We find stepping through a program [with a debugger] less productive than thinking harder and adding output statements and self-checking code at critical places. ...More important, debugging statements stay with the program; debugging sessions are transient.

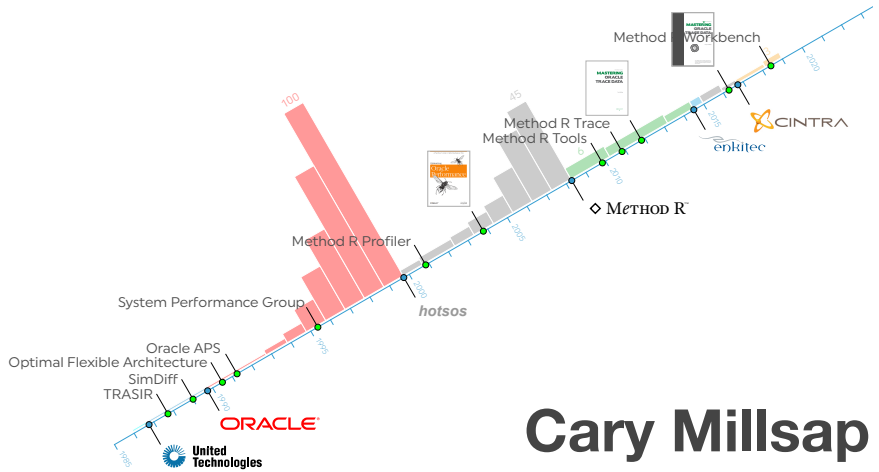
—Brian Kernighan and Rob Pike

The Practice of Programming

Innovative Specifications for Better Performance Logging and Monitoring

Cary Millsap
Cintra Software and Services · Method R Corporation
@CaryMillsap

#gloccle
Great Lakes Oracle Conference, Cleveland, Ohio
11:15a–12:15p Wednesday 15 May 2019



Cary Millsap

Oracle Extended SQL Trace

What you get with SQL trace

Database calls (dbcalls)

System calls (syscalls)

Values bound to placeholders (“bind variables”)

Query plan (row source) operations

SQL trace shows control flow

```
$ vim PSP_ora_4620_00007.trc
```


Tools make it easier to read

1.3. Profile by Statement

This experience executed sixty-three statements Note that *mrprof* aggregates into a single statement all similar but unshareable statements that could have used placeholders in their SQL text but did not. Just one statement consumed 88.6% of the total experience duration. Click the expensive statement to determine where it spent your time.

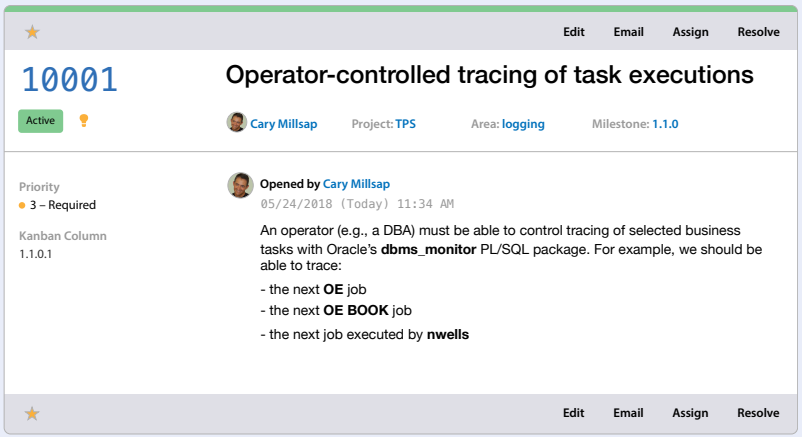
	Statement	Duration seconds	Duration % R	Rows	LIOs (cr+cu)	PIOs (r)	Distinct plans	Distinct cursors	Distinct texts	Statement size lines	Statement size bytes	Optimizer goal
1	[frjd8zfy2jfdq] SELECT executions, end_of_fetch_count, elapsed_t…	97.676370	88.6%	882	·	·	1	10	1	7	514	CHOOSE
2	[8hkasvxxk1u5x] select gsmFormulaOutputTypeMML.Name from Materia…	4.862325	4.4%	45	3,152	·	2	1	437	6	365	ALL_ROWS
3	[5453dc8c0kkvh] select gsmMaterialTypeMML.Name from MaterialSpec…	4.593040	4.2%	75	3,936	·	1	1	437	6	320	ALL_ROWS
4	[2s0d5qpr3qu3k] SELECT Thumbnail, SpecNumber, SpecName, ShortNam…	2.089259	1.9%	437	760	·	1	1	1	40	4,071	ALL_ROWS
5	59 other statements	0.978913	0.9%	3,711	12,972	·	48	166	52	238	15,104	ALL_ROWS, CHOOSE, RULE
6	Total (63)	110.199907	100.0%	5,150	20,820	·	53	179	928	297	20,374	ALL_ROWS, CHOOSE, RULE

