

Multi-Table Error Correction: Opportunities and Limitations

Fatemeh Ahmadi
BIFOLD & TU Berlin
Berlin, Germany
f.ahmadi@tu-berlin.de

Luca Zecchini
BIFOLD & TU Berlin
Berlin, Germany
luca.zecchini@tu-berlin.de

Ziawasch Abedjan
BIFOLD & TU Berlin
Berlin, Germany
abedjan@tu-berlin.de

ABSTRACT

Existing error correction systems clean tables in isolation, requiring separate rules or labels for each table. For large sets of tables, this approach is costly and does not scale. At the same time, tables in a collection may be related, creating both opportunities and risks for sharing correction signals. We present a systematic study of multi-table error correction, characterizing table relationships along two structural dimensions: schema- and record alignment. We also introduce two multi-table correction strategies: post-merge correction, which combines related tables before cleaning, and cross-table correction, which shares cell-level signals across tables. Both are evaluated against isolated correction across different scenarios, error types, and data lakes. The results show that the effectiveness depends strongly on table relationships and error types. Joining with clean keys and induced dependencies can yield gains, as well as unions with overlapping records. We compile these findings into guidelines for choosing when and how to use multi-table correction.

PVLDB Reference Format:
Fatemeh Ahmadi, Luca Zecchini, and Ziawasch Abedjan. Multi-Table Error Correction: Opportunities and Limitations. PVLDB, 20(2): 243 - 255, 2026.
doi:10.14778/3849423.3849437

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/D2IP-TUB/multi-table-error-correction.

1 INTRODUCTION

For decades, the community has studied the problem of data cleaning. A large number of systems for error detection and correction were therefore introduced, each tailored to different requirements [6, 7, 9, 24, 25, 29, 32, 35–38]. These systems apply various heuristics and signals, such as rules or user annotations, to detect and correct erroneous values. Most existing approaches consider cleaning as a procedure performed on a single dirty table, whose output is the corresponding clean or annotated table.

In practice, production tables rarely exist as standalone objects: they are either part of a larger schema, as in a data warehouse [4], or reside in large, heterogeneous data lakes [18]. Such collections often consist of thousands or even millions of tables with highly heterogeneous schemata [2]. Their records describe various types of entities, some overlapping across tables, others entirely distinct. With existing methods, a collection of tables would be cleaned by considering each table sequentially in isolation. This requires substantial resources: every table needs labeled training data or predefined constraints, and both types of signals are expensive to obtain even on a single table. Thus, when dealing with large collections of tables, it becomes fundamental to assess whether it is possible to make the cleaning process more efficient through either shared annotations or computational optimizations.

Our recent system, Matelda [2], introduced a multi-table approach to error detection. Matelda establishes different table folding techniques to group potentially similarly dirty cells together. This allows sharing labels across tables for semi-supervised error detection, reducing the amount of labels required compared to single-table counterparts. However, applying the same intuition for correction is not trivial. While detection implies a binary decision (*is this value wrong?*), correction requires choosing the right replacement for a wrong value from countless possible options. This degree of freedom makes it non-trivial to propagate corrections identified in a table to another table. Whether multi-table cleaning helps depends on the types and positions of the errors and the similarity of the entities in the collection.

In this work, we systematically study the challenges of multi-table error correction beyond sequential, isolated cleaning. We point out when connecting tables can reduce resource requirements, addressing the following research questions:

- **RQ1:** *Which scenarios benefit from table merging operations or algorithms that propagate cleaning signals across tables?*
- **RQ2:** *How can structural features of a table collection be used to guide error correction strategies in multi-table settings?*

To answer **RQ1**, we characterize table relationships along two structural dimensions: *schema alignment*, i.e., how attributes map across tables, and *record alignment*, i.e., whether their records overlap. We detect several scenarios that branch from settings with *(1) unrelated tables*, *(2) joinable tables*, and *(3) unionable tables*. These scenarios may co-exist within real-world data lakes, where relationships among tables are often not known in advance [31]. Across scenarios, we compare the effectiveness of merging related tables before cleaning with cleaning performed without merging.

To answer **RQ2**, we examine how error correction systems can be applied to multi-table settings. We identify two strategies: *(1) isolated correction*, where each table is processed independently, and *(2) post-merge correction*, where tables are combined through unions or joins and then corrected as a single table. Further, we introduce a third strategy, *(3) cross-table correction*, which performs correction at the collection level rather than on single tables. It assigns correction techniques based on the latent error type of each table cell in the entire lake. We study two variants of the approach, one where manually-defined heuristics guide the grouping of the cells and one where the grouping is carried out based on latent features. In short, we make the following contributions:

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 20, No. 2 ISSN 2150-8097.
doi:10.14778/3849423.3849437

- We introduce a categorization for table relationships based on schema- and record alignment, which enables an understanding of the limits and potential of cleaning in multi-table scenarios.
- We design and implement multi-table correction pipelines and analyze their benefits and limitations over isolated correction across scenarios, correction strategies, and error types.
- We summarize our findings into practical guidelines for applying error correction in multi-table settings.

Observations. The success of multi-table error correction through *post-merge* or *cross-table* strategies depends on two factors: whether sharing information across tables exposes usable correction evidence, and whether the underlying correction system can exploit this evidence at acceptable computational cost.

Post-merge correction leads to significant improvements for *joinable tables* when merging introduces reliable dependencies or correlations between attributes that help resolve rule violations. Benefits are also observable for *unionable tables* with overlapping entities, when clean occurrences in one table can help repair errors in another. In real-world lakes, however, joins and unions must first be inferred. As a result, the effectiveness of post-merge correction depends not only on the repair method, but also on the quality of relationship discovery. In our experiments, ad-hoc merging improves over isolated correction when the discovered relationships provide useful evidence without substantial noise or input growth. More aggressive integration is not necessarily better. None of the analyzed strategies, including isolated correction, currently scales to lakes with millions of tables.

Cross-table correction lags behind isolated and post-merge correction in almost all scenarios. Its main advantage is the reduced labeling effort, made possible by the sharing of signals across tables. However, this generalization makes it less effective for semantic and hard-to-correct errors.

2 PRELIMINARIES AND RELATED WORK

We first formalize the problem of error correction and clarify the scope of our analysis, then discuss related work.

2.1 Problem Statement

Data cleaning comprises two major tasks: error detection and error correction. In this paper, we focus on the latter. Since correcting errors requires them to be detected in advance, we assume error detection has already been performed and a set of detected errors is given. Thus, we define *multi-table error correction* as follows.

Let $\mathcal{T} = \{t_1, t_2, \ldots, t_{|\mathcal{T}|}\}$ denote a collection of tables, each containing cells that have been identified as erroneous. For each table $t_k \in \mathcal{T}$, let t_k^* denote its corresponding clean version, and let the full clean collection be $\mathcal{T}^* = \{t_1^*, t_2^*, \ldots, t_{|\mathcal{T}|}^*\}$. We define the set of erroneous cells in the collection as $E = \{\, t_k[i,j] \mid t_k \in \mathcal{T},\ t_k[i,j] \neq t_k^*[i,j] \,\}$, where i and j denote row and column indices. The goal of error correction is to replace all cells in E so that their values match those in the corresponding clean tables in $\mathcal{T}^*$.

Data Errors. Data errors are commonly categorized as either *syntactic*, i.e., values that violate the expected representation or format of a column (e.g., typos), or *semantic*, i.e., values that may be syntactically valid in isolation but are incorrect in the context of the record or table (e.g., a city name inconsistent with the corresponding country) [25]. A cell may exhibit both types. In this work, we focus on error types widely considered in prior cleaning benchmarks [1, 5, 25, 33], observed in common cleaning datasets [16, 19, 35], and can be generated in a controlled manner [3]. Specifically, we consider missing values, typos, formatting issues, numeric outliers, and functional dependency (FD) violations. While the former error types mainly capture syntactic errors, FD violations model semantic errors arising from inconsistent relationships between attributes. This selection allows us to cover both major error categories while remaining compatible with existing error generators and cleaning systems.

2.2 Related Work

Our research relates to error correction, data lakes as a resource for cleaning, as well as data discovery and table merging operations.

2.2.1 Single-Table Error Correction. Existing error correction techniques for *isolated tables* can be categorized into rule-based, learning-based, and LLM-driven approaches.

Rule-based methods. Systems such as Holistic [6], Horizon [36], and UniClean [9] rely on integrity constraints like FDs or denial constraints to repair inconsistencies. They are effective when high-quality constraints are available. In heterogeneous collections, constraints are often not available or table-specific, limiting the applicability of purely rule-based correction.

Learning-based methods. Systems such as HoloClean [35] and Baran [24] frame error correction as a probabilistic inference problem or a classification task. HoloClean integrates constraints and statistical signals into a unified probabilistic model. Baran casts correction as a candidate generation and classification task, learning column-specific models from labeled examples and contextual signals. In principle, learning-based approaches are well suited for sharing supervision across related cells. In current systems, however, prediction models and features are typically schema-dependent.

LLM-driven methods. Recent systems such as ZeroEC [38] leverage LLMs to infer implicit constraints or directly propose repairs. These approaches reduce the need for handcrafted rules or curated training data. For example, ZeroEC uses LLMs to extract correction rules, such as (conditional) FDs, from the input table. As these rules are derived for a specific table schema and value distribution, they are table-specific and not readily reusable across other tables. As a result, applying such methods to a collection translates into cleaning tables individually. Further, they introduce challenges related to privacy, explainability, cost, and reproducibility [27].

2.2.2 Cleaning using Data Lakes. Another line of work addresses cleaning using data lakes. LakeFill [39] adopts a retrieval-augmented generation (RAG) framework, retrieving evidence and applying LLM reasoning to infer missing values in an incomplete tuple. CESID [23] aims to impute missing values in a given table using information retrieved or derived from the lake. It uses a KNN classifier to choose between search-based and estimation-based imputation methods. Neither method is designed for joint correction across all tables in a data lake.

Table 1: The influence of table relationships on error correction.

Schema	Records	Operation	Error position	Merging helps if...	Merging hurts if...
Fully aligned	**Overlapping**	Union	Any	*Correct value support:* ↑ *Applicable FD:* ↑	*Wrong candidate support:* ↑
	Disjoint	Union	Any	*Correct value support:* ↑ *Applicable FD:* ↑	*Wrong candidate support:* ↑ *Misleading FD:* ↑ *Applicable FD:* ↓
Partially aligned	**Overlapping**	Join	Join key	No direct benefit under exact matching	Valid match lost or false match created Tuple expansion introduces noise (non-unique key)
			Non-key attribute	*Correct value support:* ↑ *Applicable cross-table FD or correlation:* ↑	*Wrong candidate support:* ↑ *Misleading FD or correlation:* ↑ Tuple expansion introduces noise (non-unique key)
	Overlapping/ Disjoint	Partial union	Inner-union attribute	*Correct value support:* ↑ *Applicable FD:* ↑	*Wrong candidate support:* ↑ *Misleading FD:* ↑ *Applicable FD:* ↓
			Outer-union attribute	No direct evidence for the erroneous attribute	Null growth and diluted distributions
Unrelated	**Any**	-	Any	Only fine-grained signal sharing	Spurious alternatives, dependencies, or similarities

Notation: ↑ new/strengthened evidence; ↓ weakened evidence or invalidated FD. *Example:* For fully aligned tables, union may help correct an error if support for the correct value in the aligned attribute increases.

