Schuyler: Self-Supervised Clustering of Tables in Relational Databases

Lukas Laskowski
lukas.laskowski@hpi.de
Hasso Plattner Institute
University of Potsdam, Germany

Fabian Panse
fabian.panse@uni-a.de
University of Augsburg
Augsburg, Germany

Michael Hladik
michael.hladik@sap.com
SAP SE
Walldorf, Germany

Jan Portisch
jan.portisch@sap.com
SAP SE
Walldorf, Germany

Felix Naumann
felix.naumann@hpi.de
Hasso Plattner Institute
University of Potsdam, Germany

ABSTRACT

Databases are integral to modern applications. They enable the efficient processing of vast amounts of data, making it possible to build services that support millions of users. However, these systems often comprise hundreds of interconnected tables, complicating maintenance and comprehension. To effectively operate them, having an overview of the database is of utmost importance. Database table clustering involves grouping semantically related tables, which simplifies many database management, analysis, and integration tasks.

We present Schuyler, a system that clusters database tables by combining structural and semantic features of the database. Specifically, Schuyler fine-tunes a large language model in a self-supervised manner using triplet-loss to produce high-quality embeddings representing table semantics. Subsequently, these embeddings are clustered to achieve a database table clustering. Our approach requires no labeled training data and, thus, is applicable to arbitrary databases.

To validate Schuyler and benchmark it against state-of-the-art competitors, we introduce a benchmark collection consisting of five real-world databases. These databases vary significantly in size (29–481 tables) and complexity (3–47 clusters) and reflect diverse real-world challenges. Our results demonstrate that Schuyler consistently achieves superior clustering performance, improving the state-of-the-art on average by 0.13 ARI (adjusted Rand index) and 0.10 AMI (adjusted mutual information).

PVLDB Reference Format:
Lukas Laskowski, Fabian Panse, Michael Hladik, Jan Portisch, and Felix Naumann. Schuyler: Self-Supervised Clustering of Tables in Relational Databases. PVLDB, 19(4): 657 - 669, 2025.
doi:10.14778/3785297.3785307

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/HPI-Information-Systems/schuyler.

Proceedings of the VLDB Endowment, Vol. 19, No. 4 ISSN 2150-8097.
doi:10.14778/3785297.3785307

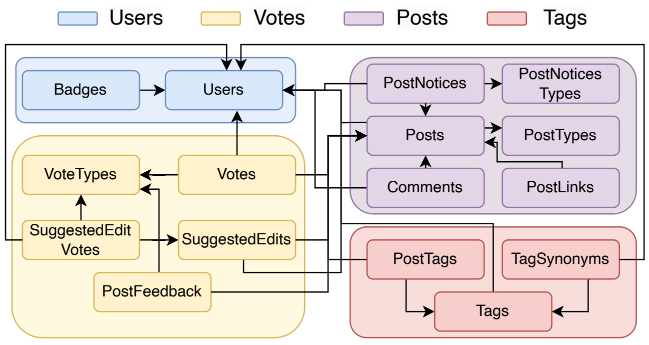

Figure 1: Database schema clustered by topics. This database schema is a subset of the Stack Exchange schema. Each color represents a ground truth cluster of tables. Often, databases consist of hundreds of tables.

1 DATABASE TABLE CLUSTERING

Nowadays, even databases of mid-sized applications, such as the e-commerce software Magento, consist of more than 200 tables [14]. Such large schemata grow over time and complicate the long-term sustainability of these databases. To extend and maintain a large schema [17, 28, 34], data engineers need to understand the schema and which tables belong to which part of the database. Without this understanding, redundancies and inconsistencies can arise. They can lead to confusion about the purpose of tables or incorrect assumptions during development, which results in poor data quality and errors. To avoid these quality issues and to be able to maintain databases even with large schemata, an abstracted overview of the database is crucial. For example, clustering the tables of the database helps a data engineer to reduce complexity and to effectively understand the database's schema. Figure 1 visualizes a possible clustering. The illustrated database is a subset of the large stack exchange database that hosts several question-answering-forums, such as StackOverflow[1]. Here, each color represents a cluster of semantically connected tables. The previously complex database schema is now clustered into several groups. This grouping allows users to better understand which tables are semantically connected

[1]https://www.stackoverflow.com

and makes it much easier for them to grasp the general topics covered by the database. Additionally, such a clustering can be used to leverage the integration and mapping of multiple databases into a common schema [22]. In this scenario, a table clustering approach helps to reduce the search space to find joinable and unionable tables. By clustering related tables within each database, the integration process can focus on matching groups rather than matching individual tables. Once the clusters are aligned, the individual tables can be matched more easily. In general, database table clustering can be applied to numerous real-world tasks where a simplification of the overall database schema is beneficial. Using table clustering in a divide-and-conquer-inspired way to extract smaller parts of the database decreases and distributes the complexity of the downstream task. Each of these parts can be worked on individually, and afterwards, the solution for each of these parts can be merged.

Database table clustering aims to find a clustering C that groups the tables T of a database D into *topically coherent* clusters (see Section 1.1 for a detailed discussion). This clustering is defined as a set of k disjoint clusters $C = \{c_1, c_2, ..., c_k\}$ with each cluster c_i being a subset of T, i.e., $c_i \subseteq T$ with $1 \leq i \leq k$ and $T = \bigcup_{i=1}^{k} c_i$.

1.1 Desiderata for Database Table Clusterings

Related tables in a database often follow certain patterns that characterize a clustering. Generally, tables within the same cluster should indicate strong semantic and functional relationships. By analyzing existing real-world database table clusterings, we identified four desiderata (soft requirements) that a well-formed database table clustering should fulfill.

(1) **Common semantic context**: Database tables within one cluster are semantically similar, e.g., by belonging to the same overarching topic. Considering Figure 1, all tables of the same cluster revolve around a shared subject. For example, all tables of the "Post"-cluster store relevant information of the users' posts in the stack exchange network.
(2) **Entity-centricity**: Databases model a specific part of the real-world. In that model, some tables typically correspond to particular types of real-world entities, which are then enhanced by additional tables. Thus, a database often consists of entity tables and supplementary tables, such as cross-reference tables, e.g., for many-to-many relationships, or entity-detail tables. A cluster containing an entity table should also contain its supplementary tables. In Figure 1, each cluster has one main table that is enhanced by further tables. For example, the `Posts` table is the entity table of the "Posts"-cluster. Other tables in the same table add further information, such as the `PostType`-table.
(3) **Functional connectivity**: Tables within clusters are usually connected via foreign keys. Since the clusters hold closely related tables, connections between tables within the same cluster tend to be more frequent than connections between tables of different clusters.
(4) **Autonomy**: Tables within one cluster represent a self-contained topic of a database that could also serve as an individual, smaller database. For example, the "Post"-cluster in Figure 1 is an encapsulated topic that could also mostly serve as an individual database.

Our database table clustering approach takes these desiderata into account and considers them during the creation of clusters.

1.2 Contributions

Databases often consist of hundreds of tables and are difficult to understand. Thus, for users to understand an unknown database, it is important to get an overview of that database. We present Schuyler, a self-supervised system for clustering database tables. While existing work relies only on syntactic information, such as foreign keys, Schuyler leverages both semantic and syntactic information. We make the following contributions:

(1) **Benchmark Collection**: We provide a suite of five easy-to-use real-world databases and their corresponding table clusterings. This benchmark enables a precise evaluation under in-practice conditions.
(2) **Schuyler system**: We introduce a self-supervised system based on triplet loss fine-tuning to create high-quality clusterings of database tables. We present and discuss multiple configurations of Schuyler.
(3) **Comprehensive Evaluation**: We prove the quality of Schuyler by evaluating the system on the presented real-world benchmark. To do so, we experiment with different configurations and perform an ablation study. Additionally, we compare Schuyler to multiple competing methods.

Schuyler is published as open-source. Its code and further artifacts, in particular the real-world benchmark databases and ground truths, are available at https://github.com/HPI-Information-Systems/schuyler.

The paper has the following outline: In Section 2, we discuss related work and contextualize our work. Section 3 presents the architecture of Schuyler and discusses design decisions. Then, we introduce our database clustering benchmark in Section 4 by presenting each database and discussing its challenges. In Section 5, we evaluate Schuyler and prove its effectiveness using the presented real-world benchmark and compare it to existing methods. Finally, we conclude and provide an outlook on future research directions in Section 6.

2 RELATED WORK

Methods to improve the understanding of a database have been developed for a long time. The following section presents similar research areas and places Schuyler into the context of existing work. An important area of research on understanding databases is database summarization [15, 25, 29, 31, 42]. The goal of this research is to generate compact representations of the database to enable new users of a database to quickly comprehend it. For example, Yang et al. [42] determine the importance of tables using random walks over the schema graph. Then, they cluster the database tables using a weighted k-Center algorithm. Similarly to Schuyler, this approach outputs a clustering of database tables.

