## Postmortem: Production Database Performance Degradation - June 5, 2024

horizontal line

# Placeholder image

### Issue Summary

**Duration:** 2 hours, 48 minutes - 3:12 PM - 5:00 PM

**Impact:** Slow response times for all user-facing applications that rely on our main production database. Users experienced delays in page loading, form submissions, and data retrieval across the platform (estimated 80% user impact).

**Root Cause:** A database query optimization issue caused excessive resource consumption on the primary database server.

### Timeline

* **3:12 PM :** User reports and application monitoring alerts indicate significantly increased response times for all user-facing applications.
* **3:15 PM :** The database operations team investigates the issue. Initial focus is on high server load due to unexpected traffic surge.
* **3:45 PM :** Analysis of database logs reveals a specific query experiencing abnormally long execution times.
* **4:00 PM :** The team investigates the problematic query and identifies a poorly optimized join operation that is causing excessive table scans.
* **4:15 PM :** The incident is escalated to the database administrator and application development team.
* **4:30 PM :** Developers analyze the problematic query and identify the inefficient join condition.
* **4:45 PM :** Developers deploy a hotfix to the application code, modifying the database query to use a more efficient join strategy.
* **5:00 PM :** Database performance returns to normal levels, and user-facing applications function as expected.

### Root Cause and Resolution

The root cause of the performance degradation was an inefficient database query within one of our core applications. The query, designed to join data from two large tables, utilized a poorly optimized join condition, leading to excessive table scans and high resource consumption on the database server.

The issue was resolved by deploying a hotfix to the application code. This hotfix modified the problematic query to leverage a more efficient join strategy, significantly reducing its execution time.

### Corrective and preventative measures

* **Code review process:** Integrate database performance analysis into the code review process, ensuring developers consider database efficiency during application development.
* **Automated query analysis:** Implement automated tools to analyze database queries and flag potentially inefficient operations before deployment.
* **Database indexing review:** Regularly review and optimize database indexes to improve query performance on frequently accessed data.
* **Performance testing:** Increase focus on performance testing during development and integration phases to identify and address potential database bottlenecks.

This incident highlights the importance of collaboration between application developers and database operations teams. By implementing the corrective and preventative measures outlined above, we can ensure efficient database utilization and prevent similar performance issues in the future.