2.2.3 Data Lakes & Dataset Discovery. Dataset discovery techniques [12, 31] aim to identify table relationships in large-scale data lakes, such as joins [10, 11, 41] and unions [14, 21, 28]. GEN-T [13] performs *table reclamation*: given a target table together with its key, it tries to reproduce that table using a set of tables retrieved from the lake. ALITE [20] integrates a set of discovered tables via full disjunction. It has recently been extended to fuzzy matches over inconsistent values such as synonyms and abbreviations [22]. We compare our proposed merging strategy with ALITE to analyze how table relationships impact multi-table error correction.

3 TABLE RELATIONSHIPS IN DATA LAKES

By categorizing table relationships, we reason about scenarios in which sharing correction signals can help the cleaning process.

3.1 Structural Dimensions

We characterize relationships between tables along two dimensions: *schema alignment* and *record alignment*.

Schema Alignment. Let $\mathcal{S}(t)$ denote the set of attributes of table t. For any two tables $t_i, t_j \in \mathcal{T}$, we distinguish three cases. *(1) Fully aligned schemata:* $\mathcal{S}(t_i) = \mathcal{S}(t_j)$. The tables consist of the same set of attributes with known correspondences and can be directly unioned. *(2) Partially aligned schemata:* $\mathcal{S}(t_i) \cap \mathcal{S}(t_j) \neq \emptyset$. The tables overlap in some attributes but differ in others. This partial overlap enables partial unions or relational joins on common attributes. *(3) Unrelated schemata:* $\mathcal{S}(t_i) \cap \mathcal{S}(t_j) = \emptyset$. Tables have no common or mappable attributes and cannot be unioned or joined.

Record Alignment. Two tables have *overlapping records* only if at least one record from each table refers to the same real-world entity. Otherwise, they are *disjoint*.

3.2 Scenarios

Table 1 combines the two aforementioned dimensions and summarizes how the determined scenarios can influence error correction. We further illustrate these scenarios through the running example in Figure 1, which consists of six Hogwarts tables with erroneous values (red) and the corresponding ground-truth values (green).

3.2.1 Unrelated Tables. In this scenario, tables have unrelated schemata and the sets of records are disjoint. Spurious overlaps of values can still occur. In Figure 1, Tables A (quidditch team members) and F (exam metadata) illustrate this case.

Opportunities. Unrelated tables offer minimal opportunities for cross-table correction. In some rare cases, identical errors such as typos might appear in different semantic contexts. For example, "Gryffindor**e**" may appear both as a house name and as part of "Gryffindor**e** Common Room". In such cases, sharing fine-grained signals across tables may still support correction.
Challenges. Coincidental similarities may introduce spurious repair candidates or misleading signals.

3.2.2 Unionable Tables. In this scenario, tables have fully aligned schemata, e.g., Tables A and B, and can be unioned.
Opportunities. For errors in aligned attributes, a union can strengthen correction evidence by increasing support for correct alternatives in the same column. Overlapping records referring to the same entity provide direct value evidence, whereas disjoint records only reshape distributions and dependencies, giving weaker, indirect evidence. In our example, "Ron Weasley" in Table B provides a clean alternative for "R. Weasley" in Table A.
Challenges. Unioning is risky when aligned columns differ in patterns, value distributions, or dependency structures. For example, the *Date* columns in Tables A and B use different formats and distributions, which may mislead pattern- or frequency-based methods. Unioning may also invalidate existing dependencies: *House* → *Role* holds in Table B but not in the unioned table.

3.2.3 Partially Unionable Tables. In this scenario, tables have partially aligned schemata and can be partially unioned over shared attributes, e.g., Tables A and C on *Student* and *House*. Such a union introduces missing values for unshared attributes across the tables.
Opportunities and Challenges. For errors in inner-union (shared) attributes, a partial union has the same benefits and risks as a union of fully aligned tables (Section 3.2.2). For outer-union (unshared) attributes, the union provides no direct value evidence, while the introduced null values may weaken source-specific dependencies.

3.2.4 Joinable Tables. In this scenario, tables share one or more attributes so that relational joins are possible. For instance, Tables A and C can be joined on the attribute *Student*.
Opportunities. Joining tables can provide additional context for correction. Consider the missing *House* value for Ginny Weasley in Table A. In isolation, Table A offers limited correction signals. However, if we join Table A with Table C on *Student*, the FD *Advisor* → *House* becomes available. Ginny's advisor is McGonagall,

A) QuidditchTeam (Half-Blood Prince)

Student	House	Role	Date
Harry Potter	Gryffindor	Captain	96.09.21
Ginny Weasley	NULL (Gryff.)	Chaser	1996
R. Weasley (Ron Weasley)	Gryffindor	Keeper	1996
Demelza Robins	Gryffindor	Chaser	1996
Draco Malfoy	Slytherin	Seeker	1996
Urquhart	Slytherin	Captain	1996
Vaisey	Hufflep.(Slyth.)	Chaser	1996
Bole	Slytherin	Beeter(Beater)	1996

B) DuelingClub (Chamber of Secrets)

Student	House	Role	Date
Draco Malfoy	Slytherin	Particip.	1992-10-12
Fergus Cowley	Slytherin	Particip.	1992-10-12
Millicent B.	Slytherin	Particip.	1992-10-12
Bridget Wenlock	Slytherin	Particip.	1992.10.12
Ron Weasley	Gryffindor	Particip. (Spect.)	1992-10-12
Ginny Weasley	Gryffindor	Spect.	1992-10-12
Vaisey	Hufflep. (Slyth.)	Particip.	1992-10-12
Justin F.-Fletch.	Hufflepuff	Particip.	1992-10-12

C) StudentHouseInfo (Hogwarts Registry)

Student	House	Blood	Advisor
Harry Potter	Gryffindor	Half	McGonagall
H. Granger	Gryffindor	Muggle	McGonagall
Draco Malfoy	Slytherin	Pure	Snape
Ginny Weasley	∅ (Gryff.)	Pure	McGonagall
N. Longbottom	Gryffindor	Pure	McGonagall
Luna Lovegood	Ravenclaw	Pure	Flitwick
J. Finch-Flet. (J. F.-Fletch.)	Hufflepuff	Muggle	Sprout
S. Finnegan (S. Finnigan)	Gryffindor	Half	McGonagall

D) QuidditchTeam

Student	House	Role	Year
Vaisey (Rogers)	Ravenclaw	Chaser	1996
Flint (Randall)	Ravenclaw	Keeper	1996
Ginny Weasley	Gryffindor	Chaser	1996
Ron Weasley	Gryffindor	Keeper	1996

E) PotionsClub (Slug Club – Advanced)

Student	Grade	Time
Hermione Granger	93	45
Lucian Bole	98	50
D. Malfoi (Draco Malfoy)	82	60.00 (60)
Cormac McLaggen	70	80

F) Exams

Exam	Max-Score
Potions	100 (98)
Defense Against Dark Arts	100
Transfiguration	100
Charms	100

Figure 1: Six Hogwarts tables with cell-level errors (red) and the corresponding ground truth values (green).

who advises Gryffindor students; this cross-table signal therefore favors *Gryffindor* over competing values.

Challenges. Joins can introduce misleading evidence when the join attribute is non-unique or erroneous. A non-unique key creates many-to-many matches, replicating tuples and errors, shifting value distributions, and introducing irrelevant cross-table dependencies. For example, joining Tables C and D only on *House* would pair every Gryffindor student with every Gryffindor player, making attributes such as *Role* and *Year* possibly misleading for correction. Errors in the join attribute may instead prevent valid matches or create false ones. For example, "J. Finch-Flet." in Table C may not match "Justin F.-Fletch." in Table B when joining on *Student*.

Overall, indicators such as schema- and record alignment, error position, join-key uniqueness, and tuple expansion are measurable before correction and can flag risky merges upfront.

3.3 Real-World Setup: Unknown Relationships

Real-world table collections may contain an unknown and heterogeneous mixture of unionable, joinable, and unrelated tables, as illustrated by the complete Hogwarts dataset (Tables A–F). The main challenge is to first identify all relevant relationships and judiciously decide which tables to merge through joins or unions and what signals to share across which tables. Due to the possibly fuzzy nature of relationships that exacerbates with the number of tables, one would need an approach that automatically organizes tables or table fragments into folds to share cleaning signals for higher accuracy or lower cleaning cost.

4 MULTI-TABLE CORRECTION STRATEGIES

We identify three correction strategies that differ in how they share information across tables during correction.

1) Isolated Correction. In isolated correction, each table $t_k \in \mathcal{T}$ is cleaned in isolation. Its advantages are simplicity and reliance on local context. No cross-table matching, alignment, or shared resources are required. Also, the correction is not influenced by potential noise from poor inter-table relationships.

2) Post-merge Correction. When tables share schema elements, they can be combined into a single relation through joins or unions before correction. This strategy provides the cleaning system with richer contextual evidence, allowing learned rules or models to generalize across tables, *without* modifying the logic of the system. However, this may also introduce noise into the correction process. We describe how related tables are merged in Section 4.1.

3) Cross-table Correction. This strategy considers the entire table collection at once by sharing fine-grained signals across tables without merging them upfront. To abstract from table-specific signals, we formulate the correction task as a semi-supervised cell-level classification problem. We introduce two variants of this strategy, *rule-based* and *data-driven*, detailed in Section 4.2.

4.1 Post-merge Correction

Table relationships within a table collection are often not explicitly known. We therefore investigate ad-hoc merge techniques as a pre-processing step for error correction and propose a simple and intuitive process. Given a collection $\mathcal{T}$ with erroneous cells E, Algorithm 1 produces a novel merged collection $\mathcal{T}_m$ by unioning and joining related table pairs. Every table $t \in \mathcal{T}$ appears in $\mathcal{T}_m$ exactly once, in isolation or merged with another table. Algorithm 1 has two steps: discovering related tables in $\mathcal{T}$ (Lines 1-4) and constructing $\mathcal{T}_m$ from the most promising candidates (Lines 5-20).