Another system for creating clusters of database tables is iDisc, developed by Wu et al. [41]. iDisc aims to discover topical structures within a database. To do so, the authors propose a framework of multiple clustering approaches of database tables. Based on clusterings created by the different approaches and further optimizations, such as clustering boosting, they return a final output clustering.

We discuss our failed re-implementations of the approach by Yang et al. [42] and iDisc in Section 5.2.

Several approaches use community detection to create clusters of topically coherent database tables. To do so, they interpret the database schema as a graph where every table represents a node and every foreign key between two tables represents an edge. Then, these approaches apply community detection algorithms, such as the Louvain algorithm by Blondel et al. [3] or the Girvan-Newman algorithm [9]. For example, Wang et al. [40] apply the Louvain algorithm to detect communities on the database graph using a database-specific weighting scheme. In the experimental evaluation, we compare Schuyler to state-of-the-art community detection approaches for database table clustering.

3 SELF-SUPERVISED FINE-TUNING FOR DATABASE CLUSTERING

Existing approaches mainly rely on relationships between tables expressed by foreign key (FK) constraints. For example, Wang et al. [40] adopt a common community detection approach to the database table clustering scenario with foreign keys as edges in the database graph. However, while foreign key relationships provide initial hints on the cluster affiliation, solely relying on them is error-prone. Studying the provided benchmark databases shows that only 60% of all foreign keys connect two tables within one cluster. Thus, although foreign keys encode important information, using them exclusively does not necessarily help to identify communities or clusters in database schemata. To overcome this challenge, we do not mainly rely on foreign key constraints, but aim to learn table representations based on various characteristics that can then be clustered. Schuyler fuses syntactic features, such as foreign key relationships, with contextual and semantic insights of the tables and their instance data, enabling a richer understanding of relational tables. In the following, we present the architecture of Schuyler and discuss the most important design decisions.

3.1 Architecture

Schuyler aims to learn a high-dimensional vector representation (embeddings) e for each table $t \in T$ of a given database D. The embedding model is fine-tuned using triplet loss [32] (see Section 3.3 for a discussion of various triplet mining strategies). Subsequently, the learned representations of the tables are clustered using a classical clustering algorithm, such as DBSCAN [7]. As depicted in Figure 2, Schuyler consists of the following modules:

1. Database preprocessor. The database preprocessor loads and processes the database. To do so, the system follows multiple steps. First, a description of each table is generated using a general-domain LLM, such as Llama 3.1 [10]. The description later serves as input for the language model that will be fine-tuned. The goal of the generated description is to contextualize the database table to generate higher-quality embeddings. To enable the language model to create meaningful descriptions, we extract all necessary information. Then, we query an LLM with the following prompt for each table:

```
    Provide a detailed description of the database table
{table_name} with the following schema and sample data.
Also include the foreign keys.
The primary key is {pk}.
The foreign keys are {fk_description}
Schema: {schema}
Sample Data: {sample_data}
Please be concise and create a short description that
describes and contextualizes the database table well,
but not longer than ca. 80 words.
```

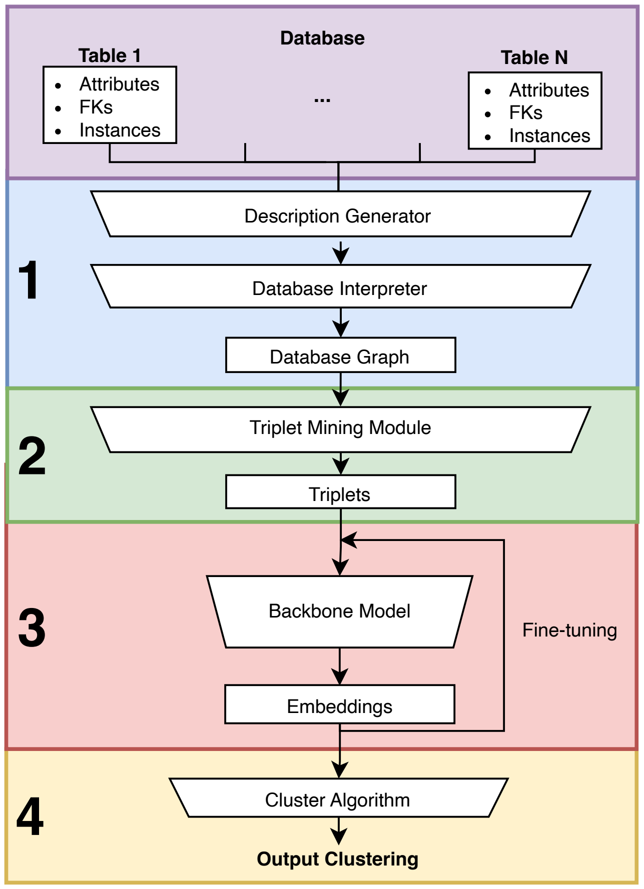

Figure 2: Architecture of Schuyler. Trapezoids represent modules, rectangulars represent data artefacts.

For each database table, we select five representative records as sample data. To identify the representative records, we first vectorize all textual representations using tf-idf [30]. Then, we cluster the resulting vectors using k-Means [18]. For each resulting centroid, we select the closest record using cosine similarity [20].

After generating the table descriptions, the module interprets the database D as a directed graph. Accordingly, every table is interpreted as a node, and two nodes share an undirected edge if a foreign key relationship exists between the corresponding tables. Then, for each node, the respective table embeddings are calculated using a pre-trained backbone model. Using such a model to create table embeddings ensures that tables with a common semantic context have similar embeddings. Then, those tables are more likely to be clustered in the same cluster as described in the table clustering desiderata (see "common semantic context" in Section 1.1). Then, as a pre-filtering step, we remove the edges of very dissimilar tables based on these initial embeddings.

2. Triplet mining module. To fine-tune the pre-trained backbone, we apply triplet loss (see Section 3.2) in a self-supervised manner. Triplet loss [32] is a common loss function for fine-tuning embedding models. Usually, triplet loss is applied in supervised scenarios, where labeled training data exists. There, triplets are selected using strategies such as semi-hard triplet selection [32]. However, when clustering database tables, no training labels are available. Each database is unique and assumed to be unknown to the users; thus they cannot provide training instances. To solve this challenge, we mine triplets using database heuristics. These triplets are not guaranteed to be correct, but follow common database practices and are guided by the clustering desiderata, as introduced in Section 1.1. They teach the model which tables are expected to belong to the same cluster. Thus, the embedding of tables within the same cluster become more similar and those embeddings being in different clusters become less similar. We present four triplet mining strategies for database table clustering in Section 3.3. In Section 5, we prove the effectiveness and generality of these strategies and compare them with each other.

3. Fine-tuning. After identifying suitable triplets, we aim to learn an embedding function emb that maps each table t to an n-dimensional numerical embedding $emb\colon T \rightarrow \mathbb{R}^n$ with $n \in \mathbb{N}$. Given a similarity function $sim\colon \mathbb{R}^n \times \mathbb{R}^n \rightarrow [0, 1]$, the similarity $sim(e_1, e_2)$ of the embeddings e_1, e_2 of two tables t_1, t_2 should be as large as possible if they belong to the same cluster. Vice versa, the similarity should be as small as possible if t_1 and t_2 belong to two different clusters.

4. Clustering. The clustering module creates the final output clustering. It takes as input the embeddings of all tables provided by the fine-tuned embedding model. Then a state-of-the-art clustering algorithm, such as DBScan [7] or AffinityPropagation [8] is applied. As users do not know the number of clusters k, we determine k for those clustering algorithms that require it beforehand, as follows: We run the clustering algorithm for multiple values of $k \in [2, 20)$ and calculate for each iteration the silhouette score by Rousseeuw [27]. The silhouette score is a metric to assess cluster quality. It intuitively measures how well a data point fits to its cluster (cohesion) compared to other clusters (separation). We select the k where the silhouette score is the highest. In Section 5.5, we experiment with multiple clustering algorithms and discuss their advantages and disadvantages.

3.2 Fine-tuning with triplet loss

Pre-trained backbone models, such as SentenceTransformer [26], provide meaningful embeddings for a given input. They are capable of providing generalized representations with embeddings of related or semantically close concepts having a high similarity. However, these models are not tuned for database table clustering. Thus, we use *Triplet Loss* for fine-tuning the backbone model and achieving high-quality representations for database table clustering. Triplet loss is a distance-based loss function that is particularly used in computer vision tasks, such as face recognition. The key idea of that loss function is to push a model producing similar embeddings for same-class instances. Thereby, it penalizes a model when two instances of the same class have different embeddings and vice versa. It rewards a model when two instances of the same class have similar embeddings and vice versa. To accomplish this reward model, the input of the loss function consists of three parts:

- **Anchor A**: The anchor input represents a reference instance, i.e., table.
- **Positive P**: The positive input represents an instance that belongs to the same class as the anchor. In database clustering, this would be a table that belongs to the same cluster.
- **Negative N**: The negative input represents an instance that is of a different class than the anchor. Thus, it would be a table of a different cluster.

