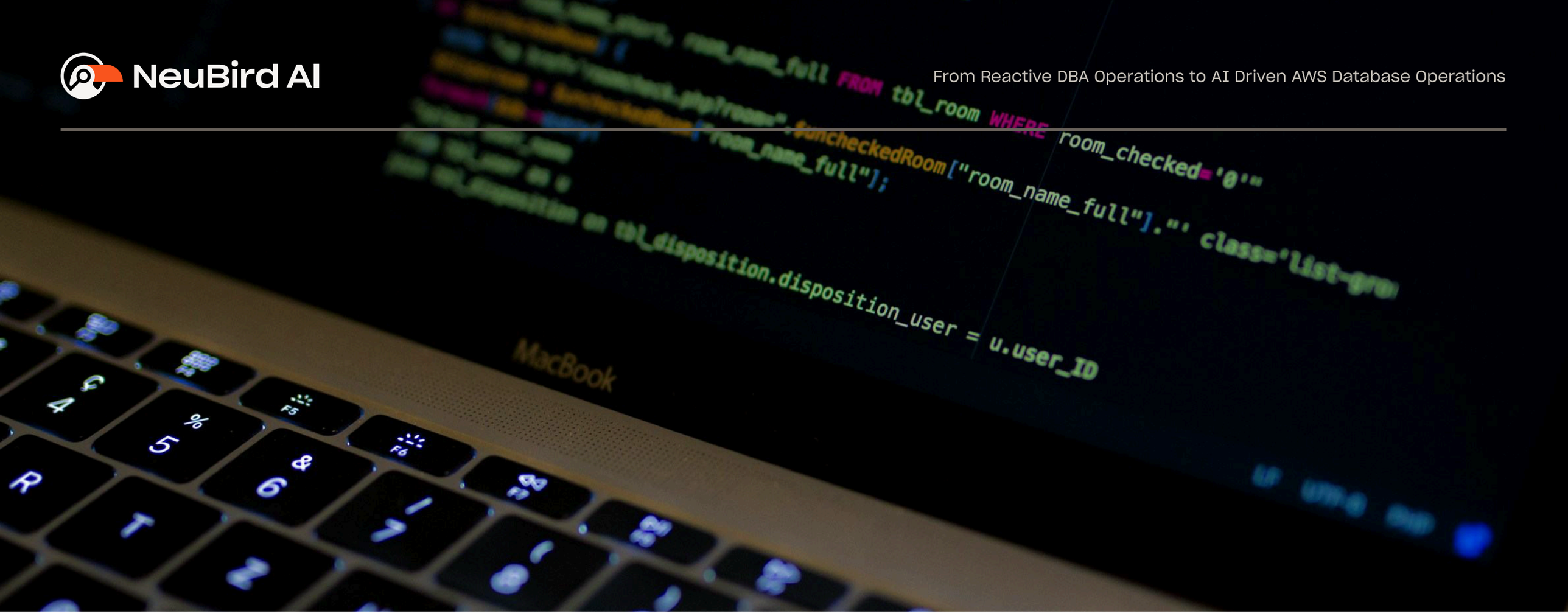
NeuBird AI

# From Reactive DBA Operations to AI Driven AWS Database Operations

Prevent. Resolve. Optimize




# AWS Database Operations Overview

Modern applications are powered by databases. In AWS, organizations now operate across a mix of relational, distributed, and serverless database services, each optimized for different workload patterns, scalability requirements, and operational models.

As environments scale, database operations become increasingly difficult to manage manually. Performance bottlenecks, replication lag, inefficient queries, autovacuum issues, storage growth, locking, hot partitions, and unpredictable workload spikes often emerge long before traditional monitoring tools surface actionable alerts.

This creates a growing operational challenge.

Database telemetry exists across CloudWatch, Performance Insights, query plans, logs, wait events, operating system metrics, application traces, and infrastructure services. But connecting these signals fast enough to prevent incidents or resolve outages remains heavily manual.

This is where AI driven operational intelligence becomes critical.

NeuBird AI's Production Ops Agent continuously analyzes AWS database environments using the three operational pillars of:

- Prevent
- Resolve
- Optimize

Rather than functioning as a passive reporting platform, NeuBird continuously investigates database behavior, correlates telemetry across services, identifies emerging risks, assists with incident resolution, and continuously improves performance and reliability over time.

# AWS Database Operations Overview

AWS offers multiple database services designed for different workload requirements. This whitepaper focuses on the following services.

| AWS Database Services | Primary Use Case | Key Strengths | Common Challenges |
|---|---|---|---|
| Amazon RDS | Managed relational databases | Simplicity and broad engine support | Rightsizing, storage growth, backups |
| Amazon RDS PostgreSQL | Open source transactional workloads | Advanced indexing and analytics | Autovacuum, bloat, I/O tuning |
| Amazon RDS SQL Server | Enterprise Microsoft workloads | Compatibility and transactional reliability | Licensing costs, locking, TempDB contention |
| Amazon Aurora | Cloud native relational database | High performance and fast failover | I/O optimization, replica lag, scaling |
| Amazon DynamoDB | Serverless NoSQL applications | Massive scalability and low latency | Hot partitions, throttling, inefficient access patterns |

While AWS offers strong native visibility tools, operational teams still face a major challenge:

Understanding not just what is happening, but why it is happening & what should happen next. That operational gap becomes even more critical as organizations adopt cloud native architectures, microservices, Kubernetes, and distributed applications built on RDS and DynamoDB.

## Why RDS and DynamoDB Matter Most

Among AWS database services, Amazon RDS and DynamoDB represent two of the most widely deployed operational data platforms.

Amazon RDS powers a massive percentage of enterprise transactional systems across engines including PostgreSQL, MySQL, MariaDB, Oracle, SQL Server, and Amazon Aurora. These services form the operational backbone of financial systems, SaaS platforms, ecommerce applications, ERP environments, customer facing applications, and enterprise line of business workloads running in AWS.

