Tool giving intuitive and analytical index suggestions

A Project Report

Presented to

Professor Ramin Moazenni

Department of Computer Science

San José State University

In Partial Fulfilment

Of the Requirements for the Class

CS 257

By

Jayant Shelke

Mustafa Badshah

Muthaiah Ramanathan

November 2017

**Abstract:**

The relational databases have been under use for more than two decades. In early days, we encountered a lot of performance issues but now since there is huge growth in resources in terms of hardware and software, the performance improvement factors can be narrowed down to two issues. Firstly, we need to address the issue of finding the information from the relational database in an efficient way. Secondly, the issue of choosing the way in which the indexes and tables are scanned must be addressed. The objective of this project is to use all the statistical data from the relational database and use them efficiently to create indexes and quantify the success of the indexes created in terms of the time units difference, for a work load that executed on the database, before and after deploying the new indexes identified. By this, we mean that the indexes that are created are based on the workload that the database experiences. we build a tool that analyses a workload of queries that are executed against a database management system. This analysis will understand the indexes used, the number of records fetched, the amount of time taken, the number of tables queried, etc. After this analysis, the tool suggests improvements that can be achieved by creating any missing indexes, updating the statistics, point of subtle ways a query could be re-written, etc. We use the three-star algorithm along with traditional indexing rules that are covered in the later part of this report.

**Introduction:**

Indexes are structures that help the data base management system for faster access to tuples that are requested. The database management system first traverses the index structure, if one exists. This is because the index structure may be ordered, and this might help in reducing the query processing time for the data base management improving its efficiency. Since, these index structures contain the physical addresses of the actual data, we can avoid the search time with in the actual table. This said, for the whole of this idea to be on the side of efficiency, the tables must have the right indexes. Using an index that does not suit the query against the table will lead to drastic results that creates additional over heads, pulling down the performance of the system. Since, this process is not straight forward, we need to consider several optimization factors from the perspective of the workload before we create an index and test against the queries in the workload. In this project to get the anticipates results, we analyse the current snapshot of the database and the planned workload. We execute the workload to get the execution time for each of the query in the workload. From this data, we mine for the query that consumes more time and analyse the table related information, statistics and patterns. We use certain rules along with the three-star algorithm to deduce the right choice of index. We bench mark the workload against the database with new intuitive indexes added. The success of the missing indexes that we created will be quantifies in terms of the execution time differences between the bench marks, the one with indexes missed and the one with the missing indexed identified and created.

**Background:**

Amongst all the significant efforts that has been worked around by the administrator of the databases, improving the performance of the databases remain a research and challenge. The challenge can be resolved only when we understand how data records are logically organized in a computational storage device and, especially, to how quickly those records can be accessed and processed. One way to make a SELECT operation efficient is to create indexes on one or more columns that are tested in the query. These index entries allow the query to swiftly determine the tuples that match the condition in the WHERE clause, and retrieve the other column values for those tuples. Though we can create indexes for every column used in a query, unnecessary indexes waste space and pose to be an overhead for data base management system to determine the choice of index to use. Indexes also add to the cost of inserts, updates, and deletes because each index must be updated after these operations. The database management system should always have the optimal set of indexes so that it can be produce useful results faster.

Architecture and modules:

Estimate execution time difference

Credential check

Parse the query

Benchmark workload

Create hypothetical indexes

Benchmark workload

Collect and analyse statistics

Indexing rules and algorithms

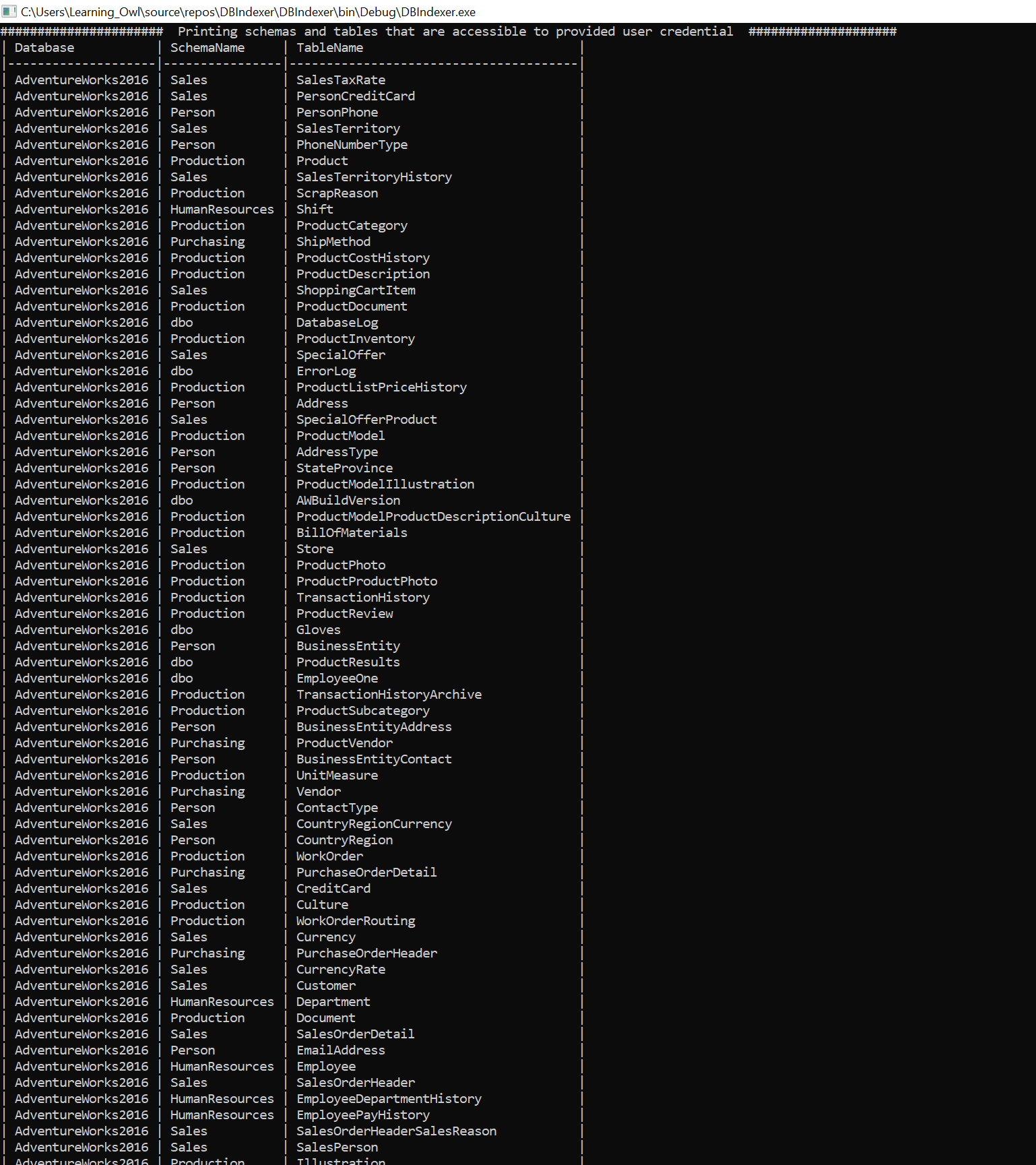
**Fig. 1 Architecture diagram for Index suggestion**

**Workload:**

The input to this project will be the database and the workload – a set of queries that must be triggered against the database chosen.

