain repair lens:*

```
CREATE LENS SaneRatings AS
SELECT pid, category, rating, review_ct FROM AllRatings
USING DOMAIN_REPAIR(rating DECIMAL NOT NULL)
```

The C-Table for `SaneRatings` *is straightforward to construct, as both lenses involved can be expressed as VG-RA expressions. However, the domain repair lens must still train a model to fill in distributions for missing values. In contrast to Example 3, where the model was trained on deterministic input, here the input is a PC-Table.*

The closure of VG-RA over PC-Tables requires that any non-deterministic query $\mathcal{F}$ be defined alongside a process that extends the input PC-Table's probability measure P_{in} to cover any variables introduced by $\mathcal{F}$. For lenses, there are three possibilities. In the trivial case where $\mathcal{F}$ introduces no new variables, P_{in} remains unmodified. In the second case, variables introduced by $\mathcal{F}$ are independent of P_{in} and a joint distribution is defined trivially as the product of the original and new distributions. If any new variables depend on P_{in}, a grey-box distribution definition [27] can be used to express these dependencies directly.

However, it may not always be possible to explicitly define dependencies, particularly when adapting existing on-demand cleaning solutions. On-Demand ETL provides three separate mechanisms to enable support for lenses that require deterministic inputs: (i) Train the lens on the most-likely output of the source lens (see Section 5), (ii) Train the lens on samples of rows drawn from random instances of the source model, or (iii) Train the lens on the subset of the source data that is fully deterministic (i.e., certain).

EXAMPLE 6. *Alice issues the following query:*

```
SELECT p.pid, p.category, r.rating, r.review_ct
FROM SaneRatings r NATURAL JOIN Product p
WHERE p.category IN ('phone','TV') OR r.rating > 4
```

The resulting C-Table is shown in Figure 7. The first two products are entirely deterministic. P125 is a phone and deterministically satisfies the query, but has attribute-level uncertainty from schema matching (Example 4). P2345 has a missing `category` *(Example 3) and* `rating` *(Example 5), so the row's condition is effectively the entire selection predicate. P34234 and P34235 are laptops and fail the test on* `category`*, so their presence in the result set depends entirely on how* `rating` *is matched (Example 4). Recall that there are three candidates:* `evaluation`*,* `num_ratings`*, or neither. In the last case, domain repair (Example 5) replaces the* `NULL` *with* $Var('Z', R11)$ *and* $Var('Z', R12)$*. P34234 and P34235 have functional* `if` *expressions in their conditions, with the form* (`if` ϕ_1 `then` e_2 `else` e_3) > 4*. These expressions can be simplified by recursively pushing the comparison into the branches:* `if` ϕ_1 `then` $e_2 > 4$ `else` $e_3 > 4$*, in-lining the branches into the condition:* $(\phi_1 \wedge (e_2 > 4)) \vee (\neg\phi_1 \wedge (e_3 > 4))$*, and then further simplifying the resulting boolean expression.*

4. VIRTUAL C-TABLES

We next address the challenge of deploying the PQP techniques necessary to support Lenses into an existing database or ETL pipeline. Our approach, called *Virtual C-Tables* or VC-Tables, works by decomposing VG-RA queries into deterministic and non-deterministic components. Non-deterministic components are percolated out of queries, making it possible for the bulk of the ETL process to remain within a classical deterministic system. A small On-Demand ETL shim layer wraps around the database, and provides a minimally-invasive interface for uncertainty-aware users and applications. This shim layer is also responsible for managing several views, discussed in Section 5, that provide backwards compatibility for legacy applications.

Let $\mathcal{F}(D)$ denote a VG-RA query over a deterministic database D. When combined with a probability measure P, $(\mathcal{F}(D), P)$ defines a PC-Table. Semantics for deterministic queries over PC-Tables are well defined, but rely on support for labeled nulls, a feature not commonly found in popular data management systems. Existing probabilistic query processing systems address this limitation by restricting themselves to special cases like finite-discrete probability distributions over categorical data [13,22], relying on costly user-defined types [24, 27, 28], or by specializing the entire database for uncertain data management [1, 16, 36]. Ultimately, each of these solutions is either too specialized for On-Demand ETL, or too disruptive to be deployed into an existing classical ETL pipeline or databases.

Virtual C-Tables decouple the deterministic components of a query from the non-deterministic components that define a PC-Table. This decomposition is enabled by the observation that once the probability measure P of a PC-Table

pid	...	rating	review_ct
P125	...	if $Var('rat = eval')$ then 3 else if $Var('rat = num\_r')$ then 121 else NULL	if $Var('rev\_ct = eval')$ then 3 else if $Var('rev\_ct = num\_r')$ then 121 else NULL
P34234	...	if $Var('rat = eval')$ then 5 else if $Var('rat = num\_r')$ then 5 else NULL	if $Var('rev\_ct = eval')$ then 5 else if $Var('rev\_ct = num\_r')$ then 5 else NULL
P34235	...	if $Var('rat = eval')$ then 4.5 else if $Var('rat = num\_r')$ then 4 else NULL	if $Var('rev\_ct = eval')$ then 4.5 else if $Var('rev\_ct = num\_r')$ then 4 else NULL

Figure 6: The C-Table for MatchedRatings2

id	category	rating	review_ct	ϕ (condition)
P123	phone	4.5	50	$\top$
P124	phone	4	100	$\top$
P125	phone	if $Var('rat = eval')$ then ... ...else $Var('Z', R10)$	if $Var('rat = eval')$ then ... ...else NULL	$\top$
P2345	$Var('Y', R4)$	$Var('Z', R8)$	245	$(Var('Y', R4) = 'phone') \vee (Var('Y', R4) = 'TV') \vee Var('Z', R8) > 4$
P34234	laptop	if $Var('rat = eval')$ then ... ...else $Var('Z', R11)$	if $Var('rat = eval')$ then ... ...else NULL	$Var('rat = eval') \vee Var('rat = num\_r') \vee (Var('Z', R11) > 4)$
P34235	laptop	if $Var('rat = eval')$ then ... ...else $Var('Z', R12)$	if $Var('rat = eval')$ then ... ...else NULL	$Var('rat = eval') \vee (\neg Var('rat = num\_r') \wedge (Var('Z', R12) > 4))$

