Database Tuning and Self-Tuning

Dr.-Ing. Eike Schallehn

OvG Universität Magdeburg
Fakultät für Informatik
Institut für Technische und Betriebliche Informationssysteme

2012
Overview

- **Database Tuning:**
  What are issues, basic principles, and common techniques for optimizing the performance of a database system?

- **Database Self-Tuning:**
  What are current approaches to automate the database tuning process and what techniques can be used for a possible full automation in the future?
Literature: Database Tuning

- **Dennis Shasha**: *Database Tuning - A Principled Approach*, Prentice-Hall 1992
  - According tutorial at VLDB 2002
  - According tutorial at SIGMOD 2002

- **Sitansu S. Mittra**: *Database Performance Tuning and Optimization*, Springer Verlag 2002

- **Michael J. Corey, Michael Abbey, Daniel J. Dechichio**: *Tuning Oracle*, Oracle Press 1994

- **Gunter Saake, Andreas Heuer**: *Datenbanken: Implementierungstechniken*, MITP-Verlag Bonn, 1999

- **Klemens Böhm**: Vorlesung *Datenbankimplementierung und -Tuning*, Kapitel 11: Tuning relationaler Datenbanken
Literature: Database Self-Tuning

- **Surajit Chaudhuri, Vivek Narasayya**: *Self-Tuning Database Systems: A Decade of Progress*, VLDB 2007, Ten Year Best Paper Award

- **Kai-Uwe Sattler**: *Self-Tuning in DBMS: Techniken und aktuelle Systemunterstützung, DB-Stammtisch an der HTW Dresden*, November, 2006

- **Surajit Chaudhuri, Benoit Dageville, Guy Lohman**: *Self-Managing Technology in Database Management Systems*, VLDB 2004 Tutorial
Part I

Database Tuning
Database Tuning: Overview

- What is Database Tuning? Which aspects does the term include?
- What are basic principles of database tuning?
- Why is database tuning such a hard task?
- An overview of common techniques for database tuning
- One example technique in some detail: Index Tuning
Database Tuning Definition

- According to [Shasha 1992]:
  
  *Database tuning is the activity of making a database application run more quickly. „More quickly“ usually means higher throughput, though it may mean lower response time for some applications. To make a system run more quickly, the database tuner may have to change the way applications are constructed, the data structures and parameters of a database system, the configuration of the operating system, or the hardware.*

- A rather general definition used for this lecture

**Definition (Database Tuning)**

*Database Tuning* comprises all activities carried out to meet requirements regarding the performance of a database system.
What is the goal of Database Tuning?

- **Improve performance**! But performance may mean several things for a computer system: quality of processing (and results), availability, usability, etc.

- Database tuning mostly refers to runtime performance and related aspects, e.g.
  - **Throughput**: number of queries or transactions that can be processed in a fixed time
  - **Response time**: time from initiation of single query until full result is available
  - **Resource consumption**: temporary (e.g. CPU, memory) or constant (e.g. harddisk) use of resources

- **Watch out**: some of these goals can be contradicting!

- General approach of optimization: set some goal(s) as a constraint (e.g. maximum resource usage) and find the best possible solution for a specific goal of interest (e.g. throughput)
What can be tuned?

- **Hardware**
  - Used components for CPU, main memory, harddisks, backup solutions, network Communication, ...

- **Operating System (OS)**
  - System parameters and configuration for IO, network communication, process management, ...

- **Database Management System (DBMS)**
  - System configuration and parameters, database schema (views, tables), storage structures, ...

- **Application**
  - Users, queries, transactions, interfaces, Mappings, …
Focus in this Lecture

- **Here focus on actual Database Tuning**, i.e. tuning properties of the running DBMS (database server) and the managed database (DB)

- In an ideal world application tuning should actually not be necessary because DBMS optimizes accesses, but
  - SQL Tuning: reformulating queries (semantically equivalent or not) may result in better runtime performance
  - Transaction Tuning: adjusting transactions (and possibly application logic) for better runtime performance, e.g. break long transactions to avoid lock conflicts, relax consistency requirements (MVCC), etc.

- **Operating System tuning**
  - Provide sufficient processing power and data flow (IO, network) channels for the database system
  - Things to be tuned: process and thread management, virtual memory management, file system, network
A Short Remark on Hardware Tuning

- **KIWI**: Kill It With Iron!
  - „Why should we call in an expensive expert to tune our system, when buying cheap hardware can solve the problem?“
  - Can always be considered first
  - Limits of KIWI-Approach: can usually only improve performance by some linear factor and does not scale for future requirements

- General task of hardware tuning: provide suitable and fitting components that support the database system in an optimal way (multiprocessor architectures, fast and sufficient RAM, RAID-system as secondary storage, etc.)
The Database Tuning Quadrilemma

**Application**
How is the database used and what are the tuning goals?

**DBMS**
What possibilities of optimization does it provide?

**Operating System (OS)**
How does it manage hardware resources and services used by the overall system?

**Hardware**
What are suitable hardware components to support the performance requirements?

Fully efficient Database Tuning requires deep knowledge about …
Who does Database Tuning?

