Methods of forming the join table

S V Zykin
Sobolev Institute of Mathematics, Siberian Branch of the RAS, Acad. Koptyug ave.,
h. 4, 630090, Novosibirsk, Russia
szykin@mail.ru

Abstract. The present work considers the methods of forming data views as a join table intended to be used in table-driven user applications, such as spreadsheets, multidimensional arrays, etc. The main purpose of the join table is to organize the interaction of such applications with the corporate database. The join table can be used at a physical level. This will significantly reduce the time required to generate custom data views in the applications. The use of the join table becomes inefficient, if it needs to be changed in a short period of time. However, the use of storage devices with associative access or "logic-per-track" devices for data processing will significantly reduce the join table transformation time.

1. Introduction
Data description and data view traditionally have three levels in databases (DB): physical level includes a description of the place and structure of the data storage; global logical level is designed to describe the database as a whole (DB schema); external level includes the description of the user data view (the description of the data in the application). The development of new technologies for data view and processing (cloud technologies) [1-4] puts forward additional requirements to the structure and composition of the data description. A need for additional levels (layers) arises: the external layer performs the same functions; the conceptual middleware layer provides the ability to hide the conceptual level heterogeneity among different databases, such as DB2, Oracle, etc.; the conceptual layer using the DB schema is utilized for efficient query processing (optimization, security, etc.); the physical middleware layer provides the ability to hide the heterogeneity between different platforms (Windows, Mac OS, Linux, etc.), the physical layer is responsible for continuous monitoring and configuration of the database to achieve optimal scaling and efficient allocation of resources in the cloud. The paper [5] introduces a new transactional "database-as-a-service" (DBaaS) called "Relational cloud". In DBaaS, the operational costs on configuration, scaling, performance tuning, backup, privacy, and access control are the responsibility of the service operators. Thus, three main tasks are solved: efficient multi-tenancy, scalability, and privacy of the database. During system operation, by monitoring query patterns and data access, the system obtains useful information for various optimization and security functions, reducing the configuration effort for user and operators. In some cases, data integration between local databases and cloud data is required. The paper [6] presents a system called BigIntegrator enabling execution of queries that combine data in cloud-based storages with relational DB. A scenario of query execution from both different kinds of data sources is used as an illustration. BigIntegrator also utilizes knowledge of the constraints to produce efficient query execution.
The work [7] considers the basic concepts and methods of implementing the five-layered model and the main improvements achieved at the system level. Moreover, the interaction of a multi-layered model with transactional ACID properties (ensuring reliable and predictable operation) is considered. Such database management system (DBMS) allows one to satisfy numerous new requirements and changes so far as processing environments, data types, functional extensions, heterogeneity, autonomy, scalability, etc. It is shown that the five-layered hierarchical model, proposed more than 20 years ago, is able to accommodate all the extensions and optimizations of the originally established paradigm.

In [8] semantic dependencies are derived between entities that do not take into account the data models, and the databases themselves do not see these dependencies. To compensate for this situation, it is proposed to create storage from flat disaggregated data, and then process any queries and hidden records in the updated denormalized storage. In the graph world, linked nodes are stored as linked data. Where there are joins in the domain, there are joins in the data. In fact, it is proposed to store the semantic network on the example of queries in social networks that require calculating the degree of a particular connection, for example, "friends".

In work [9] new ways of access to the memory – Storage Class Memory (SCM) – are considered. For this purpose, the problems of SCM and existing programming models that help solve these problems are discussed. Moreover, new data structures based on these models are reviewed. Special attention is paid to the problem of testing and fault tolerance. For this purpose, the SCM emulation methods for end-to-end testing of software components based on SCM are considered.

Article [10] presents the Semantics Toolkit (SemTK) platform, which provides access to Polyglot-persistent Big Data stores, while giving the impression that all data are fully captured within a knowledge graph. SemTK allows data to be stored across several storage platforms (e.g. Big Data stores such as Hadoop, graph databases, and semantic stores) with the best-suited platform adopted for each data type, while maintaining a single logical interface and access point. Thereby, users are provided with a knowledge-driven view of their data. The ease of the use and the benefits of querying and constructing a knowledge graph with SemTK are demonstrated in the mentioned work.