Figure 7: C-Table for the query over SaneRatings and SaneProduct

$(\mathcal{F}(D), P)$ is constructed, further *deterministic* queries Q over the PC-Table do not affect P. Consequently, we are free to rewrite the C-Table $Q(\mathcal{F}(D))$ defined by any query over $(\mathcal{F}(D), P)$ into any *equivalent* query $\mathcal{F}'(Q'(D))$, where Q' is deterministic and $\mathcal{F}'$ is non-deterministic. In this new, normalized form, the heavy-weight deterministic inner query Q' can be evaluated by a traditional database, while a much simpler $\mathcal{F}'$ can be evaluated or analyzed by a small shim layer sitting between the database and its users. Further queries $q(\mathcal{F}'(Q'(D)))$ can likewise be rewritten into normal form $\mathcal{F}''(q'(Q'(D)))$, enabling views and SELECT INTO queries, both of which frequently appear in ETL workflows.

4.1 Normal Form VG-RA

Non-determinism arises in VG-RA queries through expressions containing variable terms — that is, only through projection and selection. Correspondingly, we propose a normal form of VG-RA: $\pi_{a_i \leftarrow e_i}(\sigma_\phi(Q(D)))$, where the source query $Q(D)$ is expressible using classical bag-relational algebra. The two outer operators, which we represent jointly as $\mathcal{F}(\langle\, a_i \leftarrow e_i \,\rangle, \phi)$, fully encode the branching possibilities of the C-Table. Figure 8 shows how any query in VG-RA can be rewritten into this form by percolating all expressions with a VG-Term $Var(\ldots)$ up through the relational algebra tree.

Projection and selection operators wrapping around $\mathcal{F}$ may be in-lined into $\mathcal{F}$ according to rewrites 1 and 2. As an optimization, expressions and conditions are simplified through lazy evaluation, and predicates ψ are partitioned into two components: $\psi_{var} \wedge \psi_{det} \equiv [[\psi(a_i \leftarrow e_i)]]_{lazy}$, having and not having variable terms, respectively.

Cross-products of two normalized expressions are composed in the straightforward way by concatenating attribute sets and conjunctively combining local conditions as shown in rewrite 3. We use alpha-renaming to avoid schema conflicts between $Q(D)$ and $Q'(D)$, and without loss of generality, assume that the intersection of a_i and a'_j is empty.

Finally, rewrite 4 shows how bag-unions can be rewritten by injecting a provenance marker into the deterministic queries. A fresh attribute *src* distinguishes rows originating from each of two source queries Q and Q'.

$$\pi_{a'_j \leftarrow e'_j}\left(\mathcal{F}(\langle\, a_i \leftarrow e_i \,\rangle, \phi)(Q(D))\right) \equiv \mathcal{F}(\langle\, a'_j \leftarrow [[e'_j(a_i \leftarrow e_i)]]_{lazy} \,\rangle, \phi)(Q(D)) \quad (1)$$

$$\sigma_\psi\left(\mathcal{F}(\langle\, a_i \leftarrow e_i \,\rangle, \phi)(Q(D))\right) \equiv \mathcal{F}(\langle\, a_i \leftarrow e_i \,\rangle, \phi \wedge \psi_{var})(\sigma_{\psi_{det}}(Q(D))) \quad (2)$$

$$\mathcal{F}(\langle\, a_i \leftarrow e_i \,\rangle, \phi)(Q(D)) \times \mathcal{F}(\langle\, a'_j \leftarrow e'_j \,\rangle, \phi')(Q'(D)) \equiv \mathcal{F}(\langle\, a_i \leftarrow e_i, a'_j \leftarrow e'_j \,\rangle, \phi \wedge \phi')(Q(D) \times Q'(D)) \quad (3)$$

$$\mathcal{F}(\langle\, a_i \leftarrow e_i \,\rangle, \phi)(Q(D)) \uplus \mathcal{F}(\langle\, a_i \leftarrow e'_i \,\rangle, \phi')(Q'(D)) \equiv \mathcal{F}(\langle\, a_i \leftarrow [[\texttt{if } src = 1 \texttt{ then } e_i \texttt{ else } e'_i]]_{lazy} \,\rangle, [[\texttt{if } src = 1 \texttt{ then } \phi \texttt{ else } \phi']]_{lazy})(\pi_{*,src \leftarrow 1}(Q(D)) \uplus \pi_{*,src \leftarrow 2}(Q'(D))) \quad (4)$$

Figure 8: Recursive reduction to Normal Form.

4.2 Virtual Views

Normalization allows lenses to be incorporated into existing ETL pipelines. A lens constructs a probability measure P out of its input $Q(D)$, and a C-Table using $\mathcal{F}_{lens}(Q(D))$. The lens query and any subsequent queries over it are normalized into a normal form query $\mathcal{F}'(Q'(D))$, and the view $Q'(D)$ is constructed and materialized by the traditional database. $\mathcal{F}'$ is stored alongside Q' and defines a *virtual view* for $\mathcal{F}'(Q'(D))$. The shim interface transparently normalizes queries over virtual views $q(\mathcal{F}'(Q'(D)))$ to $\mathcal{F}''(q'(Q'(D)))$, allowing $q'(Q'(D))$ to be evaluated by the traditional database. View definitions and SELECT INTO queries are similarly rewritten, defining new virtual views instead of their normal behavior.

5. ANALYSIS

Using virtual views, queries over lens outputs are rewritten into the normal form $\mathcal{F}(Q(D))$, and $Q(D)$ is evaluated by the database. However, the C-Table construction query $\mathcal{F}$ is of minimal use in its raw form. We next turn to the construction of user-consumable summaries of $\mathcal{F}$.

id	category	rating	review_ct	
P123	phone	4.5	50	
P124	phone	4	100	
P125	phone	2 *	3 *	
P34235	laptop	5 *	4.5 *	*

(Up to 2 results may be missing. *)

Figure 9: The best-guess summary of the C-Table from Figure 7 that Alice actually sees.