In the discovery step, we identify all unionable and joinable pairs of tables in $\mathcal{T}$. In join discovery (Line 3), we search for potential PK/FK relationships. For every column c of the query table t with unique, non-null values in its clean domain $dom(c)$, hence a possible primary key, we retrieve all columns in $\mathcal{T} \setminus \{t\}$ whose clean domain shares values with $dom(c)$. For each candidate, we compute a joinability score as the fraction of matching tuples among all tuples in the corresponding full outer join [40], without materializing the join. For each possible primary key c, we retain the top-k_j candidates whose score is at least θ_j.

In union discovery (Line 4), we use clean-domain overlap to determine column unionability [12]. Fuzzy unioning [14] would interfere with the goals of data cleaning, which accounts for value variations and aims at reducing semantic heterogeneity. For every column of the query table t, we retrieve the columns in $\mathcal{T} \setminus \{t\}$ with overlapping clean domain and score them by their clean-domain overlap normalized by the total number of cells in both columns, pruning scores below θ_u. We aggregate these pairs at the table level using the Hungarian method, requiring the mapping to cover at least half of each schema. The global unionability score is the sum of the mapped-column scores. We retain the top-k_u candidates per query table. Our implementation relies on the BLEND data discovery

Algorithm 1: Greedy Table Merging

Input: Collection $\mathcal{T}$, erroneous cells E, numbers of top candidates k_u and k_j and thresholds θ_u and θ_j for union and join discovery
Output: Collection $\mathcal{T}_m$ built by unioning and joining related table pairs in $\mathcal{T}$

1 $\mathcal{U} \leftarrow \emptyset, \mathcal{J} \leftarrow \emptyset$ // union and join candidates
2 **for** $t \in \mathcal{T}$ **do**
3 $\quad \mathcal{J} \leftarrow \mathcal{J} \cup$ getJoinableTables$(t, \mathcal{T} \setminus \{t\}, k_j, \theta_j)$
4 $\quad \mathcal{U} \leftarrow \mathcal{U} \cup$ getUnionableTables$(t, \mathcal{T} \setminus \{t\}, k_u, \theta_u)$
5 $\mathcal{T}_m \leftarrow \emptyset$
6 **while** $\mathcal{T} \neq \emptyset$ **do**
7 $\quad$ **if** $\mathcal{U} \neq \emptyset$ **then**
8 $\quad\quad (t_1, t_2) \leftarrow$ getTopCandidate$(\mathcal{U})$
9 $\quad\quad t_u \leftarrow$ materializeUnion(t_1, t_2)
10 $\quad\quad \mathcal{T}_m \leftarrow \mathcal{T}_m \cup \{t_u\}, \mathcal{T} \leftarrow \mathcal{T} \setminus \{t_1, t_2\}$
11 $\quad\quad \mathcal{U} \leftarrow \mathcal{U} \setminus$ (getCandidates$(\mathcal{U}, t_1) \cup$ getCandidates$(\mathcal{U}, t_2)$)
12 $\quad\quad \mathcal{J} \leftarrow \mathcal{J} \setminus$ (getCandidates$(\mathcal{J}, t_1) \cup$ getCandidates$(\mathcal{J}, t_2)$)
13 $\quad$ **else if** $\mathcal{J} \neq \emptyset$ **then**
14 $\quad\quad (t_1, t_2) \leftarrow$ getTopCandidate$(\mathcal{J})$
15 $\quad\quad t_j \leftarrow$ materializeJoin(t_1, t_2)
16 $\quad\quad \mathcal{T}_m \leftarrow \mathcal{T}_m \cup \{t_j\}, \mathcal{T} \leftarrow \mathcal{T} \setminus \{t_1, t_2\}$
17 $\quad\quad \mathcal{J} \leftarrow \mathcal{J} \setminus$ (getCandidates$(\mathcal{J}, t_1) \cup$ getCandidates$(\mathcal{J}, t_2)$)
18 $\quad$ **else**
19 $\quad\quad$ **for** $t \in \mathcal{T}$ **do**
20 $\quad\quad\quad \mathcal{T}_m \leftarrow \mathcal{T}_m \cup \{t\}, \mathcal{T} \leftarrow \mathcal{T} \setminus \{t\}$
21 return $\mathcal{T}_m$

system for both searches [12]. We set θ_j and θ_u to 0.5 as default values. The impact of these parameters is analyzed in Section 5.9.2.

We iteratively materialize the most promising candidates until every table in $\mathcal{T}$ is included in $\mathcal{T}_m$, either isolated or merged (Lines 5-20). We prioritize unions over joins because most correction systems exploit intra-column redundancy through frequency and co-occurrence signals, which unions increase. Moreover, unions are safer to select greedily, as they do not multiply tuples. We analyze the impact of this ordering in Section 5.9.5. If union candidates are present (Lines 7-12), we therefore materialize the top one and include the unioned table t_u into $\mathcal{T}_m$, removing the original tables t_1 and t_2 from $\mathcal{T}$ and filtering out their further union and join candidates. We then proceed with join candidates (Lines 13-17), materializing the top candidate and including it in $\mathcal{T}_m$. Finally, we add the remaining unmerged tables to $\mathcal{T}_m$.

Quantities after Table Merging. Joins can introduce additional tuples when the relationship is not one-to-one. Consequently, a single erroneous cell may appear multiple times in the materialized join, even though our discovery procedure prioritizes candidate primary keys. This replication does not introduce new erroneous cells into the collection. The merged tables are only temporary structures used to expose additional correction signals, and the final corrections are always projected back to the corresponding cells in the original source tables. We consolidate corrections proposed for duplicated instances via majority voting.

Union operations require a related design choice. In RDBMSs, unions can follow either *set* or *multiset* semantics [15]. For the set-based variant, duplicate *clean* tuples only appear once in the merged tables. We evaluate both variants in Section 5.9.1.

Computational Complexity. We analyze the computational complexity of the greedy merging strategy (Algorithm 1). The algorithm has four computationally independent steps: Blend indexing, relationship discovery, column alignment, and materialization.

1) Indexing: Let $m = \max_{t \in \mathcal{T}} |\mathcal{S}(t)|$ be the maximum number of columns in a table. Let r denote the maximum number of rows in an input table. The total number of cells in the lake is $|\mathcal{T}| \cdot m \cdot r$. Processing every cell and recording metadata for every column requires $O(|\mathcal{T}| \cdot m \cdot r + |\mathcal{T}| \cdot m)$.

2) Discovery: For each table $t \in \mathcal{T}$, we retrieve its top-k_j join and top-k_u union candidates. This requires comparing each column's domain ($dom(c)$) with the domains of all other columns in the corpus. In the worst case, this requires one lookup per cell for both joins and unions, yielding $O(2 \cdot |\mathcal{T}| \cdot m \cdot r \cdot O(\text{look-up}))$. Look-up complexity depends on Blend's database index and is typically constant, resulting in $O(2 \cdot |\mathcal{T}| \cdot m \cdot r)$.

3) Column alignment: For each union candidate pair, we compute a maximum-weight bipartite matching in $O(m^3)$ time. In the worst case, each source table has $|\mathcal{T}| - 1$ candidate target tables. Therefore, across the entire lake, the worst-case complexity of column alignment is $O(|\mathcal{T}| \cdot (|\mathcal{T}| - 1) \cdot m^3)$, or equivalently $O(|\mathcal{T}|^2 \cdot m^3)$.

4) Materialization: The greedy strategy iteratively materializes the top candidate table pair, which can be either a union or a join candidate, materializing at most once for each table t. Materializing one union costs $O(m \cdot r)$, while materializing one join costs $O(m \cdot r^2)$. Thus, in the worst case, each table has to be joined with another table, yielding $O(|\mathcal{T}| \cdot m \cdot r^2)$.

Aggregating all the previous steps, we obtain a total asymptotic complexity of $O(|\mathcal{T}|^2 \cdot m^3 + |\mathcal{T}| \cdot m \cdot r^2)$.

4.2 Cross-table Correction

In cross-table correction, tables are cleaned by sharing correction signals between related cells across all tables without fully aligning entire schema elements as in post-merge correction. To relate cells of different tables, we need schema-independent heuristics. Rule-based approaches [9, 36] directly disqualify for adoption. Supervised approaches also rely on schema-dependent, table-specific features extracted from individual datasets.

Inspired by Baran [24], we develop a model that, given an erroneous value and a suggestion predicts whether the suggestion fits or not. We use all schema-independent features from Baran, adapt schema-dependent ones, and add two entirely new features.

Given a table collection $\mathcal{T}$ and its set of erroneous cells E, let V_e denote the set of candidate repairs for a cell $e \in E$. For each pair (e, v), with $v \in V_e$, we aim to determine whether v is the correct repair for e. Following the modeling of Baran [24], we represent (e, v) as a feature vector consisting of several correction signals.

Value- and Domain-based Features. We adopt the eight value-based models and the domain-based model from Baran [24]. The value-based models generate candidates through diff checking, combining two encodings of the erroneous value with four correction operators: substring insertion, removal, replacement, or full-value swapping. The domain-based model scores candidates by their estimated probability among clean values in the column, favoring more frequent values.

Vicinity-based Features. To exploit column relationships within the same tuple, Baran [24] trains a separate dependency model for

	Inconsistent $\pi(e)$	Consistent $\pi(e)$
Unique $\kappa(j)$	Tab. E: `60.00` → `60`	Tab. D: `Vaisey` → `Rogers`
Non-unique $\kappa(j)$	Tab. A: `Hufflep.` → `Slytherin`	Tab. B: `Particip.` → `Spect.`

Figure 2: Four zones determined by rule-driven zoning.

every ordered pair of distinct columns. This modeling is schema-dependent. Inspired by Matelda [2], we obtain a schema-agnostic representation by aggregating the scores of all dependency models that are valid for a cell into four schema-independent features. For each column, we estimate co-occurrence probabilities between v and the co-occurring column value in clean rows. These per-column scores are aggregated into four features, each weighted by the corresponding FD confidence estimated on clean rows: *Average*, the mean over all columns, and three features using as determinant the *First column*, which often serves as key or identifier, the *Left neighbor*, and the *Right neighbor* of the erroneous cell's column.

Syntactic Features. We introduce two schema-independent syntactic features. The first measures the normalized edit similarity between the erroneous value e and the candidate v, capturing corrections that require only small syntactic changes. The second captures column-level patterns: using Microsoft PROSE [17], we learn transformations from clean values and labeled corrections and select the pattern with the highest support. We use this support as a feature indicating the reliability of pattern-based corrections.

