DOCUMENT 2: HIERARCHICAL DNA SYSTEM

Implementation and Algorithms for Infinite Traceability

For: Cyril Amegah - Technical Assessment THE BRIDGE
Date: August 10, 2025
Purpose: Implement hierarchical DNA with anti-collision algorithms

---

PART A: DNA CONCEPT - WHY IT'S REVOLUTIONARY

🧬 DNA = COMPLETE HISTORY OF EVERY PROCESS

Simple analogy: Just as human DNA tells the genetic history of a person, the processual DNA tells the complete history of every business activity.

CONCRETE EXAMPLE:

```
TSK25001→PRJ25002→RCH25003→TEH25004→APR25005
```

This code means:

```
├── TSK25001: Everything started with a CLIENT PHONE CALL
├── PRJ25002: From the phone call a PROJECT was born
├── RCH25003: The project generated SAMPLE REQUEST
├── TEH25004: The samples led to TECHNICAL SHEET
└── APR25005: The technical sheet is under APPROVAL
```

DNA POWER:

- **Infinite traceability:** Always trace back to origin
- **Semantic navigation:** The code tells the story
- **Automatic aggregation:** Reports generate themselves
- **Consistency control:** Errors become impossible

⚡ ADVANTAGES vs TRADITIONAL SYSTEMS

Traditional System:

- TASK_001, TASK_002, TASK_003 (meaningless numbers)
- Business Intelligence needed to understand connections
- Manual reporting subject to errors

3P3 DNA System:

- TSK25001→PRJ25002→RCH25003 (readable history)
- Automatic connections from the code itself
- Reports generated automatically following DNA

PART B: DEFINITIVE DNA FORMAT

📋 STANDARD STRUCTURE

FORMAT: PRXYYNNNN

```
WHERE:
├── PRX = Process Type (TSK, PRJ, RCH, TEH, APR, etc.)
├── YY = Year (25 = 2025)
└── NNNN = Progressive (0001, 0002, 0003, etc.)
```

🔍 REAL KOOL TOOL EXAMPLES

COLOR CHART WORKFLOW:

1. CLIENT PHONE CALL:

DNA: TSK25001 ├── TSK = Type: Task/Phone call ├── 25 = Year: 2025
└── 001 = First phone call of the year

2. PROJECT OPENING:

DNA: PRJ25001
Parent: TSK25001

MEANING: Project opened, generated from phone call TSK25001

3. SAMPLE REQUEST:

DNA: RCH25001
Parent: PRJ25001

MEANING: Sample request for project PRJ25001

4. TECHNICAL SHEET:

DNA: TEH25001
Parent: RCH25001

MEANING: Technical sheet based on samples RCH25001

5. FINAL APPROVAL:

DNA: APR25001
Parent: TEH25001

MEANING: Approval of technical sheet TEH25001

SUB-PROCESSES RCH (Managing 120 Hair Strands):

Registration Swatches:

DNA: RCH25002
Parent: RCH25001
Type: Registration of the 120 requested strands

Sample Preparation:

DNA: RCH25003
Parent: RCH25002
Type: Workspace preparation for sampling

Specific Strand (example BLONDE_L8):

DNA: RCH25004
Parent: RCH25003
Specification: BLONDE_L8
Recipe: BOB001(3parts) + BOB002(2parts)

Label Printing:

DNA: RCH25005
Parent: RCH25004
Type: Label printing for strands

Client Approval (Web Interface):

DNA: APR25002
Parent: RCH25005
Status: PENDING_CLIENT_APPROVAL

PART C: DNA GENERATION ALGORITHMS

⚙️ ANTI-COLLISION ALGORITHM

PROBLEM: How to guarantee uniqueness in multi-user environment?

SOLUTION: SEQUENCE Table + Lock Mechanism

1. SEQUENCE TABLE:

```sql
SEQUENCE TABLE:
├── SequenceDate (Date, Primary Key)
├── ProcessType (Text, Primary Key)
├── LastNumber (Number)
├── Lock_UserID (Text)
├── Lock_Timestamp (Timestamp)
└── Lock_Duration (Number, seconds)
```