- **Database and application designers**
  - During database development (physical database design) and initial testing and evaluation
  - Database designers usually have strong knowledge about the application, fair to good knowledge about the DBMS, but maybe only fair to no knowledge about OS and hardware

- **Database administrators**
  - During ongoing system maintenance
  - Adjustment to changing requirements, system properties (e.g. data volume), and system environment (e.g. new hardware)
  - Administrators usually have a fair knowledge about DBMS, OS, and Hardware, and their knowledge about the application depends on the given organizational structure

- **Database experts** (consultants, in-house experts)
  - During system re-design, troubleshooting (solving constant or possible future problems), or fire fighting (solving urgent problems)
  - Consultants usually have a very strong knowledge about DBMS, OS, and Hardware, but have little knowledge about the current application
Four Basic Principles (according to [Shasha 1992])

1. **Think globally, fix locally**
   - Measure the right quantities and come to the right conclusions
   - Localize problem by identifying a bottleneck (a part of the system that limits overall performance) and resolve it

2. **Partitioning breaks bottlenecks**
   - When you find a bottleneck, first try to speed up that component
   - If that does not work, then partition: divide the load over more resources or spread the load over time

3. **Startup and running costs**
   - Most components (hardware, OS services, functionality of the dbms) devote a substantial portion of their resources to starting up
   - Try to „keep things up and running“, avoid startups

4. **Render onto server what is due onto server**
   - Try to balance computation load between application and server
   - Let server do, what the server dos best, and application respectively
Basic Principles: DB Tuning as a (continuous) Process

**Overall system continuously changes**
Data volume, # of users, # of queries/TXNs, usage patterns, used software components (versions), hardware, etc.

**Requirements may change**
New company/organizational policies, new dependencies from other applications, etc.

- **Identify Exisiting Problem**
  
  Current performance requirements are not fulfilled

- **Monitor**
  
  system behavior and **identify** cause of problem
  
  Observe and measure relevant quantities, e.g. time spent in queries, main memory, io, etc.

- **Apply changes to solve problem**
  
  Adjust system parameters, remove bottlenecks, add resources, add indexes, etc.

**Problem solved**
Basic Principles: Controlling Trade-Offs

- Database tuning very often is the process of decision about costs for a certain solution or activity compared to its benefits → trade-off situation
  - **Costs**: monetary costs (for hardware, software, working hours) or more technical costs (resource consumption, impact on other aspects)
  - **Benefits**: improved performance (monetary effect most often not easily quantifiable)

Some examples
- Adding indexes → benefit: better query response time – costs: more harddisk space used, update processing time increases
- Schema denormalization: → benefit: better query response time – costs: need to control redundancy within tables
- Replace common disk by RAID-system → benefits: improved IO-performance, consistency, and availability – costs: hardware costs

- It’s not always about trade-offs! E.g. fixing a performance problem caused by a falsely set system parameter
Basic Principles: Don’t forget the 80/20 rule!

- **80/20 Rule (Pareto Principle):** by applying 20% of the efforts one can achieve 80% of the desired effect, while to achieve the remaining 20% effect takes 80% of the efforts invested.

- **Consequences for database tuning**
  - 100% effect = fully optimized system
  - Fully optimized system probably beyond necessary requirements
  - Colloquial: "a little bit of DB tuning can help a lot"
  - Solves DB tuning quadrilemma: one does not need to be an expert on all levels of the system to be able to implement a sufficient solution
  - So ... don’t panic!
Basic Principles: Tuning Tools

- Special programs with support for **monitoring**, i.e.
  - online inspection and/or
  - statistics gathering

  and **analysis**, i.e. mapping
  - usage of DBMS (queries, TXNs) to
  - resource consumption (CPU, IO)

- Most often specific tools for certain DBMS (deeply integrated with system itself using internal interfaces)
Database Tuning and Self-Tuning

DBMS Reference Architecture

Data System: translation and optimization of user queries, access and integrity control, access path selection, …

Access System: implementation of operations (e.g. relation and index scans, sorting, joins), concurrency control, data dictionary, …

Storage System: managing records on pages, access path management, lock management, log and recovery, …

Buffer Management: manage main memory region (buffer) to optimize IO accesses, page replacement, …
DB Hardware Tuning Overview

- Add, increase, or improve components
  - Memory
  - CPUs
  - Disks
  - Bus and network bandwidth
  - ...

