Root Cause Analysis

Incident Report: Vertex Database Outage

SEVERITY: CRITICAL
Incident ID: **INC-2026-8941**

Date of Incident	Incident Lead	Impacted Service	Document Version
July 14, 2026	Dr. Marcus Chen	Vertex Core API / DB	v1.2 (Final)

1 Executive Summary

On July 14, 2026, at 08:15 UTC, the Vertex core database platform experienced a critical resource exhaustion failure, resulting in 45 minutes of complete API service degradation and timeout issues for end users. The incident was triggered by a rapid, compounding memory leak in the primary database server's buffer cache. The root cause was traced to a newly deployed analytics query that executed a full-table scan without appropriate database indexing. The incident was resolved at 09:00 UTC after database promotion of a read-only replica and temporary isolation of the offending query.

1.1 Key Metrics

- **Total Downtime:** 45 minutes (08:15 UTC – 09:00 UTC)
- **Impacted Users:** Approximately 12,000 active customer sessions on Vertex Cloud
- **Financial SLA Impact:** Estimated $35,000 USD in SLA credit obligations
- **Recovery Time Objective (RTO) Status:** Exceeded target of 15 minutes by 30 minutes

2 Incident Timeline

The timeline below details the chronological sequence of events, from initial system warnings to final recovery verification and incident sign-off.

Time (UTC)	Incident Event / Observation	Logged By
08:15	Automated health check fails on DB-Primary. Latency spike $>$ 5,000ms.	Nagios Monitoring Agent
08:18	PagerDuty alerts database reliability engineer (DBRE) on duty.	PagerDuty Automation
08:22	Sarah Jenkins (Senior DBRE) logs into console. Memory utilization at 98%.	Operations Console
08:28	System memory exhausted. Kernel OOM-killer terminates the PostgreSQL service.	Linux Syslog Daemon
08:35	DB fails to restart automatically due to lock file corruption. Active failover initiated.	Dr. Marcus Chen (Lead)
08:42	Read-only replica promoted to Primary. API services resume partial read function.	Sarah Jenkins
08:50	Offending query isolated and blocked at API gateway layer. Write functions restored.	David Vance (Core Dev)
09:00	Core database fully online. Latency returns to normal ($<$ 50ms). Incident resolved.	Sarah Jenkins

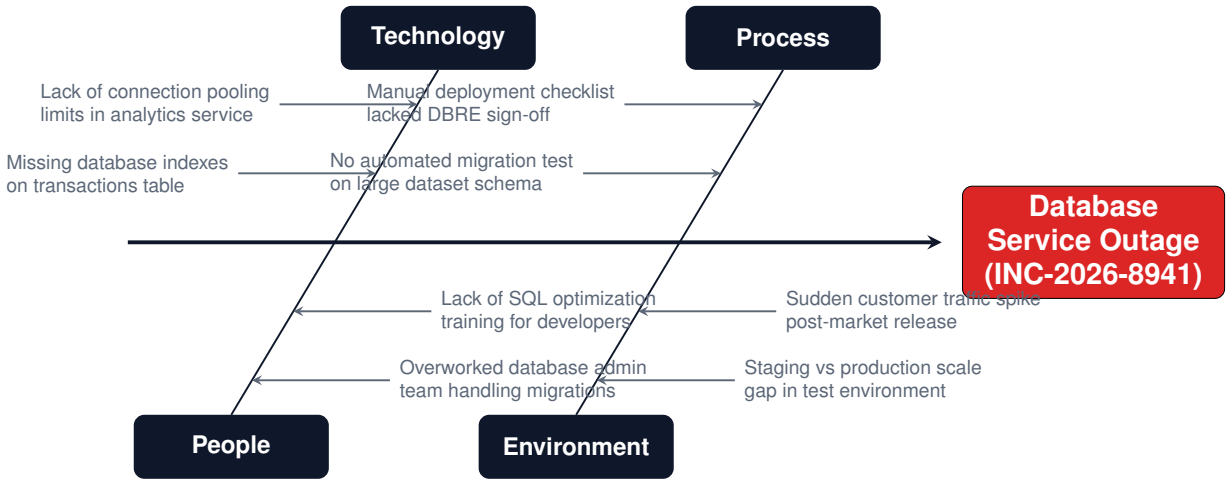
3 Problem Investigation

Post-incident analysis revealed that the crash occurred due to memory starvation (Out-Of-Memory) on the primary database host. The profiling data collected during the outage showed that the active memory pool was consumed entirely by query execution plans associated with the `/v2/analytics/transactions` endpoint.

A newly deployed backend service update included a reporting query that attempted to aggregate transaction histories by customer ID and creation timestamp. However, because the database migration script did not apply a composite index across the `customer_id` and `created_at` columns, PostgreSQL was forced to execute a sequential scan of 45 million rows. This query, executed concurrently by hundreds of active sessions, led to severe thrashing and eventual system shutdown.

3.1 Ishikawa (Fishbone) Diagram

The Ishikawa diagram below categorizes the contributing factors that led to this critical database outage, highlighting technical deficiencies, process gaps, and organizational constraints.



4 Lessons Learned

This incident highlights the risk of scaling database operations without formal governance and automated enforcement. Key lessons include:

1. Relying on manual checklists for schema changes is insufficient at our current scale.
2. Query profiling must be integrated into pre-production environments to catch full-table scans.
3. High-severity alerts should trigger automated failover systems faster than the manual intervention timeframe.

5 Root Cause Analysis (5-Whys)

To determine the systemic root cause of the database failure, the engineering team performed a structured 5-Whys analysis:

5-Whys Analysis

Problem Statement: The production database host crashed under load, resulting in 45 minutes of complete service disruption.

→ **Why 1?** Why did the database server crash?
Answer: System memory was exhausted, triggering the Linux Kernel's Out-Of-Memory (OOM) killer.

→ **Why 2?** Why was the system memory exhausted?
Answer: A single query scanning the large `transactions` table grew exponentially in memory under concurrent requests.

→ **Why 3?** Why did the query require so much memory?
Answer: The query lacked a composite index, forcing PostgreSQL to perform sequential full-table scans.

→ **Why 4?** Why was there no index on these columns?
Answer: The backend feature was deployed without a database migration review by a Database Reliability Engineer.

→ **Why 5?** Why was the database migration review bypassed?
Answer: Our CI/CD deployment pipeline did not enforce automated checks for database migrations, and the development sprint checklist lacked a mandatory sign-off block for schema changes. **[Root Cause]**

6 Corrective & Preventative Actions

The following action plan outlines the specific steps required to resolve the immediate issues and prevent recurrence in the future.

Corrective Action Item	Owner	Target Date	Status
Deploy index for `customer_id` and `created_at`.	Sarah Jenkins	July 15, 2026	**Completed**
Configure connection pool memory limits at gateway.	David Vance	July 18, 2026	**Completed**
Implement automated query plan validation in CI/CD pipeline.	David Vance	July 25, 2026	**In Progress**
Conduct training on SQL execution plans and indexing.	Elena Rostova	August 10, 2026	**Scheduled**
Establish DBRE review guidelines for all schema changes.	Dr. Marcus Chen	July 30, 2026	**In Progress**

7 Incident Report Sign-Off

By signing below, the stakeholders acknowledge the root causes identified and commit to the execution of the corrective action plan.

Role	Name / Signature	Date
Incident Lead	Dr. Marcus Chen / _____	_____
Engineering VP	Elena Rostova / _____	_____
DBRE Lead	Sarah Jenkins / _____	_____