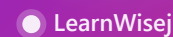


Caricamenti asincroni, dati correlati, filtri e prestazioni

Guida della lezione · Guida al lab / esame



Guida della lezione

La tabella di decisione per i dati correlati (proiezione, Include, Include filtrato, caricamento esplicito, nessun lazy loading nelle griglie), tracking contro no-tracking, paginazione e filtri indicizzati, coordinamento asincrono con stati di caricamento e attività in background, e misurare prima di ottimizzare. Che cosa significa per uno sviluppatore Wisej.NET che lavora con database reali — e come affrontare questo modulo.



Focus di produzione

La tabella di decisione per i dati correlati (proiezione, Include, Include filtrato, caricamento esplicito, nessun lazy loading nelle griglie), tracking contro no-tracking, paginazione e filtri indicizzati, coordinamento asincrono con stati di caricamento e attività in background, e misurare prima di ottimizzare.

1. Il problema in produzione

Il browser dei ticket di Support Desk funzionava nel Modulo 3 con cinquanta ticket di prova. Con cinquantamila non funzionerà, a meno che la query non faccia meno lavoro e nel posto giusto. Il guasto è silenzioso: una griglia che impiega tre secondi perché il codice di formattazione ha toccato una proprietà di navigazione per ogni riga, una sessione che tiene migliaia di entità tracciate che nessuno modificherà, una ricerca che scansiona una tabella. Questo modulo parla di fare meno lavoro, in SQL, di proposito, e di dimostrarlo con un log invece che con un'impressione.

2. Concetti chiave

Concetto	Perché è importante
proiezione	Una <code>Select</code> che restituisce esattamente i campi che servono alla schermata, come <code>TicketListItem</code> , così il database unisce i nomi correlati in un'unica istruzione.
Include e Include filtrato	Caricamento eager delle entità correlate insieme alla radice; un <code>Include</code> filtrato restringe una collezione (gli ultimi cinque commenti) così un editor carica un aggregato controllato.
caricamento esplicito	Caricare una navigazione più tardi, su richiesta, tramite <code>Collection</code> o <code>Reference</code> dell'entry e <code>LoadAsync</code> , quando l'utente espande una sezione di dettaglio.
lazy loading e N+1	Proprietà di navigazione caricate in modo trasparente al primo accesso; in una griglia significa una query nascosta per ogni riga o per ogni accesso a una navigazione.
AsNoTracking e identity resolution	Risultati che scavalcano il change tracker per le schermate di sola lettura; <code>AsNoTrackingWithIdentityResolution</code> restituisce comunque una sola istanza per entità quando la stessa riga compare più volte.
paginazione a livello di query	<code>Where</code> , <code>OrderBy</code> , <code>Skip</code> , <code>Take</code> e <code>Select</code> composti prima di <code>ToListAsync</code> , con un <code>CountAsync</code> separato per il totale.
logging di EF Core	L'SQL generato di ogni comando con la sua durata, tramite <code>LogTo</code> o <code>Microsoft.Extensions.Logging</code> .

3. Il filo della lezione

Da dove nasce la griglia lenta

Parti dal browser dei ticket volutamente lento che il lab misura prima di correggerlo. Il suo `SearchTicketsAsync` carica entità `Ticket` tracciate senza paginazione, e `CellFormatting` legge `ticket.Customer.Name`, `ticket.Agent?.DisplayName` e `ticket.Category.Name`. Con il lazy loading attivo, ogni primo accesso a una navigazione non caricata esegue una propria query: cinquanta righe costano un'istruzione per i ticket più centocinquanta per i nomi. Nessuna eccezione; lo mostra solo il log.

La regola che lo elimina: il codice di visualizzazione non deve mai causare una query. Le righe collegate a una `DataGridView` Wisej.NET tramite la `BindingSource` sono record `TicketListItem` che portano già i nomi che mostrano, e `CellFormatting` si limita a formattare. La stessa regola governa le schermate ad albero e a lista: una `TreeView` carica i figli di un nodo una volta sola, quando viene espanso, mai percorrendo le proprietà di navigazione nodo per nodo.