**Check admin privilege:**

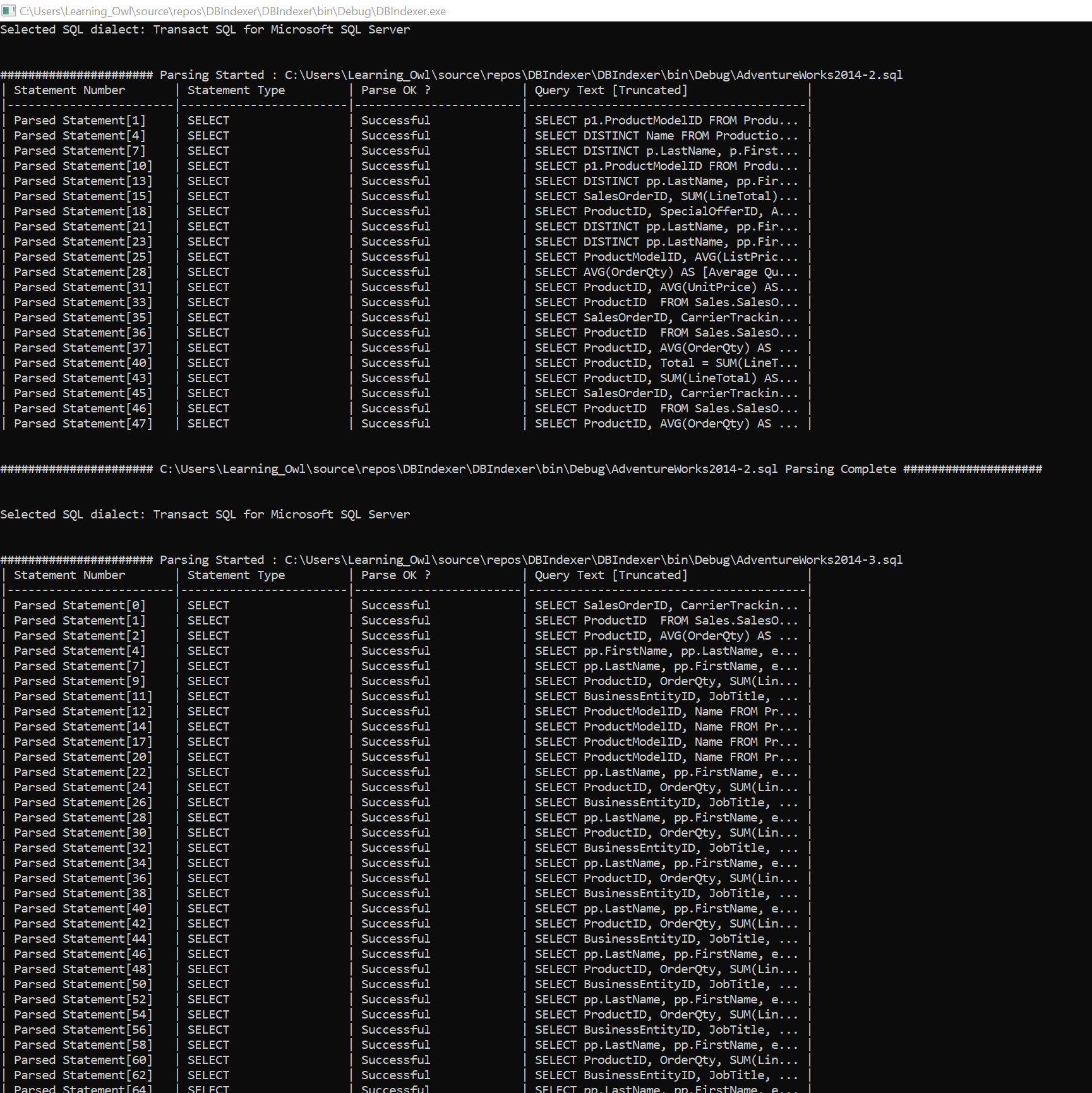
We check if the user has the permission to trigger those queries in the workload and we also validate the credentials of the user with the database.