- Use RAID systems
DB Operating System Tuning Overview

- Threads
  - Priorities
  - Switching
  - Multiprogramming Level (MPL) of the DB

- Adjust file system/disk layout

- Driver configuration for specific hardware components
DB Buffer/Memory Tuning Overview

- Adjust memory usage
- Adjust page replacement
- Control prefetching strategy
- Adjust logging and recovery strategy
DB Storage System Tuning Overview

- Adjust page and file properties
- Placement (allocation) of logical files (tables and indexes) and logs
- Partitioning of files (physical aspects: how to partition?)
- Index tuning (physical aspects: how to implement indexes?)
- Adjust locking strategies of TXNs
- Adjust deadlock handling of TXNs
- Distribution and replication design for Distributed DBS
DB Access System Tuning Overview

- Index tuning (logical aspects: which indexes?)
- Partitioning (logical aspects: which tables should be partitioned?)
- Materialized views
DB Data System Tuning Overview

- Optimizer hints
- Database statistics and cost models
- Control optimization goal
DB Application Tuning Overview

- **Transaction tuning**
  - TXN chopping
  - Adjusting isolation levels

- **Query tuning**
  - Semantically equivalent re-writing
  - Semantically non-equivalent re-writing

- **Schema tuning**
  - Normalization
  - Denormalization
  - Vertical partitioning
One Tuning Technique in Detail

- **Index Tuning**

- Discussed in *some* detail (a separate lecture could be held on this topic alone)
Index Tuning

- Index tuning one of the most often applied tuning measures
- Great benefits (improved response time) with little effort for the database designer/administrator (if applied correctly)
- Cost of additional resource consumption (disk space) most often acceptable
- Strong support within all available DBMS (index structures, index usage controlled by optimizer)
Index Tuning: Basic Aspects of Indexes

- Main goal: avoid searching the full table (relation/table scan) to find few records
- Pre-computed and stored access path (tree, hash table) to provide fast access based on specific value(s) for attribute(s) → key(s) and key value(s)
- Two main kinds of indexes:
  - One primary index for a table typically based on primary key → each key value corresponds to one record → most often data is stored according to that index (clustered) → most DBMS create this index automatically if a PRIMARY KEY is specified within the CREATE TABLE-statement
  - Several secondary indexes to support access via other keys → one key value may correspond to several records in list of references → no influence on data organization (non-clustered)
Index Tuning: Common Index Types

- **B+-Trees**: supported by all DBMS for primary and secondary indexes (details →)

- **Hash Indexes**: hash table to control allocation (primary index) in some systems (e.g. optional in Oracle)

- **Multidimensional Indexes**: R-Trees for multimedia or spatial data defined within DBMS extensions for according data types (e.g. Oracle, IBM DB2, MySQL)
Index Tuning: Where Indexes can be used

- **Conditional selection** of tuples in **WHERE**-clause (point queries, range queries, multi-point queries, etc.)
  - Mapping constant predicates, e.g. `matrnr=123456` or `age BETWEEN 42 AND 47`, to key + key-value(s) for index lookup

- **Grouping** according to the **GROUP BY**-clause
  - Index on grouping attributes, e.g. `country, year` for `GROUP BY country, year`
  - Tuples of one group must be adjacent in the corresponding index and can easily be scanned

- **Sorting** according to the **ORDER BY**-clause
  - Index on sorting attributes, e.g. `revenue` for `ORDER BY revenue DESC`
  - Index represents per-computed order, no further processing except for index scan required
**Duplicate removal** caused by \texttt{SELECT DISTINCT} (explicit) or \texttt{UNION} (implicit)

- Duplicates must be adjacent in any (!) index on the input relation
- Can be easily detected based on index scan

**Projection** to columns in \texttt{SELECT}-clause

- If all returned columns are included in the index result can be returned from the index without touching the data pages → *Covering Index*
- E.g.
  ```sql
  SELECT name, firstname
  FROM student
  WHERE matrnr=123456
  can be answered from index \texttt{idx1(matrnr,name,firstname)} without reading any record
  ```
- Some systems, e.g. IBM DB2, support \texttt{INCLUDE ONLY}-columns which are not part of the actual key, but are stored with it inside the tree
Natural or equi-join

- Sort-merge-join special implementation of join which exploits ordering of relations based on join keys
- Allows parallel scans of input relations (sizes $n$ and $m$) with $O(n + m)$ (better than Nested-loop-join with $O(n \times m)$
- Efficient, even when order has to be established first
  $O(n \log n + m \log m)$
- Indexes on join key in relations represent pre-computed order
  $O(n + m)$ complexity for join
- Especially efficient if indexes are clustered
Index Tuning: Further Important Considerations

- **Table size**
  - For small tables (especially when fitting into a small fraction of the main memory) index usage may cause an unnecessary overhead.

