# Adaptive Schema Databases *

William Spoth[b], Bahareh Sadat Arab[i], Eric S. Chan[o], Dieter Gawlick[o], Adel Ghoneimy[o], Boris Glavic[i], Beda Hammerschmidt[o], Oliver Kennedy[b], Seokki Lee[i], Zhen Hua Liu[o], Xing Niu[i], Ying Yang[b]
b: University at Buffalo i: Illinois Inst. Tech. o: Oracle
{wmspoth|okennedy|yyang25}@buffalo.edu
{barab|slee195|xniu7}@hawk.iit.edu bglavic@iit.edu
{eric.s.chan|dieter.gawlick|adel.ghoneimy|beda.hammerschmidt|zhen.liu}@oracle.com

## ABSTRACT

The rigid schemas of classical relational databases help users in specifying queries and inform the storage organization of data. However, the advantages of schemas come at a high upfront cost through schema and ETL process design. In this work, we propose a new paradigm where the database system takes a more active role in schema development and data integration. We refer to this approach as *adaptive schema databases* (*ASDs*). An ASD ingests semi-structured or unstructured data directly using a pluggable combination of extraction and data integration techniques. Over time it discovers and adapts schemas for the ingested data using information provided by queries and user-feedback. In contrast to relational databases, ASDs maintain multiple *schema workspaces* that represent individualized views over the data which are fine-tuned to the needs of a particular user or group of users. A novel aspect of ASDs is that probabilistic database techniques are used to encode ambiguity in automatically generated data extraction workflows and in generated schemas. ASDs can provide users with context-dependent feedback on the quality of a schema, both in terms of its ability to satisfy a user's queries, and the quality of the resulting answers. We outline our vision for ASDs, and present a proof-of concept implementation as part of the Mimir probabilistic data curation system.

## 1. INTRODUCTION

Classical relational systems rely on schema-on-load, requiring analysts to design a schema upfront before posing any queries. The schema of a relational database serves both a navigational purpose (it exposes the structure of data for querying) as well as an organizational purpose (it informs storage layout of data). If raw data is available in unstructured or semi-structured form, then an ETL (i.e., Extract, Transform, and Load) process needs to be designed to translate the input data into relational form. Thus, classical relational systems require a lot of upfront investment. This makes them unattractive when upfront costs cannot be amortized, such as in workloads with rapidly evolving data or where individual elements of a schema are queried infrequently. Furthermore, in settings like data exploration, schema design simply takes too long to be practical.

Schema-on-query is an alternative approach popularized by NoSQL and Big Data systems that avoids the upfront investment in schema design by performing data extraction and integration at query-time. For semi-structured and unstructured data, data integration tasks such as *natural language processing* (NLP), entity resolution, and schema matching have to be performed on a per-query basis. This enables data to be queried immediately, but sacrifices the navigational and performance benefits of a schema and frequently leads to reduced data quality (e.g., many task-specific versions of a dataset) and lost productivity (increased work being done at query time).

One significant benefit of schema-on-query is that queries often only access a subset of all available data. Thus, to answer a specific query, it may be sufficient to limit integration and extraction to only relevant parts of the data. Furthermore, there may be multiple "correct" relational representations of semi-structured data and what constitutes a correct schema may be highly application dependent. This implies that imposing a single flat relational schema will lead to schemas that are the lowest common denominator of the entire workload and not well-suited for *any* of the workload's queries. Consider a dataset with tweets and re-tweets. Some queries over a tweet relation may want to consider re-tweets as tweets while others may prefer to ignore them.

In this work, we propose ***adaptive schema databases*** (*ASDs*), a new paradigm that addresses the shortcomings of both the classical relational and Big Data approaches mentioned above. ASDs enjoy the navigational and organizational benefits of a schema without incurring the upfront investment in schema and ETL process development. This is achieved by automating schema inference, information extraction, and integration to reduce the load on the user. Instead of enforcing one global schema, ASDs build and adapt idiosyncratic schemas that are specialized to users' needs.

We propose the probabilistic framework shown in Figure 1 as a reference architecture for ASDs. When unstructured or semi-structured data are loaded into an ASD, this framework applies a sequence of data extraction and integration components that we refer to as an **extraction workflow** to compute possible relational schemas for this data.

