# WUM: A Web Utilization Miner

Myra Spiliopoulou
*Institut für Wirtschaftsinformatik, HU Berlin*
http://www.wiwi.hu-berlin.de/~myra
myra@wiwi.hu-berlin.de

Lukas C. Faulstich *
*Institut für Informatik, FU Berlin*
http://www.inf.fu-berlin.de/~faulstic
faulstic@inf.fu-berlin.de

## Abstract

Most web sites are set up with little knowledge on the navigational behaviour of the users accessing them. Feedback on the occurring navigation patterns can significantly aid site owners in efficiently (re)organizing the hyperspace they present to their visitors.

In this study, we present the Web Utilization Miner WUM, a mining system for the discovery of interesting navigation patterns. The interestingness criteria for navigation patterns are dynamically specified by the human expert using WUM's mining language MINT. MINT supports the specification of criteria of statistical, structural and textual nature. To discover the navigation patterns satisfying the expert's criteria, WUM exploits an innovative aggregated storage representation for the information in the web server log.

## 1 Introduction

Web sites are most often organized in a way the providers consider appropriate for the majority of the site's visitors. However, our knowledge of the actual navigational behaviour of the visitors is still sparse and fragmentary. Simple access statistics provide only rudimentary feedback, while studies on specific behavioural patterns, e.g. page revisits [TG97], are of rather ad hoc nature.

Knowledge about the navigation patterns occurring in or dominating the usage of a web site can greatly help the site's owner or administrator in improving its quality. Data mining can assist in this task by effectively extracting knowledge from the past, i.e. from the site access recordings. In [CMS97b], the term "web usage mining" is suggested to describe this type of mining activity.

To aid an expert in reorganizing a web site, a web usage miner should provide feedback on (i) access to certain nodes and paths considered of importance, (ii) nodes and paths preferred or avoided by the visitors and (iii) generic traversal behaviour, e.g. trends for backward moves, cycles etc. Miners generating sequential patterns that satisfy certain support or confidence thresholds are not adequate. Rather, we need a tool that can identify navigation patterns satisfying properties specified by the expert in an ad hoc manner. Those properties can concern the statistics or the contents of the pattern but may be as vague as the existence of cycles or the repeated access to some, otherwise undefined, node.

The web miner proposed in [CPY96] simply discovers statistically dominant paths. The "WEBMINER" tool of [CMS97b] provides a query language on top of external mining software for association rules and for sequential patterns. However, the expressiveness of the language is restricted by the input parameters acceptable by the miner. To the best of our knowledge, current miners do not support generic specifications on the structure of the patterns to be discovered, e.g. page revisits, cycles etc.

In this study, we present a mining mechanism that avoids those shortcomings, by incorporating the query processor to the miner. Our Web Utilization Miner WUM employs an innovative technique for the discovery of navigation patterns over an aggregated materialized view of the web log. This technique offers a mining language as interface to the expert, so that the generic characteristics can be given, which make a pattern interesting to the specific person. Thus, only patterns having the desired characteristics are constructed, while uninteresting patterns are pruned out early. The focus of this study is on the mining language, MINT.

In the next section, we present the architecture of WUM. Section 3 describes the process of aggregating web log data into an Aggregated Log, on which mining is applied. In section 4 we describe our mining language MINT and its processing mechanism. Relevant literature is presented in section 5. We conclude in section 6.

## 2 The Architecture of WUM

The architecture of our Web Utilization Miner is depicted in Fig. 1. There are two major modules: the *Aggrega-*

*Supported by the German Research Society, Berlin-Brandenburg Graduate School in Distributed Information Systems (DFG grant no. GRK 316).

*tion Service* prepares the web log data for mining and the *MINT-Processor* does the mining.

The *Aggregation Service* extracts information on the activities of the users visiting the web site and groups consecutive activities of the same user into a transaction. It then transforms transactions into sequences. Its major task is to merge those sequences into a trie structure, on which aggregated statistical information is retained. This process is described informally in the next section.