- **Data distribution**
  - Whether an index is useful for a query also depends on semantics of columns (number of values, upper and lower bound for values, etc.).
  - E.g. scanning for `student.age>20` will return almost all tuples → even with index full relation will be scanned (or worse: pages are read more than once with non-clustered index).
  - E.g. scanning for `student.gender='female'` will end up as a relation scan (or worse ...) for non-clustered index on `gender` because all pages contain male and female students.

- General: response time improvement heavily depends on both criteria.

- Only tune indexes for critical queries (slow response time, many IO operations)!

Schallehn (FIN/ITI)
Index Tuning: Considering the Downsides of Indexes

- **Storage costs for indexes**
  - Indexes may use considerable harddisk space (most often acceptable)
  - Indexes are primary objects for storage in buffer (main memory)
  - Placement of indexes on (dedicated) disk should also be considered

- **Costs for index updates**
  - Updates on key attributes may trigger index update or possibly expensive reorganization
  - Index usage may be prohibitive in scenario with many constant updates on database objects
  - E.g. positions of moving objects, surveillance data, sensor data

- **Locking overhead and lock conflicts**
  - Indexes are hot spot objects (especially root and upper levels of trees)
  - Write operations and reorganizations may more easily cause lock conflicts and deadlocks
Index Tuning: the Global View

- So far only **local view**: are indexes useful for **one query**
- Now **global view**: which indexes are (most) useful for the (typical) **workload** (all queries)
- May require trade-off decisions
  - Compare benefit/cost ratio for indexes and decide which one to implement
  - Requires consideration of space constraints

**Index subsumption**
- Some indexes may provide same or similar benefit of other indexes
- E.g. prefix indexes → an index $\text{idx1}(a, b, c)$ can replace indexes $\text{idx2}(a, b)$ and $\text{idx3}(a)$
- Subsumed indexes need not to be implemented

**Index merging**
- Given certain rules, two indexes supporting two different queries can be merged into one index supporting both queries with similar behavior
Index Tuning: Index Merging

```
SELECT DISTINCT region, product
FROM sales
WHERE region = 'East'
```

```
SELECT region, year, count(*)
FROM sales
GROUP BY region, year
```

Supporting index: idx1(region, product)

Supporting index: idx2(region, year)

Index supporting both queries: idx3(region, year, product)
Index Tuning: the Global View /2

Overall index tuning process for a database could be as follows:

1. Identify critical queries
2. Create possible indexes to tune single queries
3. From set of all indexes remove subsumed indexes
4. Merge indexes where possible
5. Evaluate benefit/cost ratio for remaining indexes (need to consider frequency of queries/index usage!)
6. Pick optimal index configuration satisfying storage constraints
Index Tuning: Concluding Remarks

- Typical aspects of query tuning mentioned
- Many more things to consider for
  - Specific DBMS
  - Specific application
  - Specific data types
  - ...

- Dependencies with other tuning measures, e.g.
  - Materialized views can be indexed
  - Materialized views can replace indexes
  - Indexes can replace materialized views
  - Indexes can be partitioned
  - Schema tuning may make index re-design necessary
  - TXN lock tuning for indexes
  - ...
Part II

Database Self-Tuning
Database Self-Tuning: Overview

- **Introduction and Motivation**
  - Why is it necessary?
  - What is self-tuning?
  - What are related terms?

- **Basic Principles**
  - The Self-Tuning Cycle (Feedback Control Loop, IBM’s MAPE)
  - Trade-off elimination
  - Static vs. online optimization
  - Overhead for Self-Tuning

- **Overview of Self-Tuning Approaches**

- **Some Details**
  - Physical design tuning (mostly index tuning)
Database Management Systems were created to make data access easy.

- Declarative query language
- Query optimizer to provide most efficient access
- Internal storage structures hidden from the user
- ...
The implementation and maintenance of a (high-end) database application and meeting specific performance requirements is a very complex task.

- Choosing the right hardware
- Configuring the OS and DBMS settings
- Implementing a suitable physical design (indexes, MVs, partitioning)
- ...

DB Tuning: Reasons for ongoing Costs

- See DB Tuning principles → continuous tuning necessary
- DB systems constantly change
  - Data is changing (schema, size of tables, data distribution, ...)
  - Data usage is changing (number of users, new user groups, typical access patterns, applications, ...)
  - Database environment (hardware, network, operating system, concurrent applications, ...)
- Requirements may change frequently
  - Performance
  - Scalability
  - Availability, safety, security, ...
- DB administrators spend more than 50% of their time just to keep the system „up and running“
DB Self-Tuning: Motivation

Main goals:

▶ Decrease running costs for maintaining and administrating a database system
▶ Automate as many tasks as possible
▶ „Reduce number of tuning knobs“
▶ Meeting performance requirements with less efforts

DBMS providers currently working on making their systems more easily manageable

→ support/propagate efficient usage of DBMS product
→ increased usability
→ decreased running costs
→ increased customer acceptance
→ competitive advantage

According activities

▶ IBM Autonomic Computing Initiative
▶ Microsoft AutoAdmin project for MS SQL Server
▶ Oracle Self-Tuning Architecture since Oracle 10g
Remember, ...

