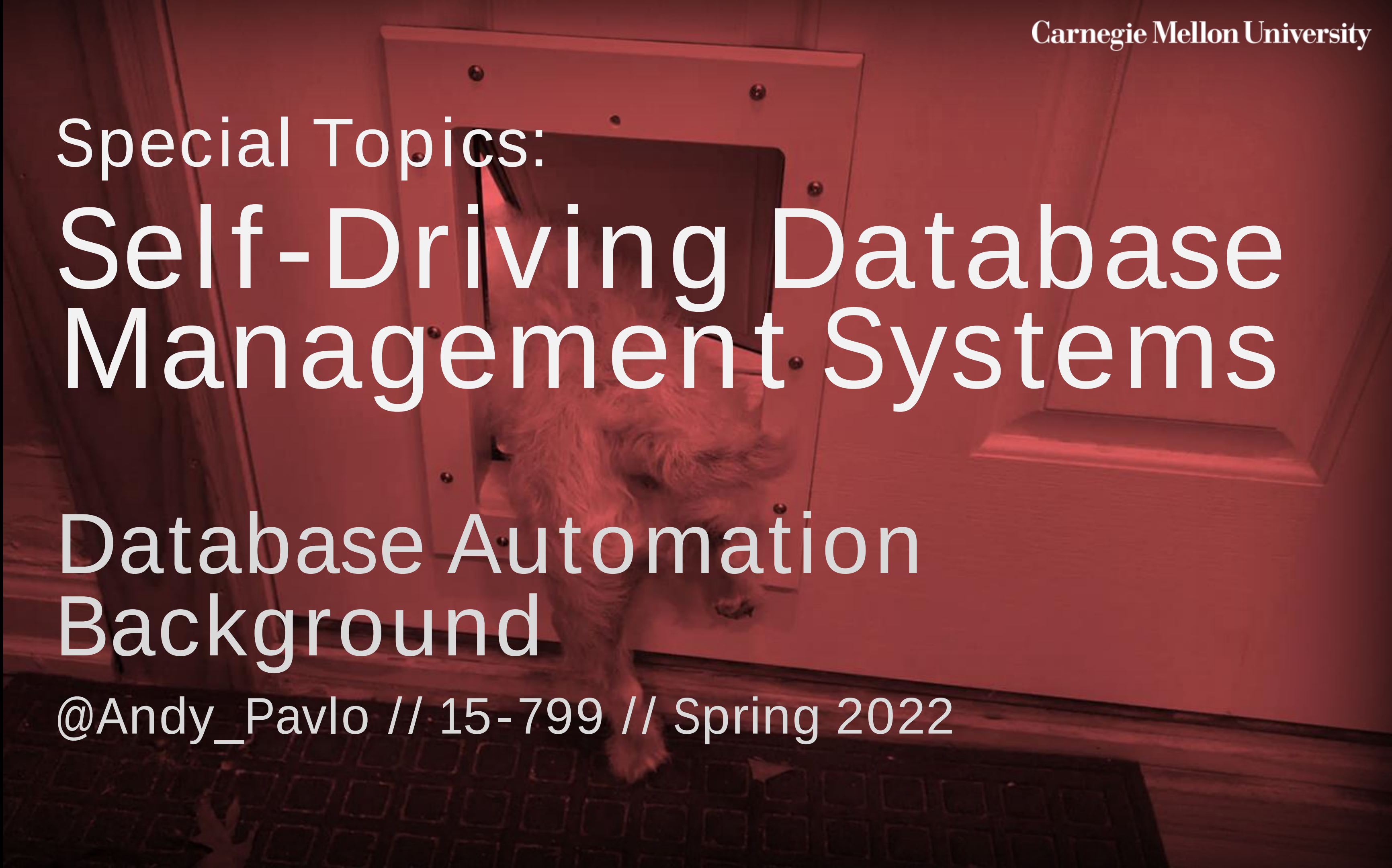
Lecture #01

Carnegie Mellon University



Special Topics:

Self-Driving Database Management Systems

Database Automation Background

@Andy_Pavlo // 15-799 // Spring 2022

amazon

OTTERTUNE

TODAY'S AGENDA

Course Logistics Overview
History of Database Automation

WHY YOU SHOULD TAKE THIS COURSE

There are more databases than ever.

