BookingQueue + FxDag — Deep Dive & Troubleshooting Guide

Versione draft — include flussi, strutture dati, algoritmi, env di calcolo, hook sequence, testing con ApiScenario e checklist di debug.

1) High-level Flow

1. Trigger: Create/Update/Delete di BookingQueue o mutazioni di Booking.
2. Normalize: sostituzione `this.` → `.`, raccolta dipendenze; set `referenced_queues`.
3. Build DAG: pre-crea nodi self (act/min/max/linked_bookings), collega upstream, espandi downstream (BFS) figli che referenziano i nodi attuali.
4. Validate: anti-cycle (AddEdge con check), TopologicalSort.
5. Evaluate: per ciascun nodo ordinato → env minimale in base a `ReferencedVars`, `expr.Run`, `Dirty`, salvataggi senza hooks.
6. Propagate: i figli sono già nel grafo; altri trigger (es. Booking create) ripartono dal nodo opportuno.

2) Strutture Dati

```
// dag/fx_dag.go
type FxNode struct {
    ID              string
    QueueID         string
    ValField        string // act_val|min_val|max_val|linked_bookings
    Queue           *BookingQueue
    GetFx           func() string
    SetFx           func(string)
    GetVal          func() any
    SetVal          func(int)
    OriginalVal     any
    CompiledProgram *vm.Program
    Children        map[string]*FxNode
    ReferencedVars  *commons.Set[string]
    Dirty           bool
}
func (g *FxDag) NewNode(queue *BookingQueue, valField string,
    getFx func() string, setFx func(string),
    getVal func() any, setVal func(int)) (*FxNode, error) {

    nodeID := fmt.Sprintf("%s.%s", queue.Id, valField)
    formula := ""
    if getFx != nil { formula = getFx() }

    var compiled *vm.Program
    var err error
    if formula != "" {
        compiled, err = expr.Compile(formula, expr.AsFloat64())
        if err != nil { return nil, fmt.Errorf("compile %s: %w", nodeID, err) }
    }

    referenced := dag.GetFxVars(formula)

    node := &FxNode{
        ID: nodeID, QueueID: queue.Id, ValField: valField,
        Queue: queue, GetFx: getFx, SetFx: setFx,
        GetVal: getVal, SetVal: setVal,
        OriginalVal: getVal(),
        CompiledProgram: compiled,
        Children: make(map[string]*FxNode),
        ReferencedVars: referenced,
    }
    g.Nodes[node.ID] = node
    return node, nil
}
```

3) Normalize & updateDepQueuesFldAndCheckNormalizeFx

```
func updateDepQueuesFldAndCheckNormalizeFx(txApp core.App, checkingQueue *BookingQueue) *BookingQueue {
    parentIds := commons.New[string]()

    process := func(getFx func() string, setFx func(string)) {
        fx := strings.ReplaceAll(getFx(), "this.", checkingQueue.Id+".")
        setFx(fx)
        for _, fxVar := range dag.GetFxVars(fx).ToSlice() {
            if parentId, _, _ := strings.Cut(fxVar, "."); parentId != "" {
                parentIds.Add(parentId)
            }
        }
    }

    process(checkingQueue.ActFormula, checkingQueue.SetActFormula)
    process(checkingQueue.MinFormula, checkingQueue.SetMinFormula)
    process(checkingQueue.MaxFormula, checkingQueue.SetMaxFormula)

    recs, _ := txApp.FindRecordsByIds(CName[BookingQueue](), parentIds.ToSlice())
    refs := make([]*BookingQueue, len(recs))
    for i, rec := range recs { p,_ := WrapRecord[BookingQueue](rec); refs[i] = p }
    checkingQueue.SetReferencedQueues(refs)
    return checkingQueue
}
```