Research [11] proposes distributed and unstructured database HBase under the Hadoop platform to store massive temporal data and to build a data storage model with a single store. A multi-level distributed hash table indexing algorithm (t-DHT) is developed. This algorithm guarantees an efficient and rapid search for the temporal attribute values of the temporal column family. By mapping attribute value to two-dimensional space, the transformation of the data to spatial object is realized. Dividing the data area with the spatial data processing method results in a multilevel index table HBase by using the distributed hash table.

Paper [12] summarizes the requirements to spatial raster database management systems and raster data processing platforms from a domain-specific perspective as well as from a computing point of view. The necessity of tight integration between the database system and the processing system is also discussed. These requirements resulted in the development of a global platform Oracle Spatial GeoRaster designed for a raster data processing. GeoRaster defines an integrated raster data model, supports image compression, data manipulation, general and spatial indices, content and context based queries and updates, versioning, concurrency, security, replication, backup and recovery.

Article [13] proposes a physical model design language called PhyDL that allows describing all inputs and outputs of the physical design phase. To enhance the reuse of the existing advisors, a storage called Meta-Advisor is developed that stores all components of the physical design.

The publication [14] researches the multilingual platform messaging problem in e-commerce. Special methods of data storage and exchange provide high system throughput and efficient user concurrency. Moreover, the high-performance server Tomcat is used, and MySQL database is utilized for data storage.

Data view as a cluster on a physical level is close in structure to our study subject [15]. A cluster in Oracle DBMS allows one to store multiple logical-level relations that are used concurrently at the requests execution. Related data of different relations are stored together in the same blocks of the DB
file, thus reducing the iteration time of such relations. An index file based on a set of attributes is automatically created together with the cluster. The only constraint is that there is only one set of indexed attributes for the cluster. In this paper, a data structure without this constraint is proposed. Article [16] contains the iteration algorithms for the relations designed for jointly stored data. This approach to the estimates calculation and data processing fully applies to the join table.

2. Principles of forming the join table

Let us consider a formal definition of a data model as a join table. Information system model is the description of the data view, a set of operations to manipulate the data, and the constraints of data integrity. This paper considers in detail the formal definition and formation of the join table view. Let the initial DB be given as a set of relations \( \{ R_1, R_2, \ldots, R_k \} \), each of which is defined by the subsets of the attributes set \( U = \{ A_1, A_2, \ldots, A_n \} \). Tuple \( t_i[X] \in R_i \) is a set of values, each corresponding to its attribute \( A_j \in X \subseteq U \). Let \( \langle R_i \rangle = X \) be a set of attributes corresponding to the relation \( R_i \). The value of the attribute \( A_j \) in the tuple \( t \) is: \( t[A_j] = emp \), if it is the result of a missing tuple in the relation when the tuple join operation is performed. An empty value \( emp \) differs from undefined values \( Null \) in the database.

Henceforth, the formal definition of the natural join operation is to be used [17]:
\[
R_i \bowtie R_j = \pi_{(R_i \times R_j) \setminus (R_i \setminus R_j)}(\sigma_{(R_i \setminus R_i) \setminus (R_i \setminus R_j)}(R_i \times R_j)),
\]
where \( \times \) is the Cartesian product operation on relations \( R_i \) and \( R_j \); \( \sigma \) is the selection operation, which results in tuples with equal values of the homonymous attributes \( A_m \), where \( A_m \in R_i \) and \( A_m \in R_j \); \( \pi \) is the projection operation by the set of \( R_i \) and \( R_j \) relations attributes, resulting in only one of the homonymous columns left, since their values coincide.

Definition 1. A natural join of relations \( R_i \bowtie R_j \bowtie \ldots \bowtie R_k \) will be called existing if there is at least one ordered sequence of relations \( R_1, R_2, \ldots, R_k \), such that:
\[
(\bigcup_{i=1}^j \langle R_i \rangle) \cap (R_{j+1}) \neq \emptyset, \ j = 1, k - 1.
\]
Remark. Due to the natural join operation commutativity, there will be existing any relations permutation if there is at least one ordered permutation found that satisfies definition 1.

