

# The ATLaS System and its Powerful Database Language Based on Simple Extensions of SQL (Extended Abstract)

Haixun Wang  
IBM T. J. Watson Research Center  
Hawthorne, NY 10532  
haixun@us.ibm.com

Carlo Zaniolo  
University of California  
Los Angeles, CA 90095  
zaniolo@cs.ucla.edu

## Abstract

A lack of power and extensibility in their query languages has seriously limited the generality of DBMSs and hampered their ability to support new applications domains, such as datamining. In this paper, we solve this problem by stream-oriented aggregate functions and generalized table functions definable by users in the SQL language itself—rather than in an external programming language. These simple extensions turn SQL into a powerful database language, which can express a wide range of applications, including recursive queries, ROLAP aggregates, time-series queries, stream-oriented processing, and datamining functions.

## 1 Introduction

Current DBMSs do not offer a good basis for building advanced database applications because of two limitations of SQL. The first is the lack of expressive power that still cripples SQL after so many years (and millions of lines of code) of extensions, to support recursive queries, ROLAP aggregates, triggers, datablades, and many others. The problems of SQL in supporting data mining algorithms (stream-oriented or otherwise) is particularly well-documented [1]. The second problem is that SQL contains many blocking constructs, in particular set aggregates, that are now considered unsuitable for stream-oriented processing [3, 2]. With ATLaS, we demonstrate how to make SQL much more powerful, and conducive to stream-oriented processing, by supporting User Defined Aggregates and Table functions (ATLaS stands for Aggregate & Table Language and System). ATLaS' User Defined Aggregates (UDAs) implement a stream-oriented computation model, whereby UDAs accept a stream as input and produce a stream as output. They can express both traditional blocking aggregates and non-blocking aggregates—such as online aggregates [4] and the continuous aggregates used for time series [9]—in a syn-

tactic framework that is very conducive to stream-oriented processing. Finally, UDAs are defined in SQL itself (rather than in external procedural languages required for new functions in current O-R systems). This closure property is the source of great power and flexibility, and the same property holds for ATLaS generalized table functions, which are also defined in SQL.

ATLaS adopts SQL3 idea of specifying user-defined-aggregates by an *initialize*, an *iterate*, and a *terminate* computation [6, 5]<sup>1</sup>. Then it takes from AXL [11] the idea of defining these three computations by a single procedure written in SQL (rather than three written in procedural languages as proposed in SQL3). We will now illustrate ATLaS using some simple examples.

A well-known problem in temporal databases is how to support the operation of interval coalescing. It is simple to express this computation in ATLaS.

### Example 1 *Coalescing*

```
AGGREGATE coalesce(from TIME, to TIME);
      (start TIME, end TIME)
{  TABLE state(cFrom TIME, cTo TIME);
   INITIALIZE: {
     INSERT INTO state VALUES(from,to); }
   ITERATE :{
     UPDATE state SET cTo = to
       WHERE cTo >= from AND cTo < to;
     INSERT INTO RETURN
       SELECT cFrom, cTo FROM state
       WHERE cTo < from;
     UPDATE state
       SET cFrom = from, cTo = to
       WHERE cTo < from; }
   TERMINATE: {
     INSERT INTO RETURN
       SELECT cFrom, cTo FROM state; } }
```

<sup>1</sup>Although UDAs have been left out from SQL-99 specifications, they were part of early SQL3 proposals, and supported by some commercial DBMS [7].

The UDA `coalesce` of Example 1 takes two parameters: `from` is the start time, and `to` is the end time. Under the assumption that tuples are sorted by increasing start time, then we can perform the task in one scan of the data. In the `ITERATE` routine, we merge the new interval with the current interval, which is kept in table `state`, whenever the two overlap. Otherwise, the current interval is returned, while the new interval becomes the current one.

ATLaS queries also make extensive use of table functions, which add significant flexibility to SQL [8] and dovetail perfectly with UDAs. For instance, let us consider the table function `dissemble` that is particularly useful in expressing classifiers in SQL. Take for instance the well-known play-tennis example in Table 1; here we want to classify the value of `Play` as a ‘Yes’ or a ‘No’ given a training set such as that shown in Table 1.

