The Curious Disguises of LIST Aggregate Function

Witold Litwin
University Paris Dauphine, PSL
Witold.litwin@dauphine.fr

Abstract. *We present just discovered curious disguises of the aggregate function we proposed almost 20 years ago, naming it LIST. The proposal seemed ignored by the mainstream DBSs. But, it appears now that all adopted it. However, curiously, all disguised the function under names different from the original. Also surprisingly, we could not locate any publicly available research article or technical report about any of those offerings. We could only see the grey literature: reference manuals, blogs, tutorials, forums…*

Facts. In 1993-94, we proposed a new aggregate function for SQL Database Systems (DBSs) called LIST, [1], [2] and, lastly, [3]. Pdfs and even the slides of my talks about are still on DBPL, CiteseerX… LIST transformed selected sets of values produced by a query, assimilated each to a set of strings, into lists of these values, one per set, assimilated then each to a single string. Through this formal twist, the result could fit the 1NF constraint of the relational model, i.e., of the atomic attribute values only, we recall. In practice, the result of a query retrieving groups of (possibly several) tuples could get then aggregated into a single tuple per group. E.g., with LIST function, the tuples (Book title, Author) grouped by title in a query could become, for every group, a single tuple with one title and all the authors. Analogously, for char type attributes, to what SUM, AVG… functions did for numerical ones. The same query, but without LIST, for a book with four authors, would output at least four (Book title, Author) tuples, with the title at least four times uselessly repeated.

Our proposal was theoretical only. But, even just through our example, the utility of LIST function seems obvious. We hoped for its rapid appearance in mainstream DBSs. I.e., DB2, Oracle, SQL Server, MySQL, PostgreSQL,… We pointed out in [3], as the "bait", that Version 9.0.1 of Sybase coming by then, should propose LIST, although with limitations. Sybase delivered, but not its popularity hoped for. The product became somehow marginal under SAP ownership. Besides, we did not spot any info from any major. Eighteen years passed, making hopes for LIST's popularity seemingly sadly dead.

On May 5, 2022 however, unrelated googling brought a reference to PostgreSQL aggregate function named string_agg. Surprisingly, we were not aware of it. Also, its sole name rang familiar bells. Indeed, string_agg turned out to be a specific implementation of LIST, despite syntactical differences and limitations. Our infructuous quests were for LIST related info. No idea of googling different names. What names should we google, besides?

From that moment, we started intensive "forensic analysis", googling everything related to string_agg. The current outcomes are as follows. The function itself apparently appeared in PostgreSQL V9, with numerous successive releases between 2010 and 2015. Before, PostgreSQL V8.4, 2009-14, introduced a more general function termed array_agg. It seems that the latter was uselessly procedural for strings that constituted the bulk of the practical cases. SQL-Server picked up the string_agg name, but not the entire syntax, in 2017. Strangely somehow, U-SQL of Azure Cloud DBS, picked up array_agg instead.

MySQL, likely since 2010, preferred to call LIST as Group_Concat. SQLite liked the name, apparently since 2013 or perhaps, since 2017. There are nevertheless differences in the syntax with respect to MySQL. Although it owns MySQL, Oracle itself preferred to name LIST as LISTAGG, introducing the function in version 11g release 2, since Sept. 2009. In its SQL 2 product it appeared however, under the same name, in 2016 only. Noticeably, only since Oracle 19c, so since Feb. 2019 at earliest, LISTAGG with SQL Distinct keyword became available, while it was already, i.e., fifteen years earlier in our original proposal. In the meantime DB2 caught up with LISTAGG, with some syntactical differences, in 2016-17.

Then, Google's BigQuery implemented LIST variants called array_agg, array_concat_agg and string_agg in its, so-called Standard SQL, likely since 2017. Before, they used so called now Legacy SQL with group_concat named function instead. This one had more limitation, e.g., no Order By option. Actually, Vertica proposed the function naming it LISTAGG, likely in 2017, with the same limitation till the time of this write-up.

Next, MariaDB picked up Group_concat. We could not find since when. But, Snowflake opted for LISTAGG. Like did Amazon AWS Redshift. Like did also Exasol 7.1, since 2022 apparently. Noticeably, SQL Standard ISO 9075 Features opted for Array-agg and LISTAGG. String_agg and Group_concat are mentioned there as vendor renamed variants of LISTAGG. Perhaps because of all that "distinction", the latter name entered even "SQL for Dummies".

The googling for all the above info brought hundreds of references. All pointed to the grey literature on user interface only: reference manuals, forums, blogs, tutorials…. No publicly available technical article appeared, whether from research or from the industry. In particular, no reference pointed to any company literature comparing the proposal to any related or previous work. Only some blogs discussed the portage of specific queries from one DBS to another and troubles with such exploits.

Analysis. On the bright side: LIST attained its intended universal popularity. This occurred, curiously through different from the original and heterogeneous function naming among mainstream DBSs. Besides, even for the same naming, details varied among DBSs. Some of the details evolved with successive versions of the DBS.

On the dark side, none of the references we looked upon had any pointers to our LIST proposal. Given even the disguised naming, we can hardly expect folks in the DB arena aware of the former. Also regretfully, no reference brought any knowledge of any implementation. The latter seems the "company's secret" for everyone. So, only our early analysis of the issue in [3] is publicly available.

Finally, on the same "dark" side, some capabilities proposed for LIST remain still unused. E.g., this concerns, the so-called implicit LIST in [3], despite the least procedurality of this option whenever possible. Also, some industrial choices seem uselessly procedural. E.g., it is the case of WITHIN GROUP keywords in Oracle's version of Order By for LIST. Notice that Oracle's syntax was picked up also by some other present offerings.

In fact, even every sheer "disguise name" chosen by a mainstream DBS is more procedural, i.e., longer than the original. Oracle's LISTAGG's choice appears besides especially surprising. SQL names for more "classical" aggregates are indeed SUM, AVG… (and LIST), and not the more procedural SUMAGG, AVGAGG… The name Group_concat seems also comparatively somehow cryptic. In any case, we did not find any justification for any naming proposed.

Conclusion. Applying non-patented research without indicating the source is a common industrial practice, including the open-source industrial offerings. Whether it is fortunate, earnest, or even somehow honest is a different debate. We recall nevertheless that not being aware of the previous/related work is hardly an excuse in the research arena. Thus if, say, an Oracle team attempted to publish in a decent research conference or journal an article proposing LISTAGG aggregate function and the submission was declined as not referring to the LIST function, claiming the absence of knowledge of the latter would, likely, not help the authors. Perhaps a similar constraint should be required from the new industry offerings. Recall that it is so for the patents. For cases like ours, the constraint could be, e.g., that the provider of an industrial offering omitting in its publicly available documentation the previous/related research or even industrial work, especially the easily accessible one, through Google, Citeseer, Dbpl, ACM Library…, should be liable of some monetary penalty to the omitted. Both, the products and the public should benefit from. Besides, our case is the only we're aware of, so predominantly disguised under different naming. We hope the rationale for this curious fate appearing one day.

Acknowledgements. *We thank Ron Fagin for help with access to the public IBM technical documentation.*

References

[3] Litwin, W. Explicit and Implicit List Aggregate Function for Relational Databases. IASTED Intl. Conf. On Databases and Applications. Feb. 2004.
[2] Litwin, W. Case for Explicit and Implicit LIST Aggregate Function for Relational Databases. CERIA Research Report 2003-10-01, Oct. 2003, pdf.
[1] Litwin, W. The LIST Aggregate Function for Relational Databases. CERIA Research Report 2003-06-09, June 2003, pdf.