*Authors Listed in Alphabetical Order

Any existing techniques for schema extraction or information integration can be used as long as they can expose the ambiguity inherent in these tasks in a probabilistic form. For example, an entity resolution algorithm might identify two possible instances representing the same entity. Classically, the algorithm would include heuristics that resolve this uncertainty and allow it to produce a single deterministic output. In contrast, our approach requires that extraction workflow stages produce non-deterministic, probabilistic outputs instead of using heuristics. The final result of such an **extraction workflow** is a **set of candidate schemas** and a probability distribution describing the likelihood of each of these schemas. In ASDs, users create **schema workspaces** that represent individual views over the schema candidates created by the extraction workflow. The schema of a workspace is created incrementally based on queries asked by a user of the workspace. Outputs from the extraction workflow are dynamically imported into the workspace as they are used, or users may suggest new relations and attributes not readily available to the database. In the latter case, the ASD will apply schema matching to determine how the new schema elements relate to the elements in the candidate schemas. Similar to extraction workflows, this schema matching consists of candidate matches with associated probabilities. Based on these probabilities and feedback provided by users through queries, ASDs can incrementally modify the extraction workflow and schema workspaces to correct errors, to improve their quality, to adapt to changing requirements, and to evolve schemas based on updates to input datasets. The use of feedback is made possible based on our previous work on probabilistic curation operators [19] and provenance [1]. By modelling schemas as views over a non-relational input dataset, we decouple data representation from content. Thus, we gain flexibility in **storage organization** — for a given schema we may choose not to materialize anything, we may fully materialize the schema, or materialize selectively based on access patterns.
Concretely, this paper makes the following contributions:

- We introduce our vision of ASDs, which enable access to unstructured and semi-structured data through personalized relational schemas.
- We show how ASDs leverage information extraction and data integration to automatically infer and adapt schemas based on evidence provided by these components, by queries, and through user feedback.
- We show how ASDs enable adaptive task-specific "personalized schemas" through schema workspaces.
- We illustrate how ASDs communicate potential sources of error and low-quality data, and how this communication enables analysts to provide feedback.
- We present a proof-of-concept implementation of ASDs based on the Mimir [14] data curation system.
- We demonstrate through experiments that the instrumentation required to embed information extraction into an ASD has minimal overhead.

## 2. EXTRACTION AND DISCOVERY

An ASD allows users to pose relational queries over the content of semi-structured and unstructured datasets. We call the steps taken to transform an input dataset into relational form an **extraction workflow**. For example, one possible extraction workflow is to first apply NLP to extract

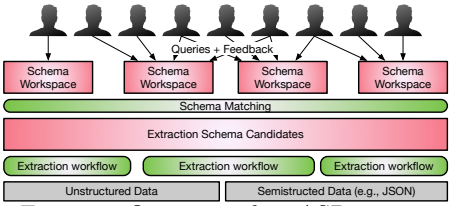


Figure 1: Overview of an ASD system

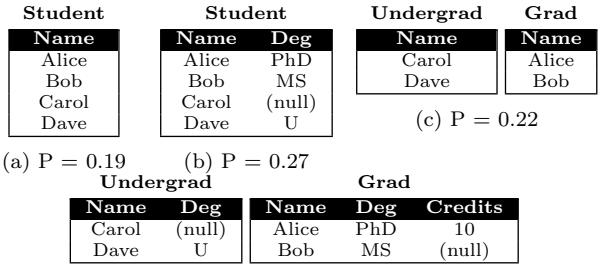
semi-structured data (e.g., RDF triples) from an unstructured input, and then shred the semi-structured data into a relational form. The user can then ask queries against the resultant relational dataset. Such a workflow frequently relies on heuristics to create seemingly deterministic outputs, obscuring the possibility that the heuristics may choose incorrectly. In an ASD, one or more modular information extraction components instead produce a *set* of possible ways to shred the raw data with associated probabilities. This is achieved by exposing ambiguity arising in the components of an extraction workflow. Any NLP, information retrieval, and data integration algorithm may be used as an information extraction component, as long as the ambiguity in its heuristic choices can be exposed. The set of schema candidates are then used to seed the development of schemas individualized for a particular purpose and/or user. The ASD's goal is to figure out which of these candidates is the correct one for the analyst's current requirements, to communicate any potential sources of error, and to adapt itself as those requirements change.

