

# Self-Managing Technology in Database Management Systems

Kyuseok Shim

1

## Motivation

- Hardware & Software becomes cheaper and more powerful
  - CPU is getting faster
  - Sizes of Memory and Disk is getting larger
  - DELL XPS 600 costs only \$2749
    - 2 GB of memory
    - 1TB RAID
    - Pentium® D Processor 820 (2.8GHz, 800FSB)
    - Include 17 inch Flat Monitor
- Complexity of DBMS is increasing
- DBA is getting more expensive, but not more reliable

2



## Motivation

- Total cost of database management is significantly dominated by human costs.
- 55% of DBA spend time in management, monitoring and tuning
- Increased business competitiveness forces to cut operating expenses
- Self-configuration and self-tuning is a viable solution!

3




## Ongoing System Management

- Performance Diagnosis & Troubleshooting
- SQL and Application tuning
  - Response time, throughput, schema, index, views
- System Resource Tuning
  - CPU utilization, buffer, memory
- Space Management
  - Disk configuration, data fragmentation
- Backup & Recovery

Source: International Oracle User Group (IOUG) 2001 DBA Survey


4



---

# Microsoft SQL Server

5



---

## Self Tuning for SQL Server

- Monitoring tools
  - SQL Profiler
  - SQLCM
  - Workload analyzer
  - SQL Query Execution Progress Estimator
- Self Tuning Memory Management
- Statistics Management
- Self Tuning Physical Database Design

6



## Monitoring Tools

---

- Check connections, space, long running jobs
  - PERFMON command
- Best Practices Analyzer
  - Detect common oversights in SQL Server installation

7



## SQL Profiler

---

- GUI tool for SQL Trace
  - SQL Trace
    - Located at server
  - Event log

8



## SQLCM

---

- Located at SQL Server
- Online grouping/aggregation processed at server help monitoring
- ECA rule are used to control monitoring

9



## SQLCM

---

- Logical query signature
  - Tree structure
  - Exact matching is allowed between signatures
- Light-weight aggregation table (LAT)
  - Grouping attributes and aggregations

10



## Workload Analyzer

---

- Obtained using SQL Profiler from logs of workloads in server
- Useful for
  - DBA
  - Self tuning tools
- Workload can be very large
- Scalability is important

11



## Workload Analyzer

---

- Summarization
  - Random sampling
  - Clustering

12



## SQL Query Execution Progress Estimator

---

- Provide accurate progress estimator during query execution
- Use feedback from query execution

13



## Automatic Statistics Management

---

- Invoked by Query Optimizer
- Refreshed when a certain number of tuples are updated
- Use sampling
  - Uniform random sampling is expensive
  - Block-level sampling is cheap, but may degrade the quality of estimation
    - Pick a few blocks at random and use all tuples in them

14



## Self Tuning Physical Database Design - Index Tuning Wizard

- Provide recommendations for indexes, materialized views and partitioning at SQL Server 2005
- Constraints: Tuning time and space limit on the space for indexes and views
- Estimate the impact of physical design on workload without making actual changes to database
  - Consult to the query optimizer
  - Each physical design simulated by faking statistics
- Search space is reduced by pruning used in frequent itemset mining
- Incremental cost estimation for each different physical design

15



## Cost-Driven Index Selection

- [Chaudhuri, Narasayya: VLDB 1997]
- Efficiency of selection and quality of its solution to the optimal
- Important measures of efficiency
  - Number of indexes considered
  - Number of configurations enumerated
  - Number of optimizer invocations

16





## Cost-Driven Index Selection

- Optimizations
  - Invoke optimizers only for a selected subset of the configurations
  - Since an index considered may not be present in the current database, need to simulate the presence of index for the optimizer

17



## Cost-Driven Index Selection

- Indexable column for a query
  - a column R.a such that there is a condition in WHERE clause
  - A column in GROUP BY or ORDER BY clause
- Admissible index for a query
  - An index that is on one or more indexable columns of the query
- Admissible index for a workload
  - Admissible index for at least one query in the workload

