Kai Herrmann, Hannes Voigt, Wolfgang Lehner

Cinderella — Adaptive online partitioning of irregularly structured data

Erstveröffentlichung in / First published in:

International Conference on Data Engineering Workshops. Chicago, 31.03. – 04.04.2014. IEEE Xplore, S. 284 - 291. ISBN 978-1-4799-3481-2.

DOI: https://doi.org/10.1109/ICDEW.2014.6818342

Final edited form was published in "International Conference on Data Engineering Workshops. Chicago 2014", S. 284 - 291. ISBN: 978-1-4799-3481-2
https://doi.org/10.1109/ICDEW.2014.6818342

Cinderella – Adaptive Online Partitioning of Irregularly Structured Data

Kai Herrmann, Hannes Voigt, Wolfgang Lehner

Database Technology Group
Technische Universität Dresden
01062 Dresden, Germany
{firstname.lastname}@tu-dresden.de

***Abstract*—In an increasing number of use cases, databases face the challenge of managing irregularly structured data. Irregularly structured data is characterized by a quickly evolving variety of entities without a common set of attributes. These entities do not show enough regularity to be captured in a traditional database schema. A common solution is to centralize the diverse entities in a universal table. Usually, this leads to a very sparse table. Although today's techniques allow efficient storage of sparse universal tables, query efficiency is still a problem. Queries that reference only a subset of attributes have to read the whole universal table including many irrelevant entities. One possible solution is to use a partitioning of the table, which allows pruning partitions of irrelevant entities before they are touched. Creating and maintaining such a partitioning manually is very laborious or even infeasible, due to the enormous complexity. Thus an autonomous solution is desirable.**

In this paper, we define the Online Partitioning Problem for irregularly structured data and present Cinderella. Cinderella is an autonomous online algorithm for horizontal partitioning of irregularly structured entities in universal tables. It is designed to keep its overhead low by incrementally assigning entities to partitions while they are touched anyway during modifications. The achieved partitioning allows queries that retrieve only entities with a subset of attributes easily pruning partitions of irrelevant entities. Cinderella increases the locality of queries and reduces query execution cost.

I. Introduction

As ubiquitous technological means for managing data, databases are exposed to the persistent acceleration of society and technological development. Traditionally modeled database schemas become quickly outdated by reality and application requirements. Where databases should comprehensively capture a large and quickly evolving variety of entities, the traditional database design principles are stretched to their limits. Examples for such application areas are product catalogs (e.g., for electronic devices), clinical findings in patient databases, multi-tenancy databases, or analytic databases, which are constantly enhanced with derived data. Here, new entities frequently appear with new combinations of attributes or totally new attributes. Typically, entities show some regularity but not enough to allow modeling a sound database schema.

A common solution in such areas is to model on a more abstract level and centralize the diverse entities in a universal table. Instead of a table for each type of product in a traditional product catalog, the universal table approach has a single product table containing all product properties. Usually, this leads to a very sparse table, which most of today's database systems can store efficiently [1], [2], [3]. Retrieval operations, however, suffer from the universal table modeling. Queries that reference only a subset of attributes have to read over the whole universal table including many irrelevant entities that do not have the referenced attributes. Horizontal partitioning presents a simple technique to increase the efficiency of such queries. A partitioning scheme taking into account the irregularity of the entities allows queries pruning partitions of irrelevant entities before they touch the data. Designing and maintaining such a partitioning, though, is a laborious and never ending task most DBAs are not willing to commit to.

In this article, we define the Online Partitioning Problem for irregularly structured data. To solve the problem, a technique with little overhead is a necessity. As the main contribution of the article, we present Cinderella, an autonomous online algorithm for horizontal partitioning of irregularly structured entities in universal tables. Cinderella partitions entities into homogeneous fix-sized partitions, such that the entities in a partition share most of their attributes. Cinderella maintains the partitioning while entities are added, modified, and removed. It is designed to keep overhead low by operating online; it incrementally assigns entities to partitions while they are touched anyway during modifications. Queries that retrieve only entities with a subset of attributes can easily prune partitions which contain entities with only irrelevant attributes. This way, Cinderella increases the locality of queries and reduces query execution cost.

In the remainder of the paper, we first define the Online Partitioning Problem in Section II. Afterwards, Sections III and Section IV present Cinderella followed by an evaluation in Section V. Finally, Sections VI and VII briefly discuss related work and conclude the paper, respectively.

II. Online Partitioning

Entities in a universal table are typically very heterogeneous regarding their attribute sets. Together, the entities feature a large number of attributes while most of the entities instantiate only a small subset of these attributes. Figure 1 illustrates such a universal table for the product catalog scenario. As can be seen, attributes appear irregularly among the entities. Some attributes are very common, e.g. *name* or *weight*, while others

Final edited form was published in "International Conference on Data Engineering Workshops. Chicago 2014", S. 284 - 291. ISBN: 978-1-4799-3481-2
https://doi.org/10.1109/ICDEW.2014.6818342

name	resolution	aperture	screen	storage	tuner	rotation	form factor	weight	...
Canon PowerShot S120	12.1	2.0	3					198	...
Sony SLT-A99	24		3					733	...
Samsung Galaxy S4	13		4.3	32GB				133	...
Apple iPod touch	5		4	64GB				88	...
LG 60LA7408	Full HD		40		DVB-T/C/S			9800	...
WD4000FYYZ				4TB		7200	3.5”		...
Garmin Dakota 20			2.6					150	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

Fig. 1. Example of a universal table for electronic devices.

are specific to only certain kinds of entities, e.g. *aperture* for cameras with built-in lenses. Studies observed that distribution of attributes obeys the Zipf's law [4], [5]. Although the entities exhibit some regularity, it is hard to find reasonably stable and reliable partitioning scheme. While all of today's cameras feature a sensor, a screen, and a storage card slot, some of them are also equipped with a flash, a GPS sensor, or Wi-Fi. Soon, we will see cameras with mobile connectivity but lacking a storage card slot. If there would be a stable partitioning schema, after all, it is reasonable to assume that the database modeler would have modeled the database accordingly.