**Extraction Schema Candidates.** When a collection of unstructured or semi-structured datasets $D$ is loaded into an ASD, then information extraction and integration techniques are automatically applied to extract relational content and compute candidate schemas for the extracted information. The choice of techniques is based on the input data type (JSON, CSV, natural language text, etc...). We associate with this data a **schema candidate set** $\mathcal{C}_{ext} = (\mathbf{S_{ext}}, P_{ext})$ where $\mathbf{S_{ext}}$ is a set of candidate schemas and $P_{ext}$ is a probability distribution over this schema. We use $S_{max}$ referred to as the **best guess schema** to denote $\arg\max_{S \in \mathbf{S_{ext}}}(P(S))$, i.e., the most likely schema from the set of candidate schemas. Similar data models have been studied extensively in probabilistic databases, allowing us to adapt existing work on probabilistic query processing, while still supporting a variety of information extraction, natural language processing, and data integration techniques.

EXAMPLE 1. *As a running example throughout the paper, consider a JSON document (a fragment is shown below) that stores a college's enrollment. Assume that for every graduate student we store name and degree, but only for some students there is a record of the number of credits achieved so far. For undergraduates we only store a name, although several undergraduates were accidentally stored with a degree. A semi-structured to relational mapper may extract schema candidates as shown in Figure 2.*

```
{"grad":{"students":[
  {name:"Alice",deg:"PhD",credits:"10"},
  {name:"Bob",deg:"MS"}, ...]},
 "undergrad":{"students":[
  {name:"Carol"},{name:"Dave",deg:"U"}, ...]}}
```

**Querying Extracted Data.** We would like to expose to users an initial schema that allows $D$ to be queried (i.e.,




(a) P = 0.19 (b) P = 0.27 (c) P = 0.22

(d) P = 0.32

Figure 2: Extracted Schema Candidate Set and Data

the best guess schema $S_{max}$), while at the same time acknowledging that this schema may be inappropriate for the analyst, incorrect for her current task, or simply outright wrong. Manifestations of extraction errors appear in three forms: (1) A query incompatible with $S_{max}$, (2) An update with data that violates $S_{max}$, or (3) An extraction error resulting in the wrong data being presented to the user. The first two errors are overt and, thus easy for the database to quickly detect. In both cases, the primary challenge is to help the user first determine whether the operation was correct, and if necessary, to repair the schema accordingly. Here, the distribution $P_{ext}$ serves as a metric for schema suggestions. Given a query (resp., update or insert) $Q$, the goal is to compute $\arg\max_{S \in \mathbf{S_{ext}} \wedge S \models Q}(P(S))$, where the $S \models Q$ denotes compatibility between schema and query, i.e., the schema contains the relations and attributes postulated by the query. While $S_{max}$ has the highest probability of all schema candidates in $\mathbf{S_{ext}}$, that does not imply that it has the highest probability with respect to the schema elements mentioned in the query. Thus, we use $S_{max}$ as a generic best guess to enable the user to express queries at first, but then adapt the best schema over time. Note that the personalized schemas we introduce in the next section even allow queries to postulate new relations and attributes.

Detecting extraction errors is harder and typically only possible once issues with the returned query result are discovered. Rather, such errors are most often detected as a result of inconsistencies observed while the analyst explores her data. Thus, the goal of an ASD is to make the process of detecting and repairing extraction errors as seamless as possible. Our approach is based on pay-as-you-go or on-demand approaches to curation [11, 14, 18], and is the focus of Sections 4 and 6 below.