The *MINT-Processor* mines the aggregated data according to the directives of the human expert. "MINT" is the mining language serving as interface between the user and the miner. The expert uses MINT to instruct the miner on the formulation of the output, and, most importantly, on the interestingness criteria to be satisfied by the desired patterns.

The *MINT-Processor* is the mining core of WUM, responsible for the discovery of navigation patterns from the aggregated information extracted by the *Aggregation Service* from the weblog. The *MINT-Processor* can be invoked in two modes: As *Notifier*, it executes preprocessed queries periodically. As *Explorer*, it accepts ad hoc queries.

The purpose of the *Notifier* is to discover whether the web access data show deviations from the expected usage. Hence, the "alert queries" it executes should correspond to the beliefs of the site's owner on the statistics and structure of the dominant navigation patterns. If those beliefs are not satisfied, the user should be alerted to invoke the *Explorer*, which can help in discovering the actually dominant navigation patterns.

We would like to stress here that WUM, similarly to most miners, is not an expert system. The expertise belongs to its user and is indispensable, on the one hand to describe what is interesting to her, and on the other hand to avoid misinterpretations of the results. In particular, the user should be familiar with the site's content and its intended use. She must possess expertise on human behaviour in hypermedia environments, so that she may transform the expectations of usage into mining directives and correctly interpret the mining results; these issues are addressed e.g. in [TG97, Wex97]. Finally, she should own the appropriate background to properly interpret the statistics of the results.

# 3 Aggregating Visitor Transactions

The discovery of navigation patterns in WUM is performed on the basis of the information extracted from the web server log after some (currently simple) data cleaning and inserted into the "Aggregated Log". The Aggregated Log contains *aggregated* data on the visitor trails in the site, where a "visitor trail" is a sequence of page requests comprising a visitor transaction.

We support two criteria for grouping consecutive page requests of a visitor into a transaction: (i) a maximal duration or (ii) a maximal elapsed time between any two subsequent page accesses. Other criteria, as proposed e.g. in [CMS97a] can also be incorporated.

Deciding whether two subsequent accesses to web pages stem from the same visitor is not a trivial task. Methodologies based on the usage of cookies, user registration, exploitation of knowledge on the network's topology etc have been proposed. A thorough discussion appears in [CMS97b]. Currently, our *Aggregation Service* assumes that accesses from the same host come from the same visitor. We are intending to incorporate a more sophisticated mechanism, though.

**Aggregate Trees.** The *Aggregation Service* of WUM extracts the visitor trails from the web log and aggregates them by merging trails with the same prefix into a tree structure, the "aggregate tree".

An aggregate tree is a trie, a node of which corresponds to the occurence of a page in a trail. Common trail prefixes are identified, and their respective nodes are merged into a trie node. This node is annotated with the number of visitors having reached the node across the same trail prefix. We call this the "support" of the node.

**Example 1:** At the left side of Fig. 2 we show the topological graph of a tiny web site and a number of trails recorded in the web log. Along with each trail we show the number of visitors that have followed it.

In Fig. 2, `a` is the first page of trails 1, 4 and 5; by summing up the visitors having traversed them, we compute 21 as the support of `a`. In trails 1, 4, page `b` was visited after `a`; the respective trie node has a support of 11. Note that trails 2, 3, 6 starting at `b` cannot be merged with trails 1 and 4 starting at `a`.

In trails 2 and 6, page `b` has been accessed twice. When constructing the aggregate tree, we see that a total of 13 visitors accessed `b` (via trails 2, 3, 6), but only 6 of them came back to `b`, across trails 2 and 6 that have the same prefix. We must distinguish between different occurences of the same page, so that page revisits can be discovered. Hence, during trail merging, the *Aggregation Service* assigns to each page its occurence number.

The aggregate tree of Fig. 2 has a dummy node ^ as root. This allows us to model the Aggregated Log as a single large aggregate tree. The support of ^ is the total number of entries recorded in the weblog.