Let us consider the method of forming a join table. Let us denote it as \((S, l)\)-table corresponding to the transformation \( R_1, R_2, \ldots, R_k \rightarrow (S, l) \). The symbol \( S \) denotes the table header, which will coincide with the whole set of attributes \( A_1, A_2, \ldots, A_n \). The symbol \( l \) denotes the vector of occurrences. The following method of forming \((S, l)\)-table does not imply consideration of optimization issues in this article. Let the joins satisfying definition 1 be given for all combinations without repetitions from \( k \) to \( m \) of relations \( R_i \), \( m = 1, k \). For the current tuple \( u \) of such a join, a tuple \( t \) is formed: \( t[A_j] = u[A_j] \), if \( A_j \) belongs to at least one of the relations of the current join, otherwise \( t[A_j] = emp \). Each tuple \( t \) is assigned attributes of the vector of occurrences: \( l(t) = (l_1(t), l_2(t), \ldots, l_k(t)) \), where \( l_j(t) = 1 \) if the relation \( R_j \) belongs to the current join, otherwise \( l_j(t) = 0 \).

For the tuples of \((S, l)\)-table, the relation of subordination is to be defined.

Definition 2. Tuple \( t_1 \in (S, l) \) is subordinate of tuple \( t_2 \in (S, l) \), if for its attributes values the following is fulfilled: if \( t_1[A_j] \neq t_2[A_i] \), then \( t_1[A_i] = emp \) and \( l_j(t_1) \leq l_j(t_2) \), \( j = 1, k \). Henceforth, it will be written: \( t_1 \leq t_2 \).

Remark 1. The considered definition formally corresponds to the definition of the partial order which means that tuple \( t_2 \) contains all the subordinate tuples. Therefore, when implementing \((S, l)\)-table, it is sufficient to keep tuple \( t_2 \), while all the other will be obtained from it (the corresponding property is discussed below).

Remark 2. Vector of occurrences \( l(t) \) is used to determine the use of relation \( R_i \) when forming the current tuple \( t \). If every relation \( R_i \) has attributes belonging only to this relation in \((S, l)\)-table, then these attributes value \( emp \) is an answer to the set question. This, however, will limit the applicability of this approach if, for example, there is a superkey in the list of binding relations. These relations often have no unique attributes, however, they are useful to achieve any properties of \((S, l)\)-table.
Remark 3. If the sequence order of the attributes values is located on the physical level, then values
emp need not be stored in tuple t. Current attribute A_j in t has an empty value emp, if for all R_i,
where A_j \in R_i, l_i(t) = 0. Otherwise, the attribute value is determined or Null.

Definition 3. Implementation of (S,l)-table has no subordinated tuples.

Let us define the projection operation on (S,l)-table.

Definition 4. As a projection π_{R_1,R_2,\ldots,R_m}(S, l), one will call a set of tuples u[X] such that there is
tuple t: u[X] = t[X], where X = \bigcup_{i=1}^{l} (R_i, l) and l_i(t) = 1, i = 1, m.

The size of (S,l)-table can be enormous. However, the absence of a large number of duplicate
attribute values used for the join allows this effect to be avoided. Article [18] estimates the join size in
a general form. This estimation can be used to decide on viability of using (S,l)-table. Let us consider a
simple example when the source database consists of three relations:

\[ R_1 = \begin{array}{ccc} A & B & C \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{array} \]

\[ R_2 = \begin{array}{ccc} B & D & E \\ b_1 & d_1 & e_1 \\ b_2 & d_2 & e_2 \end{array} \]

\[ R_3 = \begin{array}{ccc} A & D & F \\ a_1 & d_1 & f_1 \\ a_2 & d_2 & f_2 \\ a_3 & d_1 & f_2 \end{array} \]

The corresponding (S,l)-table will be of a form:

\[
(S, l) = \begin{array}{ccccccc} A & B & C & D & E & F & l \\ a_1 & b_1 & c_1 & d_1 & e_1 & f_1 & 111 \\ a_2 & b_2 & c_2 & d_2 & e_2 & f_2 & 111 \\ a_3 & b_1 & emp & d_1 & e_1 & f_2 & 011 \end{array}
\]

As seen from the example, the volume of (S,l)-table is less than the total volume of the source DB.
Moreover, the relations join request for (S,l)-table will be performed significantly faster due to the
following property.

