

Lessons Learned from Automatically Optimizing **Databases** Using **Machine Learning** in the Real World

Bohan Zhang
Cofounder @ OtterTune



01. Background

02. The Real World

03. Lessons Learned

04. LLMs For Databases





01. Background

01. Background

Databases are **notoriously** complex to deploy, optimize, and maintain.

Physical Design (Indexes, Partitioning)

Knob Configuration

Query Optimization

Hardware Provisioning

Human experts are **scant, unscalable,** and **expensive.**



U.S. BUREAU OF LABOR STATISTICS

Occupational Employment and Wages, May 2021

15-1242 Database Administrators

Administer, test, and implement computer databases, applying knowledge of database management systems. Coordinate changes to computer databases. Identify, investigate, and resolve database performance issues, database capacity, and database scalability. May plan, coordinate, and implement security measures to safeguard computer databases. Excludes "Information Security Analysts" (15-1212) and "Database Architects" (15-1243).

National estimates for Database Administrators:

Employment estimate and mean wage estimates for Database Administrators:

Employment (1)	Employment RSE (3)	Mean hourly wage	Mean annual wage (2)	Wage RSE (3)
85,870	1.5 %	\$ 46.42	\$ 96,550	0.9 %

Percentile wage estimates for Database Administrators:

Percentile	10%	25%	50% (Median)	75%	90%
Hourly Wage	\$ 23.50	\$ 30.36	\$ 46.50	\$ 59.88	\$ 72.79
Annual Wage (2)	\$ 48,880	\$ 63,160	\$ 96,710	\$ 124,550	\$ 151,400

01. Automated Database Optimization

There is a long history of attempts in research to automate database management systems.

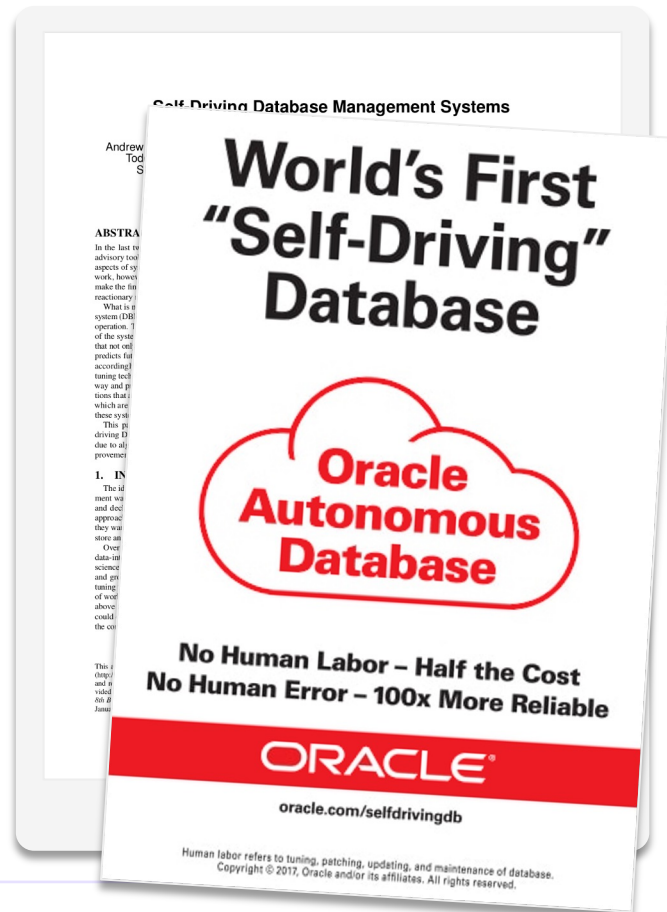
1970s: Self-Adaptive Databases

1990s/2000s: Self-Tuning Databases

2010/2020s: Self-Driving Databases



Research in the last decade has focused on applying **machine learning** (ML) methods to solve the tuning problem for databases.