**Application**
How is the database used and what are the tuning goals?

**DBMS**
What possibilities of optimization does it provide?

**Operating System (OS)**
How does it manage hardware resources and services used by the overall system?

**Hardware**
What are suitable hardware components to support the performance requirements?

Fully efficient Database Tuning **requires** deep knowledge about …
DB Self-Tuning: Basic Idea

**Application**

Knowledge about application from analyzing queries, TXNs, schema, data, etc.

**DBMS**

Naturally, DBMS knows best about its functionality and possible tuning options.

**DBMS itself is best Tuning Expert!!!**

Knowledge about OS encoded in platform-specific code + runtime information via OS interfaces.

**Operating System (OS)**

Knowledge about hardware encoded in platform-specific code + runtime information via OS interfaces.

**Hardware**
DB Self-Tuning: Basic Idea /2

- Analyzing information about previous and current database usage allows prediction of future behavior and applying necessary changes

- **Necessary**: requirements need to be specified
  - Future task of DB designers/administrators
  - Lifts DB tuning from a technical to a strategical level

- **Limitations**:
  - Can not exploit information outside the overall system (e.g. possible improvements with new hardware) or about a foreseeable future (e.g. "the number of users will double next month")
  - Possible lack of transparency of DBMS functionality ("there’s a new performance problem ... what has the DBMS just done??")
DB Self-Tuning: Definition

**Definition (Database Self-Tuning)**

**Database Self-Tuning** describes the capability of a DBMS to optimize its own functionality, parameters, and internal structures for given a database system to improve the performance and meet specified requirements.

- Shifts responsibility for tuning from users of DBMS (designers, administrators, consultants) to providers of DBMS (by developing and implementing according solutions for DBMS)
- Can be seen as a special field in the more general context of self-management of systems (self-*-techniques)
IBM Autonomic Computing Initiative (2001)

Autonomic System
Self Management :=

- Self Configuring
- Self Healing
- Self Optimizing
- Self Protecting

Controlled by general policies and rules

[Source: http://www.ibm.com/autonomic]
DB Self-Tuning: Basic Principles

1. **Trade-off elimination** describes (one) important goal of self-tuning: let the DBMS decide as much, as it can decide.

2. **Static vs. online tuning**: temporal aspect of how and when tuning decisions can or must be made.

3. **The self-tuning cycle**: outline of automatic online (continuous) decision process.

4. **Self-tuning overhead**: considering the negative impact of self-tuning measures.
Self-Tuning Principles: Trade-off Elimination

- **Trade-off Elimination:** if possible, remove parameter/tuning knob, or otherwise, replace hard to control, low-level parameter(s) with easy to control, high-level parameter(s) (policies, strategies)
  1. „Automate straight-forward decisions“
  2. „Replace hard decisions with easy decisions“

- Both aspects must be based on using available information in decision (support) process

- Examples:
  - Adjust memory usage (buffer size and others)
  - Choosing a buffer replacement strategy
  - Choosing an optimal page size
  - Index tuning (improves query response time, slows down updates)
  - B+-trees vs. Hash indexes
  - ...
Self-Tuning Principles: Static vs. Online

- **Static self-tuning:**
  - Self-tuning is performed once or frequently
  - Initiated manually or triggered by DBMS
  - Actual processing (analysis + execution of tuning measures) can be decoupled from DBMS to a large degree
  - Can be supported by external tools
  - Suitable for adjustments to slowly changing or stable properties of a database system

- **Online self-tuning:**
  - Self-tuning is performed continuously
  - Deeply integrated with DBMS functionality
  - Self-tuning algorithms (e.g. ARC)
  - Suitable for adjustment to frequent or continuously changing properties of a database system
Static Self-Tuning: Example

Physical Design Tuning

- Decision about:
  - Indexes
  - Materialized views
  - Partitioning
  - ... (depending on given DBMS)

- State of the art (→)
  - External tools (advisors, wizards) for creating recommendations
  - Based on (automatically) gathered information about workload and database statistics
  - Manually controlled, but automatic decision process
  - Incorporates query optimizer to estimate benefits/costs of physical design structures for single queries („what if“-analysis)
Online Self-Tuning: Example

Statistics Management

- Two basic principles
  1. Automatically monitor number of changes and trigger re-computation if critical threshold is reached
  2. Use query feedback to improve quality (granularity, correction factors, etc.) of statistics

