OPTIMIZATION OF SQL QUERIES IN FIREBIRD, PART 3 – PRACTICAL USAGE, TIPS AND TRICKS

Dmitry Yemanov, Firebird
Alexey Kovyazin, IBSurgeon

Firebird Conference 2019

Berlin, 17-19 October

IBSurgeon

How to identify slow queries?

- In Firebird 2.5 and 3.0 use Trace capabilities of Firebird:
  - With standard tools included in Firebird
  - With third-party tools
- In Firebird 2.1, 2.0, 1.5 – IBSurgeon FBScanner

Trace with standard tools (FB3)

- 1. Create trace config file C:\Temp\mytrace1.conf

```
database
{
        enabled = true
        log_statement_finish = true
        log_errors = true
        log_initfini = false
        time_threshold = 10000
        max_sql_length = 65000
}
```

Trace with standard tools

- 2. Run trace session

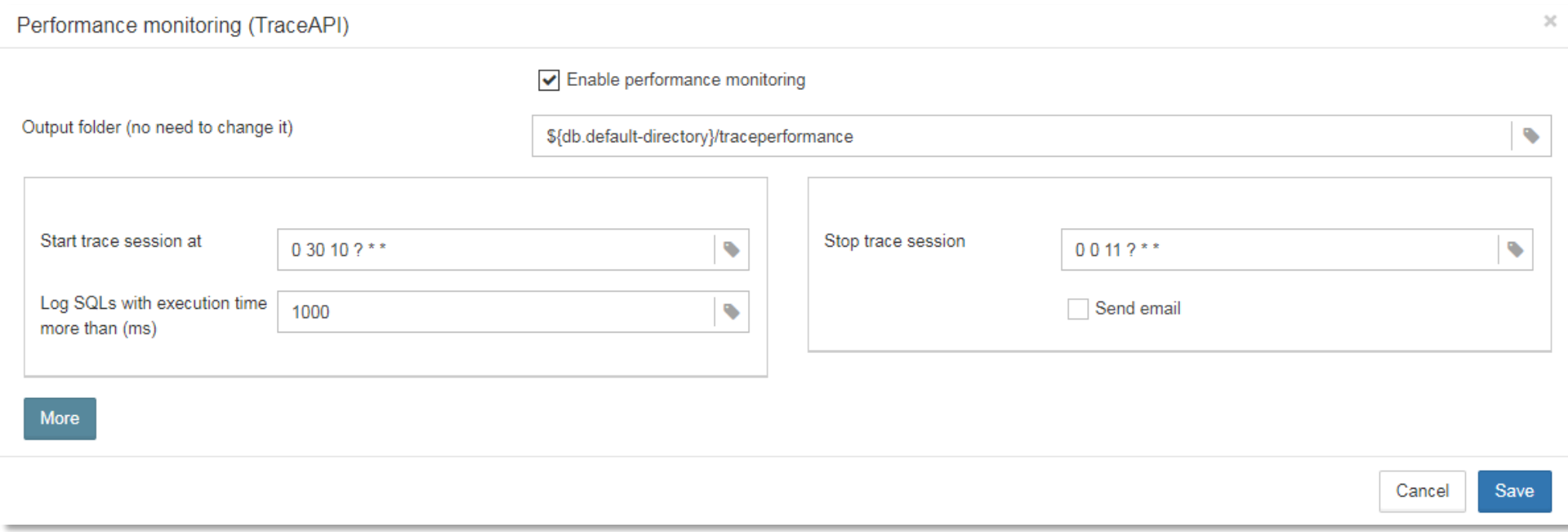
```
fbtracemgr.exe -se localhost:service_mgr -
start -conf "C:\temp\mytrace1.conf" -user
SYSDBA -pass masterkey > output.txt
```

```
3. Result will be the text file which contains
all queries and stored procedures which took
more than 10 seconds
```