# Why RDS and DynamoDB Matter Most

While this guide primarily focuses on Amazon RDS PostgreSQL, Amazon RDS SQL Server, Amazon Aurora, and DynamoDB, the operational patterns discussed throughout this paper apply broadly across modern RDS environments. Challenges such as query inefficiency, storage growth, replication lag, locking, scaling bottlenecks, backup sprawl, and operational drift are common across nearly every managed database deployment in AWS.

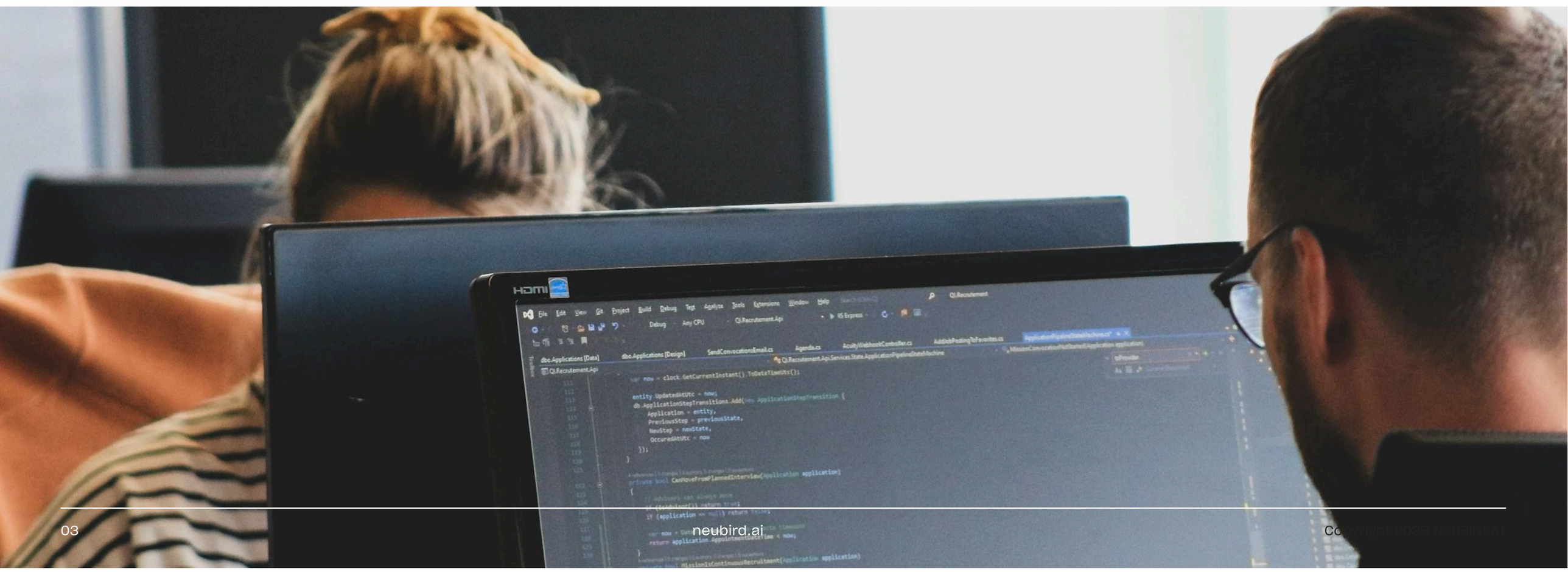
DynamoDB powers highly distributed cloud native applications requiring low latency and elastic scale. It is commonly used for serverless architectures, event driven systems, gaming, ecommerce, IoT, and high throughput APIs where traditional relational architectures struggle to scale efficiently.

Together, Amazon RDS and DynamoDB represent the operational foundation of modern AWS applications.

And increasingly, the operational complexity surrounding these environments exceeds what engineering teams can manually investigate in real time.

## Key Optimization Areas by Service

| Database Service | Optimization Focus |
|---|---|
| RDS PostgreSQL | Index tuning, autovacuum optimization, memory efficiency |
| RDS SQL Server | Wait analysis, TempDB optimization, parallelism tuning |
| RDS Aurora | I/O optimization, replica scaling, Serverless v2 efficiency |
| DynamoDB | Partition efficiency, access pattern optimization, GSI tuning |



# PREVENT:

## Detecting Database Problems Before They Become Incidents

Most database incidents do not begin as outages or catastrophic failures. They begin as subtle operational signals that quietly accumulate over time until they eventually impact application performance, availability, or customer experience.

In PostgreSQL environments, this may appear as growing table bloat, increasing IO:DataFileRead wait events, or autovacuum processes struggling to keep pace with transaction growth. In SQL Server, it often surfaces through lock contention, TempDB pressure, stale statistics, or excessive PAGEIOLATCH waits. Aurora workloads may show early signs through rising replica lag, storage I/O growth, increasing DBLoad, or memory pressure during scaling events. DynamoDB environments experience similar patterns through hot partitions, throttled requests, uneven throughput distribution, or excessive write amplification caused by inefficient access patterns.

Common early warning indicators include:

- Growing replication lag
- Increasing I/O wait events
- Expanding table or index bloat
- Slow checkpoint activity
- Lock contention and blocking sessions
- Excessive storage growth
- Hot DynamoDB partitions
- Rising write amplification
- Backup accumulation
- Inefficient query plans and sequential scans

Individually, these signals may appear minor. Together, they often represent the earliest indicators of larger operational instability.

Traditional monitoring platforms typically detect problems only after customer impact begins. Dashboards and alerts may show elevated latency, high CPU utilization, or increased error rates, but they rarely provide enough operational context to explain why the issue is developing or which upstream changes contributed to it.