01. Machine Learning for Databases

Indexes:

[Azure Auto Indexing](#), [Oracle Autonomous Database Service](#), [Cornell UDO](#), [OpenGauss](#)

Partitioning:

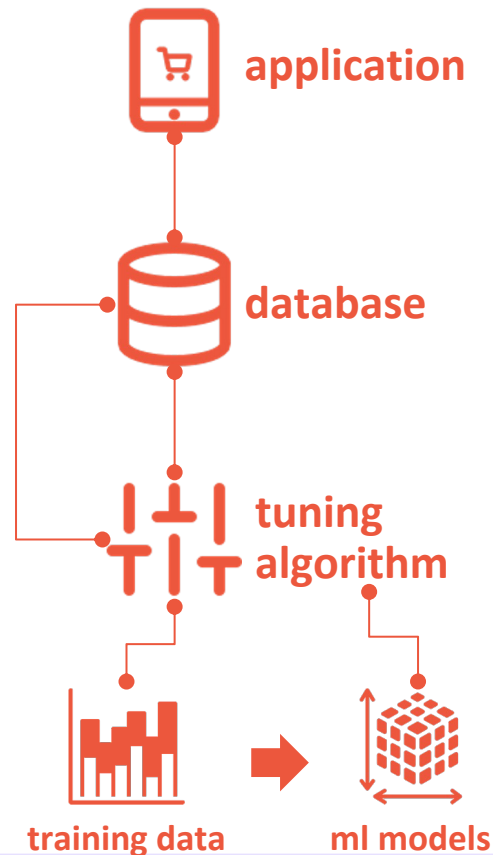
[Cloud Partition Adviser](#)

Knob Configuration:

[OtterTune](#), [CDBTune](#), [Akamas](#), [ResTune](#), [QTune](#)

Query Optimization:

[Bao](#) (Join Algos), [Neo](#) (Join Ordering), [MySQL Heatwave Autopilot](#) (Plan Stitching)



01. OtterTune Overview

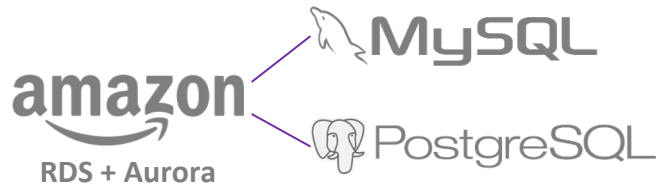
OtterTune is an automated database tuning and resource optimization service.

Based on research developed at [Carnegie Mellon University](#) Database Group.

It uses **machine learning** to automatically optimize the configurations of DBMSs to improve performance, reduce costs, and maintain healthy operations.

Research: Knobs

Commercial: Knobs, Indexes, Queries, Cloud Config





02. The Real World

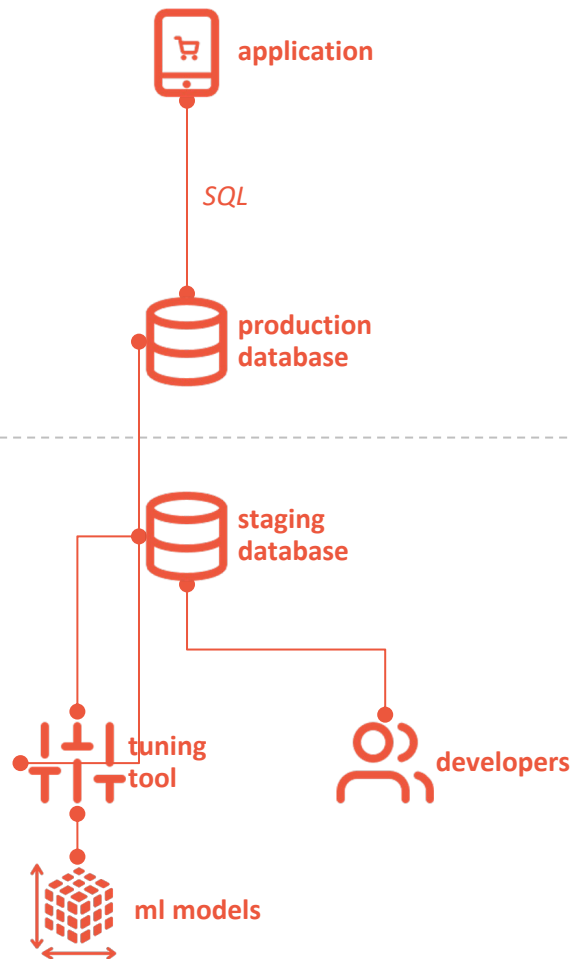
02. Real-World Databases

Challenge #1: Users do not maintain suitable staging databases.

Training models on staging DBs is **bad** because of inconsistent workloads and hardware.

Dev/Staging databases run on smaller hardware with a subset of the production databases because of cost.

Dynamic hardware scaling (burst credits) and serverless instances make this worse.