4) Build DAG (upstream + BFS downstream)

```
func buildDagForValField(txApp core.App, g *dag.FxDag, root *dag.FxNode) (*dag.FxDag, error) {
    if _, ok := g.Nodes[root.ID]; !ok { g.Nodes[root.ID] = root }

    for _, fxVar := range root.ReferencedVars.ToSlice() {
        up, err := fetchUpstreamNodeFromDB(txApp, g, fxVar)
        if err != nil { return g, err }
        if err := g.AddEdge(up, root); err != nil { return g, err }
    }

    queue := []*dag.FxNode{root}
    for i := 0; i < len(queue); i++ {
        cur := queue[i]
        if errs := txApp.ExpandRecord(cur.Queue.Record, []string{"booking_queue_via_referenced_queues"}, ni
            return g, fmt.Errorf("expand: %v", errs)
        }
        for _, childRec := range cur.Queue.ExpandedAll("booking_queue_via_referenced_queues") {
            child, _ := WrapRecord[BookingQueue](childRec)
            for _, mk := range []func(*BookingQueue)(*dag.FxNode,error){ g.NewActNode, g.NewMinNode, g.NewM
                n, _ := mk(child)
                if n.ReferencedVars.IsElement(cur.ID) {
                    if err := g.AddEdge(cur, n); err != nil { return g, err }
                    queue = append(queue, n)
                }
            }
        }
    }
    return g, nil
}
```

5) Env per expr.Run (on-demand, per nodo)

```
env := map[string]map[string]any{}
for _, fxVar := range node.ReferencedVars.ToSlice() {
    qid, field, _ := strings.Cut(fxVar, ".")
    if _, ok := env[qid]; !ok { env[qid] = map[string]any{} }
    env[qid][field] = g.Nodes[fxVar].GetVal() // int or []string
}
```

6) Nodo linked_bookings

```
func (g *FxDag) NewLinkedBookingsNode(txApp core.App, queue *BookingQueue) (*FxNode, error) {
    return g.NewNode(
        queue,
        "linked_bookings",
        nil, nil,
```

```
        func() any {
            txApp.ExpandRecord(queue.Record, []string{"booking_via_queue"}, nil)
            var ids []string
            for _, r := range queue.Record.ExpandedAll("booking_via_queue") { ids = append(ids, r.Id) }
            return ids
        },
        nil,
    )
}
```

7) CalculateFormulas (entrypoint hook)

```
func CalculateFormulas(txApp core.App, q *BookingQueue) error {
    q = updateDepQueuesFldAndCheckNormalizeFx(txApp, q)
    _ = txApp.UnsafeWithoutHooks().Save(q)
    g := dag.NewFxDag()
    prev, _ := WrapRecord[BookingQueue](q.Original())

    act,_ := g.NewActNode(q)
    min,_ := g.NewMinNode(q)
    max,_ := g.NewMaxNode(q)
    lbk,_ := g.NewLinkedBookingsNode(txApp, q)

    if prev.ActFormula() != q.ActFormula() { if _, err := buildDagForValField(txApp, g, act); err != nil {
    if prev.MinFormula() != q.MinFormula() { if _, err := buildDagForValField(txApp, g, min); err != nil {
    if prev.MaxFormula() != q.MaxFormula() { if _, err := buildDagForValField(txApp, g, max); err != nil {
    _, _ = buildDagForValField(txApp, g, lbk)

    ordered, err := g.TopologicalSort()
    if err != nil { return err }

    for _, n := range ordered {
        if n.CompiledProgram == nil { continue }
        env := map[string]map[string]any{}
        for _, v := range n.ReferencedVars.ToSlice() {
            qid, fld, _ := strings.Cut(v, ".")
            if _, ok := env[qid]; !ok { env[qid] = map[string]any{} }
            env[qid][fld] = g.Nodes[v].GetVal()
        }
        out, err := expr.Run(n.CompiledProgram, env)
        if err != nil { return fmt.Errorf("eval %s: %w", n.ID, err) }
        newVal := int(out.(float64))
        if newVal != n.OriginalVal {
            n.SetVal(newVal)
            if err := txApp.UnsafeWithoutHooks().Save(n.Queue.Record); err != nil {
                return fmt.Errorf("save %s: %w", n.QueueID, err)
            }
        }
    }
    return nil
}
```

