

Cargas asíncronas, datos relacionados, filtrado y rendimiento

Guía de la lección · Guía del laboratorio / examen

● LearnWisej

Guía de la lección

La tabla de decisión para datos relacionados (proyección, Include, Include filtrado, carga explícita, nada de carga diferida en cuadrículas), con seguimiento frente a sin seguimiento, paginación y filtros indexados, coordinación asíncrona con estados de carga y tareas en segundo plano, y medir antes de optimizar. Qué significa esto para un desarrollador de Wisej.NET que trabaja con bases de datos reales, y cómo leer este módulo.



Enfoque de producción

La tabla de decisión para datos relacionados (proyección, Include, Include filtrado, carga explícita, nada de carga diferida en las cuadrículas), con seguimiento frente a sin seguimiento, paginación y filtros indexados, coordinación asíncrona con estados de carga y tareas en segundo plano, y medir antes de optimizar.

1. El problema en producción

El explorador de tickets de Support Desk funcionaba en el Módulo 3 con cincuenta tickets de ejemplo. Con cincuenta mil no funcionará, a menos que la consulta haga menos trabajo y en el lugar adecuado. El fallo es silencioso: una cuadrícula que tarda tres segundos porque el código de formato tocó una propiedad de navegación por fila, una sesión que retiene miles de entidades con seguimiento que nadie va a editar, una búsqueda que recorre una tabla entera. Este módulo trata de hacer menos trabajo, en SQL, a propósito, y de demostrarlo con un registro en lugar de con una corazonada.

2. Conceptos clave

Concepto	Por qué importa
proyección	Un <code>Select</code> que devuelve exactamente los campos que necesita la pantalla, como <code>TicketListItem</code> , para que la base de datos combine los nombres relacionados en una sola sentencia.
Include e Include filtrado	Carga anticipada de las entidades relacionadas junto con la raíz; un <code>Include</code> filtrado acota una colección (los últimos cinco comentarios) para que un editor cargue un agregado controlado.
carga explícita	Cargar una navegación más tarde, bajo demanda, mediante <code>Collection</code> o <code>Reference</code> de la entrada y <code>LoadAsync</code> , cuando el usuario despliega una sección de detalle.
carga diferida y N+1	Propiedades de navegación que se cargan de forma transparente en el primer acceso; en una cuadrícula eso supone una consulta oculta por fila o por acceso a navegación.
AsNoTracking y resolución de identidad	Resultados que evitan el change tracker en las pantallas de solo lectura; <code>AsNoTrackingWithIdentityResolution</code> sigue devolviendo una única instancia por entidad cuando la misma fila aparece varias veces.
paginación en la consulta	<code>Where</code> , <code>OrderBy</code> , <code>Skip</code> , <code>Take</code> y <code>Select</code> compuestos antes de <code>ToListAsync</code> , con un <code>CountAsync</code> aparte para el total.
registro de EF Core	El SQL generado de cada comando junto con su duración, mediante <code>LogTo</code> o <code>Microsoft.Extensions.Logging</code> .

3. Desarrollo de la lección

De dónde sale la cuadrícula lenta

Empieza por el explorador de tickets deliberadamente lento que el laboratorio mide antes de corregirlo. Su `SearchTicketsAsync` carga entidades `Ticket` con seguimiento y sin paginación, y `CellFormatting` lee `ticket.Customer.Name`, `ticket.Agent?.DisplayName` y `ticket.Category.Name`. Con la carga diferida activada, cada primer acceso a una navegación no cargada ejecuta su propia consulta: cincuenta filas cuestan una sentencia para los tickets más ciento cincuenta para los nombres. No se lanza ninguna excepción; solo el registro lo revela.

La regla que lo elimina: el código de visualización nunca debe provocar una consulta. Las filas enlazadas a un `DataGridView` de Wisej.NET a través del `BindingSource` son records `TicketListItem` que ya traen los nombres que muestran, y `CellFormatting` solo da formato. La misma regla rige las pantallas de árbol y de lista: un `TreeView` carga los hijos de un nodo una sola vez, cuando se expande, nunca recorriendo propiedades de navegación por nodo.