**Navigation Patterns.** On Fig. 2 we can see that trails with different prefixes may still have pages in common. We construct navigation patterns coercing at those common pages by discovering the respective branches of the

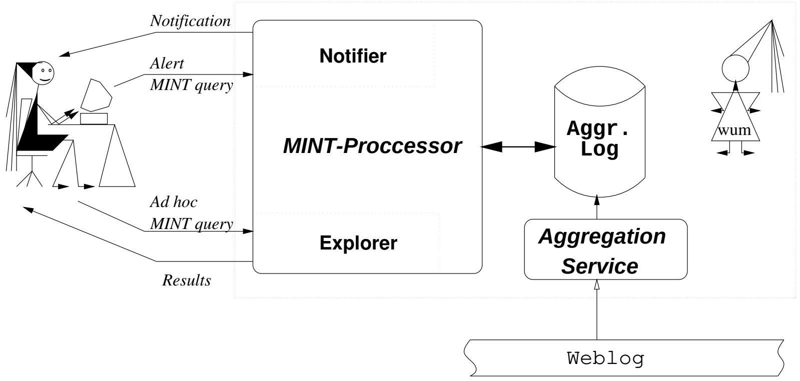


Figure 1: The architecture of WUM

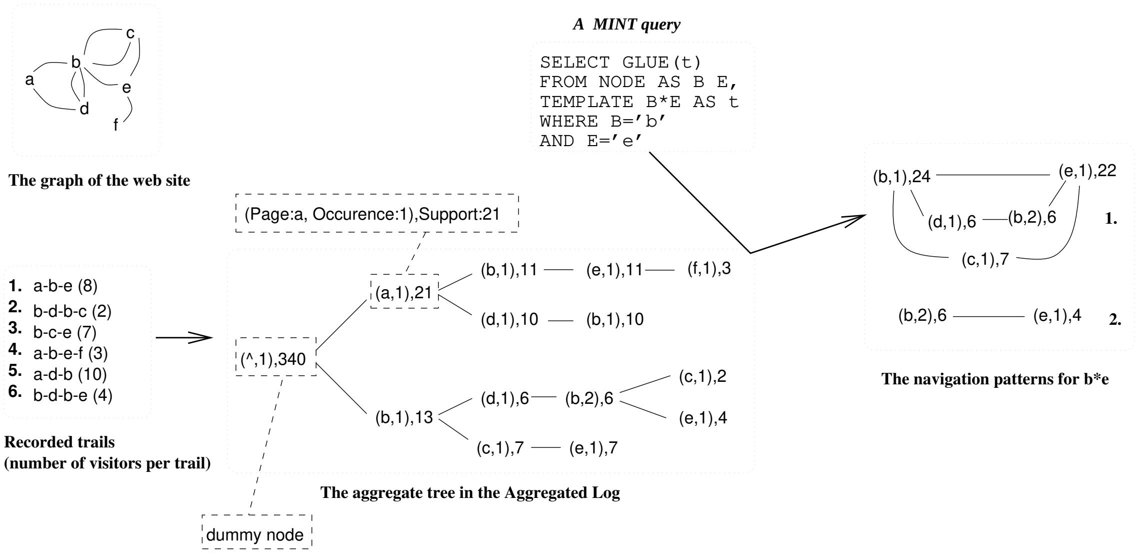


Figure 2: Merging trails in the Aggregated Log

aggregate tree, merging them and recomputing the support of the junction nodes.

For example, the two navigation patterns between `b` and `e` are shown at the right side of Fig. 2. Pattern 1 shows that 22 of the 24 visitors of `(b,1)` have reached `e` across different routes. 6 of them returned to `b`. Only 4 of those 6 reached `e` afterwards. This last fact cannot be deduced from pattern 1; it is assessed from pattern 2.

Above the two navigation patterns, we show the MINT query that produced them. We present MINT in the next section.