The goal of the partitioning is to increase the query efficiency by allowing the database system the early pruning of partitions with entities irrelevant for a query. Each partition is described in the system catalog using a partition synopsis p, which lists the attributes of the entities in the partition. Likewise, we can list all attributes relevant to a query in a query synopsis q. Based on the synopses, queries can easily prune partitions that contain only entities irrelevant to the query, i.e., partition for which $|p \wedge q| = 0$ holds. Correspondingly, the efficiency of a given partitioning is the ratio of how much data is relevant to a workload and how much data is actually read.

Definition 1 (Partitioning Efficiency): Given a universal table T containing the entities $\{e_1, e_2, \ldots\}$, a query set $W = \{q_1, q_2, \ldots\}$, and a partitioning $P = \{p_1, p_2, \ldots\}$, the efficiency of P is

$$\textsc{Efficiency}(P) = \frac{\sum_{q \in W, e \in T} \text{sgn}(|e \wedge q|) \cdot \textsc{Size}(e)}{\sum_{q \in W, p \in P} \text{sgn}(|p \wedge q|) \cdot \textsc{Size}(p)} \quad .$$

The function $\textsc{Size}()$ yields the size of an entity or a partition, indicating how much has to be read to scan the entity or all entities in a partition, respectively. The function $\text{sgn}(|e \wedge q|)$ results in one if entity e is relevant to query q and zero if not. Likewise, $\text{sgn}(|p \wedge q|)$ results in one if partition p contains a relevant entity and zero if not. The $\textsc{Efficiency}(P)$ returns a value between zero and one representing the fraction of accessed data which is actually relevant for the query set W.

The aim of online partitioning of a universal table is to continuously maximize the efficiency of a given partitioning under the presence of modification operations. Modification operations are inserts, updates, and deletes that change the set of entities or manipulate the attribute sets of the entities.

Definition 2 (Online Partitioning Problem): Given a universal table T, a query set W, a partitioning P, and a modification m, online partitioning updates P so that $\textsc{Efficiency}(P)$ is maximized for W after m is applied to T.

The online partitioning problem can be solved based on the workload or solely on the entities. A workload-based solution tries to find a partitioning so that, ideally, entities in the same partition are relevant to the same set of queries. Hence, the resulting partitioning is tailored for the given workload. Whenever a workload is not available or where the solution should be more general and robust, an entity-based solution is more appropriate. An entity-based solution favors partitions that contain entities with attribute sets as similar as possible. The resulting partitioning is independent from a particular workload.

The online partitioning problem applies to many different database architectures and to various levels in an architecture. Most obviously in distributed databases or distributed file systems, partitions are distributed among the nodes. In modern main-memory database systems running on a large shared-memory NUMA system, partitions resemble the local memory of each CPU core. In traditional disk-based systems, pages may represent a partition granularity where solving the online partitioning problem can help to increase the query efficiency on universal tables.

III. Cinderella

Cinderella works incrementally. It relies on the basic assumption that the data is already well partitioned. Triggered by a modification operation, Cinderella merely adjusts the partitioning so that the modified entity fits in well. Cinderella creates partitions of a fixed maximum size. Partitions that reach their capacity limit are reorganized with a split operation. Since, among the common data modification operations, inserts affect the partitioning most, we will focus the discussion of Cinderella on the insert operation. Cinderella can create a workload-based or an entity-based partitioning. For a workload-based partitioning, an entity synopsis lists the queries an entity is relevant to, while for a workload-based partitioning, an entity synopsis lists the attributes an entity instantiates. For simplicity in the discussion, we will assume the entity-based setup.

The basic insert procedure is illustrated in Figure 2. Given two partitions cataloged with their synopses and a new entity,

Final edited form was published in "International Conference on Data Engineering Workshops. Chicago 2014", S. 284 - 291. ISBN: 978-1-4799-3481-2
https://doi.org/10.1109/ICDEW.2014.6818342

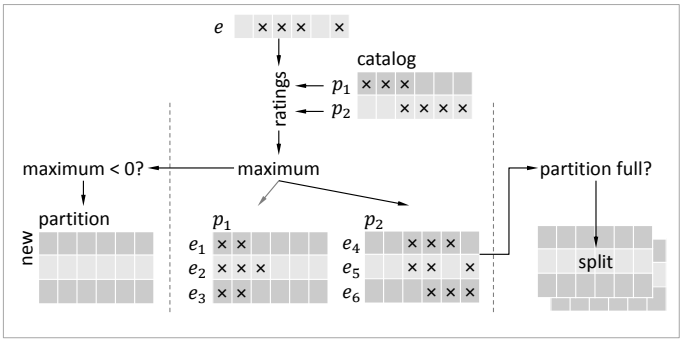


Fig. 2. Insert procedure.