## 3. ADAPTIVE, PERSONALIZED SCHEMAS

An ASD maintains a set of **schema workspaces** $\mathcal{W} = \{W_1, \ldots, W_n\}$. Each workspace $W_i$ has an associated mapped context $\mathcal{C}_i = (S_i, \mathcal{M}_i, P_i)$ where $S_i$ is a schema, $\mathcal{M}_i$ is a set of possible schema matchings [2], each between the elements of $S_i$ and one $S \in \mathbf{S_{ext}}$, and $P_i$ assigns probabilities to these matches. Users may maintain their own personal schema workspace or share workspaces within a group of users that have common interests (e.g., a business analyst workspace for the sales data of a company). In the future, we plan to provide version control style features for the schemas of workspaces including access to data through past schema versions and importing of schema elements from one workspace into another. Recent work on schema evolution [4] and schema versioning demonstrates that it is possible to maintain multiple versions of schemas in parallel where data is only stored according to one of these schemas. Specifically, we plan to extend our own work [15] on flexible versioning of data.

**Importing Schema Elements.** Initially, the schema of a workspace is created empty. When posing a query over an extracted dataset, the user can refer to elements from schema $S_{max}$, schema $S_i$, or new relations and attributes that do not occur in either. References of the first two types are resolved by applying the extraction workflow to compute the instances for these schema elements (and potentially mapping the data based on the matches in $\mathcal{M}_i$). If a query references schema elements from $S_{max}$, then these schema elements are added to the current workspace schema plus one-to-one matches with probability 1 between these elements in $S_{max}$ and $S_i$ are added to the matching $\mathcal{M}_i$.

**Probabilistic Semantics of ASD Queries.** Note that ASD queries are inherently probabilistic, as the result varies depending on the distribution of possible extractions. However, we do not have to overwhelm the user with full probabilistic query semantics. Instead, we apply the approach from [14] to return a deterministic best guess result based on $S_i$ and expose uncertainty through user interface cues and through human-readable on-demand explanations.

EXAMPLE 2. *Continuing with our running example, assume a user operating in workspace $W_1$ would like to retrieve all the names of students based on the enrollment JSON document. One option the user can take is to query the relations exposed by the best guess schema $S_{max}$. For instance, one way to express this query over the schema in Figure 2 is:*

```
SELECT name FROM Undergrad UNION
SELECT name FROM Grad
```

*To process this query, the ASD would run the extraction workflow to create the relational content of the* Undergrad *and* Grad *relations (it would be sufficient to create the projections of these relations on* name *only). The query is then evaluated over these extracted data. As a side-effect, by accessing these schema elements the user declares interest in them and they are added to the schema workspace. Note that only accessed attributes are added to the workspace. If the workspace's schema was empty before, the resulting schema would be $S_1 = \{$*Undergrad(name), Grad(name)*$\}$. Additionally, in $\mathcal{M}_1$ these elements are matched with their counterpart in $S_{max}$. If afterwards the user retrieves the degree of a graduate student then* deg *would be added to* Grad(name).

**Declaring New Schema Elements.** So far we have only discussed the case where a query refers to existing schema elements (either in the user schema or the extracted schema). If a query uses schema elements that are so far unknown, then this is interpreted as a request by the user to add these schema elements to the schema workspace. It is the responsibility of the ASD to determine how schema elements in the extracted schema are related to these new elements. Any existing schema matching (and mapping discovery) approach could be used for this purpose. For instance, we could complement schema matching with schema mapping discovery [3, 17] to establish more expressive relationships between schema elements. Based on such matches we can then rewrite the user's query to access only relations from the extracted schema using query rewriting with views (a common technique from virtual data integration [8]) or materialize its

content using data exchange techniques [7]. Again we take a probabilistic view by storing all possible matches with associated probabilities and choosing the matches with the highest probability for the given query.

EXAMPLE 3. *Assume that a user would like to find names of students without having to figure out which relations in $S_{max}$ store student information. A user may ask:*

```
SELECT name FROM Student
```

*Since relation* Student *occurs in neither $S_1$ nor $S_{max}$, the ASD would run a schema matcher to determine which elements from $S_{max}$ match with* Student *and its attribute* name*, for instance by probabilistically combining the* name *attribute of* Grad *and* Undergrad *as in the query from Example 2.*

In the example above, three Student(name) relations could reasonably be extracted from the dataset: One with just graduate students, one with just undergraduates, and one with both. Although it may be possible to heuristically select one of the available extraction options, it is virtually impossible for a single heuristic to cover all use cases. Instead, ASDs use heuristics only as a starting point for schema definitions. Thus, an ASD decouples its information extraction heuristic from the space of possible extractions that could result from it. In the next section, we present how these uncertain heuristic choices can be validated or corrected as needed in a pay-as-you-go manner [11, 14]. Note that we can use any existing schema matching algorithm for schema matching $\mathcal{M}_i$ as long as it can be modified to expose probabilities. As we have demonstrated in previous work this assumption is reasonable — Mimir [14] already supports a simple probabilistic schema matching operator.