**Fig. 2 Listing the database objects and privilege checking**

**Parse the queries:**

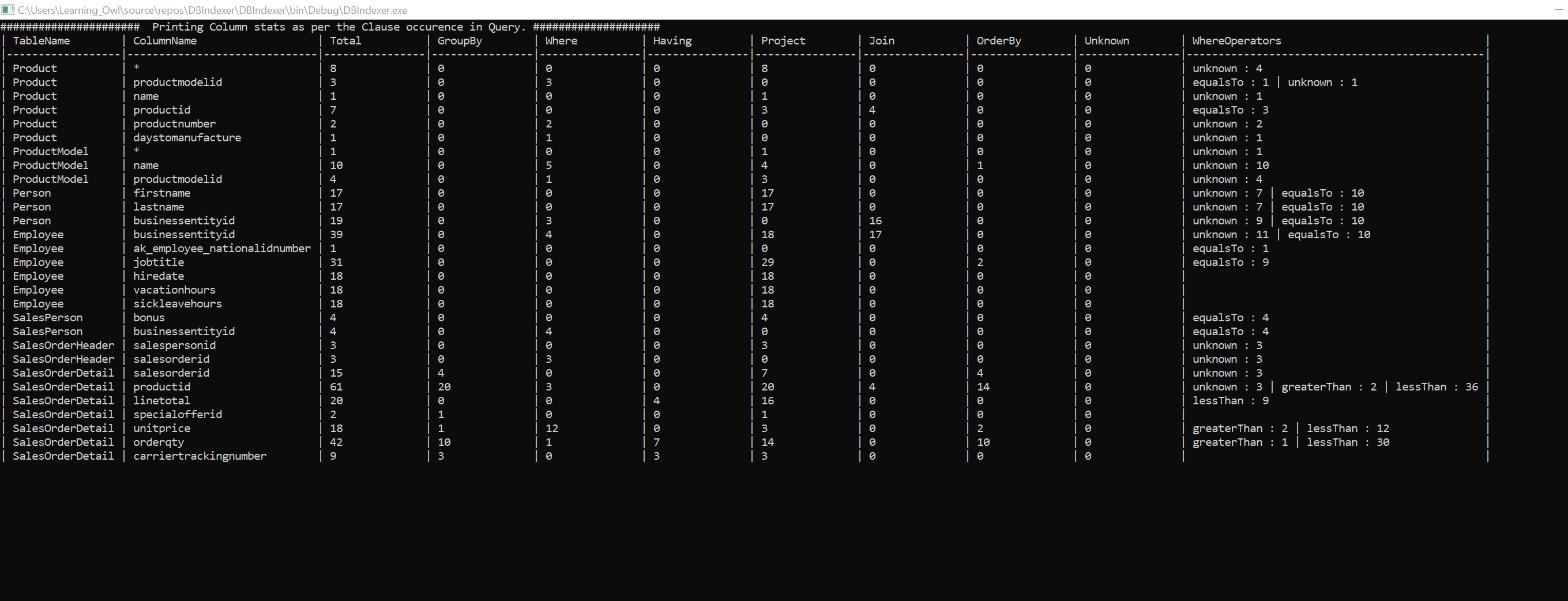
We parse the queries to develop the statistic data for each of the attributes in the queries. This module also takes into consideration, the sub queries, JOINs etc.,