Cinderella scans the partition catalog to find the partition which fits best to the new entity. Every partition is rated and the entity is inserted to the partition with the highest rating. We will discuss the rating in detail in Section IV. There are two possible exceptions from this basic procedure, illustrated in the left and in the right of the Figure 2. First, the rating can become negative indicating that the new entity fits none of the existing partitions well. In this case, Cinderella creates a new partition for the new entity. Second, the highest rated partition has reached the maximum capacity B. Here, Cinderella splits the partition into two new partitions.

For the split, the insert procedure maintains a pair of so-called split starters for each partition in the system catalog. The split starters are two entities from a given partition that differ as much as possible in their synopses. The first two entities added to a partition form the initial pair of split starters. With every additional entity added, the insert procedure checks whether this entity would make a better, i.e. more differential, starters pair with one of the original starter entities. If that is the case the insert procedure updates the partition's pair of split starters accordingly. The difference between entity synopses e_1 and e_2 is calculated as the number of different schema properties $|e_1 \oplus e_2|$. This incremental maintenance heuristically tries to maximize the difference of the split starters. It does not guarantee to yield the most differential pair of entities in a partition, but it avoids the cubic effort necessary to determine the most differential pair.

To split a partition, the insert procedure creates two new partitions and moves each of the two split starter entities to one of the new partitions. The remaining entities are assigned to the new partitions using the insert procedure itself, while limiting the set of possible target partitions to the two new partitions. The procedure does not necessarily result in a balanced split; the result strictly depends on the schema properties of the involved entities. The recursive use of the insert procedure can result in a split cascade. Neither will such a split cascade be very long nor is it a very likely event.

Algorithm 1 lists the complete insert routine. First, Cinderella iterates over the partition catalog to find the partition best rated for the new entity (Line 3–7). Then, depending on the best rating, the algorithm either creates a new partition for the entity (Line 9–13), splits the best rated partition (Line 26–33), or simply inserts the entity into the best rated partition (Line 36). In the case of a split or a normal insert, Cinderella updates the pair of split starters to reflect the new entity (Line 15–24). Specifically, the new entity will replace one of the split starters if it has a difference to the other split starter larger than the difference between the original split starters.

Algorithm 1 Online horizontal partitioning.

```
1:  procedure InsertEntity(e, s_e, C)                    ▷ e: entity
2:                ▷ s_e: entity synopsis; C: catalog with partition synopses
3:      r_best ← −∞                                      ▷ init best rating
4:      for (s_p, p) ∈ C do                              ▷ scan partition synopses in catalog
5:          r ← Rate(s_e, s_p)                           ▷ calculate rating
6:          if r_best < r then                           ▷ if is best rating so far
7:              r_best ← r;   p_best ← p                 ▷ save current best
8:      ▷ if best rating is negative
9:      if r_best < 0 then
10:         p_best ← CreateNewPartition()
11:         p_best.Add(e)                                ▷ add entity to new partition
12:         p_best.e_A ← e                               ▷ set entity as first split starter
13:         return
14:     ▷ update split starters of partition r_best
15:     if p_best.e_B = Null then                        ▷ if second split starter is missing
16:         p_best.e_B ← e                               ▷ set entity as second split starter
17:     else                                             ▷ check if new entity is a better split starter
18:         r_eA ← Diff(e, p_best.e_A)
19:         r_eB ← Diff(e, p_best.e_B)
20:         r_AB ← Diff(p_best.e_A, p_best.e_B)
21:         if r_eA = Max(r_eA, r_eB, r_AB) then
22:             p_best.e_B ← e                           ▷ e becomes split starter B
23:         else if r_eB = Max(r_eA, r_eB, r_AB) then
24:             p_best.e_A ← e                           ▷ e becomes split starter A
25:     ▷ if partition is full, split
26:     if Size(p_best) + Size(e) > MaxSize then
27:         p_A ← CreateNewPartition()
28:         p_B ← CreateNewPartition()
29:         p_A.Add(p_best.e_A);   p_best.Remove(p_best.e_A)
30:         p_B.Add(p_best.e_B);   p_best.Remove(p_best.e_B)
31:         for e_split ∈ p_best do                      ▷ split remaining entities
32:             InsertEntity(e_split, s_e_split, {p_A, p_B})
33:             p_best.Remove(e_split)
34:         return
35:     ▷ normal case: just add entity to partition r_best
36:     p_best.Add(e)
```

As can be seen, the complexity of finding the best rated partition depends on the number of partitions and the cardinality of the synopses. For a split, the complexity depends on the partition size and, again, the cardinality of the synopses. However, in I/O bound systems the performance will be dominated by the moving of the actual entities from partition to partition. Consequently, the split, although linear in complexity, is the most expensive part of the algorithm.

The adjustment routines that Cinderella performs for the other modification operations rely on the insert routine. Upon

Final edited form was published in "International Conference on Data Engineering Workshops. Chicago 2014", S. 284 - 291. ISBN: 978-1-4799-3481-2
https://doi.org/10.1109/ICDEW.2014.6818342

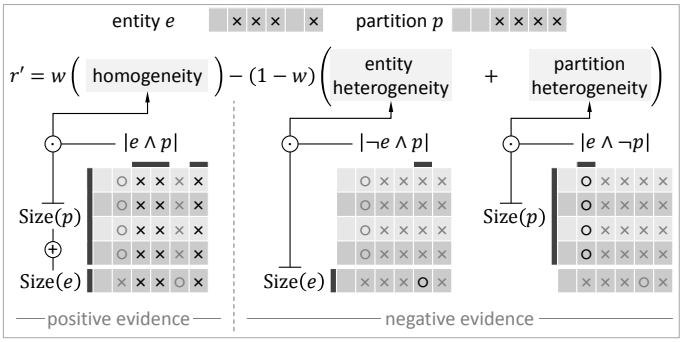


Fig. 3. Cinderella's local rating.

deletes, Cinderella merely removes the deleted entity from its partition. The partitioning itself remains unchanged. Empty partitions will be deleted. Upon updates, Cinderella also runs the insert routine but without actually inserting. In case the updated entity is assigned to a new partition it is moved. Otherwise, Cinderella updates the entity in place.