# 4. EXPLANATIONS AND FEEDBACK

Allowing multiple schemas to co-exist simultaneously opens up opportunities for ambiguity to enter into an analyst's interaction with the database. To minimize confusion, it is critical that the analyst be given insight into how the ASD is presently interpreting the data. Our approach to communicating the ASD's decision process leverages our previous work in explaining results of probabilistic queries and data curation operators as developed in Mimir [14] and our previous provenance frameworks for database queries [1, 15]. We discuss the details of these systems along with our proof of concept implementation in Section 6. An ASD must be able to: (1) Warn the analyst when ambiguity could impact her interaction, (2) Explain the ambiguity, (3) Evaluate the magnitude of the ambiguity's potential impact, and (4) Assist the analyst in resolving the ambiguity. In this section, we explore how an ASD can achieve each of these goals in the context of three forms of interaction between the ASD and the outside world: Schema, Data, and Update.

**Schema Interactions.** Schema interactions are those that take place between the analyst and the ASD as she composes queries and explores the relations available in her present workspace. Recall that referencing a relation that does not exist in the workspace and extraction schema $S_{max}$ does not necessarily constitute a problem since this triggers the ASD to add this relation to the workspace and figure out which relations in $\mathbf{S_{ext}}$ it could be matched with. However, it may be the case that no feasible match can be found. This either means that the user is asking for data that is simply not present in the dataset $D$ or that errors in the extraction workflow or matching caused the ASD to miss the correct match. To explain the failure, we may provide the user with a summary of why matching with $\mathbf{S_{ext}}$ failed, e.g., there are no relations with similar names in $\mathbf{S_{ext}}$.

**Data Interactions.** Data interactions happen when the ASD produces query results. Here, ambiguity can typically not be detected by static analysis. For example, the query in Example 3 can have three distinct responses, depending on which relations from the extraction schema are matched against the workspace Student relation. At this level unintrusive interface cues [14] are critical for alerting the analyst to the possibility that the results she is seeing may not be what she had in mind. The Mimir system uses a form of attribute provenance [14] to track the effects of sources of ambiguity on the output of queries. In addition to flagging potentially ambiguous query result cells and rows (e.g., attributes computed based on a schema match that is uncertain), Mimir allows users to explore the effects of ambiguity through both human-readable explanations of their causes and statistical precision measures like standard deviation and confidence. Linking results to sources of ambiguity also makes it easier for the analyst to provide feedback that resolves the ambiguity. In Section 6 we show how we leverage Mimir to streamline data interactions in our prototype ASD.

**Update Interactions.** Finally, update interactions take place when the ASD receives new data or changes to existing data. In comparison to the other two cases, there may not be an analyst directly involved in an update interaction, e.g., a nightly bulk data import. Thus, the ASD must be able to communicate the ambiguity arising from update interactions to analysts indirectly. The main problem with updates is that the extraction schema candidates $\mathbf{S_{ext}}$ may get out of sync with the data it is describing. However, rather than blocking an insertion or update which does not conform with the extraction schema outright, the ASD will represent schema mismatches as missing values when data is accessed through the out of sync schema. Alternatively, we can attempt to resolve data errors with a probabilistic repair. The ASD can also adjust the information extractor to adapt the schema candidate set $\mathcal{C}_{ext}$ and its probability distribution. However, this change could invalidate the matches of an existing workspace. For instance, consider an extracted schema that contains a relation Student(name,credits). If subsequent updates to the dataset insert many students without credits, then eventually the relation Student(name,credits) should be replaced with Student(name). If a workspace schema contains an attribute matched to Student.credits, then this attribute is no longer matched with any attribute from the extraction schema. When explaining missing values to the user, we plan to highlight their cause: 1) data does not match the schema or 2) a workspace attribute is orphaned (it is no longer matched to any attribute in $\mathcal{C}_{ext}$).

