On-the-fly Table Generation

Shuo Zhang
University of Stavanger
shuo.zhang@uis.no

Krisztian Balog
University of Stavanger
krisztian.balog@uis.no

ABSTRACT

Many information needs revolve around entities, which would be better answered by summarizing results in a tabular format, rather than presenting them as a ranked list. Unlike previous work, which is limited to retrieving existing tables, we aim to answer queries by automatically compiling a table in response to a query. We introduce and address the task of on-the-fly table generation: given a query, generate a relational table that contains relevant entities (as rows) along with their key properties (as columns). This problem is decomposed into three specific subtasks: (i) core column entity ranking, (ii) schema determination, and (iii) value lookup. We employ a feature-based approach for entity ranking and schema determination, combining deep semantic features with task-specific signals. We further show that these two subtasks are not independent of each other and can assist each other in an iterative manner. For value lookup, we combine information from existing tables and a knowledge base. Using two sets of entity-oriented queries, we evaluate our approach both on the component level and on the end-to-end table generation task.

CCS CONCEPTS

• **Information systems → Environment-specific retrieval**; *Users and interactive retrieval*; *Retrieval models and ranking*; Search in structured data;

KEYWORDS

Table generation; structured data search; entity-oriented search

ACM Reference Format:
Shuo Zhang and Krisztian Balog. 2018. On-the-fly Table Generation. In *SIGIR '18: The 41st International ACM SIGIR Conference on Research and Development in Information Retrieval, July 8–12, 2018, Ann Arbor, MI, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3209978.3209988

1 INTRODUCTION

Tables are popular on the Web because of their convenience for organizing and managing data. Tables can also be useful for presenting search results [20, 31]. Users often search for a set of things, like music albums by a singer, films by an actor, restaurants nearby, etc. In a typical information retrieval system, the matched entities are presented as a list. Search, however, is often part of a larger work task, where the user might be interested in specific attributes of these entities. Organizing results, that is, entities and their attributes, in a tabular format facilitates a better overview. E.g., for the query "video albums of Taylor Swift," we can list the albums in a table, as shown in Fig. 1.

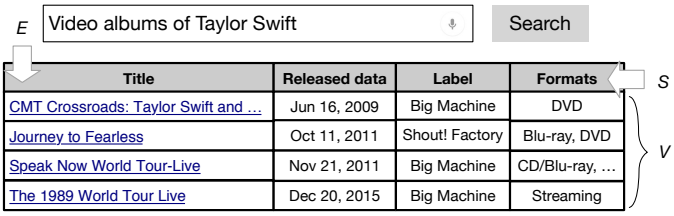


Figure 1: Answering a search query with an on-the-fly generated table, consisting of core column entities E, table schema S, and data cells V.

There exist two main families of methods that can return a table as answer to a keyword query by: (i) performing table search to find existing tables on the Web [4, 5, 19, 20, 25, 36], or (ii) assembling a table in a row-by-row fashion [31] or by joining columns from multiple tables [20]. However, these methods are limited to returning tables that already exist in their entirety or at least partially (as complete rows/columns). Another line of work aims to translate a keyword or natural language query to a structured query language (e.g., SPARQL), which can be executed over a knowledge base [29]. While in principle these techniques could return a list of tuples as the answer, in practice, they are targeted for factoid questions or at most a single attribute per answer entity. More importantly, they require data to be available in a clean, structured form in a consolidated knowledge base. Instead, we propose to generate tables on the fly in a cell-by-cell basis, by combining information from existing tables as well as from a knowledge base, such that each cell's value can originate from a different source.

In this study, we focus on *relational tables* (also referred to as *genuine tables* [27, 28]), which describe a set of entities along with their attributes [15]. A relational table consists of three main elements: (i) the *core column entities* E, (ii) the *table schema* S, which consists of the table's heading column labels, corresponding to entity attributes, and (iii) *data cells*, V, containing attribute values for each entity. The task of *on-the-fly table generation* is defined as follows: answering a free text query with an output table, where the core column lists all relevant entities and columns correspond the attributes of those entities. This task can naturally be decomposed into three main components:

(1) *Core column entity ranking*, which is about identifying the entities to be included in the core column of the table.
(2) *Schema determination*, which is concerned with finding out what should be the column headings of the table, such that these attributes can effectively summarize answer entities.
(3) *Value lookup*, which is to find the values of corresponding attributes in existing tables or in a knowledge base.

SIGIR '18, July 8–12, 2018, Ann Arbor, MI, USA

ACM ISBN 978-1-4503-5657-2/18/07…$15.00
https://doi.org/10.1145/3209978.3209988

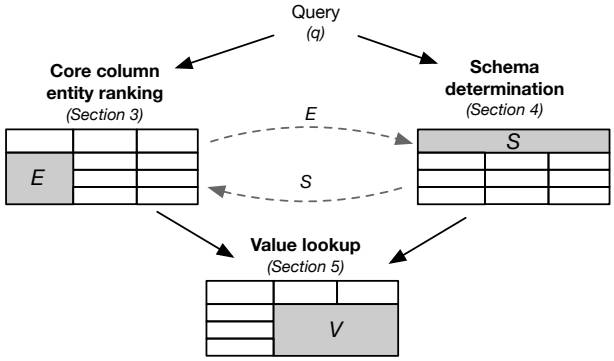


Figure 2: Overview of our table generation approach.

The first subtask is strongly related to the problem of entity retrieval [12], while the second subtask is related to the problem of attribute retrieval [14]. These two subtasks, however, are not independent of each other. We postulate that core column entity ranking can be improved by knowing the schema of the table, and vice versa, having knowledge of the core column entities can be leveraged in schema determination. Therefore, we develop a framework in which these two subtasks can be performed iteratively and can reinforce each other. As for the third subtask, value lookup, the challenge there is to find a distinct value for an entity-attribute pair, with a traceable source, from multiple sources.

In summary, the main contributions of this work are as follows:

- We introduce the task of on-the-fly table generation and propose an iterative table generation algorithm (Sect. 2).
- We develop feature-based approaches for core column entity ranking (Sect. 3) and schema determination (Sect. 4), and design an entity-oriented fact catalog for fast and effective value lookup (Sect. 5).
- We perform extensive evaluation on the component level (Sect. 7) and provide further insights and analysis (Sect. 8).

The resources developed within this study are made publicly available at https://github.com/iai-group/sigir2018-table.

2 OVERVIEW

The objective of on-the-fly table generation is to assemble and return a relational table as the answer in response to a free text query. Formally, given a keyword query q, the task is to return a table $T = (E, S, V)$, where $E = \langle e_1, \ldots e_n \rangle$ is a ranked list of core column entities, $S = \langle s_1, \ldots s_m \rangle$ is a ranked list of heading column labels, and V is an n-by-m matrix, such that v_{ij} refers to the value in row i and column j of the matrix ($i \in [1..n]$, $j \in [1..m]$). According to the needed table elements, the task boils down to (i) searching core column entities, (ii) determining the table schema, and (iii) looking up values for the data cells. Figure 2 shows how these three components are connected to each other in our proposed approach.