1.4. Profile by Cursor

This application code path executed for this experience organized its SQL and PL/SQL statements into 179 cursors. The following grove (multi-root tree) depicts the parent-child relationships among these cursors, listing the most time-consuming cursors in a sibling group first. If a statement appears many times in the grove, then it appeared in many distinct places in the code path. For example, statement frjd8zfy2jfdq › appears in ten distinct places in the code path (only the relevant time-contributing cursors are shown here). The Profile by Statement › section aggregates all the appearances of each statement into one row.

	Cursor statement	Including descendants Duration seconds	Including descendants Duration % R	Including descendants Rows	Including descendants LIOs (cr+cu)	Including descendants PIOs (r)	Excluding descendants Duration seconds	Excluding descendants Duration % R	Excluding descendants Rows	Excluding descendants LIOs (cr+cu)	Excluding descendants PIOs (r)
1	[8hkasvxxk1u5x] select gsmFormulaOutputTypeMML.Name from Materia…	53.427230	48.5%	1,390	4,149	·	4.862325	4.4%	45	3,152	·
2	[frjd8zfy2jfdq] SELECT executions, end_of_fetch_count, elapsed_t…	48.416702	43.9%	437	·	·	48.416702	43.9%	437	·	·
3	[47r1y8yn34jmj] select default$ from col$ where rowid=:1	0.088429	0.1%	437	874	·	0.088429	0.1%	437	874	·

Innovative Specifications



Trace all OE executions

PL/SQL

```
dbms_monitor.serv_mod_act_trace_enable(
    service_name   => 'SYS$USERS',
    module_name    => 'OE',
    action_name    => dbms_monitor.all_actions,
    waits          => true,
    binds          => false,
    plan_stat      => 'FIRST_EXECUTION'
);
```

But this works only if your code sets its module name.

Trace all OE BOOK executions

PL/SQL

```
dbms_monitor.serv_mod_act_trace_enable(
    service_name   => 'SYS$USERS',
    module_name    => 'OE',
    action_name    => 'BOOK',
    waits          => true,
    binds          => false,
    plan_stat      => 'FIRST_EXECUTION'
);
```

This works only if your code sets its module and action names.

Trace all nwells executions

PL/SQL

```
dbms_monitor.client_id_trace_enable(
    client_id      => 'nwells',
    waits          => true,
    binds          => false,
    plan_stat      => 'FIRST_EXECUTION'
);
```

This works only if your code sets its client ID.

User session handle attributes

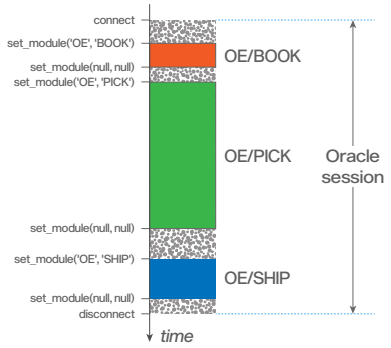
Settable and gettable with `DBMS_APPLICATION_INFO`, OCI, JDBC, ...

