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:

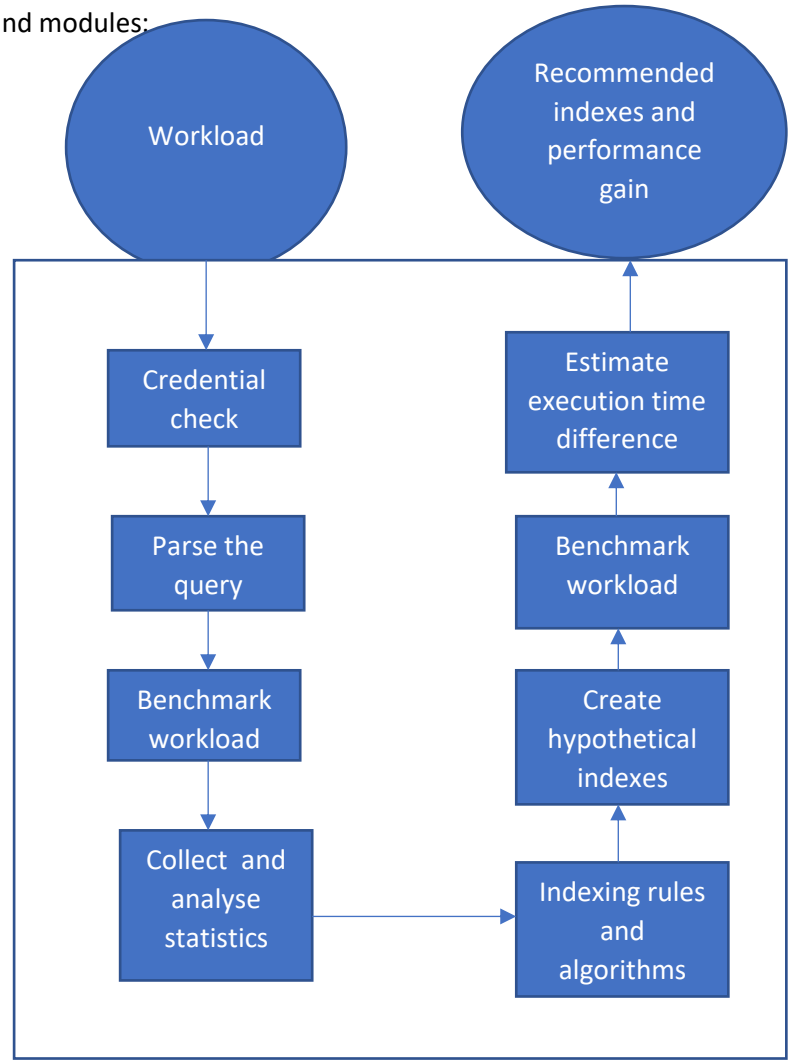


Fig. 1 Architecture diagram for Index suggestion

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.