5.1 Summarizing the Result Relation

Users consume a Virtual C-Table $\mathcal{F}(\langle\, a_i \leftarrow e_i \,\rangle, \phi)(Q(D))$ through one of two deterministic summary relations: A deterministic relation $\mathcal{R}_{det}$, and a best-guess relation $\mathcal{R}_{guess}$. The deterministic relation $\mathcal{R}_{det}$ represents the certain answers [12] of the virtual C-Table, and is constructed by replacing every variable reference in each e_i and ϕ with `NULL`, and dropping rows where $\phi \neq \top$:

$$\texttt{SELECT } e_i(* \to \texttt{NULL}) \texttt{ AS } a_i \texttt{ FROM } Q(D) \texttt{ WHERE } \phi(* \to \texttt{NULL})$$

The resulting relation contains all of the rows deterministically present in $\mathcal{F}(Q(D))$, with `NULL` taking the place of any non-deterministic values. $\mathcal{R}_{det}$ can be computed entirely within a classical deterministic database, making it backwards compatible with legacy ETL components.

The best-guess relation $\mathcal{R}_{guess}$ is constructed in two stages. First, the deterministic database system executes $Q(D)$. As the classical database streams results for $Q(D)$, the shim layer evaluates each e_i and ϕ based on the valuation given by $\texttt{argmax}_v(P(v))$, the most-likely possible world. Field values or row confidences in the best guess relation that depend on v are annotated in the shim layer's output. Legacy applications can quietly ignore this annotation. In uncertainty-aware applications, this annotation is used to indicate which parts of the result are uncertain to the end-user.

Example 7. *Continuing Example 6, the database now responds to Alice's query with the most-likely result shown in Figure 9. Every non-deterministic (i.e., guessed) field is annotated with an asterisk. Every row with a non-deterministic condition is similarly marked. A footer indicates how many rows were dropped due to a non-deterministic condition evaluating to false in the most likely possible world. Note that this best-guess estimate is not entirely accurate:* `evaluation` *has been mapped to* `review_ct`*, and* `rating` *has not been matched, resulting in best-effort guesses of 2 and 5 for the last two rows of the result. In spite of the error, Alice can quickly see the role uncertainty plays in her results.*

5.2 Summarizing Result Quality

Once the user is made aware that parts of a query result may be uncertain, two questions likely to be asked are "How bad?" and "Why?". Answering the latter question is trivial: $\mathcal{F}$ contains a reference to all of the variables that introduce uncertainty, each of which is explicitly linked to the lens that constructed it. In other words, $\mathcal{F}$ serves as a form of provenance that can be used to explain sources of uncertainty to the end-user.

The former question requires us to develop a notion of result quality. Our approach is based on the ideas of *noise*: intuitively, the less noise is present in the model, the higher the quality of the best-guess relation's predictions. We abstractly define result quality as the level of confidence that the user should have in the annotated best-guess results. These results include both non-deterministic attribute values, as well as possible tuples.

Recall that a non-deterministic value in $\mathcal{R}_{guess}$ is obtained from non-deterministic expressions in $\mathcal{R}_{guess}$. Numerous metrics that effectively convey the quality of attribute values drawn from a probabilistic database have been proposed, including pessimistic hard bounds [29], variance [24,27], and $\epsilon - \delta$ bounds [20].

As a simplification, we assume that with respect to understanding uncertainty in a specific attribute value, the cognitive burden on the user is constant, while for the presence or absence of rows in the output, it scales linearly. Intuitively, guessing wrong about tuple presence can mean the difference between overwhelming the user with a flood of unnecessary results, and hiding the presence of potentially critical information. Under this assumption, tuple-level uncertainty adds more noise to the result, and we will focus primarily on this type of uncertainty from here on.

The appearance of a tuple t in a query result is determined by the ground truth of its local condition $t.\phi$. Valuations $v(\Sigma)$ map $t.\phi$ to a deterministic boolean value $t.\phi[v]$. From the PC-Table's probability measure $P(v)$, we get the binomial distribution $P(t.\phi[v])$, often called the confidence of t. We use the confidence of t to measure how difficult it is for the analyst to predict the ground truth of $t.\phi$. Intuitively, if $P(t.\phi[v])$ is skewed towards 0 or 1, we expect to predict the value of $t.\phi$ with reasonable accuracy; on the other hand, if $P(t.\phi[v])$ is a fair coin flip, we have no reliable information about the expected result of $t.\phi$. It is natural to use Shannon entropy as a metric to quantify the quality of the query result. We define the entropy of a tuple in terms of its confidence $p_t = P(t.\phi[v])$ as:

$$entropy(t) = -(\; p_t \cdot \log_2(p_t) + (1 - p_t) \cdot \log_2(1 - p_t) \;)$$

Efficiently approximating tuple confidences by sampling from $P(v)$ is well studied in probabilistic databases [22, 34], and we use similar techniques for estimating tuple entropies.

To unify the individual per-attribute and per-row metrics, we define a relation-wise noise function $\mathcal{N}(\mathcal{R})$ as a linear combination of individual metrics. For example, a relation $\mathcal{R}$ without non-deterministic attributes might have $\mathcal{N}(\mathcal{R}) = \sum_{t \in \mathcal{R}} entropy(t)$. To account for the entropy generated by non-deterministic attributes, we start with the intuition that each attribute in the output provides $\frac{1}{N}$th of the information content of a tuple, where N is the arity of $\mathcal{R}$. Thus, by default, we assume each non-deterministic value contributes to the noise seen in the final result a fraction in the range $[0, \frac{1}{N}]$ inversely proportional to the attribute's estimated variance.