4.2.1 Zoning Strategies. Rather than training one classifier per column, we group erroneous cells into *zones* whose members are expected to share correction logic across tables and columns. Each zone uses a single classifier over the unified feature space from Section 4.2, trained on pooled labels from its cells. We explore a *rule-driven* zoning strategy that assigns cells to predefined zones using interpretable criteria, and a *data-driven* one that discovers zones automatically from the data.

Rule-driven Zoning (RCC). We define zones using two cell characteristics: column uniqueness and pattern consistency. We distinguish between cells in unique columns, where all values differ, and cells in non-unique columns, where repeated values allow corrections to exploit redundancy and cardinality. Further, we assess whether the erroneous value e conforms to the syntactic pattern of the clean domain D_j^c of its column j. Pattern consistency $\pi(e)$ denotes the pattern of e as *consistent* if $e \in D_j^c$ or e conforms to the dominant pattern p_j for column j, and *inconsistent* otherwise. An erroneous cell with a consistent pattern contains a value that is syntactically valid but semantically incorrect, such as a well-formed date that is factually wrong, whereas an inconsistent-pattern one contains a value that violates the expected format entirely.

The combination of column uniqueness and pattern consistency partitions erroneous cells into four zones, as depicted in Figure 2. For unique columns, where the correct value is unlikely to occur in the clean domain, we restrict candidate generation to value-based models to avoid noisy domain- and vicinity-based candidates. For pattern-inconsistent cells, we first enforce the dominant column pattern using PROSE; cells that remain erroneous are then handled by model-based candidate generation and classification.

Table 2: Cell characterization vector $\mathbf{x}(e) \in [0,1]^{10}$.

Feature	Definition	Feature	Definition
Syntactic Deviation		*Semantic Deviation*	
x_1	$1 - P_{\text{mask}}(\text{mask}(e) \mid j)$	x_6	$1 - P_{\text{val}}(e \mid j)$
x_2	$\min\left(\frac{\lvert\lvert e\rvert - \mu_j^{\text{len}}\rvert}{\sigma_j^{\text{len}}} \frac{1}{3}, 1\right)$	x_7	$\min_{v' \in D_j^c} d_{\cos}(\mathbf{emb}(e), \mathbf{emb}(v'))$
x_3	$\mathbb{1}[\text{type}(e) \neq \text{type}_j]$	x_8	$1 - \frac{\lvert\text{bigrams}(e) \cap G_j\rvert}{\lvert\text{bigrams}(e)\rvert}$
x_4	$\mathbb{1}[e \notin [\min_j, \max_j]]$	x_9	$\min_{v' \in D_j^c} \frac{\text{Lev}(e,v')}{\max(\lvert e\rvert, \lvert v'\rvert)}$
x_5	$\mathbb{1}[\text{format}(e) \notin \mathcal{F}_j]$	*Contextual Deviation*	
		x_{10}	$\max_{j' \to j} \text{FD_conf}(j', j) \mathbb{1}[e \neq e^*_{j' \to j}]$

Data-driven Zoning (DCC). Rule-driven zoning is interpretable but limited to a fixed two-dimensional taxonomy that may miss latent error patterns. We therefore introduce data-driven zoning, which automatically clusters cells using richer features that capture deviations from their columns' expected clean domains.

Table 2 defines the characterization vector $\mathbf{x}(e)$ for a cell value $e = t_k[i, j]$. Features x_1–x_5 capture syntactic deviations from clean values in column j: mask rarity, length deviation, data-type mismatch, numeric range violation, and unseen numeric formats. For x_1, values are mapped to Unicode masks representing character classes (letters, digits, punctuation). Mask rarity measures how uncommon a pattern is among clean values. Features x_6–x_9 capture semantic deviations from the clean domain D_j^c through value rarity, distance to clean-value embeddings, unseen character bigrams, and normalized edit distance. Finally, x_{10} measures contextual deviation through the strongest confidence-weighted FD violation, where $e^*_{j' \to j}$ is the value supported by clean rows with the same determinant value and FD_conf(j', j) is the dependency confidence.

We then apply mini-batch k-means clustering to these features, with $k = \lceil \sqrt{B} \rceil$, where B represents the labeling budget. This ties the zoning granularity to the available supervision. Each cluster defines a zone, whose cells share a single classifier trained on the unified feature space.

4.2.2 Sampling and Zone-level Classification. In both zoning variants, we train one binary classifier per zone on the unified feature space, pooling labeled examples across all assigned tables and columns. As classifier, we use AdaBoost based on the ablation study for Baran [24], whose classification formulation we adapt in cross-table correction. As an ensemble method it is less prone to overfitting when adequately regularized.

Given a labeling budget, rule-driven zoning first labels cells in inconsistent-pattern columns without clean values to bootstrap pattern enforcement. The remaining labels are distributed across zones proportionally to their size while preserving column diversity. Data-driven zoning applies the same strategy across clusters. In each zone, a labeled correction (e, v) forms a positive training pair, while the remaining candidates $(e, *)$ are negative examples.

5 EXPERIMENTS

We systematically evaluate error correction in multi-table settings, across the scenarios in Section 3.2. We further analyze performance by error type and validate key design choices through ablations and micro-benchmarks.

Table 3: Dataset characteristics.

MV: Missing values, T: typos, FI: formatting issues, FD: FD violations, O: numeric outliers.

Name	# Tables	# Cells	Error Rate (cell level)	Error Types
OpenData-UK	93	5,574,950	14%	T, FI, FD
MIT-DW	86	1,954,216	14%	O, T, FI, FD
DGov	363	490,986	8%	O, T, FI, FD
Quintet	5	159,822	11%	MV, T, FI, FD
Address	2	660,142	14%	MV, FI, FD
Flights	2	19,008	25%	MV, FI, FD
Soccer	2	601,686	3%	FD

5.1 Experimental Setup

5.1.1 Datasets. Table 3 displays the datasets used in our experiments. To the best of our knowledge, there are no benchmarks of real-world dirty data lakes with ground truth. We therefore simulate this setting. We use two datasets: a subset of *OpenData-UK* from the *LakeBench* collection [8], which contains unionable tables, and a subset of the *MIT Data Warehouse* [26], which contains both joinable and unionable tables. We construct the dirty versions of the tables using BART [3], an error generation tool that can inject random syntactic errors, numeric outliers, and FD violations. We use HyFD [30] to extract FDs and configure BART to introduce errors randomly on either side. We distribute errors evenly across the three types. Error generation gets computationally expensive as table size increases, so we had to exclude files larger than 4 MB and tables that BART could not parse or process successfully. Scripts for dataset construction are available in a dedicated repository[1].

For settings in which tables are unrelated in both structure and semantics, we use the *Quintet* dataset. This benchmark is widely used in the data cleaning community and includes five datasets: *Flights*, *Hospital*, *Beers*, *Rayyan*, and *Movies* [2]. We removed the columns with consistency issues in the ground truth from *Rayyan* and *Movies*. To evaluate the impact of joins, we partition the existing *Address* [24], *Flights* [24], and *Soccer* [34] datasets into joinable tables. To evaluate unions in a similarly controlled setting, we use a subset of the *DGov* dataset [2]. The construction details of these controlled environments are described in Sections 5.4 and 5.5.

5.1.2 Baselines. We consider rule-based, learning-based, and LLM-driven approaches, picking representative systems for each, and the cross-table correction systems introduced in Section 4.2.

Rule-based Systems. We include Horizon [36] and UniClean [9], both designed for scalable rule-based error correction. We use an author-provided Java implementation for Horizon and the official one for UniClean. On *Quintet*, for which UniClean provides dataset-specific configurations, we consider them alongside a variant using only FD-based AttrRelation cleaners; the latter is then used on all other datasets, reflecting the data lake setting in which those configurations are not known. We use the rules provided by dataset owners for common datasets and the FDs extracted with HyFD [30] for the tables with synthesized errors.

Learning-based Systems. We consider HoloClean [35] and Baran [24], using their official implementations. We provide HoloClean with the same set of FDs used for rule-based systems.

LLM-driven Systems. We include ZeroEC [38], which uses LLMs to correct errors and generate correction rules without training. We use the official implementation with Gemini 2.5 Flash-Lite.

[1] https://github.com/LUH-DBS/Data-Lake-Error-Generator

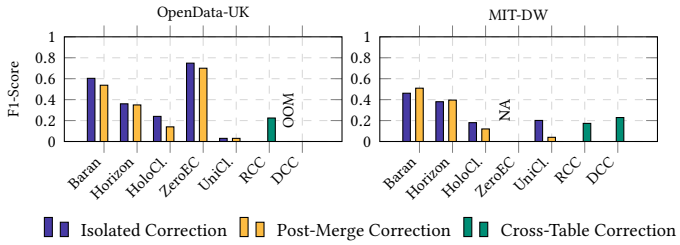


Figure 3: F1-Score comparison on OpenData-UK & MIT-DW.

Table 4: F1-score comparison for post-merge correction.

	OpenData-UK			MIT-DW		
System	Isolated	Greedy	ALITE-based	Isolated	Greedy	ALITE-based
Baran (1)	0.29	**0.32**	0.24	0.19	**0.32**	0.30
Horizon	**0.36**	**0.35**	0.07	0.38	**0.39**	0.24
HoloClean	**0.24**	0.15	0.01	**0.18**	0.12	0.01

Cross-table Correction Systems. We include cross-table correction systems with rule-driven (RCC) and data-driven zoning (DCC), introduced in Section 4.2 and implemented in Python.

5.1.3 Configurations. All runtime experiments were conducted on a server with an AMD EPYC 7543P processor (32 cores) and 512 GB of main memory. We run each experiment three times and report the average results. We cap execution time at one hour per table. For supervised approaches, we use a default labeling budget of 10 tuples in the isolated setting. Post-merge and cross-table approaches receive the same number of labeled cells as isolated counterparts.

5.2 Error Correction in Real-World Data Lakes

We evaluate isolated, post-merge, and cross-table correction on *OpenData-UK* and *MIT-DW*. Figure 3 reports the F1-scores. We exclude ZeroEC from the evaluation on *MIT-DW* due to the lack of compliance approval for passing the data to external LLMs.

Overall, the highest F1-scores are achieved by ZeroEC, Baran, and Horizon. Post-merge correction produces mixed results that vary across datasets and systems. With 10 labeled tuples per table, Baran improves by 5% on *MIT-DW*, the largest gain observed across all systems, while Horizon improves by 2%. On *OpenData-UK*, however, the same systems drop by 7% and 1%, respectively. HoloClean and UniClean perform worse in the post-merge setup on both data lakes. Both maintain high precision in the isolated and post-merge settings, but their recall is limited by the per-table timeout. These outcomes also depend on the available supervision: with one labeled tuple per table (Table 4), merging improves Baran on both lakes, including *OpenData-UK*. We discuss the impact of the labeling budget in Section 5.7.