  - Query optimizer selects plan and keeps information about estimated cardinalities of intermediate results
  - Plan is executed and real cardinalities are derived
  - In case of „significant“ differences changes to database statistics are triggered
Self-Tuning Principles: The Self-Tuning Cycle

- Based on manual tuning process described before
- **In general:** abstract description of typical tasks in automated decision processes
- **For specific self-tuning tasks:** fine-grained definition of single steps, their input and output parameters and according interrelations
- Based (more or less loosely) on concepts from **Control Theory**
Core principle in IBM’s Autonomic Computing Vision

MAPE = Monitor + Analyze + Plan + Execute
Self-Tuning: State of the Art

- Currently growing support for Monitoring
  - DBMS automatically stores detailed information about queries, runtimes, resource usage, etc.
  - E.g. Oracle’s Automatic Database Diagnostics Monitor (ADDM) and Automatic Workload Repository (AWR)

- Static support for aspects of Analysis and Plan
  - External tools to suggest tuning measures
  - Advisors and wizards for physical design, configuration, SQL query analysis for all big commercial DBMS

- Some support for automatic Execution of tuning measures (full self-tuning cycle)
  - Self-tuning algorithms for trade-off elimination
  - Current solutions incorporated in major DBMS for memory and buffer management
  - IBM’s self-learning optimizer
One field in some detail: **Physical Design Tuning**

Often most successful tuning measure for database systems

Strong support by all vendors: advisors and wizards to recommend suitable set of indexes, MVs, partitioning, etc.

In the following, some details on how these advisors work + a look into a possible future

To avoid complexity: **focus on index self-tuning**
Index Self-Tuning: Problem

- Automatic decision about optimal index configuration theoretically requires considering all possible indexes in a database → combinatorial explosion
- Number of possible indexes for one table (!) with \( n \) columns:
  \[
  \sum_{k=1}^{n} \frac{2^k \cdot n!}{(n-k)!}
  \]
  
  This does not yet include different index types and variants, sorting order, etc.
- Can be narrowed down by just considering „useful“ indexes → analyze queries according to specific rules (→ where indexes can be used)
- Useful indexes can be discovered by query optimizer during „what if“-analysis
Index Self-Tuning: „What if“-Analysis /1

- **Local view:** special mode of current optimizers to recommend indexes for a single query

- **WHAT IF** all possible indexes would exist? Which ones were used for this query?

- **Two approaches:**
  - Create metadata for possibly useful (hypothetical) indexes according to analysis of attributes used in queries before optimization → query optimizer works based on these metadata and proposes to use some of the indexes
  - More recent optimization: query optimizer creates hypothetical index (and according metadata) on-the-fly wherever an index appears to be useful

- If hypothetical indexes are used in the found query plan, these are recommended indexes for this query
Index Self-Tuning: „What if“-Analysis /2

Query \rightarrow \text{Heuristics} \rightarrow \text{Hypothetical Objects}

\text{Optimizer} \rightarrow \text{Plan } P_A

Profit? \rightarrow \text{yes} \rightarrow \text{Hypothetical Objects in } P_B = \text{Recommendation}

\text{Optimizer} \rightarrow \text{Plan } P_B
Global view: for a given workload (including updates → consider indexing overhead), what is the best index configuration to support all queries?

Based on „what if“-analysis of all single queries: workload → set of recommended indexes

Index set can be pruned (e.g. remove subsumed indexes) and indexes can be merged (→ index tuning) → reduced index candidate set

...
The **problem definition** now is:

**Input:**
- Reduced set $\mathcal{I}_C$ of candidate indexes $I \in \mathcal{I}_C$ ($\text{benefit}(I)$: response time improvement, $\text{cost}(I)$: size) for single queries
- Space constraint $S$ of available space for indexes

**Output:**
- Set of recommended indexes $\mathcal{I}_R \subset \mathcal{I}_C$ for workload with optimal overall benefit

\[
\sum_{I \in \mathcal{I}_R} \text{benefit}(I) \rightarrow \max
\]

fulfilling space constraint

\[
\sum_{I \in \mathcal{I}_R} \text{size}(I) < S
\]

Handled as knapsack problem (though it is not quite because of dependencies, e.g. benefit of two indexes used in merge-join greater than sum of benefits of single indexes)

Applied solution: mix of greedy selection + genetic algorithms to improve solution
Index Self-Tuning: Index Advisors

- Workload
- Advisor Tool
  - Index Candidate Selection
  - Index Merging
  - Enumeration of Candidates
- "What if" analysis for single queries
- Query Optimizer
- Recommendations