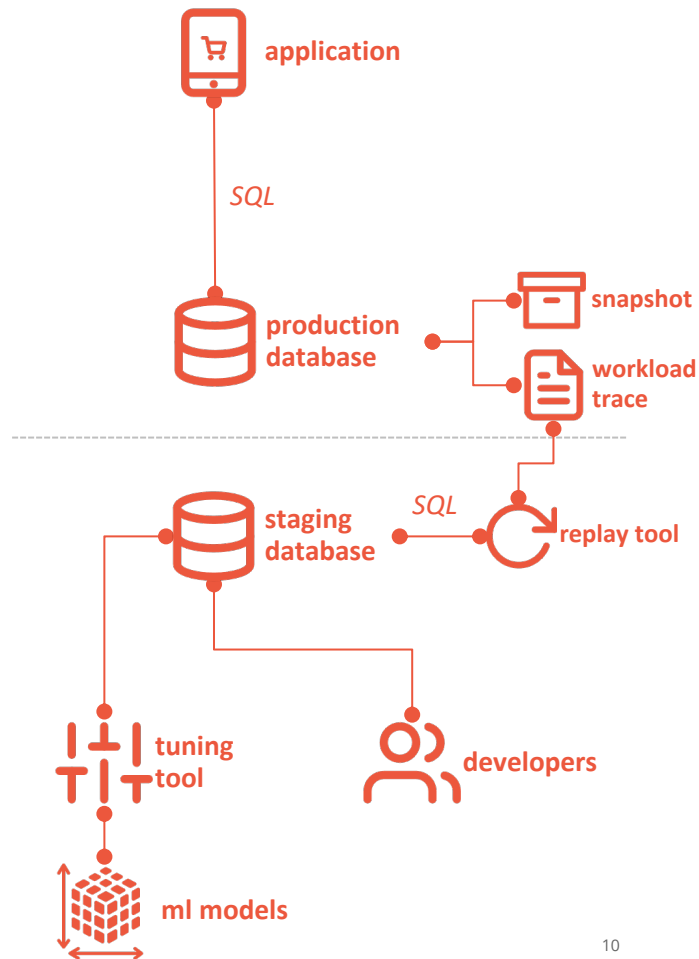
02. Real-World Databases

Challenge #2: Users cannot capture workloads and replay them.

Without a **repeatable workload** as a baseline, it is difficult for the ML models to learn whether they are improving a database.

Tools for open-source DBMSs are less sophisticated than commercial DBMSs.

Existing built-in slow query log methods do not capture transaction boundaries. It's super expensive to log all queries to disk.



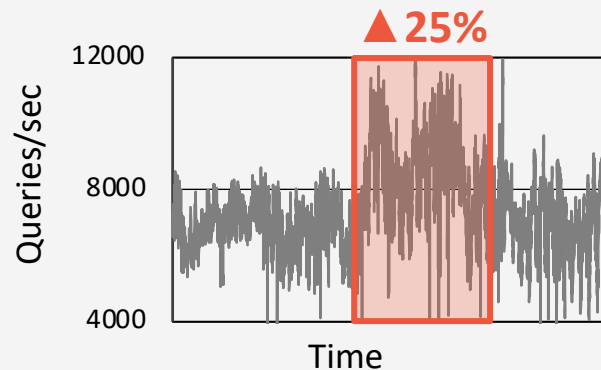
02. Real-World Databases

Challenge #3: Users mostly do not know what their database is doing. The production workload is dynamic.

Workload patterns and application **changes** make it difficult to measure whether a tuning tool is making things better or worse.

Most production workloads have daily or weekly patterns.

SELECT Throughput



 MySQL





03. Lessons Learned

03. Production Database Tuning

Tuning a staging database using a replayed workload is impractical in real-world scenarios.

Users do not maintain suitable staging databases.

Users cannot capture workloads and replay them.

Most customers allow us to **carefully** tune their production databases. But they need to have control over **what**, **when**, and **why** changes are applied.



03. Manual Controls & Explanations

To reduce untimely performance degradations or downtime, a tuning service must provide controls to allow humans to specify **what** and **when** the service will optimize the database.

Tuning Periods

Restart Tracking & Scheduling

Human-in-the-loop Approval

To help build trust, provide users with data-driven explanations about recommendations. (**why** they should apply them)

Approval Method
Choose how knob recommendations are applied to your database, either automatically or after your approval.

Manual: Recommendations applied after your approval.
Manual: Otter Tune will only make changes to your database when you approve a new recommendation. Otter Tune will send you an email when a configuration is ready for review.