Property 1. For existing join of relations R_{j_1}, R_{j_2}, \ldots, R_{j_m}, where m = 1, k, equation holds:

\[ R_{j_1} \bowtie R_{j_2} \bowtie \cdots \bowtie R_{j_m} = \pi_{R_{j_1},R_{j_2},\ldots,R_{j_m}}(S, l). \]

Remark. The considered property demonstrated that the join operation for a relation set can be
replaced with a projection operation on (S,l)-table. Duplicate tuples in the projection are a common
disadvantage of the projection operation.

3. Forming data views for (S,l)-table

The previous section has considered the principles of forming (S,l)-table. Let us formalize this
problem as algorithms. The target algorithm will be LOADL. Auxiliary algorithms are necessary for its
operation:

- COMB is formation of the current combination from k elements under m without repetition.
- EXIST is an algorithm to check the current join for the existence condition.
- LIST is the transformation of the current join in a component of (S,l)-table.

Let us introduce the notations for the parameters of the algorithms:

- \((A_1, A_2, \ldots, A_n) = \vec{A}\) is a vector constructed from DB attributes;
- \((R_1, R_2, \ldots, R_k) = \vec{R}\) is a vector constructed from DB relations;
- \((i_1, i_2, \ldots, i_m) = \vec{l}\) is a current combination from k elements under m without repetitions, and
  the initial state is \((0, 0, \ldots, 0)\).

Algorithm COMB: input(l_m, m, k), output((TRUE, FALSE)), value TRUE indicates the presence of the current combination, FALSE indicates the absence of the current combination (none left).

Algorithm EXIST: input(l_m, \vec{R}, m, k), output(l_m, \{TRUE, FALSE\}), value TRUE indicates that the join existence condition is fulfilled for the relations with numbers l_m, FALSE is for the opposite situation.

Let us consider a pseudocode of the algorithm for transforming the existing join in (S,l)-table.
LIST: input \((I_m, \vec{R}, A, m, k, n)\)
output \((S, l)\)

\((S, l) = \emptyset\)

\(R' = \langle R_{i_1} \rangle \cup \langle R_{i_2} \rangle \ldots \cup \langle R_{i_m} \rangle\)

\(R = R_{i_1} \bowtie R_{i_2} \bowtie \cdots \bowtie R_{i_m}\)

for each tuple \(u\) in \(R\)

\(t = (\text{emp}_1, \text{emp}_2, \ldots, \text{emp}_n)\)

for \(i = 1\) to \(n\)

if \(A_i \in R'\) then \(t[A_i] = u[A_i]\)

endfor

\(l(t) = (0_1, 0_2, \ldots, 0_k)\)

for \(i = 1\) to \(k\)

if \(i \in I_m\) then \(l_i[t] = 1\)

endfor

\((S, l) = (S, l) \cup (t, l(t))\)

endfor

doi:10.1088/1742-6596/1260/4/042005

The complexity of the algorithm is caused by performing the natural join operation. As a solution to this problem, one can suggest using the results of previous joins. The presented version of the algorithm \(LIST\) is intended to demonstrate the correct formation of the view. It should be noted that there can be no subordinate tuples within a single join \(R = R_{i_1} \bowtie R_{i_2} \bowtie \cdots \bowtie R_{i_m}\).

Next basic algorithm will be considered for forming \((S,l)\)-table, the algorithm designed as the main procedure. There are no input and output parameters for such a procedure and the result of its use is the view of \((S,l)\)-table stored on permanent storage devices.