**The schema of the Aggregated Log.** The conceptual schema of the Aggregated Log is depicted in Fig. 3.

For a `Page` in the web site we retain meta-information gained from the HTML-file, the URL of the page and the total number of `accesses` to the page. This number is independent of the trails followed to reach the page and can also be computed as the summation of the `support` values of the aggregate tree nodes in the Aggregated Log that refer to this page. We retain this redundant property on reasons of efficiency.

A `Node` corresponds to the page, to which it points. A page can be accessed more than once within the trail of a visitor, so a branch of the aggregate tree can be cyclic. Since we cannot model cycles on an aggregate tree without information loss, we attach to each node an `occurence` number. This number can be queried in MINT, so that repeated accesses to pages can be discovered. A node is uniquely identified by the page it corresponds to and the `occurence` of the page in the branch of the aggregate tree, where the node belongs.

A node exists only in the context of the graph structure to which it belongs. Its `support` value is computed in the context of its predecessors in the graph. The class `AgGraph` models the graph structure of aggregate trees and navigation patterns. The persistent large aggregate tree of the Aggregated Log and the aggregate trees and navigation patterns generated as query results on the fly have the same properties and graph structure.

# 4 Knowledge Discovery Queries

Similarly to [FKZ97], we believe that good mining results require a close interaction of the human expert and the mining tool, in which the expert uses her domain knowledge to guide the miner. Therefore, WUM provides a mining query language, with which the expert can specify the subjective characteristics that make a navigation pattern of interest to her.

The notion of interestingness based on beliefs is discussed in [ST96]: a belief is a rule of the form $A \to B$, which is expected to be true. The same study proposes mechanisms for the verification of beliefs and the discovery of belief violations *in the context of association rules*. To the best of our knowledge, there is no respective formalism for beliefs on sequential patterns. However, MINT allows the specification of beliefs or belief violations as predicates. Predicates can also be used to specify the structure or statistics a navigation pattern should have to be of significance. Thus, besides the classical mining criterion of a support threshold, much more elaborate criteria are supported.

## 4.1 MINT and its Miner

A MINT query is evaluated against the schema of Fig. 3.

**Obtaining simple statistical information.** An important question on web navigation is: "where do visitors of page a go afterwards?" Some restrictions are often placed on content and support of a and of the subsequent pages, as in the following query **Q1**:

```
SELECT a.url, b.url
FROM NODE AS a b, TEMPLATE a*b
WHERE a.support > 100
AND a.title LIKE "%Corba%"
AND b.support / a.support > 0.1
```

Here, we receive all URLs reached after a page a on Corba, which has been accessed 100 or more times, provided that those URLs have been accessed by at least 10% of the visitors of a. Note that the syntax hides part of the schema complexity from the user.

Conceptually, **Q1** is executed as follows: All nodes in the Aggregated Log whose title refers to CORBA are retrieved. All subtrees rooted at each such node are identified, their common prefixes are merged as described in section 3, and the supports of the nodes of the new tree are computed. For those nodes on Corba that have a support larger than 100, the nodes subsequently visited by at least 10% of the visitors are identified; all other subtrees are pruned away. Finally, for the required properties of the remaining nodes are returned.

For each subtree extracted during query processing, the supports of a and b must be recomputed: Consider two subtrees $X, Y$ whose roots $a_X, a_Y$ refer to the same page $P$ on Corba and have the same occurence number, so that they can be merged. The support of $a_X$ is 50 and of $a_Y$ is 60. Their merging produces a node $a_0$ having been accessed 110 times, 50 in the context of $a_X$ and 60 in the context of $a_Y$. $a_0$ is the root of a tree $T$ produced by merging $X, Y$, similarly to the procedure shown in Fig. 2.