2.1 Iterative Table Generation Algorithm

There are some clear sequential dependencies between the three main components: *core column entity ranking* and *schema determination* need to be performed before *value lookup*. Other than that, the former two may be conducted independently of and parallel to each other. However, we postulate that better overall performance may be achieved if core column entity ranking and schema determination would supplement each other. That is, each would make use of not only the input query, but the other's output as well. To this end, we propose an iterative algorithm that gradually updates core column entity ranking and schema determination results.

Algorithm 1: Iterative Table Generation

```
Data: q, a keyword query
Result: T = (E, S, V), a result table
begin
    E^0 ← rankEntites(q, {});
    S^0 ← rankLabels(q, {});
    t ← 0 ;
    while ¬terminate do
        t ← t + 1 ;
        E^t ← rankEntites(q, S^{t−1});
        S^t ← rankLabels(q, E^{t−1});
    end
    V ← lookupValues(E^t, S^t);
    return (E^t, S^t, V)
end
```

The pseudocode of our approach is provided in Algorithm 1, where *rankEntites*(), *rankLabels*(), and *lookupValues*() refer to the subtasks of core column entity ranking, schema determination, and value lookup, respectively. Initially, we issue the query q to search entities and schema labels, by *rankEntites*(q, {}) and *rankLabels*(q, {}). Then, in a series of successive iterations, indexed by t, core column entity ranking will consider the top-k ranked schema labels from iteration $t-1$ (*rankEntites*(q, S^{t-1})). Analogously, schema determination will take the top-k ranked core column entities from the previous iteration (*rankLabels*(q, E^{t-1})). These steps are repeated until some termination condition is met, e.g., the rankings do not change beyond a certain extent anymore. We leave the determination of a suitable termination condition to future work and will use a fixed number of iterations in our experiments. In the final step of our algorithm, we look up values V using the core column entities and schema (*lookupValues*(E^t, S^t)). Then, the resulting table (E^t, S^t, V) is returned as output.

2.2 Data Sources

Another innovative element of our approach is that we do not rely on a single data source. We combine information both from a collection of existing tables, referred to as the *table corpus*, and from a knowledge base. We shall assume that there is some process in place that can identify relational tables in the table corpus, based on the presence of a core column. We further assume that entities in the core column are linked to the corresponding entries in the knowledge base. The technical details are described in Sect. 6. Based on the information stored about each entity in the knowledge base, we consider multiple entity representations: (i) *all* refers to the concatenation of all textual material that is available about the entity (referred to as "catchall" in [12]), (ii) *description* is based on the entity's short textual description (i.e., abstract or summary), and (iii) *properties* consists of a restricted set of facts (property-value pairs) about the entity. We will use DBpedia in our experiments,

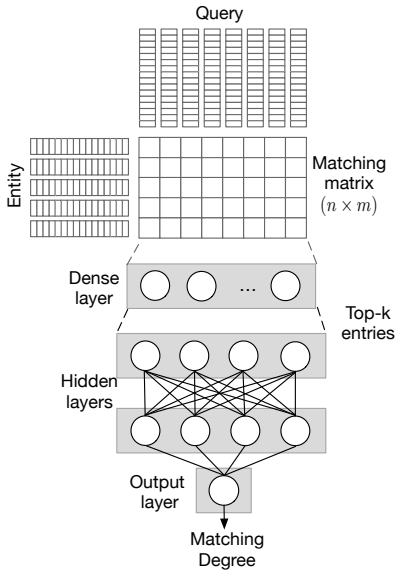


Figure 3: Architecture of the DRRM_TKS deep semantic matching method.

but it can be assumed, without loss of generality, that the above information is available in any general-purpose knowledge base.

3 CORE COLUMN ENTITY RANKING

In this section, we address the subtask of *core column entity ranking*: given a query, identify entities that should be placed in the core column of the generated output table. This task is closely related to the problem of ad hoc entity retrieval. Indeed, our initial scoring function is based on existing entity retrieval approaches. However, this scoring can be iteratively improved by leveraging the identified table schema. Our iterative scoring function combines multiple features as ranking signals in a linear fashion:

$$score_t(e, q) = \sum_i w_i \phi_i(e, q, S^{t-1}) \,, \tag{1}$$

where ϕ_i is a ranking feature and w_i is the corresponding weight. In the first round of the iteration ($t = 0$), the table schema is not yet available, thus S^{t-1} by definition is an empty list. For later iterations ($t > 0$), S^{t-1} is computed using the methods described in Sect. 4. For notational convenience, we shall write S to denote the set of top-k schema labels from S^{t-1}. In the remainder of this section, we present the features we developed for core column entity ranking; see Table 1 for a summary.

3.1 Query-based Entity Ranking

Initially, we only have the query q as input. We consider term-based and semantic matching as features.

3.1.1 Term-based matching. There is a wide variety of retrieval models for term-based entity ranking [12]. We rank document-based entity representations using Language Modeling techniques. Despite its simplicity, this model has shown to deliver competitive performance [12]. Specifically, following [12], we use the *all* entity representation, concatenating all textual material available about a given entity.

Table 1: Features used for core column entity retrieval.

Feature		Iter. (t)
Term-based matching		
ϕ_1:	$LM(q, e_a)$	≥ 0
Deep semantic matching		
ϕ_2:	$DRRM\_TKS(q, e_d)$	≥ 0
ϕ_3:	$DRRM\_TKS(q, e_p)$	≥ 0
ϕ_4:	$DRRM\_TKS(s, e_d)$	≥ 1
ϕ_5:	$DRRM\_TKS(s, e_p)$	≥ 1
ϕ_6:	$DRRM\_TKS(q \oplus s, e_d \oplus e_p)$	≥ 1
Entity-schema compatibility		
ϕ_7:	$ESC(S, e)$	≥ 1

3.1.2 Deep semantic matching. We employ a deep semantic matching method, referred to as DRRM_TKS [9]. It is an enhancement of DRRM [11] for short text, where the matching histograms are replaced with the top-k strongest signals. Specifically, the entity and the query are represented as sequences of embedding vectors, denoted as $e = [w_1^e, w_2^e, ..., w_n^e]$ and $q = [w_1^q, w_2^q, ..., w_m^q]$. An $n \times m$ matching matrix M is computed for their joint representation, by setting $M_{ij} = w_i^e \cdot (w_j^q)^\intercal$. The values of this matrix are used as input to the dense layer of the network. Then, the top-k strongest signals, based on a softmax operation, are selected and fed into the hidden layers. The output layer computes the final matching score between the query and entity. The architecture of DRRM_TKS is shown in Fig. 3.

We instantiate this neural network with two different entity representations: (i) using the entity's textual description, e_d, and (ii) using the properties of the entity in the knowledge base, e_p. The matching degree scores computed using these two representations, $DRRM\_TKS(q, e_d)$ and $DRRM\_TKS(q, e_p)$, are used as ranking features ϕ_2 and ϕ_3, respectively.

3.2 Schema-assisted Entity Ranking

After the first iteration, core column entity ranking can be assisted by utilizing the determined table schema from the previous iteration. We present a number of additional features that incorporate schema information.