Everybody has database problems.

Humans are not scalable.

Automation is the future.

Research: The problem is hard / interesting.

Industry: Every database company needs this.

COURSE OBJECTIVES

Learn about state-of-the-art methods for automated database optimization and management.

Students will become proficient in:

→ Writing database automation code
→ Proper documentation + testing
→ Code reviews
→ Working on a large code base

COURSE TOPICS

We will focus on how to optimize single-node relational database management systems via existing interfaces (e.g., SQL).

→ Non-relational DBMSs (NoSQL) also have these problems, but Andy is an apostle of Stonebraker.

We will ignore distributed deployment problems and ML-derived internal components ("learned").

COURSE TOPICS

Indexes

Knobs / Parameter Configuration

Partitioning

Query Optimization

Workload Modeling

System Behavior Modeling

Autonomous Database Systems

BACKGROUND

I assume that you have already taken an intro course on databases (e.g., 15-445/645).

→ Things that we will **not** cover: SQL, Serializability Theory, Relational Algebra, Basic Algorithms + Data Structures.

This is also **not** a ML course. We will not cover ML algorithms beyond what is discussed in papers.

→ Andy only cares about databases and whatever he can use to make databases run better.

COURSE LOGISTICS

Course Policies + Schedule:

→ Refer to course web page.

Academic Honesty:

→ Refer to CMU policy page.
→ If you're not sure, ask me.

OFFICE HOURS

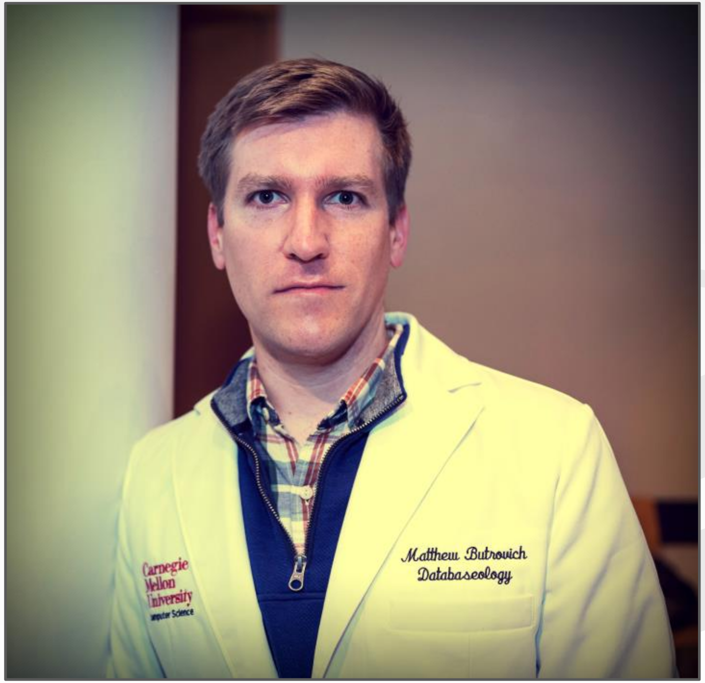
Fridays @ 2:30pm ET over Zoom
→ https://cmu.zoom.us/j/99297623809?pwd=NnNRczR6S2UvRW5DNXB5Ymk3VWtxdz09

Things that we can talk about:
→ Issues on implementing projects
→ Paper clarifications/discussion.
→ How to get a database dev job.
→ Life after the pandemic.

TEACHING ASSISTANTS

Head TA: Matt Butrovich

→ 4th Year PhD Student (CSD)
→ Reformed Gang Member (LAX)
→ #1 Ranked CMU-DB PhD Student
→ Recently married and ready to put his felonious past behind him.