**Obtaining new aggregate trees.** Queries returning only statistical information for some pages do not give any insight on how those pages have been accessed. Query **Q2** returns this information using the same predicates as in **Q1**:

```
SELECT t
FROM NODE AS a b, TEMPLATE a*b AS t
WHERE a.support > 100
AND a.owner LIKE "%Corba%"
AND b.support / a.support > 0.1
```

For each value $a_0$ of a, **Q2** returns the tree rooted at $a_0$ and consisting of the paths ending at values $b_1, \ldots, b_n$ for b, such that the pairs $(a_0, b_1), \ldots, (a_0, b_n)$ satisfy the predicates of **Q2**. Note that the query template has been assigned a name t, so that it can be referenced in the output clause.

In general, each node variable in a template obtains values in the context of the variables on its left and provides the context for the variables on its right. For each binding of a variable, the query processor builds the subtree rooted at it and containing all permissible values of the remaining variables in the root's context. Templates of the form *a obtain their context from the dummy node ^.

**Combining aggregate trees into a navigation pattern.** A query returning simply the template on which its predicates are applied produces a tree per value of the leftmost node variable, containing a subtree for each other variable value in this context. However, nodes in different subtrees may still refer to the same occurance of the same page and can thus be merged, as in query **Q3**:

```
SELECT GLUE(t)
FROM NODE AS a b, TEMPLATE a*b AS t
WHERE a.support > 100
AND a.owner LIKE "%Corba%"
AND b.support / a.support > 0.1
```

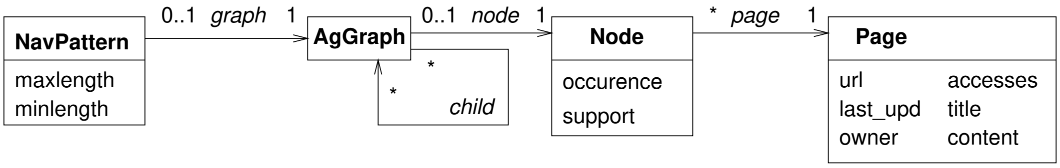


Figure 3: The schema of the Aggregated Log (using UML notation)

For each value of a, the GLUE operator combines all nodes with the same b value into one junction point, whose support is the summation of the original nodes' supports. Hence, GLUE is the operator that produces navigation patterns, as specified in section 3 (see also Fig. 2).

Grouping nodes are also permitted. The clause "GROUP BY a" would compute the summation of the supports of all distinct values of b within the context of a given a value. MINT also allows conventional aggregates. The full syntax is given in the Appendix.

**Revisited nodes.** The study of [TG97] showed that 58% of accesses in the studied domain were node revisits, also called "recencies". For a given site, this statement can be verified by issuing a query that sums up the support of all revisited nodes and compares it to the total number of accesses. Recall that this number is the support of the dummy node ^. This test is performed in the following query **Q4**:

```
SELECT SUM(a.support) / ^.support
FROM NODE AS a, TEMPLATE *a
WHERE a.occurence = 2
```

Note that the occurence of a revisited node must be exactly 2. Otherwise, nodes revisited more than once would contribute to the support summation multiple times.

The study of [TG97] also states that recencies to any *specific* node do not exceed 19% of the total accesses. To find nodes revisited more often than that we can issue query **Q5**:

```
SELECT b
FROM NODE AS a b, TEMPLATE *a*b
WHERE a.page = b.page
AND a.occurence = 1
AND b.occurence = 2
AND b.support / a.support > 0.19
```

### 4.2 Complexity of the Mining Process

The *MINT-Processor* is responsible for identifying common patterns in the large aggregate tree of the Aggregated Log, merging them to aggregate graphobjects, computing the node supports and evaluating the query predicates. Any navigation patterns not satisfying the predicates should be recognized and filtered out as soon as possible.

This mechanism can be quite expensive: A query asking for all navigation patterns containing nodes with supports beyond a threshold will imply considering all possible variable bindings. We are working on optimization techniques to alleviate this problem. One of them is to retain for each web page a list of the nodes corresponding to it, thus trading speed for space.

