AI-Driven Database Tuning in DevOps: Real-World Lessons from Self-Optimizing Systems

Drawing from production deployments in Finance, E-Commerce, and Healthcare

Presenter:- Nagamalleswararao Bellamkonda
Pune University, India
Conf42 Database DevOps 2026

Why Manual Tuning Can't Keep Up

The Speed Gap

Performance degradation from a bad deployment begins within **minutes**. Manual database tuning operates on timescales of **hours or days**.

The Core Problem

Modern DevOps environments deploy continuously and scale dynamically. Traditional tuning was built for a world that no longer exists. The gap between the speed of change and the speed of remediation is precisely where self-optimizing systems deliver their greatest value.

Beyond Monitoring: Continuous Adaptation

Self-tuning database systems do more than alert they act. The shift is from reactive troubleshooting to proactive, continuous optimization.

Workload Monitoring

Continuous analysis of query patterns as they evolve in production

Index Management

Automatic index creation and removal based on observed access patterns

Query Optimization

Execution plan selection refined continuously against real workloads

Resource Allocation

Memory and compute adjusted dynamically to match demand

Security Patching

Automated patching with minimal downtime and full audit trails

Where We Deployed

Three sectors. Three distinct challenges. Each required a different entry point into AI-driven tuning.

Finance

Query plan regressions causing SLA breaches after releases. The need: **plan stability at deployment speed.**

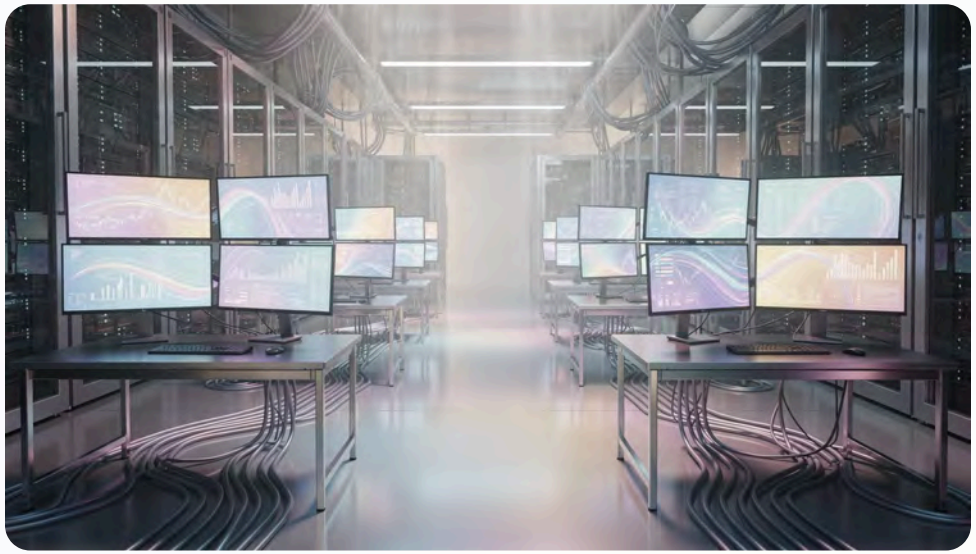
E-Commerce

Workload spikes around promotional events. The need: **proactive scaling, not reactive scrambling.**

Healthcare

Strict regulatory compliance requirements. The need: **auditability before automation.**

Financial Services: Stopping Plan Regressions at the Gate



The Problem

Execution plan regressions following application releases caused intermittent failures in transaction processing detected only after production users were impacted.

What We Built

AI-driven query plan monitoring embedded directly into the CI/CD pipeline. Regressions detected within seconds of deployment and automatically rolled back to stable configurations.

E-Commerce: Predicting the Surge

The Problem

Static over-provisioning was expensive. Reactive scaling was too slow for promotional traffic spikes by the time the system responded, revenue was already lost.

What We Built

Machine learning models trained on historical promotional patterns, connected to the marketing events pipeline for advance signals. Resources were allocated **before** traffic arrived not in response to it.