Assigning entities to partitions during insert relies on the rating of how well an entity and a partition fit together. The next section will detail this rating.

IV. Cinderella Rating

Cinderella's partition rating compares an entity synopsis with a partition synopsis to determine how well the entity would fit in the partition. Therefore, the rating takes positive evidence as well as negative evidence into account, as illustrated in Figure 3. Positive evidence is the amount of regularly structured data a partition will contain if the entity is added. We refer to this positive evidence also as homogeneity between entity and partition. Cinderella determines the evidence as the number of attributes that can be found in both, entity and partition, multiplied with the sum of the size of the partition and the size of the entity.

Homogeneity score: $h^+ = (\text{SIZE}(p) + \text{SIZE}(e)) \cdot |e \wedge p|$

Negative evidence is the amount of irregularly structured data that will result from adding the entity to a partition – in Figure 3 indicated by the bold circles. There are two kinds of negative evidence. The first one is heterogeneity on side of the entity and originates from attributes the partition has but the entity lacks. Analogously to the homogeneity, we measure this entity heterogeneity as the number of the entity's missing attributes multiplied by the size of the entity.

Entity heterogeneity score: $h_e^- = \text{SIZE}(e) \cdot |\neg e \wedge p|$

The second negative evidence is heterogeneity on side of the partition and originates from attributes the entity has but the partition lacks. We measure this partition heterogeneity as the number of the partition's missing attributes multiplied by the size of the partition.

Partition heterogeneity score: $h_p^- = \text{SIZE}(p) \cdot |e \wedge \neg p|$

Note that Cinderella heuristically assumes that each partition it already created is rather homogeneous. Irregularity already existing in a partition is not considered since this would require more information than the simple synopses.

To get a local rating r', Cinderella subtracts the total of the negative evidence from the positive evidence.

$$r' = wh^+ - (1-w)\left(h_e^- + h_p^-\right)$$

The weight w allows balancing between the influence of positive and negative evidence. For a given data set, higher weights result in a smaller number of more heterogeneous partitions, while lower weights result in a larger number of rather homogeneous partitions. In the extreme setting of $w = 0$, any heterogeneity will result in a negative rating. As a result, Cinderella creates only perfectly homogeneous partitions like in a normal database for regularly structured data. First experimental results suggest that a weight between 0.2 and 0.5 is a reasonable setting.

The local rating r' is not comparable between partitions because the amount of data and size of the attribute set varies from partition to partition. To compensate for that, Cinderella uses the global rating r which is normalized with the partition size and the number of the involved attributes:

$$r = \frac{r'}{(\text{SIZE}(p) + \text{SIZE}(e)) \cdot |e \vee p|}$$

V. Evaluation

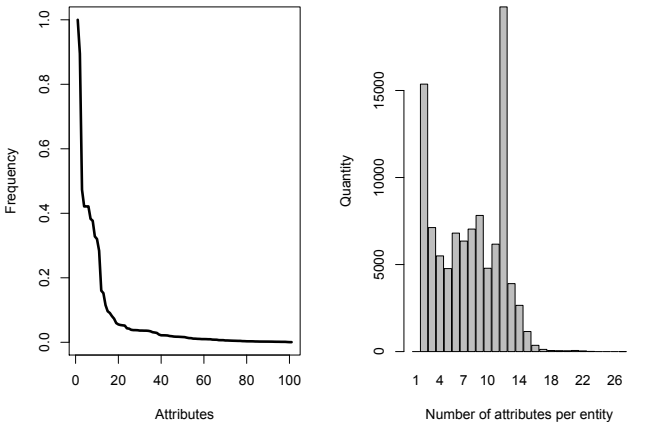
We created a prototypical implementation of Cinderella and studied Cinderella's performance on the irregularly structured data of DBpedia. Additionally, we used the TPC-H benchmark to evaluate Cinderella on regularly structured data. In this section, we first discuss the setup, which contains details on the used implementation, and then present the results.

A. Setup

We implemented Cinderella in PostgreSQL 9.3 using views, triggers, and stored procedures. Given a universal table, each data manipulation operation triggers a stored procedure, which implements the entity-based Cinderella algorithms presented in this paper. The prototype creates a regular table for each partition as well as a single catalog table for the meta data of all partitions. Cinderella uses the meta data to rewrite incoming queries to a `UNION ALL` over all partitions that contain the set of requested attributes. The prototype provides transparent data access through Cinderella, as the user inserts data to the universal table using regular SQL statements.

Hardware platform for our experiments was a Windows 8 computer with an i7 CPU and 8 GB of memory. We executed two kinds of experiments. With data taken from DBpedia, we evaluated the performance of insert operations and how much selective queries on irregularly structured data can benefit from Cinderella. We also studied the influence of Cinderella's two main parameters, the partition size limit B and the weight w in the DBpedia setup. With the TPC-H benchmark, we investigated the effect of Cinderella on regularly structured data. In the following, we detail the two experiments on irregularly and regularly structured data.

Final edited form was published in "International Conference on Data Engineering Workshops. Chicago 2014", S. 284 - 291. ISBN: 978-1-4799-3481-2
https://doi.org/10.1109/ICDEW.2014.6818342



(a) Attribute frequency. (b) Number of attributes per entity.