3.2.1 Deep semantic matching. We employ the same neural network as before, in Sect. 3.1.2, to compute semantic similarity by considering the table schema. Specifically, all schema labels in S are concatenated into a single string s. For the candidate entities, we keep the same representations as in Sect. 3.1.2. By comparing all schema labels s against the entity, we obtain the schema-assisted deep features $DRRM\_TKS(s, e_d)$ and $DRRM\_TKS(s, e_p)$. Additionally, we combine the input query with the schema labels, $q \oplus s$, and match it against a combined representation of the entity, $e_d \oplus e_p$, where $\oplus$ refers to the string concatenation operation. The resulting matching score is denoted as $DRRM\_TKS(q \oplus s, e_d \oplus e_p)$.

3.2.2 Entity-schema compatibility. Intuitively, core column entities in a given table are from the same semantic class, for example, athletes, digital products, films, etc. We aim to capture their semantic compatibility with the table schema, by introducing a measure called *entity-schema compatibility*.

We compare the property labels of core column entities E against schema S to build the compatibility matrix C. Element C_{ij} of the

Table 2: Features used for schema determination.

Feature			Iter. (t)
Column population	ϕ_1:	$P(s\|q)$	≥ 0
	ϕ_2:	$P(s\|q, E)$	≥ 1
Deep semantic matching	ϕ_3:	$DRRM\_TKS(s, q)$	≥ 0
Attribute retrieval	ϕ_4:	$AR(s, E)$	≥ 1
Entity-schema compatibility	ϕ_5:	$ESC(s, E)$	≥ 1

matrix is a binary indicator between the jth schema label and the ith entity, which equals to 1 if entity e_i has property s_j. To check if an entity has a given property, we look for evidence both in the knowledge base and in the table corpus. Formally:

$$C_{ij} = \begin{cases} 1, & \text{if } match_{KB}(e_i, s_j) \vee match_{TC}(e_i, s_j) \\ 0, & \text{otherwise .} \end{cases}$$

where $match_{KB}(e_i, s_j)$ and $match_{TC}(e_i, s_j)$ are binary indicator functions. The former is true if entity e_i has property s_j in the knowledge base, the latter is true if there exists a table in the table corpus where e_i is a core column entity and s_j is a schema label. Then, the entity-schema compatibility score, which is used as ranking feature ϕ_7, is computed as follows:

$$ESC(S, e_i) = \frac{1}{|S|} \sum_j C_{ij} \ .$$

For example, for query "Apollo astronauts walked on the Moon" and schema {country, date of birth, time in space, age at first step, ...}, the ESC scores of entities Alan Shepard, Charles Duke, and Bill Kaysing are 1, 0.85, and 0.4, respectively. The former two are Apollo astronauts who walked on the Moon, while the latter is a writer claiming that the six Apollo Moon landings were a hoax.

4 SCHEMA DETERMINATION

In this section, we address the subtask of *schema determination*, which is to return a ranked list of labels to be used as heading column labels (*labels*, for short) of the generated output table. The initial ranking is based on the input query only. Then, this ranking is iteratively improved by also considering the core column entities. Our scoring function is defined as follows:

$$score_t(s, q) = \sum_i w_i \phi_i(s, q, E^{t-1}), \tag{2}$$

where ϕ_i is a ranking feature with a corresponding weight w_i. For the initial ranking ($t = 0$), core column entities are not yet available, thus E^{t-1} is an empty list. For successive iterations ($t > 0$), E^{t-1} is computed using the methods described in Sect. 3. Since we are only interested in the top-k entities, and not their actual retrieval scores, we shall write E to denote the set of top-k entities in E^{t-1}. Below, we discuss various feature functions ϕ_i for this task, which are also summarized in Table 2.

4.1 Query-based Schema Determination

At the start, only the input query q is available for ranking labels. To collect candidate labels, we first search for tables in our table corpus that are relevant to the query. We let $\mathcal{T}$ denote the set of top-k ranked tables. Following [35], we use BM25 to rank tables based on their textual content. Then, the column heading labels are extracted from these tables as candidates: $\mathcal{S} = \{s | s \in T_S, T \in \mathcal{T}\}$.

4.1.1 Column population. Zhang and Balog [35] introduce the task of *column population*: generating a ranked list of column labels to be added to the column headings of a given seed table. We can adapt their method by treating the query as if it was the caption of the seed table. Then, the scoring of schema labels is performed according to the following probabilistic formula:

$$P(s|q) = \sum_{T \in \mathcal{T}} P(s|T) P(T|q) \ ,$$

where related tables serve as a bridge to connect the query q and label s. Specifically, $P(s|T)$ is the likelihood of the schema label given table T and is calculated based on the maximum edit distance [16], $dist$,[1] between the s and the schema labels of T:

$$P(s|T) = \begin{cases} 1, & \max_{s' \in T_S} dist(s, s') \geq \gamma \\ 0, & \text{otherwise .} \end{cases} \tag{3}$$

The probability $P(T|q)$ expresses the relevance of T given the query, and is set proportional to the table's retrieval score (here: BM25).

4.1.2 Deep semantic matching. We employ the same neural network architecture as in Sect. 3.1.2 for comparing labels against the query. For training the network, we use our table corpus and treat table captions as queries. All caption-label pairs that co-occur in an existing table are treated as positive training instances. Negative training instances are sampled from the table corpus by selecting candidate labels that do not co-occur with that caption. The resulting matching score, $DRRM\_TKS(s, q)$, is used as feature ϕ_3.

4.2 Entity-assisted Schema Determination

After the initial round, schema determination can be assisted by considering the set of top-k core column entities, E. The set of candidate labels, from before, is expanded with (i) schema labels from tables that contain any of the entities in E in their core column and (ii) the properties of E in the knowledge base.

4.2.1 Entity enhanced column population. We employ a variant of the column population method from [35] that makes use of core column entities:

$$P(s|q, E) = \sum_T P(s|T) P(T|q, E) \ .$$

The schema label likelihood $P(s|T)$ is computed the same as before, cf. Eq. (3). The main difference is in the table relevance estimation component, which now also considers the core column entities:

$$P(T|q, E) = \frac{P(T|E) P(T|q)}{P(T)} \ .$$

Here, $P(T|E)$ is the fraction of the core column entities covered by a related table, i.e., $|T_E \cap E|/|E|$, and $P(T|q)$ is the same as in §4.1.1.

4.2.2 Attribute retrieval. Attribute retrieval refers to the task of returning a ranked list of attributes that are relevant given a set of entities [14]. Using the core column entities as input, we employ the method proposed by Kopliku et al. [14], which is a linear combination of several features:

$$AR(s, E) = \frac{1}{|E|} \sum_{e \in E} \big(match(s, e, T) + drel(d, e) + sh(s, e) + kb(s, e) \big) \ .$$

[1] Note that despite the name used in [16], it is in fact a similarity measure.