Error común: carga diferida desde el formato de filas o el código de visualización

Una propiedad de navegación leída dentro de `CellFormatting` o del código que dibuja un nodo del árbol es una consulta disfrazada que se ejecuta una vez por fila. Incluye los nombres en la proyección.

La tabla de decisión para datos relacionados

EF Core ofrece cuatro formas de llevar datos relacionados a una pantalla; la elección depende de lo que la pantalla vaya a hacer con ellos.

- Usa la **proyección** cuando la interfaz necesite un modelo de lectura aplanado: el explorador de tickets, los dashboards, las exportaciones.
- Usa **Include** cuando un editor necesite un agregado completo y su tamaño esté controlado: un ticket con su cliente, su categoría y su agente.
- Usa el **Include filtrado** cuando solo se necesite un pequeño subconjunto de una colección: los últimos cinco comentarios públicos.
- Usa la **carga explícita** cuando el usuario despliegue una sección de detalle bajo demanda: el historial de comentarios tras un panel contraído.
- No confíes nunca en la **carga diferida** en cuadrículas, listas o árboles: produce en silencio una consulta por fila o por acceso a navegación.

En Support Desk el explorador proyecta a `TicketListItem`; el `LoadEditorAsync` del editor usa `Include` para las tres referencias y un `Include` filtrado para los comentarios que previsualiza; el panel de comentarios carga el historial completo con `db.Entry(ticket).Collection(t => t.Comments).LoadAsync()` dentro del propio Context del editor.

Con seguimiento frente a sin seguimiento

El seguimiento es lo correcto cuando el mismo Context va a guardar cambios en las mismas instancias de entidad, como hace el editor cuando `SaveChangesAsync` escribe la diferencia. Sin seguimiento es lo correcto para cuadrículas de solo lectura, dashboards, listas de consulta, exportaciones y resultados de autocompletado. `AsNoTracking` se salta el change tracker: menos memoria por sesión, materialización más rápida y ninguna actualización accidental desde una pantalla que solo lee.

`AsNoTrackingWithIdentityResolution` queda entre ambos. Una consulta sin seguimiento que devuelve entidades puede entregar un objeto `Customer` distinto por cada ticket del mismo cliente; la resolución de identidad devuelve una única instancia por clave. Úsala cuando una pantalla de solo lectura construya un grafo de objetos; prescinde de ella cuando proyectes a filas planas.



Error común: usar entidades sin seguimiento y esperar después la detección automática de cambios

Editar un `Ticket` obtenido de una consulta `AsNoTracking`, o un `TicketListItem` de la cuadrícula, y luego llamar a `SaveChangesAsync` no cambia nada. El editor carga por clave una entidad nueva con seguimiento en su propio Context y copia sobre ella los valores aprobados.

Paginación y filtros indexados en la consulta

Una cuadrícula de Wisej.NET no debería cargar 100.000 filas solo porque pueda mostrarlas. Compón los filtros, la ordenación, `Skip`, `Take` y `Select` en la consulta para que el trabajo lo haga la base de datos, y devuelve un recuento total cuando la interfaz muestre números de página, mediante un `CountAsync` aparte sobre la consulta filtrada:

```

var total = await query.CountAsync(token);

var page = await query
    .AsNoTracking()
    .OrderByDescending(t => t.UpdatedAt)
    .Skip(criteria.PageIndex * 50)
    .Take(50)
    .Select(t => new TicketListItem(
        t.Id, t.Number, t.Title, t.Customer.Name,
        t.Agent == null ? null : t.Agent.DisplayName,
        t.Category.Name, t.Status, t.Priority, t.DueDate, t.UpdatedAt))
    .ToListAsync(token);

```

Los índices del Módulo 2 sobre `Status`, `DueDate`, `CustomerId` y `UpdatedAt` coinciden con estos filtros y con esta ordenación. Mantén la búsqueda de texto compatible con los índices: un `StartsWith` sobre el número de ticket se convierte en un `LIKE` por prefijo que un índice puede atender, mientras que un `Contains` sobre el título se convierte en un comodín inicial que recorre la tabla. Cuando el registro muestre un filtro frecuente sin índice, añade uno y vuelve a medir.

