

Uso de Campos JSON en MySQL

Resumen

Con la migración a MySQL, los campos que antes eran arrays (`String[]`) ahora son de tipo `Json` en el esquema de Prisma. Sin embargo, **el código TypeScript sigue funcionando igual** gracias a la conversión automática de Prisma.

Campos Afectados

Los siguientes campos ahora son de tipo `Json` en la base de datos, pero se siguen usando como arrays en el código:

Modelo	Campo	Tipo en Código	Tipo en DB
Deporte	categorias	<code>string[]</code>	<code>Json</code>
Escuela	certificaciones	<code>string[]</code>	<code>Json</code>
Instructor	certificaciones	<code>string[]</code>	<code>Json</code>
Instructor	especialidades	<code>string[]</code>	<code>Json</code>
InstructorDeporte	certificaciones	<code>string[]</code>	<code>Json</code>
Actividad	diasSemana	<code>string[]</code>	<code>Json</code>
Inscripcion	diasPreferidos	<code>string[]</code>	<code>Json</code>

✓ Código Correcto (No Necesita Cambios)

Crear con Arrays

```
// ✓ Puedes pasar arrays directamente
const deporte = await prisma.deporte.create({
  data: {
    nombre: "Fútbol",
    categorias: ["Futsal", "11", "Playa"], // Array normal de JavaScript
    activo: true
  }
});

// ✓ Crear instructor con múltiples arrays
const instructor = await prisma.instructor.create({
  data: {
    nombre: "Juan",
    apellido: "Pérez",
    certificaciones: ["FIFA Level 1", "UEFA B"], // Array
    especialidades: ["Fútbol juvenil", "Preparación física"], // Array
    provinciaId: "abc123",
    comunaId: "def456"
  }
});

// ✓ Crear actividad con días de la semana
const actividad = await prisma.actividad.create({
  data: {
    nombre: "Fútbol Sub-12",
    deporteId: "xyz789",
    diasSemana: ["lunes", "miércoles", "viernes"], // Array
    horaInicio: "16:00",
    horaFin: "18:00"
  }
});
```

Leer y Usar como Arrays

```
// ✔ Los datos vienen como arrays normales
const deportes = await prisma.deporte.findMany();

deportes.forEach(deporte => {
  console.log(deporte.nombre);

  // ✔ Puedes usar métodos de array directamente
  deporte.categorias.forEach(cat => {
    console.log(` - ${cat}`);
  });

  // ✔ También funcionan map, filter, etc.
  const categoriasUppercase = deporte.categorias.map(c => c.toUpperCase());
});

// ✔ Mapear directamente en React
{deportes.map(deporte => (
  <div key={deporte.id}>
    <h3>{deporte.nombre}</h3>
    <ul>
      {deporte.categorias.map(cat => (
        <li key={cat}>{cat}</li>
      ))}
    </ul>
  </div>
))}
```

Actualizar Arrays

```
// ✔ Actualizar reemplazando todo el array
await prisma.deporte.update({
  where: { id: deporteId },
  data: {
    categorias: ["Nueva Cat 1", "Nueva Cat 2"] // Array completo nuevo
  }
});

// ✔ Agregar elemento a un array existente
const deporte = await prisma.deporte.findUnique({ where: { id: deporteId } });
const nuevasCategorias = [...deporte.categorias, "Nueva Categoría"];

await prisma.deporte.update({
  where: { id: deporteId },
  data: {
    categorias: nuevasCategorias
  }
});

// ✔ Eliminar elemento de un array
const deporte = await prisma.deporte.findUnique({ where: { id: deporteId } });
const categoriasFiltradas = deporte.categorias.filter(c => c !== "Categoría a eliminar");

await prisma.deporte.update({
  where: { id: deporteId },
  data: {
    categorias: categoriasFiltradas
  }
});
```

Validación con Zod

```
// ✔ Zod también sigue funcionando igual
const actividadSchema = z.object({
  nombre: z.string(),
  diasSemana: z.array(z.string()).min(1, "Debe seleccionar al menos un día"),
  // ... otros campos
});

// Los datos validados son arrays normales
const data = actividadSchema.parse(input);
console.log(data.diasSemana); // ["lunes", "miércoles"]
```

TypeScript Tipos

Para TypeScript, puedes usar los tipos generados automáticamente por Prisma:

```
import { Deporte, Instructor, Actividad } from '@prisma/client';

// ✅ Los tipos incluyen los campos como arrays
const deporte: Deporte = {
  id: "123",
  nombre: "Fútbol",
  categorias: ["Futsal", "11"], // TypeScript reconoce esto como string[]
  activo: true,
  createdAt: new Date(),
  updatedAt: new Date()
};

// ✅ También puedes usar Prisma.JsonValue si necesitas ser explícito
import { Prisma } from '@prisma/client';

type CategoriasType = Prisma.JsonValue; // Acepta arrays, objects, primitivos
```

⚠ Limitaciones de MySQL con JSON

A diferencia de PostgreSQL que tiene soporte nativo para arrays, MySQL usa JSON. Esto significa:

❌ No Puedes Filtrar Arrays Directamente

```
// ❌ ESTO NO FUNCIONA en MySQL como en PostgreSQL
const deportes = await prisma.deporte.findMany({
  where: {
    categorias: {
      has: "Futsal" // ❌ No existe en MySQL
    }
  }
});
```

✅ Alternativas para Búsquedas

Opción 1: Filtrar en JavaScript después de obtener los datos

```
// ✅ Obtener todos y filtrar en memoria
const todosLosDeportes = await prisma.deporte.findMany();
const deportesConFutsal = todosLosDeportes.filter(d =>
  d.categorias.includes("Futsal")
);
```

Opción 2: Usar búsqueda de texto en JSON (MySQL 5.7+)

```
// ✅ Buscar en el texto JSON (menos eficiente pero funciona)
const deportes = await prisma.$queryRaw`
  SELECT * FROM Deporte
  WHERE JSON_SEARCH(categorias, 'one', 'Futsal') IS NOT NULL
`;
```

Opción 3: Crear tabla de relaciones (recomendado para búsquedas frecuentes)

Si necesitas buscar frecuentemente por categorías, considera crear una tabla intermedia:

```

model DeporteCategoría {
  id          String @id @default(cuid())
  deporteId   String
  categoria   String
  deporte     Deporte @relation(fields: [deporteId], references: [id])

  @@unique([deporteId, categoria])
  @@index([categoria])
}

```

Debugging

Si encuentras problemas, verifica que los datos estén en formato correcto:

```

// Verificar tipo de dato
const deporte = await prisma.deporte.findUnique({
  where: { id: deporteId }
});

console.log('Tipo:', typeof deporte.categorias); // "object"
console.log('Es Array?', Array.isArray(deporte.categorias)); // true
console.log('Contenido:', JSON.stringify(deporte.categorias));

// Si algo sale mal, puedes forzar la conversión
const categorias = Array.isArray(deporte.categorias)
  ? deporte.categorias
  : [];

```

Conclusión

No necesitas cambiar tu código existente. Prisma maneja automáticamente la conversión entre JSON (MySQL) y arrays de JavaScript. Simplemente usa los arrays como siempre lo has hecho en tu código TypeScript.

La única diferencia que notarás es en búsquedas complejas dentro de arrays, donde MySQL tiene limitaciones comparado con PostgreSQL.