ARTICLE SOURCE

What is the N + 1 problem?

In simple terms, the N + 1 query problem happens when you:

1. Run 1 query to fetch a list of records
2. Then, for each of those N records, you run another query.

That's why it's called N + 1: one main query plus N additional queries.

Why It's a Problem

- Performance bottlenecks

Instead of running a single efficient query, you are now hitting the database multiple times. This slows your application significantly.

- Scalability issues

The more records you have in the database, severe it gets. With 10 users, you will run 11 queries. With 10,000 users you will run 10, 001 queries — making the application unscalable.

Example of N + 1 Problem

Example of fetching all authors and their posts.

github code solution

Inefficient approach:

- Query 1: Get all the authors
- Query 2:
    - … } For each author, fetch their posts
- Query N:

Example in Go with Gin

```
package controller

import (
    "net/http"
    // "database/sql"
    "github.com/gin-gonic/gin"

    "n-plus-one-problem/db"
    "n-plus-one-problem/models"
)

func GetAuthorsAndPosts(ctx *gin.Context) {
    // fetching all the authors
    var authors []models.Author
    query := "SELECT * FROM authors"
    if err := db.DB.Select(&authors, query); err != nil {
        ctx.JSON(http.StatusInternalServerError, gin.H{
                "error": "Error fetching the authors",
```

```
            })
            return
        }

        for i, author := range authors {
            var posts []models.Post
            query = "SELECT * FROM posts WHERE author_id = $1"
            if err := db.DB.Select(&posts, query, author.ID); err ==
nil {
                authors[i].Posts = posts
            }
        }

        ctx.JSON(http.StatusOK, authors)
}
```

Efficient approach:- Fetch all authors and their posts in one query using a SQL JOIN

```
package controller

import (
    "net/http"
    // "database/sql"
    "github.com/gin-gonic/gin"

    "n-plus-one-problem/db"
    "n-plus-one-problem/models"
)

func GetAuthorsAndPosts(ctx *gin.Context) {
    type AuthorWithPosts struct {
        AuthorID int    `db:"author_id" json:"author_id"`
        Name     string `db:"name" json:"name"`
        PostID   int    `db:"post_id" json:"post_id"`
```

```
        Title    string `db:"title" json:"title"`
    }

    var results []AuthorWithPosts
    query := `
        SELECT a.id as author_id, a.name, p.id as post_id,
p.title
        FROM authors a
        LEFT JOIN posts p ON p.author_id = a.id
    `

    if err := db.DB.Select(&results, query); err != nil {
        ctx.JSON(http.StatusInternalServerError, gin.H{
                "error": "Failed to fetch data",
         })
        return
    }

    ctx.JSON(http.StatusOK, results)
}
```

How to Avoid N + 1 Queries

1. Use SQL joins (JOIN, LEFT JOIN) instead of looping queries.
2. Leverage ORM features like:
    - Django: .select_related, .prefetch_related
    - SQLAlchemy: joinedload, subqueryload
    - Hibernate: fetch joins
3. Batch queries instead of per-record queries.

My conclusion

The N + 1 problem is sneaky — your code still works, but performance suffers silently. The solution is almost always about fetching smarter, not harder: reduce round trips, load related data in bulk, and let your database do the heavy lifting.