NeuBird AI approaches prevention differently. The Production Ops Agent continuously evaluates database telemetry, workload behavior, wait events, infrastructure utilization, replication activity, query performance, and operational drift across AWS database environments in real time. Instead of relying solely on threshold based alerting, NeuBird correlates operational signals across services and identifies emerging risks before they escalate into incidents.

This shift from reactive monitoring to continuous operational intelligence becomes increasingly important as environments scale. Modern cloud native architectures generate more telemetry and operational dependencies than engineering teams can manually investigate fast enough, particularly during periods of rapid growth or infrastructure change.

# Why Prevention Matters

Reactive database operations are expensive, operationally disruptive, and difficult to scale.

The longer performance degradation remains undetected:

- The harder root cause analysis becomes
- The greater customer impact grows
- The more operational teams rely on war rooms
- The more difficult remediation becomes
- The more technical debt accumulates across the environment

In many organizations, engineers spend significant amounts of time manually correlating logs, metrics, query plans, wait events, and infrastructure telemetry simply to determine where an issue originated. By the time the root cause is identified, the operational and business impact has often already grown substantially.

Preventive operational intelligence changes this model.

By continuously identifying abnormal behavior patterns, workload drift, inefficient queries, replication anomalies, storage growth, and emerging resource bottlenecks, NeuBird helps operational teams stabilize database environments before incidents occur. Instead of reacting to outages after they impact production systems, teams can proactively investigate and remediate risks while they are still manageable.

The result is:

- Fewer escalations
- Faster investigations
- Improved database reliability
- Reduced operational overhead
- More predictable performance across AWS database environments

## Common Database Risks, NeuBird Helps Prevent

| Database Service | Common Preventable Risks |
|---|---|
| RDS PostgreSQL | Autovacuum failures, table bloat, excessive I/O waits |
| RDS SQL Server | Blocking, TempDB contention, stale statistics |
| Aurora | Replica lag, memory pressure, excessive storage I/O |
| DynamoDB | Hot partitions, throttling, poor partition key distribution |
| All Services | Storage growth, backup sprawl, inefficient scaling |

# Example NeuBird Prevent Prompts

## RDS PostgreSQL

- Which PostgreSQL tables have excessive dead tuples indicating autovacuum issues
- Which queries are generating the highest IO:DataFileRead waits
- Are there indexes experiencing significant bloat
- Which databases are approaching transaction ID wraparound risk

## RDS SQL Server

- Which SQL Server instances are experiencing locking contention
- Are there queries generating excessive PAGEIOLATCH waits
- Which databases have stale statistics impacting execution plans
- Is TempDB contention increasing during peak hours

## RDS Aurora

- Which Aurora clusters are experiencing growing replica lag
- Are there workloads generating excessive VolumeReadIOPS
- Which Aurora clusters would benefit from I/O Optimized storage
- Which queries are causing sustained DBLoad spikes

## DynamoDB

- Which DynamoDB tables are experiencing throttled requests
- Are there hot partitions driving uneven workload distribution
- Which GSIs are generating excessive throughput consumption
- Which tables would benefit from On Demand scaling

# RESOLVE:

## Accelerating Database Incident Resolution

Database incidents are rarely isolated to a single metric or service. What initially appears as a database slowdown often involves a much broader chain of operational dependencies spanning infrastructure, applications, APIs, storage systems, network traffic, scaling behavior, and workload changes occurring simultaneously across the environment.

A single performance incident may involve:

- Query regressions after a deployment
- Storage latency and increasing I/O wait events
- Lock contention or blocking transactions
- Replica lag across Aurora or PostgreSQL clusters
- Memory exhaustion or cache pressure
- Network bottlenecks and cross AZ traffic
- Application retry storms amplifying load
- Kubernetes autoscaling behavior
- API traffic spikes or uneven workload distribution
- Connection storms overwhelming database pools

These issues rarely exist in isolation. A poorly optimized query may increase storage reads, which increases I/O latency, which then causes connection saturation and application retries that further amplify the original problem. In distributed cloud environments, small operational inefficiencies can rapidly cascade across multiple systems.

Traditional troubleshooting workflows are not designed for this level of operational complexity.

Engineers are often forced to manually correlate telemetry across dashboards, CloudWatch metrics, logs, Performance Insights, query plans, traces, Kubernetes events, and infrastructure monitoring tools to determine where the issue originated. This process is slow, fragmented, and heavily dependent on tribal knowledge and operator experience.

NeuBird changes this operational model.

The Production Ops Agent continuously investigates incidents using cross system operational context and evidence based workflows. Instead of simply surfacing alerts, NeuBird correlates changes across infrastructure, workloads, database telemetry, application behavior, and dependencies to reconstruct what happened and identify the most likely root cause.

Rather than forcing teams to manually pivot across tools during an outage, the Production Ops Aget helps operational teams move directly from detection to investigation and guided resolution.

# Why Resolution Speed Matters

Database incidents directly impact business operations in real time.

Even relatively short periods of degraded database performance can affect:

- Revenue generating transactions
- Customer experience and application responsiveness
- API latency and downstream services
- Transaction processing and data consistency
- Operational confidence across engineering teams
- Service availability and SLA commitments

The longer incidents persist, the more difficult investigations become. Telemetry volumes increase, operational noise compounds, and teams often become trapped in lengthy war room troubleshooting sessions where engineers manually attempt to reconstruct the timeline of events.

Reducing Mean Time to Resolution is not simply about identifying alerts faster. It requires understanding the relationships between infrastructure behavior, workload changes, query execution, replication activity, application traffic, and operational dependencies across the environment.

This is where context aware operational intelligence becomes essential.
NeuBird continuously analyzes operational signals in real time to identify contributing factors, correlate related anomalies, surface likely root causes, and recommend the next investigative or remediation steps. Instead of relying solely on dashboards and static alerts, operational teams gain an evidence based understanding of how the incident evolved and what actions are most likely to restore stability quickly.

