Guía de Optimización de Queries con Prisma

Introducción

Esta guía proporciona mejores prácticas y patrones para escribir queries optimizadas con Prisma en Inmova.

Principios Fundamentales

1. Select Solo lo Necesario

❌ **Evitar:**

```
// Fetch completo - trae TODOS los campos y relaciones
const buildings = await prisma.building.findMany({
  where: { companyId },
  include: {
    units: true,
    expenses: true,
    documents: true,
    // ... muchas más relaciones
  }
});
```

✅ **Hacer:**

```
// Select específico - solo lo que necesitas
const buildings = await prisma.building.findMany({
  where: { companyId },
  select: {
    id: true,
    nombre: true,
    direccion: true,
    tipo: true,
    units: {
      select: {
        id: true,
        estado: true,
        rentaMensual: true,
      },
    },
  },
});
```

Impacto: Reducción de ~80-90% en datos transferidos

2. Usar Agregaciones en lugar de Fetch + Cálculo

❌ **Evitar:**

```
// Fetch todos los pagos y calcular en JS
const payments = await prisma.payment.findMany({
  where: {
    contract: { unit: { building: { companyId } } },
    estado: 'pagado'
  }
});
const totalIncome = payments.reduce((sum, p) => sum + p.monto, 0);
```

✅ **Hacer:**

```
// Usar agregación en la BD
const result = await prisma.payment.aggregate({
  where: {
    contract: { unit: { building: { companyId } } },
    estado: 'pagado'
  },
  _sum: { monto: true },
  _count: true,
});
const totalIncome = result._sum.monto || 0;
```

Impacto: ~100x más rápido con grandes datasets

3. Paginación Eficiente (Cursor-based)

❌ **Evitar:**

```
// Offset-based pagination - ineficiente para páginas grandes
const page = 100;
const pageSize = 20;
const units = await prisma.unit.findMany({
  skip: page * pageSize, // Muy lento para páginas grandes
  take: pageSize,
});
```

✅ **Hacer:**

```
// Cursor-based pagination - escala bien
const units = await prisma.unit.findMany({
  take: 20,
  skip: cursor ? 1 : 0,
  cursor: cursor ? { id: cursor } : undefined,
  where: { estado: 'disponible' },
  orderBy: { createdAt: 'desc' },
});
```

Impacto: Tiempo constante independiente de la página

4. Limitar Resultados

❌ **Evitar:**

```
// Sin límite - puede traer millones de registros
const allPayments = await prisma.payment.findMany({
  where: { estado: 'pendiente' }
});
```

✅ **Hacer:**

```
// Con límite razonable
const recentPayments = await prisma.payment.findMany({
  where: { estado: 'pendiente' },
  take: 100,
  orderBy: { fechaVencimiento: 'asc' },
});
```

5. Evitar N+1 Queries

❌ **Evitar:**

```
// N+1: 1 query + N queries para cada unidad
const units = await prisma.unit.findMany();
for (const unit of units) {
  const building = await prisma.building.findUnique({
    where: { id: unit.buildingId }
  });
  // ...
}
```

✅ **Hacer:**

```
// 1 query con include/select
const units = await prisma.unit.findMany({
  include: {
    building: {
      select: {
        id: true,
        nombre: true,
      },
    },
  },
});
```

Helpers Predefinidos

Usar los helpers en `lib/query-optimization.ts`:

```
import {
  getBuildingsWithMetrics,
  getAvailableUnits,
  getActiveContracts,
  getPendingPayments,
  getExpiringContracts,
  getBuildingOccupancyStats,
  getMonthlyFinancialSummary,
} from '@/lib/query-optimization';

// Ejemplo: Dashboard de edificios
const buildings = await getBuildingsWithMetrics(companyId);

// Ejemplo: Unidades disponibles
const availableUnits = await getAvailableUnits();

// Ejemplo: Resumen financiero
const summary = await getMonthlyFinancialSummary(companyId, new Date());
```

Patrones Avanzados

Query Paralela con Promise.all

```
// Ejecutar múltiples queries en paralelo
const [buildings, units, contracts, payments] = await Promise.all([
  getBuildingsWithMetrics(companyId),
  getAvailableUnits(),
  getActiveContracts(companyId),
  getPendingPayments(companyId),
]);
```

Transacciones para Operaciones Múltiples

```
await prisma.$transaction(async (tx) => {
  // Todas las operaciones usan la misma transacción
  const contract = await tx.contract.create({ data: contractData });
  await tx.payment.createMany({ data: paymentsData });
  await tx.unit.update({
    where: { id: unitId },
    data: { estado: 'ocupada', tenantId: tenantId },
  });
});
```

Raw Queries para Casos Especiales

```
// Para queries muy específicas o con performance crítica
const result = await prisma.$queryRaw`
  SELECT
    b.id,
    b.nombre,
    COUNT(u.id) as total_units,
    COUNT(CASE WHEN u.estado = 'ocupada' THEN 1 END) as occupied_units,
    AVG(u."rentaMensual") as avg_rent
  FROM buildings b
  LEFT JOIN units u ON u."buildingId" = b.id
  WHERE b."companyId" = ${companyId}
  GROUP BY b.id, b.nombre
`;
```

Métricas y Monitoreo

Medir Performance de Queries

```
import { performance } from 'perf_hooks';
import logger from '@/lib/logger';

async function measureQuery<T>(
  name: string,
  query: () => Promise<T>
): Promise<T> {
  const start = performance.now();
  const result = await query();
  const duration = performance.now() - start;

  if (duration > 1000) { // Log slow queries (>1s)
    logger.warn(`Slow query detected: ${name} took ${duration}ms`);
  }

  return result;
}

// Uso
const buildings = await measureQuery(
  'getBuildingsWithMetrics',
  () => getBuildingsWithMetrics(companyId)
);
```

Analizar Query Plan

```
// Ver el plan de ejecución
const plan = await prisma.$queryRaw`
  EXPLAIN ANALYZE
  SELECT * FROM units
  WHERE "buildingId" = ${buildingId}
    AND estado = 'disponible'
`;
console.log(plan);
```

Checklist de Optimización

Antes de enviar a producción, verificar:

- [ ] ¿Usa `select` específico en lugar de fetch completo?
- [ ] ¿Limita resultados con `take` cuando sea apropiado?
- [ ] ¿Usa agregaciones en lugar de cálculo en JS?
- [ ] ¿Evita N+1 queries con `include` / `select` apropiados?
- [ ] ¿Usa paginación cursor-based para grandes datasets?
- [ ] ¿Las relaciones cargadas son realmente necesarias?
- [ ] ¿Hay índices para los campos en `where` y `orderBy`?
- [ ] ¿Se logean queries lentas para monitoreo?

Referencias

- Prisma Performance Guide (https://www.prisma.io/docs/guides/performance-and-optimization)
- PostgreSQL Query Performance (https://www.postgresql.org/docs/current/performance-tips.html)
- Database Indexes Optimization (./DATABASE_INDEXES_OPTIMIZATION.md)