Database Evolution: Partitioning and Indexing for Cloud-Native Systems

How modern cloud adoption is reshaping database design from rigid on-premises models to intelligent, adaptive architectures built for scale, resilience, and developer productivity.

PURUSHOTHAM JINKA

CONF42 CLOUD NATIVE 2026

VICE PRESIDENT AT CITI BANK

The Challenge

The Problem with Legacy Database Design

Traditional databases were architected for predictable, on-premises workloa fixed hardware, stable query patterns, and manual tuning cycles measured in days or weeks. This approach fundamentally breaks down in cloud environments where workloads are dynamic, infrastructure is elastic, and availability is non-negotiable.

As enterprises accelerate cloud adoption, database teams face a widening gap between legacy operational models and the demands of modern distributed systems. The cost of that gap is measured in query latency, DBA overhead, and missed SLAs.

Static Partitioning

Predefined schemas that can't adapt to shifting workload patterns or data distribution changes.

Manual Index Tuning

DBA-driven index selection creates bottlenecks and delays as query volumes scale.

Operational Fragility

On-premises models lack elastic scaling, making workload surges a risk to availability.

Session Roadmap

What We'll Cover Today

This session builds a structured narrative from the architectural foundations of cloud-native databases to the emerging role of machine learning in self-optimizing systems.

- **The Problem Space**

  Why legacy partitioning and indexing strategies fail under cloud-native workloads

- **Modern Partitioning**

  Adaptive, workload-aware partitioning strategies that evolve with query patterns

- **Indexing Transformation**

  From static B-trees to multi-dimensional, adaptive index structures

- **ML-Driven Optimization**

  Learned indexes, adaptive query processors, and intelligent workload managers

- **Takeaways & Next Steps**

  Practical insights for platform architects and senior DBAs to act on immediately

Chapter 1 Partitioning

Partitioning Has Left the Building

Cloud-native systems demand partitioning that adapts continuously as elastic compute changes the cluster, multi-tenant workloads create hotspots, and teams expect zero-downtime rebalancing. Static shard boundaries quickly become uneven as traffic shifts, so what was balanced at deployment time can become a bottleneck minutes later. In practice, partitioning has to respond to change instead of assuming the data and workload stay still.

Adaptive Partitioning

Rebalances data placement as query patterns and traffic evolve.

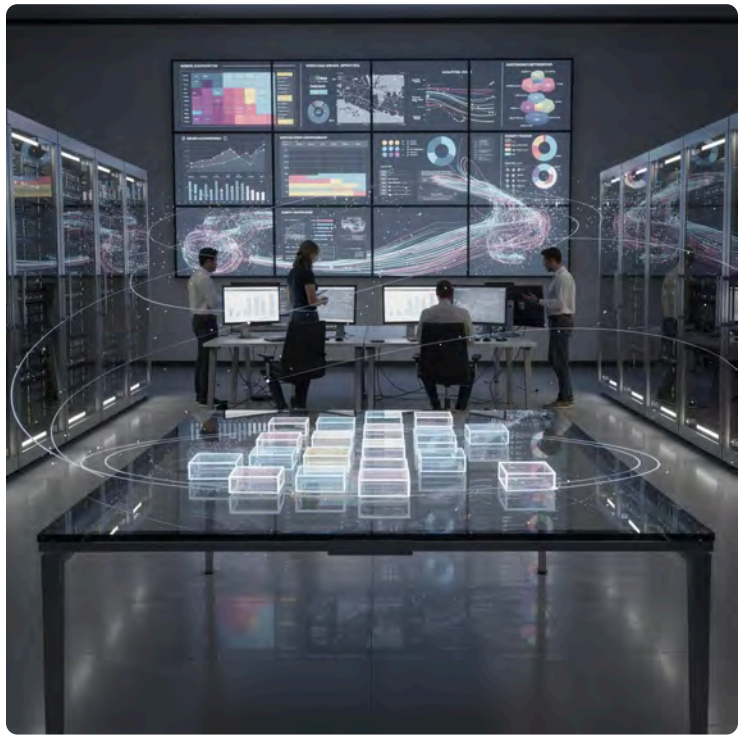
Elastic Scaling

Handles rapid growth and shrinkage without manual rework.