6. FEEDBACK

When the analyst is given a query result $\mathcal{R}$ that does not meet her quality expectations, she can allocate additional resources for gathering more evidence. For example, she may spend some time gathering ground-truth values for variables in the output C-Table. By construction, variables represent uncertainty about basic facts. For example, a schema matching lens generates expressions of the form $Var('rat = eval')$ that could be stated as a simple question like "*Do* `rating` *and* `evaluation` *mean the same thing?*". Replacing variables with their ground truths means performing these basic curation tasks, with the goal of reducing the

noise seen in the final result. Since $\mathcal{N}(\mathcal{R})$ depends on the entropy generated by the tuples in $\mathcal{R}$, in expectation, each curation task will reduce the noise seen in the final result. Identifying the variable[1] that affects $\mathcal{N}(\mathcal{R})$ the most is not trivial: depending on $\{ t.\phi \mid t \in \mathcal{R} \}$ some variables may generate more noise in the final result than others. In a perfect world, the analyst would simply replace variables in $\mathcal{R}$ until the required level of quality is reached. In the real world, curation tasks are expensive, and the optimal cleaning strategy depends on both quality goals and budget constraints. Hence, deciding a good strategy is essentially a resource allocation problem. We assume that a cost model is given by each lens in the form of a cost function $c(\cdot)$. This function maps variables to positive real numbers that represent the effort, or dollar cost of discovering the ground truth for the given variable.

6.1 Prioritizing Curation Tasks

Prioritizing curation tasks is a dynamic decision process, as the outcome of one curation task affects the choice of the next one to be performed. Let's assume, for the moment, that the analyst has no budget constraints and her goal is simply to determine the ground truth of a given condition formula ϕ, minimizing the *expected* amount of resources spent in the process. In the literature, this optimization problem is known as *stochastic boolean function evaluation* [11, 39]. Both exact and approximated algorithms have been proposed for several classes of formulas. In its general form, the problem can be thought of as a *Markov Decision Process*[2], having one state for each partial assignment to the variables in ϕ and one action for each variable (a curation task). Rewards are determined by $-c(\cdot)$ and state-transitions are determined by $P(v)$. Final states consist of assignments that make ϕ either true or false with certainty. The planning horizon is finite and equal to the number of variables in ϕ. A simple solution to the problem consists of a *policy*, prescribing a curation task for each non-terminating assignment to perform. The application of a policy is an interactive process: the system instructs the analyst to address a particular curation task ("*Do* `rating` *and* `evaluation` *mean the same thing?*"), the analyst provides the required ground truth, and asks the system for the next move. This feedback loop continues until the deterministic value of ϕ is obtained. As a baseline for evaluation (Section 7), On-Demand ETL implements a naïve algorithm for computing policies of this kind, named naïve minimum expected total cost (NMETC).

6.2 Balancing Result Quality and Cost

Real-world ETL applications are unlikely to be free from budget constraints. Even when budget is not a problem, the average analyst will rarely aim for perfect information. Instead, she would rather target a reasonable approximation of the value of ϕ, setting an upper-bound on the entropy of the formula. Hence, we generalize the approach discussed above and make the assumption that the analyst wants to plan her curation tasks so to maximize a hidden value function $\mathcal{V}(\cdot)$, which depends on $c(\cdot)$ and $\mathcal{N}(\cdot)$ and is unknown to the system. Clearly, we assume $\mathcal{V}(\cdot)$ decreases monotonically as the cumulative cost increases, and increases monotonically as the noise decreases. In simple words, $\mathcal{V}$ determines how much the analyst is willing to pay for an improvement in the estimation of the value of ϕ, on a case-by-case basis. We call this trade-off the *cost of perfect information* (CPI). Since the details of $\mathcal{V}(\cdot)$ are unknown, the goal of the system is to propose several candidate policies. Each policy should guarantee a certain expected entropy at the price of a certain expected cumulative cost. The user is then able to choose the candidate policy that best matches her hidden value function. Since the analyst may be subject to budget constraints hidden to the system, the candidate list includes greedy versions of the policies, computed progressively over limited planning horizons. Inspired by the algorithms EG2 [33], CS_ID3 [38] and CS_C4.5 [14], On-Demand ETL supports the following four greedy strategies:

[1]A generalization to curation tasks that span multiple variables is possible, but left to future work.

[2]Clearly, the translation of the problem into a Markov Decision Process is not polynomial, neither in space nor in time.

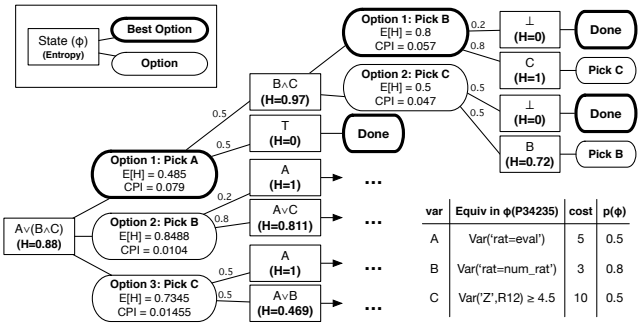

Figure 10: An example of the CS_IDX algorithm optimizing CPI.

Distances Metric	Correct Matches by Index								
	1	2	3	4	5	6	...	9	n/a
Levenstein	24	2	2	0	0	1	0	1	0
JaroWinkler	20	4	2	1	1	0	0	1	1
NGram	25	0	3	1	0	0	0	0	1

Figure 11: Distance evaluated by the index of the correct match in the ranked list of matches output by the algorithm, or n/a if the correct match was discarded. Results include 30 test cases.

Algorithm	$CPI(v_i)$
EG2	$(2^{IG[\mathcal{R}(v_i)]} - 1)/c_i$
CS_ID3	$(IG[\mathcal{R}(v_i)]^2)/c_i$
CS_IDX	$IG[\mathcal{R}(v_i)]/c_i$
CS_C4.5	$IG[\mathcal{R}(v_i)]/\sqrt{c_i}$

Here, IG denotes the *information gain*, or the reduction in noise produced by the curation task on variable v_i.