18



## Indexable Columns : An Example

```
select *  
from onektup, tenktup1  
where onektup.unique1 = tenktup1.unique1 and  
       tenktup1.unique2 between 0 and 100
```

{onektup.unique1, tenktup1.unique1, tenktup1.unique2} is the set  
of indexable columns

19

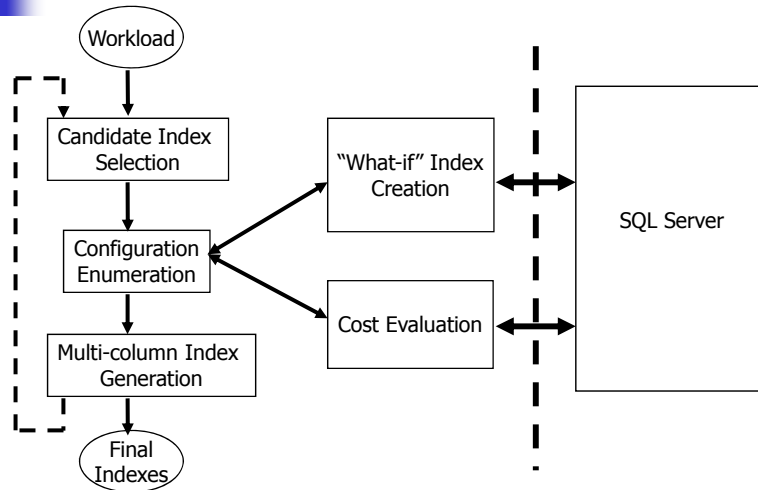


## Cost-Driven Index Selection

- A configuration C is *atomic* for a workload if for some query in the workload, there is a possible plan that uses all indexes in C
- Instead of evaluating all M configurations, it is sufficient to ask optimizer to evaluate only M' that contains all atomic configurations among M

20

## Architecture of Index Selection Tool



21

## Cost Evaluation of Atomic Configurations

- For select query
  - Inclusion of index can only reduce the cost
  - Suffice to take the minimum cost over the largest atomic configurations

## Cost Evaluation of Atomic Configurations

- For insert/delete query
  - Cost consists of
    - (a) selection
    - (b) updating the table and the index that may be used for selection
    - (c) updating index that do not affect the selection cost
  - Cost of updating index that do not affect the selection cost is independent of each other and independent of (a) and (b)
  - Suffice to pick a plan with the minimum cost for (a) and (b)

© TIM & Kyuseok Shim

23

## Cost Evaluation of Atomic Configurations

- Cost consists of
  - (a) selection
  - (b) updating the table and the index that may be used for selection
  - (c) updating index that do not affect the selection cost
- For select/update query
  - Let T be the minimum cost of all atomic configurations of Q that are subsets of C in (a) and (b)
  - The cost of updating an index I in (c) is  $Cost(Q, \{I\}) - Cost(Q, \{\})$
  - The total cost can be computed without invoking the optimizer for C as:

$$T + \sum_k (Cost(Q, \{I_k\}) - Cost(Q, \{\}))$$

© TIM & Kyuseok Shim

24



## Atomic Configurations

- Number of atomic configuration is exponential in number of tables
- Query Processor based restrictions
  - For a query execution, allow up to a certain number (J) of indexes per table
  - For multi-table query execution, allow the indexes from at most a certain number of (K) tables
  - J=2 and K=2 provide good quality
  - We call single join atomic configuration

25



## Indexable Columns : An Example

```
select *  
from T1, T2, T3  
where T1.a < 20 and T1.a = T2.B and  
      T3.c between 30 and 50
```

- One 3-table atomic configuration is (T1.a, T2.b, T3.c)
- Due to the single-join atomic configuration based pruning, it is not evaluated.
  - Rather be computed by taking minimum cost of the atomic configurations: (T1.a, T2.b), (T1.a, T3.c), (T2.b, T3.c)