2. STEP-BY-STEP ALGORITHM:

```javascript
FUNCTION generateDNA(processType, ownerCode) {
    // STEP 1: Lock sequence per process/year
    currentYear = getYear(today())
    acquireLock(processType, currentYear, userID)

    // STEP 2: Get next number
    currentSequence = getSequence(processType, currentYear)
    nextNumber = currentSequence + 1

    // STEP 3: Update sequence
    updateSequence(processType, currentYear, nextNumber)

    // STEP 4: Generate DNA
    yearString = String(currentYear).substring(2,4) // "25"
    sequenceString = padLeft(nextNumber, 4) // "0001", "0002", etc.
    DNA = processType + yearString + sequenceString

    // STEP 5: Release lock
    releaseLock(processType, currentYear)

    return DNA
}
```

3. PRACTICAL EXAMPLE:

Scenario: Marco and Giuseppe create RCH simultaneously on 20/08/2025

Marco (14:30:15):

1. acquireLock("RCH", "2025", "MAR") → SUCCESS
2. getSequence("RCH", "2025") → 0
3. nextNumber = 1

4. updateSequence("RCH", "2025", 1)
5. DNA = "RCH25001"
6. releaseLock("RCH", "2025")

Giuseppe (14:30:16):

1. acquireLock("RCH", "2025", "GIU") → WAIT (Marco has lock)
2. acquireLock("RCH", "2025", "GIU") → SUCCESS (after 0.5 sec)
3. getSequence("RCH", "2025") → 1 (updated by Marco)
4. nextNumber = 2
5. updateSequence("RCH", "2025", 2)
6. DNA = "RCH25002"
7. releaseLock("RCH", "2025")

RESULT: Zero collisions, sequence automatically correct

🛠 COMPLETE FILEMAKER SCRIPTS

SCRIPT 1: Generate_DNA

applescript

```
# GENERATE_DNA Script
# Input: $processType, $ownerCode, $parentDNA

# Step 1: Prepare variables
Set Variable [ $currentYear ; Year(Get(CurrentDate)) ]
Set Variable [ $yearString ; Right("00" & ($currentYear - 2000) ; 2) ]
Set Variable [ $lockKey ; $processType & "_" & $currentYear ]

# Step 2: Acquire lock with retry
Set Variable [ $lockAcquired ; False ]
Loop
  Go to Layout [ "SEQUENCE" ]
  Perform Find [ Find Records: SEQUENCE::LockKey = $lockKey AND SEQUENCE::LockUserID = "" ]
  If [ Get(FoundCount) > 0 ]
    Set Field [ SEQUENCE::LockUserID ; Get(AccountName) ]
    Set Field [ SEQUENCE::LockTimestamp ; Get(CurrentTimestamp) ]
    Set Variable [ $lockAcquired ; True ]
    Exit Loop If [ True ]
  Else
    Pause/Resume Script [ Duration (seconds): 0.1 ]
  End If
End Loop

# Step 3: Get and increment sequence
Set Variable [ $currentSeq ; SEQUENCE::LastNumber ]
Set Variable [ $nextSeq ; $currentSeq + 1 ]
Set Field [ SEQUENCE::LastNumber ; $nextSeq ]

# Step 4: Build DNA
Set Variable [ $seqString ; Right("0000" & $nextSeq ; 4) ]
Set Variable [ $newDNA ; $processType & $yearString & $seqString ]

# Step 5: Create LOG record
Go to Layout [ "LOG" ]
New Record/Request
Set Field [ LOG::ProcessID ; $newDNA ]
Set Field [ LOG::ProcessType ; $processType ]
Set Field [ LOG::ParentProcessID ; $parentDNA ]
Set Field [ LOG::ResponsibleID ; $ownerCode ]
Set Field [ LOG::Status ; "TO_DO" ]
Set Field [ LOG::DateCreated ; Get(CurrentTimestamp) ]

# Step 6: Release lock
Go to Layout [ "SEQUENCE" ]
Perform Find [ Find Records: SEQUENCE::LockKey = $lockKey ]
Set Field [ SEQUENCE::LockUserID ; "" ]
Set Field [ SEQUENCE::LockTimestamp ; "" ]

# Step 7: Return to original layout
Go to Layout [ original layout ]
Exit Script [ Text Result: $newDNA ]
```

