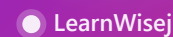


Modellazione, configurazione del DbContext e migrazioni

Guida della lezione · Guida al lab / esame



Guida della lezione

Entità pensate per schermate con data binding, Fluent API per chiavi, lunghezze, indici, comportamento di eliminazione e RowVersion, migrazioni trattate come sorgente da rivedere con script e bundle, e la factory di design time. Che cosa significa per uno sviluppatore Wisej.NET che lavora con database reali — e come affrontare questo modulo.



Focus di produzione

Entità pensate per schermate con data binding, Fluent API per chiavi, lunghezze, indici, comportamento di eliminazione e RowVersion, migrazioni trattate come sorgente da rivedere con script e bundle, e la factory di design time.

1. Il problema in produzione

La Support Desk Data Console mostrerà i ticket in una griglia paginabile, li modificherà in un form modale e lascerà che più agenti lavorino contemporaneamente sulla stessa coda, e ogni schermata vale quanto il modello che ha sotto. Un filtro senza un indice corrispondente diventa una scansione completa della tabella a cinquantamila ticket, due agenti che salvano lo stesso ticket perdono in silenzio il lavoro di uno dei due a meno che la riga non porti un token di concorrenza, e uno schema corretto a mano rompe il rilascio successivo. Il modello è il contratto tra l'interfaccia e il database, quindi questo modulo lo costruisce in modo esplicito e lo versiona come codice sorgente.

2. Concetti chiave

Concetto	Perché è importante
entità adatta al binding	Un'entità con una chiave prevedibile, campi di visualizzazione stabili, proprietà di navigazione esplicite e piccole tabelle di lookup, così una griglia, una <code>ComboBox</code> e un editor possono tutti collegarsi a essa.
view model <code>TicketListItem</code>	Una proiezione piatta delle sole colonne che la griglia mostra, così la griglia non riceve mai l'intero grafo del <code>Ticket</code> e resta indipendente dalla unit of work di modifica.
configurazione con Fluent API	Regole di mapping scritte in <code>OnModelCreating</code> (stringhe obbligatorie, lunghezze, indici, comportamento di eliminazione, precisione, valori predefiniti) così le regole di persistenza vivono accanto al <code>DbContext</code> .
token di concorrenza <code>RowVersion</code>	Un array di byte generato dal database e marcato <code>IsRowVersion</code> ; EF Core aggiunge il suo valore originale alla clausola WHERE dell'UPDATE, così un salvataggio su dati obsoleti aggiorna zero righe e lancia <code>DbUpdateConcurrencyException</code> .
comportamento di eliminazione	La regola <code>OnDelete</code> di una relazione (Restrict, SetNull, Cascade) che decide se eliminare un cliente viene rifiutato, toglie l'assegnazione ai ticket o rimuove anche i commenti.
migrazione	Una classe C# più lo snapshot del modello che descrive una modifica di schema rivista, salvata nel controllo di versione e applicata in ordine per allineare qualsiasi database al modello.
factory di design time	Un <code>IDesignTimeDbContextFactory</code> che dice agli strumenti <code>dotnet ef</code> come costruire il Context senza eseguire lo startup dell'applicazione Wisej.NET.

3. Il filo della lezione

Entità pensate per schermate master-detail

Le interfacce con data binding funzionano meglio quando il modello ha chiavi prevedibili, campi di visualizzazione stabili, relazioni esplicite e piccole tabelle di lookup. Nella Support Desk Data Console l'entità `Ticket` porta `Customer`, `Agent`, `Category`, `Status`, `Priority`, `DueDate`, `Title`, `Description`, `RowVersion`, `CreatedAt` e `UpdatedAt`. `Customer`, `Agent` e `Category` sono piccole entità di lookup, ciascuna con una collezione `Tickets`, e `TicketComment` è la collezione di dettaglio sotto un ticket.