LOADL:

\((S, l) = \emptyset\)

for \(m = k\) to \(1\) step \(-1\)

\(l_m = (0, 0, \ldots, 0)\)

do while \(\text{COMB}(l_m; m; k)\)

if \(\text{EXIST}(l_m, \vec{R}, m, k)\) then

\((S', l') = \text{LIST}(I_m, \vec{R}, A, m, k, n)\)

for each tuple \(u\) in \((S', l')\)

\(\text{tup}le\_out = \text{TRUE}\)

for each tuple \(t\) in \((S, l)\)

if \(u \lessdot t\) then \(\text{tup}le\_out = \text{FALSE}\)

endforeach

if \(\text{tup}le\_out\) then \((S, l) = (S, l) \cup u\)

endforeach

dodо

dо

The presented algorithm is called basic since it fulfills the requirements of simplicity and unambiguous interpretation. This cannot be said about its efficiency in terms of iterations number. However, there are ways to transform this algorithm to improve its efficiency. For example, \((S,l)\)-table can be formed from the bottom up. Then it is possible to use intermediate joins stored in the cache. At the physical level, indexed files can be used to reduce I/O operations. In any case, the correctness of the algorithm modification is proven by its equivalence to the basic one.
4. Search operation in \((S,I)\)-table

When searching for data in \((S,I)\)-table, the basis is the projection operator over \((S,I)\)-table used to select the subset of the tuples. Moreover, the longest relations join operation, which is most often used in relational databases, is replaced by the projection operation on \((S,I)\)-table. This significantly reduces the response time for the request. To implement joins, indexed files can be created for generic attributes or files can be sorted. The efficiency of sorting for query execution is highly questionable, since the operation itself requires significant resources, primarily, time. The use of indexed files is effective if there are few different values for the attributes on which the join is performed. Data view in the form of \((S,I)\)-tables is implemented more efficiently, if join attributes can take more of different values. Therefore, an effective method of implementing search queries will be realized when the methods of both data indexing and view in the form of \((S,I)\)-table are used simultaneously.

Let us analyze the situation when \((S,I)\)-table is used for physical data view. In this paper, additional options for \((S,I)\)-table compression without changing its table structure will be considered, which is important for performing operations in content-addressable memory.

**Definition 5.** (Condition of incomplete gluing). Tuples \(t_1\) and \(t_2\) can be combined by the condition of incomplete gluing, if for each attribute \(A_m\) one of the following conditions is satisfied:

\[ t_1[A_m] = t_2[A_m], \text{ either } t_1[A_m] = \text{emp}, \text{ or } t_2[A_m] = \text{emp}, \]

moreover, a set of all schemes \(R_j\), for which \(l_j(t_1) = 1\) or \(l_j(t_2) = 1\) forms a current join.

**Definition 6.** (Condition of complete gluing). Tuples \(t_1\) and \(t_2\) can be combined by the condition of complete gluing, if for each attribute \(A_m\) one of the following conditions is satisfied:

\[ t_1[A_m] = t_2[A_m], \text{ either } t_1[A_m] = \text{emp}, \text{ or } t_2[A_m] = \text{emp}. \]

The results of both types of gluing is tuple \(t: t[A_m] = t_1[A_m]\), if \(t_1[A_m] \neq \text{emp}\), and \(t[A_m] = t_2[A_m]\), if \(t_2[A_m] \neq \text{emp}\), and \(t[A_m] = \text{emp}\) in all other cases, \(m = 1, \ldots, n, l_j(t) = \max\{l_j(t_1), l_j(t_2)\}, j = 1, \ldots, k\).

The analysis of the considered algorithms shows that tuples of \((S,I)\)-table agree with the final result of the incomplete gluing, i.e. one cannot glue tuples from \((S,I)\)-table due to the rule of incomplete gluing. Let us assume the opposite. Let tuples \(t_1 \in (S,I)\) and \(t_2 \in (S,I)\) satisfy the conditions of incomplete gluing, then for the result of gluing tuple \(t\) all the schemes \(R_j\) determined in it form an existing join together, therefore, \(t \in (S,I)\), and by constructing \(t_1 \leq t\) and \(t_2 \leq t\). In other words, tuples \(t_1\) and \(t_2\) will be deleted.