High Availability

Preserves service continuity during failures and live reshuffles.

From Static Schemas to Adaptive Partitioning



Legacy partitioning depended on upfront range, list, or hash decisions that worked for predictable workloads but struggled as cloud-native access patterns shifted. Adaptive partitioning instead monitors live query behavior and data distribution to rebalance partitions automatically, without downtime or manual DBA intervention.

Workload-Aware

Zero-Downtime Rebalancing

Reduced Admin Overhead

Key Partitioning Strategies in Cloud-Native Systems

Modern cloud-native databases blend multiple partitioning approaches to balance query performance, write throughput, and storage distribution across distributed nodes.

Range Partitioning

Optimized for time-series and ordered datasets. Enables efficient range scans and partition pruning, significantly reducing I/O for temporal queries.

Hash Partitioning

Distributes data uniformly across nodes using key hashing. Prevents hot spots in high-write OLTP workloads and ensures balanced storage utilization.

Composite Partitioning

Combines range and hash strategies for complex multi-tenant and analytics workloads, balancing both scan efficiency and write distribution.

Elastic Scaling and High Availability Through Partitioning

The Cloud Advantage

Cloud-native partitioning is not just about query performance it is the architectural foundation for elastic scaling and continuous availability. When workload surges arrive, the system must absorb them without manual intervention.

Partition-aware routing, combined with distributed replication, enables cloud databases to scale horizontally within minutes adding capacity without repartitioning the entire dataset or taking the system offline.

Partition-Level Replication

Individual partitions are replicated independently, enabling targeted failover and reducing recovery scope during node failures.

Dynamic Shard Splitting

Hot partitions are automatically split and redistributed across available nodes as traffic concentrates, preventing performance degradation.

Geo-Aware Placement

Partition placement policies can pin data to specific regions or availability zones, satisfying data residency and latency requirements simultaneously.