EXAMPLE 8. *Consider the condition ϕ for P34235 in Figure 7, which has the form $A \vee (B \wedge C)$. Figure 10 illustrates the decision tree that ranks curation tasks (the three variables), given lens-defined ground-truth costs and marginal probabilities as shown in the figure. The expected entropy after performing the curation task for v_i, denoted by $E[H(v_i)]$, is computed as a weighted average over all possible outcomes of the task. $CPI(v_i)$ is computed according to the CS_IDX formula given above, with $IG[\mathcal{R}(v_i)] = H - E[H(v_i)]$.*

7. EXPERIMENTS

In this section we show the feasibility of On-Demand ETL and explore several points in its design space. Specifically,

we find that: (i) The greedy approach of minimizing CPI produces higher-quality query results at lower costs than optimizing for total cost when the hidden value function is not known, (ii) The precise formula used to compute CPI is not critical to achieving high quality results, (iii) When composing lenses, order is relevant, as open-ended lenses like domain-constraint repair can fix issues created by other lenses earlier in the pipeline, and (iv) Tree-based classifiers work best for domain constraint repair lenses.

7.1 Experimental Setup

Our experimental setup consists of three data sets drawn from publicly available sources. To simulate data-entry error, a portion of the data values are randomly removed. To simulate an analyst's querying behavior, we identify one attribute in each data set, remove the attribute from the source data, and use a tree-based classifier to construct a query that the analyst might issue to recover the attribute. For each data source, we also provide simulated user-defined costs for available curation tasks.

Product Data. We used the product search APIs of two major electronics retailers[3] to extract product data for a total of 586 items (346 and 240 items respectively). The products extracted fall into three categories: TVs, cell phones and laptops. There are ten attributes in the schema of each data source. We randomly replaced 45% of the data values with `NULL`, and coerce both data sets into the schema of a third retailer's search API[4]. On this data-set, we simulate an analyst trying to predict what factors go into a good product rating. We trained a tree based classifier on the partial data, used the resulting decision tree to simulate the analyst's query:

```
SELECT * FROM products
WHERE brand in (4,5,6,7) AND category in (1,2,3)
AND totalReviews < 3 AND instoreAvailability = 0
AND (onsale_clearance = 0 OR (quantityAvailableHint = 0
        AND shippingCost in (0,1,2,3,4)));
```

Curation tasks fall into four categories: Trivial schema matching tasks, simple data gathering of boolean values like item availability, more detailed data gathering of values like strings, and more open-ended data gathering tasks such as soliciting item reviews from focus groups. We assign the cost of these four curation tasks to be 1, 5, 10, and 30 units of effort respectively.

Credit Data. We used the German and Japanese Credit Data-sets from the UCI data repository [31]. These data sets contain 1000 and 125 items, respectively, and have 20 and 8 attributes, respectively. As in the product dataset, we randomly replaced 45% of data values with `NULL` values. The German data is coerced into the schema of the Japanese data set. We simulate an analyst searching for low-risk customers by using the following classifier-constructed query:

```
SELECT * FROM PD
WHERE (purchase_item < 0.5 AND monthly_payment >= 3.5
       AND num_of_years_in_company in (2,3) )
   OR (num_of_months >= 6.5 AND married_gender >= 2.5);
```

In addition to trivial schema-matching tasks, there are two kinds of missing attributes: Some attributes can be computed from other values (e.g., a customer's monthly payment can be computed from the total loan value and duration) or retrieved from other parts of the bank. Other attributes require personal information about the client. We set the cost of these three classes of task to be 1, 10, and 20 respectively.

Real Estate Data. We obtain house listing information from five real estate websites[5]. Unlike the prior cases, where the number of data-sets is small and the number of records per data set is comparatively large, the Real Estate data set emulates web-tables where the number of data sets is comparatively large and the number of records per data set is small. We further reduce the data size by randomly sampling only 20 items from each dataset. As above, 45% of data values are replaced by `NULL`. All source data is coerced into a globally selected target-schema. For this data set, we simulate an analyst trying to identify houses likely to have a price rating of 3 out of 4 points, where all curation tasks have a flat cost of 1:

```
SELECT * FROM PD WHERE Baths < 2.5
 AND (Beds >= 3.5 OR Garage >= 2.5);
```

7.2 Lens Configuration

For each experiment, we simulate an analyst using the Domain-Constraint Repair and Schema Matching lenses described in Section 3.2. We use the values randomly removed from each data-set as ground truths for evaluating the Domain-Constraint Repair lens, and a manually defined mapping as ground truth for evaluating the Schema Matching lens. Both lenses define curation tasks as summarized in the description of each data-set.

Schema Matching. We employ a combination of schema matchers [35] that hybridize structure-level and element-level matchers. We first use constraint-based (data type and data range) filters to eliminate candidate matches, and then use the Levenstein, JaroWinkler, and NGram distance metrics to rank attribute matches based on string similarity with a threshold. The performance of these three strategies is shown in Figure 11. We take an average of the similarity scores from the three distance metrics and normalize them to approximate the probability of a match.

Domain Constraint Repair. We use incremental classifiers from the massive online analysis (MOA) framework [6] for the Domain-Constraint Repair lens. We use classifiers in five categories: active, bayes, stochastic gradient descent, ensemble and tree. For each attribute in the source table, we train a classifier using tuples in which the value is not missing. The estimation results for missing values are probability distributions of all candidate repairs.

7.3 Ranking Curation Tasks

We compare three ranking policies over curation tasks (one per variable v_i). Each policy implements a ranking over the available curation tasks, the top-ranked task is performed. Curation costs are as listed in Section 7.1.

NMETC. This (naive, exponential-time) policy calculates an optimal long-term strategy based on repeatedly selecting the variable that minimizes the global expected total cost of obtaining a deterministic value. Potential curation tasks are ranked in descending order of their expected total cost, weighted over all possible paths through the decision tree.