COURSE RUBRIC

Reading Assignments

Paper Presentations

Programming Projects

READING ASSIGNMENTS

One mandatory reading per class (♛). You can skip **three** readings during the semester.

You must submit a synopsis **before** class:
→ Overview of the main idea (three sentences).
→ Three strengths of method (one sentence each).
→ Three weaknesses of method (one sentence each).
→ Workloads evaluated (one sentence).

Submission Form:
https://cmudb.io/15799-s22-submit

☠ PLAGIARISM WARNING ☠

Each review must be your own writing.

You may **not** copy text from the papers or other sources that you find on the web.

Plagiarism will **not** be tolerated.
See CMU's Policy on Academic Integrity for additional information.

PAPER PRESENTATIONS

Starting January 26th, one student will be assigned to present the assigned reading for that day and lead the discussion.

- → There are not an even number of papers to present per student, so we can double up.
- → You are allowed / encouraged to reach out to authors and use their slides as the starting point.

The rest of the class should participate in the discussion during and after presentation.

I will send out the sign-up sheet later this week.

PROGRAMMING PROJECTS

All projects will be implemented using PostgreSQL and NoisePage (pilot not DBMS).

You should do all development and testing on your local machine.

You will do all benchmarking using self-managed databases on Amazon EC2 (don't use RDS)

→ We will provide details next week and credits.
→ If you blow out your credit card, I cannot help you.

PROJECT #1

You will build an automatic tuning tool for PostgreSQL. It is purposely designed to be open ended to let to play around.

→ Anything is on the table as long as the DBMS doesn't crash, lose data, or produce incorrect query results.

We will provide you with infrastructure to run workloads with BenchBase.

Project #1 will be completed individually.

PROJECT #2

Each group (3-4 people) will choose a project that satisfies the following criteria:

→ Relevant to the materials discussed in class.
→ Requires a significant programming effort from <u>all</u> team members.
→ Unique (i.e., two groups cannot pick same idea).
→ Approved by me.

You don't have to pick a topic until after you come back from Spring Break.
We will provide sample project topics.

PROJECT #3

The class will work together to build a working end-to-end autonomous database prototype.

We will spend a portion of each class time after Spring Break to discuss development issues.

☠ PLAGIARISM WARNING ☠

These projects must be all your own code.

You may **not** copy source code from other groups or the web.

Plagiarism will **not** be tolerated.
See CMU's Policy on Academic Integrity for additional information.

GRADE BREAKDOWN

Reading Reviews (20%)

Presentations + Participation (20%)

Project #1 (20%)

Project #2 (25%)