Fig. 4. Attribute distribution in the DBpedia data set.

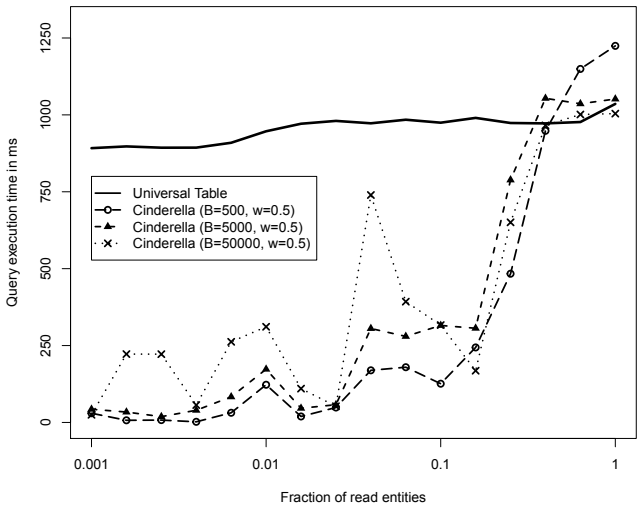


Fig. 5. Average query execution time for different partition size limits B.

B. Irregularly Structured Data

DBpedia [6] is a large database of irregularly structured data, created by a diversity of web users. Given DBpedia's heterogeneity, selective queries can benefit from horizontal partitioning by early elimination of irrelevant partitions from data access. For this experiment, we extracted $100\,000$ person entities with a total of 100 attributes. Although we limit the data set to person records, it exhibits the typical long tail distribution of irregularly structured data. Figure 4 shows the distribution of the attributes in the data set. In particular, Figure 4(a) shows the distribution of the attribute frequency, i.e. how many entities of the data set instantiate a given attribute. As can be seen, two attributes are extremely common and appear on almost every entity. Eleven attributes are fairly common and appear on over $30\,\%$ of the entities, while $85\,\%$ of the attributes appear on less than $10\,\%$ of the entities. Figure 4(b) shows the distribution of the number of attributes per entity. While the majority of entities have between two and 15 attributes, a few entities have up to 27 attributes.

We inserted the data into an Cinderella-partitioned universal table, under different settings of partition size limit and weight. The DBpedia person entities were inserted in random order. In the process, we measured the execution of the inserts; afterwards we recorded metrics about the partitioning Cinderella had created in the particular setting and measured the execution time of selective queries. For this purpose, we generated a synthetic workload since there is no common or standardized DBpedia workload. The goal was to obtain queries with different selectivity to evaluate the effect of Cinderella according to the query selectivity. We created multiple sets of attributes. Each of the individual attributes forms an attribute set. Additionally, we combined the 20 most frequent attributes to pairs and triples. For each of these attribute sets we generated a query of the form

```
SELECT a1, a2, ... FROM universalTable
WHERE a1 IS NOT NULL OR a2 IS NOT NULL ...
```

where $\{a_1, a_2, \ldots\}$ is the corresponding attribute set. Each of these queries returns only entities that instantiate the given attributes. The selectivity of the queries varies depending on the attributes queried. We collected representative queries to reasonably cover the range of possible selectivities; three representative queries for each selectivity.

We start the discussion of the results with the measurements of query execution time. This is followed by a more detailed study of the influence of the weight in the partition rating on the resulting partitioning. Finally, we discuss the measurements of insert execution time.

Query Execution Time: For the representative queries of our synthetic workload, we measured the execution time using different partition size limits and weights. For comparison, we measured the execution time on the original universal table. Neither the Cinderella partitions nor the universal tables had an index on any column. Figure 5 shows the average query execution times depending on the selectivity for a partition size limit of 500, 5000, and $50\,000$ entities. The weight was set to 0.5. As expected, the query execution time increases with a decreasing selectivity of the queries since more data has to be read. In contrast, the query execution time increases only slightly on the original universal table. Regardless of the actual selectivity, queries have to scan the whole table here. Consequently, Cinderella achieves a significant speedup for selective queries (selectivity < 0.2). In general, the more entities a query reads, the smaller the benefit of horizontal partitioning is. Queries of low selectivity (> 0.3) are likely to access every partition and do not profit from Cinderella. With our prototype, these queries show a longer execution time with Cinderella than on the universal table. We attribute large parts of this overhead to the implementation of our prototype. For instance,

Final edited form was published in "International Conference on Data Engineering Workshops. Chicago 2014", S. 284 - 291. ISBN: 978-1-4799-3481-2
https://doi.org/10.1109/ICDEW.2014.6818342