Still, our *MINT-Processor* is de facto faster than a miner operating on web log data [CMS97b, CPY96], because of the off line aggregation of weblog information into the Aggregated Log. Moreover, our mechanism can exploit restrictive predicates to avoid building uninteresting navigation patterns. A miner not accepting ad hoc predicates needs a postprocessing module that removes all unnecessarily computed patterns.

## 5 Related Work

The discovery of navigation patterns in web logs has been studied in [CPY96, CMS97b, Wex96]. The "Footprints" tool of [Wex96] focusses on the visualization of frequently accessed patterns and on the identification of pattern *types* that may be of importance [Wex97]. Our work on MINT is complementary to Footprints, since we are interested in ways of formally describing such types.

The studies of [CPY96, CMS97b] focus on pattern discovery. They are both based on the adaptation of an existing miner to the particular problem. More precisely, a preprocessing algorithm groups consecutive page accesses of the same visitor into a transaction, according to some criterion. Then, a miner for association rules or sequential patterns is invoked to discover similar patterns among the transactions. In [CMS97b], the association rules' miner has been further customized to guarantee that no feasible patterns are erroneously skipped.

This approach has some drawbacks. First of all, the generic characteristics that make a navigation pattern interesting are different than those making an association rule interesting, and which are studied e.g. in [ST96]. Interestingness in the mining of sequential patterns is expressed in rather simple criteria on the support/confidence and length of the patterns [MT96]. Hence, the miners used in [CPY96, CMS97b] do not allow an expert to guide the mining process to discover e.g. node revisits, patterns of unexpectedly low support etc.

This problem is alleviated in [CMS97b] by the introduction of a query language, with which the expert can give instructions to the miner. The language of [CMS97b] is syntactically similar to MINT but, in contrast to MINT, it is only a front end to an existing miner. Thus, instructions that cannot be understood by the miner must be postponed to a postprocessing phase: the interestingness of some plans cannot be tested prior to this phase. This decreases the overall system performance.

A further performance drawback of miners on sequential patterns [AS95, MT96] is caused by activating them over the whole set of transactions. By aggregating transactions into an Aggregated Log, WUM guarantees a performance improvement that is at least linear to the degree of similarity between transaction prefixes in the original log.

Our work is conceptually related to the study of [FKZ97] for text mining. A query language is introduced to drive a miner that discovers association rules in document collections. This miner is applied on *aggregated data*, organized in a trie structure [AFK97] in a way similar to our Aggregated Log. The performance gains of using such an aggregated representation are demonstrated in [FKZ97].

However, the exploitation of the aggregated information in [AFK97] is different than in WUM: Association rules have no order, so any rule can be discovered by traversing a single subtree of the trie. Web navigation patterns have order, so that the discovery of all matching branches of the large aggregate tree in our Aggregated Log are necessary. Thus, the *MINT-Processor* uses a much more complicated technique than a simple trie traversal.

# 6 Conclusions

We have presented an overview of the Web Utilization Miner, our system for the discovery of interesting navigation patterns in a web site. We have primarily focussed on the MINT query language and its execution mechanism. MINT can be used by a human expert to guide the mining system by specifying the generic structural and statistical characteristics that make a navigation pattern interesting.

WUM is intended for web site authors and administrators who are trying to improve the organization of their web documents and adapt it better to the needs of the information consumers. This applies to research and educational institutions, but also to other organizations, e.g. public authorities providing citizens with environmental information.

Of the modules of WUM, we have implemented the *Aggregation Service* and are now developing the MINT query processor and a simple visualization tool for presenting the discovered navigation patterns to the user. Our current work focusses on the experimentation with WUM, on optimization techniques for *MINT-Processor* to reduce the complexity of pattern discovery, and on the formulation of beliefs for navigation patterns. The *Notifier* module of WUM, intended to periodically execute preprocessed queries, will then be designed to discover belief violations.