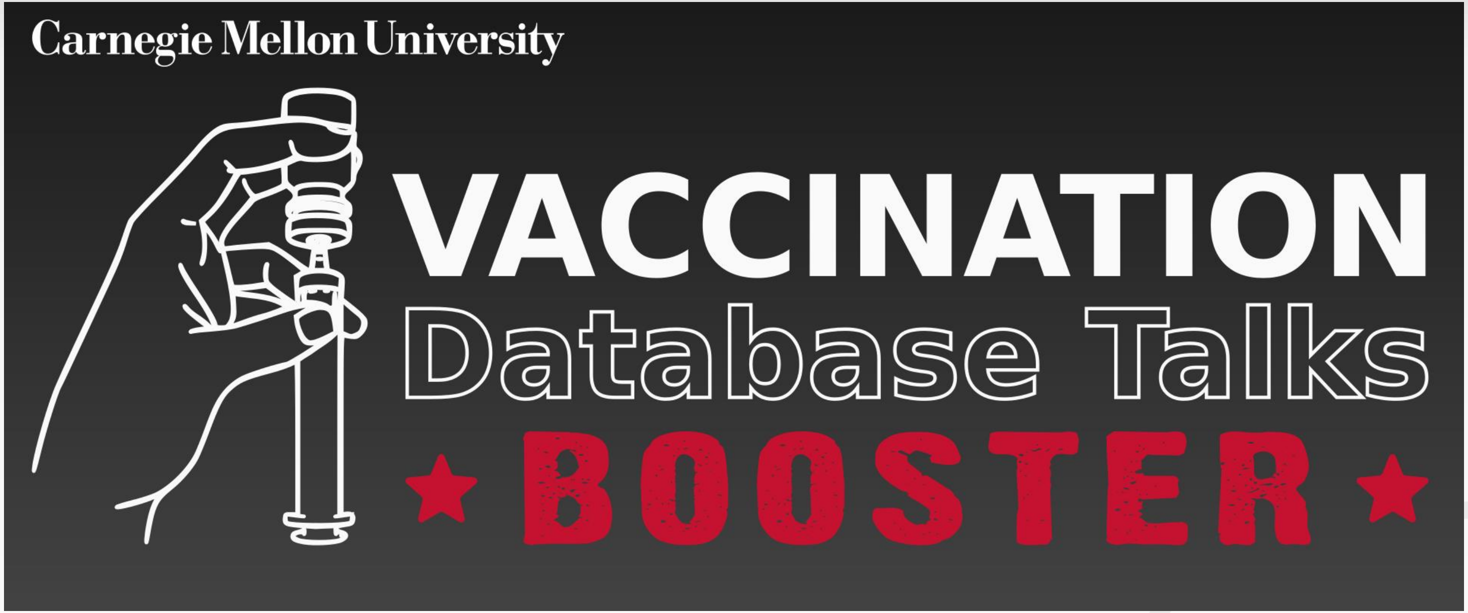
Project #3 (15%)

COURSE SLACK CHANNEL

Andy will send invitations to everyone enrolled to the CMU-DB Slack channel for the course.

If you have a technical question about the projects, please use Slack.
→ Don't email me or TAs directly.

All non-project questions should be sent to me.



Vaccination Database Tech Talks

→ Mondays @ 4:30pm (starting on Jan 31st)
→ https://db.cs.cmu.edu/seminar2022-booster

Database Automation

DATABASES

The proliferation of the Internet means that it is easy to create an application that can have thousands of users in a short amount of time.

Instead of managing data files on its own, the application should use a DBMS that is responsible for storing and querying the database.

RELATIONAL DATABASES

Physical data independence allows a DBMS to change many aspects of a database without having to change application code.

Example: If a query's execution time is too slow because it performs a sequential scan, one can add a new index to make it run faster without having to change the query's SQL.

DATABASE OPTIMIZATION

Physical Database Design
Knob Configuration
Query Tuning
Hardware Provisioning

PHYSICAL DATABASE DESIGN

Storage
→ Organization (row vs. columnar), Compression, Encoding, Denormalization, Clustering.

Indexes
→ Keys, Data Structures, Include Columns, Partial Indexes, Expression Indexes.

Partitioning
→ Sharding Keys (Horizontal), Column Splits (Vertical), Schemes, Replication.

Result Caching
→ Materialized Views, Memoization, Zone Maps.

KNOB CONFIGURATION

Resource Knobs

→ How much hardware resources to use for an internal component or task.

Policy Knobs

→ The execution behavior of the DBMS for some task.

Location Knobs

→ Filepaths, directories, network info (host/port).

QUERY TUNING

Plan Hints

→ Force the DBMS to execute a query in a specific manner.
→ Join ordering, access methods, execution strategies

Rewriting

→ Changing SQL syntax to improve the execution behavior of the DBMS.

Rerouting

→ Redirect read-only queries to replicas with idle resources.

Optimizer Statistics

→ Make sure that the DBMS has the freshest stats about the database as possible so that it makes "good" choices.

HARDWARE PROVISIONING

Vertical Scaling

→ Change the hardware capabilities of a single node (CPU, memory, disk, network).

Horizontal Scaling

→ Change the number of physical nodes in a logical database.
→ This for either distributed DBMS or replicated DBMS.