8) Hook Registration (BookingQueue)

```
func BindBookingQueueHook() {
    if PHooks == nil { panic("PHooks not initialized") }
    PHooks.OnBookingQueueCreate.BindFunc(OnBookingQueueCreateUpdate)
    PHooks.OnBookingQueueUpdate.BindFunc(OnBookingQueueCreateUpdate)
    PHooks.OnBookingQueueDelete.BindFunc(OnBookingQueueRecordBeforeDelete)
}
```

9) Testing con tests.ApiScenario

```
// body costruito via Proxy + MarshalJSON
q, _ := generated.NewProxy[generated.BookingQueue](app)
q.SetQueueName("TestQueue")
q.SetBookingStatus(generated.BookingStatusBooked)
q.SetCourse("COURSE_ID_EXISTENTE")
q.SetActFormula("0"); q.SetMinFormula("0"); q.SetMaxFormula("0")
body, _ := q.MarshalJSON()

sc := tests.ApiScenario{
  Name: "Create BookingQueue triggers CalculateFormulas hook",
  Method: http.MethodPost,
```

```
  URL: "/api/collections/booking_queue/records",
  Body: bytes.NewReader(body),
  Headers: map[string]string{"Content-Type":"application/json"},
  ExpectedStatus: 200,
}
```

10) Troubleshooting Matrix

Sintomo	Possibile causa	Dove intervenire
400 Failed to create record	Campi richiesti mancanti (es. booking_status, owner)	Test body/validazione collection
Ciclo rilevato AddEdge	Formula referenzia a catena la stessa query	normalize + buildDagForValField (verifica vars
Valore non si aggiorna	Env incompleto o CompiledProgram nil	Node.ReferencedVars, NewNode compile, env
Figli non ricalcolati	BFS non aggiunge nodi perché non referenziano ctx.id	CheckingGetExVars() e normalize this.
Expr error type	linked_bookings usato come numero	Usa len(<qid>.linked_bookings) nella formula

11) Esempio dati passo-passo

A. Input (POST)

```
{
  "queue_name": "Q_A",
  "booking_status": "booked",
  "course": "COURSE_1",
  "act_formula": "len(Q_A.linked_bookings)",
  "min_formula": "0",
  "max_formula": "Q_B.max_val + 5"
}
```

B. Dopo normalize()

```
act_formula = "len(Q_A.linked_bookings)"
min_formula = "0"
max_formula = "Q_B.max_val + 5"
referenced_queues = ["Q_A","Q_B"]
```

C. Nodi creati

```
Q_A.act_val, Q_A.min_val, Q_A.max_val, Q_A.linked_bookings
```

D. Edges upstream

```
Q_A.linked_bookings → Q_A.act_val
Q_B.max_val → Q_A.max_val
```

E. Env per Q_A.max_val

```
{
  "Q_B": { "max_val": 10 }
}
```

F. Env per Q_A.act_val

```
{
  "Q_A": { "linked_bookings": ["bk1","bk2","bk3"] }
}
```

G. Risultati e salvataggi

```
Q_A.act_val = 3 (dirty → Save)
Q_A.max_val = 15 (dirty → Save)
```

12) Checklist rapida

- Sostituito `this.` correttamente?
- GetFxVars() popola ReferencedVars?
- linked_bookings node creato sempre?
- AddEdge → no self-ref, no cycle?
- TopologicalSort ok?
- Env minimale creato per ogni nodo?
- Salvataggio UnsafeWithoutHooks solo quando serve?
- Hooks registrati una sola volta?
- Test ApiScenario passa con body valido?