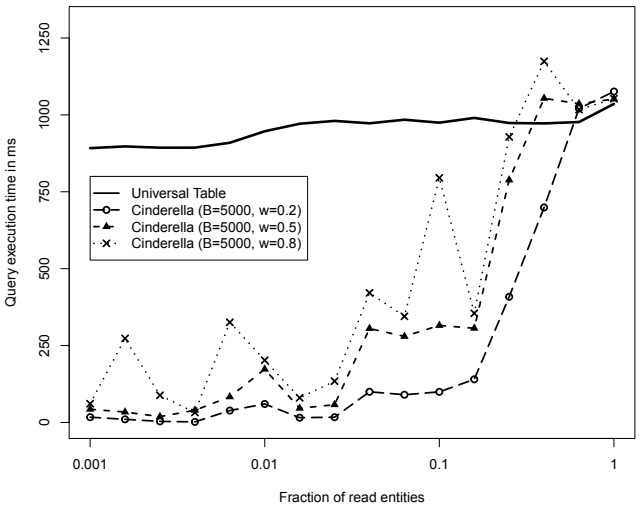


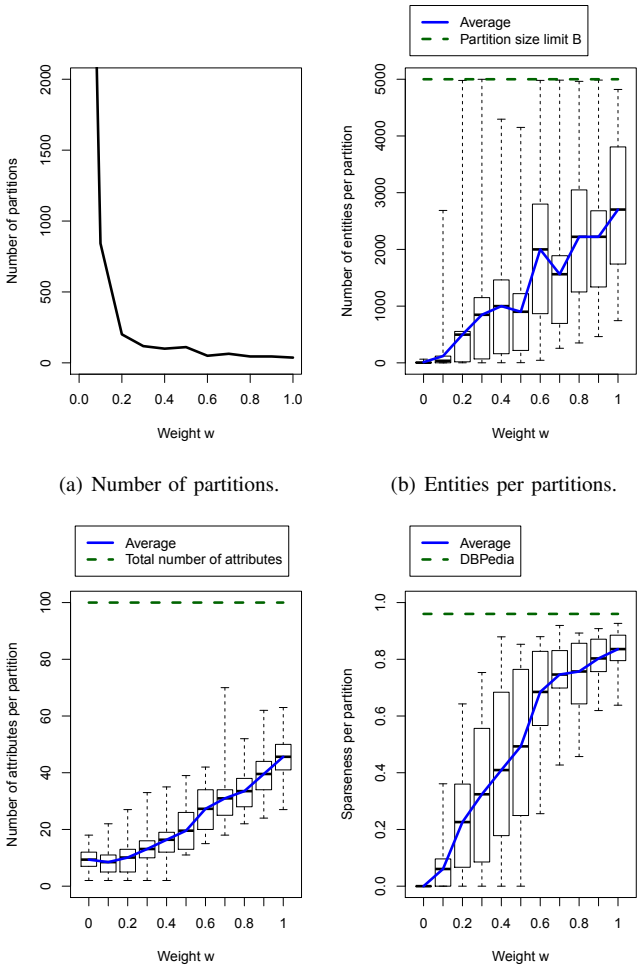
Fig. 6. Average query execution time for different weights w.

during the union operation, the database system has to project all tuples of every involved partition to the common schema. This cost can be avoided by an implementation directly in a specialized database engine with a native presentation for irregular data. There, we expect low selectivity queries not to suffer more than queries suffer from normal range or hash partitioned tables. Nevertheless, the aim of Cinderella is to speed up selective queries and it is seen to be capable of doing so.

Figure 5 also shows the impact of the partition size limit on Cinderella's benefit. Given a data set, a smaller partition size limit allows Cinderella to build more homogeneous partitions. This leads to lower and more stable query execution time, particularly for queries of medium selectivity. On the downside, a smaller partition size limit also results in a larger number of partitions necessary to host the data. This requires less selective queries to unite more partitions, which increases the overhead for such queries. Consequently, the partition size limit should be set lower for very selective workloads and higher for less selective workloads.

Figure 6 shows the impact of the weight of the partition rating on Cinderella's benefit. We see a similar picture here. Higher weights typically result in fewer but larger partitions. For very selective queries, a lower weight is beneficial, while queries of very low selectivity slightly profit from a higher weight. However, the optimal weight depends more on the irregularity of the data set than on the workload. For the DBpedia data set we used in the evaluation, 0.2 is seen to be a good balance between positive and negative evidence in the partition rating. For other data sets with another irregularity another weight is likely to be optimal.

Influence of Weight on Partitioning: For a more detailed picture of the influence of the weight w on Cinderella's partitioning, we partitioned the DBpedia data set with Cinderella using different settings for the weight. For each of the resulting partitionings, we recorded (1) the number of partitions, (2) the number of entities per partition, (3) the number of attributes per partition, and (4) the sparseness per partition. The maximum partition size limit B was set to 5000. Figure 7 illustrates the results.



(a) Number of partitions. (b) Entities per partitions.

(c) Attributes per partitions. (d) Sparseness per partition.

Fig. 7. Influence of the weight w on the partitioning of the DBpedia data set.

As Figure 7(a) clearly shows, the lower the weight the more partitions Cinderella creates. Particularly with a weight less than 0.2 the number of partitions explodes. With a very low weight, the homogeneity score loses its influence on the partition rating. As a result, most entity–partition pairs get a negative rating, causing Cinderella to create a new partition even for entities that have a large overlap with the schema of existing partitions. In the extreme case of $w = 0$ all created partitions are completely homogeneous.