Coordinación asíncrona con estados de carga y tareas en segundo plano

EF Core ofrece métodos asíncronos para cada operación de E/S. Espera cada uno con `await` antes de iniciar el siguiente sobre el mismo Context, porque un `DbContext` no admite operaciones en paralelo; los dos `await` de arriba se ejecutan en secuencia por diseño. En Wisej.NET un controlador de eventos asíncrono devuelve el control al navegador mientras se ejecuta la consulta, así que la protección de carga importa: muestra un estado de carga con `ShowLoader` de la cuadrícula o con un `Label` de estado, deshabilita los botones Search, Next y Previous que iniciarían trabajo concurrente y vuelve a habilitarlos en un bloque `finally`.

Los trabajos largos de importación, exportación e informes no deben ir en un controlador de eventos. Inicialos con `Application.StartTask`, crea el `DbContext` dentro de la tarea a partir de la factoría en lugar de capturar el Context o las entidades de la página, y envía el progreso solo mediante `Application.Update`, el mecanismo de actualización de Wisej.NET que conoce el contexto de sesión. Un hilo sin más que toca un control no tiene contexto de sesión, y el token de cancelación que le pasas es la forma de detener el trabajo cuando termina la sesión.

Mide antes de optimizar

No adivines. Activa el registro de EF Core en desarrollo con `LogTo` o `Microsoft.Extensions.Logging`, lee el SQL y la duración de cada comando, y activa `EnableSensitiveDataLogging` solo en una máquina de desarrollo, ya que imprime los valores de los parámetros. Para una sentencia lenta, examina el plan de ejecución de la base de datos, donde se ven un índice que falta o un recorrido por comodín. El registro de antes y después del laboratorio es esta disciplina llevada a la práctica.

Solo cuando el perfilado muestra el cuello de botella se justifica la complejidad de las herramientas especializadas. Las consultas compiladas ayudan cuando el coste de compilación es medible en una consulta muy frecuente. El pooling de Contexts con `AddPooledDbContextFactory` ayuda cuando domina la construcción del Context, y requiere cuidado cuando el estado del Context varía por usuario o por

tenant, ya que un Context del pool se reinicia y se reutiliza. Las consultas divididas ayudan cuando varios **Include** de colecciones multiplican las filas. El SQL directo es el último recurso. Cada una responde a un número del registro, nunca es la opción por defecto.



Error común: optimizar sin medir

Las consultas compiladas, el pooling o el SQL directo en una pantalla sin perfilar cambian legibilidad por una mejora que quizá no exista. Registra, cambia una sola cosa y vuelve a registrar.



Regla de enlace de datos

Mantén separados los dos ciclos de vida. Los formularios de Wisej.NET, los BindingSource y la fila seleccionada viven lo que dura la sesión; un DbContext vive lo que dura una operación. Créalo desde la factoría, consulta o guarda, deséchalo y no dejes nunca que se convierta en estado de sesión.



Objetivos de aprendizaje

Al terminar este módulo serás capaz de:

- ✓ Evitar el comportamiento N+1 de la carga diferida en cuadrículas y en pantallas de árbol o lista.
- ✓ Usar Include, Include filtrado, carga explícita y proyección de forma deliberada.
- ✓ Usar AsNoTracking, AsNoTrackingWithIdentityResolution, paginación, índices y mejoras de rendimiento medidas.
- ✓ Coordinar las operaciones asíncronas de EF Core con los indicadores de carga y las tareas en segundo plano de Wisej.NET.

Guía del laboratorio / examen

Qué construirás en el laboratorio práctico, el enfoque sugerido y los entregables requeridos.