The components of this formula are as follows:

- $match(s, e, T)$ compares the similarity between an entity and a schema label with respect to a given table T. We take T to be the table that is the most relevant to the query ($\arg\max_{T \in \mathcal{T}} P(T|q)$). This matching score is the difference between the table match score and shadow match score:

  $$match(s, e, T) = match(e, T) - match(e, shadow(a)) .$$

  The table match score is computed by representing both the entity and table cells T_{xy} as term vectors, then taking the maximum cosine distance between the two:

  $$match(e, T) = max_{T_{xy} \in T} cos(e, T_{xy}) .$$

  For latter component, the notion of a *shadow area* is introduced: $shadow(a)$ is set of cells in the table that are in the same row with e or are in the same column with the s. Then, the shadow match score is estimated as:

  $$match(e, shadow(a)) = \max_{T_{xy} \in shadow(a)} \cos(e, T_{xy}) .$$

- $drel(d, e)$ denotes the relevance of the document d that contains T:

  $$drel(e) = \frac{\#results - rank(d)}{\#results} ,$$

  where $\#results$ is the number of retrieved results for entity e and $rank(d)$ is the rank of document d within this list.
- $sh(s, e)$ corresponds to the number of search results returned by a Web search engine to a query "$\langle s \rangle$ of $\langle e \rangle$," where s and e are substituted with the label and entity, respectively. If the base-10 logarithm of the number of hits exceeds a certain threshold (10^6 in [14]) then the feature takes a value of 1, otherwise it is 0.
- $kb(s, e)$ is a binary score indicating whether label s is a property of entity e in the knowledge base (i.e., $s \in e_p$).

4.2.3 Entity-schema compatibility. Similar to Sect. 3.2.2, we employ the entity-schema compatibility feature for schema determination as well. As before, C is a compatibility matrix, where C_{ij} denotes whether entity e_i has property s_j. The ESC score is then computed as follows:

$$ESC(s_j, E) = \frac{1}{|E|} \sum_i C_{ij} .$$

5 VALUE LOOKUP

Having the core column entities and the schema determined, the last component in our table generation approach is concerned with the retrieval of the data cells' values. Formally, for each row (entity) $i \in [1..n]$ and column (schema label) $j \in [1..m]$, our task is to find the value V_{ij}. This value may originate from an existing table in our table corpus or from the knowledge base. The challenges here are twofold: (i) how to match the schema label s_j against the labels of existing tables and knowledge base predicates, and (ii) how to deal with the case when multiple, possibly conflicting values may be found for a given cell.

We go about this task by first creating a catalogue $\mathcal{V}$ of all possible cell values. Each possible cell value is represented as a quadruple $\langle e, s, v, p \rangle$, where e is an entity, s is a schema label, v is a value, and p is provenance, indicating the source of the information. The source may be a knowledge base fact or a particular table in the table corpus. An entity-oriented view of this catalog is a filtered set of triples where the given entity stands as the first component of the quadruple: $e_V = \{\langle s, v, p \rangle | \langle e, s, v, p \rangle \in \mathcal{V}\}$. We select a single value for a given entity e and schema label s according to:

$$score(v, e, s, q) = \max_{\substack{\langle s', v, p \rangle \in e_V \\ match(s, s')}} conf(p, q) ,$$

where $match(s, s')$ is a soft string matching function (detailed in Sect. 6.3) and $conf(p, q)$ is the confidence associated with provenance p. Motivated by the fact that the knowledge base is expected to contain high-quality manually curated data, we set the confidence score such that the knowledge base is always given priority over the table corpus. If the schema label does not match any predicate from the knowledge base, then we chose the value from the table that is the most relevant to the query. That is, $conf(p, q)$ is based on the corresponding table's relevance score; see Sect. 7.3 for the details. Notice that we pick a single source for each value rather than aggregating evidence from multiple sources. The reason for that is that on the user interface, we would like to display a single traceable source where the given value originates from.

6 EXPERIMENTAL SETUP

Queries, dataset, data preprocessing methods and relevance assessments are introduced in this section.

6.1 Test Queries

We use two sets of queries in our experiments:

QS-1 We consider list type queries from the DBpedia-Entity v2 test collection [12], that is, queries from SemSearch LS, TREC Entity, and QALD2. Out of these, we use the queries that have at least three highly relevant entities in the ground truth. This set contains 119 queries in total.

QS-2 The RELink Query Collection [21] consists of 600 complex entity-relationship queries that are answered by entity tuples. That is, the answer table has two or three columns (including the core entity column) and all cell values are entities. The queries and corresponding relevance judgments in this collection are obtained from Wikipedia lists that contain relational tables. For each answer table, human annotators were asked to formulate the corresponding information need as a natural language query, e.g., "find peaks above 6000m in the mountains of Peru."

For both sets, we remove stop words and perform spell correction.

6.2 Data Sources

We rely on two main data sources simultaneously: a knowledge base and a table corpus.

6.2.1 Knowledge base. The knowledge base we use is DBpedia (version 2015-10). We consider entities for which a short textual description is given in the `dbo:abstract` property (4.6M in total). We limit ourselves to properties that are extracted from Wikipedia infoboxes.

6.2.2 Table corpus. We use the WikiTables corpus [3], which contains 1.65M tables extracted from Wikipedia. The mean number

of rows is 11 and the median is 5. For columns, the mean is 5 and the median is 4. We preprocess tables as follows. For each cell that contains a hyperlink we check if it points to an entity that is present in DBpedia. If yes, we use the DBpedia identifier of the linked entity as the cell's content (with redirects resolved); otherwise, we replace the link with the anchor text (i.e., treat it as a string).

Further, each table is classified as relational or non-relational according to the existence of a core entity column and the size of the table. We set the following conditions for detecting the core column of a table: (i) the core column should contain the most entities compared to other columns; (ii) if there are more than one columns that have the highest number of entities, then the one with lowest index, i.e., the leftmost one, is regarded as the core column; (iii) the core column must contain at least two entities. Tables without a core column or having less than two rows or columns are regarded as non-relational. In the end, we classify the WikiTables corpus into 973,840 relational and 678,931 non-relational tables. Based on a random sample of 100 tables from each category, we find that all the sampled tables are correctly classified.

6.3 Schema Normalization

Different schema labels may be used for expressing the same meaning, e.g., "birthday" vs. "day of birth" or "nation" vs. "country." For the former case, where similar terms are used, we employ a FastJoin match [26] to normalize the strings (with stopwords removed). Specifically, we take the maximum edit distance as in [16] to measure string similarity. When it exceeds a threshold of δ, we regard them as the same label. We set δ as 0.8 which is consistent with [16], where headings are matched for table column join. For the latter case, where different terms are used, we consider predicates connecting the same subject and object as synonyms. These pairs are then checked and erroneous ones are eliminated manually. Whenever schema labels are compared in the paper, we use their normalized versions.

6.4 Relevance Assessments

For QS-1, we consider the highly relevant entities as the ground truth for the core column entity ranking task. For the task of schema determination, we annotated all candidate labels using crowdsourcing. Specifically, we used the CrowdFlower platform and presented annotators with the query, three example core column entities, and a label, and asked them to judge the relevance of that label on a three point scale: highly relevant, relevant, or non-relevant. Each query-entity-label triple was annotated by at least three and at most five annotators. The labelling instructions were as follows: a label is highly relevant if it corresponds to an essential table column for the given query and core column entities; a label is relevant when it corresponds to a property shared by most core column entities and provides useful information, but it is not essential for the given query; a label is non-relevant otherwise (e.g., hard to understand, not informative, not relevant, etc.). We take the majority vote to decide the relevance of a label. Statistically, we have 7000 triples annotated, and on average, there are 4.2 highly relevant labels, 1.9 relevant labels, and 49.4 non-relevant labels for each query. The Fleiss' Kappa test statistics for inter-annotator agreement is 0.61, which is considered as substantial agreement [10]. For the value lookup task, we sampled 25 queries and fetched values from the table corpus and the knowledge base. We again set up a crowdsourcing experiment on CrowdFlower for annotation. Given a query, an entity, a schema label, a value, and a source (Wikipedia or DBpedia page), three to five annotators were asked to validate if the value can be found and whether it is correct, according to the provided source. Overall, 14,219 table cell values were validated. The total expense of the crowdsourcing experiments was $560.