Using the to-be-fine-tuned model, the embeddings of the respective instances are generated, and their distances to each other are calculated. Then, the loss is computed as follows:

$$\mathcal{L}(A, P, N) = \tag{1}$$
$$max(0, sim(emb(A), emb(N)) - sim(emb(A), emb(P)) + \alpha)$$

sim is a similarity function, such as Euclidean or cosine similarity, and the function emb computes the embedding out of the input. α is a hyperparameter, called "margin", which is used to ensure that negative samples are kept far apart. In simple words, if the negative is already enough apart from the anchor compared to the positive, the loss is zero. Otherwise, the model is penalized. While triplet loss is usually applied in supervised settings, Schuyler does not know the cluster affiliation of tables beforehand for an unknown database. Thus, the triplets to fine-tune the model need to be mined using heuristics and patterns.

3.3 Database-aware triplet mining

The selection of appropriate triplets is a challenging task. It has a strong influence on the quality of the fine-tuned model. When easy triplets are selected, such as when anchor and positive belong obviously to the same cluster, or anchor and negative are obviously of different clusters, the model does not yield new information out of the triplets. Thus, it might either overfit or not adopt at all. In the following section, we present four triplet mining strategies that suit the use case of database table clustering. We evaluate these strategies in Section 5. As described, we deploy triplet loss in an unsupervised setting. As the cluster affiliation is unknown, it is also not known which tables are positive or negative instances to a specific anchor. Consequently, next to selecting challenging triplets, it is also important to select as precisely as possible correct triplets.

Naive mining. The naive triplet mining strategy serves as a baseline. For each table (the anchor) one triple is created by randomly sampling two other tables. We select the table that is more similar to the anchor based on the initial table embeddings as positive input and the other as negative input. While this approach is very simple, it is also prone to errors. There is no inherent logic to how triplets are selected.

Similarity-based mining. The hypothesis of this mining strategy is that the semantic similarity of instances indicates cluster affiliation. Each instance is encoded using the initial pre-trained backbone model. Then, a similarity matrix between all tables is calculated. For each table, the most similar table is selected as positive input and the most dissimilar table is selected as negative input.

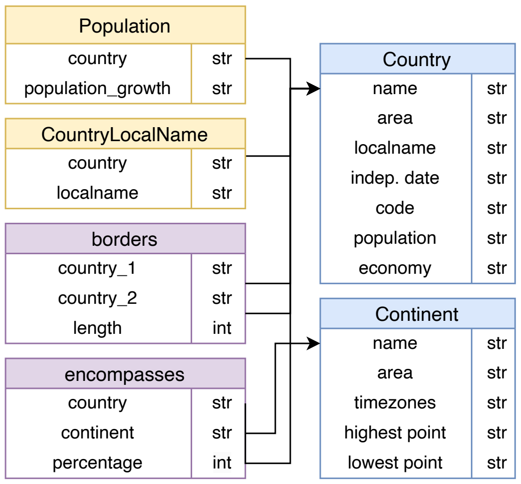

Figure 3: Database Example. Each box represents a table. Yellow boxes indicate entity-detail tables, purple boxes indicate cross-reference tables and blue boxes indicate entity tables.

Neighborhood mining. This triplet mining strategy relies on the foreign key constraints of the input database. It assumes that tables that share a foreign key also have a high probability of belonging to the same cluster. Thus, for each table as anchor input a table is selected as positive input that is directly connected to the anchor table via a foreign key. Then, a random non-neighboring table is selected as negative input. This approach harbors the risk that a chain of positives is created. Then, all tables would appear to be in the same cluster because of transitivity (table B could be positive to anchor table A, and table C could be positive to the anchor table B, and so on). The model would learn to output similar representations for all of those instances.

Database-aware mining. Database-aware triplet mining leverages the common database table clustering desideratas introduced in Section 1.1. Each database is unique, thus, we kept the mining rules simple to render them universally applicable. To construct triplets, the system analyzes the database schema and builds triplets based on schema information.

As stated in the introduction, each cluster of a clustering usually consists of one central table that reflects a certain entity with some "helper"-tables. For database-aware mining, we differentiate between the following three table types: (1) **Entity tables** describe primary, real-world objects/concepts that the database is designed to model. In Figure 3 the tables `Country` and `Continent` represent real-world concepts. Thus, they are considered as entity tables. (2) **entity-detail tables** store additional information related to records in an entity table and, thus, enhance entity tables. They refer only to their entity table and do not have any further foreign key references. For example, the tables `Population` and `CountryLocalName` add additional information to the entity table `Country` (see Figure 3). (3) **Cross-reference tables** are used to model relationships between tables. They can store additional columns that describe the relationship in more detail. In Figure 3, the two tables `Country` and `Continent` are connected via the table `encompasses`. It stores additional information, such as how much

Algorithm 1 Detection of Entity Tables

```
Require: List of tables $T$, Database Graph $G$
Ensure: List of entity tables $\mathcal{E}$
 1: $\mathcal{E} \leftarrow \emptyset$                          ▷ List of all entity tables
 2: $T \leftarrow$ RemoveReferenceTables($T$)
 3: for each table $t \in T$ do
 4:     ComputeNodeMetrics($t$, $G$)
 5: end for
 6: while $T \neq \emptyset$ do
 7:     $T \leftarrow$ sort($T$)               ▷ Sort by node metrics (descending)
 8:     $\mathcal{E} \leftarrow \mathcal{E} \cup T[0]$             ▷ Add first element to output
 9:     $N_{T[0]} \leftarrow$ GetNeighbors($G$, $T[0]$)
10:     $T \leftarrow T \setminus N_{T[0]}$     ▷ Remove all neighbours of $T[0]$ from $T$
11: end while
12: return $\mathcal{E}$
```

the continent covers the country (`percentage`). Cross-reference tables can also reference the same table twice, such as in `borders`, which stores adjacent countries.

Our database-aware triplet mining approach aims to detect such patterns and build triplets accordingly. To do so, it executes the following algorithm:

(1) Cross-reference table detection. First, the approach selects all cross-reference tables of the database. We identify cross-reference tables as tables that have at least two foreign keys and at least 50% of all columns are part of foreign key constraints.

(2) Entity table detection. As a second step, we aim to detect entity tables within the database that store primarly real-world objects. Such tables are usually less dependent on other tables compared to cross-reference tables, however, they are usually referenced a lot by other tables. Additionally, they consist of a lot of attributes compared to other tables. Algorithm 1 explains the detection algorithm in more detail. The key idea is to detect important tables by analyzing their role in the database graph and exploiting node features, such as degree or page rank. First, the algorithm collects all tables of the database and removes those that were detected as cross-reference table. Then, for each table node features are calculated and standardized using the z-score. We sort the list of tables in descending order by the sum of these standardized features. This table represents a ranking of the probability of a table being an entity table. Thus, the first table is selected as being an entity table and added to the output. Next, the table itself and all neighbors of that table are removed from the list of tables. This procedure is repeated until no tables are left in the list of tables. Then, the list of entity tables is returned. In subsequent steps, we use this list of entity tables to construct triplets, each of which consists of one entity table and two helper tables. Furthermore, we make sure that there are no triplets that contain an entity table as the anchor and positive input. These triplets teach the model to create embeddings so that the resulting clusters are entity-centric, as described in the desiderata for table clustering (see Section 1.1).

(3) Detection of entity-detail tables. We aim to detect entity-detail tables, i.e., tables that enhance entity types with supplementary information. To do so, we iterate over the non-cross reference neighbors of each entity table that are not an entity table themselves. Then, we select all tables as entity-detail tables that only refer to the entity table and do not have any additional foreign keys. Furthermore, the foreign key needs to be part of the primary key.

(4) Triplet construction. This step represents the construction of the actual triplets that are later used for fine-tuning. Its input is the combined output of the previous three steps, i.e., the list of cross-reference tables, list of entity-detail tables, and the list of entity tables. We first introduce how to mine positive inputs for each of the three table types, and then present how we mine the negative inputs.

