


# Handling Concurrency Conflicts

03/03/2018 • 3 minutes to read •  +6

## In this article

[How concurrency control works in EF Core](#)

[Resolving concurrency conflicts](#)

[Additional resources](#)

### Note

This page documents how concurrency works in EF Core and how to handle concurrency conflicts in your application. See [Concurrency Tokens](#) for details on how to configure concurrency tokens in your model.

### Tip

You can view this article's [sample](#) on GitHub.

*Database concurrency* refers to situations in which multiple processes or users access or change the same data in a database at the same time. *Concurrency control* refers to specific mechanisms used to ensure data consistency in presence of concurrent changes.

EF Core implements *optimistic concurrency control*, meaning that it will let multiple processes or users make changes independently without the overhead of synchronization or locking. In the ideal situation, these changes will not interfere with each other and therefore will be able to succeed. In the worst case scenario, two or more processes will attempt to make conflicting changes, and only one of them should succeed.

## How concurrency control works in EF Core

Properties configured as concurrency tokens are used to implement optimistic concurrency control: whenever an update or delete operation is performed during `SaveChanges`, the value of the concurrency token on the database is compared against the original value read by EF Core.

- If the values match, the operation can complete.
- If the values do not match, EF Core assumes that another user has performed a conflicting operation and aborts the current transaction.


The situation when another user has performed an operation that conflicts with the current operation is known as *concurrency conflict*.

Database providers are responsible for implementing the comparison of concurrency token values.

On relational databases EF Core includes a check for the value of the concurrency token in the `WHERE` clause of any `UPDATE` or `DELETE` statements. After executing the statements, EF Core reads the number of rows that were affected.

If no rows are affected, a concurrency conflict is detected, and EF Core throws `DbUpdateConcurrencyException`.

For example, we may want to configure `LastName` on `Person` to be a concurrency token. Then any update operation on `Person` will include the concurrency check in the `WHERE` clause:

SQL	 Copy
<pre>UPDATE [Person] SET [FirstName] = @p1 WHERE [PersonId] = @p0 AND [LastName] = @p2;</pre>	

## Resolving concurrency conflicts

Continuing with the previous example, if one user tries to save some changes to a `Person`, but another user has already changed the `LastName`, then an exception will be thrown.

At this point, the application could simply inform the user that the update was not successful due to conflicting changes and move on. But it may be desirable to prompt the user to ensure this record still represents the same actual person and to retry the operation.

This process is an example of *resolving a concurrency conflict*.

Resolving a concurrency conflict involves merging the pending changes from the current `DbContext` with the values in the database. What values get merged will vary based on the application and may be directed by user input.

## There are three sets of values available to help resolve a concurrency conflict:

- **Current values** are the values that the application was attempting to write to the database.
- **Original values** are the values that were originally retrieved from the database, before any edits were made.
- **Database values** are the values currently stored in the database.

The general approach to handle a concurrency conflicts is:

1. Catch `DbUpdateConcurrencyException` during `SaveChanges`.
2. Use `DbUpdateConcurrencyException.Entries` to prepare a new set of changes for the affected entities.
3. Refresh the original values of the concurrency token to reflect the current values in the database.
4. Retry the process until no conflicts occur.

In the following example, `Person.FirstName` and `Person.LastName` are set up as concurrency tokens. There is a `// TODO:` comment in the location where you include application specific logic to choose the value to be saved.

C#



```
using var context = new PersonContext();
// Fetch a person from database and change phone number
var person = context.People.Single(p => p.PersonId == 1);
person.PhoneNumber = "555-555-5555";

// Change the person's name in the database to simulate a concurrency conflict
context.Database.ExecuteSqlRaw(
    "UPDATE dbo.People SET FirstName = 'Jane' WHERE PersonId = 1");

var saved = false;
while (!saved)
{
    try
    {
        // Attempt to save changes to the database
        context.SaveChanges();
        saved = true;
    }
    catch (DbUpdateConcurrencyException ex)
    {
        foreach (var entry in ex.Entries)
        {
            if (entry.Entity is Person)
            {
                // TODO: application specific logic to choose the value to be saved
            }
        }
    }
}
```

```
var proposedValues = entry.CurrentValues;
var databaseValues = entry.GetDatabaseValues();

foreach (var property in proposedValues.Properties)
{
    var proposedValue = proposedValues[property];
    var databaseValue = databaseValues[property];

    // TODO: decide which value should be written to database
    // proposedValues[property] = <value to be saved>;
}

// Refresh original values to bypass next concurrency check
entry.OriginalValues.SetValues(databaseValues);
}
else
{
    throw new NotSupportedException(
        "Don't know how to handle concurrency conflicts for "
        + entry.Metadata.Name);
}
}
}
```

## Additional resources

See [Conflict detection in EF Core](#) for an ASP.NET Core sample with conflict detection.

## Is this page helpful?

 Yes  No

## Recommended content

### [Client vs. Server Evaluation - EF Core](#)

Client and server evaluation of queries with Entity Framework Core

### [Global Query Filters - EF Core](#)

Using global query filters to filter results with Entity Framework Core

## Efficient Querying - EF Core

Performance guide for efficient querying using Entity Framework Core

## Tracking vs. No-Tracking Queries - EF Core

Information on tracking and no-tracking queries in Entity Framework Core

Show more ▼