QS-2: Since for this query set we are given the ground truth in a tabular format, based on existing Wikipedia tables, we do not need to perform additional manual annotation. The main entities are taken as the ground truth for the core column entity ranking task, heading labels are taken as the ground truth for the schema determination task, and the table cells (for a sample of 25 queries) are taken as the ground truth for the value lookup task.

6.5 Evaluation Measures

We evaluate core column entity ranking and schema determination in terms of Normalized Discounted Cumulative Gain (NDCG) at cut-off points 5 and 10. The value lookup task is measured by Mean Average Precision (MAP) and Mean Reciprocal Rank (MRR). To test significance, we use a two-tailed paired t-test and write †/‡ to denote significance at the 0.05 and 0.005 levels, respectively.

7 EXPERIMENTAL EVALUATION

We evaluate the three main components of our approach, core column entity ranking, schema determination, and value lookup, and assess the effectiveness of our iterative table generation algorithm.

7.1 Core Column Entity Ranking

We discuss core column entity ranking results in two parts: (i) using only the query as input and (ii) leveraging the table schema as well.

7.1.1 Query-based Entity Ranking. The results are reported in top block of Table 3. The following methods are compared:

- **LM** For term-based matching we use Language Modeling with Dirichlet smoothing, with the smoothing parameter set to 2000, following [12]. This method is also used for obtaining the candidate set (top 100 entities per query) that are re-ranked by the methods below.
- **DRRM_TKS** We train the deep matching model using 5-fold cross-validation. We use a four-layer architecture, with 50 nodes in the input layer, two hidden layers in the feed forward matching networks, and one output layer. The optimizer is ADAM [13], with hinge loss as the loss function. We set the learning rate to 0.0001 and we report the results after 50 iterations.[2] We employ two instantiations of this network, using entity descriptions (e_d) and entity properties (e_p) as input.
- **Combined** We combine the previous three methods, with equal weights, using a linear combination (cf. Eq. 1). Later, in our analysis in Sect. 8.2, we will also experiment with learning the weights for the combination.

On the first query set, QS-1, LM performs best of the single rankers. Combining it with deep features results in 16% and 9% relative improvement for NDCG@5 and NDCG@10, respectively. On QS-2,

[2]We also experimented with C-DSSM and DSSM. However, their overall performance was much lower than that of DRRM_TKS for this task.

Table 3: Core column entity ranking results. The top block of the table uses only the keyword query as input. The bottom block of the table uses the table schema; Round #1–#3 rely on automatically determined schema, while the Oracle method uses the ground truth schema. Statistical significance for query-based entity ranking is compared against LM, for schema-assisted entity ranking is compared against the Combined method.

Method	QS-1		QS-2	
	NDCG@5	NDCG@10	NDCG@5	NDCG@10
Query-based Entity Ranking (Round #0)				
LM	0.2419	0.2591	0.0708	0.0823
DRRM_TKS (e_d)	0.2015	0.2028	0.0501	0.0540
DRRM_TKS (e_p)	0.1780	0.1808	**0.1089**‡	**0.1083**‡
Combined	**0.2821**†	**0.2834**	0.0852‡	0.0920†
Schema-assisted Entity Ranking				
Round #1	0.3012	0.2892	0.1232‡	0.1201‡
Round #2	0.3369‡	0.3221‡	0.1307‡	0.1264‡
Round #3	0.3445‡	0.3250‡	0.1345‡	0.1270‡
Oracle	**0.3518**‡	**0.3355**‡	**0.1587**‡	**0.1555**‡

a slightly different picture emerges. The best individual ranker is DRRM_TKS using entity properties. Nevertheless, the Combined method still improves significantly over the LM baseline.

7.1.2 Schema-assisted Entity Ranking. Next, we also consider the table schema for core column entity ranking. The results are presented in the bottom block of Table 3. Note that on top of to the three features we have used before, we have four additional features (cf. Table 1). As before, we use uniform weight for all features. We report results for three additional iterations, Rounds #1–#3, where the schema is taken from the previous iteration of the schema determination component. Further, we report on an Oracle method, which uses the ground truth schema. In all cases, we take the top 10 schema labels ($k = 10$); we analyze the effect of using different k values in Sect. 8.1. These methods are to be compared against the Combined method, which corresponds to Round #0. We find that our iterative algorithm is able to gradually improve results, in each iteration, for both of the query sets and evaluation metrics; with the exception of QS-1 in Round #1, all improvements are highly significant. Notice that the relative improvement made between Round #0 and Round #3 is substantial: 22% and 86% in terms of NDCG@5 for QS-1 and QS-2, respectively.

7.2 Schema Determination

Schema determination results are presented in two parts: (i) using only the query as input and (ii) also leveraging core column entities.

7.2.1 Query-based Schema Determination. In the top block of Table 4 we compare the following three methods:

CP We employ the column population method from [35] to determine the top 100 labels for each query. Following [16], the γ parameter for the edit distance threshold is set to 0.8. This method is also used for obtaining the candidate label set (top 100 per query) that is re-ranked by the methods below.

Table 4: Schema determination results. The top block of the table uses only the keyword query as input. The bottom block of the table uses the core column entities as well; Round #1–#3 rely on automatic entity ranking, while the Oracle method uses the ground truth entities. Statistical significance for query-based schema determination is compared against CP, for entity-assisted entity ranking is compared against the Combined method.

Method	QS-1		QS-2	
	NDCG@5	NDCG@10	NDCG@5	NDCG@10
Query-based Entity Ranking (Round #0)				
CP	0.0561	0.0675	0.1770	0.2092
DRRM_TKS	0.0380	0.0427	0.0920	0.1415
Combined	**0.0786**†	**0.0878**†	**0.2310**‡	**0.2695**‡
Entity-assisted Schema Determination				
Round #1	0.1676‡	0.1869‡	0.3342‡	0.3845‡
Round #2	0.1775‡	0.2046‡	0.3614‡	0.4143‡
Round #3	0.1910‡	0.2136‡	0.3683‡	0.4350‡
Oracle	**0.2002**‡	**0.2434**‡	**0.4239**‡	**0.4825**‡

DRRM_TKS We use the same neural network architecture as for core column entity ranking. For training the network, we make use of Wikipedia tables. If an entity and a schema label co-occur in an existing Wikipedia table, then we consider it as a positive pair. Negative training instances are generated by sampling, for each entity, a set of schema labels that do not co-occur with that entity in any existing table. In the end, we generate a total of 10.7M training examples, split evenly between the positive and negative classes.

Combined We combine the above two methods in a linear fashion, with equal weights (cf. Eq. 2). Later, in our analysis in Sect. 8.2, we will also experiment with learning the weights for the combination.

We find that the CP performs better than DRRM_TKS, especially on the QS-2 query set. The Combined method substantially and significantly outperforms both of them, with a relative improvement of 40% and 30% over CP in terms of NDCG@5 on QS-1 and QS-2, respectively.

