Example Analysis Output

Sample Debug Log Scenario

A Salesforce trigger fires when 100 Account records are updated. The trigger has a poorly written loop that queries for related Contacts for each Account individually.

Traditional Debug Log Tool Output

```
EXECUTION_STARTED
CODE_UNIT_STARTED|[EXTERNAL]|01p000000ABC123|AccountTrigger on Account trigger
event AfterUpdate
METHOD_ENTRY|[1]|AccountTriggerHandler.handleAfterUpdate(List<Account>)
SOQL_EXECUTE_BEGIN|[3]|Aggregations:0|SELECT Id, FirstName, LastName FROM Contact
WHERE AccountId = '001000000ABC123'
SOQL_EXECUTE_END|[3]|Rows:12
SOQL_EXECUTE_BEGIN|[3]|Aggregations:0|SELECT Id, FirstName, LastName FROM Contact
WHERE AccountId = '001000000ABC124'
SOQL_EXECUTE_END|[3]|Rows:8
SOQL_EXECUTE_BEGIN|[3]|Aggregations:0|SELECT Id, FirstName, LastName FROM Contact
WHERE AccountId = '001000000ABC125'
SOQL_EXECUTE_END|[3]|Rows:15
... (97 more similar queries)
CUMULATIVE_LIMIT_USAGE
  SOQL_QUERIES: 100/100
  TOTAL_CPU_TIME: 8500/10000
  HEAP_SIZE: 3500000/6000000

ANALYSIS:
- 100 SOQL queries executed
- CPU: 8500ms
- Heap: 3.5MB
- Governor limits approaching maximum
```

What the user sees: Technical jargon, raw numbers, no context about what went wrong or how to fix it.

Our Tool's Output

📋 Summary

☑️ **Transaction completed successfully** but with significant performance issues.
⏱️ **Execution time:** 8.5 seconds

What Your Code Did:

• **Trigger fired:** AccountTrigger (when 100 Account records were updated) • **Called 1 method:** AccountTriggerHandler.handleAfterUpdate • **Talked to the database 100 times** to query Contact records

Performance Assessment:

⚠️ **Your code hit the absolute maximum** number of database queries allowed (100 out of 100)!
⏱️ **Used 85% of processing time** - dangerously close to the limit
💾 **Memory usage:** Moderate (58% used)

Result:

❌ **This code barely worked this time, but will fail if even one more query is added.**

🚨 Issues Detected

1. ⚠️ Critical: Maximum Query Limit Reached

You've used **100 out of 100 allowed queries (100%)**.

What this means:
Salesforce limits how many times you can query the database to keep things running smoothly for everyone. You've hit the absolute maximum! If your code tries to make even one more query, it will fail with an error and your users will see a red error message.

Why this happened:
Your code is asking the database the same question 100 separate times (once for each Account).

---

2. 🔁 Repetitive Query Pattern (N+1 Problem)

Your code is asking the database the same question over and over - this is called an "N+1 query pattern."

The Problem (What Your Code Does):

```
For each Account:
    Ask database: "Give me all Contacts for Account #1"
    Ask database: "Give me all Contacts for Account #2"
    Ask database: "Give me all Contacts for Account #3"
    ... (100 times!)
```

The Analogy:
This is like calling a friend 100 times asking "What's the weather in New York?" instead of asking once and remembering the answer. Or going to the store 100 times to buy one item each time instead of making one trip with a shopping list.

Why It's Slow:
Each database query takes time - even a fast query takes 50-100 milliseconds. When you do 100 queries, that's 5-10 seconds of wasted time just waiting!

---

3. ⏱️ Processing Time Warning

Your code used **8500ms out of 10000ms allowed (85%)**.

What this means:
Salesforce gives your code 10 seconds of "thinking time" to do calculations and process data. You've used 8.5 seconds - that's cutting it very close! If you add any more logic or process a few more records, you'll hit the timeout and your code will fail.