Merging can expose new correction signals but can also reinforce spurious ones and shift distributions. In our data lakes, we discover through finer-grained analysis that joins tend to be more reliable, as they improve the performance on more tables than they hurt. This contributes to the small overall gains on *MIT-DW*, where most meaningful joins are located. Unions, in contrast, are less stable: while on *MIT-DW* their gains and losses are nearly balanced, on *OpenData-UK* they benefit more tables overall, but the negative impact on degraded tables is substantially larger.

We also compare our greedy merging strategy with ALITE [20], a state-of-the-art integration method based on full disjunction, using Blend for data discovery. Correction effectiveness is reported in Table 4. ALITE's more aggressive merges introduce additional noise, reducing effectiveness for all systems on both datasets. Thus, a more aggressive integration provides no advantages in terms of effectiveness compared to the greedy strategy.

Cross-table variants perform poorly on our real-world lakes, due to misallocation of the labeling budget across zones, and a lack of semantic correction signals. On *MIT-DW* (*OpenData-UK*), RCC achieves 37% (39%) F1-score on inconsistent-pattern zones, containing 45% (50%) of the errors, but only 2% on consistent-pattern zones, despite receiving 80% (98%) of the labeling budget. Thus, proportional allocation overspends on semantically difficult errors while neglecting easier pattern-based ones. DCC shows similar shortcomings. On *OpenData-UK*, it exceeds memory limits even with 900 GB of RAM. On *MIT-DW*, DCC improves over RCC to 23%, but clustering still fails to isolate semantic errors: the largest of 52 clusters contains 7% of all cells, 90% of which receive no labels. Many require inter-column semantic relationships absent from the unified feature space.

Effectiveness across error types. Table 5 breaks down F1-scores by error type. Errors are mostly FD violations on both datasets (64% on *OpenData-UK*, 60% on *MIT-DW*); the remainder is mainly typos/formatting issues, with less than 1% numeric outliers on *MIT-DW*. For FD violations, table merging is expected to help by manifesting additional FDs or strengthening the support for existing ones. Indeed, on *MIT-DW*, post-merge correction improves Baran by 8%. However, PM-Baran's performance drops by 10% on *OpenData-UK*: here merged tables are formed through unions, which do not contribute useful FDs but dilute existing correction signals by adding irrelevant tuples and weakening the consistency of previously useful rules. Similarly, Horizon drops by 7% on *OpenData-UK* and remains unchanged on *MIT-DW*, suggesting that merging often fails to provide additional useful rules. HoloClean is unaffected on *OpenData-UK* but loses 11% on MIT-DW. Both cross-table variants perform poorly on this error type because their shared feature space only covers aggregated dependencies.

Merging tables can substantially change their rules. On *OpenData-UK*, 101 FDs appear in both merge partners. After merging, their 202 instances collapse to 101. In addition, 115 original FDs no longer hold and 224 new FDs emerge, resulting in the net increase from 813 to 821 FDs reported in Table 8. However, the aggregate FD count is a poor proxy for repair effectiveness. Consider two tables about police reports. Their union loses the shared FD *lsoacode* ↔ *lsoaname*, which helps the correction of 13.8k FD violations, and introduces ten new rules. Although the latter correct some previously unsupported typos, the F1-score of Baran and Horizon on the two tables drops by about 50%.

Joining the academic_terms and library_reserves tables on *term_code* in *MIT-DW* improves Baran's F1-score on library_reserves (academic_terms) from 12% (21%) to 44% (24%), as the join provides correlated semester attributes as additional repair context. Horizon shows no comparable gain: its F1-score remains 57% on library_reserves and decreases from 12% to 8% on academic_terms. Because our merging strategy joins only on clean key values, matched rows receive consistent semester context while noisy keys remain unmatched. Thus, one-to-many joins can benefit co-occurrence-based systems such as Baran without improving support for rule-based systems such as Horizon.

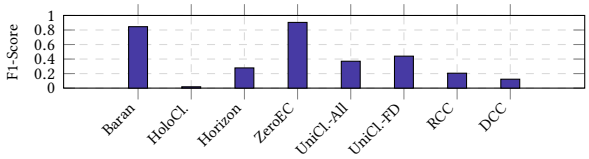


Figure 4: Isolated correction on the Quintet dataset.

For typos and formatting errors, most systems show only minor differences between isolated and post-merge correction. Baran remains nearly unchanged, while Horizon improves by 10% on *MIT-DW* and 17% on *OpenData-UK* due to additional rules that apply after merging. HoloClean drops by 15% on *OpenData-UK* and remains stable on *MIT-DW*. Cross-table correction performs better on typos than on FD violations: RCC reaches 47% F1-score on *OpenData-UK* and 27% on *MIT-DW*, while DCC reaches 30% F1-score on *MIT-DW*. Finally, numeric outliers on *MIT-DW* show clearer benefits from merging, which improves the performance of Baran, Horizon, and HoloClean. DCC also excels (96% F1-score) because 97% of the cells are in columns with one clean value, hence a single label corrects the column. RCC fails as rule-based zoning generally categorizes numeric outliers as pattern violations, and pattern enforcement cannot repair them.

Overall, post-merge and cross-table correction do not consistently outperform isolated correction. Joins can introduce helpful repair rules. Unions may provide useful redundancy or weaken correction signals by shifting value distributions. Cross-table correction struggles with semantic errors, but is promising for syntactic ones.

5.3 Evaluation on Unrelated Tables

To simulate *unrelated tables*, we use *Quintet*, whose tables share neither schema nor records and describe distinct domains. Figure 4 shows the F1-scores. For supervised approaches, the budget is 10 labeled tuples per table.

Similar to real-world lakes, ZeroEC (90%) and Baran (84%) achieve the highest F1-scores. For UniClean, we compare two configurations. UniClean-FD (using only FDs as input rules) achieves an F1-score of 44%, while UniClean-All (using the full set of type-specific cleaners such as Pattern, Date, Numbers, and Outliers) stops at 37%. However, the F1-score for the latter increases to 63% disabling the Date and Pattern cleaners. This is because UniClean normalizes date and time values to a format that differs from the *Quintet* ground truth, causing semantically valid corrections to be penalized under exact string matching. Cross-table approaches perform poorly in this setting. With RCC, most errors fall in non-unique zones, with inconsistent (80% of errors) or consistent patterns (18%). For inconsistent patterns, RCC uses the accepted pattern-enforced repairs as training data for the remaining errors. However, pattern enforcement reaches only 14% recall, leaving supervision sparse and some columns unlabeled. For consistent patterns, supervision is sufficient, but performance remains low because errors are mainly semantic and rely on table-specific inter-column signals weakened by the shared cross-table feature space. Similarly, DCC struggles

Table 5: Effectiveness across different error types (NA: not available due to compliance issues; OOM: out of memory).

	Baran	PM-Baran	Horizon	PM-Horizon	HoloCl.	PM-HoloCl.	UniCl.	PM-UniCl.	ZeroEC	PM-ZeroEC	RCC	DCC
OpenData-UK (FD)	0.54	0.44	0.50	0.43	0.02	0.02	0.05	0.02	0.70	0.64	0.03	OOM
OpenData-UK (T)	0.71	0.70	0.00	0.17	0.50	0.35	0.00	0.01	0.83	0.82	0.47	OOM
MIT-DW (FD)	0.44	0.52	0.52	0.52	0.17	0.06	0.31	0.03	NA	NA	0.08	0.15
MIT-DW (T)	0.49	0.48	0.01	0.11	0.21	0.21	0.00	0.01	NA	NA	0.27	0.30
MIT-DW (O)	0.83	0.99	0.00	0.66	0.12	0.44	0.00	0.03	NA	NA	0.01	0.96

Table 6: F1-score comparison for joinable tables.

Dataset	Baran	PM-Baran	RCC	DCC	HoloCl.	PM-HoloCl.	Horizon	PM-Horiz.	UniCl.	PM-UniCl.	ZeroEC	PM-ZeroEC
Address	**0.54**	0.42	**0.59**	0.14	0.01	0.00	0.06	**0.11**	0.11	**0.32**	0.83	**0.95**
Flights	0.58	**0.99**	0.23	0.16	0.00	0.00	0.32	**0.56**	0.57	**0.70**	0.53	**0.99**
Soccer	0.55	**0.70**	0.03	0.15	0.00	OOM	**0.66**	0.27	**0.66**	0.00	–	–

because its 14 zones remain heterogeneous: the five largest contain about 70% of erroneous cells and group errors with similar features but different correction needs.

When tables share neither schema nor entities and are topically only loosely related, isolated correction is preferable.

5.4 Impact of Joins

To isolate the effects of joins, we evaluate three table pairs with distinct configurations: *Address* uses a clean primary key, *Flights* contains errors in the join keys, and *Soccer* joins on a non-unique attribute, causing tuple expansion.

5.4.1 Joining on Clean Primary Keys. The *Address* table [24, 25] contains missing values, formatting errors, and pattern and FD violations. It satisfies three FDs: *address* → *state*; *address* → *zip*; and *zip* → *state*, as well as patterns requiring two-letter states and numeric zip codes [24]. We split it on *recordID* into identity-related attributes (*firstName*, *lastName*, and *zip*) and location-related attributes (*address* and *state*). The first table retains no FD, while the second retains *address* → *state*. Table 6 summarizes the results. Joining tables improves the performance of ZeroEC, UniClean, and Horizon. This improvement can be attributed to the two restored FDs. Baran's performance (10 labeled tuples) drops after joining. The frequent co-occurrence of errors in *zip* and *state* hurts its vicinity-based correction models. HoloClean fails to produce any corrections in the joined setting. Moving to cross-table correction, RCC achieves the second-best overall performance (59% F1-score), as it identifies the dominant pattern violations in *zip* and applies pattern enforcement. DCC achieves only 14% F1-score because its classification-only approach lacks direct pattern enforcement and struggles with mixed noise and missing dependencies.

We also use *Flights*, which contains diverse error types and four FDs with *flight* as determinant [24]. We split the dataset on *tupleID*, placing *flight* and the scheduled-time attributes in one table and the actual-time attributes in the other. As shown in Table 6, joining the two tables substantially improves correction performance for nearly all baselines, most significantly for Baran (+41%) and ZeroEC (+46%). Cross-table variants perform worse because their schema-independent signals do not preserve the explicit inter-column relationships required to correct the predominantly semantic errors in *Flights*.

5.4.2 Joining on Noisy Keys. We inject 10% errors into the join key (*tupleID*) of the second *Flights* table and perform a left outer join. Due to corrupted keys, some tuples cannot be joined, and are therefore retained as partial tuples. Further, the left-outer join introduces missing values that become noise in the cleaning process. Nevertheless, all systems still perform better after joining than in isolation, indicating that the additional contextual signals introduced by the first table outweigh the noise caused by imperfect joins.

