Ajay Patel

EF Core

EFCore hidden feature

dbContext.SaveChangesAsync(false)

```
(acceptAllChangesOnSuccess: false);
```

Ajay Patel

Save as Regular way

This is how we usually save an entity.

await dbContext.SaveChangesAsync();

```
app.MapPost("api/books", async (AppDbContext dbContext, Book book) =>
{
    dbContext.Books.Add(book);

    var before = dbContext.ChangeTracker.DebugView.LongView;

    await dbContext.SaveChangesAsync();

    var after = dbContext.ChangeTracker.DebugView.LongView;
});
```

Ajay Patel

Let see, how ChangeTracker works?

Add the Book entity

```
dbContext.Books.Add(book);

var before = dbContext.ChangeTracker.DebugView.LongView;
```

Text Visualizer

Expression: dbContext.ChangeTracker.DebugView.LongView

String manipulation: None

Value:

```
Book {BookId: -2147482647} Added
    BookId: -2147482647 PK Temporary
    AuthorId: 1 FK
    Description: 'Book 2 Description'
    PublishedDate: '1/1/0001'
    Title: 'Book 2'
```

- ChangeTracker tracked the entity.
- Marked the status: Added
- No PrimaryKey added yet.

Ajay Patel

Save to database

```
await dbContext.SaveChangesAsync();

var after = dbContext.ChangeTracker.DebugView.LongView;
```

Text Visualizer

Expression: dbContext.ChangeTracker.DebugView.LongView

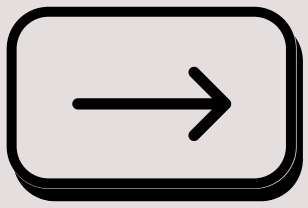
String manipulation: None

Value:

```
Book {BookId: 6} Unchanged
    BookId: 6 PK
    AuthorId: 1 FK
    Description: 'Book 2 Description'
    PublishedDate: '1/1/0001'
    Title: 'Book 2'
```

```
Context dbContext

nclude(a=> a.Book
```

- ChangeTracker accepted the BookId as a result of the SaveChanges execution.
- Marked the status: Unchanged
- This means that there are no changes left to send to the database.



Ajay Patel

Save in an unusual way

(acceptAllChangesOnSuccess: false)

```
app.MapPost("api/books", async (AppDbContext dbContext, Book book) =>
{
    dbContext.Books.Add(book);

    await dbContext.SaveChangesAsync(acceptAllChangesOnSuccess:false);

    var before = dbContext.ChangeTracker.DebugView.LongView;

    dbContext.ChangeTracker.AcceptAllChanges();

    var after = dbContext.ChangeTracker.DebugView.LongView;
});
```

Ajay Patel

Let’s see, how ChangeTracker works now

Add the Book entity

```
dbContext.Books.Add(book);

var before = dbContext.ChangeTracker.DebugView.LongView;
```

Text Visualizer

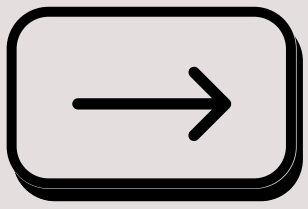
Expression: dbContext.ChangeTracker.DebugView.LongView

String manipulation: None

Value:

```
Book {BookId: -2147482647} Added
    BookId: -2147482647 PK Temporary
    AuthorId: 1 FK
    Description: 'Book 2 Description'
    PublishedDate: '1/1/0001'
    Title: 'Book 2'
```

- ChangeTracker tracked the entity.
- Marked the status: Added
- No PrimaryKey added yet.



Ajay Patel

Save to database

```
dbContext.Books.Add(book);

await dbContext.SaveChangesAsync(acceptAllChangesOnSuccess:false);

var before = dbContext.ChangeTracker.DebugView.LongView;   ≤ 272ms ela
```

Text Visualizer

Expression: dbContext.ChangeTracker.DebugView.LongView

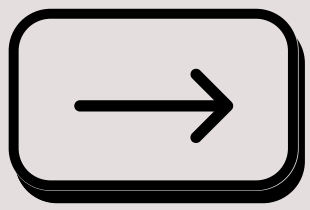
String manipulation: None

Value:

```
Book {BookId: 5} Added
    BookId: 5 PK
    AuthorId: 1 FK
    Description: 'Book 2 Description'
    PublishedDate: '1/1/0001'
    Title: 'Book 2'
```