**User Feedback.** We envision to let the user provide various kind of feedback about errors in query results such as marking sets of invalid attribute values and unexpected rows. Based on provenance we can then back-propagate this information to interpret these as feedback on matches and extraction workflow decisions. To determine a precise method for determining the best fixes based on such information is an interesting avenue for future work.

# 5. ADAPTIVE ORGANIZATION

The schemas discovered by ASDs can be used for performance tuning by caching relations that are used frequently and are stable (their schema and content are not modified frequently). Furthermore, we can cache the outputs of data extraction or projections over this output to benefit multiple workspace schema elements. This leads to interesting optimization problems because of the sharing of elements. The difference to other automated approaches for tuning physical design is that the sharing of such elements by workspace schemas is encoded in their matchings. It remains to be seen whether this information can be exploited to devise caching strategies that are fine tuned for ASDs. Note that in constrast to relational databases which materialize data according to a schema, this caching is optional for ASDs.

# 6. PROOF OF CONCEPT

We now outline a proof of concept ASD implementation, leveraging the Mimir [14, 18] system for its provenance and feedback harvesting capabilities. For this preliminary implementation, we chose to implement a normalizing relational data extractor for JSON data recently proposed by DiScala and Abadi [5]. This extractor uses heuristics based on functional dependencies (FDs) between the objects' attributes to create a narrow, normalized relational schema for the input data.

## 6.1 Deterministic Extraction

The DiScala extractor [5] runs on collections of JSON objects with a shared schema. The first step in the extraction process is to create a FD graph from the objects. The objects are first flattened, discarding nesting structure and decomposing nested arrays, resulting in a single, very wide and very sparse relation. The extractor creates a FD graph for this relation using a fuzzy FD detection algorithm [9], originally proposed by Ilyas et. al., that keeps it resilient to minor data errors. Any subtree of this graph can serve as a relation, with the root of the tree being the relation's key. At this point, the original, deterministic DiScala extractor heuristically selects one set of relations upfront.

In a second pass, the extractor attempts to establish correlations between the domains of pairs of attributes. Relations with keys drawn from overlapping domains become candidates for being merged together. Once two potentially overlapping relations are discovered, the DiScala extractor uses constraints given by FDs to identify potential mappings between the relation attributes.

## 6.2 Non-Deterministic Extraction

The DiScala extractor makes three heuristic decisions: (1) Which relations to define from the FD graph, (2) Which relations to merge together, and (3) How to combine the schemas of merged relations. Errors in the first of these heuristics appear in the visible schema, and as discussed in Section 2 are easy to detect and resolve. The remaining heuristics can cause data errors that are not visible until the user begins to issue queries. As a result, the primary focus of our proof of concept is on the latter two heuristics.

Concretely, our prototype ASD generalizes the DiScala extractor by providing: (1) Provenance services, allowing users to quickly assess the impact of the extractor's heuristics on query results, (2) Sensitivity analysis, allowing users to evaluate the magnitude of that impact, and (3) Easy feedback, allowing users to easily repair mistakes in the extractor's heuristics. We leverage a modular data curation system called Mimir [14,18] that encodes heuristic decisions through a generalization of C-Tables [10]. A relation in Mimir may include placeholders, or *variables* that stand in for heuristic choices. During normal query evaluation, variables are replaced according to the heuristic. However, the terms themselves allow the use of program analysis as a form of provenance, making it possible to communicate the presence and magnitude of heuristic ambiguity in query results, and also allow users to easily override heuristic choices.