The result is:

- Faster root cause identification
- Reduced war room duration
- Lower operational overhead
- Improved application uptime
- Faster recovery across AWS database environments

Reducing Mean Time to Resolution is not simply about identifying alerts faster. It requires understanding relationships across the operational environment.

# Why Root Cause Analysis Remains Manual

AWS provides powerful monitoring and observability capabilities. Using Amazon RDS PostgreSQL as an example, services such as CloudWatch, Performance Insights, Enhanced Monitoring, CloudWatch Logs, and CloudTrail provide visibility into infrastructure health, database performance, query activity, operating system metrics, log events, and configuration changes.

*“The challenge is not a lack of telemetry.”*

The challenge is that each tool provides only a portion of the operational story. During an incident, database administrators, SREs, and operations teams are often forced to pivot across multiple consoles, dashboards, logs, and data sources to understand how seemingly unrelated events connect together. A CPU spike may originate from a query regression. A query regression may be tied to a recent deployment. A deployment may have triggered storage pressure, lock contention, application retries, and cascading failures across dependent services.

While AWS native tools excel at reporting symptoms, engineers remain responsible for reconstructing operational context, determining root cause, and identifying the most effective remediation path. This process is often manual, time consuming, and highly dependent on individual expertise.

The following examples illustrate the reality of modern database operations: what teams must investigate using native AWS tools today, where operational gaps exist, and how AI driven operational intelligence transforms the investigation process.

## What Each Native AWS Tool Contributes

Amazon RDS PostgreSQL investigations typically require information from multiple AWS services. Each service provides valuable insight into a specific operational domain, but none delivers a complete end to end understanding of database behavior on its own.

The following matrix illustrates the primary functions provided by each AWS native monitoring service and highlights where capabilities overlap across the monitoring stack.

# Amazon RDS PostgreSQL AWS Native Monitoring Tools Matrix

| Capability | Cloudwatch | Performance Insights | Enhanced Monitoring | CloudWatch Logs | Cloud Trails | Devops Guru for RDS |
|---|---|---|---|---|---|---|
| CPU Monitoring | ✓ | ⚠ Indirect | ✓ | ✗ | ✗ | ✓ |
| Memory Monitoring | ⚠ Limited | ✗ | ✓ | ✗ | ✗ | ✓ |
| Disk I/O Monitoring | ✓ | ⚠ Waits | ✓ | ✗ | ✗ | ✓ |
| Network Monitoring | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ |
| Connection Counts | ✓ | ✓ | ✓ | ⚠ | ✗ | ✓ |
| Storage Utilization | ✓ | ✗ | ✗ | ✗ | ✗ | ⚠ |
| Replication Lag | ✓ | ✗ | ✗ | ⚠ | ✗ | ⚠ |
| Query Performance | ✗ | ✓ | ✗ | ⚠ | ✗ | ⚠ |
| Top SQL Analysis | ✗ | ✓ | ✗ | ⚠ | ✗ | ⚠ |
| Wait Event Analysis | ✗ | ✓ | ✗ | ✗ | ✗ | ⚠ |
| Blocking Sessions | ✗ | ✓ | ✗ | ⚠ | ✗ | ✗ |
| Lock Contention | ✗ | ✓ | ✗ | ⚠ | ✗ | ⚠ |
| Deadlock Detection | ✗ | ⚠ | ✗ | ✓ | ✗ | ⚠ |
| Slow Query Detection | ✗ | ✓ | ✗ | ✓ | ✗ | ⚠ |
| Error Monitoring | ⚠ | ✗ | ✗ | ✓ | ✗ | ⚠ |
| PostgreSQL Analysis | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ |
| Autovacuum Monitoring | ✗ | ⚠ | ✗ | ✓ | ✗ | ⚠ |
| Configuration Changes | ✗ | ✗ | ✗ | ✗ | ✓ | ⚠ |
| Parameter Group Changes | ✗ | ✗ | ✗ | ✗ | ✓ | ⚠ |
| Security Group Changes | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ |
| Instance Modification | ✗ | ✗ | ✗ | ✗ | ✓ | ⚠ |
| Failover Events | ⚠ | ✗ | ✗ | ⚠ | ✓ | ⚠ |
| Maintenance Tracking | ⚠ | ✗ | ✗ | ✗ | ✓ | ✗ |
| Root Cause Suggestions | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ |
| Anamoly Detection | ⚠ | ✗ | ✗ | ✗ | ✗ | ✓ |
| Alerting | ✓ | ⚠ | ⚠ | ⚠ | ✗ | ✓ |
| Historical Trending | ✓ | ✓ | Limited | Limited | Limited | Limited |

# The Anatomy of an RDS PostgreSQL Investigation

When a production database slowdown occurs, operations teams rarely find the answer in a single dashboard. Instead, engineers must piece together evidence from multiple sources including CloudWatch metrics, Performance Insights wait events, PostgreSQL logs, infrastructure changes, deployment history, and application telemetry.

While AWS provides excellent visibility into database performance, determining root cause often requires navigating across several tools and manually correlating findings. A CPU spike may ultimately be traced to a query regression. A query regression may be linked to a recent deployment. That deployment may have triggered storage I/O pressure, lock contention, application retries, and cascading performance issues across dependent services.