Laboratorio práctico

Optimiza el explorador de tickets de Support Desk: activa el registro de EF Core en desarrollo, anota el SQL y los tiempos de la búsqueda sin optimizar, sustituye la carga de entidades por una proyección, añade AsNoTracking a las consultas de la cuadrícula, añade paginación con un tamaño de página de 50, sustituye el acceso diferido a navegaciones por proyección o Include, y documenta en tus notas del laboratorio los resultados medidos antes y después.

Punto de partida

Abre la solución Support Desk Data Console del Módulo 5 y la página del explorador de tickets del Módulo 3. Para este laboratorio parte de una versión deliberadamente ingenua de

`TicketQueryService.SearchTicketsAsync`: una que carga entidades `Ticket` con seguimiento y sin paginación y deja que el `CellFormatting` de la cuadrícula lea los nombres del cliente, del agente y de la categoría a través de propiedades de navegación, de modo que haya un N+1 real que medir antes de eliminarlo.

Enfoque sugerido

- ✓ Activa el registro de EF Core en desarrollo con `LogTo` o `Microsoft.Extensions.Logging` en `SupportDeskContext`, deja `EnableSensitiveDataLogging` detrás de una comprobación de entorno de desarrollo y ejecuta la búsqueda ingenua una vez; cuenta las sentencias, anota los milisegundos y pega el SQL en tus notas del laboratorio.
- ✓ Reescribe la consulta de la cuadrícula como una proyección sin seguimiento: `AsNoTracking`, los filtros `Where` compuestos, `OrderByDescending(t => t.UpdatedAt)` y después `Select` a `TicketListItem` con los nombres del cliente, del agente y de la categoría combinados por la base de datos, terminando en `ToListAsync`.
- ✓ Añade paginación con un tamaño de página de 50: un `CountAsync` sobre la consulta filtrada para el total y luego `Skip` y `Take` antes de la proyección; asegúrate de que todos los filtros y la ordenación usen los índices sobre `Status`, `DueDate`, `CustomerId` y `UpdatedAt`, y mantén la búsqueda por número de ticket como coincidencia por prefijo.
- ✓ Elimina todo acceso a navegaciones del código de visualización: `CellFormatting` solo da formato a los campos que ya están en la fila, y cualquier dato relacionado que necesite el editor se carga con `Include`, un `Include` filtrado o carga explícita dentro del propio Context del editor.
- ✓ Mantén rigurosa la protección asíncrona: espera con `await` cada operación de base de datos antes de la siguiente sobre el mismo Context, muestra el estado de carga y deshabilita Search, Next Page y Previous Page mientras se carga una página, vuelve a habilitarlos en `finally`, y después ejecuta la búsqueda optimizada y anota las cifras de después junto a las de antes.

Entregables requeridos

- ✓ Registro de EF Core activado en desarrollo, con el SQL y los tiempos sin optimizar anotados
- ✓ Consulta de la cuadrícula reescrita como proyección sin seguimiento
- ✓ Paginación con un tamaño de página de 50 y filtros indexados
- ✓ Ningún comportamiento N+1 de carga diferida en el código de visualización
- ✓ Notas del laboratorio que documenten la mejora medida antes y después

Preguntas de repaso

- ✓ Tu registro mostró 151 sentencias para una pantalla de 50 tickets. ¿Qué línea de código provocó las 150 adicionales, y por qué nada en la aplicación te avisó?
- ✓ El editor necesita el ticket, su cliente y su categoría, y los últimos cinco comentarios; el panel de comentarios abre el historial completo bajo demanda. ¿Qué estrategia de carga atiende cada una de esas tres necesidades, y por qué la proyección es la respuesta equivocada para el editor?
- ✓ Un compañero de equipo propone consultas compiladas y pooling de Contexts para el explorador de tickets antes de mirar el registro. ¿Qué medirías primero, y qué tendrían que mostrar las cifras para que cualquiera de los dos cambios merezca la pena?



Regla de enlace de datos

Mantén separados los dos ciclos de vida. Los formularios de Wisej.NET, los BindingSource y la fila seleccionada viven lo que dura la sesión; un DbContext vive lo que dura una operación. Créalo desde la factoría, consulta o guarda, deséchalo y no dejes nunca que se convierta en estado de sesión.