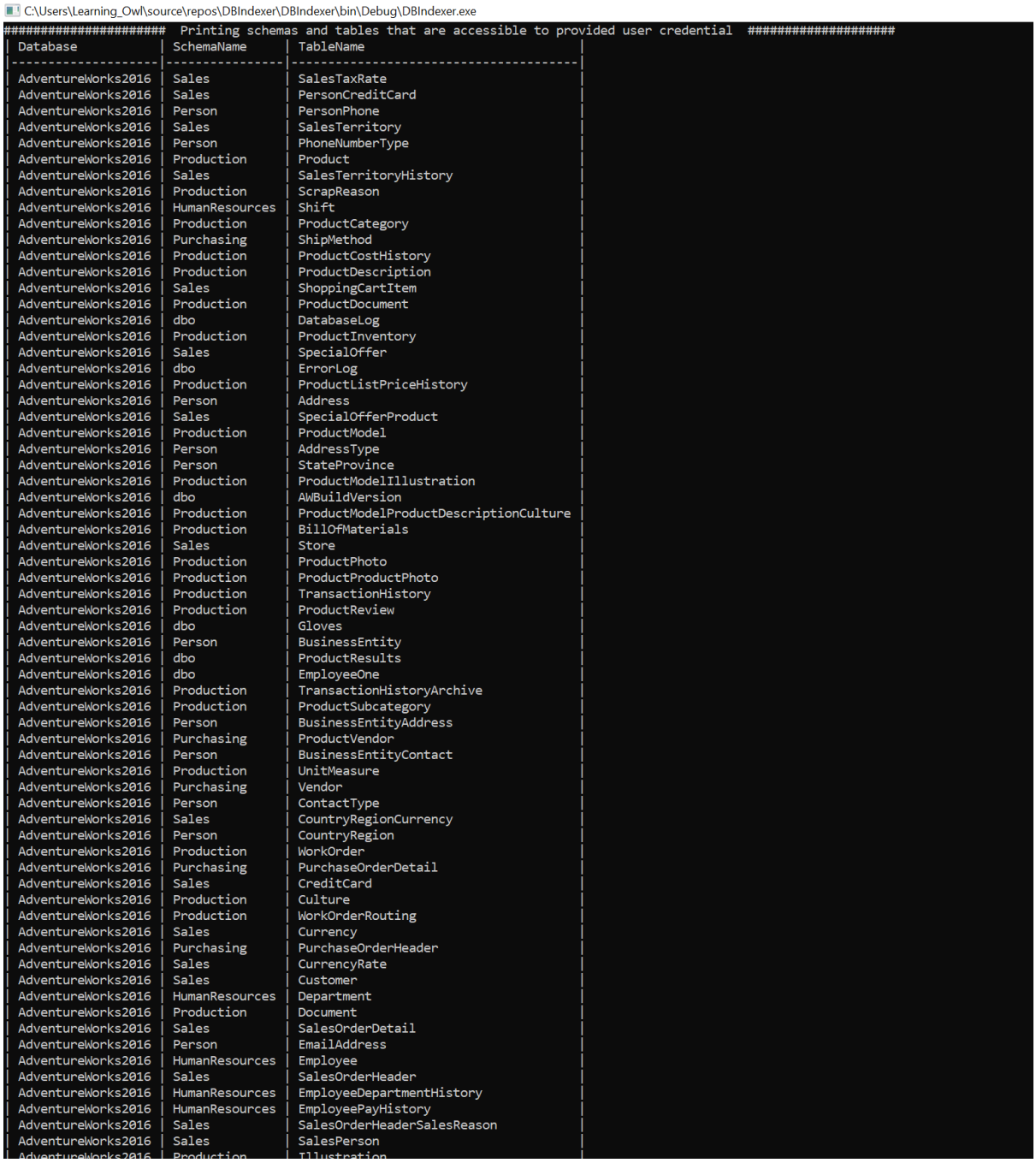


Fig. 2 Listing the database objects and privilege checking