The workflow below illustrates a typical investigation process followed by DBAs, SREs, and cloud operations teams during an Amazon RDS for PostgreSQL incident. It demonstrates how teams move between AWS native tools, deployment systems, and application telemetry to reconstruct what happened, identify contributing factors, and determine the most likely root cause.

| Step | Question | Primary Tool/Source | Investigation Output |
|---|---|---|---|
| 1 | Is there a problem? | Amazon CloudWatch | CPU, memory, IOPS, connections, latency alerts |
| 2 | When did the issue start? | Amazon CloudWatch | Identify the exact start time and baseline deviation |
| 3 | Is the database overloaded? | CloudWatch & Enhanced Monitoring | Resource bottleneck analysis across CPU, memory, disk, and OS metrics |
| 4 | What is consuming database load? | Performance Insights | Top SQL, database load, waits, and active sessions |
| 5 | Are queries blocked? | Performance Insights | Lock contention, blocking sessions, and wait event details |
| 6 | Are errors occuring? | CloudWatch Logs | PostgreSQL errors, deadlocks, failed connections, and autovacuum activity |
| 7 | Did something change? | CloudTrail | Parameter, instance, security group, maintenance, or failover changes |
| 8 | Was a deployment released? | GitHub, GitLab, Jenkins, ArgoCD | Code, schema, infrastructure, or application release history |
| 9 | Is the issue application related? | X-Ray, Datadog, Dynatrace, New Relic, or app logs | Trace, service dependency, and application level latency analysis |
| 9 | Is the issue application related? | X-Ray, Datadog, Dynatrace, New Relic, or app logs | Trace, service dependency, and application level latency analysis |
| 10 | Determine likely root cause | Human Operator | Manual correlation of all findings |
| 11 | Implement fix | DBA or Engineering team | Query tuning, rollback, scaling, parameter change, or index creation |
| 10 | Verify resolution | CloudWatch & Performance Insights | Confirm database load, waits, errors, & application latency returned to normal |

# The Manual Step-by-Step Investigation Process

1. Detect the symptom in CloudWatch: The investigation usually starts with a CloudWatch alarm. Common signals include high CPU, high read or write IOPS, connection saturation, increased latency, low free storage, or replication lag. CloudWatch establishes that a problem exists and helps identify when the incident began.

2. Confirm resource pressure with CloudWatch and Enhanced Monitoring. CloudWatch provides database level metrics, while Enhanced Monitoring adds operating system level detail. Together, they help determine whether the issue is CPU, memory, storage, disk queue, network, or connection related.

3. Use Performance Insights to identify database load. Performance Insights is the primary tool for understanding what is consuming PostgreSQL capacity. It shows database load, top SQL, wait events, and active sessions. This is where teams typically find whether the bottleneck is a query, lock, I/O wait, or connection behavior.

4. Check for blocking, locks, and wait events. If latency is high but infrastructure looks stable, the next step is to look for blocking sessions, lock contention, deadlocks, or dominant wait events. This helps determine whether the problem is caused by concurrency, transaction behavior, or query design.

5. Review PostgreSQL logs in CloudWatch Logs. Logs provide evidence that metrics alone do not show. Teams look for deadlocks, failed connections, authentication errors, autovacuum activity, checkpoint behavior, and slow query log entries when enabled.

6. Review CloudTrail for infrastructure or configuration changes. CloudTrail helps answer whether something changed before the incident. Useful events include parameter group changes, instance modifications, security group changes, manual failover, maintenance activity, or scaling related changes.

7. Correlate with deployment and application context. AWS native RDS tools do not automatically correlate database symptoms with code deployments, schema changes, CI/CD activity, application traces, or ticket history. Engineers usually check GitHub, GitLab, Jenkins, ArgoCD, X-Ray, Datadog, Dynatrace, New Relic, or application logs to connect the database issue to an application level cause.

8. Determine root cause and take action. After collecting evidence, the operator determines the likely root cause. Common fixes include rolling back a deployment, adding an index, tuning a query, increasing capacity, changing a parameter, resolving lock contention, or adjusting connection pooling.

9. Verify resolution. After remediation, CloudWatch and Performance Insights confirm whether resource utilization, database load, wait events, errors, and application latency returned to normal.

# Where AI Driven Investigation Changes the Process

The goal of AI driven operational intelligence is not to replace existing monitoring investments. CloudWatch, Performance Insights, CloudTrail, and other observability platforms remain essential sources of telemetry.

*“The operational challenge begins after an alert is generated”*

Teams must determine what changed, which systems are affected, how events relate to one another, whether similar incidents have occurred previously, and what actions should happen next. These activities consume the majority of investigation time during complex incidents. The following comparison highlights the operational capabilities that extend beyond traditional monitoring and demonstrates where NeuBird’s AI driven investigation negates manual effort, accelerates root cause identification, and improves overall Mean Time to Resolution.

## NeuBird AI Value Matrix

| Capability | AWS Native Coverage | NeuBird AI Value |
|---|---|---|
| Metrics Collection | Yes | Uses existing tools and analyzes their data |
| Query Performance | Yes | Uses existing tools and ties query behavior to incidents |
| Logs | Yes | Uses existing logs and extracts incident evidence |
| Change Analysis | Partial | Correlates configuration, infrastructure, and operational changes with symptoms |
| Deployment Correlation | No | Connects incidents to code, schema and CI/CD activity |
| Historical Incident Learning | No | Finds similar prior incidents and what resolved them |
| Investigation Plans | No | Creates step-by-step investigation paths based on the single pattern |
| Root Cause Hypothesis | No | Generates likely root cause hypotheses with supporting evidence |
| DBA Level Recommendations | No | Recommends query, index, configuration, connection, or capacity actions |
| Cross System Correlation | No | Connects RDS symptoms with app, Kubernetes, network, ticket, and deployment context |
| Natural Language Troubleshooting | No | Lets teams ask operational questions in plain language |
| Automated Investigation Workflow | No | Runs the investigation across multiple tools and summarizes findings |

# Where AI Driven Investigation Changes the Process

AWS native tools provide excellent visibility into the health and performance of Amazon RDS PostgreSQL environments. CloudWatch, Performance Insights, Enhanced Monitoring, CloudWatch Logs, and CloudTrail collectively deliver rich telemetry across infrastructure, database workloads, operating system metrics, query activity, logs, and configuration changes. These tools are highly effective at helping teams understand what is happening within the environment.