Visible in `V$SESSION`

```
SERVICE_NAME
MODULE
ACTION
CLIENT_IDENTIFIER
```

Values can change during a session

Identify sessions or parts of sessions



Marking your code

PL/SQL

```
dbms_session.set_identifier(sys_context('userenv', 'session_user'));
dbms_application_info.set_module('OE', 'BOOK');

-- Your OE BOOK code

dbms_session.set_identifier(null);
dbms_application_info.set_module(null, null);
```


Marking your code

JDBC 12c

```
conn.setClientInfo("OCSID.MODULE",   "OE");
conn.setClientInfo("OCSID.ACTION",   "BOOK");
conn.setClientInfo("OCSID.CLIENTID", getUserName());

/* Your OE BOOK code */

conn.setClientInfo("OCSID.MODULE",   "");
conn.setClientInfo("OCSID.ACTION",   "");
conn.setClientInfo("OCSID.CLIENTID", "");
```


Better factoring

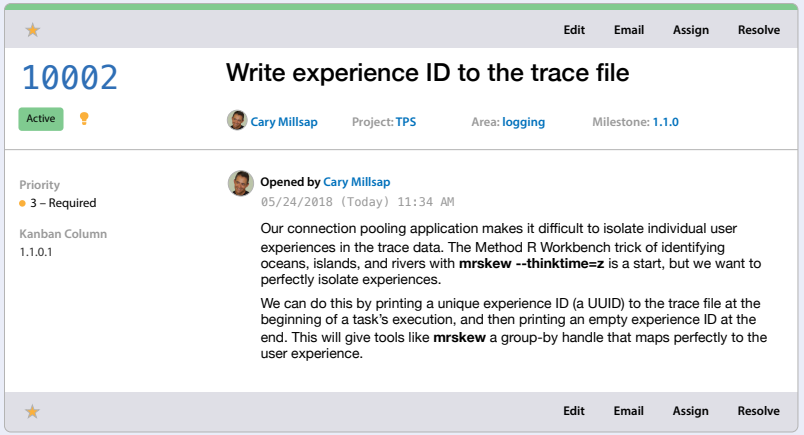
JDBC 12c

```
conn.begin_task("OE BOOK");

/* Your OE BOOK code */

conn.end_task();
```

The new code

pseudocode

```
sub begin_task(t) {
     (mod, act) = split(/ /, t, 2);
     setClientInfo(properties(mod, act, getUserName()));
}

sub end_task() {
     setClientInfo(properties("", "", ""));
}
```

List of experience durations

```
$ mrskew *.trc --group='sprintf("%-8s %-36s %s", $client_id, $experience_id, $file)' …
```

CLIENT-ID	EXPERIENCE-ID	FILE	DURATION	%	CALLS	MEAN	MIN	MAX
nwells	c6a1c359-5766-4263-9e3e-8e600bb7dba9	gxl_ora_131168.trc	9.447247	0.5%	8	1.180906	0.000000	8.075276
cshaftig	822a94c6-d27a-4804-84fe-5a7fed03ed9f	gxl_ora_110636.trc	7.560207	0.4%	9	0.840023	0.000000	7.543424
nwells	733f8c60-2bd3-4fdd-a638-d304f75b1aef	gxl_ora_91144.trc	7.525153	0.4%	15	0.501677	0.000000	7.489669
rmorbisun	afcab6e4-02e8-47ed-a3f8-ca394ddd17cb	gxl_ora_56988.trc	6.342819	0.3%	16	0.396426	0.000000	5.449377
krajita	64e57451-bf65-4022-a927-0cf2567c2fe1	gxl_ora_61704.trc	5.521687	0.3%	7	0.788812	0.000000	5.505351
89,074 others			1,975.121691	98.2%	1,233,058	0.001602	0.000000	4.900948
TOTAL (89,079)			2,011.518804	100.0%	1,233,113	0.001631	0.000000	8.075276