Healthcare: Building Trust Before Automating

The Problem

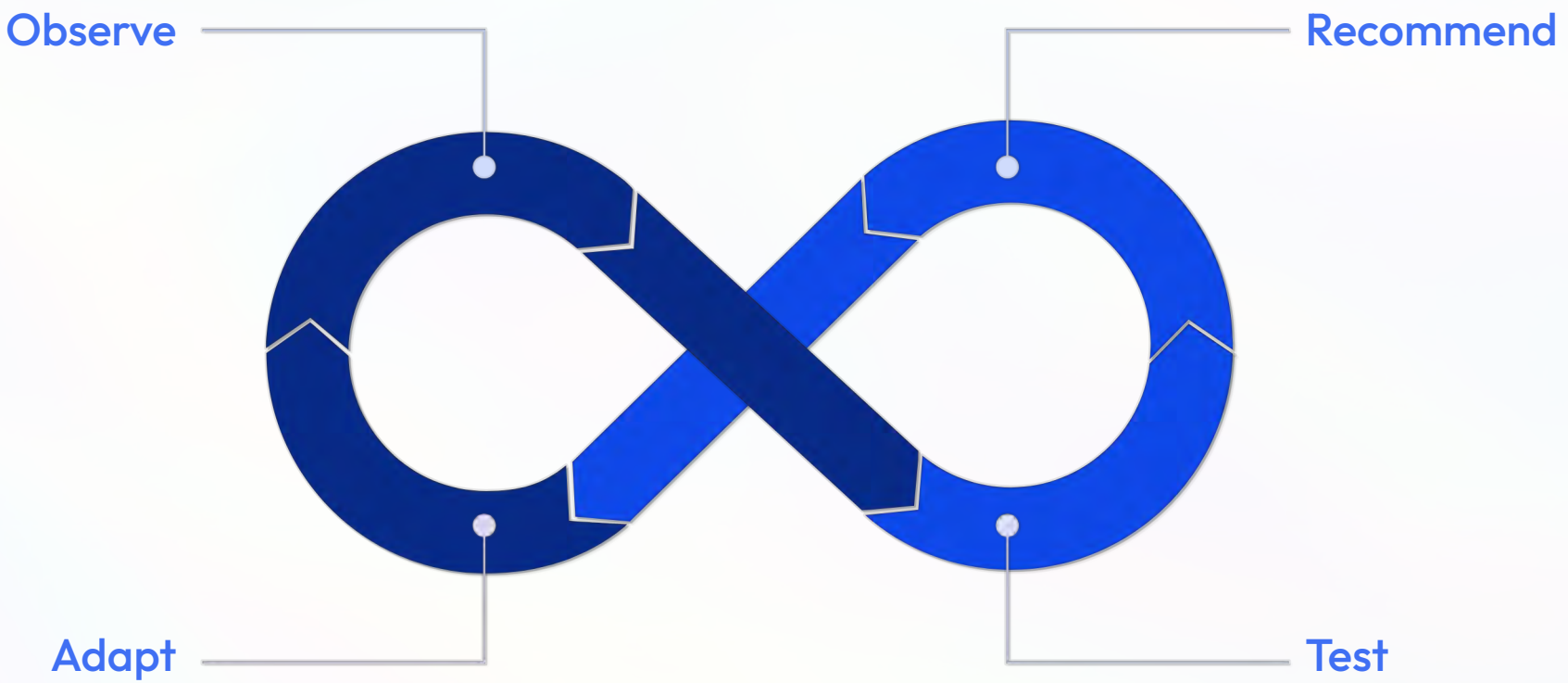
Any automated configuration change required a full, auditable approval trail. Early automation would have created compliance risk even when technically sound.

What We Built

A staged approach: AI recommendations surfaced through a review dashboard first. Only after demonstrating reliability were low-risk, reversible changes automated with complete audit logging throughout.

How Machine Learning Improves Queries

AI-driven optimization moves beyond static rules by continuously learning from real workloads. The system improves because it observes the consequences of its own decisions.



This closed-loop architecture is what distinguishes genuine self-optimization from sophisticated alerting.

Three Failures That Reshaped Our Approach

Model Staleness

A major application refactor shifted query patterns faster than the model could adapt. Degraded recommendations went undetected for **nearly two weeks**.

Transfer Learning Limits

A model trained in one environment did not port cleanly to a similar client. Residual gaps required **weeks of fine-tuning** to close.

Integration Complexity

Connecting AI engines to existing CI/CD pipelines and change management systems consistently took **longer than building the models themselves**.

Technical sophistication without operational integration delivers no value.

Designing Indexing Automation You Can Trust

Automated indexing is high-value and high-risk. Architecture is what makes it safe.