26



## Relevant Index Set Optimization

- When asked to evaluate  $\text{COST}(Q, C)$ 
  - Let the set of indexable columns for select/update query  $Q$  be  $P$
  - Only indexes in  $C$  that are on at least one column in  $P$  affect the cost of  $Q$
  - If  $C'$  is the configurations consisting of only such indexes,  $\text{COST}(Q, C) = \text{COST}(Q, C')$
  - If  $\text{COST}(Q, C')$  has already evaluated, reuse it

27



## Candidate Index Set

- Observation: An index that is not part of the best design for even a single query is unlikely to be part of the best design for the entire workload
- Strategy
  - Determine the best configuration for each query independently
  - Consider all indexes belonging to at least one of these best configurations as the candidate index set

28



## Configuration Enumeration

- Greedy(m,k)
  - Naive enumeration algorithm picking up to k indexes
  - Greedily choose rest of indexes
- First select single column indexes
- Then, enumerate multi-column indexes in increasing width
- Observation: If two-column index is desirable, a single index for its leading column must also be desirable
  - 2-column (a,b) – a is the leading column

29



## AutoAdmin – “What-if” Index Analysis Utility

- [Chaudhuri, Narasayya: SIGMOD 2005]
- Workload
  - SQL Server Profiler can provide from logging events
  - Can be generated dynamically from Query Analyzer
- Hypothetical Configuration Analysis (HCA) Engine
  - Estimate the cost of a query in the workload and the index usage with respect to a hypothetical configuration
  - Hypothetical configuration
    - Indexes and database scaling values
    - Represents a database where each table  $T_j$  has  $m_j$  times the number of rows where  $m_j$  captures the size of the database for each table  $T_j$

30

## Interface for Hypothetical Configuration Simulation

- DEFINE WORKLOAD <workload\_name> [FROM <file> | AS (Q<sub>1</sub>,f<sub>1</sub>), Q<sub>2</sub>,f<sub>2</sub>),..., (Q<sub>n</sub>,f<sub>n</sub>)]
- DEFINE CONFIGURATION <configuration\_name> AS (Table<sub>1</sub>, column\_list<sub>1</sub>),..., (Table<sub>m</sub>, column\_list<sub>m</sub>)
- SET DATABASE SIZE OF <configuration\_name> AS (Table<sub>1</sub>, row\_count<sub>1</sub>),..., (Table<sub>m</sub>, row\_count<sub>m</sub>)
- ESTIMATE CONFIGURATION OF <workload\_name> for <configuration\_name>
- REMOVE [WORKLOAD <workload\_name> | Configuration <configuration\_name> | COST-USAGE <workload\_name>, <configuration\_name>

31

## DEFINE WORKLOAD command

- DEFINE WORKLOAD <workload\_name> [FROM <file> | AS (Q<sub>1</sub>,f<sub>1</sub>), Q<sub>2</sub>,f<sub>2</sub>),..., (Q<sub>n</sub>,f<sub>n</sub>)]
- Associates a name with a set of queries
- These queries can be specified from a file or directly through the command
- The frequency f<sub>i</sub> is interpreted the workload consisting of f<sub>i</sub> copies of query Q<sub>i</sub>

Workload name	Query ID	Frequency	Query Properties
WrkId_A	1	1	<SQL Text>, {T <sub>1</sub> , T <sub>2</sub> }, etc.