List of experience durations

```
$ mrskew *.trc --group='sprintf("%-8s %-36s %s", $client_id, $experience_id, $file)' …
```

CLIENT-ID	EXPERIENCE-ID	FILE	DURATION	%	CALLS	MEAN	MIN	MAX
nwells	c6a1c359-5766-4263-9e3e-8e600bb7dba9	gxl_ora_131168.trc	9.447247	0.5%	8	1.180906	0.000000	8.075276
cshaftig	822a94c6-d27a-4804-84fe-5a7fed03ed9f	gxl_ora_110636.trc	7.560207	0.4%	9	0.840023	0.000000	7.543424
nwells	733f8c60-2bd3-4fdd-a638-d304f75b1aef	gxl_ora_91144.trc	7.525153	0.4%	15	0.501677	0.000000	7.489669
rmorbisun	afcab6e4-02e8-47ed-a3f8-ca394ddd17cb	gxl_ora_56988.trc	6.342819	0.3%	16	0.396426	0.000000	5.449377
krajita	64e57451-bf65-4022-a927-0cf2567c2fe1	gxl_ora_61704.trc	5.521687	0.3%	7	0.788812	0.000000	5.505351
89,074 others			1,975.121691	98.2%	1,233,058	0.001602	0.000000	4.900948
TOTAL (89,079)			2,011.518804	100.0%	1,233,113	0.001631	0.000000	8.075276

Drill into an experience

```
$ mrskew gxl_ora_131168.trc --where='$experience_id eq "c6a1c359-5766-4263-9e3e-8e600bb7dba9"'
```

CALL-NAME	DURATION	%	CALLS	MEAN	MIN	MAX
enq: TX - row lock contention	8.075276	85.5%	1	8.075276	8.075276	8.075276
log file sync	1.355818	14.4%	1	1.355818	1.355818	1.355818
EXEC	0.015600	0.2%	1	0.015600	0.015600	0.015600
SQL*Net message from client	0.000548	0.0%	1	0.000548	0.000548	0.000548
SQL*Net message to client	0.000005	0.0%	2	0.000002	0.000002	0.000003
2 others	0.000000	0.0%	2	0.000000	0.000000	0.000000
TOTAL (7)	9.447247	100.0%	8	1.180906	0.000000	8.075276

Oceans–Islands–Rivers


```
WAIT … nam='SQL*Net message from client' ela= 1202689 …
```

stuff for Experience A

```
WAIT … nam='SQL*Net message from client' ela= 4260917 …
```

stuff for Experience B

```
WAIT … nam='SQL*Net message from client' ela= 5213365 …
```

stuff for Experience C

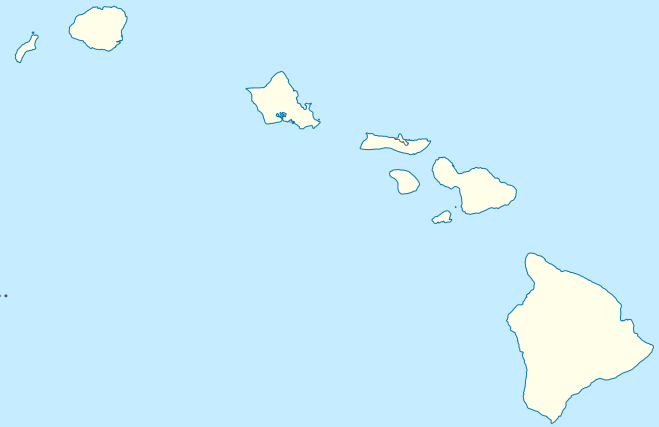
```
WAIT … nam='SQL*Net message from client' ela= 2044420 …
```


Such trace files have

islands of activity

in an ocean of idleness.