7.2.2 Entity-assisted Schema Determination. Next, we incorporate three additional features that make use of core column entities (cf. Table 2), using uniform feature weights. For the attribute retrieval feature (§4.2.2), we rely on the Google Custom Search API to get search hits and use the same parameter setting (feature weights) as in [14]. For all features, we use the top 10 ranked entities (and analyze different k values later, in Sect. 8.1).

The results are shown in the bottom block of Table 4. Already Round #1 shows a significant jump in performance compared to the Combined method (corresponding to Round #0). Subsequent iterations results in further improvements, reaching a relative improvement of 243% and 159% for Round #3 in terms of NDCG@5 for QS-1 and QS-2, respectively. Judging from the performance of the Oracle method, there is further potential for improvement, especially for QS-2.

Table 5: Value lookup results.

Source	QS-1 MAP	QS-1 MRR	QS-2 MAP	QS-2 MRR
KB	0.7759	0.7990	0.0745	0.0745
TC	0.1614	0.1746	0.9564	0.9564
KB+TC	0.9270	0.9427	0.9564	0.9564

7.3 Value Lookup

For value lookup evaluation we take the core column entities and schema labels from the ground truth. This is to ensure that this component is evaluated on its own merit, without being negatively influenced by errors that incur earlier in the processing pipeline. In our evaluation, we ignore cells that have empty values according to the ground truth (approximately 12% of the cells have empty values in the Wikitables corpus). The overall evaluation results are reported in Table 5. We rely on two sources for value lookup, the knowledge base (KB) and the table corpus (TC). Overall, we reach excellent performance on both query sets. On QS-1, the knowledge base is the primary source, but the table corpus also contributes new values. On QS-2, since all values originate from existing Wikipedia tables, using the knowledge base does not bring additional benefits. This, however, is the peculiarity of that particular dataset. Also, according to the ground truth there is a single correct value for each cell, hence the MAP and MRR scores are the same for QS-2.

8 ANALYSIS

In this section, we conduct further analysis to provide insights on our iterative algorithm and on feature importance.

8.1 Iterative Algorithm

We start our discussion with Fig. 5, which displays the overall effectiveness of our iterative algorithm on both tasks. Indeed, as it is clearly shown by these plots, our algorithm performs well. The improvements are the most pronounced when going from Round #0 to Round #1. Performance continues to rise with later iterations, but, as it can be expected, the level of improvement decreases over time. The rightmost bars in the plots correspond to the Oracle method, which represents the upper limit that could be achieved, given a perfect schema determination method for core column entity ranking and vice versa. We can observe that for core column entity ranking on QS-1 (Fig. 5a), has already reached this upper performance limit at iteration #3. For the other task/query set combinations there remains some performance to be gained. It is left for future work to devise a mechanism for determining the number of iterations needed.

Next, we assess the impact of the number of feedback items leveraged, that is, the value of k when using the top-k schema labels in core column entity ranking and top-k entities in schema determination. Figure 6 shows how performance changes with different k values. For brevity, we report only on NDCG@10 and note that a similar trend was observed for NDCG@5. We find that the differences between the different k values are generally small, with $k = 10$ being a good overall choice across the board.

To further analyze how individual queries are affected over iterations, Table 6 reports the number of queries that are helped (↑), hurt (↓), and remained unchanged (–). We define change as a difference of ≥0.05 in terms of NDCG@10. We observe that with the exception of schema determination on QS-2, the number of queries hurt always decreases between successive iterations. Further, the number of queries helped always increases from Round #1 to #3.

Table 6: The number queries helped (ΔNDCG@10≥0.05), hurt (ΔNDCG@10≤-0.05), and unchanged (remaining) for core column entity ranking (CCER) and schema determination (SD).

	CCER			SD		
QS-1	↑	↓	–	↑	↓	–
Round #0 vs. #1	43	38	38	52	7	60
Round #0 vs. #2	50	30	39	61	5	53
Round #0 vs. #3	49	26	44	59	2	58
QS-2	↑	↓	–	↑	↓	–
Round #0 vs. #1	166	82	346	386	56	158
Round #0 vs. #2	173	74	347	388	86	126
Round #0 vs. #3	173	72	349	403	103	94

Lastly, we demonstrate how results change over the course of iterations, we show one specific example table in Fig. 4 that is generated in response to the query "Towns in the Republic of Ireland in 2006 Census Records."

8.2 Parameter Learning

For simplicity, we have so far used all features with equal weights for core column entity ranking (cf. Eq. 1) and schema determination (cf. Eq. 2). Here, we aim to learn the feature weights from training data. In Tables 7 and 8 we report results with weights learned using five-fold cross-validation. These results are to be compared against the uniform weight settings in Tables 3 and 4, respectively. We notice that on QS-1, most evaluation scores are lower with learned weights than with uniform weights, for both core column entity ranking and schema determination. This is due to the fact that queries in this set are very heterogeneous [12], which makes it difficult to learn weights that perform well across the whole set. On QS-2, according to expectations, learning the weights can yield

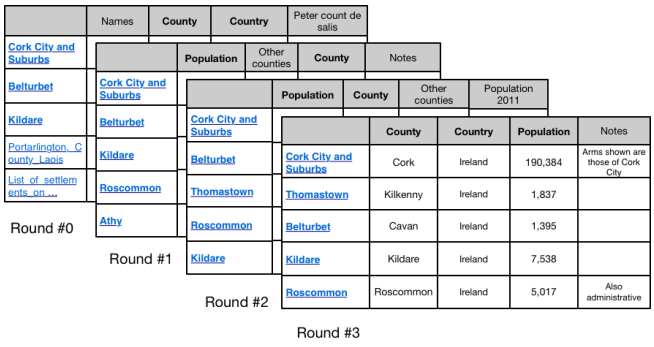
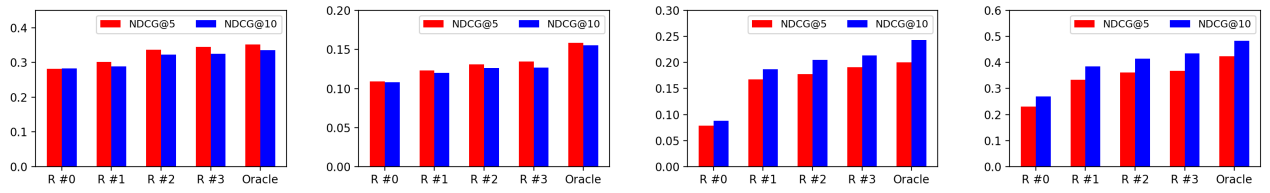
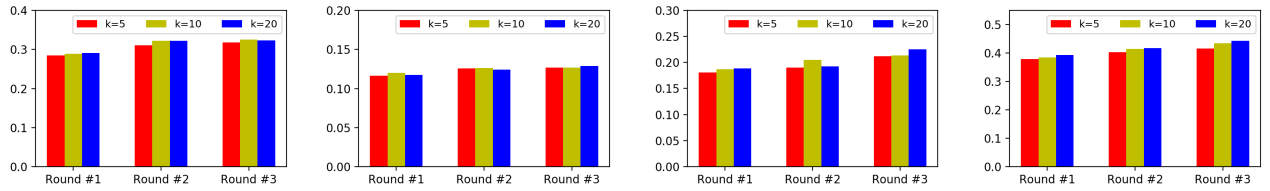


Figure 4: Generated table in response to the query "Towns in the Republic of Ireland in 2006 Census Records." Relevant entities and schema labels are boldfaced.