Errore tipico: lazy loading dalla formattazione delle righe o dal codice di visualizzazione

Una proprietà di navigazione letta dentro `CellFormatting` o dal disegno di un nodo di albero è una query travestita che gira una volta per riga. Porta invece i nomi nella proiezione.

La tabella di decisione per i dati correlati

EF Core offre quattro modi per portare dati correlati in una schermata; la scelta dipende da che cosa la schermata ne farà.

- Usa la **proiezione** quando l'interfaccia ha bisogno di un modello di lettura appiattito: il browser dei ticket, le dashboard, le esportazioni.
- Usa **Include** quando un editor ha bisogno di un aggregato completo e la sua dimensione è controllata: un ticket con il suo cliente, la sua categoria e il suo agente.
- Usa l'**Include filtrato** quando serve solo un piccolo sottoinsieme di una collezione: gli ultimi cinque commenti pubblici.
- Usa il **caricamento esplicito** quando l'utente espande su richiesta una sezione di dettaglio: lo storico dei commenti dietro un pannello chiuso.
- Non affidarti mai al **lazy loading** in griglie, liste o alberi: produce in silenzio una query per riga o per accesso a una navigazione.

In Support Desk il browser proietta in `TicketListItem`; il `LoadEditorAsync` dell'editor usa `Include` per i tre riferimenti e un `Include` filtrato per i commenti che mostra in anteprima; il pannello dei commenti carica lo storico completo con `db.Entry(ticket).Collection(t => t.Comments).LoadAsync()` dentro il Context dell'editor stesso.

Tracking contro no-tracking

Il tracking è corretto quando lo stesso Context salverà le modifiche sulle stesse istanze di entità, come fa l'editor quando `SaveChangesAsync` scrive la differenza. Il no-tracking è corretto per griglie di sola lettura, dashboard, liste di lookup, esportazioni e risultati di autocompletamento. `AsNoTracking` salta il change tracker: meno memoria per sessione, materializzazione più veloce, nessun aggiornamento accidentale da una schermata che si limita a leggere.

`AsNoTrackingWithIdentityResolution` sta a metà strada. Una query senza tracking che restituisce entità può riconsegnare un oggetto `Customer` separato per ogni ticket dello stesso cliente; la identity resolution restituisce un'istanza per chiave. Usala quando una schermata di sola lettura costruisce un grafo di oggetti; evitala quando proietti in righe piatte.



Errore tipico: usare entità senza tracking e aspettarsi poi il rilevamento automatico degli aggiornamenti

Modificare un `Ticket` ottenuto da una query `AsNoTracking`, o un `TicketListItem` della griglia, e poi chiamare `SaveChangesAsync` non cambia nulla. L'editor carica per chiave un'entità tracciata nuova nel proprio Context e vi mappa sopra i valori approvati.

Paginazione e filtri indicizzati a livello di query

Una griglia Wisej.NET non dovrebbe caricare 100.000 righe solo perché è in grado di mostrarle. Componi filtri, ordinamento, `Skip`, `Take` e `Select` a livello di query così il lavoro lo fa il database, e restituisci un conteggio totale quando l'interfaccia mostra i numeri di pagina, da un `CountAsync` separato sulla query filtrata:

```

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);

```

Gli indici del Modulo 2 su `Status`, `DueDate`, `CustomerId` e `UpdatedAt` corrispondono a questi filtri e a questo ordinamento. Mantieni la ricerca testuale adatta agli indici: uno `StartsWith` sul numero di ticket diventa un `LIKE` per prefisso che un indice può servire, mentre un `Contains` sul titolo diventa un carattere jolly iniziale che scansiona. Quando il log mostra un filtro frequente senza indice, aggiungine uno e misura di nuovo.