Tuning Schedule
The window of time Otter Tune can apply tuning recommendations to your database instance.

Mondays	00:00 GMT	to	23:59 GMT
Tuesdays	00:00 GMT	to	23:59 GMT
Wednesdays	00:00 GMT	to	23:59 GMT
Thursdays	00:00 GMT	to	23:59 GMT
Fridays	00:00 GMT	to	23:59 GMT

Knobs | Indexes | Queries | Tables

Recommendation Type

Increase table_open_cache_size from 6000 to 12000
Your database opens 596.3 tables per minute on average. This high value can indicate inefficient table cache management, which may be improved by increasing your database's table_open_cache setting.



03. Machine Learning & Domain Knowledge

Machine Learning models can help find the (nearly) optimal database configurations.

- No explanations about recommendations

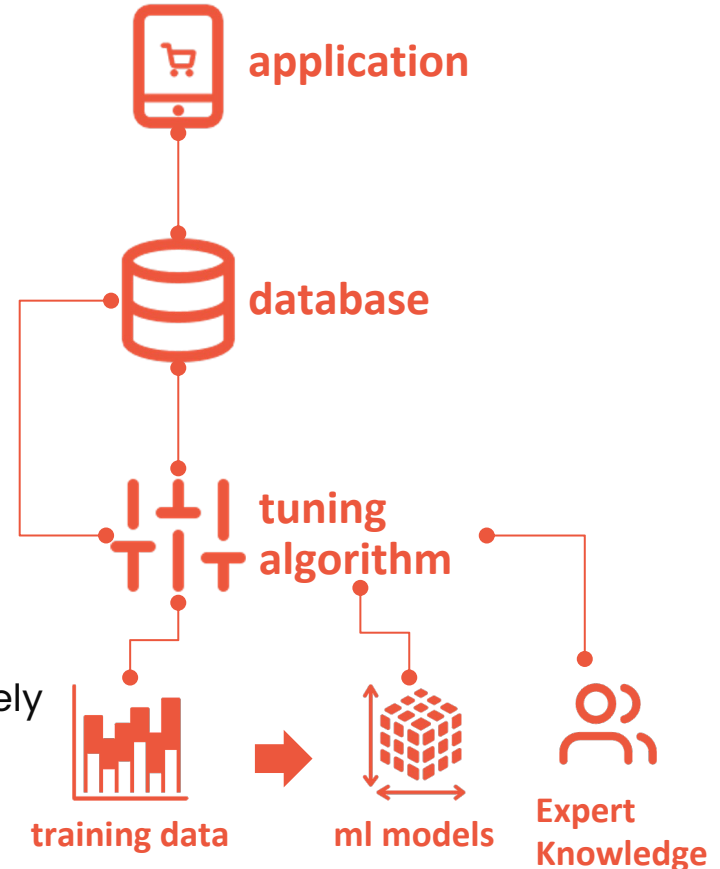
- Need time to converge

Heuristic-based approach using domain knowledge may not yield the optimal recommendations.

- Provide explanations about recommendations

- Do not need training data, recommend immediately

You don't need to pick just one. Integrate **both** methods for better recommendations.



03. Not A Machine Learning Problem

Automated database tuning with ML works **better** in the real-world than in the research lab, but getting the full benefit of these optimizations and tuning production databases **safely** is **not a ML problem**.

Knob Bounds

Tuning Periods & Schedule

Explanations about recommendations

...

We extend OtterTune to support three ML models. They all have **similar** performance:

Gaussian Process Regression

Deep Neural Network

Deep Deterministic Policy Gradient



An Inquiry into Machine Learning-based Automatic Configuration Tuning Services on Real-World Database Management Systems

VLDB 2021





04. LLMs For Databases

04. Database Questions

LLMs like ChatGPT may **not** be the ideal resource for addressing your database-related questions because it can give you **inaccurate** answers.

From ChatGPT, **jit** parameter can have the following values:

off ✓

on ✓

normal ✗

all ✗

B jit related knobs in postgres



In PostgreSQL, JIT (Just-In-Time) compilation is a feature that allows the database to dynamically compile SQL queries into native machine code for improved query execution performance. As of my last update in September 2021, PostgreSQL introduced JIT compilation in version 11, and there might have been further developments since then. Here are some relevant parameters or "knobs" related to JIT compilation in PostgreSQL:

1. **jit**: This is a configuration parameter that can be set to control the overall behavior of JIT compilation. It can have the following values:
 - * `off`: Disables JIT compilation.
 - * `on`: Enables JIT compilation for all statements.
 - * `normal`: Enables JIT compilation for statements considered worthwhile by the query planner (default).
 - * `all`: Enables JIT compilation for all statements.