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.,

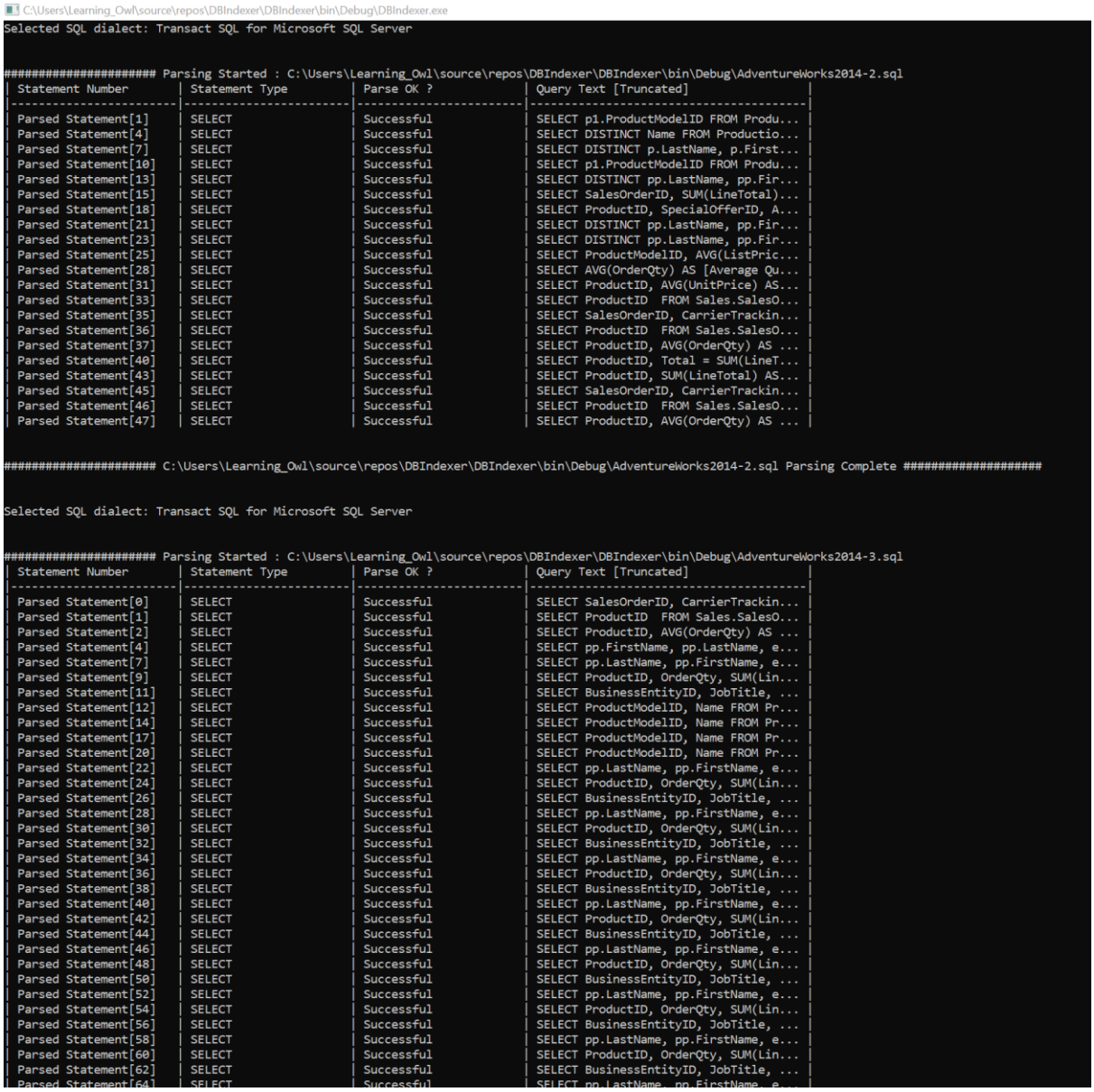
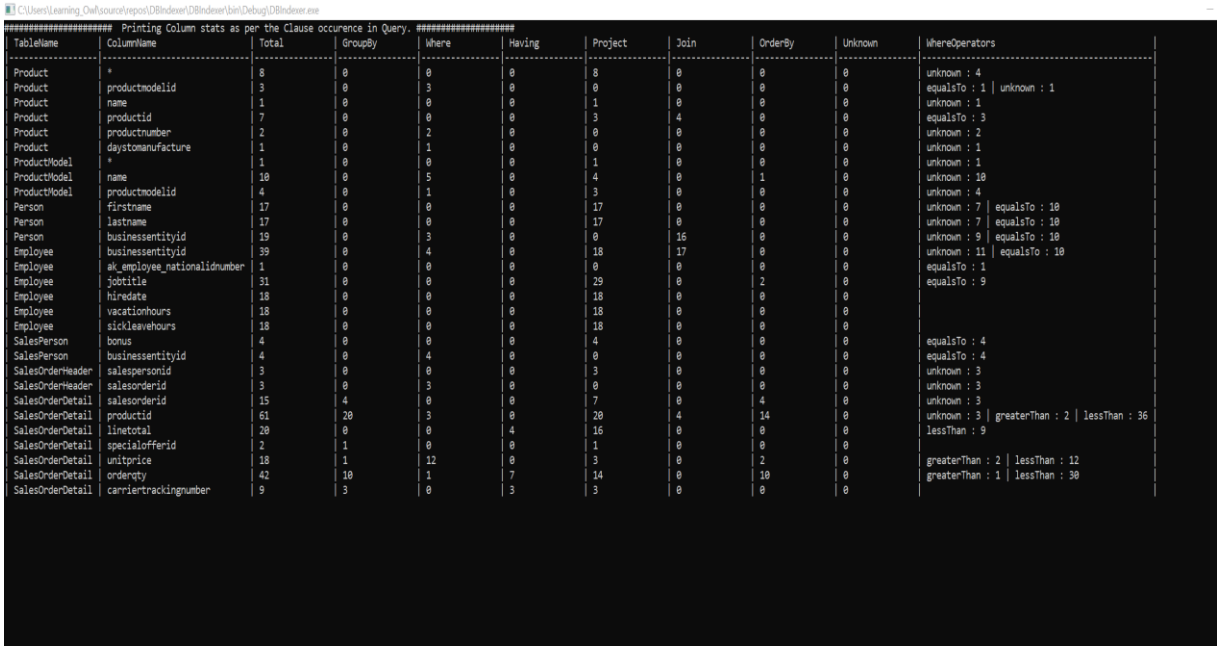