Positive input selection. First, for each entity-detail table, we create one triplet. Here, the referenced entity table represents the anchor input, and the entity-detail table represents the positive input. Following the desideratum of "Entity-centricity" introduced in Section 1.1, the created triplets suggest including entity tables and their related supplementary tables in the same cluster. As we select the entity-detail tables based on foreign keys, we create anchor-positive pairs that are connected via foreign keys. Thus, such table pairs have similar embeddings and, then, are more likely to be selected for the same cluster (see the desideratum "Functional connectivity" in Section 1.1).

Next, we create the triplets for the cross-reference tables. As cross-reference tables usually connect two entity tables, their cluster affiliation is unambiguous. However, if multiple cross-reference tables refer to the same table with the same foreign keys, they can be expected to belong to the same cluster as the table to which both cross-reference tables refer. Thus, to reduce the amount of falsely created triplets, we focus on such groups of cross-reference tables. For each of these groups, we select for each table the most similar neighbor based on their initial embeddings as anchor input and the cross-reference table as positive input.

After mining triplets for cross-reference and entity-detail tables, we construct triplets for the entity tables. Here, we iterate over the set of entity tables and select the entity table as anchor. As positive input, we choose the most similar neighbor, based on the initial table embeddings. By doing so, we promote similar embeddings of tables that have a relationship to the same entity table and share a common semantic context ("common semantic context'' and "functional connectivity" in clustering desiderata). Preliminary experiments have shown that $n = 5$ achieves reliable results.

Negative input selection. We select the negative input of all triplets similarly to semi-hard triplet mining [32], which chooses negatives that are harder than easy negatives but not harder than positives. This approach encourages learning meaningful distinctions without destabilizing training. Formally, we mine triplets satisfying $sim(A, P) > sim(A, N) \land sim(A, N) > sim(A, P) - \alpha$, where α is a small positive constant that enforces a margin between positive and negative similarities.

We leverage that technique to select inputs as negatives that are less similar to the anchor than the positive input. We select the negative candidate whose similarity to the anchor is closest to $sim(A, P) - \alpha$, but within that margin. To ensure diversity, each anchor-negative pair is included only once. If a duplicate anchor-negative pair candidate is encountered, the next most similar negative is selected.

Table 1: Overview of Database Schemata.

Database	#Tables	#FKs	#Clusters	ØTables / Cluster
TPC-E	29	49	3	$11.0_{\pm 0.8}$
Magento	215	319	47	$4.6_{\pm 3.6}$
AdventureWorks	68	71	5	$13.6_{\pm 7.6}$
MusicBrainz	481	661	26	$18.5_{\pm 9.5}$
StackExchange	29	57	8	$3.6_{\pm 1.7}$

4 REAL-WORLD BENCHMARK DATABASES

We provide a carefully curated collection of five real-world benchmark databases that are suitable for evaluating the database table clustering task. As shown in Table 1, each database consists of a complex schema (including foreign key constraints) and actual instance data. Additionally, we provide a ground truth clustering that defines the cluster membership, created by the engineers of the respective databases. As every database originates from a real-world scenario, this benchmark can be used to predict the quality of Schuyler when applied in practice. Such databases represent the complexity and diversity in the real-world. All database scripts and their corresponding groundtruth clustering are available at https://github.com/HPI-Information-Systems/schuyler. The following section describes each benchmark database and their specific challenges in more detail.

TPC-E. TPC-E [38] is a benchmark database simulating real-world trades of a brokerage firm, including customer data and trade executions. It was originally developed to measure the efficiency of online transaction processing systems (OLTP) handling the TPC-E's workload. As shown in Table 1, the database has a relatively small schema. The creators of the TPC-E benchmark provide a clustering of the database tables in their documentation. Originally, it consisted of four groups: Broker, Customer, Dimension, and Market. Following the argumentation of Yang et al. [42], we do not include the "dimension"-cluster, as it has no semantic meaning and, therefore, removed the corresponding four tables, leaving 29 tables. The database consists of multiple entity-detail tables, such as `Last Trade` or `Settlement`. These tables only refer to exactly one (entity) table and extend that table by additional information.

Magento. Magento [14] is a real online e-commerce platform, published as open source. More than 150,000 stores have already been created using Magento [4]. The Magento database has a complex schema with 215 tables and 319 foreign keys. It stores, among other things, the data of customers, administrational system data, and product data. The related database table clustering provided by the maintainers is a hierarchical clustering with up to three hierarchy levels. As clusters, we selected the finest-grained clusters as ground truth clusters. The clustering has the most clusters among all databases, with 47 clusters. On average, each cluster consists of only 4.6 tables. The ground truth clustering also provides labeled entity tables. The labels indicate that many clusters include exactly one entity table, which supports our initially defined clustering characteristics. The schema consists of the following five overarching clusters: System, Product, Entity-attribute-value (EAV), Customer, and Sales. Only 39% of the foreign keys connect tables that belong to the same fine-grained cluster. This low value indicates that foreign keys do not help in detecting correct clusters for that schema.

AdventureWorks. The AdventureWorks database [19] simulates a fictional bicycle manufacturer. Thereby, it includes all the necessary components to holistically store the data of a company, such as tables for human resources or the manufacturing process. The database consists of 68 tables and 71 foreign keys. The ground truth clusters, provided by Microsoft, have varying sizes, with the smallest cluster having six tables and the largest cluster having 25 tables (standard deviation of 7.6). This large diversity in cluster size makes it challenging for clustering algorithms to correctly identify the small but also the large clusters. The algorithms face the risk of over-splitting or merging clusters that are not meant to be split or merged.

MusicBrainz. The MusicBrainz database by Swartz [36] is a popular community-driven open-source database. It stores structured data about music, such as artists, songs, labels, and albums. The database is the largest database in the benchmark database collection of Schuyler with 481 tables and 661 foreign key constraints. The maintainers of that database provide a set of entities in their database. Each entity consists of a set of tables that describe the entity. We derived the clusters based on these specified entities. It can happen that some of the tables occur in multiple entities, e.g., when connecting two tables of different clusters. For such tables, For those tables, we selected the cluster with which they have the strongest relational tie. To do so, we based our analysis on the relationship guide provided by MusicBrainz[2]. The clustering consists of 26 clusters. Therefore, each cluster includes on average 18.6 tables of the database. Despite having the largest database schema, the ground truth clustering of MusicBrainz has fewer clusters than the clustering of the Magento database. Also here, the ground truth clustering has a large variance in cluster size, which is challenging for many clustering algorithms.

Stack Exchange. The Stack Exchange database [35] is a publicly available data dump of the popular network of question-answering-forums, such as Stack Overflow. This database includes posts, comments, votes, and further interactions of users posting questions and answers to various topics. The ground truth clustering is provided by Leersen [16] and consists of eight clusters.

5 EXPERIMENTAL EVALUATION

In this section, we extensively evaluate Schuyler by analyzing its individual components and comparing it to competitors. First, we describe our experimental setup and, then, we answer the following questions:

(1) **Competitors**: How does Schuyler perform in quality and runtime compared to other approaches that cluster database tables? (Section 5.2)
(2) **Prompt Strategy**: Which information included in the prompt to create database table descriptions achieves the highest clustering quality? (Section 5.3)
(3) **Triplet Mining**: Which triplet mining strategy achieves the highest clustering quality (Section 5.4)
(4) **Clustering**: Which clustering algorithm achieves the highest quality? (Section 5.5)

[2]https://musicbrainz.org/relationships

5.1 Experimental setup

We executed all of our experiments on a server with an AMD EPYC 7702P 64-core processor with 504 GB of memory. Additionally, each experiment was provided with a NVIDIA RTX A6000 graphic card with 49 GB of VRAM. We evaluated the presented solutions with the benchmark databases introduced in Section 4. As quality metric, we mostly used the *adjusted Rand index (ARI)* by Hubert and Arabie [13] as quality metric. ARI is a metric to compare two clustering results. It is an advancement of the Rand index [24], which is calculated as follows: Given a set of n elements and two clusterings $A = \{A_1, A_2, ..., A_r\}$ and $B = \{B_1, B_2, ..., B_s\}$, we define a as the number of pairs of elements that are in the same cluster in A and B. Additionally, we define b as the number of pairs of elements that are in different clusters in both A and B. Then, the Rand index RI is RI $= (a+b)/\binom{n}{2}$. However, some of the correct clustering can happen randomly. Thus, the adjusted Rand index ARI is the Rand index, but corrected for chance. To do so, it defines a contingency table, in which an entry n_{ij} defines the number of elements that are shared by clusters A_i and B_j. Additionally, it defines a_i as the number of elements in cluster A_i and b_j as the number of elements in B_j. Then, ARI is calculated as follows:

$$\text{ARI} = \frac{\sum_{ij}\binom{n_{ij}}{2} - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}]/\binom{n}{2}}{\frac{1}{2}[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}] - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}]/\binom{n}{2}} \quad (2)$$

Intuitively, the ARI expresses how much the observed clustering agrees with the ideal clustering, correcting for the fact that some agreements might happen by chance. For completeness, we also report the adjusted mutual information (AMI) by Vinh et al. [39] when comparing Schuyler with the competitor methods. This metric measures the information overlap of two clusterings, corrected for chance. However, we focus the evaluation on the ARI. To obtain statistically robust averages, we ran each experiment 20 times and applied bootstrapping [6] by repeatedly sampling with replacement from these runs 500 times.