Coordinamento asincrono con stati di caricamento e attività in background

EF Core offre metodi asincroni per ogni operazione di I/O. Attendi ciascuno prima di avviare il successivo sullo stesso Context, perché un `DbContext` non supporta operazioni parallele; i due `await` qui sopra girano in sequenza per progetto. In Wisej.NET un gestore di eventi async restituisce il controllo al browser mentre la query è in corso, quindi la protezione di caricamento conta: mostra uno stato di caricamento con `ShowLoader` della griglia o con una `Label` di stato, disabilita i pulsanti Search, Next e Previous che avvierebbero lavoro concorrente, e riabilitali in un blocco `finally`.

I job di importazione, esportazione e reportistica di lunga durata non appartengono a un gestore di eventi. Avviali con `Application.StartTask`, crea il `DbContext` dentro l'attività a partire dalla factory invece di catturare il Context o le entità della pagina, e spingi l'avanzamento solo tramite `Application.Update`, il meccanismo di aggiornamento di Wisej.NET consapevole del Context. Un thread grezzo che tocca un controllo non ha un contesto di sessione, e il cancellation token che passi è il modo in cui un job si ferma quando la sessione termina.

Misura prima di ottimizzare

Non tirare a indovinare. Abilita il logging di EF Core in sviluppo con `LogTo` o `Microsoft.Extensions.Logging`, leggi l'SQL e la durata di ogni comando, e attiva `EnableSensitiveDataLogging` solo sulla macchina di uno sviluppatore, dato che stampa i valori dei parametri. Per un'istruzione lenta, esamina il piano di esecuzione del database, dove emergono un indice mancante o una scansione da carattere jolly. Il log prima-e-dopo del lab è questa disciplina resa concreta.

Solo dopo che la profilazione ha mostrato il collo di bottiglia gli strumenti specializzati valgono la loro complessità. Le query compilate aiutano quando il costo di compilazione è misurabile su una query molto usata. Il pooling dei Context con `AddPooledDbContextFactory` aiuta quando domina la costruzione del Context, e richiede attenzione quando lo stato del Context varia per utente o tenant, perché un Context in pool viene azzerato e riutilizzato. Le split query aiutano quando più `Include` di collezioni moltiplicano le

righe. L'SQL grezzo è l'ultima risorsa. Ognuno risponde a un numero nel log, mai a un'impostazione predefinita.



Errore tipico: ottimizzare senza misurare

Query compile, pooling o SQL grezzo su una schermata non profilata scambiano leggibilità con un miglioramento che potrebbe non esistere. Registra, cambia una cosa, registra di nuovo.



La regola del data binding

Tieni separati i due cicli di vita. Form Wisej.NET, BindingSource e riga selezionata vivono per la sessione; un DbContext vive per un'operazione. Crealo dalla factory, interroga o salva, eliminalo, e non lasciarlo mai diventare stato di sessione.



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Evitare il comportamento N+1 da lazy loading nelle griglie e nelle schermate ad albero o a lista.
- ✓ Usare in modo intenzionale Include, Include filtrato, caricamento esplicito e proiezione.
- ✓ Usare AsNoTracking, AsNoTrackingWithIdentityResolution, paginazione, indici e miglioramenti di prestazioni misurati.
- ✓ Coordinare le operazioni asincrone di EF Core con gli indicatori di caricamento e le attività in background di Wisej.NET.

Guida al lab / esame

Che cosa costruirai nel lab pratico, l'approccio suggerito e i deliverable richiesti.



Lab pratico

Ottimizza il browser dei ticket di Support Desk: abilita il logging di EF Core in sviluppo, registra l'SQL e i tempi della ricerca non ottimizzata, sostituisci il caricamento delle entità con una proiezione, aggiungi `AsNoTracking` alle query delle griglie, aggiungi la paginazione con una dimensione di pagina di 50, sostituisci l'accesso lazy alle navigazioni con una proiezione o con `Include`, e documenta nelle note del lab i risultati misurati prima e dopo.