`CustomerId` e `CategoryId` sono interi obbligatori perché un ticket senza di essi non ha senso su nessuna schermata; `AgentId` è un `int?` nullable perché un nuovo ticket esiste prima che qualcuno venga assegnato, così la `ComboBox` dell'agente può mostrare "Unassigned" senza una riga sentinella. Una griglia non ha bisogno di ogni proprietà, quindi il corso definisce anche i view model `TicketListItem` (numero, titolo, cliente, stato, data di scadenza, agente, ultimo aggiornamento) per la griglia di sola lettura, mentre un editor si collega direttamente a un `Ticket` tracciato oppure a un modello di modifica rimappato sull'entità.

Prima la Fluent API per i modelli di produzione

Le convenzioni di EF Core individuano chiavi e chiavi esterne dai nomi, ma un'applicazione di produzione dovrebbe configurare esplicitamente tutto ciò che influisce su schema e comportamento: stringhe obbligatorie, lunghezze massime, gli indici usati dai filtri di ricerca, comportamento di eliminazione, precisione dei decimali, valori predefiniti e token di concorrenza. Le data annotation vanno bene per progetti semplici; la Fluent API tiene le regole di persistenza vicine al `DbContext` ed evita di mescolare la validazione dell'interfaccia con il mapping del database. Il cuore di

`SupportDeskContext.OnModelCreating` :

```
modelBuilder.Entity<Ticket>(b =>
{
    b.Property(x => x.Title).HasMaxLength(180).IsRequired();
    b.Property(x => x.Status).HasMaxLength(30).IsRequired();
    b.Property(x => x.RowVersion).IsRowVersion();
    b.HasIndex(x => new { x.Status, x.DueDate });
    b.HasIndex(x => x.CustomerId);
    b.HasIndex(x => x.UpdatedAt);
    b.HasOne(x => x.Customer).WithMany(c => c.Tickets)
        .HasForeignKey(x => x.CustomerId).OnDelete(DeleteBehavior.Restrict);
    b.HasOne(x => x.Agent).WithMany(a => a.Tickets)
        .HasForeignKey(x => x.AgentId).OnDelete(DeleteBehavior.SetNull);
});
```

Ogni riga è una decisione da cui dipende il browser dei ticket. L'indice composto su `Status` e `DueDate` serve la coda che filtra per stato e ordina per scadenza, `CustomerId` serve il filtro per cliente e `UpdatedAt` l'ordinamento "modificati di recente". `Restrict` significa che un cliente o una categoria con ticket non può essere eliminato per sbaglio, `SetNull` significa che un agente che se ne va si limita a togliere l'assegnazione ai suoi ticket, e i commenti usano `Cascade`. Una colonna di tipo monetario riceverebbe qui `HasPrecision(18, 2)`, e uno stato predefinito "Open" si imposta qui invece che nel codice del form.

Il token di concorrenza è ciò che conta di più in un'applicazione Wisej.NET multiutente. Un `byte[] RowVersion` configurato con `IsRowVersion` diventa una colonna SQL Server `rowversion` che il database cambia a ogni scrittura, ed EF Core include il valore originale nella condizione di aggiornamento: `UPDATE ... WHERE Id = @p0 AND RowVersion = @p1`. Se un altro utente ha modificato la riga dopo il caricamento, l'aggiornamento tocca zero righe e `SaveChangesAsync` lancia `DbUpdateConcurrencyException`; l'interfaccia può allora mostrare i valori correnti del database e offrire ricarica, sovrascrittura o unione. SQLite non ha un tipo `rowversion`: lì la proprietà resta un token di concorrenza, ma deve incrementarla un trigger o il codice.



Errore tipico: annotation per tutto

Decorare ogni entità con `[Required]`, `[MaxLength]` e `[Timestamp]` quando la Fluent API sarebbe più chiara, così l'entità diventa insieme mapping di persistenza e contratto di validazione dell'interfaccia.