5.2 Comparison to state-of-the-art

In the following section, we evaluate the quality of Schuyler and compare it against several baseline methods for database table clustering, some of which are newly introduced to solve this task. To do so, we implemented the following methods:

5.2.1 *Competitors.* **GPT-4o**. Pre-trained large language models have achieved state-of-the-art performance in many research areas. As such models are trained on a large amount of textual data, they excel in tasks that require a semantic understanding of natural language. We include GPT-4o [21] from OpenAI as a competitor to Schuyler. To do so, we request the model to cluster tables of a database in a zero-shot fashion with the following prompt:

```
Here is an SQL script: {sql_content}. Please return
a clustering of the database tables as an array of
arrays. Do not return anything else but a list of lists
consisting of only database tables. Do not format your
```

Table 2: Quality of Schuyler and its competitors. Results are Adjusted Rand Index (ARI), Adjusted Mutual Information (AMI), and Runtime (RT) across five databases. For the Schuyler columns, the small text indicates the performance difference to the best competing model, which could be different for each database. Runtime (RT) is in m:ss. To obtain statistically robust results, all averages are repeatedly resampled with replacement (bootstrapped). Best results are bold, second-best results are underlined.

Database	**Clustering**			**ComDet**			**ClustCom**			**Node2Vec**			**GPT-4o**			**Schuyler**		
	ARI	AMI	RT	ARI	AMI	RT	ARI	AMI	RT	ARI	AMI	RT	ARI	AMI	RT	ARI	AMI	RT
MusicBrainz (MB)	0.33	0.49	5:12	0.37	0.55	0:27	0.24	0.44	5:46	0.52	0.66	5:31	0.03	0.28	0:43	**0.58** +0.06	**0.69** +0.03	5:36
Magento	0.57	0.70	2:53	0.18	0.48	0:14	0.57	0.69	3:09	0.43	0.63	3:04	0.36	0.68	0:39	**0.68** +0.11	**0.81** +0.11	3:17
TPC-E	0.43	0.50	1:36	0.40	**0.55**	0:02	0.40	0.46	1:39	0.29	0.41	1:39	0.43	**0.55**	0:07	**0.45** +0.02	0.53 -0.02	2:05
AdventureWorks (AW)	0.32	0.47	1:28	0.30	0.44	0:04	0.29	0.46	1:31	0.28	0.44	1:34	0.37	0.41	0:10	**0.39** +0.02	**0.52** +0.05	1:48
StackExchange (SE)	**0.61**	0.67	0:58	0.35	0.47	0:03	0.38	0.51	1:02	0.36	0.57	1:00	0.27	0.35	0:13	**0.61** +0.00	**0.69** +0.02	1:12
Average	0.45	0.56	2:25	0.32	0.50	0:10	0.37	0.51	2:37	0.38	0.54	2:34	0.29	0.45	0:22	**0.55** +0.10	**0.65** +0.09	2:48

```
response or add any ``` or similar. Make sure that
you include all provided tables of the sql file in the
clustering.
```

Despite the strict request in the prompt to maintain consistency, GPT-4o did not manage to include all tables in its output and even hallucinated additional tables for the databases "Magento" and "MusicBrainz" (due to their large number of tables). To make GPT-4o comparable to Schuyler, we manually adopted the result sets for both databases by including all missing tables into a single cluster and removing all hallucinated tables. This procedure achieved better results than considering each missing table as an individual cluster.

Clustering. We implemented a basic clustering approach to explore the effectiveness of a simple methodology. We achieved best results, when using the pre-trained model "all-mpnet-base-v2" to generate embeddings that are clustered subsequently. Based on preliminary experiments, we selected the best-performing algorithm, affinity propagation, to cluster the embeddings. Unlike the feature vectors proposed by iDisc, our approach leverages the semantic understanding of database tables through a large language model (LLM), resulting in improved clustering performance. The main difference to Schuyler is that we use the original model without finetuning it. Thus, the clustering method is simpler than the Schuyler system.

ComDet. Similarly to the clustering approach, we implemented a basic community detection baseline, which follows the table clustering algorithm proposed by Wang et al. [40]. Thus, we detect communities using the greedy modularity maximization algorithm by Clauset et al. [5].

ClustCom. Inspired by iDisc [41], we implemented an approach that leverages multiple techniques to cluster database tables and then aggregate the results. To do so, we execute a weighted community detection algorithm as well as a clustering algorithm. For the community detection algorithm, we selected the Leiden algorithm by Traag et al. [37], an improved version of the Louvain method [3]. As clustering approach, we cluster the embeddings of a pre-trained language model using affinity propagation [8].

We then aggregate both clustering results using a shared weighted score for every table pair, whether they end up in the same cluster. By doing so, we combine the structural capabilities of the community detection approach with the semantic LLM-based features of the clustering approach. Finally, we rerun an affinity propagation clustering on the aggregated scores to determine the output clustering.

Node2Vec. Node2Vec [11] is famous for learning effective node embeddings in graphs using random walks. We leverage this technique to learn embeddings for each database table in a graph, with tables being nodes and foreign key relationships being edges. The learned embeddings are again clustered using affinity propagation. Thus, this solution is as Schuyler based on an embedding model, which is in this case finetuned using Node2Vec.

As introduced in Section 2, iDisc and the database summarization solution by Yang et al. [42] represent suitable competitors. However, the systems were developed in 2008 and 2009, respectively, and do not provide public source code or benchmark datasets. We re-implemented the approaches, but the papers lacked sufficient information to reproduce their claims. Our implementation did not yield meaningful or consistent results on our benchmark data, and thus, we do not consider them in our experimental evaluation. Nevertheless, ClustCom shares core ideas with iDisc, such as combining clustering methods, while extending them with semantic information via a large language model. Thus, we expect it to achieve more robust and higher-quality results.

5.2.2 *Results.*

Table 2 compares the quality of Schuyler and its competitors using the benchmark databases. Generally, Schuyler achieves the highest quality among all systems. When considering the ARI, Schuyler achieves an average quality of 0.55, which improves the state-of-the-art (SOTA) by 0.10 points. For the AMI, Schuyler has an average quality of 0.65, improving the SOTA by 0.09 points. Schuyler is the best performing solution for all databases when considering ARI and for all but one database when considering the AMI. By combining semantic as well as syntactic features, Schuyler leverages more information required for clustering than the other systems. All systems achieve at least one database competitive result. However, Schuyler is the most stable system and the only system consistently operating at high quality.

