Reporte de Performance

NGCheck - Análisis de Base de Datos y Código

Generado: 19 de Enero de 2026

Resumen Ejecutivo

Período Analizado

60 días

MySQL Slow Query Log

Queries Lentas Identificadas

5 críticas

>7 segundos promedio

Índices Faltantes

7 recomendados

En tablas críticas

Top Operaciones Más Lentas

MySQL Slow Query Log (queries >7s)

#	Query / Operación	Promedio	Ejecuciones/mes	Fila examinada
1	sp_auditcompanies(6)	362 seg	31	55,000,00
2	sp_detailofproductssoldbysalestats	15.45 seg	6	
3	SELECT PedidosYaOrder WHERE RestaurantId...	12.04 seg	21	

#	Query / Operación	Promedio	Ejecuciones/mes	Fila examinada
4	SELECT User WHERE EmailAddress	**7.73 seg**	21	
5	SELECT PrintJob WHERE isSuccess	**7.61 seg**	23	

Application Insights (endpoints alto volumen)

#	Endpoint	Promedio	Ejecuciones/día	Impacto total/día
6	POST Invoice/SaveInvoice	**1,110 ms**	~565	627 seg
7	POST Order/SaveOrder	**546 ms**	~1,160	633 seg
8	GET Order/GetOrderEditorData	**363 ms**	~1,870	679 seg

Detalle: sp_auditcompanies(6)

Patrón Identificado

- Se ejecuta **TODOS LOS DÍAS** a las 07:06 UTC
- Equivale a las **04:06 hora Argentina**
- Es un **job programado** (scheduler)
- Crecimiento: ~170,000 filas nuevas por día

Estadísticas

- Total ejecuciones: **31**
- Promedio duración: **362 seg (~6 min)**
- Mínimo: 345.6 seg (23-dic)
- Máximo: 417.6 seg (6-ene)
- Filas actuales: **59.2 millones**

Historial de Ejecuciones (últimos 60 días)

Fecha	Hora (UTC)	Duración	Filas examinadas
2026-01-19	07:06:46	**406.6 seg**	59,229,459
2026-01-18	07:06:43	403.1 seg	59,018,994
2026-01-17	07:06:47	406.7 seg	58,766,546
2026-01-16	07:06:37	396.2 seg	58,396,643
2026-01-15	07:06:16	375.2 seg	58,272,099
2026-01-14	12:00:08	**7.3 seg**	5,344
2026-01-14	07:06:17	376.3 seg	58,064,968
2026-01-13	07:06:28	387.4 seg	57,880,423
2026-01-12	07:06:15	374.4 seg	57,720,036
2026-01-11	07:06:20	379.3 seg	57,527,082
2026-01-10	07:06:08	367.7 seg	57,304,625

¿Qué hace este SP?

Es un reporte de auditoría que recorre **TODAS las empresas activas** y por cada una ejecuta ~10 queries para obtener:

- Cantidad de productos
- Fecha primer/último pedido
- Facturas activas
- Ciudad y Provincia
- Cantidad de pedidos activos
- Días sin ventas
- Facturas sin CAE
- % pedidos facturados

⚠️ Predicción de mejora con índices

Mejora estimada: 15-30% (de ~360 seg a ~250-300 seg)
El problema principal NO son los índices, sino el **CURSOR** que ejecuta ~20,000 queries secuenciales. Para mejoras significativas se requiere reescribir el SP eliminando el cursor.

Problemas Identificados en Código

Patrón N+1 con Includes Excesivos

```
// NG.Service\Services\InvoiceService.cs:82-119
// GetInvoiceByIdAsync tiene 15+ ThenIncludes anidados
.Include(x => x.Customer).ThenInclude(p => p.Addresses)
.Include(x => x.Customer).ThenInclude(p => p.Phones)
.Include(x => x.Items)
.Include(x => x.Order).ThenInclude(p => p.CustomerPayments)
    .ThenInclude(p => p.CustomerPaymentCards)
    .ThenInclude(p => p.CustomerPaymentCashes)
    .ThenInclude(p => p.CustomerPaymentVales)
    // ... 10+ más
```

Archivos afectados: InvoiceService.cs, OrderService.cs
Impacto: Queries con múltiples JOINs que degradan performance

Índices Faltantes en Entity Configurations

```
User.EmailAddress
Usado en login - 7.73s promedio
```

```
PrintJob.isSuccess
Filtro de jobs pendientes - 7.61s promedio
```

```
PedidosYaOrder (RestaurantId, OrderStatus, IsActive)
Integración PedidosYa - 12.04s promedio
```

```
Order (IsFinished, IsActive)
Filtros comunes en pedidos
```

Scripts de Índices Recomendados

```
-- ================================================
-- ÍNDICES CRÍTICOS PARA MYSQL
-- Servidor: nucleo-check-prod-srv
-- ================================================

-- 1. Índice para login de usuarios (reduce 7.73s a <100ms)
CREATE INDEX IX_User_EmailAddress ON `User` (EmailAddress);

-- 2. Índice para PrintJob (reduce 7.61s a <100ms)
CREATE INDEX IX_PrintJob_isSuccess ON PrintJob (isSuccess);

-- 3. Índice compuesto para PedidosYaOrder (reduce 12.04s a <500ms)
```

```
CREATE INDEX IX_PedidosYaOrder_RestaurantId_Status_Active
ON PedidosYaOrder (RestaurantId, OrderStatus, IsActive, DateCreated);

-- 4. Índice para búsquedas por OrderId y RestaurantId
CREATE INDEX IX_PedidosYaOrder_OrderId_RestaurantId
ON PedidosYaOrder (OrderId, RestaurantId);

-- 5. Índice para Order (filtros comunes)
CREATE INDEX IX_Order_IsFinished_IsActive
ON `Order` (IsFinished, IsActive, DateCreated);

-- 6. Índice para CustomerPayment
CREATE INDEX IX_CustomerPayment_OrderId_IsActive
ON CustomerPayment (OrderId, IsActive);

-- 7. Índice para Invoice
CREATE INDEX IX_Invoice_OrderId ON Invoice (OrderId);
CREATE INDEX IX_Invoice_Date_IsActive ON Invoice (Date, IsActive);
```

Plan de Acción Recomendado

Prioridad	Acción	Impacto Esperado	Esfuerzo
URGENTE	Crear índice en User.EmailAddress	-95% tiempo login	5 min
URGENTE	Crear índice en PrintJob.isSuccess	-95% tiempo impresión	5 min
URGENTE	Crear índice compuesto en PedidosYaOrder	-95% tiempo integración	5 min
ALTA	Reescribir sp_auditcompanies sin cursor	-90% tiempo reportes	4-8 hrs
ALTA	Usar AsSplitQuery() en InvoiceService	-50% tiempo SaveInvoice	30 min
MEDIA	Usar AsSplitQuery() en OrderService	-30% tiempo SaveOrder	30 min

---

Reporte generado automáticamente

Fuentes: Azure Log Analytics, Application Insights, Análisis de código