Errore tipico: nessun token di concorrenza sulle schermate multiutente

Lasciare fuori `RowVersion` perché "la griglia è di sola lettura". L'editor non lo è, e senza un token il secondo agente che salva sovrascrive in silenzio il primo.

Migrazioni come sorgente da rivedere

Le migrazioni sono codice sorgente. Durante lo sviluppo aggiungi una ogni volta che il modello cambia con `dotnet ef migrations add InitialCreate --project SupportDesk.Data --startup-project SupportDesk.Web`, rivedi le operazioni `Up` e `Down` generate, poi applicala con `dotnet ef database update`. La revisione intercetta una rinomina generata come eliminazione e aggiunta di colonna che perde dati, un `Cascade` dove intendevi `Restrict`, un indice mancante o una colonna `rowversion` che non è mai comparsa. La migrazione e lo snapshot del modello vengono salvati insieme nel controllo di versione.

La produzione è un'altra cosa. Evita modifiche automatiche di schema non riviste all'avvio dell'applicazione in ambienti con più istanze: due istanze che partono insieme fanno a gara per applicare la stessa migrazione. Preferisci uno script SQL rivisto da `dotnet ef migrations script --idempotent` quando servono approvazione del database e piano di rollback, oppure un bundle di migrazione da `dotnet ef migrations bundle`, un eseguibile autonomo che la pipeline di rilascio esegue prima di spostare il traffico.

- Stringhe di connessione esternalizzate, mai nel codice sorgente.
- Ogni migrazione rivista; script o bundle generato e conservato con il rilascio.
- Rischio delle migrazioni all'avvio con più istanze affrontato per iscritto.

Gli strumenti `dotnet ef` devono poter costruire `SupportDeskContext` a design time, e una factory di design time è il modo più esplicito per consentirlo. `SupportDeskDesignTimeFactory` implementa `IDesignTimeDbContextFactory<SupportDeskContext>`; il suo `CreateDbContext` costruisce le opzioni con il provider dell'applicazione e una connessione locale sicura (`SupportDeskDev` di LocalDB, connessione trusted, nessuna password) e restituisce un nuovo Context. Conta perché lo startup di Wisej.NET include la configurazione del framework di interfaccia, che non dovrebbe girare durante lo scaffolding delle migrazioni; con la factory gli strumenti non lo toccano mai.



Errore tipico: migrazioni come file generati che non vanno mai rivisti

Trattare la cartella Migrations come output di compilazione. Una migrazione è una modifica di schema che qualcuno eseguirà in produzione; rivedila come qualsiasi altro commit.

I vincoli di schema non sono validazione dell'interfaccia

`HasMaxLength(180).IsRequired()` su `Title` produce una colonna `NOT NULL nvarchar(180)`, e violarla produce una `DbUpdateException` dopo un round trip, con un messaggio del provider che nessun utente dovrebbe leggere. È l'ultimo livello di sicurezza, non il primo. Il messaggio comprensibile "Inserisci un titolo" appartiene all' `ErrorProvider` dell'editor e al `TicketValidator` che aggiunge il Modulo 5, così come "i ticket chiusi non possono avere una data di scadenza futura", che nessun vincolo di colonna può esprimere. Tieni i livelli separati di proposito: la Fluent API dichiara ciò che il database deve garantire (nullabilità, lunghezza, l'indice univoco sul numero di ticket, chiavi esterne, il token di concorrenza); `DataAnnotations` e validatori sul modello di modifica dichiarano ciò che bisogna dire all'utente.

I dati iniziali seguono la stessa disciplina. Le righe di lookup fisse come le categorie possono passare da `HasData` così viaggiano con la migrazione; i cinquanta ticket è meglio produrli con un metodo di popolamento solo per lo sviluppo che crea un Context dalla factory, verifica se esistono già dei ticket, aggiunge le righe, chiama `SaveChangesAsync` ed elimina il Context. Dietro un pulsante "Seed development data" con una protezione di caricamento e un `try / catch`, è il punto in cui la query di conteggio del Modulo 1 restituisce per la prima volta più di zero.