Naturally, the number of entities in the partitions behave the opposite way, as shown in Figure 7(b). Higher weights allow more heterogeneity within partitions, so that more entities are assigned to the partitions. With a very low weight ($w < 0.2$),

Final edited form was published in "International Conference on Data Engineering Workshops. Chicago 2014", S. 284 - 291. ISBN: 978-1-4799-3481-2
https://doi.org/10.1109/ICDEW.2014.6818342

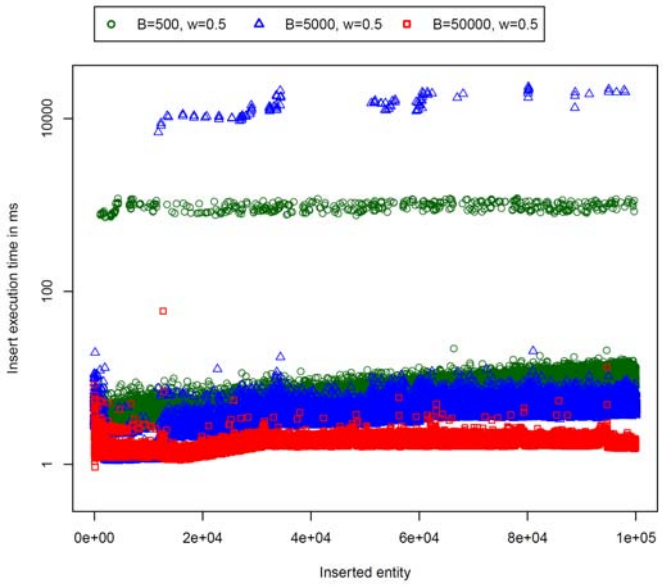


Fig. 8. Insert execution time for different partition size limits B.

all partitions remain very small. Medium weights produce a few partitions filled to the maximum capacity, although the majority is less than half full. The large spread results from the attribute distribution. On the one hand, a large fraction of entities has only a small number of attributes and many of these attributes are also the most common attributes. Hence, these entities pile up in a large partition. On the other hand, a very small fraction of entities has a large number of very uncommon attributes. These entities result in very small partitions.

Figure 7(c) shows the number of attributes per partition. Again, the higher the weight the more heterogeneity per partition is allowed and the higher the number of attributes. However, in all settings all partitions have significantly fewer attributes than the universal table has. This clearly shows how Cinderella facilitates the pruning of irrelevant data. The lower the number of attributes the higher the probability that a partition can be pruned from query processing.

Since the number of entities and the number of attributes per partition increase with the weight, the sparseness per partition has to increase as well for a given data set. Figure 7(d) shows this effect for the DBpedia data set. For the homogeneity-preserving setting of $w = 0$, the sparseness per partition is obviously zero. In contrast, higher weights ($w \geq 0.6$) do not form any partition of very low sparseness. With all medium weight settings, Cinderella forms mainly partitions with a sparseness considerably lower than the sparseness of the original data set (0.94).

All results in Figure 7 underline that medium weights produce the most reasonable results. A perfect partitioning has to reach contradicting goals. On the one hand a small number of partitions that are well filled up to the partition size limit minimizes the overhead for queries of low selectivity.

TABLE I
QUERY EXECUTION TIME ON REGULAR DATA (TPC-H).

Scenario	Partition size limit	Total query execution time	
Standard TPC-H	–	24.23 s	(100.00 %)
Cinderella I	500 entities	26.38 s	(108.87 %)
Cinderella II	2000 entities	25.61 s	(105.69 %)
Cinderella III	10 000 entities	24.54 s	(101.27 %)

On the other hand very dense partitions with small attribute sets facilitate the highest speedup for very selective queries. Medium weights balance between these contradicting goals.

Insert Execution Time: We also measured the execution time of the insert operations, when we loaded the data set. Figure 8 shows the results for different partition size limits and a weight of 0.5. As can be seen, the majority of insert operations finishes in between 1 ms and 10 ms. With a lower partition size limit and a larger number of partitions, the inserts take a little longer, because the size of the partition catalog increases. A small fraction of inserts needs considerably longer, though. These are insert operations where a partition split occurs. As Figure 8 clearly shows, the number of insert operations with a split decreases with an increasing partition size limit. With a partition size limit of 500 entities, Cinderella performs a split 448 times, for a limit of 5000 entities 100 times, and for a limit of 50 000 no split occurs. At the same time, the cost of a split increases with the partition size limit, because more entities have to be reassigned and physically moved during a split.

C. Regularly Structured Data

The TPC-H benchmark [7] has, in contrast to DBpedia, a concrete specification of the workload. For this experiment, we loaded TPC-H data with a scale factor of 0.5 into an Cinderella-partitioned universal table. Views on the partitions created by Cinderella emulated the standard TPC-H tables. The TPC-H data is perfectly regular. While inserting this data into an Cinderella-partitioned universal table, Cinderella should be able to find a partitioning which is similar to the TPC-H table schema. Any partitioning different from the TPC-H schema would result in a significantly higher execution time for the TPC-H queries. Accordingly, we measured the execution time of the benchmark's queries on the TPC-H-schema-emulating views using different partition sizes and, for comparison, the execution time on the regular tables of the TPC-H schema.

Table I shows the results. The table lists the total execution time of the 22 TPC-H queries in four scenarios. One scenario is the normal TPC-H benchmark without Cinderella and provides the baseline. The other three scenarios show Cinderella with different settings for the partition size. As can clearly be seen, the presence of Cinderella adds only a small overhead to the query execution time on regularly structured data. In fact, Cinderella finds only partitions which exactly fit the TPC-H schema in any of the three settings. Again, we see that a larger partition size decreases the cost of the additionally necessary union operations for queries that span multiple partitions.

Final edited form was published in "International Conference on Data Engineering Workshops. Chicago 2014", S. 284 - 291. ISBN: 978-1-4799-3481-2
https://doi.org/10.1109/ICDEW.2014.6818342

VI. Related Work

Naturally, the online partitioning problem of universal table is related to partitioning techniques traditionally applied in database systems. Other related research field are schema mining and inference of hidden schemas.