But…



An island can have **rivers.**

So can a **trace file.**

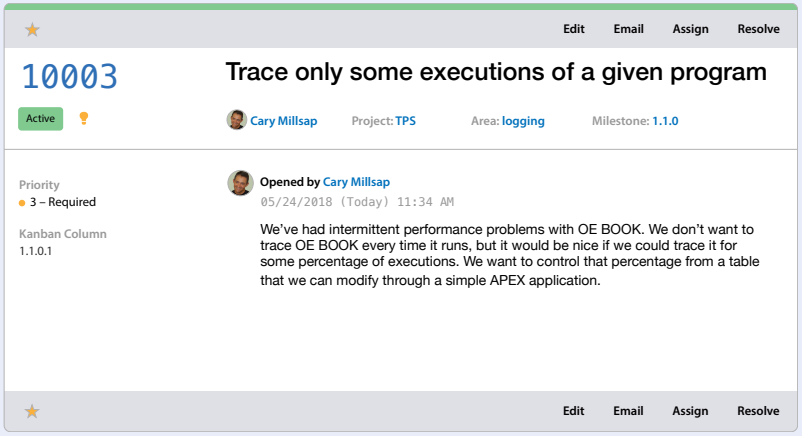

```
WAIT … nam='SQL*Net message from client' ela= 1202689 …
*** CLIENT ID:(nwells)

stuff for Experience A
WAIT … nam='SQL*Net message from client' ela= 342
more stuff for Experience A
WAIT … nam='SQL*Net message from client' ela= 1492
yet more stuff for Experience A
…
WAIT … nam='SQL*Net message from client' ela= 4260917 …
*** CLIENT ID:(cshaftig)

stuff for Experience B
WAIT … nam='SQL*Net message from client' ela= 2928
more stuff for Experience B
…
WAIT … nam='SQL*Net message from client' ela= 5213365 …
*** CLIENT ID:(nwells)

stuff for Experience C
WAIT … nam='SQL*Net message from client' ela= 855
more stuff for Experience C
…
WAIT … nam='SQL*Net message from client' ela= 2044420 …
```

```
WAIT … nam='SQL*Net message from client' ela= 1202689 …
*** CLIENT ID:(nwells)
*** EXPERIENCE ID:(c6a1c359-5766-4263-9e3e-8e600bb7dba9)
stuff for Experience A
WAIT … nam='SQL*Net message from client' ela= 342
more stuff for Experience A
WAIT … nam='SQL*Net message from client' ela= 1492
yet more stuff for Experience A
…
WAIT … nam='SQL*Net message from client' ela= 4260917 …
*** CLIENT ID:(cshaftig)
*** EXPERIENCE ID:(822a94c6-d27a-4804-84fe-5a7fed03ed9f)
stuff for Experience B
WAIT … nam='SQL*Net message from client' ela= 2928
more stuff for Experience B
…
WAIT … nam='SQL*Net message from client' ela= 5213365 …
*** CLIENT ID:(nwells)
*** EXPERIENCE ID:(733f8c60-2bd3-4fdd-a638-d304f75b1aef)
stuff for Experience C
WAIT … nam='SQL*Net message from client' ela= 855
more stuff for Experience C
…
WAIT … nam='SQL*Net message from client' ela= 2044420 …
```


The new code

pseudocode

```
sub begin_task(t) {
    (mod, act) = split(/ /, t, 2);
    setClientInfo(properties(mod, act, getUserName());
    xid = uuid();
    push(xid_stack, (xid, t));
    dbms_log.ksdwrt(1, sprintf("*** EXPERIENCE ID:(%s)", xid));
}

sub end_task() {
    setClientInfo(properties("", "", ""));
    pop(xid_stack);
    dbms_log.ksdwrt(1, "*** EXPERIENCE ID:()");
}
```

Trace every execution

pseudocode

```
    dbms_session.session_trace_enable(true, false, 'FIRST_EXECUTION');

-- Your OE BOOK code
dbms_session.session_trace_disable();
```

Trace only some executions

pseudocode

```
if (should_trace('OE BOOK') {
    dbms_session.session_trace_enable(true, false, 'FIRST_EXECUTION');
}
-- Your OE BOOK code
dbms_session.session_trace_disable();
```