© TIM & Kyuseok Shim

32





## DEFINE CONFIGURATION command

- DEFINE CONFIGURATION <configuration\_name> AS (Table<sub>1</sub>, column\_list<sub>1</sub>),..., (Table<sub>m</sub>, column\_list<sub>m</sub>)

### Configuration Information

Configuration name	Indexes in Configuration	Scaling values
Current_Conf	Ind_A, Ind_B, Ind_D	(T <sub>1</sub> ,1), (T <sub>2</sub> ,5)

### Index Information

Index name	Table name	Number of rows	Number of pages	Columns	Index Statistics
Ind_A	R	10,000	1685	R.a	<histogram>

© TIM & Kyuseok Shim

33



## ESTIMATE CONFIGURATION command

- ESTIMATE CONFIGURATION OF <workload\_name> for <configuration\_name>
- The unit of cost below is relative to the total cost of the workload in current configuration
- The Used Indexes are expected to be used by the server if the hypothetical configuration existed

Configuration name	Query ID	Cost	Used Indexes
New_Conf	1	0.02	Ind_A, Ind_D
New_Conf	2	0.11	Ind_B

© TIM & Kyuseok Shim

34



## REMOVE COMMAND

- REMOVE [WORKLOAD <workload\_name> | Configuration <configuration\_name> | COST-USAGE <workload\_name>, <configuration\_name>
- Removes analysis data generated by previous commands
- All information about the workload is removed including cost and usage information
- When REMOVE COST-USAGE is invoked
  - only the cost-usage information for the specified workload and configuration is removed
  - Workload and configuration is retained

35



## Implementation Details

- Simulating a hypothetical configuration by physically altering the current configuration is not viable – due to serious overhead of dropping and creating indexes
- Updating system tables with database scaling value can lead to error in optimizer's estimates of potential queries
- An optimizer's decision on whether or not to use an index is solely based on the statistical information on the columns in the index
- These statistical measures can be obtained via sampling without significantly compromising accuracy

36



## ESTIMATE CONFIGURATION

- The following steps are repeated for each query in the workload
  1. Create all needed hypothetical indexes in the configuration
  2. Request the optimizer to
    1. Restrict its choice of indexes to those in the give configuration
    2. Consider the table and index sizes to be as adjusted by scaling values
  3. Request the optimizer to produce the optimal plan for the query and gather the results
    1. The cost of the query
    2. Indexes used to answer the query

37



## Creation of Hypothetical Indexes

- Extend the CREATE INDEX statement in SQL with the qualifer WITH STATISTICS\_ONLY [= <fraction>]
- It is optionally possible to specify the fraction of the table to be scanned when generating sample data on columns for the index.
- If <fraction> is not given, system determines the appropriate fraction of rows to be scanned

38



## Creation of Hypothetical Indexes

- Sampling Strategy
  - Adaptive page-level sampling algorithm is used
  - The server maintains sorted list of values in the Sample-Table and the set of statistical measures (density of the data set and Equi-depth histograms) based on the Sample-Table
  - The data in New-Sample is used for cross validation purpose
    - Checked if the values in New-Sample are divided approximately in equal numbers in each bin of the histogram
    - If the above test is true, the density measure also reaches convergence
    - If fails, the new sample is added to Sample-Table. This addition is done via merging to build a new Sample-Table
    - Repeat the above step until converges

39



## Defining Hypothetical Configuration

- A hypothetical configuration cannot be supported by updating system catalog
- Instead the information for the hypothetical configuration must be conveyed to the optimizer in a connection-specific manner
- Augment the server with a connection-specific HC mode call
- HC mode call takes as arguments:
  - Set of indexes corresponding to the hypothetical configuration to be used
  - The base index for each table in the configuration
  - Sizes of tables and indexes in the database
- HCA engine projects the size of each index in the configuration based on the database scaling value

40



## Summary Analysis Interface

```
ANALYZE [WORKLOAD | CONFIGURATION | COST-USAGE]
WITH <parameter-list>
[TOP <number> | SUMMARIZE USING <aggregation-function>]
BY <measure>
WHERE <filter-expression>
{PARTITION BY <partition-parameter> IN <number> STEPS}
```