The operational challenge begins when teams must determine why an issue occurred. Root cause analysis often requires engineers to manually correlate database symptoms with recent deployments, configuration changes, application behavior, infrastructure events, historical incidents, and operational knowledge scattered across multiple systems. This investigative process is frequently time consuming, fragmented, and dependent on individual expertise.

AI driven investigation platforms address this gap by automatically connecting signals across the operational environment, reconstructing the sequence of events that led to an incident, identifying likely root causes, and recommending the most effective next steps. Rather than replacing existing monitoring investments, these platforms build upon them to transform raw telemetry into operational intelligence.

For this reason, NeuBird should not be viewed as another PostgreSQL monitoring tool. AWS already provides strong monitoring capabilities. NeuBird serves as the investigation and operational intelligence layer that connects AWS native telemetry, correlates changes across systems, identifies likely causes of performance degradation, and guides teams toward faster resolution with evidence based recommendations and next actions.

## Common Database Incident Patterns

| Incident Type | Common Root Cause |
|---|---|
| Slow Queries | Missing indexes, stale statistics, inefficient execution plans |
| High I/O Latency | Table scans, storage pressure, memory exhaustion |
| Connection Storms | Poor pooling, application retries |
| Aurora Slowdown | Replica lag, insufficient ACUs, storage contention |
| DynamoDB Throttling | Hot partitions, improper scaling |
| SQL Server Blocking | Long running transactions, locking contention |

# Example NeuBird Resolve Prompts

## RDS PostgreSQL

- What queries are driving the highest database load right now
- Which wait events are dominating Performance Insights
- Are there sequential scans causing excessive I/O
- Which queries experienced regressions after deployment

## RDS SQL Server

- Which sessions are currently blocking other transactions
- What are the top SQL Server wait types over the past hour
- Which queries are generating excessive CPU utilization
- Are execution plan regressions contributing to performance degradation

## RDS Aurora

- Which Aurora queries are driving high DBLoadNonCPU
- Are storage latency issues contributing to query slowdown
- Which replicas are falling behind the primary cluster
- What changed immediately before the slowdown began

## DynamoDB

- Which partition keys are generating throttling
- Are GSIs contributing to write amplification
- Which workloads are causing uneven capacity utilization
- What API calls correlate with increased DynamoDB latency

# OPTIMIZE:

## Continuously Improving Database Performance and Reliability

Optimization is not a one time tuning exercise. AWS database environments are continuously changing as applications evolve, workloads expand, and infrastructure becomes increasingly distributed.

Over time:

- Data volumes grow
- Query patterns shift
- Applications scale unpredictably
- Infrastructure changes occur continuously
- Workloads fluctuate throughout the day
- Replication demands increase
- Storage consumption expands
- Architectures become more distributed and service oriented

Even well tuned environments naturally drift toward inefficiency over time.

A PostgreSQL workload that performed efficiently six months ago may now experience table bloat, inefficient execution plans, or increasing I/O pressure as data scales. SQL Server environments may gradually accumulate index fragmentation, stale statistics, blocking contention, or TempDB bottlenecks. Aurora clusters often experience changing storage I/O behavior, replica scaling challenges, or Serverless v2 tuning inefficiencies as application demand evolves. DynamoDB environments may begin developing hot partitions, uneven throughput consumption, or inefficient access patterns as application traffic changes.

Traditional optimization approaches are typically reactive and periodic. Teams review dashboards, analyze reports, or perform tuning exercises after performance degradation has already become noticeable. This creates a cycle where optimization efforts quickly become outdated as environments continue evolving.

NeuBird approaches optimization as a continuous operational discipline rather than a scheduled maintenance activity.

The Optimize pillar within the Production Ops Agent continuously evaluates workload efficiency, database performance, scaling behavior, operational stability, reliability patterns, and infrastructure health across AWS database services in real time. Instead of relying on static thresholds or periodic reviews, NeuBird continuously investigates how workloads behave as the environment changes.

This allows teams to proactively improve performance, stabilize workloads, and optimize operational efficiency before degradation impacts production systems.

# Optimization Focus Area

Unlike the Ultimate Guide to AWS Cost Optimization white paper, which focused primarily on reducing cloud spend, this guide focuses specifically on continuously improving database performance, scalability, and operational reliability across AWS database environments.