Partitioning is an old means of physical design. Entity sets can be either partitioned vertically or horizontally. Partitioning advisor tools became a popular research topic along with research on index advisor tools [8], [9], [10]. An online partitioning tool was presented only recently [11]. For horizontal partitioning, all these tools consider mainly range partitioning because it is the partitioning mostly used in traditional relational database setups. In web-scale databases, where load balancing over a large number of nodes is the main concern, hash partitioning is the common choice [12], [13], [14]. To the best of our knowledge, horizontal partitioning according to schema properties has not been considered so far.

Schema mining aims at finding interesting structures and structural regularities in databases of semi-structured data. It became a popular research topic in the second half of the 1990's, when the rise of the web rapidly increased the amounts of irregular, semi-structured data. Commonly, schema mining approaches use an edge-labeled graph as in the object exchange model [15] to represent the semi-structured data. They differ, however, in the used mining technique. One technique used is clustering [16]. Here, a set of extracted structural types is clustered into k groups of similar types. Another technique extracts frequent structures using the notion of minimum relative support [17]. Although schema mining is a very closely related topic, the schema mining techniques are not applicable to our horizontal partition problem. Schema mining does not partition data; it finds structural representatives.

Closely related to schema mining is the inference of so-called hidden schemas [18]. The goal is also to partition a universal table but vertically and offline. The presented approach clusters attributes based on their co-occurrence. Co-occurrence of two attributes is measured with the Jaccard coefficient. With the coefficients the authors create an adjacency matrix and apply a k-nearest-neighbor clustering to obtain the partitions. Although very closely related, the technique is not directly applicable to our problem. First, it requires additional knowledge about the data to provide a reasonably good k. Second, the algorithm is meant to work offline since the complex clustering algorithm comes with considerable overhead.

VII. Conclusions

The universal table is a common setup in databases involving a significant share of irregularly structured, hard-to-model data. Horizontal partitioning can help to increase the efficiency of queries on such universal tables. Maintaining such a partitioning poses an optimization problem in the field of physical design. We defined this as Online Partitioning Problem for irregularly structured data. With Cinderella, we proposed an online algorithm for horizontal partitioning of irregularly structured data in universal tables. Cinderella autonomously separates entities into fix-sized partitions based on the schema properties of the entities, which allows queries to prune partitions of irrelevant entities from processing before actually touching them. In the evaluation, Cinderella showed capable of significantly decreasing the execution time of selective queries compared to an unpartitioned universal table. Cinderella is still ongoing work. We are currently extending our evaluation to further improve Cinderella. A promising next step is an specialized data management. The smaller the partition size the better the achieved efficiency. As smaller partitions, however, increase the total number of partitions and thereby the overhead of Cinderella, we will continue our research and aim to further improve Cinderella. Particularly, we will look to improve the management of a large number of partition synopses with specialized data structures and include further aspects of physical database design like caching or indexing.

References

[1] S. Acharya, P. Carlin, C. A. Galindo-Legaria, K. Kozielczyk, P. Terlecki, and P. Zabback, "Relational support for flexible schema scenarios," *The Proceedings of the VLDB Endowment*, vol. 1, no. 2, 2008.

[2] R. Agrawal, A. Somani, and Y. Xu, "Storage and Querying of E-Commerce Data," in *VLDB'01*, 2001.

[3] J. L. Beckmann, A. Halverson, R. Krishnamurthy, and J. F. Naughton, "Extending RDBMSs To Support Sparse Datasets Using An Interpreted Attribute Storage Format," in *ICDE'06*, 2006.

[4] K. C.-C. Chang, B. He, C. Li, M. Patel, and Z. Zhang, "Structured Databases on the Web: Observations and Implications," *SIGMOD Record*, vol. 33, no. 3, 2004.

[5] J. L. Beckmann, "The CNET E-Commerce Data Set," University of Wisconsin, Madison, Tech. Rep., Jul. 2005.

[6] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer, "Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia," *Semantic Web Journal*, 2014.

[7] Transaction Processing Performance Council, "TPC Benchmark^{TM} H, Revision 2.16.0," Jun. 2013.

[8] J. Rao, C. Zhang, N. Megiddo, and G. M. Lohman, "Automating physical database design in a parallel database," in *SIGMOD'02*, 2002.

[9] S. Agrawal, V. R. Narasayya, and B. Yang, "Integrating Vertical and Horizontal Partitioning Into Automated Physical Database Design," in *SIGMOD'04*, 2004.

[10] D. C. Zilio, J. Rao, S. Lightstone, G. M. Lohman, A. Storm, C. Garcia-Arellano, and S. Fadden, "DB2 Design Advisor: Integrated Automatic Physical Database Design," in *VLDB'04*, 2004.

[11] A. Jindal and J. Dittrich, "Relax and Let the Database Do the Partitioning Online," in *BIRTE'11*, vol. 126, 2011.

[12] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, "Bigtable: A Distributed Storage System for Structured Data," in *OSDI'06*, 2006.

[13] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in *SOSP'07*, 2007.

[14] A. Lakshman and P. Malik, "Cassandra - A Decentralized Structured Storage System," *Operating Systems Review*, vol. 44, no. 2, 2010.

[15] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom, "Object Exchange Across Heterogeneous Information Sources," in *ICDE'95*, 1995.

[16] S. Nestorov, S. Abiteboul, and R. Motwani, "Extracting Schema from Semistructured Data," in *SIGMOD'98*, 1998.

[17] K. Wang and H. Liu, "Discovering Structural Association of Semistructured Data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 12, no. 3, 2000.

[18] E. Chu, J. L. Beckmann, and J. F. Naughton, "The Case for a Wide-Table Approach to Manage Sparse Relational Data Sets," in *SIGMOD'07*, 2007.