DBMS

Schallehn (FIN/ITI)
Index Self-Tuning: MS SQL Server DTA

Following slides: screenshots of MS SQL Server Database Tuning Advisor (DTA)
Database Engine Tuning Advisor will recommend clustered and nonclustered indexes to improve performance of your workload. Newly recommended structures will be partitioned to provide the best performance for the specified workload. All existing structures will remain intact in the database at the conclusion of the tuning process.
Database Engine Tuning Advisor

Estimated improvement: 72%

Partition Recommendations:

Index Recommendations:

- Database Name: [dbo]CUSTOMER
- Object Name: create
- Recommendation: _dia_stal_437576557_1_4
- Target of Recommendation: _dia_stal_437576557_7_1
- Partition Size (KB): 4456
- Definition: [IC_CUSTKEY],[IC_NATIONKEY]

- Database Name: [dbo]CUSTOMER
- Object Name: create
- Recommendation: _dia_index_CUSTOMER
- Target of Recommendation: _dia_index_CUSTOMER
- Partition Size (KB): 656
- Definition: [IC_CUSTKEY],[IC_CUSTKEYAsc]

- Database Name: [dbo]CUSTOMER
- Object Name: create
- Recommendation: _dia_index_CUSTOMER
- Target of Recommendation: _dia_index_CUSTOMER
- Partition Size (KB): 3632
- Definition: [IC_CUSTKEY],[IC_CUSTKEYAsc],[IC_NAME]

- Database Name: [dbo]LINEITEM
- Object Name: create
- Recommendation: _dia_stal_463576711_13
- Target of Recommendation: _dia_stal_463576711_14_2
- Partition Size (KB): 132032
- Definition: [IL_RECEIPTKEY],[IL_ORDERKEY],[IL_SHIPINSTRUCT],[IL_PARTKEY]

- Database Name: [dbo]LINEITEM
- Object Name: create
- Recommendation: _dia_stal_463576711_15
- Target of Recommendation: _dia_stal_463576711_16
- Partition Size (KB): 211248
- Definition: [IL_RECEIPTKEY],[IL_ORDERKEY],[IL_SHIPDATE],[IL_MONTH],[IL_QUANTITY],[IL_SHIPINSTRUCT],[IL_SHIPMETHOD]
Index Self-Tuning: Oracle 11g SQL Access Advisor

- Following slides: screenshots of Oracle 11g SQL Access Advisor
- Sorry ... German only!
**Ergebnisse für Task:SQLACCESTPCHALL**

Task-Name: SQLACCESTPCHALL  
Status: COMPLETED  
Advisor-Modus: COMPREHENSIVE  
Scheduler-Job: ADV_SQLACCESTPCHALL  
Veröffentlichungspunkt: 1

**Gesamte Arbeitsblatt-Performance**

- Potenzial zur Verbesserung

**Workload-I/O-Kosten**

- Originalkosten (3316259)  
- Neue Kosten (237511)

**Verbesserung der Ausführungszeit der Abfrage**

- Keine Performance-Verbesserung  
- Potenzielle Performance-Verbesserung

**Empfehlungen**

- Empfehlungen  
  - Anforderungen an Speicherplatz (GB): 1,620
  - vom Benutzer angegebene Speicherplatzanpassung

- Anzahl von Empfehlungsaktionen anzeigen

**SQL-Anweisungen**

- SQL-Anweisungen: 20
  - Anweisungen, die nach Anwendung von Filtern bestehen bleiben

- Anzahl von Anweisungen anzeigen
**Gesamte Arbeitsblast-Performance**

**Empfehlungen**

**Workload-I/O-Kosten**

<table>
<thead>
<tr>
<th>Originalkosten (3315259)</th>
<th>Neue Kosten (237594)</th>
</tr>
</thead>
</table>

**Verbesserung der Ausführungszeit der Abfrage**

- Keine Performance-Verbesserung
- Potenzielle Performance-Verbesserung

**SQL-Anweisungen**

**Empfehlungen**

- **Anforderungen an Speicherplatz (GB)**
  - 1,620
- **Vom Benutzer angegebene Speicherplatzanpassung**
  - Begrenzt

**Anzahl von Empfehlungsaktionen verbergen**

<table>
<thead>
<tr>
<th>Indizes</th>
<th>Erstellen</th>
<th>Löschen</th>
<th>Behalten</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Materialized Views</td>
<td>Erstellen</td>
<td>Löschen</td>
<td>Behalten</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Materialized View Logs</td>
<td>Erstellen</td>
<td>Behalten</td>
<td>Ändern</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Partitionen</td>
<td>Tabellen</td>
<td>Indizes</td>
<td>Materialized Views</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

**SQL-Anweisungen**

- **Anzahl von Anweisungen verbergen**
- **Einfügen**
  - 0
- **Auswählen**
  - 20
- **Aktualisieren**
  - 0
- **Löschen**
  - 0
- **Konsolidieren**
  - 0
- **Übersprünge (Parsing- oder Berechtigungsfehler)**
  - 0
### Empfehlungen zur Implementierung auswählen