SCRIPT 2: Create_Child_Process

```applescript
# CREATE_CHILD_PROCESS Script
# Input: $parentDNA, $childProcessType, $ownerCode

# Step 1: Validate parent exists
Go to Layout [ "LOG" ]
Perform Find [ Find Records: LOG::ProcessID = $parentDNA ]
If [ Get(FoundCount) = 0 ]
    Show Custom Dialog [ "Error: Parent process not found" ]
    Exit Script [ Text Result: "ERROR" ]
End If

# Step 2: Generate child DNA
Perform Script [ "Generate_DNA" ; Parameter: $childProcessType & "¶" & $ownerCode & "¶" & $parentDNA ]
Set Variable [ $childDNA ; Get(ScriptResult) ]

# Step 3: Update hierarchy relationship
Set Field [ LOG::ParentProcessID ; $parentDNA ]

# Step 4: Return child DNA
Exit Script [ Text Result: $childDNA ]
```

PART D: OPERATIONAL BENEFITS WITH KOOL TOOL EXAMPLES

🎯 INFINITE TRACEABILITY

Scenario: LILA COSMETIC client complains about BLONDE_L8 color quality

AUTOMATIC INVESTIGATION via DNA:

1. PROBLEM: BLONDE_L8 color non-compliant
2. Strand DNA: RCH25004
3. Parent RCH25003: Workspace preparation
4. Parent RCH25002: Registration 120 strands
5. Parent RCH25001: Original sample request

6. Parent PRJ25001: LILA project
7. Parent TSK25001: Initial phone call Simona

```
AUTOMATIC RESULT:
├── Sampling responsible: Giuseppe (RCH25004)
├── Preparation responsible: Marco (RCH25003)
├── Registration responsible: Giuseppe (RCH25002)
├── Machine used: Found via timestamp in machine log
├── Spools used: BOB001(3parts) + BOB002(2parts)
├── Spool supplier: Retrieved from spool purchases
└── Dyeing batch: Identified from spool production date
```

INVESTIGATION TIME: 30 seconds vs 3 hours manual

📊 SEMANTIC NAVIGATION

EXAMPLE: Automatic Dashboard for Project

Input: Selection PRJ25001

Automatic output via DNA:

```sql
-- FileMaker Find automatic based on DNA pattern
FIND: LOG::ProcessID = "PRJ25001" OR
    LOG::ParentProcessID LIKE "*PRJ25001*"

RESULT:
├── 1 Project (PRJ25001)
├── 1 Sample Request (RCH25001)
├── 4 RCH Sub-processes (RCH25002-25005)
├── 1 Technical Sheet (TEH25001)
├── 1 Approval (APR25001)
└── 120 Specific Strands (RCH25004 + variants)

TOTAL PROCESSES: 128 automatically connected
```

🔍 AUTOMATIC AGGREGATION

EXAMPLE: Project Cost Report

Automatic query via DNA hierarchy:

```sql
```

```
-- All processes connected to project
GET ALL CHILDREN OF: PRJ25001

-- Automatic cost calculation
FOR EACH child_process:
    cost += process.duration * resource.hourly_rate
    cost += process.materials_used * material.unit_cost
    cost += process.machine_time * machine.hourly_rate

-- Automatic final report
PROJECT PRJ25001 "LILA COSMETIC CHARTS":
├── Phone Call Cost (TSK25001): 15€ (0.5h Simona)
├── Project Cost (PRJ25001): 40€ (1h Luca setup)
├── Sample Cost (RCH25001): 280€ (detail below)
│   ├── RCH25002 Registration: 30€ (1h Giuseppe)
│   ├── RCH25003 Sample Prep: 120€ (4h Marco + materials)
│   ├── RCH25004 120 Strands: 120€ (4h assembly)
│   └── RCH25005 Labels: 10€ (labels + printing)
├── Technical Sheet Cost (TEH25001): 80€ (2h Marco design)
└── Approval Cost (APR25001): 25€ (0.5h control)

TOTAL PROJECT: 440€
MARGIN: 76€ quote - 440€ costs = -364€ (RED!)
ACTION: Review prices or optimize processes
```

PART E: INFINITE STRAND RECIPE MANAGEMENT

🎨 PROBLEM: 120 Strands with Different Recipes

TECHNICAL CHALLENGE:

- Each strand has unique recipe (e.g., BOB001:3parts + BOB002:2parts)
- Infinite possible combinations with available spools
- Traceability of every single strand in final product

DNA SOLUTION for Infinite Recipes:

1. SPECIFIC STRAND DNA:

FORMAT: RCHYYNNN + RECIPE_EXTENSION

EXAMPLES:
├── RCH25006_BLONDE_L8_R001 (first blonde L8 recipe)
├── RCH25007_BROWN_M7_R045 (recipe 45 for brown M7)