Complete gluing can be applied to tuples from \((S,I)\)-table. This, however, will complicate the algorithms of editing tuples in \((S,I)\)-table. Depending on the order in which the tuples are edited, the opposite effect can be obtained: the size of \((S,I)\)-table at complete gluing will be larger than at incomplete one. The conclusion is obvious: further \((S,I)\)-table compression is possible, but not viable.

5. Conclusion

The considered data view as \((S,I)\)-table is the basis for forming various user view of, for example, spreadsheets, multidimensional arrays, etc. On the other hand, \((S,I)\)-table can be used to store data in DB physically, when the data volume in \((S,I)\)-table is acceptable. This method of data storage is a generalization of the cluster used in ORACLE DBMS. Moreover, theoretically \((S,I)\)-table is a generalization of the notion «universal relational» since it exists for any RDB schema.

Acknowledgments

The work was supported by the program of fundamental scientific researches of the SB RAS № I.5.1., project № 0314-2019-0020.

References

[1] Sakr S, Lui A, Batista D M and Alomari M 2011 A survey of large scale data management approaches in cloud environment IEEE Communication Surveys and Tutorials 13 3 pp 311–36

[2] Hsieh M, Chang C, Ho L, Wu J and Lui P 2011 SQLMR: a scalable database management
system for cloud computing *International Conference on Parallel Processing* (Taipei City) pp 315–24

[3] Alam B, Doja M, Alam M and Mongia S 2013 5-layered architecture of cloud database management system *AASRI Procedia* 5 pp 194–9

[4] Abadi D J 2009 Data management in the cloud: limitation and opportunities *IEEE Data Engineering Bulletin* 32 1 pp 3–12

[5] Curino C, Jones E P C, Popa R A, Malviya N, Wu E, Madden S, Balakrishnan H and Zeldovich N 2011 Relational cloud: a database as a service for the cloud *5th Biennial Conference on Innovative Data Systems Research* (Asilomar, CA, USA) pp 235–40

[6] Zhu M and Risch T 2011 Querying combined cloud-based and relational databases *International Conference on cloud and service computing* (Hong Kong, China, 12–14 December) pp 330–5

[7] Härder T 2005 DBMS architecture – the layer model and its evolution *Datenbank-Spektrum* 13 pp 45–57

[8] Robinson I, Webber J and Eifrem E 2015 *Graph Databases 2nd Edition* (Sebastopol: O’Reilly Media) p 238

[9] Oukid I and Lehner W 2017 Data structure engineering for byte-addressable non-volatile memory *Proceedings of ACM International Conference on Management of Data* (Chicago IL, May 14–19) pp 1759–64

[10] McHugh J, Cuddihy P E, Williams J W, Aggour K S, Kumar V S and Mulwad V 2017 Integrated Access to Big Data Polystore through a Knowledge-driven Framework *IEEE International Conference on Big Data* (Boston MA, Dec 11-14) pp 1494–503

[11] Feng C and Li B A 2017 Research of temporal information index strategy based on HBase *Advances In Information And Communication Technology* (Sanya, Jan 01–02) 107 pp 367–72

[12] Xie Q J 2016 The design of a high performance earth imagery and raster data management and processing platform *International Archives of the Photogrammetry Remote Sensing and Spatial Information Sciences* 41 B4 pp 551–5

[13] Ouared A, Ouhammou Y and Roukh A 2016 A meta-advisor repository for database physical design *Lecture Notes in Computer Science* 9893 pp 72–87

[14] Yang H, Chen X, Li J and Cao Y 2017 Design of instant messaging system of multi-language e-commerce platform *IOP Conference Series: Materials Science and Engineering* 231 012023

[15] Kumar A 2018 *Database development guide* (Oracle) p 779

[16] Zykin S and Poluyanov A Iteration of database relations for queries of special form *Journal of Physics: Conference Series* (IOP Publishing) 1050 012109

[17] Ullman J 1980 *Principles of database systems* (Computer Science Press) p 334

[18] Zykin S V 1993 Estimating list sizes in partial joins *Cybernetics and Systems Analysis* 29 6 pp 907–15