**Fig. 3 Parsing the workload**

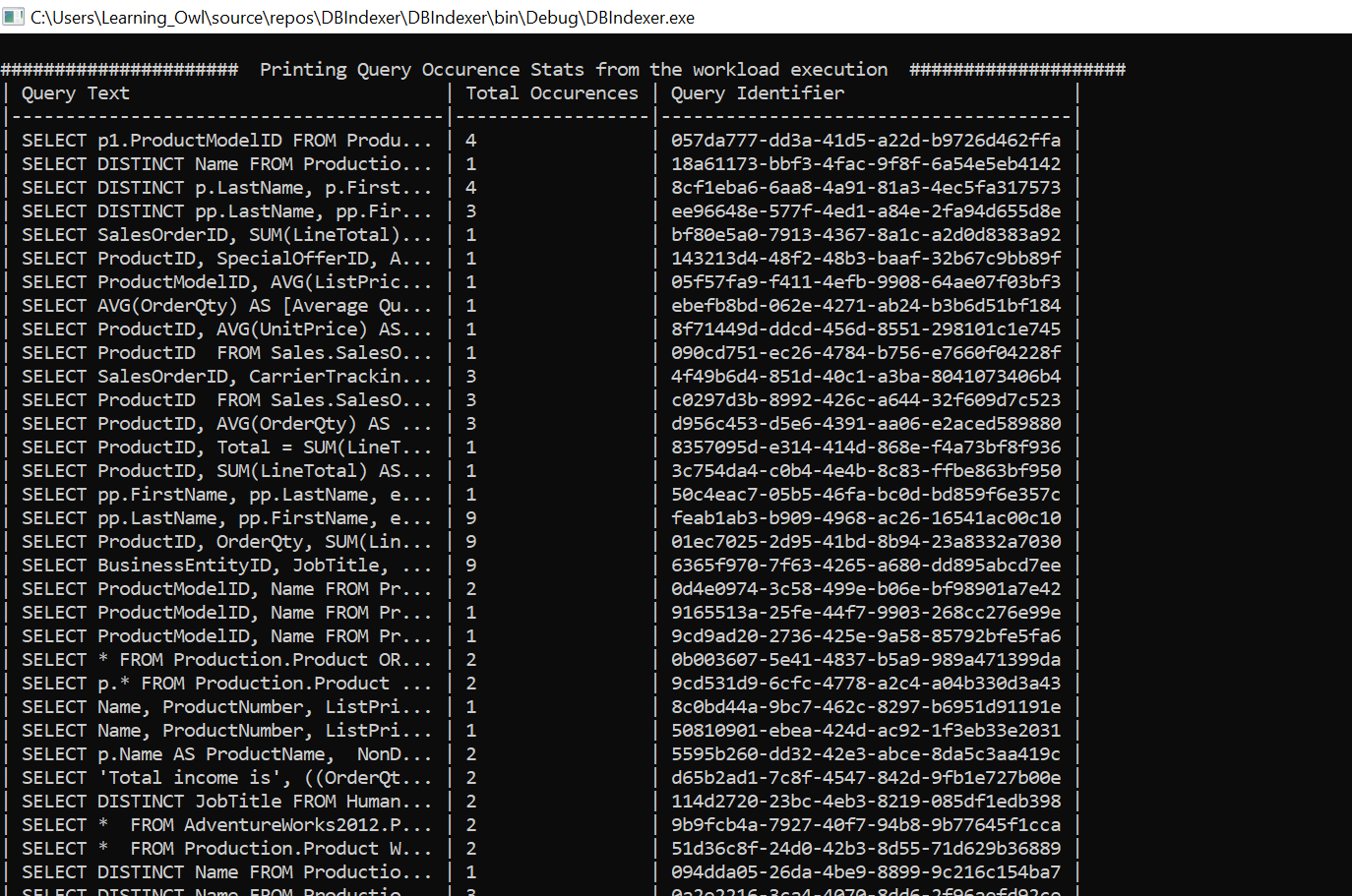
**Execute the workload and benchmark:**

We execute the queries in the workload against the database with the system generated existing indexes and capture the execution time for each query. We store this data in a table.