Fig. 3 Parsing the workload

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.



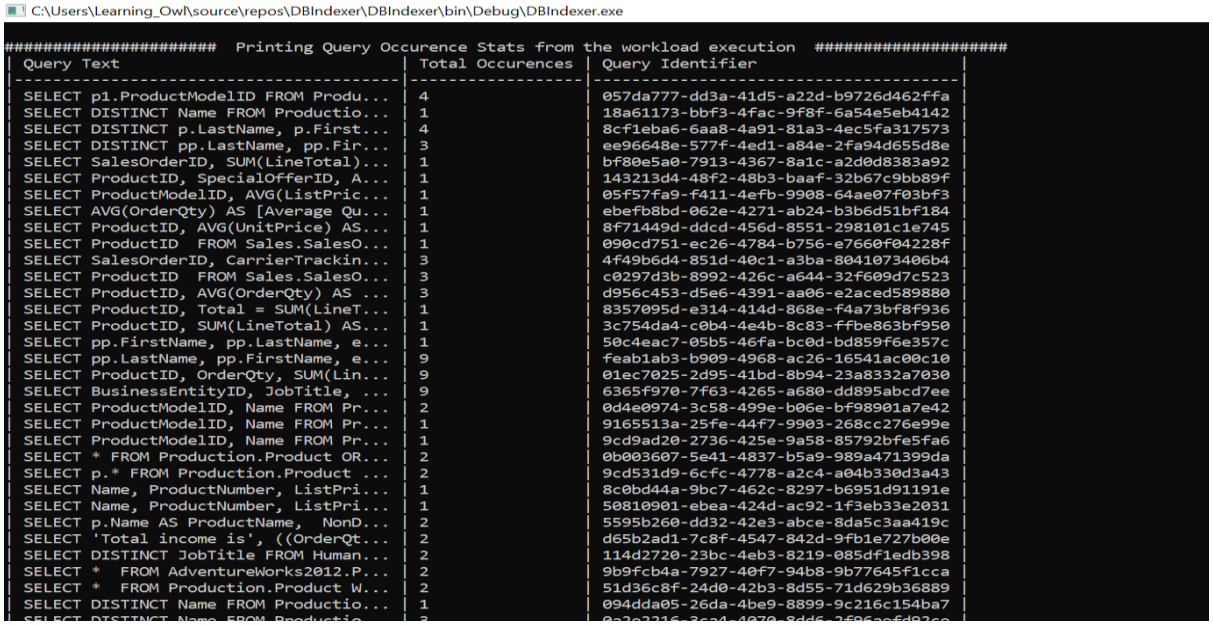
```
C:\Users\Learning_Owl\source\repos\DBIndexer\DBIndexer\bin\Debug\DBIndexer.exe
#################### Printing Column stats as per the Clause occurence in Query. ##################
```