EXAMPLE 4. *Consider a relation $R(A, B)$. To remap $R$ into a new relation $R'(C)$, Mimir uses a query of the form:*

```
SELECT CASE {C} WHEN 'A' THEN A WHEN 'B' THEN B
       ELSE NULL END AS C FROM R
```

*where `{C}` denotes a boolean-valued variable that is true if $A$ maps to $C$ and false otherwise. The resulting C-Table includes a labeled null for each row output that can take the values of `R.A`, `R.B`, or `NULL`, depending on how the model assigns variable values. Consider this example instance:*

| $R$ | $A$ | $B$ |
|---|---|---|
| | 1 | 2 |
| | 3 | 4 |

| $R'$ | $C$ |
|---|---|
| | `{C}`='A' ? 1 : (`{C}`='B' ? 2 : NULL) |
| | `{C}`='A' ? 3 : (`{C}`='B' ? 4 : NULL) |

*Conceptually, every value of $R'.C$ is expressed as deferred computation (a future). A valuation for $\{C\}$ allows $R'$ to be resolved into a classical relation. Conversely, program analysis on the future links each value of $R'.C$, as well as any derived values back to the choice of how to populate $C$.*

Heuristic data transformations, or *lenses*, in Mimir consist of two components. First, the lens defines a view query that serves as a proxy for the transformed data with variables standing in for deferred heuristic choices. Second, a model component abstractly encodes a joint probability distribution for each variable through two operations: (1) Compute the most likely value of the variable, and (2) Draw a random sample from the distribution. Additionally, the model is required to provide a human-readable explanation of the ambiguity that the variable captures.

For this proof of concept we adopt an interaction model where the ASD dynamically synthesizes workspace relations by extending one extracted relation (the primary) with data from extracted relations (the secondaries) containing similar data. Figure 3 illustrates the structure of one such view query: The primary and all possible secondaries are initialized by a projection on the source data. A a single-variable selection predicate (i.e., `{relation?}`) reduces the set of secondaries included in the view. Second, a schema matching projection as illustrated in Example 4 adapts the schema of each secondary to that of the primary.

Our adapted DiScala extractor interfaces with this structure by providing models for relation- and schema-matching, respectively. We refer the reader to [5] for the details of how the extractor computes relation and attribute pairing strength. The best-guess operations for both models use the native DiScala selection heuristics, while samples are generated and weighted according to the pairing strength computed by the extractor. By embedding the DiScala extractor as a lens, we are able to leverage Mimir's program analysis capabilities to provide feedback. When a lens is queried, Mimir highlights result cells and rows that are ambiguous.

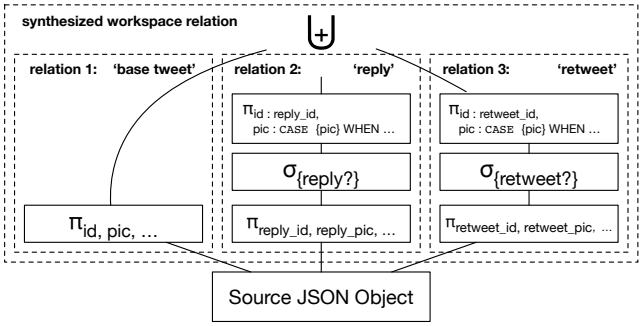



Figure 3: Structure of an extracted relation's merged view

| | Time | | |
|---|---|---|---|
| **Dataset** | **Precompute** | **ASD** | **Classic** |
| **TwitterS** | 223.18s (1.53) | 1.56s (0.19) | 0.77s (0.04) |

Figure 4: Overhead of the provenance-aware extractor (5-trial average, with one standard deviation in parenthesis)

On-request, Mimir constructs human-readable explanations of why they are ambiguous, as well as statistical metrics that capture the magnitude of the potential result error. Crucially, this added functionality requires only lightweight instrumentation and compile-time query rewrites.

## 6.3 Evaluation

Mimir and the prototype ASD are implemented in Scala 2.10.5 being run on the 64-bit Java HotSpot VM v1.8-b132. Mimir was used with SQLite as a backend. Tests were run single-threaded on a 12x2.5 GHz Core Intel Xeon running Ubuntu 16.04.1 LTS. The primary goal of our experiments is to evaluate the overhead of our provenance-instrumented implementation of the DiScala extractor compared to the behavior of the classical extractor. We used a dataset, **TwitterS** consisting of 100 thousand rows and the 100 most common fields taken from Twitter's Firehose. Figure 4 shows the performance of the extractor. We show pre-processing time, the time necessary to compile and evaluate `SELECT * FROM workspace_relation`, and the same query against a table containing the output of the classical DiScala extractor. Note that both these queries return the same set of results, but the former is instrumented, allowing it to provide provenance and feedback capabilities. Runtime for the instrumented extractor is a factor of 2 larger than the original, but is dominated by fixed compile-time costs.