DATABASE OPTIMIZATION

Traditionally, it has been the role of the database administrator to manage all aspects of the DBMS.

With so many different options for optimizing each a DBMS, DBAs are typically an expert in one or DBMS type.

→ Core concepts are universal, but implementation specifics and behavior of all DBMSs are different.

HUMANS ARE EXPENSIVE

What is the mean salary of a DBA in the US?

→ Average DBA Salary (2020): $101,090

→ Personnel is estimated to be ~50% of the total-cost-of-ownership (TCO) of a DBMS.

The scale and complexity of DBMS installations have surpassed the ability of humans to effectively manage and optimize them individually.

Source: https://www.bls.gov/oes/current/oes151141.htm
Source: https://www.highbeam.com/doc/1P3-1149052351.html

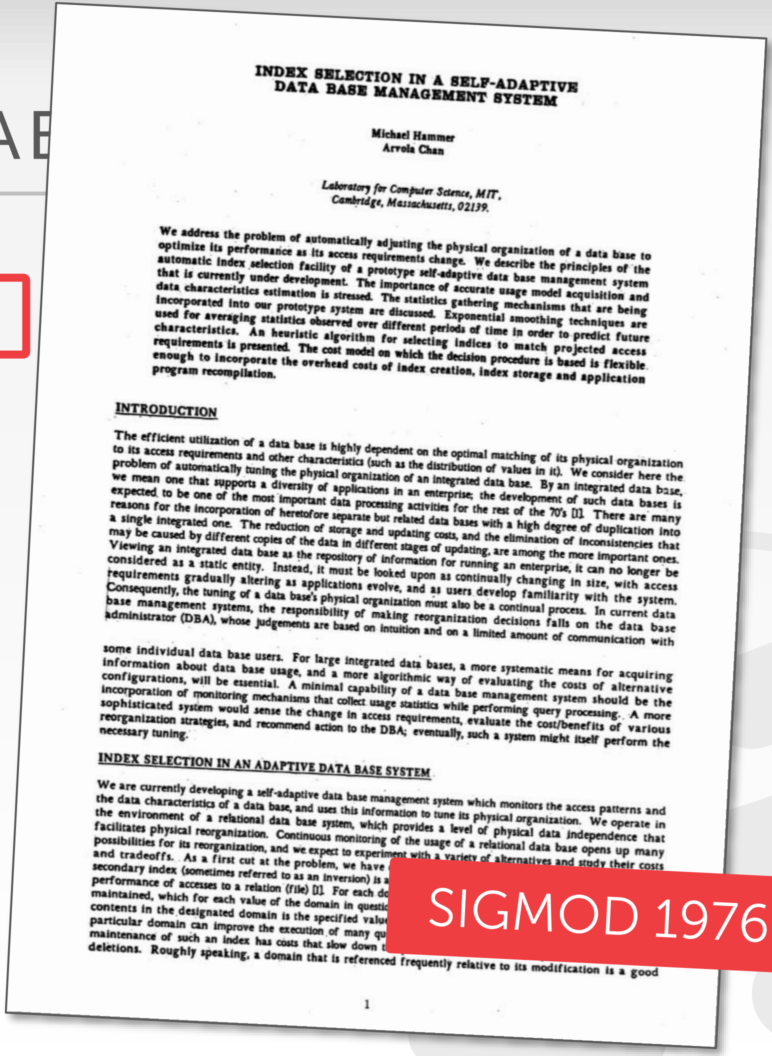
AUTOMATED DATABASE TUNING

The first work on automated database optimization tools started in the 1970s.

These were tools that made recommendations to human operators on what changes to make to a database.

AUTOMATED DATA

- 1970s: Physical Design Tuners
- 1980s: Physical Design w/ Traces
- 1990s: Partitioning + Placement
- 2000s: Microsoft AutoAdmin
- 2010s: ML-enhanced Methods
- 2020s: Self-Driving Systems



INDEX SELECTION IN A SELF-ADAPTIVE
DATA BASE MANAGEMENT SYSTEM

Michael Hammer
Arvola Chan

Laboratory for Computer Science, MIT,
Cambridge, Massachusetts, 02139.