5.4.3 Joining on Non-Unique Keys. We split *Soccer* [34] into two tables with player-related (*name*, *surname*, *team*) and team-related attributes (*team*, *season*, *manager*). Errors occur in *surname* and *season*, and four FDs span the two tables. While Baran's performance improves by 15%, Horizon's drops by 39%. The join expands the table from 200k to 3.7M rows (18.25×) and replicates 19k erroneous cells into 5M occurrences (262.95×). We also measure the average candidate-set size for erroneous cells. For each erroneous RHS cell, we count the distinct RHS values observed for its LHS key and average this count across all erroneous cells. For *surname* under *name* → *surname*, this average remains nearly unchanged (1.87 vs. 1.89). For *season*, grouping by *team*, *manager* in the joined table yields a smaller average than grouping by *team* in the isolated table (100.2 vs. 140.7). Thus, the main effect is not broader repair domains, but massive error replication and increased row-level processing. Consequently, HoloClean runs out of memory (>900 GB). The replicated errors also weaken the rule support, explaining Horizon's sharp decline. Baran, by contrast, benefits from the join induced frequency increase of correct values, which strengthens its value-frequency-based models. We omit ZeroEC from this experiment due to its high cost. It already demonstrates on other datasets the benefit of the post-merge correction.

Joining tables is highly effective when the join key is unique and low-noise, increasing informative attributes per tuple without multiplying records, thereby strengthening semantic signals.

5.5 Impact of Unions

To analyze how error correction behaves under the different union scenarios, we generate several synthetic data lakes with tables that overlap at different ratios. We begin with 17 tables obtained from data.gov [2]. These tables partition into eight schema groups, containing 2-3 tables each. All tables within a group share the same schema. We replicate these tables three times and inject different types of errors into each replica. We further split each of the 17 tables horizontally into up to 10 different tables. In total, we obtain 363 tables. Using these partitions, we generate five data lakes that vary along the record-overlap dimension while preserving schema alignment. Record overlap denotes the share of exact duplicates between clean tables before corruption.

Table 7: F1-score comparison for unionable tables.

System	Error Type	Isolated Correction	Union with No Record Overlap	Union with Record Overlap 25%	50%	75%	100%
#Tables		*363*	*24*	*141*	*119*	*85*	*17*
Baran	T	0.65	0.54	0.68	0.69	0.69	**0.97**
	O	**0.99**	0.90	0.89	**0.99**	**0.99**	0.94
	FD	0.66	**0.73**	0.63	0.65	0.65	0.61
	All	0.68	0.67	0.67	0.69	0.66	**0.78**
Horizon	T	0.31	0.22	0.32	0.33	0.34	**0.35**
	O	0.00	**0.02**	0.00	0.00	0.00	0.01
	FD	0.26	**0.37**	0.28	0.35	0.36	0.35
	All	0.27	0.29	0.28	0.33	0.34	**0.35**
UniClean	T	0.32	0.01	0.04	0.03	0.03	**0.39**
	O	0.01	0.00	**0.02**	**0.02**	**0.02**	0.00
	FD	**0.03**	0.00	0.02	0.03	**0.03**	0.01
	All	0.09	0.01	0.03	0.03	0.03	**0.20**
ZeroEC	T	0.81	0.84	0.85	0.86	0.87	**0.96**
	O	0.30	0.36	0.78	0.69	0.87	**0.91**
	FD	0.53	0.52	0.48	0.52	0.50	**0.68**
	All	0.64	0.65	0.65	0.68	0.68	**0.81**
HoloClean	All	0.00	0.00	0.00	0.00	0.00	0.00

Cross-Table (RCC/DCC): T = 0.55/0.52, O = 0.99/0.90, FD = 0.46/0.46, All = 0.55/0.52.

Table 7 summarizes the performance of all systems across overlap levels. Considering the aggregation of all error types (All), the highest F1-scores are observed at 100% overlap for all systems. ZeroEC improves from 64% (no union) to 81% at 100% overlap (+17 points), while Baran improves from 68% to 78% (+10 points). Horizon and UniClean also show consistent increases (+8 and +11 points, respectively). For intermediate overlap levels (25%–75%), the effect is less consistent: ZeroEC improves gradually from 65% to 68%, as well as Horizon from 28% to 34%, while Baran fluctuates between 67% and 69%. In contrast, UniClean remains around 3% and only improves at 100% overlap. At 0% overlap, the impact of unions is mixed, with slight improvements (+1% for ZeroEC and +2% for Horizon) or deteriorations (−1%, −8% for Baran and UniClean). As expected, full-overlap unions yield the strongest improvements across all systems due to the rich redundancy. However, comparing intermediate overlap levels to zero overlap, unions with more value overlaps do not consistently lead to performance improvements.

Analysis by Error Type. The effect of unioning varies across error types. For typos (T), all systems benefit strongly from full overlap. For example, Baran improves from 65% to 97%, and ZeroEC from 81% to 96%. For FD violations (FD), the behavior differs. Baran (73%) and Horizon (37%) achieve their highest performance at 0% overlap: in both cases, merging tables increases the number of attempted corrections. In contrast, as outliers (O) depend on column distributions, unioning with no record overlap can introduce noise. For instance, Baran drops from 99% to 90%. Post-merge variants of ZeroEC and Baran consistently outperform cross-table approaches (RCC and DCC). Although the latter achieve strong results on numeric outliers (up to 99% and 90% F1-score), they underperform on FD violations due to their lack of schema-dependent features.

Post-union correction is effective at 100% record overlap or consistent distribution. Schema alignment alone is insufficient: without record overlap, unions yield weak signals or add noise.

5.6 Runtime Comparison

Table 9 reports the correction runtime, from reading the correction input to producing the final repairs. For post-merge correction, indexing and merging are excluded and reported separately in the same table. We consider both our greedy merging strategy and ALITE, also reporting the resulting collection sizes in Table 8. The greedy strategy adds 850 seconds on *OpenData-UK* and 308 seconds on *MIT-DW*. ALITE registers higher integration overhead and faces scalability limits: merging the largest component on *MIT-DW*, with 37 tables, 94k rows, and 437 columns, did not finish within 24 hours, so we kept those tables isolated. Its larger merged tables can also increase downstream costs, e.g., HoloClean times out after one hour on 33 of the 53 *OpenData-UK* tables.

Table 8: Table sizes before and after merging.

Dataset	Strategy	#Tables	#Rows	Cells	#FDs	Largest table
OpenData	Isolated	93	516,423	5,574,950	813	8,914 × 11
	Greedy	69	516,423	6,385,467	821	16,856 × 15
	ALITE-based	53	480,486	6,301,886	536	16,736 × 56
MIT-DW	Isolated	86	218,510	1,954,216	1,656	25,653 × 15
	Greedy	62	217,777	3,241,343	1,784	42,437 × 27
	ALITE-based*	75	211,445	2,052,199	1,721	25,758 × 19*

* *The merging of the largest component timed out, so its tables are kept isolated.*

Table 9: Runtime comparison (seconds).

System	*OpenData-UK* Isol.	*OpenData-UK* PM-Greedy	*OpenData-UK* PM-ALITE	*MIT-DW* Isol.	*MIT-DW* PM-Greedy	*MIT-DW* PM-ALITE
		+Indexing: 725 +Merging*: 125	+Indexing: 725 +Merging: 228		+Indexing: 246 +Merging: 62	+Indexing: 246 +Merging: 134†
Baran (1)	751	655	729	613	617	522
Baran (10)	11,729	10,934	15,299	10,447	10,581	8,936
Horizon	85	92	287	80	152	223
HoloClean	61,545	102,118	183,658	15,577	18,811	23,715
UniClean	176,058	133,685	66,353	114,171	95,377	84,947
ZeroEC (1)	129,026	NA	NA	NA	NA	NA
ZeroEC (10)	110,493	79,872	NA	NA	NA	NA
Cross-table‡		**Total runtime**			**Total runtime**	
RCC (1)		14,832			9,926	
RCC (10)		19,096			10,182	

*: Includes discovery and materialization. †: Plus a 1-hour timeout. ‡: DCC: OOM on both.

We now examine how the correction runtime varies across systems. Horizon is the fastest on both datasets. Its runtime scales primarily with the number of rows and FDs in the input tables [36]. Although post-merge correction processes fewer tables, the merged tables are larger, increasing its runtime. Baran is the second fastest. Its runtime is primarily driven by the labeling budget, which determines the number of iterations in its online phase. This explains the large gap between Baran (1) and Baran (10), respectively using 1 and 10 labeled tuples per table. Post-merge correction slightly reduces its runtime on *OpenData-UK*. Here, the smaller number of correction runs appears to compensate for the larger merged inputs. This effect does not appear on *MIT-DW*, where merging often produces wider joined tables. As Baran derives correction signals from relationships between columns, wider tables make feature generation more expensive. HoloClean is substantially slower, especially after merging. In contrast, the runtime decreases in the post-merge setting for UniClean, because the number of separate end-to-end Spark jobs corresponds to the number of tables. For ZeroEC, runtime is dominated by the final repair stage, which relies on latency-inducing LLM API calls [38]. Repair accounts for 64.9% of runtime with 10 labeled tuples and 78.1% with one, explaining the higher runtime. Post-merge correction reduces this cost by processing fewer tables. Due to API costs, we evaluate ZeroEC post-merge only with 10 labeled tuples.

In cross-table correction, the runtime is dominated by zone processing (~98% of total runtime for RCC). Its main bottlenecks are candidate generation and feature extraction. The runtime increases

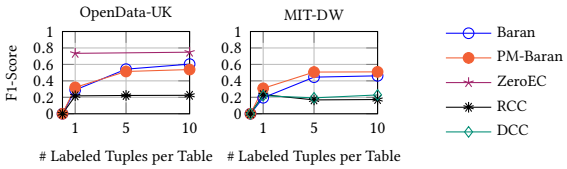


Figure 5: Impact of labeling budget.

clearly with larger labeling budgets. DCC did not finish on our machine for either dataset due to its high memory demands.

Effectiveness-cost trade-off. Runtime should be considered together with correction effectiveness (Figure 3) and merged input size (Table 8). On *OpenData-UK*, greedy merging increases the input from 5.57M to 6.39M cells, while Baran (10) remains close to isolated correction in runtime (11,784 vs. 11,729 seconds, including 850 seconds of integration) but loses 7% F1-score, favoring isolated correction. On *MIT-DW*, the input grows from 1.95M to 3.24M cells, while Baran (10) increases from 10,447 to 10,889 seconds (including 308 seconds of integration) and gains 5% F1-score. In contrast, Horizon's 2% F1-score gain comes with a 6× runtime increase. Whether merging is worthwhile therefore depends on the downstream system's main cost drivers. Expected changes in rows, columns, rules, candidate repairs, and correction runs provide system-specific indicators of its additional cost.

