cancel-operations

- Canceling database operations after a timeout {#timeout_cancel}

You can manage in-progress operations by using Go `context.Context`. A `Context` is a standard Go data value that can report whether the overall operation it represents has been canceled and is no longer needed. By passing a `context.Context` across function calls and services in your application, those can stop working early and return an error when their processing is no longer needed. For more about `Context`, see Go Concurrency Patterns: Context.

For example, you might want to:

- End long-running operations, including database operations that are taking too long to complete.
- Propagate cancellation requests from elsewhere, such as when a client closes a connection.

Many APIs for Go developers include methods that take a `Context` argument, making it easier for you to use `Context` throughout your application.

Canceling database operations after a timeout {#timeout_cancel}

You can use a `Context` to set a timeout or deadline after which an operation will be canceled. To derive a `Context` with a timeout or deadline, call `context.WithTimeout` or `context.WithDeadline`.

Code in the following timeout example derives a `Context` and passes it into the `sql.DB` `QueryContext` method.

```
func QueryWithTimeout(ctx context.Context) {
	// Create a Context with a timeout.
	queryCtx, cancel := context.WithTimeout(ctx, 5*time.Second)
	defer cancel()

	// Pass the timeout Context with a query.
```

```
□rows, err := db.QueryContext(queryCtx, "SELECT * FROM album")
□if err != nil {
□□log.Fatal(err)
□}
□defer rows.Close()

□// Handle returned rows.
}
```

When one context is derived from an outer context, as `queryCtx` is derived from `ctx` in this example, if the outer context is canceled, then the derived context is automatically canceled as well. For example, in HTTP servers, the `http.Request.Context` method returns a context associated with the request. That context is canceled if the HTTP client disconnects or cancels the HTTP request (possible in HTTP/2). Passing an HTTP request's context to `QueryWithTimeout` above would cause the database query to stop early *either* if the overall HTTP request was canceled or if the query took more than five seconds.

Note: Always defer a call to the `cancel` function that's returned when you create a new `Context` with a timeout or deadline. This releases resources held by the new `Context` when the containing function exits. It also cancels `queryCtx`, but by the time the function returns, nothing should be using `queryCtx` anymore.

□□□ #2
□ □□□ □□□ 2026-10-27 04:26:29 UTC
□ □□□ □□□ 2026-10-27 04:26:29 UTC