(a) CCER QS-1 (b) CCER QS-2 (c) SD QS-1 (d) SD QS-2

Figure 5: Performance change across iterations for core column entity ranking (CCER) and schema determination (SD).



(a) CCER QS-1 (b) CCER QS-2 (c) SD QS-1 (d) SD QS-2

Figure 6: Impact of the cutoff parameter k for Core Column Entity Ranking (CCER) and Schema Determination (SD).

Table 7: Core column entity retrieval results with parameters learned using five-fold cross-validation. In parentheses are the relative improvements w.r.t. using uniform weights.

Method	QS-1		QS-2	
	NDCG@5	NDCG@10	NDCG@5	NDCG@10
Round #0	0.2523 (-11%)	0.2653 (-6%)	0.1003 (+18%)	0.1048 (+14%)
Round #1	0.2782 (-8%)	0.2772 (-4%)	0.1308 (+6%)	0.1252 (+4%)
Round #2	0.3179 (-6%)	0.3180 (-1%)	0.1367 (+5%)	0.1323 (+5%)
Round #3	0.3192 (-7%)	0.3109 (-4%)	0.1395 (+4%)	0.1339 (+5%)
Oracle	0.3017 (-14%)	0.3042 (-9%)	0.1728 (+9%)	0.1630 (+5%)

Table 8: Schema determination results with parameters learned using five-fold cross-validation. In parentheses are the relative improvements w.r.t. using uniform weights.

Method	QS-1		QS-2	
	NDCG@5	NDCG@10	NDCG@5	NDCG@10
Round #0	0.0928 (+18%)	0.1064 (+21%)	0.2326 (+1%)	0.2710 (+1%)
Round #1	0.1663 (-1%)	0.2066 (+11%)	0.3865 (+16%)	0.4638 (+12%)
Round #2	0.1693 (-5%)	0.2212 (+8%)	0.3889 (+8%)	0.4599 (+11%)
Round #3	0.1713 (-10%)	0.2321 (+9%)	0.3915 (+6%)	0.4620 (+6%)
Oracle	0.1719 (-14%)	0.2324 (-5%)	0.4678 (+10%)	0.5307 (+10%)

up to 18% and 21% relative improvement for core column entity ranking and schema determination, respectively.

8.3 Feature Importance

To measure the importance of individual features, we use their average learned weights (linear regression coefficients) across all iterations. The ordering of features for core column entity ranking and QS-1 is: $\phi_1(0.566) > \phi_7(0.305) > \phi_6(0.244) > \phi_2(0.198) > \phi_5(0.127) > \phi_4(0.09) > \phi_3(0.0066)$. For QS-2 it is: $\phi_7(0.298) > \phi_1(0.148) > \phi_3(0.108) > \phi_4(0.085) > \phi_5(0.029) > \phi_2(-0.118) > \phi_6(-0.128)$. Overall, we find the term-based matching (Language Modeling) score (ϕ_1) and our novel entity-schema compatibility score (ϕ_7) to be the most important features for core column entity ranking. Turning to schema determination, on QS-1 the ordering is: $\phi_5(0.23) > \phi_3(0.076) > \phi_1(-0.035) > \phi_2(-0.072) > \phi_4(-0.129)$. For QS-2 it is: $\phi_5(0.27) > \phi_4(0.181) > \phi_1(0.113) > \phi_3(0.018) > \phi_2(-0.083)$. Here, entity-schema compatibility (ϕ_5) is the single most important feature on both query sets.

9 RELATED WORK

Research on web tables has drawn increasing research attention. We focus on three main related areas: table search, table augmentation, and table mining.

Table search refers to the task of returning a ranked list of tables (or tabular data) for a query. Based on the query type, table search can be categorized as keyword-based search [4, 5, 19, 20, 25] or table-based search [1, 7, 16, 17, 19, 30]. Zhang and Balog [36] propose a set of semantic features and fusion-based similarity measures [34] for table retrieval with respect to a keyword query. Focusing on result diversity, Nguyen et al. [19] design a goodness measure for table search and selection. There are some existing table search engines, e.g., Google Fusion Tables [4]. Table search is often regarded as a fundamental step in other table related tasks. For example, Das Sarma et al. [7] take an input table to search row or column complement tables whose elements can be used for augmenting a table with additional rows or columns.

Table augmentation is about supplementing a table with additional elements, e.g., new columns [2, 4, 7, 16, 30, 33]. Zhang and Balog [35] propose the tasks of row and column population, to augment the core column entity set and column heading labels. They capture relevant data from DBpedia and the WikiTables corpus. Search based on attributes, entities and classes is defined as *relational search*, which can be used for table column augmentation. Kopliku et al. [14] propose a framework to extract and rank attributes from web tables. *Data completion* refers to the problem of filling in empty table cells. Yakout et al. [30] address three core

tasks: augmentation by attribute name, augmentation by example, and attribute discovery by searching similar tables. Each of these tasks is about extracting table cell data from existing tables. In case that no existing values are captured, Ahmadov et al. [1] introduce a method to extract table values from related tables and/or to predict them using machine learning methods.

Table mining is to explore and utilize the knowledge contained in tables [3, 5, 22, 25, 32]. Munoz et al. [18] recover Wikipedia table semantics and store them as RDF triples. A similar approach is taken in [5] based on tables extracted from a Google crawl. Instead of mining the entire table corpus, a single table stores many facts, which could be answers to questions. Given a query, Sun et al. [24] identify possible entities using an entity linking method and represent them as a two-node graph question chain, where each node is an entity. Table cells of the KB table are decomposed into relational chains, which are also two-node graphs connecting two entities. The task then boils downing to matching question and table cell graphs using a deep matching model. A similar task is addressed by Yin et al. [32] using a full neural network. Information extracted from tables can be used to augment existing knowledge bases [8, 23]. Another line of work concerns table annotation and classification. By mining column content, Zwicklbauer et al. [37] propose a method to annotate table headers. Studying a large number of tables in [6], a fine-grained table type taxonomy is provided for classifying web tables.

10 CONCLUSION

We have introduced the task of on-the-fly table generation, which aims to answer queries by automatically compiling a relational table in response to a query. This problem is decomposed into three specific subtasks: (i) core column entity ranking, (ii) schema determination, and (iii) value lookup. We have employed a feature-based approach for core column entity ranking and schema determination, combining deep semantic features with task-specific signals. We have further shown that these two subtasks are not independent of each other and have developed an iterative algorithm, in which the two reinforce each other. For value lookup, we have entity-oriented fact catalog, which allows for fast and effective lookup from multiple sources. Using two sets of entity-oriented queries, we have demonstrated the effectiveness of our method. In future work, we wish to consider more heterogeneous table corpus in addition to Wikipedia tables, i.e., arbitrary tables from the Web.

REFERENCES

[1] Ahmad Ahmadov, Maik Thiele, Julian Eberius, Wolfgang Lehner, and Robert Wrembel. 2015. Towards a Hybrid Imputation Approach Using Web Tables.. In *Proc. of BDC '15*. 21–30.