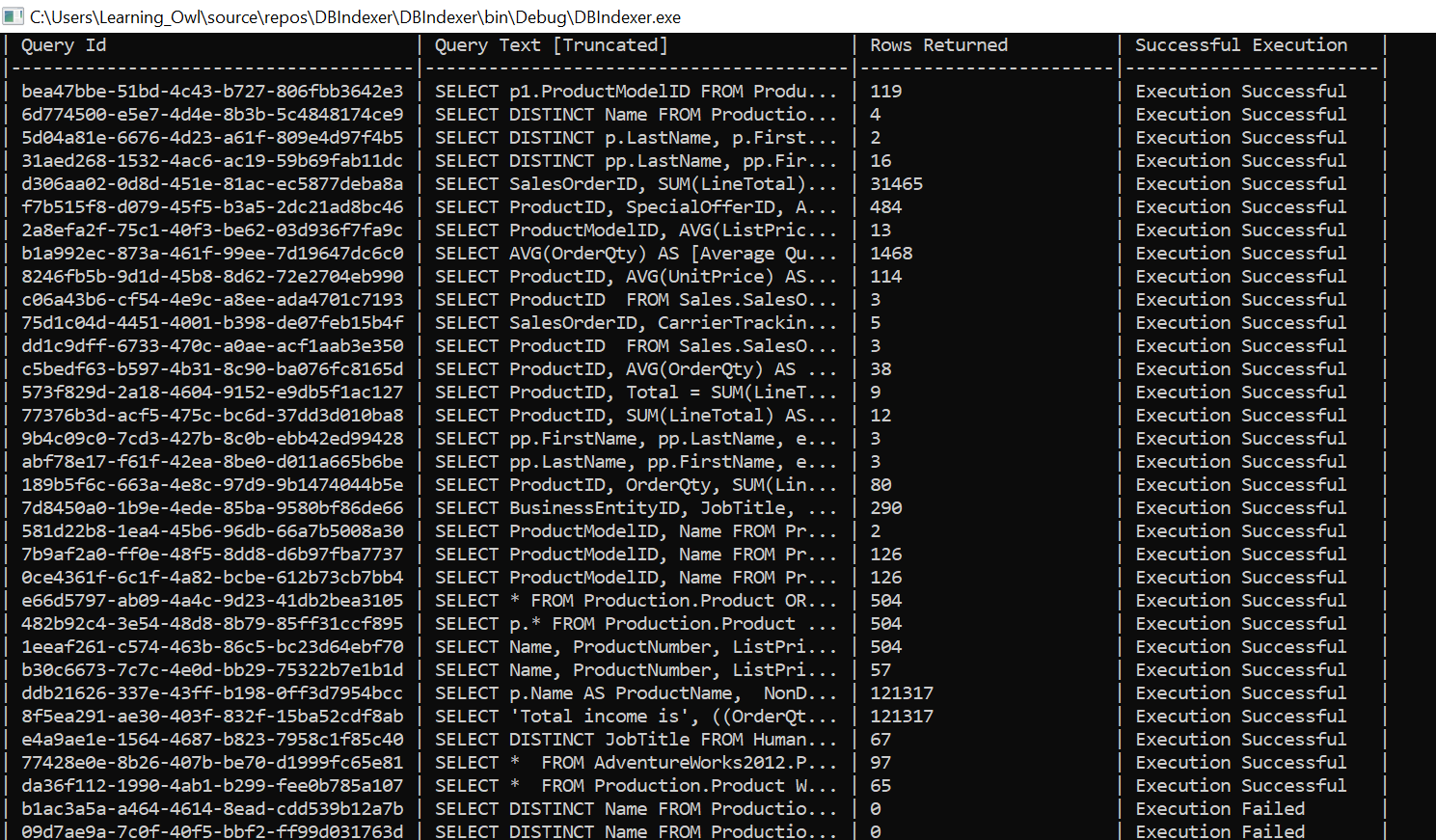


**Fig. 4 Statistics data from system tables for the workload**

**Collect the statistics:**

We traverse through the table that persists the execution time and identify those queries that has consumed larger slice of time in the total execution time of the workload. We now collect the various statistics like the selectivity, reduction factors, predicates, type of data, repetition of the columns, primary key, foreign keys, etc., for the attributes and relations involved.

**Fig. 5 Workload query occurrence count**



**Fig 6. Workload execution snapshot**

**Create Hypothetical indexes:**

Based the static data, we use the algorithm and the indexing rules to identify a heuristic to find out the columns for which index creation would seem useful. We create all these indexes in the relation.

**Bench mark the workload, again.**

To quantify the success of the intuitive indexes created, we benchmark the workload again and record the execution time. We now evaluate the difference in time taken for execution of the same query in the workload with and without intuitive indexes.

**Display the output in a presentable manner:**

This module takes the quantified time difference data and presents it to the user in the browser.

Index rules used:

1. If a query has selective # of columns in a workload, create indexes on all the columns.
2. Look for most heavily used queries and create indexed on those columns (TBD)
3. If any query needs more than 10% of data as output, see if we can drop those indexes without conflicts
4. If the table itself is tiny, we can remove the index.
5. Index multiple columns in order from high cardinality to less. It means, first the columns with more distinct values followed by columns with fewer distinct values.
6. Look if some table is heavily updated. Then, see if the indexes can be reduced/removed.
7. Create non-clustered indexes on all columns that are frequently used in predicates and join conditions in queries.
8. See if any query is acting only on few columns and if so, make index only calculation possible.
9. When there are multiple updates/inserts, drop all the index. Add them again.

We are also working on an algorithm that can help us with analysis of multiple queries chosen to have optimized indexes to improve the overall performance.

**References:**

1. A Tool for Automatic Index Selection in Database Management Systems <http://ieeexplore.ieee.org/document/684s6069/>
2. Relational Database Index Design and the Optimizers: DB2, Oracle, SQL Server, et al.

<http://onlinelibrary.wiley.com/book/10.1002/0471721379>

**Git Repository:**

<https://github.com/LearningOwl/DBIndexer>