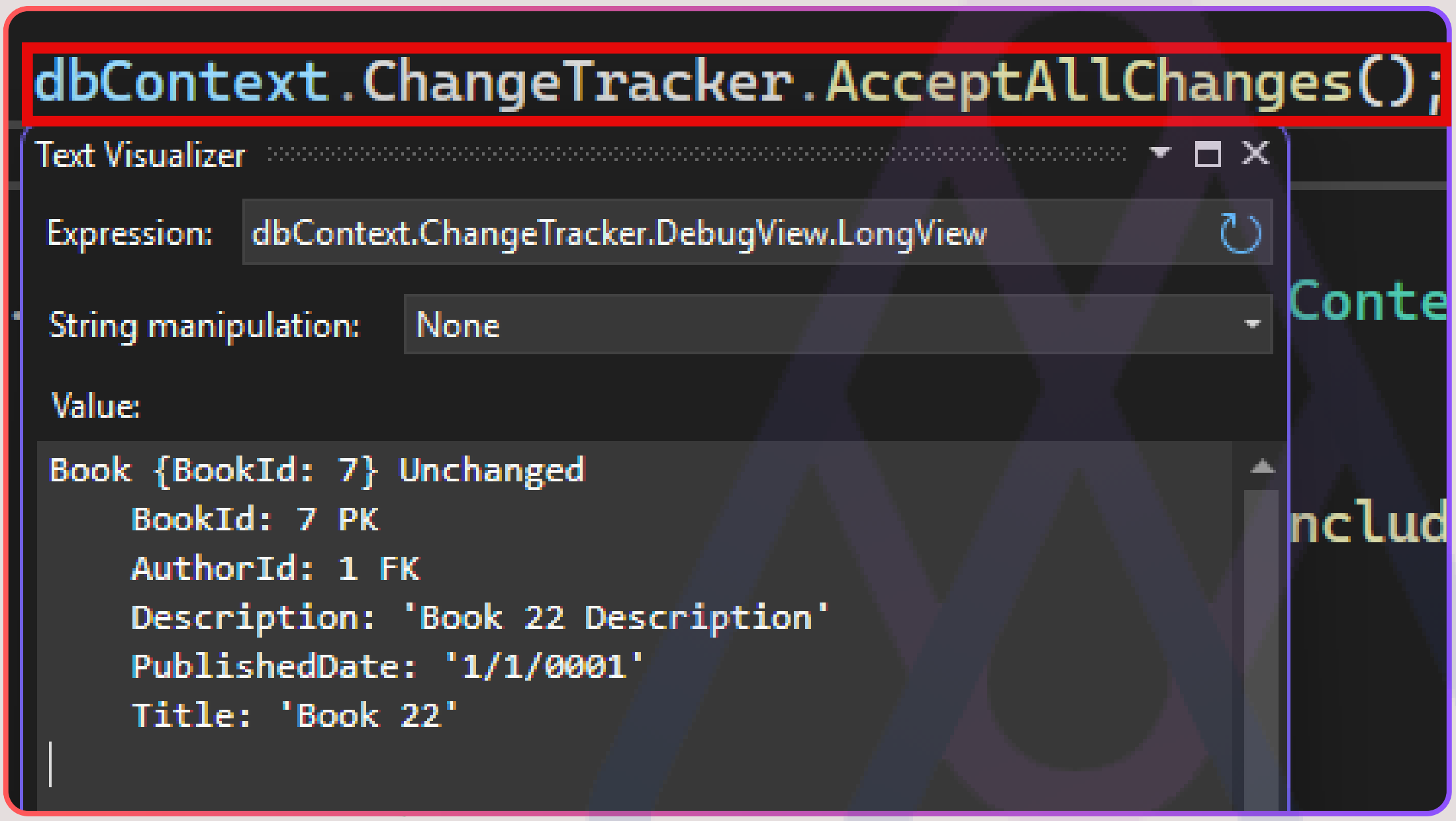
```
es();

bugView.LongView;

Context dbContext,int id) =>

nclude(a=> a.Books).AsSplitQu
```

- ChangeTracker did not reset the status of the entity.
- Status is still Added
- This means you can still retry SaveChanges if needed.



Ajay Patel

Manually AcceptAllChanges:



- ChangeTracker has reset the entity status
- Marked status as Unchanged

Ajay Patel

Read this carefully to fully understand it

If you call **SaveChangesAsync()**, EF Core simply assumes everything is okay. As a result, **it will discard the changes** it has been tracking **(Status: Unchanged)** and wait for new changes.

Unfortunately, if **something goes wrong** elsewhere in the transaction, EF Core will have already discarded the changes it was tracking, meaning **we can't recover them**.

This is where **SaveChangesAsync(false)** and **AcceptAllChanges()** come in.

SaveChanges(false) tells the EFCore to execute the necessary database commands, but **hold on to the changes**, so they can be replayed if necessary.

Now, if the broader **transaction fails you can retry** the EFCore specific bits, with another call to SaveChanges(false).

Ajay Patel

When to use acceptAllChanges?

- When implementing custom retry logic for database operations
- When you need fine-grained control over entity state tracking
- When building multi-step transaction workflows where changes should not be marked as final immediately

Ajay Patel

.NET

Knowledge is contagious, let's spread it!

DO YOU LIKE THIS POST?

REPOST IT!

THANKS FOR READING