[2] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. 2013. Methods for Exploring and Mining Tables on Wikipedia. In *Proc. of IDEA '13*. 18–26.

[3] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. 2015. TabEL: Entity Linking in Web Tables. In *Proc. of ISWC 2015*. 425–441.

[4] Michael J. Cafarella, Alon Halevy, and Nodira Khoussainova. 2009. Data Integration for the Relational Web. *Proc. of VLDB Endow.* 2 (2009), 1090–1101.

[5] Michael J. Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. 2008. WebTables: Exploring the Power of Tables on the Web. *Proc. of VLDB Endow.* 1 (2008), 538–549.

[6] Eric Crestan and Patrick Pantel. 2011. Web-scale Table Census and Classification. In *Proc. of WSDM '11*. 545–554.

[7] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. 2012. Finding Related Tables. In *Proc. of SIGMOD '12*. 817–828.

[8] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. 2014. Knowledge Vault: A Web-scale Approach to Probabilistic Knowledge Fusion. In *Proc. of KDD '14*. 601–610.

[9] Yixing Fan, Liang Pang, JianPeng Hou, Jiafeng Guo, Yanyan Lan, and Xueqi Cheng. 2017. MatchZoo: A Toolkit for Deep Text Matching. *arXiv preprint arXiv:1707.07270* (2017).

[10] J.L. Fleiss et al. 1971. Measuring nominal scale agreement among many raters. *Psychological Bulletin* 76 (1971), 378–382.

[11] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. 2016. A Deep Relevance Matching Model for Ad-hoc Retrieval. In *Proc. of CIKM '16*. 55–64.

[12] Faegheh Hasibi, Fedor Nikolaev, Chenyan Xiong, Krisztian Balog, Svein Erik Bratsberg, Alexander Kotov, and Jamie Callan. 2017. DBpedia-Entity V2: A Test Collection for Entity Search. In *Proc. of SIGIR '17*. 1265–1268.

[13] Diederik P. Kingma and Jimmy Ba. 2014. *Adam: A Method for Stochastic Optimization*. http://arxiv.org/abs/1412.6980.

[14] Arlind Kopliku, Mohand Boughanem, and Karen Pinel-Sauvagnat. 2011. Towards a Framework for Attribute Retrieval. In *Proc. of CIKM '11*. 515–524.

[15] Oliver Lehmberg, Dominique Ritze, Robert Meusel, and Christian Bizer. 2016. A Large Public Corpus of Web Tables Containing Time and Context Metadata. In *Proc. of WWW '16 Companion*. 75–76.

[16] Oliver Lehmberg, Dominique Ritze, Petar Ristoski, Robert Meusel, Heiko Paulheim, and Christian Bizer. 2015. The Mannheim Search Join Engine. *Web Semant.* 35 (2015), 159–166.

[17] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. 2010. Annotating and Searching Web Tables Using Entities, Types and Relationships. *Proc. of VLDB Endow.* 3 (2010), 1338–1347.

[18] Emir Munoz, Aidan Hogan, and Alessandra Mileo. 2014. Using Linked Data to Mine RDF from Wikipedia's Tables. In *Proc. of WSDM '14*. 533–542.

[19] Thanh Tam Nguyen, Quoc Viet Hung Nguyen, Weidlich Matthias, and Aberer Karl. 2015. Result Selection and Summarization for Web Table Search. In *ISDE '15*. 425–441.

[20] Rakesh Pimplikar and Sunita Sarawagi. 2012. Answering Table Queries on the Web Using Column Keywords. *Proc. of VLDB Endow.* 5 (2012), 908–919.

[21] Pedro Saleiro, Natasa Milic-Frayling, Eduarda Mendes Rodrigues, and Carlos Soares. 2017. RELink: A Research Framework and Test Collection for Entity-Relationship Retrieval. In *Proc. of SIGIR '17*. 1273–1276.

[22] Sunita Sarawagi and Soumen Chakrabarti. 2014. Open-domain Quantity Queries on Web Tables: Annotation, Response, and Consensus Models. In *Proc. of KDD '14*. 711–720.

[23] Yoones A. Sekhavat, Francesco Di Paolo, Denilson Barbosa, and Paolo Merialdo. 2014. Knowledge Base Augmentation using Tabular Data. In *Proc. of LDOW '14*.

[24] Huan Sun, Hao Ma, Xiaodong He, Wen-tau Yih, Yu Su, and Xifeng Yan. 2016. Table Cell Search for Question Answering. In *Proc. of WWW '16*. 771–782.

[25] Petros Venetis, Alon Halevy, Jayant Madhavan, Marius Paşca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. 2011. Recovering Semantics of Tables on the Web. *Proc. of VLDB Endow.* 4 (2011), 528–538.

[26] Jiannan Wang, Guoliang Li, and Jianhua Feng. 2014. Extending String Similarity Join to Tolerant Fuzzy Token Matching. *ACM Trans. Database Syst.* 39, 1 (2014), 1–45.

[27] Yalin Wang and Jianying Hu. 2002. Detecting Tables in HTML Documents. In *Proc. of DAS '02*. 249–260.

[28] Yalin Wang and Jianying Hu. 2002. A Machine Learning Based Approach for Table Detection on the Web. In *Proc. of WWW '02*. 242–250.

[29] Mohamed Yahya, Klaus Berberich, Shady Elbassuoni, Maya Ramanath, Volker Tresp, and Gerhard Weikum. 2012. Natural Language Questions for the Web of Data. In *Proc. of EMNLP-CoNLL '12*. 379–390.

[30] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. 2012. InfoGather: Entity Augmentation and Attribute Discovery by Holistic Matching with Web Tables. In *Proc. of SIGMOD '12*. 97–108.

[31] Mohan Yang, Bolin Ding, Surajit Chaudhuri, and Kaushik Chakrabarti. 2014. Finding Patterns in a Knowledge Base Using Keywords to Compose Table Answers. *Proc. VLDB Endow.* 7, 14 (2014), 1809–1820.

[32] Pengcheng Yin, Zhengdong Lu, Hang Li, and Ben Kao. 2016. Neural Enquirer: Learning to Query Tables in Natural Language. In *Proc. of IJCAI '16*. 2308–2314.

[33] Shuo Zhang, Vugar Abdulzada, and Krisztian Balog. 2018. SmartTable: A Spreadsheet Program with Intelligent Assistance. In *Proc. of SIGIR '18*.

[34] Shuo Zhang and Krisztian Balog. 2017. Design Patterns for Fusion-Based Object Retrieval. In *Proc. of ECIR '17*. 684–690.

[35] Shuo Zhang and Krisztian Balog. 2017. EntiTables: Smart Assistance for Entity-Focused Tables. In *Proc. of SIGIR '17*. 255–264.

[36] Shuo Zhang and Krisztian Balog. 2018. Ad Hoc Table Retrieval using Semantic Similarity. In *Proc. of WWW '18*. 1553–1562.

[37] Stefan Zwicklbauer, Christoph Einsiedler, Michael Granitzer, and Christin Seifert. 2013. Towards Disambiguating Web Tables. In *Proc. of ISWC-PD '13*. 205–208.