TableName	ColumnName	Total	GroupBy	Where	Having	Project	Join	OrderBy	Unknown	WhereOperators
Product	*	8	0	0	0	8	0	0	0	unknown : 4
Product	productmodelid	3	0	3	0	0	0	0	0	equalsTo : 1 | unknown : 1
Product	name	1	0	0	0	1	0	0	0	unknown : 1
Product	productid	7	0	0	0	3	4	0	0	equalsTo : 3
Product	productnumber	2	0	2	0	0	0	0	0	unknown : 2
Product	daystomanufacture	1	0	1	0	0	0	0	0	unknown : 1
ProductModel	*	1	0	0	0	1	0	0	0	unknown : 1
ProductModel	name	10	0	5	0	4	0	1	0	unknown : 10
ProductModel	productmodelid	4	0	1	0	3	0	0	0	unknown : 4
Person	firstname	17	0	0	0	17	0	0	0	unknown : 7 | equalsTo : 10
Person	lastname	17	0	0	0	17	0	0	0	unknown : 7 | equalsTo : 10
Person	businessentityid	19	0	3	0	0	16	0	0	unknown : 9 | equalsTo : 10
Employee	businessentityid	39	0	4	0	18	17	0	0	unknown : 11 | equalsTo : 10
Employee	ak_employee_nationalidnumber	1	0	0	0	0	0	0	0	equalsTo : 1
Employee	jobtitle	31	0	0	0	29	0	2	0	equalsTo : 9
Employee	hiredate	18	0	0	0	18	0	0	0	
Employee	vacationhours	18	0	0	0	18	0	0	0	
Employee	sickleavehours	18	0	0	0	18	0	0	0	
SalesPerson	bonus	4	0	0	0	4	0	0	0	equalsTo : 4
SalesPerson	businessentityid	4	0	4	0	0	0	0	0	equalsTo : 4
SalesOrderHeader	salespersonid	3	0	0	0	3	0	0	0	unknown : 3
SalesOrderHeader	salesorderid	3	0	3	0	0	0	0	0	unknown : 3
SalesOrderDetail	salesorderid	15	4	0	0	7	0	4	0	unknown : 3
SalesOrderDetail	productid	61	20	3	0	20	4	14	0	unknown : 3 | greaterThan : 2 | lessThan : 36
SalesOrderDetail	linetotal	20	0	0	4	16	0	0	0	lessThan : 9
SalesOrderDetail	specialofferid	2	1	0	0	1	0	0	0	
SalesOrderDetail	unitprice	18	1	12	0	3	0	2	0	greaterThan : 2 | lessThan : 12
SalesOrderDetail	orderqty	42	10	1	7	14	0	10	0	greaterThan : 1 | lessThan : 30
SalesOrderDetail	carriertrackingnumber	9	3	0	3	3	0	0	0	

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.



```
C:\Users\Learning_Owl\source\repos\DBIndexer\DBIndexer\bin\Debug\DBIndexer.exe
#######################  Printing Query Occurence Stats from the workload execution  ####################
```