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:

- ✓ Modellare entità e relazioni per un'interfaccia master-detail.
- ✓ Usare la Fluent API per chiavi, indici, campi obbligatori, lunghezza, precisione dei decimali, comportamento di eliminazione e row version.
- ✓ Creare migrazioni e capire quando usare script, bundle e factory di design time.
- ✓ Separare i vincoli dello schema del database dalla validazione di interfaccia e di dominio.

Guida al lab / esame

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



Lab pratico

Crea le entità Customer, Agent, Category, Ticket e TicketComment per la Support Desk Data Console, configura relazioni e comportamenti di eliminazione con la Fluent API, configura RowVersion con IsRowVersion, aggiungi indici per Status, DueDate, CustomerId e UpdatedAt, aggiungi una factory di design time, crea la migrazione InitialCreate e popola un database locale SQLite o SQL Server LocalDB con cinque clienti, tre agenti, le categorie e almeno cinquanta ticket.

Punto di partenza

Apri la soluzione Support Desk Data Console del Modulo 1: `SupportDesk.Web` (il progetto Wisej.NET), `SupportDesk.Data` con un `SupportDeskContext` vuoto registrato tramite `AddDbContextFactory`, `SupportDesk.Services` e `SupportDesk.Tests`. La pagina del Modulo 1 esegue ancora la sua query di conteggio asincrona su un database senza tabelle, quindi la prima cosa che vedrai a lab concluso è quel conteggio che diventa un numero reale.

Approccio suggerito

- ✓ Scrivi prima le cinque classi di entità in `SupportDesk.Data`, con chiavi intere, proprietà di navigazione in entrambe le direzioni, un `AgentId` nullable e un `byte[] RowVersion` su `Ticket`; tienile libere da attributi di mapping.
- ✓ Configura in `OnModelCreating` tutto ciò che tocca lo schema: stringhe obbligatorie e lunghezze, `IsRowVersion`, i quattro indici pianificati e `onDelete` per ogni relazione (Restrict per Customer e Category, SetNull per Agent, Cascade per i commenti).
- ✓ Aggiungi `SupportDeskDesignTimeFactory` con lo stesso provider usato dall'applicazione e una connessione locale che non contiene password, poi esegui `dotnet ef migrations add InitialCreate` dalla radice della soluzione.
- ✓ Leggi la migrazione generata prima di applicarla: controlla la colonna `rowversion`, i nomi degli indici, le clausole `ON DELETE` e lo snapshot, poi esegui `dotnet ef database update`.
- ✓ Scrivi un metodo di popolamento solo per lo sviluppo che crea un Context dalla factory, verifica se esistono già dei ticket, aggiunge clienti, agenti, categorie e ticket, salva ed elimina il Context, e chiamalo da un pulsante Seed dietro una protezione di caricamento e un `try / catch`.

Deliverable richiesti

- ✓ Entità Customer, Agent, Category, Ticket e TicketComment con relazioni e comportamenti di eliminazione
- ✓ RowVersion configurata con IsRowVersion
- ✓ Indici per Status, DueDate, CustomerId e UpdatedAt
- ✓ Factory di design time e migrazione InitialCreate
- ✓ Dati iniziali: cinque clienti, tre agenti, le categorie e almeno cinquanta ticket

Domande di revisione per lo studente

- ✓ Elimini un cliente che ha ancora ticket aperti. Che cosa succede con il comportamento di eliminazione che hai configurato, e che cosa sarebbe successo con Cascade?
- ✓ Due agenti caricano il ticket SD-1042 ed entrambi cliccano Save a un secondo di distanza. Quale salvataggio lancia l'eccezione, che cosa contiene la clausola WHERE dell'istruzione UPDATE e che cosa vede l'utente?
- ✓ Perché `IsRequired()` su `Title` non dà all'utente un messaggio "Inserisci un titolo", e da dove arriverà quel messaggio più avanti nel corso?



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.