Trace only some executions

pseudocode

```
if (should_trace('OE BOOK')) {
    dbms_session.session_trace_enable(true, false, 'FIRST_EXECUTION');
}
-- Your OE BOOK code
dbms_session.session_trace_disable();
```

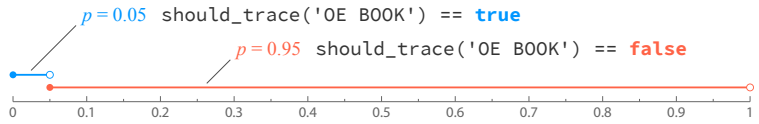
```
sub should_trace(t) {
    select proportion from trace_control where task = :t;
    return (dbms_random.value(0,1) < proportion);
}
```

trace_control

task	proportion
OE BOOK	**0.05**
OE PICK	0.02
…	…

Trace only some executions

pseudocode




```
sub should_trace(t) {
    select proportion from trace_control where task = :t;
    return (dbms_random.value(0,1) < proportion);
}
```

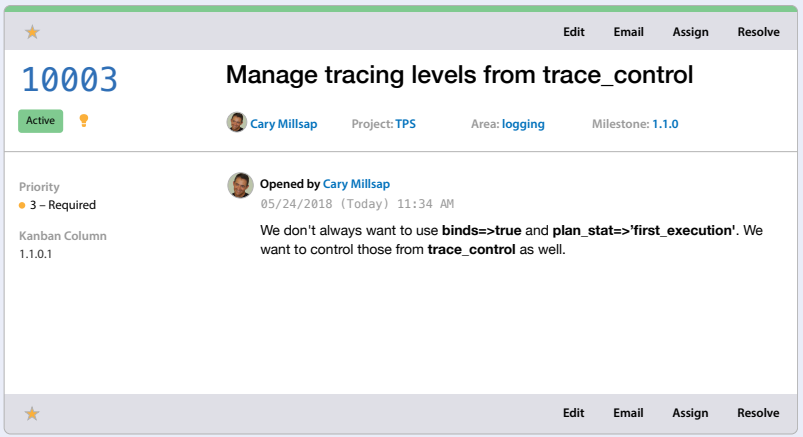
trace_control

task	proportion
OE BOOK	**0.05**
OE PICK	0.02
…	…

The new code

pseudocode

```
sub begin_task(t) {
    (mod, act) = split(/ /, t, 2);
    setClientInfo(properties(mod, act, getUserName());
    xid = uuid();
    push(xid_stack, (xid, t));
    dbms_log.ksdwrt(1, sprintf("*** EXPERIENCE ID:(%s)", xid));
    if (should_trace(t)) {
        session_trace_enable(true, true, 'first_execution');
    }
}

sub end_task() {
    setClientInfo(properties("", "", ""));
    pop(xid_stack);
    dbms_log.ksdwrt(1, "*** EXPERIENCE ID:()");
    session_trace_disable();
}
```



Measurement intrusion

Enough detail, but not too much.

Oracle uses **events** and **levels**. `$ORACLE_HOME/rdbms/mesg/oraus.msg`

- 10046 "enable SQL statement timing" levels 1, 4, 8, 16, 32, 64, ... (**sql_trace**)
- 10053 "CBO Enable optimizer trace"
- 10200 "consistent read buffer status"

How to avoid 10046 MIE

1. `WAITS=>TRUE, BINDS=>FALSE, PLAN_STAT=>'FIRST_EXECUTION'`
2. If application has lightweight row-by-row executions, fix it.
3. `WAITS=>TRUE, BINDS=>TRUE, PLAN_STAT=>'ALL_EXECUTIONS'`