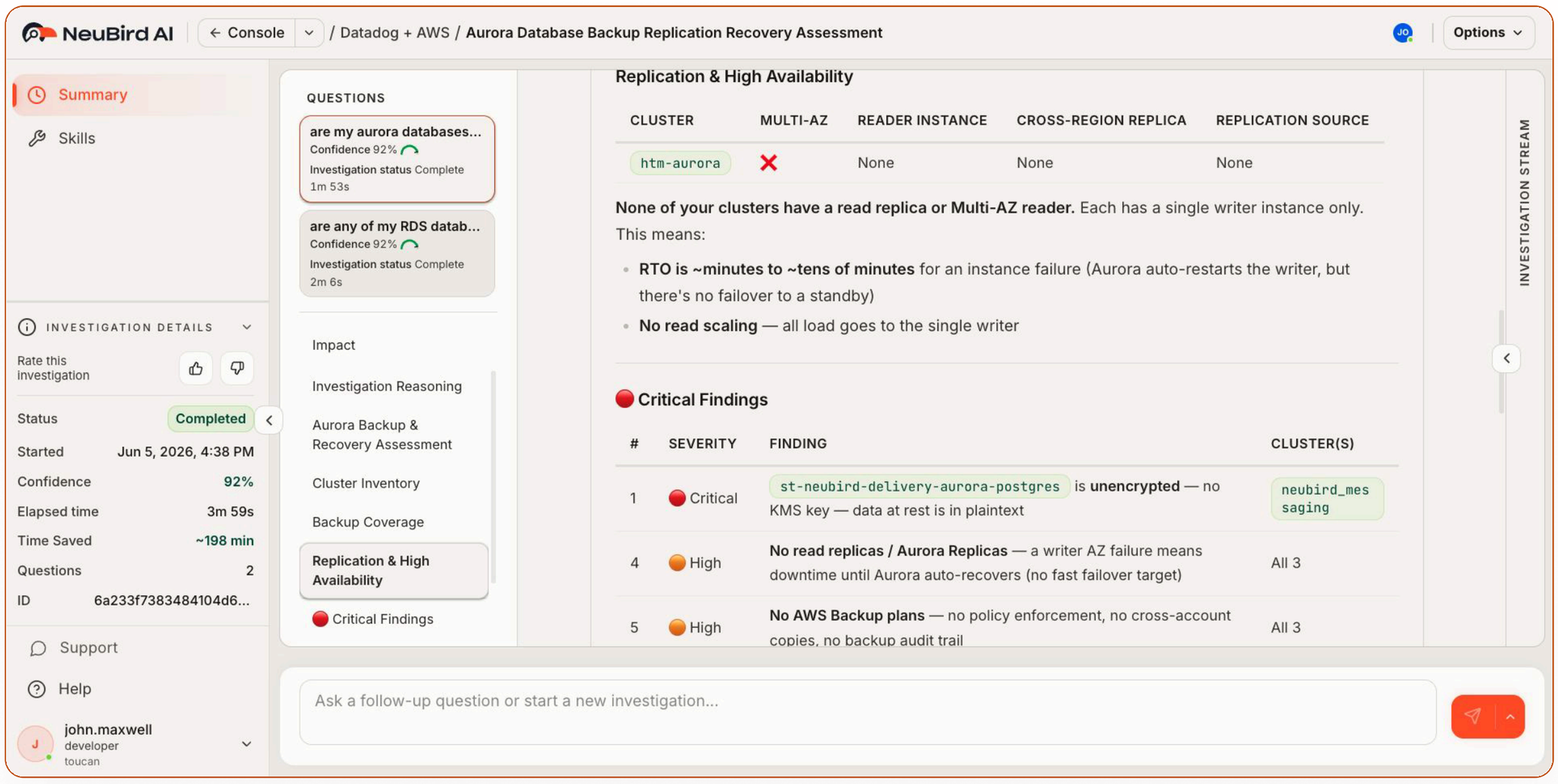
Key optimization focus areas include:

- Performance optimization across queries, indexes, memory utilization, and storage I/O
- Reliability optimization to improve uptime, replication health, and workload stability
- Scaling optimization for Aurora, DynamoDB, and cloud native distributed workloads
- Operational stability across changing infrastructure and application environments
- Workload efficiency improvements to reduce latency and resource contention
- Continuous tuning of database behavior as application demand evolves

This becomes increasingly important as organizations modernize applications and adopt cloud native architectures. Kubernetes deployments, microservices, APIs, event driven systems, and distributed applications introduce operational complexity that changes far more rapidly than traditional database administration processes were designed to handle. NeuBird continuously analyzes operational patterns to identify optimization opportunities across database services, infrastructure layers, query behavior, and workload distribution. Rather than simply reporting metrics, the platform helps teams understand why inefficiencies are developing and which changes will have the greatest operational impact.

The result is:

- More predictable database performance
- Improved workload stability
- Better scalability during traffic growth
- Reduced operational drift
- Increased reliability across AWS database environments



# Example NeuBird Optimize Prompts

## RDS PostgreSQL

- Which indexes would reduce IO:DataFileRead waits most significantly
- Which queries have the highest physical read percentage
- Are memory settings causing excessive disk spill activity
- Which workloads would benefit from covering indexes

## RDS SQL Server

- Which wait types are impacting throughput most significantly
- Are parallelism settings causing excessive CXPACKET waits
- Which databases would benefit from optimized TempDB configuration
- Which indexes are fragmented and degrading performance

## RDS Aurora

- Which Aurora clusters should migrate to I/O Optimized storage
- Are workloads exceeding optimal Serverless v2 ACU scaling thresholds
- Which queries are generating excessive VolumeWriteIOPS
- Which clusters have low BufferCacheHitRatio

## DynamoDB

- Which tables should move from Provisioned to On Demand capacity mode
- Which GSIs are consuming disproportionate throughput
- Which workloads would benefit from DAX or caching
- Which access patterns are creating uneven throughput distribution

# Why AI Changes Database Operations

Traditional database administration was built for a very different era of infrastructure.

DBAs and operations teams managed relatively static environments with predictable workloads, centralized applications, and limited operational sprawl. Performance troubleshooting was often manual, but manageable. Teams could review logs, analyze queries, inspect wait events, and isolate problems within a relatively contained operational environment.

Modern AWS database environments no longer operate that way.
Today's applications span distributed services, Kubernetes clusters, APIs, serverless workloads, relational databases, caching layers, messaging systems, and cloud infrastructure that changes continuously throughout the day.
Applications scale automatically, workloads fluctuate unpredictably, deployments occur constantly through CI/CD pipelines, and operational dependencies stretch across multiple services and environments simultaneously.

At the same time, the volume of telemetry generated across AWS database environments has exploded.

Performance Insights, CloudWatch metrics, query execution plans, wait events, traces, logs, replication metrics, infrastructure telemetry, storage activity, Kubernetes events, and application signals now generate massive amounts of operational data every second.

The challenge is no longer collecting telemetry.

The challenge is understanding operational context quickly enough to act before instability spreads across the environment. This is where traditional operational models begin to break down.

Most database monitoring platforms were designed primarily for visibility. They surface dashboards, trigger alerts, and identify when thresholds have been exceeded. These platforms remain essential, but they are fundamentally designed to report symptoms rather than investigate operational causality.