[3] `http://developer.bestbuy.com/documentation/products-api`
`http://developer.walmartlabs.com/docs/read/Search_API`

[4] `http://go.developer.ebay.com`

[5] `http://pages.cs.wisc.edu/~anhai/wisc-si-archive/domains/real_estate1.html`

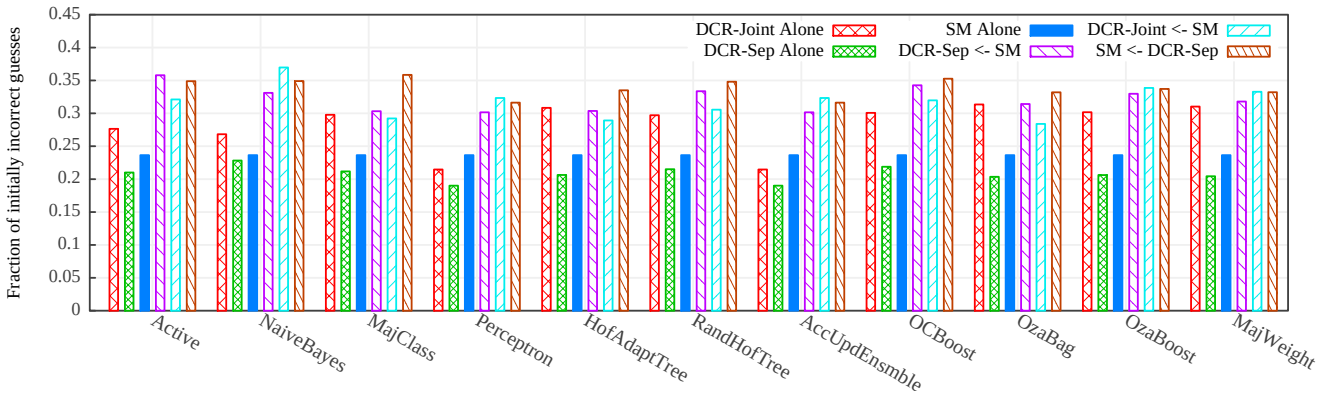

Figure 12: Composability of schema matching and domain repair for 11 classifiers (Product Data)

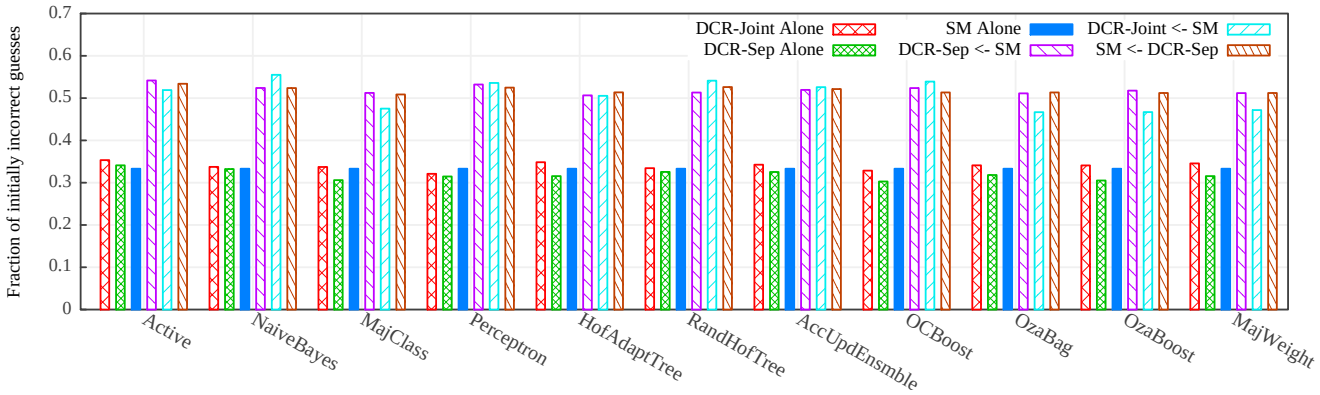

Figure 13: Composability of schema matching and domain repair for 11 classifiers (Credit Data)

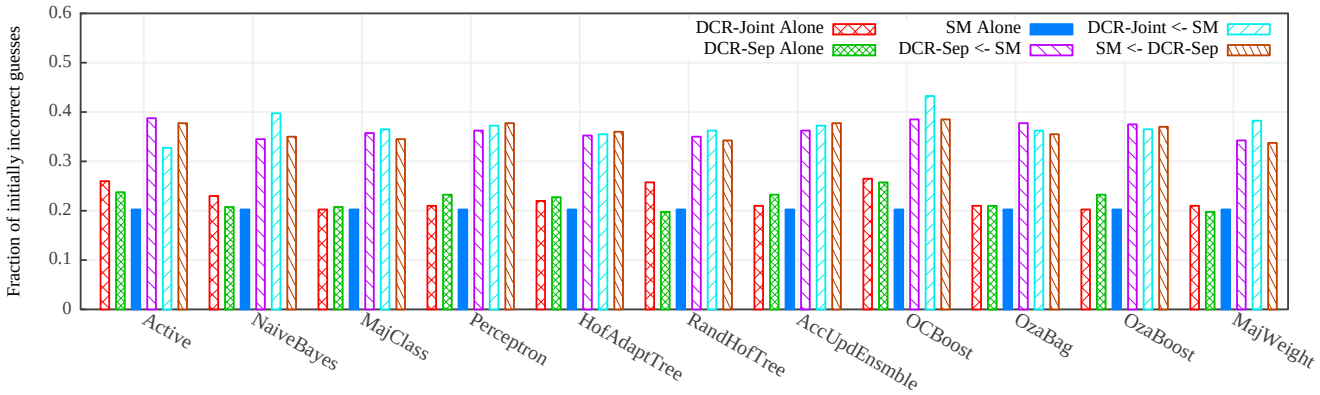

Figure 14: Composability of schema matching and domain repair for 11 classifiers (Real Estate Data)

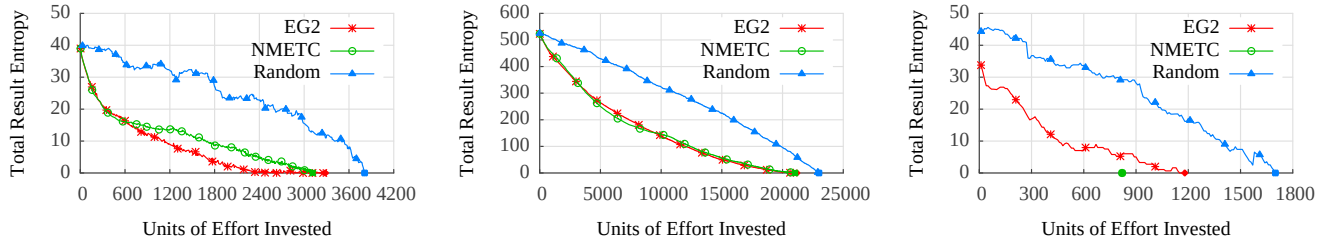

Figure 15: Performance comparison for different methods on query results of (a) product, (b) credit, and (c) real estate data-sets. Detailed step-by-step performance for the naive strategy is computationally infeasible for the real estate data-set, so only final results are shown.