💡 Recommendations

1. 🎯 Fix the N+1 Query Pattern (CRITICAL - Do This First!)

The Problem:
You're querying for Contacts inside a loop, once per Account.

The Solution:
Query for ALL Contacts at once BEFORE the loop, then match them up in memory.

Code Example - Current (Bad):

```
trigger AccountTrigger on Account (after update) {
    for (Account acc : Trigger.new) {
        // ❌ BAD: Query inside loop!
        List<Contact> contacts = [SELECT Id, FirstName, LastName
                                  FROM Contact
                                  WHERE AccountId = :acc.Id];
        // Process contacts...
    }
}
```

Code Example - Fixed (Good):

```
trigger AccountTrigger on Account (after update) {
    // ✅ GOOD: Get all Account IDs first
    Set<Id> accountIds = new Set<Id>();
    for (Account acc : Trigger.new) {
        accountIds.add(acc.Id);
    }

    // ✅ GOOD: Query ONCE for all Contacts
    Map<Id, List<Contact>> contactsByAccount = new Map<Id, List<Contact>>();
    for (Contact con : [SELECT Id, FirstName, LastName, AccountId
                        FROM Contact
                        WHERE AccountId IN :accountIds]) {
        if (!contactsByAccount.containsKey(con.AccountId)) {
            contactsByAccount.put(con.AccountId, new List<Contact>());
        }
        contactsByAccount.get(con.AccountId).add(con);
    }

    // ✅ GOOD: Now loop through and use the Map
    for (Account acc : Trigger.new) {
        List<Contact> contacts = contactsByAccount.get(acc.Id);
        if (contacts != null) {
            // Process contacts...
        }
    }
}
```

Impact of this change:

- **Before:** 100 database queries, 8.5 seconds execution time, hits governor limits
- **After:** 1 database query, ~500ms execution time, uses only 5% of limits
- **Result:** Code is **17x faster** and will never hit limits!

---

2. 📚 Learn About Bulkification

"Bulkification" is a fancy Salesforce term that means **"process multiple records at once instead of one at a time."**

Core Principle:
Never put SOQL queries or DML operations inside loops!

Common Patterns:

1. **Collect IDs first** (like in the example above)
2. **Query once** for all related data
3. **Store in a Map** for quick lookup
4. **Process in memory** instead of repeatedly querying

Resources:

- Salesforce Trailhead: "Apex Triggers" module
- Search: "Salesforce bulkification best practices"
- Search: "Salesforce N+1 query pattern"

---

3. ⚡ Add Safety Measures

Even after fixing the N+1 pattern, add defensive checks:

```
// Add early exit if processing too many records
if (Trigger.new.size() > 200) {
    System.debug('WARNING: Processing ' + Trigger.new.size() + ' records - consider batch processing');
}

// Add LIMIT to queries for safety
List<Contact> contacts = [SELECT Id FROM Contact WHERE AccountId IN :accountIds
LIMIT 50000];
```

☑️ Summary

Current State: ❌ Your code works, but barely. It's at 100% capacity and will fail with any growth.

Priority: 🚨 **CRITICAL** - Fix immediately before users encounter errors.

Effort: 🛠️ **Medium** - Restructure loop to query outside (1-2 hours of work)

Impact: ⚡ **HUGE** - Code will run 17x faster and handle 10x more records safely

Next Steps:

1. Review the fixed code example above
2. Refactor your AccountTriggerHandler to query outside the loop
3. Test with 200 records to ensure it scales
4. Read up on bulkification patterns for future development

🎓 What You Learned

- **Governor Limits:** Salesforce restricts queries/CPU time to protect the platform
- **N+1 Query Pattern:** A common anti-pattern where queries happen inside loops
- **Bulkification:** The solution - process multiple records at once
- **Best Practice:** Always query BEFORE loops, never inside them

You now know how to write high-performance Salesforce code that scales! 🎉