Query Text	Total Occurences	Query Identifier
SELECT p1.ProductModelID FROM Produ...	4	057da777-dd3a-41d5-a22d-b9726d462ffa
SELECT DISTINCT Name FROM Productio...	1	18a61173-bbf3-4fac-9f8f-6a54e5eb4142
SELECT DISTINCT p.LastName, p.First...	4	8cf1eba6-6aa8-4a91-81a3-4ec5fa317573
SELECT DISTINCT pp.LastName, pp.Fir...	3	ee96648e-577f-4ed1-a84e-2fa94d655d8e
SELECT SalesOrderID, SUM(LineTotal)...	1	bf80e5a0-7913-4367-8a1c-a2d0d8383a92
SELECT ProductID, SpecialOfferID, A...	1	143213d4-48f2-48b3-baaf-32b67c9bb89f
SELECT ProductModelID, AVG(ListPric...	1	05f57fa9-f411-4efb-9908-64ae07f03bf3
SELECT AVG(OrderQty) AS [Average Qu...	1	ebefb8bd-062e-4271-ab24-b3b6d51bf184
SELECT ProductID, AVG(UnitPrice) AS...	1	8f71449d-ddcd-456d-8551-298101c1e745
SELECT ProductID FROM Sales.SalesO...	1	090cd751-ec26-4784-b756-e7660f04228f
SELECT SalesOrderID, CarrierTrackin...	3	4f49b6d4-851d-40c1-a3ba-8041073406b4
SELECT ProductID FROM Sales.SalesO...	3	c0297d3b-8992-426c-a644-32f609d7c523
SELECT ProductID, AVG(OrderQty) AS ...	3	d956c453-d5e6-4391-aa06-e2aced589880
SELECT ProductID, Total = SUM(LineT...	1	8357095d-e314-414d-868e-f4a73bf8f936
SELECT ProductID, SUM(LineTotal) AS...	1	3c754da4-c0b4-4e4b-8c83-ffbe863bf950
SELECT pp.FirstName, pp.LastName, e...	1	50c4eac7-05b5-46fa-bc0d-bd859f6e357c
SELECT pp.LastName, pp.FirstName, e...	9	feab1ab3-b909-4968-ac26-16541ac00c10
SELECT ProductID, OrderQty, SUM(Lin...	9	01ec7025-2d95-41bd-8b94-23a8332a7030
SELECT BusinessEntityID, JobTitle, ...	9	6365f970-7f63-4265-a680-dd895abcd7ee
SELECT ProductModelID, Name FROM Pr...	2	0d4e0974-3c58-499e-b06e-bf98901a7e42
SELECT ProductModelID, Name FROM Pr...	1	9165513a-25fe-44f7-9903-268cc276e99e
SELECT ProductModelID, Name FROM Pr...	1	9cd9ad20-2736-425e-9a58-85792bfe5fa6
SELECT * FROM Production.Product OR...	2	0b003607-5e41-4837-b5a9-989a471399da
SELECT p.* FROM Production.Product ...	2	9cd531d9-6cfc-4778-a2c4-a04b330d3a43
SELECT Name, ProductNumber, ListPri...	1	8c0bd44a-9bc7-462c-8297-b6951d91191e
SELECT Name, ProductNumber, ListPri...	1	50810901-ebea-424d-ac92-1f3eb33e2031
SELECT p.Name AS ProductName, NonD...	2	5595b260-dd32-42e3-abce-8da5c3aa419c
SELECT 'Total income is', ((OrderQt...	2	d65b2ad1-7c8f-4547-842d-9fb1e727b00e
SELECT DISTINCT JobTitle FROM Human...	2	114d2720-23bc-4eb3-8219-085df1edb398
SELECT * FROM AdventureWorks2012.P...	2	9b9fcb4a-7927-40f7-94b8-9b77645f1cca
SELECT * FROM Production.Product W...	2	51d36c8f-24d0-42b3-8d55-71d629b36889
SELECT DISTINCT Name FROM Productio...	1	094dda05-26da-4be9-8899-9c216c154ba7