# 7. RELATED WORK

Information extraction and data integration have been studied intensively. Several papers study schema matching [2], entity resolution [6, 16] and mapping XML or JSON data into relational form [5]. Furthermore, there is extensive work on discovering schemas [3, 17] and ontologies [13] from collections of data. We build upon this comprehensive body of work and leverage such techniques in ASDs. With JSON as a simplified semi-structured data model, data can be stored, indexed and queried without upfront schema definition. Liu et al. [12] have introduced the idea that a logical schema can be automatically derived from JSON data and be used to materialize query-friendly adaptive in-memory structures. Curino et al. [4] study multiple schema versions can co-exist virtually in the context of schema evolution. This paper takes these ideas one step further by introducing the notion of adaptive schema databases based on flexible schema data management and establishes a practical framework of probabilistic schema inference with user feedback and provenance tracking.

# 8. CONCLUSION

We present our vision for *adaptive schema databases* (*ASDs*), a technique for querying unstructured and semi-structured data by iteratively inferring and refining personalized schemas that govern access to the data. We discuss how ASDs can be realized using probabilistic query processing techniques and by incorporating extraction and integration into the DBMS. By means of these techniques we can develop systems that enjoy the navigational and organizational benefits of schemas without requiring the high upfront cost of manual schema design, information retrieval, and data integration.

# 9. REFERENCES

[1] B. Arab, D. Gawlick, V. Krishnaswamy, V. Radhakrishnan, and B. Glavic. Reenactment for read-committed snapshot isolation. In *CIKM*, 2016.

[2] P. A. Bernstein, J. Madhavan, and E. Rahm. Generic schema matching, ten years later. *PVLDB*, 4(11):695–701, 2011.

[3] M. J. Cafarella, D. Suciu, and O. Etzioni. Navigating extracted data with schema discovery. In *WebDB*, 2007.

[4] C. Curino, H. J. Moon, A. Deutsch, and C. Zaniolo. Automating the database schema evolution process. *VLDB Journal*, 22(1):73–98, 2013.

[5] M. DiScala and D. J. Abadi. Automatic generation of normalized relational schemas from nested key-value data. In *SIGMOD*, 2016.

[6] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *TKDE*, 19(1):1–16, 2007.

[7] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *TCS*, 336(1):89–124, 2005.

[8] A. Y. Halevy. Answering queries using views: A survey. *VLDBJ*, 10(4):270–294, 2001.

[9] I. F. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga. Cords: Automatic discovery of correlations and soft functional dependencies. In *SIGMOD*, 2004.

[10] T. Imieliński and W. Lipski, Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, Sept. 1984.

[11] S. R. Jeffery, M. J. Franklin, and A. Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In *SIGMOD*, 2008.

[12] Z. H. Liu, B. Hammerschmidt, D. McMahon, Y. Liu, and H. J. Chang. Closing the functional and performance gap between SQL and NoSQL. In *ICMD*, 2016.

[13] A. Maedche and S. Staab. Learning ontologies for the semantic web. In *ICSW*, 2001.

[14] A. Nandi, Y. Yang, O. Kennedy, B. Glavic, R. Fehling, Z. H. Liu, and D. Gawlick. Mimir: Bringing ctables into practice. Technical report, The ArXiv, 2016.

[15] X. Niu, B. Arab, D. Gawlick, Z. H. Liu, V. Krishnaswamy, O. Kennedy, and B. Glavic. Provenance-aware versioned dataworkspaces. In *TaPP*, 2016.

[16] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, 2012.

[17] K. Wang and H. Liu. Schema discovery for semistructured data. In *KDD*, volume 97, pages 271–274, 1997.

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

[19] Y. Yang, N. Meneghetti, R. Fehling, Z. H. Liu, and O. Kennedy. Lenses: An on-demand approach to etl. *VLDB*, 8(12):1578–1589, 2015.