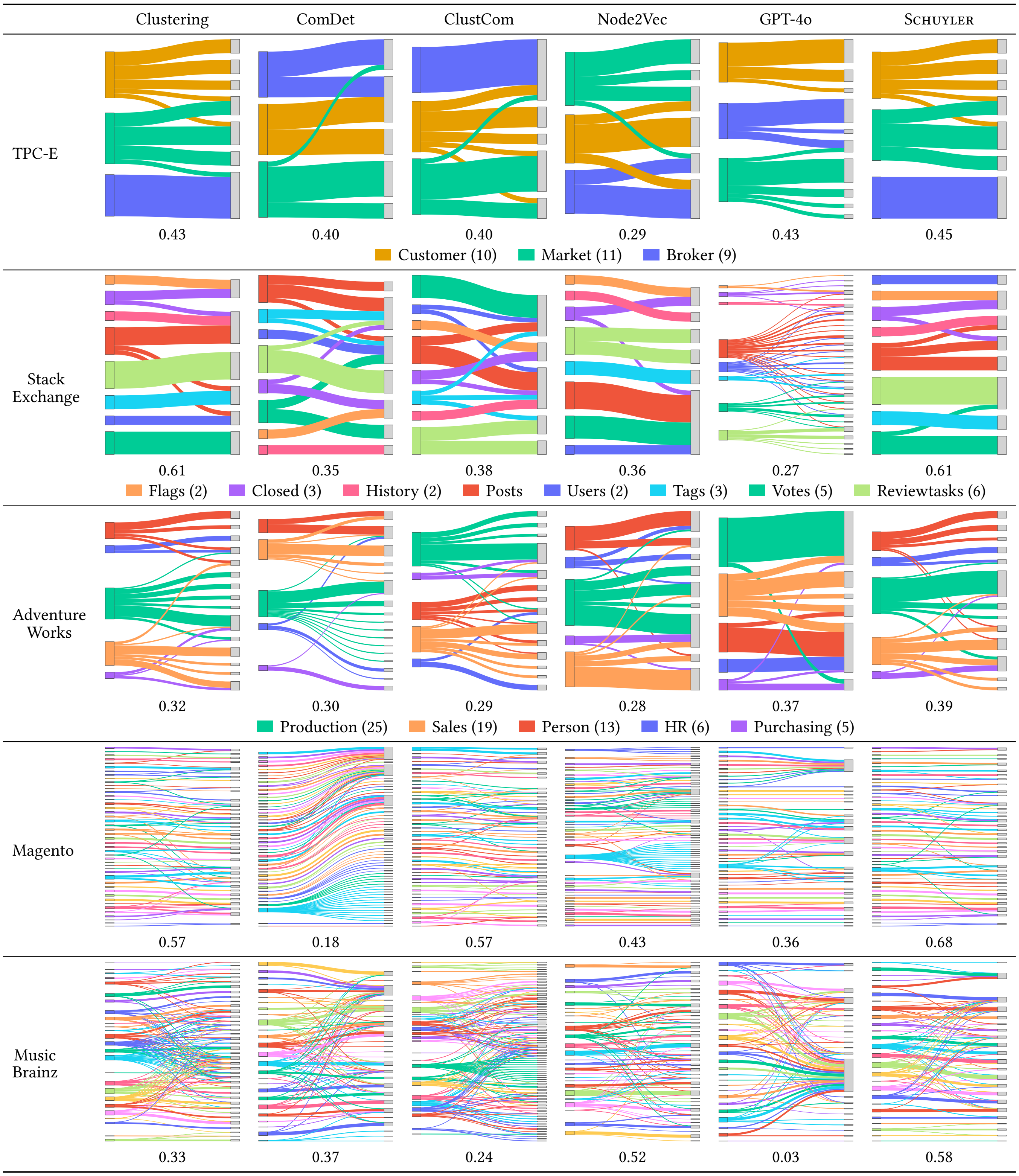

Figure 4: Alignment of ground truth and clustering results. The left represents the ground truth clusters, and the right side represents the resulting clusters for each system. The lines in between represent how well the ground truth and results align. For improved readability, we do not include a legend for the databases “Magento” and “MusicBrainz” as their groundtruth clusters consist of more than 25 clusters.

The second-best solutions, across all databases, is the basic clustering approach with an ARI of 0.45. The clustering approach works similarly to Schuyler, but does not include any fine-tuning of the embedding model. Thus, it can still extract semantic features using the large language model. The community detection approach cannot leverage any semantic information, but works only structured-based. This architectural decision might be a reason for its performance. It achieved the poorest average results over all databases (ARI of 0.32). Combining both clustering and community detection in a unified solution does yield only little quality improvements (ARI of 0.37, see ClustCom). Community detection cannot contribute a lot of valuable additional information. Although the result is better than that of the community detection approach itself, it is worse than relying solely on clustering. Node2Vec has a comparable quality to ClustCom. If the AMI is chosen as a quality measure, it becomes clear that although Node2Vec does not achieve outstanding results, it has stable results and no strong negative outliers.

The general-purpose language model GPT-4o achieves varying results. Across all databases, only the community detection approach achieves similar poor quality compared to GPT-4o. Considering the AMI, GPT-4o performs the worst. Especially with large databases, such as the MusicBrainz database, GPT-4o cannot create a high-quality clustering. In particular, it omits tables in the result clustering and hallucinates new ones. Only for the less complex TPC-E database with 29 tables, GPT-4o achieves competitive results and even the best quality of all systems for the AMI score.

5.2.3 Cluster Visualization. Figure 4 illustrates the result sets of each algorithm for each database in more detail. Each Sankey plot in the table illustrates how well the output clustering of a system aligns with the ground truth clustering. Thereby, the left side of each plot illustrates the ground truth clustering, each box represents one cluster. The right side illustrates an output clustering. The lines between both sides visualize which tables of which ground truth cluster are in which output cluster. In a perfect clustering result, each ground truth cluster box would have a corresponding box on the right side, with only one thick line connecting both boxes. Generally, the figure illustrates that all systems tend to create smaller clusters than expected by the ground truth clustering. Thereby, especially those databases with many tables and, thus, many clusters (MusicBrainz and Magento) are challenging for the clustering systems. In the following, we discuss the cluster behavior of the systems, particularly of Schuyler, in more detail.

Considering the TPC-E database, Schuyler creates mostly clusters including tables of only a single ground truth cluster; however, the clusters are smaller than the ground truth clusters. Only two tables from the customer cluster are incorrectly assigned to tables of the market cluster. The broker cluster is perfectly solved by Schuyler. GPT-4o also creates clusters consisting of tables from only one ground truth cluster, with one exception. However, many of the clusters are too small, with even four clusters consisting of only a single table. This characteristic is even more pronounced in the StackExchange database with GPT-4o. Instead of the eight ground truth clusters, GPT-4o generates 29 clusters, again with many clusters consisting of one or two tables. In contrast, the graph embedding-based system Node2Vec generates a cluster that is too large. It combines almost all tables of four different ground truth clusters. Schuyler perfectly identifies the ground truth clusters Users and Tags. For all other clusters, Schuyler creates clusters with tables from at most two different ground truth clusters.

The trend, which applies across all databases, that the systems create clusters that are too small, is particularly remarkable with the AdventureWorks database. Instead of the five ground truth clusters, all algorithms create many small clusters. Schuyler also generates many clusters that are too small. However, many of the small clusters consist exclusively of tables from the same cluster, so that Schuyler still achieves the highest quality. GPT-4o has a contrary behavior to the rest of the solutions. Rather than generating numerous small clusters, it forms only six clusters. While GPT-4o previously tended to create too many clusters for the StackExchange database, this pattern no longer appears when handling the AdventureWorks database. Compared to Schuyler, GPT-4o achieves only slightly poorer quality. The biggest shortcoming here is a large cluster consisting of the tables of the Sales, Person and HR clusters, which leads to a loss of quality.

Considering the Magento and MusicBrainz scenario, it becomes clear that Schuyler performs best when handling hundreds of database tables. Many of the ground truth clusters, are grouped by Schuyler into the same cluster. Nevertheless, no completely pure clusters are created, but for most output clusters, some misassignments exist. As described, the GPT-4o solution misses numerous existing database tables in its output clustering. As all these tables are merged into one cluster, a large cluster is created, as illustrated in Figure 4. The other solutions fail in creating meaningful clusters, resulting in many wrong assignments of tables.

5.2.4 Runtime. Table 2 presents the individual runtimes for each system and database. The two fastest systems are ComDet and GPT-4o, with an average runtime of 10s and 22s, respectively. While being the fastest solutions, they also perform the worst. All other systems require around 2min 30s on average. As the clustering system works similarly to Schuyler without the finetuning stage, their runtime differences indicate the required time to finetune the embedding model using the engineered triplets. Additionally, most of the runtime is caused by the creation of the table descriptions using the LLM. For example, for the Magento database, the description creation took 2min 24s (of 3min 17s for Schuyler). Since every table is described independently, this step scales linearly in the number of tables. In theory, these methods could also rely on faster representation methods [41]. However, preliminary experiments have shown that these do not achieve competitive qualities. In summary, Schuyler requires the most runtime compared to its competitors. However, we trade off additional runtime for quality improvements. Nevertheless, the runtime still lies within a few minutes even for databases with hundreds of tables.

5.2.5 Takeaways. Schuyler achieves the highest quality across all systems, improving the state-of-the-art by 0.10 points for ARI and 0.09 points for AMI. As all other systems, Schuyler also creates too many clusters per clustering compared to the ground truth clustering. However, most of these clusters consist only of tables that belong to the same ground truth clusters. Thus, these clusters are still helpful despite being smaller than expected. General-purpose models are omitting existing tables and hallucinating, as they are not yet ready for clustering database tables.

5.3 Prompt creation

As a first step in Schuyler, we use a pre-trained language model to create contextualized descriptions of each database table. These descriptions are then fed into an embedding model to create the table embeddings. This experiment investigates the effect on the quality of Schuyler when the input to create the tables' descriptions changes. To do so, we created prompts including different information, such as including instance data in the prompt or only the name of the table. Then, we finetuned Schuyler using database-aware triplet mining. We experimented with the following prompts:

No data. This prompt includes all schema information, such as the column names, primary and foreign keys, but no instance data.

No schema. This prompt includes representative instance data (see Section 3), but no schema information, except for the table name.

No table name. Often the table name suggests a lot of context about the table that is important for clustering. To quantify this effect, this prompt includes all data and schema information except for the table name.