├── RCH25008_RED_R4_R123 (recipe 123 for red R4)
└── ... (117 remaining strands)

2. RECIPE TABLE:

```sql
RECIPES TABLE:
├── RecipeCode (Text, Primary Key: BLONDE_L8_R001)
├── ProcessID (Text, FK: RCH25006_BLONDE_L8_R001)
├── ColorName (Text: "Blonde Level 8")
├── ColorCode (Text: "BLONDE_L8")
├── RecipeFormula (Text: "BOB001:3 + BOB002:2")
├── SpoolsUsed (Text: "BOB001,BOB002")
├── Quantities (Text: "3,2")
├── TotalWeight (Number: 5 grams)
├── ProductionTime (Number: 45 seconds)
├── QualityCheck (Text: "OK/KO/PENDING")
├── ClientApproval (Text: "APPROVED/REJECTED/PENDING")
└── Notes (Text: specific details)
```

3. SPOOL TRACEABILITY:

```sql
SPOOL_USAGE TABLE:
├── UsageID (Text, Primary Key auto-generated)
├── RecipeCode (Text, FK: BLONDE_L8_R001)
├── ProcessID (Text, FK: RCH25006_BLONDE_L8_R001)
├── SpoolCode (Text: BOB001, BOB002, etc.)
├── QuantityUsed (Number: 3 parts)
├── SpoolLot (Text: spool production batch)
├── Supplier (Text: dye supplier)
├── DyeDate (Date: dyeing date)
└── Quality (Text: spool quality control)
```

🏭 PRODUCTION WORKFLOW WITH DNA

COMPLETE EXAMPLE: BLONDE_L8 Strand Production

STEP 1 - Client Sample Registration:

DNA: RCH25002
ACTION: Giuseppe registers BLONDE_L8 sample sent by client
RESULT: Image file + target color specifications

STEP 2 - Sampling Preparation:

DNA: RCH25003
Parent: RCH25002
ACTION: Marco prepares workspace for sampling
RESULT: Spools BOB001, BOB002 prepared + tools

STEP 3 - Sample Strand Creation:

DNA: RCH25004_BLONDE_L8_R001
Parent: RCH25003
ACTION: Marco assembles: BOB001(3parts) + BOB002(2parts)
RESULT: BLONDE_L8 sample strand
MACHINE: Assembly_001 (tracked automatically)
TIME: 45 seconds (recorded automatically)

STEP 4 - Labeling:

DNA: RCH25005
Parent: RCH25004_BLONDE_L8_R001
ACTION: Anna prints label with QR code containing DNA
RESULT: Physical label applied to strand
QR_CONTENT: RCH25004_BLONDE_L8_R001

STEP 5 - Client Approval (Web Interface):

DNA: APR25002 Parent: RCH25005 ACTION: Client accesses with browser, views strand, approves/rejects WEB_URL: https://kooltool.eu/approval/APR25002 RESULT: Status = "APPROVED" → Recipe confirmed for production

STEP 6 - Series Production (120 identical strands):

DNA_BATCH: TEH25002 Parent: APR25002 ACTION: Marco produces 120 BLONDE_L8 strands with approved recipe RECIPE: Copy from BLONDE_L8_R001 (BOB001:3 + BOB002:2) RESULT: 120 identical strands for final chart

🔗 MACHINE AND OPERATOR INTEGRATION

COMPLETE TRACEABILITY via DNA:

Machines Used:

```sql
```

```
MACHINE_LOG TABLE:
├── ProcessID (Text, FK: RCH25004_BLONDE_L8_R001)
├── MachineID (Text: ASSEMBLY_001)
├── StartTime (Timestamp: 14:30:15)
├── EndTime (Timestamp: 14:31:00)
├── Duration (Number: 45 seconds)
├── OperatorID (Text: MAR - Marco)
├── QualityCheck (Text: "Tension OK, Color OK")
└── MaintenanceStatus (Text: "Machine OK")
```

Involved Operators:

```sql
OPERATOR_LOG TABLE:
├── ProcessID (Text, FK: RCH25004_BLONDE_L8_R001)
├── OperatorID (Text: MAR)
├── OperatorName (Text: "Marco Rossi")
├── TaskDescription (Text: "BLONDE_L8 strand assembly")
├── SkillLevel (Text: "Expert - Strands")
├── TimeSpent (Number: 45 seconds)
└── QualityRating (Number: 9.5/10)
```