Punto di partenza

Apri la soluzione Support Desk Data Console del Modulo 5 e la pagina del browser dei ticket del Modulo 3. Per questo lab parti da un ramo volutamente ingenuo di `TicketQueryService.SearchTicketsAsync`: uno che carica entità `Ticket` tracciate senza paginazione e lascia che il `CellFormatting` della griglia legga i nomi di cliente, agente e categoria tramite le proprietà di navigazione, così c'è un vero N+1 da misurare prima di eliminarlo.

Approccio suggerito

- ✓ Abilita il logging di EF Core in sviluppo con `LogTo` o `Microsoft.Extensions.Logging` su `SupportDeskContext`, tieni `EnableSensitiveDataLogging` dietro un controllo sullo sviluppo ed esegui una volta la ricerca ingenua; conta le istruzioni, annota i millisecondi e incolla l'SQL nelle note del lab.
- ✓ Riscrivi la query della griglia come proiezione senza tracking: `AsNoTracking`, i filtri `Where` composti, `OrderByDescending(t => t.UpdatedAt)`, poi `Select` in `TicketListItem` con i nomi di cliente, agente e categoria uniti dal database, terminando con `ToListAsync`.
- ✓ Aggiungi la paginazione con una dimensione di pagina di 50: un `CountAsync` sulla query filtrata per il totale, poi `Skip` e `Take` prima della proiezione; assicurati che ogni filtro e l'ordinamento usino gli indici su `Status`, `DueDate`, `CustomerId` e `UpdatedAt`, e mantieni la ricerca per numero di ticket come corrispondenza di prefisso.
- ✓ Rimuovi ogni accesso a una navigazione dal codice di visualizzazione: `CellFormatting` formatta solo i campi già presenti nella riga, e i dati correlati di cui l'editor ha bisogno vengono caricati con `Include`, con un `Include` filtrato o con il caricamento esplicito dentro il Context dell'editor stesso.
- ✓ Mantieni onesta la protezione async: attendi ogni operazione sul database prima della successiva sullo stesso Context, mostra lo stato di caricamento e disabilita Search, Next Page e Previous Page mentre una pagina si carica, riabilitali nel `finally`, poi esegui la ricerca ottimizzata e annota i numeri del dopo accanto a quelli del prima.

Deliverable richiesti

- ✓ Logging di EF Core abilitato in sviluppo con l'SQL e i tempi non ottimizzati registrati

- ✓ Query della griglia riscritta come proiezione senza tracking
- ✓ Paginazione con dimensione di pagina di 50 e filtri indicizzati
- ✓ Nessun comportamento N+1 da lazy loading rimasto nel codice di visualizzazione
- ✓ Note del lab che documentano il miglioramento misurato prima e dopo

Domande di revisione per lo studente

- ✓ Il tuo log ha mostrato 151 istruzioni per una schermata da 50 ticket. Quale riga di codice ha causato le 150 in più, e perché niente nell'applicazione ti ha avvisato?
- ✓ L'editor ha bisogno del ticket, del suo cliente e della sua categoria, e degli ultimi cinque commenti; il pannello dei commenti apre lo storico completo su richiesta. Quale strategia di caricamento serve ciascuna di queste tre esigenze, e perché la proiezione è la risposta sbagliata per l'editor?
- ✓ Un collega propone query compilate e pooling dei Context per il browser dei ticket prima di guardare il log. Che cosa misureresti per primo, e che cosa dovrebbero dire i numeri perché valga la pena fare una delle due modifiche?



La regola del data binding

Tieni separati i due cicli di vita. Form Wisej.NET, BindingSource e riga selezionata vivono per la sessione; un DbContext vive per un'operazione. Crealo dalla factory, interroga o salva, eliminalo, e non lasciarlo mai diventare stato di sessione.