Only table name. In contrast to "No table name", this prompt includes no other table information than the name of the table.

All information. This prompt includes all schema information as well as examples of instance data.

5.3.1 Results. Table 3 shows the results of this experiment. We find that the best working prompt provides all available information on the respective database table to the large language model. Using all information leads to an ARI of 0.55, which corresponds to the best performing configuration of Schuyler. Apparently, providing only the table name does not yield enough information to create a meaningful clustering of database tables. Here, the system achieved an ARI of only 0.44. Similarly, when providing all information except the table name, the model still does not achieve comparable qualities. Thus, the table name is relevant information to be leveraged by a table clustering system, and it is not sufficient on its own to create high-quality clusters. The quality also drops when no schema information is provided. Primary keys, foreign keys, and the respective attributes of a table provide valuable information necessary to achieve a high-quality clustering. Additionally, providing samples of representative instance data helps to improve the quality. However, the instance data seems to add the least amount of necessary information, as the quality drops only by 0.02 points.

Table 3: Quality of Prompts.

DB	No Data	No Schema	No Table Name	Only Table Name	All Info.
MB	0.61	0.59	0.46	0.59	0.58
Magento	0.66	0.61	0.57	0.63	0.68
TPC-E	0.30	0.19	0.36	0.16	0.45
AW	0.48	0.39	0.27	0.26	0.39
SE	0.59	0.56	0.48	0.58	0.61
Average	0.53	0.47	0.43	0.44	0.55

Results are adjusted Rand index (ARI) values across five databases.

In summary, as a design decision, we include all information in the prompt to achieve the highest quality across these databases.

5.4 Quality of triplet mining

The creation of triplets has a substantial influence on the quality of the clustering result. On the one hand, the triplets should not be too simple and unambiguous; otherwise, the model cannot learn how the embeddings have to be adapted to the use case at hand. On the other hand, since there is no pre-labeled training data, it is important to select correct triplets. There should be little to no triplets created where anchor and positive input belong to different clusters, or anchor and negative input belong to the same cluster. The following experiment investigates the effect of the triplet mining strategies, presented in Section 3.3, on the clustering results.

5.4.1 Results. Table 4 represents important metrics for each triplet mining strategy. Thereby, the column "% Pos." represents the proportion of anchor-positive pairs that are in the same ground truth cluster compared to all anchor-positive pairs. The column "% Neg." represents all anchor-negative pairs that are actually in different ground truth clusters compared to all anchor-negative pairs.

As expected, the triplet mining strategy `random` does not achieve good results. The average ARI of 0.39 is the lowest of all strategies. For the Magento and MusicBrainz databases in particular, the random strategy finds almost no correct anchor-positive pairs. Since both databases have many tables and numerous clusters, the probability of randomly selecting two tables from the same cluster is very low. In contrast, the proportion of correctly identified anchor-negative pairs is very high, with both having 99%. Nevertheless, these pairs are most likely easy-to-distinguish negative pairs because they are randomly selected. Thus, they do not teach the model much helpful information to better distinguish clusters.

In general, the similarity-based, neighbor-based, and database-aware mining strategies achieve all high qualities with the database-aware method being best.

It is noticeable that the proportion of correctly recognized anchor-positive pairs for the neighbor-based method is with 69% lower than for the similarity-based strategy (82%). This observation is consistent with the analysis of how high the proportion of foreign key relationships within clusters is in all foreign key relationships. It was found that only 60% of foreign keys link tables within a cluster. Consequently, there is a high probability that two tables are selected as anchor and positive input that are connected via a foreign key but are located in two different clusters. Nevertheless, the correctly selected anchor-positive pairs and anchor-negative pairs are not as trivial as in the similarity-based method, as the strategy achieves a higher quality. Consequently, the model can draw more information from them and thus better adapt the embeddings.

With an ARI of 0.55, the database-aware triplet mining strategy achieves the highest quality (default configuration of Schuyler). This strategy is the most promising, as it achieves the best results for all databases, except slightly worse results for the AdventureWorks and StackExchange databases. The database-aware strategy most reliably selects correct anchor-positive pairs. In 91% of cases, the anchor-positive pairs are correct, and in 89% of cases, the anchor-negative pairs are correct. The reliable identification of pairs is because this approach follows the database clustering desiderata

Table 4: Quality of Triplet Mining Strategies.

Database	Random			Similarity			Neighbor			Database aware		
	ARI	% Pos.	% Neg.	ARI	% Pos.	% Neg.	ARI	% Pos.	% Neg.	ARI	% Pos.	% Neg.
MusicBrainz	0.13	9%	99%	0.45	64%	94%	0.58	65%	94%	0.58	87%	97%
Magento	0.32	7%	99%	0.66	78%	96%	0.66	54%	99%	0.68	93%	96%
TPC-E	0.40	57%	87%	0.40	90%	73%	0.40	80%	60%	0.45	100%	73%
AdventureWorks	0.44	44%	84%	0.43	87%	72%	0.30	82%	71%	0.39	91%	76%
StackExchange	0.64	28%	97%	0.64	90%	79%	0.64	62%	97%	0.61	84%	100%
Average	0.39	29%	93%	0.52	82%	83%	0.52	69%	84%	0.55	91%	89%

previously defined in Section 1.1. Due to the reliably correct selection of triplets, the embedding quality is also higher than with the other triplet mining strategies.

5.4.2 Takeaways. The selection of a well-working triplet mining strategy is of utmost relevance, as it has a substantial influence on the quality of the table clustering algorithm. All of our proposed triplet mining strategies achieve a high quality (excluding the naïve random baseline). The novel database-aware triplet mining strategy leverages the identified desiderata of database table clusterings and achieves the highest quality among all examined strategies.

5.5 Quality of clustering algorithms

After the embeddings have been finetuned, the final cluster result is determined using a clustering algorithm. The technique of the clustering algorithm has a considerable influence on the final quality. In the following experiment, we investigate the effect of these clustering algorithms on table clustering: (1) AffinityPropagation [8], (2) k-Means [18], (3) SpectralClustering [33], (4) DBScan [7], (5) OPTICS [2], and (6) Gaussian Mixture Models [23].

5.5.1 Results. Table 5 illustrates the results of Schuyler when varying the choice of the clustering algorithm. They clearly show that affinity propagation can best handle the fine-tuned embeddings. Affinity propagation, the default clustering method for Schuyler, achieves the highest ARI with 0.55 and is, for each database, except for MusicBrainz, the best-performing method. The methods k-Means, GMMs, and spectral clustering all achieve a very similar quality of 0.42–0.44. A disadvantage of k-Means and GMMs is that they require the number of clusters as a parameter and cannot derive this from the data easily. Density-based approaches (OPTICS and DBSCAN) do not achieve competitive results.

Table 5: Clustering Quality. Results are adjusted Rand index (ARI) values across five databases.

DB	Aff.-Prop.	k-Means	DB-SCAN	Spect.-Clust.	OPT.	GMM
MB	0.58	0.54	0.25	0.45	0.31	0.54
Magento	0.68	0.45	0.09	0.41	0.25	0.45
TPC-E	0.45	0.36	0.00	0.33	0.20	0.36
AW	0.39	0.27	0.00	0.25	0.06	0.27
SE	0.61	0.56	0.04	0.68	0.12	0.56
Average	0.55	0.44	0.08	0.42	0.19	0.44

We analyzed the fine-tuned embedding shapes in more detail to understand the effectiveness of affinity propagation. An average Hopkins statistic [12] of 0.73 indicates that generally meaningful clusters are detectable. However, a low Silhouette-score [27] of 0.21 and a density variance of 0.34 suggest the presence of irregularly shaped clusters, which vary in density. It indicates that cluster boundaries might be fuzzy and non-spherical. Unlike DBSCAN or k-Means, affinity propagation does not assume uniform-density or spherical clusters with centroids [8]: elongated and irregular clusters can be better represented. These insights support the superior clustering quality of affinity propagation. Thus, for Schuyler we choose the most robust algorithm among the evaluated ones.

6 CONCLUSION AND FUTURE WORK

The demand for data to be stored will triple between 2023 and 2028 [1]. To cover this storage requirement, not only do databases need to be scaled, but increasingly complex data models also need to be managed. For instance, the functional diversity, which is reflected in the schemata, increases as the amount of data grows. Modular extensions and historically evolved relationships result in huge schemata that are difficult to understand, involving several hundred tables and foreign key relationships.

To maintain such large schemata effectively and guarantee smooth operations, it is important to keep an overview of the various components of the database. Thereby, a clustering of the tables of a database helps to understand the domains covered by the database.

We introduced Schuyler, a system to cluster tables of relational databases. Schuyler operates in a self-supervised way by computing embeddings for each database table using a finetuned large language model and afterward clustering the resulting embeddings.