Greedy (CPI). CPI based policies rank curation tasks in ascending order of CPI. All four CPI-based metrics produce virtually identical results for each of our test cases, so only results for the EG2 implementation of CPI are shown. The scoring function for the greedy strategy is the CPI itself.

Random. The random strategy ranks curation tasks in a random order, and provides a baseline for other methods.

7.4 Lens Composition

We first explore the default (i.e., pre-feedback) behavior of lenses under composition. Of interest to us are three questions: (i) Does the machine learning model used for domain-constraint repair matter? (ii) Can lenses be composed together safely? and (iii) Does the order in which lenses are composed matter? We evaluate the accuracy of the output of Schema Matching (**SM**) and Domain Constraint Repair (**DCR**) lenses applied to each data set. Figures 12, 13, and 14 show the fraction of cells in the output of each query that correspond to ground truth results *before any feedback is gathered*. Our results include two variants of Domain-Constraint Repair, one where all data sources are combined before being repaired (**DCR-Joint**), and one where all data sources are repaired independently (**DCR-Sep**). We consider three different lens combinations: **DCR-Joint** or **DCR-Sep** applied to the output of **SM** (**DCR-Joint** ← **SM** and **DCR-Sep** ← **SM**, respectively), and **SM** applied to the output of **DCR-Sep** (**SM** ← **DCR-Sep**). The remaining combination is not possible, as **DCR-Joint** requires **SM** first to create a unified schema. For comparison, we also present results for each lens alone, using ground truth values for the output of the other lens. Performance results are shown for 11 different machine learning models from the MOA framework.

In general, the performance of different orderings of lenses appears to differ by only a small amount, generally under 5%. An exception appears in the Product data set (Figure 12), where we can see for all estimation methods, applying **SM** first and then applying **DCR-Joint** produces the best results. By being trained on both data-sets together, **DCR** is able to detect and correct some schema matching errors. Moreover, in all cases, the combined error of composing both lenses is lower than the error introduced by either lens individually. This shows that composing different lenses is feasible. By comparison, the Credit data set (Figure 13) is extremely noisy — both lenses have initial error rates around 34%. Hence, too much noise exists in the data, and different lens orderings have little effect.

The observation above shows that reordering lenses can be beneficial in some cases. Given analyses of lenses, we can help users reorder lenses to achieve better accuracy. Another observation is that when the data is sufficiently correlated for **DCR** to have relatively small error rates, the error rate of **DCR-Joint** is typically lower than **DCR-Sep**. Intuitively, if inter-attribute correlations from different data sets are similar, **DCR-Joint** is effectively being trained on a larger dataset.

7.5 On-Demand ETL

We next study the effectiveness of On-Demand ETL and CPI-based heuristics. To study the efficacy of our CPI-based approach, we investigate the performance of different ranking strategies on product,credit, and real estate data-sets. We use the same basic setup as described above for each data-set. We present results using **DCR-Joint** applied to **SM**, but all three composition orders behave similarly.

Figure 15 shows the total entropy remaining in the query results after multiple rounds of feedback, in which the analyst repeatedly performs the curation task with best score. The rightmost dot for each line denotes the point at which the noise in the query result relation reaches zero. Since the analyst may have a limited budget to improve the quality of very noisy query results, the goal is to provide the highest level of noise reduction with as low a total cost as possible, or in other words to create a curve with as little volume under the curve as possible.

EG2 denotes the greedy EG2-based CPI heuristic (all other CPI heuristics behave almost identically). The curve is very steep until the final curation tasks, allowing EG2 to produce high-quality results with minimal investment.

NMETC denotes the naive brute-force cost-optimization strategy, while **Random** denotes a completely random ordering of curation tasks. Although the brute-force strategy produces a completely reliable result at the lowest cost, it does so at the expense of short-term benefits. For the product data-sets, a result with virtually no entropy is reached after 24,000 units of cost, while the brute force strategy requires over 30,000 units. Although NMETC requires the lowest cost to obtain a deterministic query result, it may not be optimal for a limited budget or when the user's value function is not known.

8. CONCLUSIONS

We have presented On-Demand ETL, which generalizes task-specific on-demand curation solutions such as Paygo. On-Demand ETL enables composable non-deterministic data processing operators called Lenses that provide the illusion of fully cleaned relational data that can be queried using standard SQL. Lenses use PC-Tables to encode output, and can be deployed in traditional, deterministic database environments using Virtual C-Tables. On-Demand ETL supports best-effort guesses at the contents a PC-Table, evaluation of quality measures over a PC-Table, and a family of heuristics for prioritizing curation tasks called CPI. We have demonstrated the feasibility and need for On-Demand ETL, and the effectiveness of CPI-based heuristics.

9. REFERENCES

[1] Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Chris Hayworth, Shubha U. Nabar, Tomoe Sugihara, and Jennifer Widom. Trio: A system for data, uncertainty, and lineage. In *VLDB*, pages 1151–1154. ACM, 2006.

[2] Bogdan Alexe, Laura Chiticariu, Renée J. Miller, and Wang Chiew Tan. Muse: Mapping understanding and design by example. In *ICDE*, pages 10–19. IEEE, 2008.

[3] Sarah R. Allen, Lisa Hellerstein, Devorah Kletenik, and Tonguç Ünlüyurt. Evaluation of DNF formulas. In *ISAIM*, 2014.

[4] Yael Amsterdamer, Susan B. Davidson, Tova Milo, Slava Novgorodov, and Amit Somech. OASSIS: query driven crowd mining. In *SIGMOD*, pages 589–600. ACM, 2014.