Methods with high per-table setup costs may benefit from merging, whereas those dominated by input size usually slow down. Cross-table correction is limited by zone-level candidate feature extraction, and requires high memory due to zone materialization.

5.7 Impact of Labeling Budget

Figure 5 reports F1-scores under different labeling budgets. With one labeled tuple per table, Baran uses 627 labeled cells on *OpenData-UK* and 405 on *MIT-DW*, reaching 29% and 19% F1, respectively. At 10 tuples per table, performance rises to 60% and 46%. PM-Baran shows a similar trend, but its effectiveness depends on the merge usefulness: merging improves results on *MIT-DW* and degrades them on *OpenData-UK*, particularly at the highest labeling budget. ZeroEC is comparatively robust to limited supervision. Even without labeled examples, the LLM can infer correction rules and repair many errors. Finally, the cross-table variants improve only slightly as the labeling budget increases. Labels are often allocated to zones with difficult errors. At larger budgets, negative candidates grow faster than positive ones, leading to fewer predicted corrections. For instance, on *OpenData-UK*, RCC achieves 99% precision but only 12% recall.

5.8 Impact of Imperfect Error Detection

We evaluate how imperfect detection affects downstream correction. Table 10 reports detection and correction performance for four representative end-to-end pipelines. Across all systems, imperfect detection reduces correction effectiveness compared to ground-truth detection. Yet, post-merge correction performs similarly to isolated correction because false negatives in error detection more often eliminate valid merge candidates than induce misleading merges. For example, for Horizon on *MIT-DW*, none of the 13 joins selected under ground-truth detection remain, so the affected tables are corrected in isolation, producing the same results as isolated correction. Specifically, false negatives in candidate join-key columns leave erroneous duplicates in the clean domain, causing these columns to fail the uniqueness check and their join candidates to be pruned during discovery.

Table 10: Impact of imperfect error detection on correction.

Pipeline	*OpenData-UK*		*MIT-DW*	
	Det. F1	Corr. F1 (Isolated/PM)	Det. F1	Corr. F1 (Isolated/PM)
Raha → Baran	0.74	0.41/0.37	0.67	0.36/0.37
MaTElDa → RCC	0.67	0.16	0.69	0.09
HoloClean	0.41	0.09 / 0.06	0.23	0.02 / 0.02
Horizon	0.38	0.36 / 0.33	0.45	0.38 / 0.38

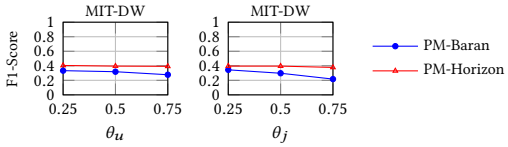


Figure 6: Discovery thresholds – Ablations.

5.9 Ablation Studies: Post-merge Correction

We evaluate the design choices of greedy post-merge correction on *OpenData-UK* and *MIT-DW*. Unless noted, Baran uses one labeled tuple per table to minimize the effect of labeling budget.

5.9.1 Set vs. Multiset Union. Multiset union may strengthen redundancy and co-occurrence signals, but yields only marginal gains: +1% F1 on *OpenData-UK* for Baran, HoloClean, and UniClean, and no change otherwise. This limited effect reflects the small number of exact duplicates, as set union removes only 2.2% of records on *OpenData-UK* and 0.9% on *MIT-DW*. Thus, multiset union is useful mainly when duplicates are frequent.

5.9.2 Discovery Thresholds. We study the influence of the discovery thresholds θ_u and θ_j on *MIT-DW*, which contains both unionable and joinable tables. We fix one threshold at 0.5 and vary the other from 0.25 to 0.75. Figure 6 shows that Horizon is highly robust because its originally effective rules remain unchanged, whereas Baran is more sensitive. Increasing θ_u from 0.5 to 0.75 reduces the selected unions from 11 to 6 and visibly decreases Baran's F1-score. Similarly, increasing θ_j from 0.5 to 0.75 reduces the joins from 13 to 4, again decreasing Baran's F1-score. We set both θ_u and θ_j to 0.5 by default.

5.9.3 FD-based Validation. We evaluate an optional FD-based validation step for candidate merges. After materializing a candidate, the validator mines FDs on the merged table and retains the merge only if at least one FD can support the correction of an erroneous dependent cell. A cell is considered correctable when its determinant values are clean and occur with a clean value of the same dependent attribute in another tuple. For joins, we consider fully joined tuples and mine cross-table FDs in both directions. For unions, we apply the same criterion to the unioned table.

We compare the variants with no validation, with validation of joins only, and with validation of both joins and unions. Table 11 shows that no validation and join-only validation yield identical

Table 11: FD-based validation — Ablations.

System	OpenData-UK			MIT-DW		
	No Val.	Join Val.	Join+Union Val.	No Val.	Join Val.	Join+Union Val.
Baran	**0.32**	**0.32**	0.30	**0.32**	**0.32**	0.30
Horizon	0.35	0.35	**0.36**	**0.39**	**0.39**	0.38
HoloClean	**0.15**	**0.15**	0.10	**0.12**	**0.12**	0.10

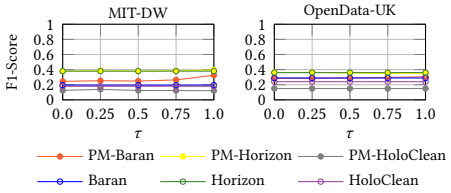


Figure 7: Distribution shift — Ablations.

Table 12: Effect of Union-First vs. Join-First Merge Ordering.

System	OpenData-UK			MIT-DW		
	Isolated	PM-Join First	PM-Union First	Isolated	PM-Join First	PM-Union First
Baran	0.29	0.29	**0.30**	0.19	**0.34**	0.32
Horizon	**0.36**	0.35	0.35	0.38	**0.39**	**0.39**
HoloClean	**0.24**	0.15	0.15	**0.18**	0.12	0.12

results across all systems and datasets. Validating unions provides no consistent benefit and decreases effectiveness by up to 5%. We therefore omit FD-based validation from the default strategy, as relationship discovery already removes weak candidates.

5.9.4 Distribution-based Validation. As an alternative to FD-based validation, we test whether merges preserve source-column distributions. For unions, we compare clean-value distributions across aligned columns. For joins, we compare each source column before and after merging. We use total variation distance for categorical columns and the Kolmogorov-Smirnov statistic for numeric columns, retaining a merge only if all metrics are at most τ. We evaluate this criterion without combining it with FD-based validation. We ablate τ for Baran, Horizon, and HoloClean in Figure 7. Higher thresholds admit more merges, but effectiveness changes only slightly and is highest when validation is nearly disabled. Thus, stricter thresholds discard mostly harmless or useful merges, suggesting that relationship discovery already filters irrelevant candidates. We therefore disable distribution validation by default.

5.9.5 Prioritizing Unions over Joins. Algorithm 1 prioritizes unions over joins. We compare union-first with join-first ordering under the default setting without validation. Table 12 reports the results. On *OpenData-UK*, both orderings produce the same merges, yielding identical results for Horizon. The small difference for Baran is due to sampling. On *MIT-DW*, join-first skips one union and treats the two tables in isolation, improving Baran by 2%, while Horizon remains unchanged. Overall, the ordering has little impact on the selected merges or correction effectiveness, supporting our assumption that favoring unions does not disadvantage correction. We therefore retain this preference, as unions increase column-level redundancy without multiplying tuples.

5.10 Ablation Studies: Cross-table Correction

We now ablate cross-table correction variants on *Quintet*.

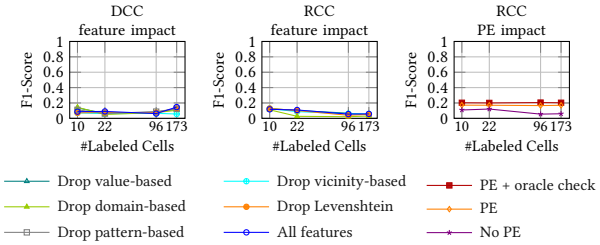


Figure 8: Cross-table correction – Ablations.

5.10.1 Feature Ablation. Figure 8 reports feature ablations for RCC and DCC across labeling budgets. We remove one feature group at a time while retaining all others. For RCC, we also disable pattern enforcement. Removing any single feature group has only a small effect, especially for DCC, whose curves are nearly indistinguishable. For RCC, slightly larger differences appear when removing vicinity-based features and, for some budgets, domain-based or syntactic features.

5.10.2 Pattern Enforcement. RCC applies dominant pattern enforcement to values with inconsistent patterns. We compare the default strategy, which accepts correct pattern-based fixes using an oracle and passes incorrect suggestions to the classifier (*PE + oracle check*), with a variant that always accepts pattern-based fixes (*PE*) and a variant that disables pattern enforcement (*No PE*). Figure 8 shows that oracle-validated pattern enforcement improves F1 by up to 20%. Without oracle validation, *PE* remains only 3–4% below, showing that it is generally a reliable pre-processing strategy.

6 CONCLUSION

We analyze several strategies for cleaning multiple tables, including isolated, post-merge, and cross-table correction. Post-merge correction is promising when merging exposes useful evidence without excessive noise or cost. While the concrete outcome requires running and evaluating the tools, several properties available *a priori* can guide this choice. For unions, relevant indicators include schema coverage, clean-value overlap, and value distributions; for joins, join-key uniqueness and errors, key overlap, and joined and dangling tuple counts. Effectiveness also depends on whether the correction system can exploit the resulting evidence through, e.g., dependencies or other signals. For supervised systems, the labeling budget can also change whether merging helps. These indicators assume detected errors are known upfront. Imperfect detection can additionally affect relationship discovery and correction.

Cross-table correction is an alternative when explicit relationships are unavailable, but struggles with label allocation and loses schema-dependent signals. Multi-table correction should therefore be applied selectively when table relationships provide reliable correction evidence. Scaling data cleaning to lake scale remains an open research challenge.

ACKNOWLEDGMENTS

We would like to thank El Kindi Rezig, Aamod Khatiwada, and Chen Yang for their help with Horizon, ALITE, and ZeroEC, respectively. This project has been supported by the German Research Foundation (DFG) under grant agreement 387872445.

