# Connection Pool Project

## Learning Objectives

Build a production-ready connection pool to learn:

- Resource pooling with semaphore pattern
- Connection lifecycle (create, validate, close)
- Circuit breaker integration
- Health checks and connection recycling
- Graceful degradation

## Requirements

### Functional

1. **Pool Management**
    - Min/Max connection limits
    - Connection reuse
    - Lazy creation (on-demand)
    - Connection validation before use
2. **Health Monitoring**
    - Ping connections periodically
    - Remove dead connections
    - Circuit breaker for failing backends
    - Automatic recovery
3. **Resource Limits**
    - Max connections (prevent exhaustion)
    - Max idle time (close stale connections)
    - Max lifetime (recycle old connections)
    - Wait timeout (don't block forever)

### Non-Functional

1. **Performance** - <1ms acquire latency
2. **Reliability** - Automatic failover
3. **Observability** - Pool utilization metrics

## Implementations

### Naive

Single mutex, unbounded growth, no health checks

### Improved

Buffered channel pool, connection validation, cleanup

### Final

Min/Max limits, circuit breaker, health checks, metrics

## Usage

```
pool := final.NewPool(final.Config{
    Factory: func() (Conn, error) {
        return sql.Open("postgres", dsn)
    },
    MinConns: 5,
    MaxConns: 100,
    MaxIdleTime: 5 * time.Minute,
})

conn, err := pool.Acquire(ctx)
if err != nil {
    return err
}
defer pool.Release(conn)

conn.Query("SELECT * FROM users")
```