[5] Khalid Belhajjame, Norman W. Paton, Alvaro A. A. Fernandes, Cornelia Hedeler, and Suzanne M. Embury.

User feedback as a first class citizen in information integration systems. In *CIDR*, pages 175–183, 2011.

[6] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *JMLR*, 11:1601–1604, 2010.

[7] Philippe Bonnet and Anthony Tomasic. Partial answers for unavailable data sources. In *FQAS*, volume 1495, pages 43–54. Springer, 1998.

[8] Jihad Boulos, Nilesh N. Dalvi, Bhushan Mandhani, Shobhit Mathur, Christopher Ré, and Dan Suciu. MYSTIQ: a system for finding more answers by using probabilities. In *SIGMOD*, pages 891–893. ACM, 2005.

[9] Huiping Cao, Yan Qi, K. Selçuk Candan, and Maria Luisa Sapino. Feedback-driven result ranking and query refinement for exploring semi-structured data collections. In *EDBT*, volume 426, pages 3–14. ACM, 2010.

[10] Xiaoyong Chai, Ba-Quy Vuong, AnHai Doan, and Jeffrey F. Naughton. Efficiently incorporating user feedback into information extraction and integration programs. In *SIGMOD*, pages 87–100. ACM, 2009.

[11] Amol Deshpande, Lisa Hellerstein, and Devorah Kletenik. Approximation algorithms for stochastic boolean function evaluation and stochastic submodular set cover. In *SODA*, pages 1453–1467. SIAM, 2014.

[12] Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa. Data exchange: Getting to the core. In *PODS*, pages 90–101. ACM, 2003.

[13] Robert Fink, Andrew Hogue, Dan Olteanu, and Swaroop Rath. Sprout^2: a squared query engine for uncertain web data. In *SIGMOD*, pages 1299–1302. ACM, 2011.

[14] Alberto Freitas, Altamiro Costa-Pereira, and Pavel Brazdil. Cost-sensitive decision trees applied to medical data. In *DaWaK*, pages 303–312. 2007.

[15] Amélie Gheerbrant, Leonid Libkin, and Cristina Sirangelo. When is naive evaluation possible? In *PODS*, pages 75–86. ACM, 2013.

[16] Todd J. Green, Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen. Provenance in ORCHESTRA. *DEBU*, 33(3):9–16, 2010.

[17] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, pages 31–40. ACM, 2007.

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

[19] Xintong Guo, Hongzhi Wang, Yangqiu Song, and Gao Hong. Brief survey of crowdsourcing for data mining. *ESWA*, 41(17):7987–7994, 2014.

[20] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. Online aggregation. In *SIGMOD*, pages 171–182, 1997.

[21] Zhen Hua Liu and Dieter Gawlick. Management of flexible schema data in RDBMSs - opportunities and limitations for NoSQL. In *CIDR*, 2015.

[22] Jiewen Huang, Lyublena Antova, Christoph Koch, and Dan Olteanu. MayBMS: a probabilistic database management system. In *SIGMOD*, pages 1071–1074. ACM, 2009.

[23] Tomasz Imielinski and Witold Lipski Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.

[24] Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Christopher Jermaine, and Peter J Haas. MCDB: a monte carlo approach to managing uncertain data. In *SIGMOD*, pages 687–700, 2008.

[25] Shawn R. Jeffery, Michael J. Franklin, and Alon Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In *SIGMOD*, pages 847–860. ACM, 2008.

[26] Haim Kaplan, Eyal Kushilevitz, and Yishay Mansour. Learning with attribute costs. In *STOC*, pages 356–365, 2005.

[27] Oliver Kennedy and Christoph Koch. PIP: A database system for great and small expectations. In *ICDE*, pages 157–168, 2010.

[28] Oliver Kennedy and Suman Nath. Jigsaw: efficient optimization over uncertain enterprise data. In *SIGMOD*, pages 829–840. ACM, 2011.

[29] Oliver Kennedy, Ying Yang, Jan Chomicki, Ronny Fehling, ZhenHua Liu, and Dieter Gawlick. Detecting the temporal context of queries. In *BIRTE*, volume 206 of *LNBIP*, pages 97–113. 2015.

[30] Willis Lang, Rimma V. Nehme, Eric Robinson, and Jeffrey F. Naughton. Partial results in database systems. In *SIGMOD*, pages 1275–1286. ACM, 2014.

[31] M. Lichman. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2013.

[32] Yosi Mass, Maya Ramanath, Yehoshua Sagiv, and Gerhard Weikum. IQ: the case for iterative querying for knowledge. In *CIDR*, pages 38–44, 2011.

[33] Marlon Núñez. The use of background knowledge in decision tree induction. *JMLR*, 6:231–250, 1991.

[34] Dan Olteanu, Jiewen Huang, and Christoph Koch. Approximate confidence computation in probabilistic databases. In *ICDE*, pages 145–156. IEEE, 2010.

[35] Erhard Rahm and Philip A Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

[36] Sarvjeet Singh, Chris Mayfield, Sagar Mittal, Sunil Prabhakar, Susanne Hambrusch, and Rahul Shah. Orion 2.0: Native support for uncertain data. In *SIGMOD*, pages 1239–1242. ACM, 2008.

[37] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. Probabilistic databases. *Synthesis Lectures on Data Management*, 3(2):1–180, 2011.

[38] Ming Tan and Jeffrey C. Schlimmer. Cost-sensitive concept learning of sensor use in approach and recognition. In *MLSB*, 1989.

[39] Tonguç Ünlüyurt. Sequential testing of complex systems: a review. *DAM*, 142(1-3):189–205, 2004.

[40] Susan V. Vrbsky and Jane W.-S. Liu. APPROXIMATE - A query processor that produces monotonically improving approximate answers. *IEEE TKDE*, 5(6):1056–1068, 1993.

[41] Daisy Zhe Wang, Eirinaios Michelakis, Minos Garofalakis, and Joseph M. Hellerstein. Bayesstore: Managing large, uncertain data repositories with probabilistic graphical models. *Proc. VLDB Endow.*, 1(1):340–351, August 2008.

[42] Ying Yang. On-demand query result cleaning. In *VLDB PhD Workshop*, 2014.