Chapter 2 Indexing

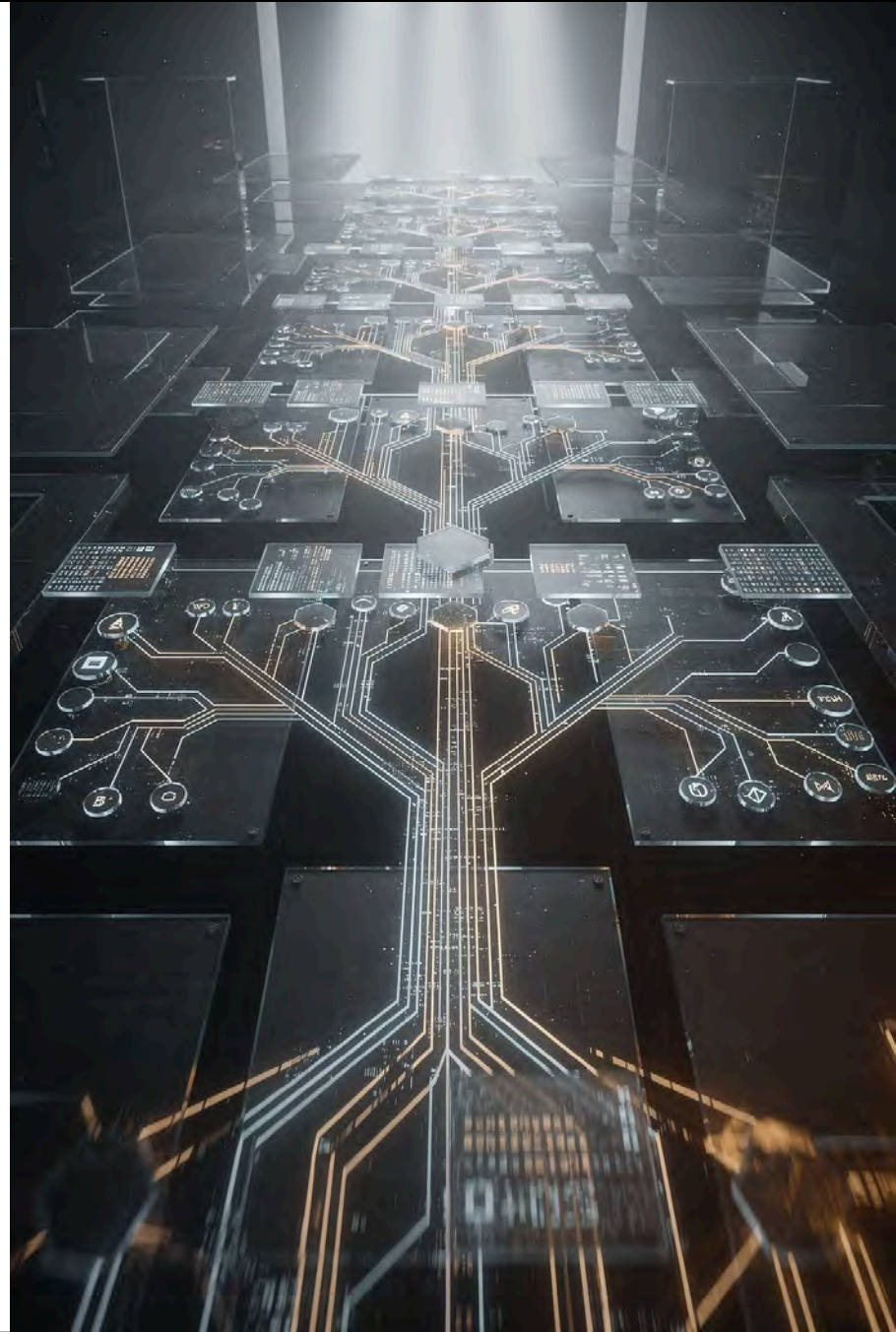
The Index Is No Longer Just a B-Tree

Traditional B-tree indexes are being redesigned to handle modern query patterns, vector search, and the rising operational costs of maintaining performance at cloud scale.

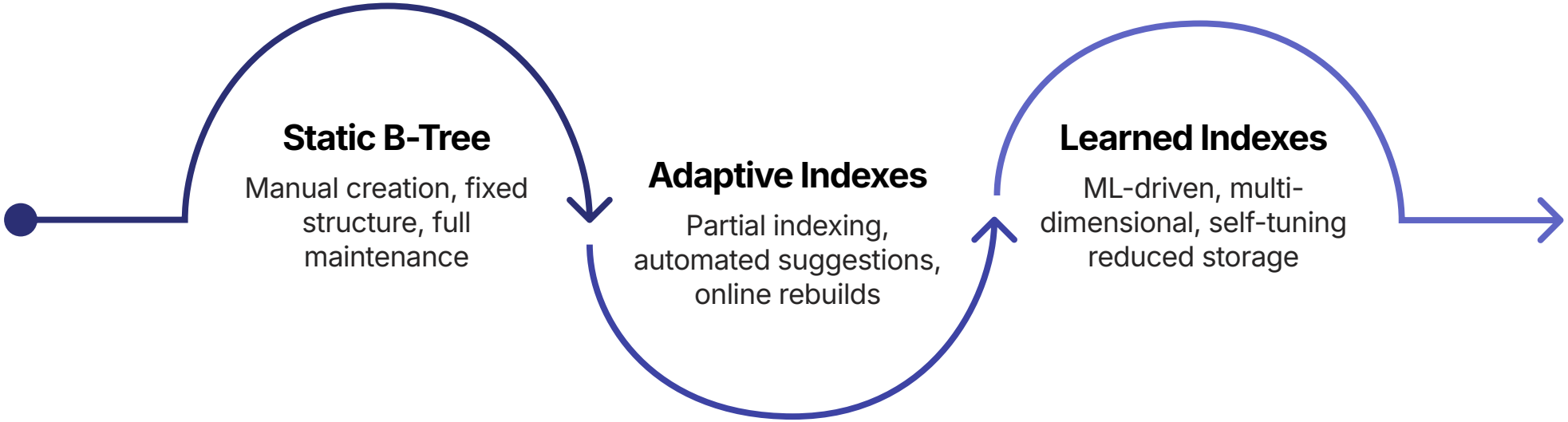
Adaptive Indexing

Learned Index Structures

Index Trade-offs



The Transformation of Indexing Architectures



The evolution from static B-tree structures to adaptive, multi-dimensional index models represents one of the most significant architectural shifts in modern database engineering. Each generation reduces operational overhead while improving query selectivity and storage efficiency at scale.