- **Beibehalten-Aktionen aufnehmen**

<table>
<thead>
<tr>
<th>Auswählen</th>
<th>Implementierungsstatus</th>
<th>ID</th>
<th>Aktionen</th>
<th>Aktionstypen</th>
<th>Kostenverbesserung</th>
<th>Kostenverbesserung (%)</th>
<th>Speicherplatz (MB)</th>
<th>Betroffene SQL-Anweisungen</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td></td>
<td>1</td>
<td>7</td>
<td></td>
<td>399872</td>
<td>12,99</td>
<td>470,056</td>
<td>1</td>
</tr>
<tr>
<td>✔</td>
<td></td>
<td>3</td>
<td>6</td>
<td></td>
<td>270770</td>
<td>8,80</td>
<td>5,906</td>
<td>1</td>
</tr>
<tr>
<td>✔</td>
<td></td>
<td>5</td>
<td>9</td>
<td></td>
<td>236265</td>
<td>7,67</td>
<td>32,320</td>
<td>1</td>
</tr>
<tr>
<td>✔</td>
<td></td>
<td>6</td>
<td>12</td>
<td></td>
<td>222251</td>
<td>7,22</td>
<td>0,773</td>
<td>1</td>
</tr>
<tr>
<td>✔</td>
<td></td>
<td>8</td>
<td>6</td>
<td></td>
<td>209055</td>
<td>6,60</td>
<td>0,180</td>
<td>1</td>
</tr>
<tr>
<td>✔</td>
<td></td>
<td>2</td>
<td>7</td>
<td></td>
<td>197120</td>
<td>6,40</td>
<td>0,117</td>
<td>1</td>
</tr>
<tr>
<td>✔</td>
<td></td>
<td>2</td>
<td>6</td>
<td></td>
<td>196372</td>
<td>6,38</td>
<td>152,576</td>
<td>1</td>
</tr>
<tr>
<td>✔</td>
<td></td>
<td>2</td>
<td>4</td>
<td></td>
<td>149050</td>
<td>4,84</td>
<td>146,202</td>
<td>1</td>
</tr>
<tr>
<td>✔</td>
<td></td>
<td>16</td>
<td>4</td>
<td></td>
<td>147690</td>
<td>4,80</td>
<td>205,331</td>
<td>1</td>
</tr>
<tr>
<td>✔</td>
<td></td>
<td>10</td>
<td>6</td>
<td></td>
<td>142092</td>
<td>4,62</td>
<td>204,06</td>
<td>1</td>
</tr>
</tbody>
</table>
Alerter: automatically notify administrators, when a change of the physical design configuration is recommended
  ▶ Bruno N. and Chaudhuri S. *To Tune or not to Tune? A Lightweight Physical Design Alerter*. VLDB 2006

Online Index Tuning: fully automated index tuning
  → screenshot on next slide
  ▶ Bruno, N., and Chaudhuri, S. *Automatic Physical Design Tuning: A Relaxation Based Approach*. ACM SIGMOD 2005

...
Index Self-Tuning: Next Steps /2

- ...  
- **On-the-fly Index Creation**: create necessary indexes while performing a query for usage in this or for further queries  
- **Dynamic Index Structures**: make index structures self-tuning (adaptable, access-balanced, etc.)  
  - Goetz Graefe, Harumi A. Kuno: *Self-selecting, self-tuning, incrementally optimized indexes*. EDBT 2010  
- ...
“Nonetheless, the challenge in making database systems truly self-tuning is a tall task. For example, the nature of tuning a buffer pool or tuning allocation of working memory for queries is very different from that of selecting the right set of indexes or statistics. Each such tuning problem has different abstractions for workloads and different constraints on the desired solution. Therefore, it will probably be impossible to make database systems self-tuning by a single architectural or algorithmic breakthrough. As a consequence, it will be a long journey before this goal is accomplished just as it took the automobile industry a sustained effort to reduce the cost of ownership.”

S. Chaudhuri, V. Narasayya: Self-Tuning Database Systems: A Decade of Progress, Microsoft Research, 2007
Will Self-tuning replace DB administrators?

- **No**, self-*-techniques just help to make DBMS more easily usable in complex applications
- New tasks for administrator on higher level: set strategies, policies, requirements, constraints
- Fully self-managed databases for specific systems: Web-databases (CloudStorage, ZeroAdmin databases, hosted databases etc.), embedded data management, etc.

Current problem of many self-*-techniques: lack of acceptance due to immaturity

Interesting developments in other fields of DBMS research, e.g. Column-oriented DBMS → Database Cracking: optimize storage structure dynamically according to usage
Invitation to Join Self-Tuning Research ;-) ... with a master or diploma thesis