- Example
  - ANALYZE WORKLOAD WITH Workload\_A SUMMARIZING USING Count PARTITION BY Query\_Type
  - ANALYZE CONFIGURATION WITH Current\_Config SUMMARIZING USING Count
  - ANALYZE CONFIGURATION New\_Config TOP 3 BY Storage
  - ANALYZE CONFIGURATION WITH Current\_Config SUMMARIZING USING Count WHERE (Num-Columns=2) AND (Is-Clustered=FALSE)

41



## Automated Selection of Materialized Views and Indexes

- [Agrawal, Shaudhuri, Narasayya: VLDB 2000]
- Candidate selection identifies relevant indexes, materialized views and indexes on materialized views potentially useful
- If there are  $m$  selection conditions in the query on a table-subset  $T$ , materialized views containing any subset of these selection conditions are syntactically relevant

42



## Automated Selection of Materialized Views and Indexes

- Selecting one candidate materialized view per query exactly matching each query does not work
  - In some Commercial DBMS, Nested subquery cannot appear in view definitions
- When storage is constrained, ignoring commonality across queries can result in sub-optimal quality
- If the materialized views lead to a small reduction in cost for the entire workload, we can prune

43



## Candidate Materialized Views Selection Steps

1. Select a smaller set of interesting table-subsets
2. From these table-subsets,
  - Generate a set of materialized views for each query in the workload
  - Select a best configuration for that query
3. Starting from these views,
  - Generate additional set of merged views such that the merged view can service multiple queries in the workload

44



## Finding Interesting Table-Subsets

- $TS\text{-Cost}(T)$  = total cost of all queries in the workload where table-subset  $T$  occurs
- $TS\text{-Weight}(T) = \sum \text{Cost}(O_i) * (\text{sum of sizes of tables in } T) / (\text{sum of sizes of all tables referenced in } O_i)$
- $TS\text{-Cost}$  metric has monotonic.
  - i.e. for tables  $T_1$  and  $T_2$ , if  $T_1$  is a subset of  $T_2$ ,  $TS\text{-Cost}(T_1) \geq TS\text{-Cost}(T_2)$
  - Efficient algorithm similar to frequent itemset mining can be used
- If  $TS\text{-Weight}(T) \geq c$ , then  $TS\text{-Cost}(T) \geq c$ 
  - We prune table-subsets using  $TS\text{-Cost}$  metric first
  - Next prune the table-subsets retained with  $TS\text{-Weight}$

45



## Pruning Candidate Materialized Views

- If a materialized view is not part of the best solution for even a single query in the workload, then it is unlikely to be part of the best solution for the entire workload
- Syntactically relevant materialized views for a query  $Q_i$  on all interesting table-subsets occurring in  $Q_i$

46



## View Merging

- Use a sequence of pair-wise merges
  - Determine the criteria governing when and how a given pair of views is merged
  - Enumerate the space of possible merged views
- Similar to index merging
- When we merge a pair of views,
  - All queries that can be answered using either of the parent views should be answerable using the merged views
  - Cost of answering these queries using the merged view should not be significantly higher than the cost of answering the queries without the merged views

47



## Integrating Vertical and Horizontal Partitioning

- [Agrawal, Narasayya, Yang: SIGMOD 2004]
- Horizontal and vertical partitioning are important aspects of physical design
- Like indexes and materialized views, Horizontal and vertical partitioning can impact significantly the performance of query workload
- Horizontal partitioning allows access methods (e.g. tables, indexes, materialized views) to be partitioned into disjoint sets of rows
- Vertical partitioning allows a table to be partitioned into disjoint set of columns

48





## Motivation - Horizontal Partitioning

- Horizontal partitioning allows access methods (e.g. tables, indexes, materialized views) to be partitioned into disjoint sets of rows
- Range and hash partitioning
- Today's DBAs use horizontal partitioning extensively to make database servers easier to manage
  - If the indexes and underlying table are partitioned identically (i.e. aligned), database operations such as backup and restore become much easier
  - Thus, DBAs in today's enterprise are faced with the challenging task of determining the appropriate choice of physical design consisting of partitioned tables, indexes and materialized views