The new code

pseudocode

```
sub begin_task(t) {
    (mod, act) = split(/ /, t, 2);
    setClientInfo(properties(mod, act, getUserName());
    xid = uuid();
    push(xid_stack, (xid, t));
    dbms_log.ksdwrt(1, sprintf("*** EXPERIENCE ID:(%s)", xid));
    if ((bind, stat) = should_trace(t)) {
        session_trace_enable(true, bind, stat);
    }
}

sub end_task() {
    setClientInfo(properties("", "", ""));
    pop(xid_stack);
    dbms_log.ksdwrt(1, "*** EXPERIENCE ID:()");
    session_trace_disable();
}
```

trace_control

task	proportion	bind	stat
OE BOOK	0.05	FALSE	FIRST_EXECUTION
OE PICK	0.02	TRUE	ALL_EXECUTIONS
…	…	…	…

Edit Email Assign Resolve

10004

User controls tracing with **Help › Debug › Trace**

Active

Cary Millsap Project: TPS Area: logging Milestone: 1.1.0

Priority
3 – Required

Kanban Column
1.1.0.1

Opened by Cary Millsap
05/24/2018 (Today) 11:34 AM

When a user has a performance problem with the application, she should be able to click **Help › Debug › Trace**, which will inspire a properly time-scoped trace of the next business task she executes.

Edit Email Assign Resolve

Help › Debug › Trace

How it usually works:

1. User clicks **Help › Debug › Trace**.
2. This triggers immediate session trace enable.
3. User eventually executes her task.
4. Clicks **Help › Debug › Stop Trace**.

Not what you want.

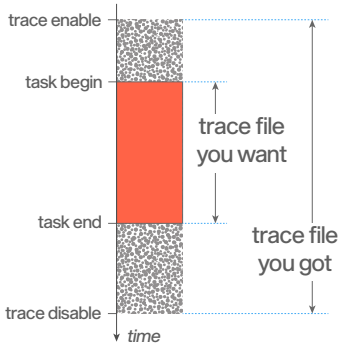
It's not the trace file you want

You need:

trace enable == task begin

trace disable == task end

Otherwise, you get either too much data, or not enough.



Help › Debug › Trace

★☆☆☆☆ A user turns on tracing.

vs.

★★★★★ A user expresses the **intention** to trace the next task execution.

Help › Debug › Trace should set a boolean.

`begin_task` queries the boolean.

The new code

pseudocode

```
sub begin_task(t) {
    (mod, act) = split(/ /, t, 2);
    setClientInfo(properties(mod, act, getUserName());
    xid = uuid();
    push(xid_stack, (xid, t));
    dbms_log.ksdwrt(1, sprintf("*** EXPERIENCE ID:(%s)", xid));
    if (isTraceIntended() or ((bind, stat) = should_trace(t))) {
        session_trace_enable(true, bind, stat);
    }
}

sub end_task() {
    setClientInfo(properties("", "", ""));
    pop(xid_stack);
    dbms_log.ksdwrt(1, "*** EXPERIENCE ID:()");
    session_trace_disable();
}
```


Wrap-up

More specs (discussion)

- Trace the first OE BOOK execution every hour.
- Trace *p*% of Larry's executions, but *q*% of Nancy's.
- Trace *p*% of executions whose durations have one of the top ten variance-to-mean ratios (VMR) in your application.
- Trace only if the job is less than *p*% complete after running for *m* minutes.

Could you do these even if you don't have source code access? How?

Hundreds of specs you could meet

Trace when you need.

Persist every response time.

Predict response time problems.

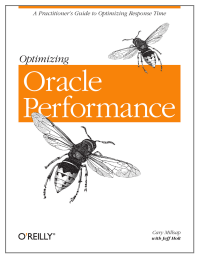
Track pathway through the application.

Write harvesters, uploaders, profilers for the data.

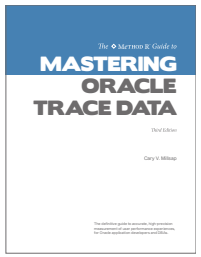
Log to shared memory segments for ultra-high throughput.

You can build anything you want; application self-measurement is no big deal.

For more information...



Available at amazon.com



Available at amazon.com

CINTRA

www.cintra.com