We address the problem of automatically adjusting the physical organization of a data base to
optimize its performance as its access requirements change. We describe the principles of the
automatic index selection facility of a prototype self-adaptive data base management system
that is currently under development. The importance of accurate usage model acquisition and
data characteristics estimation is stressed. The statistics gathering mechanisms that are being
incorporated into our prototype system are discussed. Exponential smoothing techniques are
used for averaging statistics observed over different periods of time in order to predict future
characteristics. An heuristic algorithm for selecting indices to match projected access
requirements is presented. The cost model on which the decision procedure is based is flexible
enough to incorporate the overhead costs of index creation, index storage and application
program recompilation.

INTRODUCTION

The efficient utilization of a data base is highly dependent on the optimal matching of its physical organization
to its access requirements and other characteristics (such as the distribution of values in it). We consider here the
problem of automatically tuning the physical organization of an integrated data base. By an integrated data base,
we mean one that supports a diversity of applications in an enterprise; the development of such data bases is
expected to be one of the most important data processing activities for the rest of the 70's [1]. There are many
reasons for the incorporation of heretofore separate but related data bases with a high degree of duplication into
a single integrated one. The reduction of storage and updating costs, and the elimination of inconsistencies that
may be caused by different copies of the data in different stages of updating, are among the more important ones.
Viewing an integrated data base as the repository of information for running an enterprise, it can no longer be
considered as a static entity. Instead, it must be looked upon as continually changing in size, with access
requirements gradually altering as applications evolve, and as users develop familiarity with the system.
Consequently, the tuning of a data base's physical organization must also be a continual process. In current data
base management systems, the responsibility of making reorganization decisions falls on the data base
administrator (DBA), whose judgements are based on intuition and on a limited amount of communication with

some individual data base users. For large integrated data bases, a more systematic means for acquiring
information about data base usage, and a more algorithmic way of evaluating the costs of alternative
configurations, will be essential. A minimal capability of a data base management system should be the
incorporation of monitoring mechanisms that collect usage statistics while performing query processing. A more
sophisticated system would sense the change in access requirements, evaluate the cost/benefits of various
reorganization strategies, and recommend action to the DBA; eventually, such a system might itself perform the
necessary tuning.

INDEX SELECTION IN AN ADAPTIVE DATA BASE SYSTEM

We are currently developing a self-adaptive data base management system which monitors the access patterns and
the data characteristics of a data base, and uses this information to tune its physical organization. We operate in
the environment of a relational data base system, which provides a level of physical data independence that
facilitates physical reorganization. Continuous monitoring of the usage of a relational data base opens up many
possibilities for its reorganization, and we expect to experiment with a variety of alternatives and study their costs
and tradeoffs. As a first cut at the problem, we have
secondary index (sometimes referred to as an inversion) is a
performance of accesses to a relation (file) [1]. For each do
maintained, which for each value of the domain in questio
contents in the designated domain is the specified value
particular domain can improve the execution of many qu
maintenance of such an index has costs that slow down t
deletions. Roughly speaking, a domain that is referenced frequently relative to its modification is a good