REFERENCES

[1] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. 2016. Detecting Data Errors: Where are we and what needs to be done? *Proceedings of the VLDB Endowment (PVLDB)* 9, 12 (2016), 993–1004. https://doi.org/10.14778/2994509.2994518

[2] Fatemeh Ahmadi, Marc Speckmann, Malte F. Kuhlmann, and Ziawasch Abedjan. 2025. MaTElDa: Multi-Table Error Detection. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 364–376. https://doi.org/10.48786/edbt.2025.29

[3] Patricia C. Arocena, Boris Glavic, Giansalvatore Mecca, Renée J. Miller, Paolo Papotti, and Donatello Santoro. 2015. Messing Up with BART: Error Generation for Evaluating Data-Cleaning Algorithms. *Proceedings of the VLDB Endowment (PVLDB)* 9, 2 (2015), 36–47. https://doi.org/10.14778/2850578.2850579

[4] Surajit Chaudhuri and Umeshwar Dayal. 1997. An Overview of Data Warehousing and OLAP Technology. *ACM SIGMOD Record* 26, 1 (1997), 65–74. https://doi.org/10.1145/248603.248616

[5] Xu Chu, Ihab F. Ilyas, Sanjay Krishnan, and Jiannan Wang. 2016. Data Cleaning: Overview and Emerging Challenges. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 2201–2206. https://doi.org/10.1145/2882903.2912574

[6] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Holistic Data Cleaning: Putting Violations into Context. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 458–469. https://doi.org/10.1109/ICDE.2013.6544847

[7] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed K. Elmagarmid, Ihab F. Ilyas, Mourad Ouzzani, and Nan Tang. 2013. NADEEF: A Commodity Data Cleaning System. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 541–552. https://doi.org/10.1145/2463676.2465327

[8] Yuhao Deng, Chengliang Chai, Lei Cao, Qin Yuan, Siyuan Chen, Yanrui Yu, Zhaoze Sun, Junyi Wang, Jiajun Li, Ziqi Cao, Kaisen Jin, Chi Zhang, Yuqing Jiang, Yuanfang Zhang, Yuping Wang, Ye Yuan, Guoren Wang, and Nan Tang. 2024. LakeBench: A Benchmark for Discovering Joinable and Unionable Tables in Data Lakes. *Proceedings of the VLDB Endowment (PVLDB)* 17, 8 (2024), 1925–1938. https://doi.org/10.14778/3659437.3659448

[9] Xiaoou Ding, Zekai Qian, Hongzhi Wang, Siying Chen, Yafeng Tang, Hongbin Su, Huan Hu, and Chen Wang. 2025. UniClean: A Scalable Data Cleaning Solution for Mixed Errors based on Unified Cleaners and Optimized Cleaning Workflow. *Proceedings of the VLDB Endowment (PVLDB)* 18, 11 (2025), 4117–4130. https://doi.org/10.14778/3749646.3749681

[10] Yuyang Dong, Chuan Xiao, Takuma Nozawa, Masafumi Enomoto, and Masafumi Oyamada. 2023. DeepJoin: Joinable Table Discovery with Pre-trained Language Models. *Proceedings of the VLDB Endowment (PVLDB)* 16, 10 (2023), 2458–2470. https://doi.org/10.14778/3603581.3603587

[11] Mahdi Esmailoghli, Jorge-Arnulfo Quiané-Ruiz, and Ziawasch Abedjan. 2022. MATE: Multi-Attribute Table Extraction. *Proceedings of the VLDB Endowment (PVLDB)* 15, 8 (2022), 1684–1696. https://doi.org/10.14778/3529337.3529353

[12] Mahdi Esmailoghli, Christoph Schnell, Renée J. Miller, and Ziawasch Abedjan. 2025. BLEND: A Unified Data Discovery System. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 737–750. https://doi.org/10.1109/ICDE65448.2025.00061

[13] Grace Fan, Roee Shraga, and Renée J. Miller. 2024. Gen-T: Table Reclamation in Data Lakes. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 3532–3545. https://doi.org/10.1109/ICDE60146.2024.00272

[14] Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée J. Miller. 2023. Semantics-aware Dataset Discovery from Data Lakes with Contextualized Column-based Representation Learning. *Proceedings of the VLDB Endowment (PVLDB)* 16, 7 (2023), 1726–1739. https://doi.org/10.14778/3587136.3587146

[15] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. 2009. *Database Systems: The Complete Book (2nd edition)*. Pearson Education.

[16] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. 2020. Cleaning data with Llunatic. *VLDB Journal* 29, 4 (2020), 867–892. https://doi.org/10.1007/s00778-019-00586-5

[17] Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. 2017. Program Synthesis. *Foundations and Trends in Programming Languages* 4, 1-2 (2017), 1–119. https://doi.org/10.1561/2500000010

[18] Rihan Hai, Christos Koutras, Christoph Quix, and Matthias Jarke. 2023. Data Lakes: A Survey of Functions and Systems. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 35, 12 (2023), 12571–12590. https://doi.org/10.1109/TKDE.2023.3270101

[19] Alireza Heidari, Joshua McGrath, Ihab F. Ilyas, and Theodoros Rekatsinas. 2019. HoloDetect: Few-Shot Learning for Error Detection. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 829–846. https://doi.org/10.1145/3299869.3319888

[20] Aamod Khatiwada, Roee Shraga, Wolfgang Gatterbauer, and Renée J. Miller. 2022. Integrating Data Lake Tables. *Proceedings of the VLDB Endowment (PVLDB)* 16, 4 (2022), 932–945. https://doi.org/10.14778/3574245.3574274

[21] Aamod Khatiwada, Roee Shraga, and Renée J. Miller. 2026. Diverse Unionable Tuple Search: Novelty-Driven Discovery in Data Lakes. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 42–55. https://doi.org/10.48786/edbt.2026.04

[22] Aamod Khatiwada, Roee Shraga, and Renée J. Miller. 2026. Fuzzy Integration of Data Lake Tables. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 96–102. https://doi.org/10.48786/edbt.2026.08

[23] Feng Luo, Hai Lan, Hui Luo, Zhifeng Bao, J. Shane Culpepper, Shazia Sadiq, and Xiaoli Wang. 2026. Missing Value Imputation in Tabular Data Lakes Unleashed: A Hybrid Approach. *VLDB Journal* 35, 2 (2026), 11. https://doi.org/10.1007/s00778-025-00957-1

[24] Mohammad Mahdavi and Ziawasch Abedjan. 2020. Baran: Effective Error Correction via a Unified Context Representation and Transfer Learning. *Proceedings of the VLDB Endowment (PVLDB)* 13, 11 (2020), 1948–1961. https://doi.org/10.14778/3407790.3407801

[25] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha: A Configuration-Free Error Detection System. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 865–882. https://doi.org/10.1145/3299869.3324956

[26] MIT Information Systems & Technology. 2025. MIT Data Warehouse. https://ist.mit.edu/warehouse. Accessed: 2026-02-24.

[27] Avanika Narayan, Ines Chami, Laurel J. Orr, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? *Proceedings of the VLDB Endowment (PVLDB)* 16, 4 (2022), 738–746. https://doi.org/10.14778/3574245.3574258

[28] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2018. Table Union Search on Open Data. *Proceedings of the VLDB Endowment (PVLDB)* 11, 7 (2018), 813–825. https://doi.org/10.14778/3192965.3192973

[29] Wei Ni, Kaihang Zhang, Xiaoye Miao, Xiangyu Zhao, Yangyang Wu, Yaoshu Wang, and Jianwei Yin. 2025. ZeroED: Hybrid Zero-Shot Error Detection Through Large Language Model Reasoning. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 3126–3139. https://doi.org/10.1109/ICDE65448.2025.00234

[30] Thorsten Papenbrock and Felix Naumann. 2016. A Hybrid Approach to Functional Dependency Discovery. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 821–833. https://doi.org/10.1145/2882903.2915203

[31] Norman W. Paton, Jiaoyan Chen, and Zhenyu Wu. 2024. Dataset Discovery and Exploration: A Survey. *ACM Computing Surveys (CSUR)* 56, 4, Article 102 (2024), 37 pages. https://doi.org/10.1145/3626521

[32] Jinfeng Peng, Hanghai Cui, Derong Shen, Nan Tang, Yue Kou, Tiezheng Nie, Hang Cui, and Ge Yu. 2025. GARF^{+}: Self-Supervised and Interpretable Data Cleaning with Sequence Generative Adversarial Networks. *VLDB Journal* 34, 6 (2025), 64. https://doi.org/10.1007/s00778-025-00941-9

[33] Erhard Rahm and Hong Hai Do. 2000. Data Cleaning: Problems and Current Approaches. *IEEE Data Engineering Bulletin* 23, 4 (2000), 3–13. http://sites.computer.org/debull/A00dec/A00DEC-CD.pdf

[34] Joeri Rammelaere and Floris Geerts. 2018. Explaining Repaired Data with CFDs. *Proceedings of the VLDB Endowment (PVLDB)* 11, 11 (2018), 1387–1399. https://doi.org/10.14778/3236187.3236193

[35] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *Proceedings of the VLDB Endowment (PVLDB)* 10, 11 (2017), 1190–1201. https://doi.org/10.14778/3137628.3137631

[36] El Kindi Rezig, Mourad Ouzzani, Walid G. Aref, Ahmed K. Elmagarmid, Ahmed R. Mahmood, and Michael Stonebraker. 2021. Horizon: Scalable Dependency-driven Data Cleaning. *Proceedings of the VLDB Endowment (PVLDB)* 14, 11 (2021), 2546–2554. https://doi.org/10.14778/3476249.3476301

[37] Eduardo Souza dos Reis, Mohamed Abdelaal, and Carsten Binnig. 2024. Generalizable Data Cleaning of Tabular Data in Latent Space. *Proceedings of the VLDB Endowment (PVLDB)* 17, 13 (2024), 4786–4798. https://doi.org/10.14778/3704965.3704983

[38] Yangyang Wu, Chen Yang, Mengying Zhu, Xiaoye Miao, Wei Ni, Meng Xi, Xinkui Zhao, and Jianwei Yin. 2025. A Zero-Training Error Correction System with Large Language Models. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 2949–2962. https://doi.org/10.1109/ICDE65448.2025.00221

[39] Chenyu Yang, Yuyu Luo, Chuanxuan Cui, Ju Fan, Chengliang Chai, and Nan Tang. 2025. Data Imputation with Limited Data Redundancy Using Data Lakes. *Proceedings of the VLDB Endowment (PVLDB)* 18, 10 (2025), 3354–3367. https://doi.org/10.14778/3748191.3748200

[40] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 847–864. https://doi.org/10.1145/3299869.3300065

[41] Erkang Zhu, Fatemeh Nargesian, Ken Q. Pu, and Renée J. Miller. 2016. LSH Ensemble: Internet-Scale Domain Search. *Proceedings of the VLDB Endowment (PVLDB)* 9, 12 (2016), 1185–1196. https://doi.org/10.14778/2994509.2994534