Traditional monitoring tools typically answer questions such as:

- What happened
- What metric changed
- What alert fired
- Which threshold was exceeded
- Which database or resource is under pressure
-

What they often cannot determine on their own is:

- Why the issue occurred
- Which operational change triggered the degradation
- Which workloads or applications are impacted
- Which dependencies contributed to the incident
- Whether the issue is isolated or part of a larger operational pattern
- What actions should happen next to stabilize the environment

# Why AI Changes Database Operations

This limitation becomes increasingly significant as environments scale in complexity.

A single database slowdown may involve interactions across query regressions, storage I/O pressure, replication lag, Kubernetes autoscaling behavior, application retry storms, infrastructure bottlenecks, or sudden API traffic spikes occurring simultaneously across multiple systems. Engineers are then forced to manually pivot across dashboards, logs, query plans, traces, and infrastructure telemetry attempting to reconstruct what changed under time pressure.

This process is slow, fragmented, and heavily dependent on tribal knowledge.

AI driven operational intelligence changes this operational model.

NeuBird's Production Ops Agent continuously analyzes AWS database environments in real time, correlating signals across RDS, Aurora, PostgreSQL, SQL Server, DynamoDB, Kubernetes, cloud infrastructure, and application services. Instead of simply surfacing anomalies or alerts, the platform reconstructs operational context to help teams understand not only what failed, but why it failed, what changed, which systems were impacted, and what actions should happen next.

Operational teams gain systems capable of continuously:

- Detecting emerging risks before they become incidents
- Investigating abnormal workload behavior automatically
- Correlating telemetry across databases, infrastructure, applications, and cloud services
- Identifying likely root causes using operational context
- Optimizing performance continuously as workloads evolve
- Improving reliability and operational stability over time

This is the operational shift from monitoring to intelligence.

An alert may indicate elevated latency on a PostgreSQL cluster, but operational intelligence explains that the latency increase began after a deployment introduced inefficient sequential scans, which increased storage I/O, amplified connection retries, and ultimately degraded dependent application services.

That level of operational reasoning is extremely difficult to achieve manually, particularly during high pressure incidents where operational noise and fragmented telemetry already overwhelm engineering teams.

# Why AI Changes Database Operations

The result is a fundamentally different operational model.

Instead of spending hours manually investigating incidents after customer impact begins, organizations gain continuous visibility into workload behavior, operational drift, scaling inefficiencies, query regressions, infrastructure bottlenecks, and reliability risks as they emerge.

This allows teams to move beyond reactive firefighting and toward continuously optimized, AI assisted database operations built around NeuBird's three operational pillars:

PREVENT

RESOLVE

OPTIMIZE

As AWS environments continue growing in complexity, operational intelligence will increasingly become the difference between teams that scale reliably and teams overwhelmed by alert fatigue, operational noise, and escalating database complexity.

## Closing Perspective

AWS database services provide incredible scalability and flexibility, but operational complexity grows alongside them.

The challenge is no longer collecting telemetry.

The challenge is understanding operational context fast enough to prevent incidents, resolve outages, and continuously optimize performance and reliability.

That is where AI driven operational intelligence becomes essential.

NeuBird's Production Ops Agent enables organizations to move beyond dashboards and reactive investigations toward continuous database operations built around Prevent, Resolve, and Optimize.

# Appendix

## NeuBird AI Prompt Cheat Sheet for AWS Database Operations

### Amazon RDS PostgreSQL

- Which PostgreSQL queries are generating the highest I/O wait events
- Which queries have the highest physical read percentage
- Are there tables experiencing excessive bloat or dead tuple accumulation
- Are autovacuum settings preventing transaction wraparound risk
- Which indexes are fragmented, inefficient, or missing entirely
- Which workloads would benefit from covering indexes
- Are memory settings causing excessive disk spill activity
- Which databases are approaching transaction ID exhaustion

### Amazon RDS SQL Server

- Which wait types are contributing most to latency and throughput degradation
- Are there blocking sessions impacting transaction performance
- Which queries have execution plan regressions
- Are parallelism settings causing excessive CXPACKET waits
- Which databases have stale statistics impacting performance
- Is TempDB contention contributing to slowdown
- Which indexes are fragmented and degrading query efficiency
- Which workloads are generating excessive PAGEIOLATCH waits

### Amazon Aurora

- Which Aurora clusters are generating excessive storage I/O
- Which queries are generating excessive VolumeWriteIOPS
- Are there replica lag issues impacting read performance
- Which clusters should migrate to I/O Optimized storage
- Are Serverless v2 ACU limits impacting workload performance
- Which clusters have low BufferCacheHitRatio
- Which workloads are causing CommitLatency spikes
- Which Aurora workloads are experiencing scaling inefficiencies

### Amazon DynamoDB

- Which DynamoDB tables are experiencing throttled requests
- Which partition keys are generating the hottest workloads
- Are scans generating unnecessary read consumption
- Which GSIs are over consuming throughput
- Which tables should move from Provisioned to On Demand capacity mode
- Which workloads would benefit from DAX or caching
- Which tables would benefit from TTL policies
- Which access patterns are creating uneven throughput distribution

### Cross Platform Operational Prompts

- What changed before database latency increased
- Which deployments correlate with performance degradation
- Which workloads are generating abnormal database traffic
- Which services are contributing most to operational risk
- What operational anomalies should be investigated immediately
- Which infrastructure changes correlate with database instability
- Which workloads are degrading overall application performance
- What operational patterns indicate emerging reliability risks

NeuBird AI

## About NeuBird AI

NeuBird AI delivers an AI driven Production Ops Agent that helps enterprises prevent incidents, accelerate resolution, and optimize cloud operations. Using agentic AI and context engineering, NeuBird analyzes telemetry and operational context across modern infrastructure environments to automate investigations and reduce operational complexity.

Visit NeuBird.ai to know more