Adaptive and Multi-Dimensional Indexing

Static B-tree indexes force DBAs to predict query patterns upfront and manually maintain structures as schemas evolve, which quickly breaks down in high-throughput cloud environments. Adaptive indexing instead learns from actual query execution at runtime, while multi-dimensional models handle spatial, temporal, and vectorized patterns that B-trees cannot efficiently support.

1	2
Partial Indexes	**Covering Indexes**
3	4
Multi-Column Composite	**Inverted & Spatial Indexes**

Indexing Trade-offs: The Engineering Reality

Every index is a write amplification tax on the system. Cloud-native database engineers must treat index design as an ongoing optimization problem, not a one-time schema decision.

- **Read Performance Gain**

  Well-designed indexes reduce query execution time by orders of magnitude through predicate pushdown and partition pruning the primary driver for index creation.

- **Write Amplification**

  Every INSERT, UPDATE, and DELETE must maintain all associated indexes. Excessive indexing degrades write throughput a critical concern in high-volume OLTP workloads.

- **Storage Overhead**

  Each index consumes additional disk space proportional to the indexed dataset. Learned and partial indexes address this by representing data more compactly than traditional tree structures.

- **Automated Maintenance**

  Cloud-native engines use adaptive index advisors to identify unused indexes, suggest new ones, and trigger background rebuilds without disrupting active transactions.

Chapter 3 ML in Databases

Machine Learning Enters the Query Engine

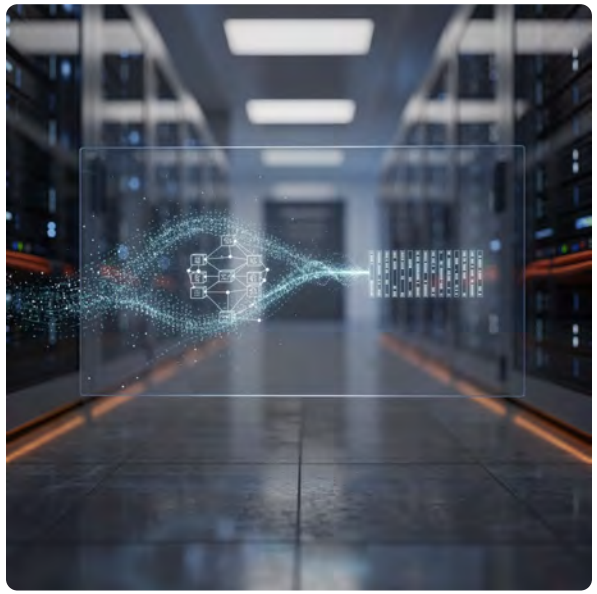
Machine learning is being embedded directly into the query pipeline, shifting databases from rule-based tuning to data-driven, self-optimizing systems. Learned cost models, predictive optimization, and workload forecasting help engines adapt plans, resources, caches, and indexing strategies as performance feedback accumulates.

Learned Indexes

Adaptive Query Processing

Workload Management

Learned Indexes: Replacing Trees with Models



What Are Learned Indexes?

A learned index replaces a traditional B-tree or hash index with a machine learning model typically a small neural network or regression model trained to predict the position of a key within a sorted dataset. The model learns the data distribution and approximates the lookup function directly.

The core advantage is **compactness**. A learned model can represent the same lookup function as a multi-level B-tree using a fraction of the memory, which is critical at the scale cloud databases operate at. Benchmarks from frameworks such as **OLTP-Bench** and **iTuned** have been used to validate these performance gains in realistic workload conditions.