49



## Complexity - Integrating Vertical and Horizontal Partitioning

- Need for an integrated approach to automating the choice of physical design
- With the inclusion of vertical and horizontal partitioning, the space of physical design significantly increases
- With incorporating alignment, different queries may lead to physical design with conflicting horizontal partitioning on the same table
  - Indexes are considered as aligned if these are horizontally partitioned in the same way as the underlying tables

50



## A Motivating Example – Integrated Approach

```
select l_returnflag, l_linestatus, sum(l_quantity), count(*)  
from lineitem  
where l_shipdate <= "1998/12/08"  
Group by l_returnflag, l_linestatus  
Order by l_returnflag, l_linestatus
```

- Plan1- First select the best un-partitioned index and next horizontally partition
  - 1. select the best un-partitioned index (I<sub>1</sub>) on columns (l\_shipdate, l\_returnflag, l\_linestatus, l\_quantity)
  - 2. Hash partition on (l\_returnflag, l\_linestatus)
- Plan 2 – Integrated Approach
  - 1. Select the best index (I<sub>2</sub>) on columns (l\_shipdate, l\_returnflag, l\_linestatus, l\_quantity) that is also range partitioned on (l\_shipdate)
- Plan 2 is about 30% faster than Plan 1 – Both indexes and horizontal partitioning can speed up the same operations in the query (grouping and selections)

51

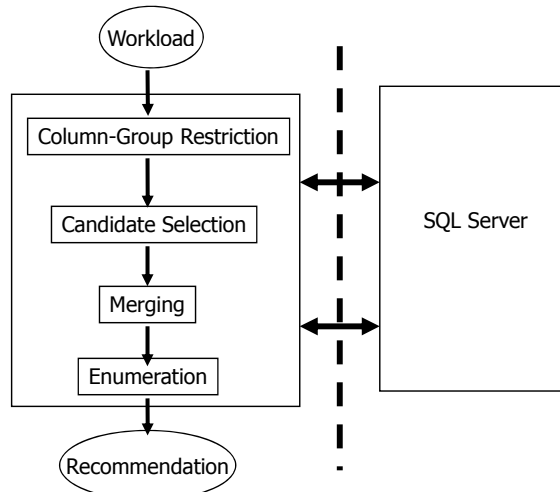


## A Motivating Example – Integrated Approach

- Selection:
  - Consider a query with the WHERE clause of Age < 30 AND Salary > 50K
  - If neither condition is very selective, but conjunction is selective, an index on salary range partitioned on the age column can improve the performance
- Join:
  - If two joining tables share identical partitioning, the join can be much faster in multiprocessor systems

52

## Architecture of Tool



53

## Architecture of Tool

- Column-Group Restriction
  - Eliminates from further consideration a large number of column-groups at least having a marginal impact
- Candidate Selection
  - Selects for each query in the workload a set of good configurations for that query
  - A physical design that is part of the selected configurations of at least one query in the workload is referred to as a candidate
- Merging
  - Consider new physical design, which can benefit multiple queries, based on candidate chosen in the Candidate Selection step
- Enumeration
  - Takes as input the candidates and produces the final solution
  - Use Greedy(m,k)

54

## An Example for Determination of Interesting Column-Group

- Define CG-Cost( $g$ ) as the fraction of the cost of all queries in the workload where column-group  $g$  is referenced
- A column-group  $g$  is interesting if  $\text{CG-Cost}(g) \geq f$  (minimum threshold)
- CG-Cost is monotonic
  - For column-group  $g_1$  and  $g_2$ , if  $g_1$  is a subset of  $g_2$ ,  $\text{CG-Cost}(g_1) \geq \text{CG-Cost}(g_2)$
  - We can use pruning used in Apriori algorithms for frequent itemset mining