Shadow Evaluation First

New recommendations tested against replayed production traffic before any promotion to live systems.

Full Change Provenance

Every change tagged with the model version and workload context that generated it.

Rollback Is Non-Negotiable

One-click rollback built into the pipeline from day one not retrofitted after the first incident.

Post-Change Monitoring

Write performance and lock contention observed immediately after every automated index change.

You Cannot Automate What You Cannot See

Every automated change was logged with full context and surfaced in the same dashboards teams already used for deployment monitoring. **Teams trusted the system because they could always see what it had done.**

Recommendation Rationale

Why the model generated each specific recommendation

Model Version

Which model version produced the change, enabling regression analysis

Shadow Validation

Results from pre-production testing against replayed traffic

Deployment Timestamp

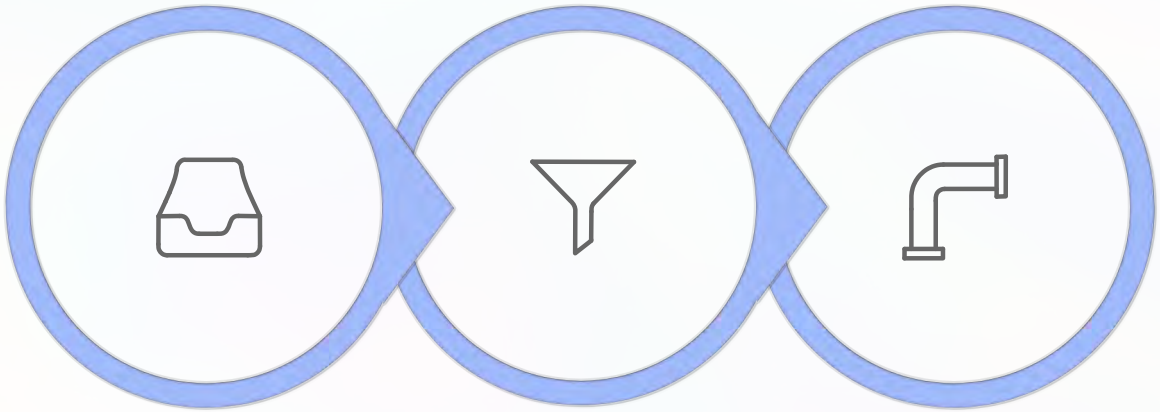
Precise timing for correlation with upstream deployment events

Observed Impact

Post-deployment performance delta fed back into model training

Real-Time Query Log Analysis: Three Components

The open-source toolkit that powered all three deployments is built around a closed feedback loop.



The closed loop between deployment outcomes and model training is what allows the system to improve continuously rather than degrading as workloads evolve.

Four Things We'd Change

1 Invest Earlier in Telemetry Infrastructure

Data quality limits recommendation quality. Build reliable pipelines **before** building models.

2 Stay in Recommendation Mode Longer

Automating before organizational trust is established creates incidents that set back adoption by months.

3 Define Retraining Triggers from Day One

Formal accuracy thresholds and retraining schedules belong in the initial design not as a reaction to the first model staleness incident.

4 Design Every Change as Reversible

If you cannot roll it back cleanly, you should not automate it yet. Reversibility is a precondition not a stretch goal.

Four Stages for Getting Started

1

Stage 1

Instrument & Baseline

Deploy telemetry. Build workload models. Take no automated action. Build insight and trust.

2

Stage 2

Recommendation-Driven

Surface AI suggestions in dashboards. Humans approve all changes. Begin the feedback loop.

3

Stage 3

Automate Low-Risk Actions

Define a bounded category of reversible changes. Full audit logging and rollback required.

4

Stage 4

Progressive Expansion

Expand automation based on demonstrated reliability. Human approval for schema changes and irreversible actions.

What Success Actually Looks Like

The organizations that succeed with AI-driven database tuning are not those that deploy the most sophisticated models. They are those that build the **operational discipline** to deploy, monitor, and iterate on AI systems within the constraints of real production environments.

The goal is not full autonomy for its own sake it is the **right level of automation for each action type**: maximizing efficiency while preserving human control over high-risk decisions.

Engage Now

The field is moving quickly. Early movers build the institutional knowledge that compounds over time.

Learn from Failure

Teams that document and share failures not just successes advance faster and more safely.

Own the Future

The teams engaging today will operate the data systems that define the next generation of enterprise infrastructure.