1

SIGMOD 1976

AUTOMATED DATABA

1970s: Physical Design Tuners

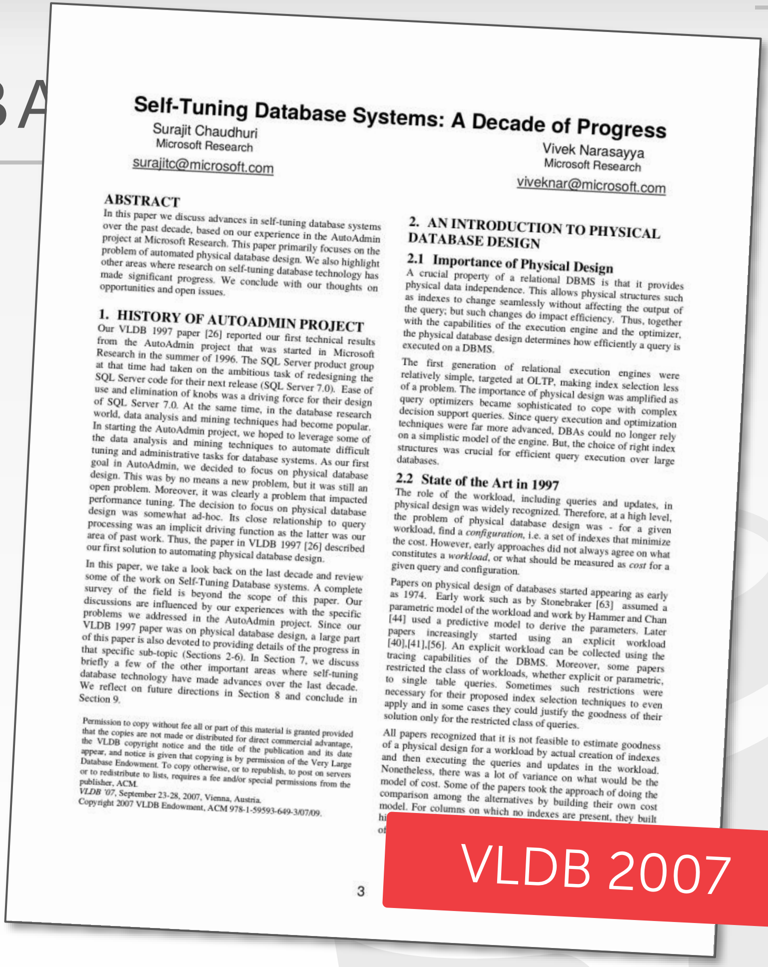
1980s: Physical Design w/ Traces

1990s: Partitioning + Placement

2000s: Microsoft AutoAdmin

2010s: ML-enhanced Methods

2020s: Self-Driving Systems



Self-Tuning Database Systems: A Decade of Progress

Surajit Chaudhuri
Microsoft Research
surajitc@microsoft.com

Vivek Narasayya
Microsoft Research
viveknar@microsoft.com

ABSTRACT

In this paper we discuss advances in self-tuning database systems over the past decade, based on our experience in the AutoAdmin project at Microsoft Research. This paper primarily focuses on the problem of automated physical database design. We also highlight other areas where research on self-tuning database technology has made significant progress. We conclude with our thoughts on opportunities and open issues.

1. HISTORY OF AUTOADMIN PROJECT

Our VLDB 1997 paper [26] reported our first technical results from the AutoAdmin project that was started in Microsoft Research in the summer of 1996. The SQL Server product group at that time had taken on the ambitious task of redesigning the SQL Server code for their next release (SQL Server 7.0). Ease of use and elimination of knobs was a driving force for their design of SQL Server 7.0. At the same time, in the database research world, data analysis and mining techniques had become popular. In starting the AutoAdmin project, we hoped to leverage some of the data analysis and mining techniques to automate difficult tuning and administrative tasks for database systems. As our first goal in AutoAdmin, we decided to focus on physical database design. This was by no means a new problem, but it was still an open problem. Moreover, it was clearly a problem that impacted performance tuning. The decision to focus on physical database design was somewhat ad-hoc. Its close relationship to query processing was an implicit driving function as the latter was our area of past work. Thus, the paper in VLDB 1997 [26] described our first solution to automating physical database design.