55

## An Example for Determination of Interesting Column-Group

	Q <sub>1</sub>	Q <sub>2</sub>	Q <sub>3</sub>	Q <sub>4</sub>	Q <sub>5</sub>	Q <sub>6</sub>	Q <sub>7</sub>	Q <sub>8</sub>	Q <sub>9</sub>	Q <sub>10</sub>
A	1	1	1	1	1	1	1	1	1	1
B	1	1	1							
C		1	1	1	1	1	1	1	1	1
D			1							

- Define CG-Cost( $g$ ) as the fraction of the cost of all queries in the workload where column-group  $g$  is referenced
- $\text{CG-Cost}(\{A\}) = 1.0$

56



## Effectiveness of a Column-Group for vertical Partitioning

- VP-Confidence of a column-group  $g$  is defined as

$$\frac{\sum_{c \in g} \text{width}(c) | \text{Occurrence}(c) |}{\sum_{c \in g} \text{width}(c) \left| \bigcup_{c \in g} \text{Occurrence}(c) \right|}$$

- where
  - $c$  is a column belonging to column-group  $g$ ,
  - $\text{width}(c)$  is the average width in bytes of  $c$
  - $\text{Occurrence}(c)$  is the set of queries in the workload where  $c$  is referenced
- $\text{VPC}(\{A,B\}) = 13/20 = 0.65$

57



## References (General Background)

- Surajit Chaudhuri, Benoît Dageville, and Guy M. Lohman. Self-Managing Technology in Database Management Systems. Tutorial presented at VLDB 2004.
- Gerhard Weikum, Axel Mönkeberg, Christof Hasse, and Peter Zabback. Self-tuning Database Technology and Information Services: from Wishful Thinking to Viable Engineering. VLDB 2002.
- Jeffrey O. Kephart and David M. Chess. The Vision of Autonomic Computing. IEEE Computer 36(1): 41-50 (2003).
- Paul Horn. Autonomic Computing: IBM's Perspective on the State of Information Technology. IBM White paper.
- Surajit Chaudhuri (editor). Special Issue on Self-Tuning Databases and Application Tuning. IEEE Data Engineering Bulletin 22(2) June 1999.

58



## References (MS SQL SERVER)

- Surajit Chaudhuri and Vivek R. Narasayya. An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. *VLDB* 1997.
- Surajit Chaudhuri, and Vivek R. Narasayya. AutoAdmin "What-if" Index Analysis Utility. *SIGMOD* 1998.
- Surajit Chaudhuri and Vivek R. Narasayya. Index Merging. *ICDE* 1999.
- Surajit Chaudhuri, Mayur Datar, and Vivek R. Narasayya. Index Selection for Databases: A Hardness Study and a Principled Heuristic Solution. *IEEE TKDE* 16(11): 1313-1323 (2004).
- Sanjay Agrawal, Surajit Chaudhuri, and Vivek R. Narasayya. Automated Selection of Materialized Views and Indexes in SQL Databases. *VLDB* 2000.
- Sanjay Agrawal, Vivek R. Narasayya, and Beverly Yang. Integrating Vertical and Horizontal Partitioning Into Automated Physical Database Design. *SIGMOD* 2004.
- Nicolas Bruno and Surajit Chaudhuri. Automatic Physical Database Tuning: A Relaxation-based Approach. *SIGMOD* 2005.
- Mariano Consens, Denilson Barbosa, Adrian Teisanu, and Laurent Mignet. Goals and Benchmarks for Autonomic Configuration Recommenders. *SIGMOD* 2005.
- Sanjay Agrawal, Surajit Chaudhuri, Lubor Kollár, Arunprasad P. Marathe, Vivek R. Narasayya, and Manoj Syamala. Database Tuning Advisor for Microsoft SQL Server 2005. *VLDB* 2004.
- Surajit Chaudhuri, Ashish Kumar Gupta, and Vivek R. Narasayya. Compressing SQL Workloads. *SIGMOD* 2002.