# References


[AFK97] Amihood Amir, Ronen Feldman, and Reuven Kashi. A new and versatile method for association generation. *Information Systems*, 22:333–347, 1997.

[AS95] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *ICDE*, Taipei, Taiwan, Mar. 1995.

[CMS97a] Robert Cooley, Bamshad Mobasher, and Jaideep Srivastava. Grouping web page references into transactions for mining world wide web browsing patterns. Technical Report TR 97-021, Dept. of Computer Science, Univ. of Minnesota, Minneapolis, USA, June 1997.

[CMS97b] Robert Cooley, Bamshad Mobasher, and Jaideep Srivastava. Web mining: Information and pattern discovery on the world wide web. In *ICTAI'97*, Dec. 1997.

[CPY96] Ming-Syan Chen, Jong Soo Park, and Philip S Yu. Data mining for path traversal patterns in a web environment. In *ICDCS*, pages 385–392, 1996.

[FKZ97] Ronen Feldman, Willi Klösgen, and Amir Zilberstein. Visualization techniques to explore data mining results for document collections. In *KDD'97*, pages 16–23. AAAI Press, 1997.

[MT96] Heikki Mannila and Hannu Toivonen. Discovering generalized episodes using minimal occurences. In *KDD'96*, pages 146–151, 1996.

[ST96] Avi Silberschatz and Alexander Tuzhilin. What makes patterns interesting in knowledge discovery systems. *IEEE Trans. on Knowledge and Data Eng.*, 8(6):970–974, Dec. 1996.

[TG97] Linda Tauscher and Saul Greenberg. Revisitation patterns in world wide web navigation. In *CHI'97*, Atlanta, Georgia, Mar. 1997.

[Wex96] Alan Wexelblat. An environment for aiding information-browsing tasks. In *Proc. of AAAI Spring Symposium on Acquisition, Learning and Demonstration: Automating Tasks for Users*, Birmingham, UK, 1996. AAAI Press.

[Wex97] Alan Wexelblat. *Footprints: History-Rich Social Navigation*. PhD thesis, MIT Media Laboratory, Dec. 1997.


## A The Syntax of MINT

```
query::=
   ’SELECT’ selectList
   fromClause [whereClause]
   [groupClause [havingClause]]
selectList ::=
   [’DISTINCT’] derivedColumn
   (’,’ derivedColumn)*
derivedColumn ::=
   (valueExpr|aggrExpr)
   [’AS’ columnName]
aggrExpression ::=
   aggrOp ’(’ [’DISTINCT’]
        (valueExpr|varName) ’)’
aggrOp ::=
   ’AVG’ | ’MAX’ | ’MIN’ |
   ’SUM’ | ’COUNT’ | ’GLUE’
fromClause ::=
   ’FROM’ tableRef (’,’ tableRef)*
tableRef ::=
   ’NODE’ ’AS’ nodeVar* |
   ’TEMPLATE’ template
   [’AS’ templateVar ]
template ::=
   [’*’] (nodeVar [’*’])*
varName ::=
   nodeVar|templateVar
whereClause ::=
   ’WHERE’ condition
   (’AND’ condition)*
condition ::=
   valueExpr compOp valueExpr
compOp::= ’=’ | ’<’ | ’>’ |
   ’<=’ | ’>=’ | ’LIKE’
valueExpr ::=
   numericExpr | stringExpr
numericExpr ::=
   [numericExpr (’+’|’-’)] term
term ::=
   [term (’*’|’/’)] factor
factor ::=
   [(’+’|’-’)] primary
primary ::=
   literal | columnReference |
   ’(’ valueExpr ’)’
stringExpr ::=
   [stringExpr ’||’] primary
columnReference ::=
   varName ’.’ columnName
groupClause ::=
   ’GROUP’ ’BY’ groupExpr
   (’,’ groupExpr)*
groupExpr ::=
   nodeVar | columnRef
havingClause ::=
   ’HAVING’ condition
   (’AND’ condition)*
```