Adaptive Query Processing with ML

Traditional query optimizers rely on static cost models and table statistics that can become stale as data distributions shift. ML-driven adaptive query processors close this gap by learning from execution history and continuously refining execution plan selection.

- **Workload Observation**

  The system continuously captures query execution metrics latency, row estimates, I/O, and cardinality errors as a training signal.

- **Model Training**

  A lightweight model learns to predict optimal join orders, access methods, and parallelism settings for recurring query patterns.

- **Plan Correction**

  At runtime, the adaptive processor overrides suboptimal plan choices identified by the model, reducing execution time for known query shapes.

- **Continuous Feedback**

  Each execution produces new feedback that refines the model, creating a self-improving loop that improves over time as workload patterns stabilize.

ML-Driven Workload Management

The Prediction Challenge

In multi-tenant cloud databases, workload composition changes continuously new query types emerge, tenant activity spikes unpredictably, and maintenance jobs compete with production traffic. Static resource allocation strategies cannot respond fast enough to prevent SLA violations.

ML-driven workload managers address this by predicting workload composition ahead of execution, pre-allocating resources, and routing queries to the most appropriate compute tier before contention develops.

- **Workload Classification**

  ML models classify incoming queries by resource profile CPU-intensive, memory-bound, or I/O-heavy enabling smarter scheduling and queue prioritization.

- **Proactive Resource Allocation**

  Predicted workload profiles trigger preemptive scaling actions, ensuring compute capacity is available before contention causes latency spikes.

- **Anomaly Detection**

  Deviations from learned workload baselines trigger automated alerts or self-healing responses, reducing mean-time-to-resolution in large deployments.

Benchmarking Cloud-Native Performance: OLTP-Bench and iTuned

Validating the real-world impact of adaptive partitioning, learned indexes, and ML-driven optimization requires rigorous, reproducible benchmarking under realistic conditions. Two frameworks are particularly relevant for cloud-native database evaluation.

OLTP-Bench

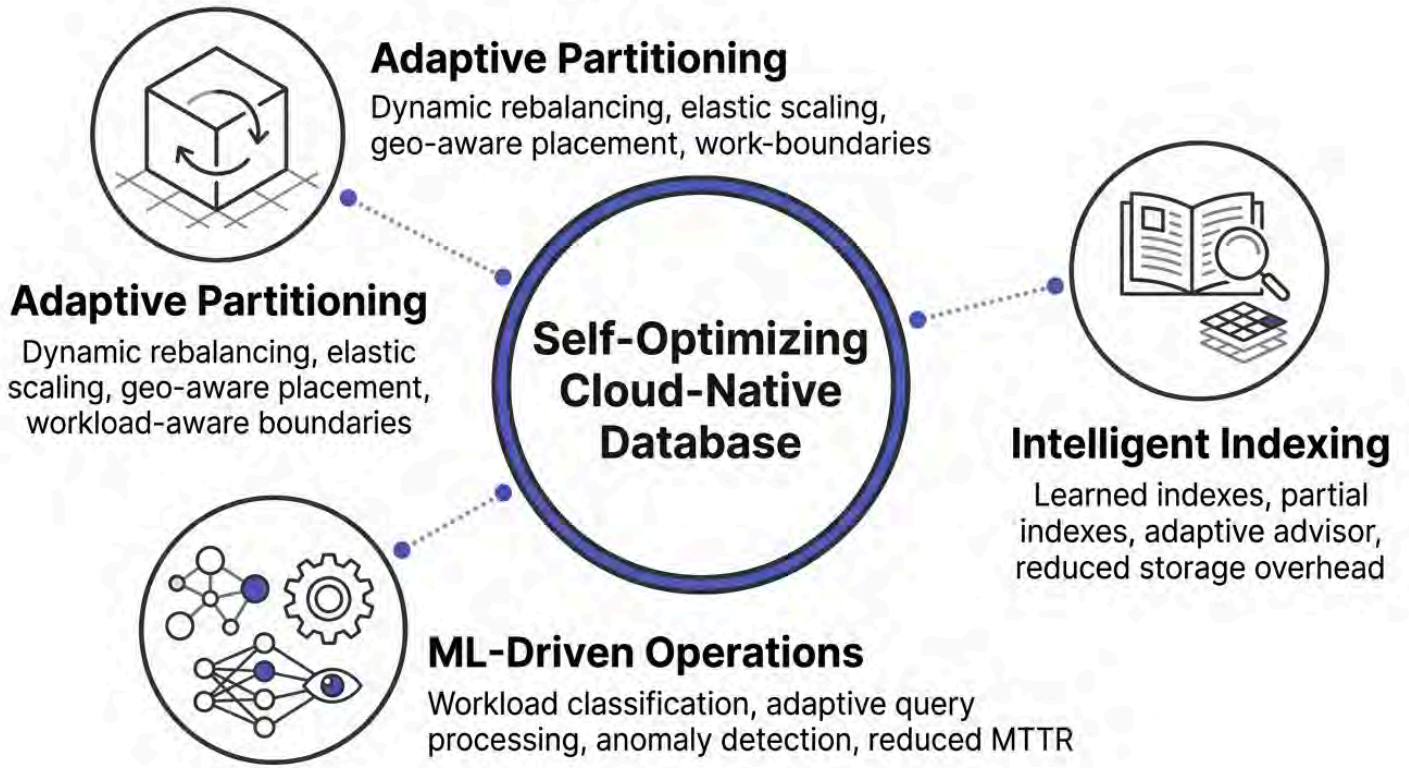
An extensible benchmarking framework that simulates diverse OLTP workloads including TPC-C, YCSB, and Twitter enabling apples-to-apples comparisons of partitioning and indexing strategies across database engines under realistic transactional load.