04. Database Optimizations

LLMs like ChatGPT are trained on existing text that cannot account for an **application's** workload patterns.

It can only give you some **generic** recommendations and cannot give you **exact** answers.

We have found that "best practice" recommendations are often **wrong**.



The `innodb_adaptive_hash_index` option in MySQL enables InnoDB to build a hash index on frequently accessed data pages in the buffer pool, which can speed up point queries (i.e., queries that retrieve a single row by its ID).

However, whether or not to enable this option depends on your workload and the nature of your queries. Given your high percentage of `SELECT` operations (77.60%), you might benefit from it, especially if those `SELECT` queries are mostly point queries.

That said, it's important to note that this setting can also consume additional memory and CPU resources for maintaining the hash index. In certain cases, especially in highly concurrent workloads, it can lead to contention issues and actually degrade performance.

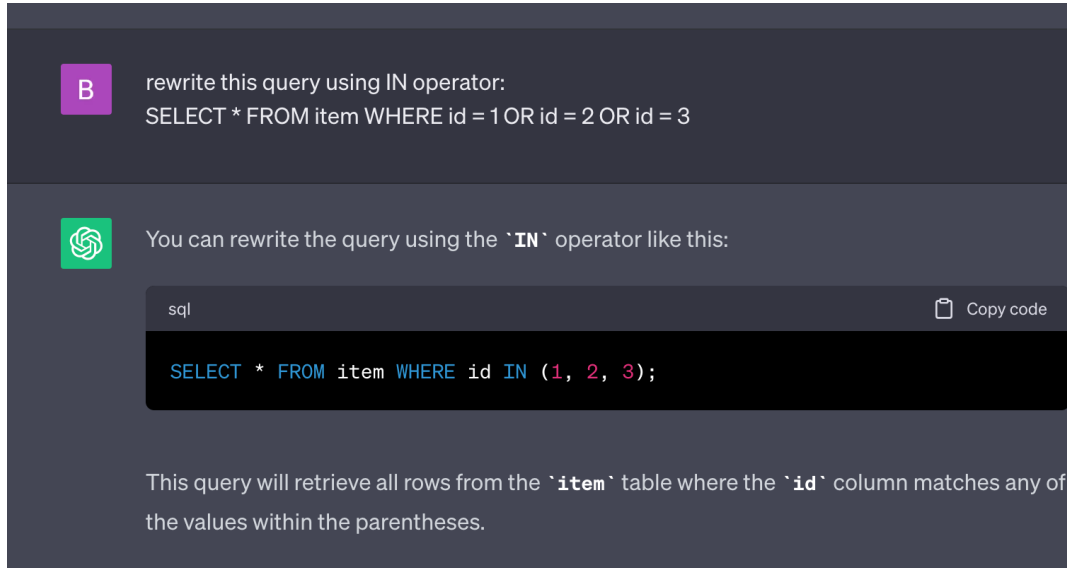
Before you decide to enable it, you should test it in a non-production environment with a workload similar to your production environment. Monitor the performance and system resource usage closely to see if it provides a net benefit.

As of MySQL 8.0.23, note that this feature is deprecated, as improvements in the MySQL server and InnoDB make it less beneficial. Therefore, if you're using a version of MySQL where it's deprecated, it's recommended to keep it turned off.



04. Database Queries

LLMs like ChatGPT are actually very **good** at writing SQL queries. It can help rewrite queries with **hints**.



The screenshot shows a chat window with a dark background. At the top, a user icon (a purple square with a white 'B') asks to rewrite a query using the IN operator. The original query is: `SELECT * FROM item WHERE id = 1 OR id = 2 OR id = 3`. Below, the AI icon (a green square with a white logo) responds that the query can be rewritten using the `IN` operator. A code block shows the rewritten query: `SELECT * FROM item WHERE id IN (1, 2, 3);`. At the bottom, a note explains that this query retrieves all rows from the `item` table where the `id` column matches any of the values in the parentheses.

Rewrite this query using IN operator:

```
select * from item where id =  
1 OR id = 2 OR id =3
```

ChatGPT:

```
select * from item where id IN  
(1, 2, 3)
```



END

<https://ottertune.com>

bohan@ottertune.com
@BohanZhangOT