RID	Outlook	Temp	Humidity	Wind	Play
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	Yes
7	Overcast	Cool	Normal	Strong	No
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Table 1: The relation `PlayTennis`

The first step for most scalable classifiers [10] is to convert the training set into column/value pairs. This conversion, although conceptually simple, is hard to express succinctly in SQL. Consider the `PlayTennis` relation as shown in Table 1. We want to convert `PlayTennis` into a new stream of three columns (`Col`, `Value`, `YorN`) by breaking down each tuple into four records, each record representing one column in the original tuple, including the column number, column value and the class label `YorN`.

In ATLaS, we can define a new table function `dissemble` to solve the problem.

**Example 2** *Dissemble a relation into column/value pairs.*

```

AGGREGATE dissemble(a1 INT, a2 INT, a3 INT, a4 INT, C INT) :
    (col CHAR(20), class INT)
{
  INITIALIZE: ITERATE: {
    INSERT INTO RETURN VALUES
      (a1, C), (a2, C), (a3, C), (a4, C), ('-all-', C);
  }
}

```

**Example 3** *Compute all counts() in one pass of the data.*

```

SELECT t.col, t.class, count(*)
FROM (SELET dissemble(Outlook, Temp, Humidity, Wind, Play)
      FROM PlayTennis) AS t
GROUP BY t.col, t.class;

```

ATLaS extends the previous AXL system [11] with significant enhancements; the two most important ones are table functions declared in SQL, and in-memory tables with OIDs. These extensions allowed us to pass by the litmus test of writing an efficient implementation of the Apriori algorithm in SQL [1]. In fact, the ATLaS implementation of Apriori matches in performance C++ implementation with a 30% overhead. Furthermore, ATLaS demonstrates the benefits of having a native extensibility mechanism for SQL, whereby new application domains can be supported efficiently without touching the SQL standards. We demonstrated this point by revisiting the major SQL extensions, such as ROLAPs and recursive queries, that required the introduction of new special constructs by standard committees and showing that these new functions can be supported efficiently in ATLaS without any new construct.

## References

- [1] R. Agrawal, R. Srikant. “Fast Algorithm for Mining Association Rules”. In *VLDB’94*.
- [2] Shivnath Babu and Jennifer Widom. Continuous Queries over Data Streams, *SIGMOD Record*, Vol. 30 No. 3, pp. 109-120 (September 2001).
- [3] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. Niagara: A scalable continuous query system for internet databases. In *Proc. of the 2000 ACM SIGMOD Intl. Conf. on Management of Data*, pages 379-390, May 2000.
- [4] J. M. Hellerstein, P. J. Haas, H. J. Wang. “Online Aggregation”. *SIGMOD*, 1997.
- [5] ISO/IEC JTC1/SC21 N10489, ISO/IEC 9075, “Committee Draft (CD), Database Language SQL”, July 1996.
- [6] ISO DBL LHR-004 and ANSI X3H2-95-364, “(ISO/ANSI Working Draft) Database language SQL3”, Jim Melton (ed), dated 1995.
- [7] Informix: Datablade Developers Kid InfoShelf, Informix 1998, <http://www.informix.co.za/answers/english/docs/dbdk/infoshelf/index.html>
- [8] B. Reinwald et al.: Heterogeneous Query Processing through SQL Table Functions. *ICDE 1999*: 366-373
- [9] Reza Sadri, Carlo Zaniolo, Amir M. Zarkesh, Jafar Adibi: A Sequential Pattern Query Language for Supporting Instant Data Mining for e-Services, *VLDB 2001*, pages 653-656.
- [10] J. C. Shafer, R. Agrawal, M. Mehta, “SPRINT: A Scalable Parallel Classifier for Data Mining,” In *VLDB 1996*.
- [11] Haixun Wang and Carlo Zaniolo: Using SQL to Build New Aggregates and Extenders for Object-Relational Systems. *VLDB 2000*.