iTuned

An automated database configuration tuning system that uses Gaussian processes to explore the configuration space and recommend optimal knob settings. iTuned has been used to validate the performance uplift of ML-assisted tuning over manual DBA configuration in production-scale deployments.

The Self-Optimizing Database: Putting It All Together

Adaptive partitioning, learned indexes, and ML-driven workload management are not independent features they compose an integrated architecture for self-optimizing, resilient, and cost-efficient database systems.



Each pillar reinforces the others: better partitioning reduces index maintenance cost, learned indexes accelerate queries that the workload manager routes efficiently, and the ML layer continuously refines both partitioning decisions and index selection over time.

Impact on Developer Productivity and DBA Workload

For Developers

- Fewer manual index hints required the engine adapts to query patterns automatically
- Partition-transparent query routing eliminates the need to encode partition logic in application code
- Faster feedback loops during development as adaptive systems surface suboptimal access patterns early

For DBAs and Platform Engineers

- ML-driven configuration tuning replaces time-consuming manual knob optimization cycles
- Automated index advisory surfaces actionable recommendations with quantified impact estimates
- Reduced mean-time-to-resolution through proactive anomaly detection and self-healing partition rebalancing

Safeguards, Guardrails, and Operational Considerations

Self-optimizing systems introduce new failure modes alongside their benefits. Production adoption requires careful governance of when and how the system is permitted to act autonomously.

- **Rollback Safety**

  All autonomous index and partition changes should be reversible with a single operator action. Maintain a change audit log with before/after execution plan snapshots.

- **Threshold Guardrails**

  Define hard limits on autonomous actions maximum index count, minimum partition size, and rebalancing frequency caps to prevent runaway optimization loops.

- **Human-in-the-Loop Mode**

  For critical production systems, configure ML advisors to surface recommendations rather than apply them automatically. Approval workflows keep DBAs in control of high-risk changes.

- **Observability First**

  Every autonomous action must emit structured telemetry. Without end-to-end observability of optimization decisions, debugging performance regressions becomes intractable at scale.

Implementation Path

Getting Started: A Practical Adoption Framework

1

Stage 1 Observe

Instrument your database layer with query-level telemetry. Identify partition skew, index bloat, and slow query patterns before making changes.

2

Stage 2 Advise

Enable advisory mode on your database platform's index and partition advisor. Evaluate recommendations against your workload benchmarks using OLTP-Bench or equivalent tooling.

3

Stage 3 Automate Selectively

Enable

Thank you!