Fig. 5 Workload query occurrence count

C:\Users\Learning_Owl\source\repos\DBIndexer\DBIndexer\bin\Debug\DBIndexer.exe

Query Id	Query Text [Truncated]	Rows Returned	Successful Execution
bea47bbe-51bd-4c43-b727-806fbb3642e3	SELECT p1.ProductModelID FROM Produ...	119	Execution Successful
6d774500-e5e7-4d4e-8b3b-5c4848174ce9	SELECT DISTINCT Name FROM Productio...	4	Execution Successful
5d04a81e-6676-4d23-a61f-809e4d97f4b5	SELECT DISTINCT p.LastName, p.First...	2	Execution Successful
31aed268-1532-4ac6-ac19-59b69fab11dc	SELECT DISTINCT pp.LastName, pp.Fir...	16	Execution Successful
d306aa02-0d8d-451e-81ac-ec5877deba8a	SELECT SalesOrderID, SUM(LineTotal)...	31465	Execution Successful
f7b515f8-d079-45f5-b3a5-2dc21ad8bc46	SELECT ProductID, SpecialOfferID, A...	484	Execution Successful
2a8efa2f-75c1-40f3-be62-03d936f7fa9c	SELECT ProductModelID, AVG(ListPric...	13	Execution Successful
b1a992ec-873a-461f-99ee-7d19647dc6c0	SELECT AVG(OrderQty) AS [Average Qu...	1468	Execution Successful
8246fb5b-9d1d-45b8-8d62-72e2704eb990	SELECT ProductID, AVG(UnitPrice) AS...	114	Execution Successful
c06a43b6-cf54-4e9c-a8ee-ada4701c7193	SELECT ProductID FROM Sales.SalesO...	3	Execution Successful
75d1c04d-4451-4001-b398-de07feb15b4f	SELECT SalesOrderID, CarrierTrackin...	5	Execution Successful
dd1c9dff-6733-470c-a0ae-acf1aab3e350	SELECT ProductID FROM Sales.SalesO...	3	Execution Successful
c5bedf63-b597-4b31-8c90-ba076fc8165d	SELECT ProductID, AVG(OrderQty) AS ...	38	Execution Successful
573f829d-2a18-4604-9152-e9db5f1ac127	SELECT ProductID, Total = SUM(LineT...	9	Execution Successful
77376b3d-acf5-475c-bc6d-37dd3d010ba8	SELECT ProductID, SUM(LineTotal) AS...	12	Execution Successful
9b4c09c0-7cd3-427b-8c0b-ebb42ed99428	SELECT pp.FirstName, pp.LastName, e...	3	Execution Successful
abf78e17-f61f-42ea-8be0-d011a665b6be	SELECT pp.LastName, pp.FirstName, e...	3	Execution Successful
189b5f6c-663a-4e8c-97d9-9b1474044b5e	SELECT ProductID, OrderQty, SUM(Lin...	80	Execution Successful
7d8450a0-1b9e-4ede-85ba-9580bf86de66	SELECT BusinessEntityID, JobTitle, ...	290	Execution Successful
581d22b8-1ea4-45b6-96db-66a7b5008a30	SELECT ProductModelID, Name FROM Pr...	2	Execution Successful
7b9af2a0-ff0e-48f5-8dd8-d6b97fba7737	SELECT ProductModelID, Name FROM Pr...	126	Execution Successful
0ce4361f-6c1f-4a82-bcbe-612b73cb7bb4	SELECT ProductModelID, Name FROM Pr...	126	Execution Successful
e66d5797-ab09-4a4c-9d23-41db2bea3105	SELECT * FROM Production.Product OR...	504	Execution Successful
482b92c4-3e54-48d8-8b79-85ff31ccf895	SELECT p.* FROM Production.Product ...	504	Execution Successful
1eeaf261-c574-463b-86c5-bc23d64ebf70	SELECT Name, ProductNumber, ListPri...	504	Execution Successful
b30c6673-7c7c-4e0d-bb29-75322b7e1b1d	SELECT Name, ProductNumber, ListPri...	57	Execution Successful
ddb21626-337e-43ff-b198-0ff3d7954bcc	SELECT p.Name AS ProductName, NonD...	121317	Execution Successful
8f5ea291-ae30-403f-832f-15ba52cdf8ab	SELECT 'Total income is', ((OrderQt...	121317	Execution Successful
e4a9ae1e-1564-4687-b823-7958c1f85c40	SELECT DISTINCT JobTitle FROM Human...	67	Execution Successful
77428e0e-8b26-407b-be70-d1999fc65e81	SELECT * FROM AdventureWorks2012.P...	97	Execution Successful
da36f112-1990-4ab1-b299-fee0b785a107	SELECT * FROM Production.Product W...	65	Execution Successful
b1ac3a5a-a464-4614-8ead-cdd539b12a7b	SELECT DISTINCT Name FROM Productio...	0	Execution Failed
09d7ae9a-7c0f-40f5-bbf2-ff99d031763d	SELECT DISTINCT Name FROM Productio...	0	Execution Failed

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