To create training data for finetuning, Schuyler relies on a database-aware triplet mining process that constructs triplets based on an analysis of the database's structure, for example by detecting entity tables. In summary, Schuyler relies on structural database features, such as foreign keys, as well as exploits semantic information, such as the table name or instance data, to create high-quality, meaningful clusters of database tables. Extending Schuyler to further data representations, such as a topical clustering of spreadsheets or non-relational databases, is a promising next step.

ACKNOWLEDGMENTS

This research was funded by SAP SE.

REFERENCES

[1] 2024. *The Long-Term Case for HDD Storage.* Technical Report. Western Digital. https://documents.westerndigital.com/content/dam/doc-library/en_us/assets/public/western-digital/collateral/white-paper/white-paper-the-long-term-case-for-hdd-storage.pdf Accessed: 2025-04-11.

[2] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. 1999. OPTICS: ordering points to identify the clustering structure. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. Association for Computing Machinery, 49–60. https://doi.org/10.1145/304182.304187

[3] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* 2008, 10 (2008), P10008. http://stacks.iop.org/1742-5468/2008/i=10/a=P10008

[4] BuiltWith. 2025. Magento Usage Statistics - Trends BuiltWith. https://trends.builtwith.com/shop/Magento Accessed: 2025-02-12.

[5] Aaron Clauset, M. E. J. Newman, and Cristopher Moore. 2004. Finding community structure in very large networks. *Physical Review E* 70, 6 (Dec. 2004), 066111. https://doi.org/10.1103/PhysRevE.70.066111

[6] B. Efron. 1979. Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics* 7, 1 (1979), 1–26. http://www.jstor.org/stable/2958830

[7] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. AAAI Press, 226–231. http://www.aaai.org/Library/KDD/1996/kdd96-037.php

[8] Brendan J Frey and Delbert Dueck. 2007. Clustering by passing messages between data points. *Science* 315, 5814 (2007), 972–976. https://doi.org/10.1126/science.1136800

[9] Michelle Girvan and M. E. J. Newman. 2002. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences* 99, 12 (2002), 7821–7826. https://www.pnas.org/doi/10.1073/pnas.122653799

[10] Aaron Grattafiori et al. 2024. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783* (2024). https://arxiv.org/abs/2407.21783

[11] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. Association for Computing Machinery, 855–864. https://doi.org/10.1145/2939672.2939754

[12] Brian Hopkins and J. G. Skellam. 1954. A New Method for determining the Type of Distribution of Plant Individuals. *Annals of Botany* 18, 70 (1954), 213–227. http://www.jstor.org/stable/42907238

[13] Lawrence Hubert and Phipps Arabie. 1985. Comparing partitions. *J. of Classification* 2, 1 (1985), 193–218. http://dx.doi.org/10.1007/BF01908075

[14] Inchoo. 2014. Magento MySQL Database Structure. https://inchoo.net/magento-1/magento-mysql-database-structure/ Accessed: 2025-02-12.

[15] Do Heon Lee and Myoung-Ho Kim. 1997. Database summarization using fuzzy ISA hierarchies. *IEEE Transactions on Systems, Man, and Cybernetics* 27, 4 (1997), 671–680. https://doi.org/10.1109/3477.604110

[16] Jesse Leerssen. 2024. SEDESchema: Stack Exchange Database Schema. https://github.com/leerssej/SEDESchema Accessed: 2025-02-12.

[17] Ling Liu. 1997. Maintaining Database Consistency in the Presence of Schema Evolution. In *Proceedings of the IFIP WG 2.6 Working Conference on Database Applications Semantics (DS-6)*. Springer US, 549–572. https://doi.org/10.1007/978-0-387-34913-8_24

[18] J. B. MacQueen. 1967. Some Methods for Classification and Analysis of MultiVariate Observations. In *Berkeley Symposium on Mathematical Statistics and Probability*, Vol. 1. University of California Press, 281–297.

[19] Microsoft Learn. 2024. AdventureWorks Sample Databases: Install and Configure. https://learn.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver16&tabs=ssms Accessed: 2025-02-12.

[20] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *Proceedings of the International Conference on Learning Representations (ICLR)*. http://arxiv.org/abs/1301.3781

[21] OpenAI. 2024. GPT-4o System Card. *arXiv preprint arXiv:2410.21276* (2024). https://arxiv.org/abs/2410.21276

[22] Christine Parent and Stefano Spaccapietra. 1998. Issues and approaches of database integration. *Commun. ACM* 41, 5es (1998), 166–178. https://doi.org/10.1145/276404.276408

[23] K. Pearson. 1899. *Mathematical Contributions to the Theory of Evolution: On the reconstruction of the stature of prehistoric races.* Number Bd. 5. Royal Society.

[24] William M. Rand. 1971. Objective Criteria for the Evaluation of Clustering Methods. *Journal of the American statistical association* 66, 336 (1971), 846–850. https://doi.org/10.1080/01621459.1971.10482356

[25] Guillaume Raschia and Noureddine Mouaddib. 2002. SEQ: a fuzzy set-based approach to database summarization. *Fuzzy Sets and Systems* 129, 2 (2002), 137–162. https://doi.org/10.1016/S0165-0114(01)00197-X

[26] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, 3982–3992. https://doi.org/10.18653/v1/D19-1410

[27] Peter J. Rousseeuw. 1987. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.* 20 (1987), 53–65. https://doi.org/10.1016/0377-0427(87)90125-7

[28] Elke A. Rundensteiner, Andreas Koeller, and Xin Zhang. 2000. Maintaining data warehouses over changing information sources. *Commun. ACM* 43, 6 (June 2000), 57–62. https://doi.org/10.1145/336460.336475

[29] Régis Saint-Paul, Guillaume Raschia, and Noureddine Mouaddib. 2005. General purpose database summarization. In *Proceedings of the International Conference on Very Large Databases (VLDB)*. 733–744. http://www.vldb.org/archives/website/2005/program/paper/thu/p733-saint-paul.pdf

[30] Gerard Salton and Michael J. McGill. 1986. *Introduction to Modern Information Retrieval.* McGraw-Hill, Inc., USA.

[31] Minyar Sassi, Amel Grissa Touzi, Habib Ounelli, and Ines Aissa. 2010. About database summarization. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 18, 02 (2010), 133–151. https://doi.org/10.1142/S0218488510006453

[32] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. FaceNet: A unified embedding for face recognition and clustering. In *IEEE Conference on Computer Vision and Pattern Recognition.* 815–823. https://doi.org/10.1109/CVPR.2015.7298682

[33] Jianbo Shi and J. Malik. 2000. Normalized cuts and image segmentation. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 22, 8 (2000), 888–905. https://doi.org/10.1109/34.868688

[34] Dag Sjøberg. 1993. Quantifying schema evolution. *Information and Software Technology* 35, 1 (1993), 35–44. https://doi.org/10.1016/0950-5849(93)90027-Z

[35] Stack Exchange, Inc. 2025. Stack Exchange Data Explorer. https://data.stackexchange.com/ Accessed: 2025-02-12.

[36] Aaron Swartz. 2002. MusicBrainz: a semantic Web service. *IEEE Intelligent Systems* 17, 1 (2002), 76–77. https://doi.org/10.1109/5254.988466

[37] Vincent Antonio Traag, Ludo Waltman, and Nees Jan van Eck. 2019. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific Reports* 9, 1 (2019), 5233. https://doi.org/10.1038/s41598-019-41695-z arXiv:1810.08473

[38] Transaction Processing Performance Council. 2023. TPC Benchmark E (TPC-E), Standard Specification, Version 1.14.0. https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-e_v1.14.0.pdf Accessed: 2025-02-12.

[39] Nguyen Xuan Vinh, Julien Epps, and James Bailey. 2010. Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance. *Journal of Machine Learning Research* 11 (Dec. 2010), 2837–2854. https://dl.acm.org/doi/10.5555/1756006.1953024

[40] Xue Wang, Xuan Zhou, and Shan Wang. 2012. Summarizing Large-Scale Database Schema Using Community Detection. *Journal of Computer Science and Technology* 27 (2012), 515 – 526. https://doi.org/10.1007/s11390-012-1240-1

[41] Wensheng Wu, Berthold Reinwald, Yannis Sismanis, and Rajesh Manjrekar. 2008. Discovering topical structures of databases. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. Association for Computing Machinery, 1019–1030. https://doi.org/10.1145/1376616.1376717

[42] Xiaoyan Yang, Cecilia M. Procopiuc, and Divesh Srivastava. 2009. Summarizing relational databases. *Proceedings of the VLDB Endowment* 2, 1 (Aug. 2009), 634–645. https://doi.org/10.14778/1687627.1687699