PART F: ADVANCED FILEMAKER SCRIPTS

🚀 SCRIPT 3: Infinite Recipe Management

```applescript
```

```
# CREATE_RECIPE_DNA Script
# Input: $baseProcess, $colorCode, $recipeFormula

# Step 1: Generate recipe code
Set Variable [ $recipeCounter ; Get_Next_Recipe_Number($colorCode) ]
Set Variable [ $recipeCode ; $colorCode & "_R" & Right("000" & $recipeCounter ; 3) ]

# Step 2: Extend base DNA with recipe
Set Variable [ $recipeDNA ; $baseProcess & "_" & $recipeCode ]

# Step 3: Create recipe record
Go to Layout [ "RECIPES" ]
New Record/Request
Set Field [ RECIPES::RecipeCode ; $recipeCode ]
Set Field [ RECIPES::ProcessID ; $recipeDNA ]
Set Field [ RECIPES::ColorCode ; $colorCode ]
Set Field [ RECIPES::RecipeFormula ; $recipeFormula ]

# Step 4: Parse and create spool usage
Set Variable [ $spoolList ; SplitFormula($recipeFormula) ]
Loop
  Exit Loop If [ ValueCount($spoolList) = 0 ]
  Set Variable [ $currentSpool ; GetValue($spoolList ; 1) ]
  Perform Script [ "Create_Spool_Usage" ; Parameter: $recipeDNA & "¶" & $currentSpool ]
  Set Variable [ $spoolList ; RightValues($spoolList ; ValueCount($spoolList) - 1) ]
End Loop

Exit Script [ Text Result: $recipeDNA ]
```

SCRIPT 4: Auto-generation QR Codes

```applescript
```

```
# GENERATE_QR_CODE Script
# Input: $processDNA

# Step 1: Build QR content
Set Variable [ $qrContent ; "https://kooltool.eu/process/" & $processDNA ]

# Step 2: Generate QR image
Set Variable [ $qrImage ; GenerateQR($qrContent) ]

# Step 3: Update process record with QR
Go to Layout [ "LOG" ]
Perform Find [ Find Records: LOG::ProcessID = $processDNA ]
Set Field [ LOG::QRCode ; $qrImage ]

# Step 4: Print label if needed
If [ $printLabel = True ]
   Go to Layout [ "LABEL_LAYOUT" ]
   Print [ Restore ; No dialog ]
End If
```

PART G: TESTING AND VALIDATION

✅ TEST CASES for DNA Generation

TEST 1: Uniqueness under stress

SCENARIO: 10 users create RCH simultaneously
EXPECTED: No collisions, sequences RCH25001-RCH25010
SCRIPT: Multi-user stress test script
VALIDATION: Check SEQUENCE table for duplicates

TEST 2: Hierarchy consistency

SCENARIO: Parent deleted, what happens to children?
EXPECTED: Warning + impossibility of deletion if children exist
SCRIPT: Delete protection script
VALIDATION: Referential integrity maintained

TEST 3: Performance with 1000+ processes

SCENARIO: Database with 1000+ connected processes
EXPECTED: Find and aggregations < 2 seconds
SCRIPT: Performance benchmark script
VALIDATION: Timing log for each operation

🔧 COMMON TROUBLESHOOTING

PROBLEM: "Duplicate DNA generated"

CAUSE: Lock mechanism failed
SOLUTION:

1. Check SEQUENCE table for orphaned locks
2. Clear locks older than 60 seconds
3. Retry generation

PROBLEM: "Parent-child relationship broken"

CAUSE: Parent DNA manually modified
SOLUTION:

1. Never modify DNA manually
2. Use "Fix_Broken_Hierarchy" script for repair
3. Implement DNA change log

PROBLEM: "Performance degraded with many processes"

CAUSE: Missing indexes or overly complex relationships
SOLUTION:

1. Index ProcessID, ParentProcessID, ProcessType
2. Limit found sets with date ranges
3. Use summary fields instead of calculations

CONCLUSION

The hierarchical DNA system transforms KOOL TOOL from "isolated task management" to "orchestration of interconnected processes".

IMMEDIATE BENEFITS:

- **Infinite traceability** without additional effort
- **Automatic reports** based on DNA relationships
- **Ultra-fast debugging** for quality problems
- **Natural scaling** for infinite recipes and processes

NEXT DOCUMENT: Complete Database Architecture with Implementation Schema

KOOL TOOL SRL - Craiova, România

"In the DNA of every process lies the history of the entire company"