In this paper, we take a look back on the last decade and review some of the work on Self-Tuning Database systems. A complete survey of the field is beyond the scope of this paper. Our discussions are influenced by our experiences with the specific problems we addressed in the AutoAdmin project. Since our VLDB 1997 paper was on physical database design, a large part of this paper is also devoted to providing details of the progress in that specific sub-topic (Sections 2-6). In Section 7, we discuss briefly a few of the other important areas where self-tuning database technology have made advances over the last decade. We reflect on future directions in Section 8 and conclude in Section 9.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Database Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permissions from the publisher, ACM.
VLDB '07, September 23-28, 2007, Vienna, Austria.
Copyright 2007 VLDB Endowment, ACM 978-1-59593-649-3/07/09.

2. AN INTRODUCTION TO PHYSICAL DATABASE DESIGN

2.1 Importance of Physical Design

A crucial property of a relational DBMS is that it provides physical data independence. This allows physical structures such as indexes to change seamlessly without affecting the output of the query; but such changes do impact efficiency. Thus, together with the capabilities of the execution engine and the optimizer, the physical database design determines how efficiently a query is executed on a DBMS.

The first generation of relational execution engines were relatively simple, targeted at OLTP, making index selection less of a problem. The importance of physical design was amplified as query optimizers became sophisticated to cope with complex decision support queries. Since query execution and optimization techniques were far more advanced, DBAs could no longer rely on a simplistic model of the engine. But, the choice of right index structures was crucial for efficient query execution over large databases.

2.2 State of the Art in 1997

The role of the workload, including queries and updates, in physical design was widely recognized. Therefore, at a high level, the problem of physical database design was - for a given workload, find a *configuration*, i.e. a set of indexes that minimize the cost. However, early approaches did not always agree on what constitutes a *workload*, or what should be measured as *cost* for a given query and configuration.

Papers on physical design of databases started appearing as early as 1974. Early work such as by Stonebraker [63] assumed a parametric model of the workload and work by Hammer and Chan [44] used a predictive model to derive the parameters. Later papers increasingly started using an explicit workload [40],[41],[56]. An explicit workload can be collected using the tracing capabilities of the DBMS. Moreover, some papers restricted the class of workloads, whether explicit or parametric, to single table queries. Sometimes such restrictions were necessary for their proposed index selection techniques to even apply and in some cases they could justify the goodness of their solution only for the restricted class of queries.

All papers recognized that it is not feasible to estimate goodness of a physical design for a workload by actual creation of indexes and then executing the queries and updates in the workload. Nonetheless, there was a lot of variance on what would be the model of cost. Some of the papers took the approach of doing the comparison among the alternatives by building their own cost model. For columns on which no indexes are present, they built

3

VLDB 2007

TARGET OBJECTIVE

An automated system must be instructed by a human on what objective it should optimize.
→ Throughput
→ Latency
→ Cost
→ Hardware Utilization

A system could use multiple objectives where some have different priorities or provided as a constraint.
→ Example: Minimize cost but maintain latency threshold.

ACTION SELECTION

A change to the database is called an ***action***.

→ Build index, change knob, etc.

An automated tool decides **what** actions to apply to the database to improve some target objective.

The number of possible actions available to optimize a DBMS is clearly NP-Complete.

→ Just picking indexes for a database is NP-Complete.

COST MODELS

It is too expensive to try out all possible design decisions in a database to find options.

→ Copying data is expensive / time-consuming.
→ Some changes also do not take effect until restarting.
→ Actions are not independent.

We will see different methods to estimate the cost/benefit of actions through a combination of external and internal cost models.

IMPORTANT TUNING RULES

Do **<u>not</u>** require changes to the application.

Do **<u>not</u>** lose data.

Do **<u>not</u>** degrade observable correctness.

Do **<u>not</u>** incur unscheduled unavailability.

Do **<u>not</u>** cause observable unexpected behavior.

In general, we must avoid any decision that requires a human to make a **value judgement**.

PARTING THOUGHTS

The previous 50 years of research focused on the problem of **what** to tune in a DBMS.

This course will also cover the **when** and **how** problems that are necessary to achieve parity with human experts.

NEXT CLASS

Self-Driving Databases Overview

Make sure that you submit the first reading review

https://cmudb.io/15799-s22-submit