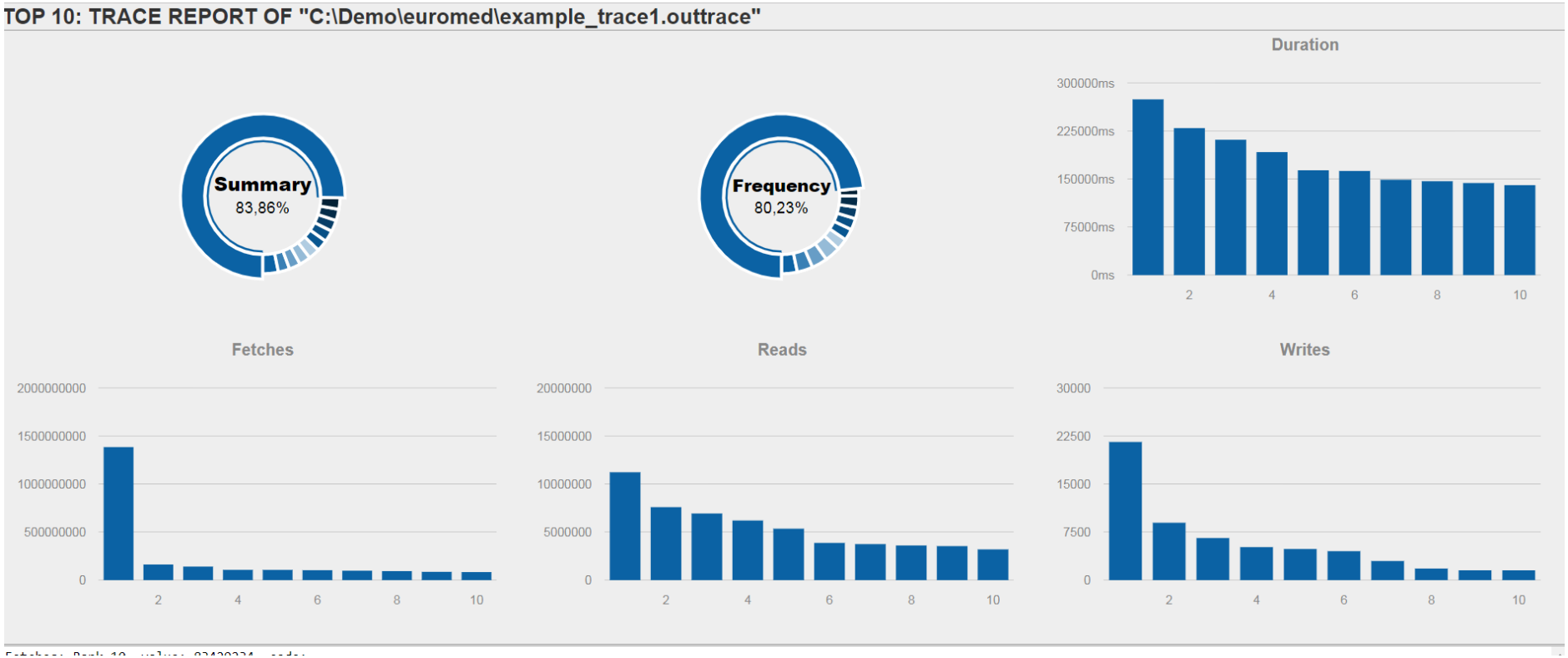
Example of output for 1 stored procedure

```
Statement 827623920:
-------------------------------------------------------------------------------
select * from CF_TREATPLACE_CHECK (?, ?, ?, ?, ?, ?, ?, ?, ?)

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
PLAN (CF_TREATPLACE_CHECK NATURAL)

param0 = bigint, "990003907"
param1 = bigint, "990000026"
param2 = integer, "12"
param3 = bigint, "990011055"
param4 = bigint, "-1"
param5 = bigint, "0"
param6 = integer, "1"
param7 = integer, "0"
param8 = varchar(64), "PDNtp782"

1 records fetched
    885 ms, 11 fetch(es)

Table                             Natural     Index    Update    Insert    Delete   Backout     Purge   Expunge
*******************************************************************************************************************
M_CONFIG                                        2
FILIALS                                         1
```

Trace with HQbird



HQbird Performance report



How to identify the reason of slowness

```
delete from schreclinks where recid in (select ctdid from clrefdet where refid = ?)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
PLAN (CLREFDET INDEX (PK_CLREFDET))
PLAN (SCHRECLINKS NATURAL)
param0 = bigint, "124070276"
param1 = integer, "73"
param2 = integer, "73"
0 records fetched
  29527 ms, 8229 read(s), 125006 fetch(es)
```

Table	Natural	Index	Update	Insert	Delete
RDB$FORMATS		1			
CLREFDET		11957			
SCHRECLINKS	21467				

Natural reads are 3 times cheaper!

- NATURAL 1 read = 1-2 physical reads
- INDEX read = 1-2+(Depth of index) physical reads

Query statistics and resources

- Fetches = CPU rounds (access the cache)
- Reads = Disk reads
- Writes = Disk writes

Slow <> Time

- 36 seconds to fetch 47992 records = good result!
- Always compare real results with the time

```
47992 records fetched
   36359 ms, 73 fetch(es)
```

Indication of the bad index

```
SELECT CONTACT.ID, CONTACT.DBID FROM CONTACT
WHERE UPPER(CONTACT.MSGID)=UPPER(?)
AND CONTACT.KIND IN (0,1)
AND CONTACT.ISDELETE<>?
ORDER BY CONTACT.ID DESC
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

PLAN SORT (CONTACT INDEX (CONTACT_IDX5, CONTACT_IDX5))
param0 = varchar(240), "support@ib-aid.com"
param1 = integer, "2"
1 records fetched
  10047 ms, 1507985 fetch(es)


Table                                    Natural     Index    Update
********************************************************************************
CONTACT                                                1465848
```

Disable wrong index with +0 and create index on expression

```
 SELECT CONTACT.ID, CONTACT.DBID FROM
CONTACT
    WHERE UPPER(CONTACT.MSGID)=UPPER(?)
    AND CONTACT.KIND+0 IN (0,1)
    AND CONTACT.ISDELETE<>?
    ORDER BY CONTACT.ID DESC

CREATE INDEX IXUPPERMSGID ON CONTACT
COMPUTED BY (UPPER(MSGID))
```

After optimization it becomes 200ms!

More complex query statistics example

```
22 records fetched
  21557 ms, 53 read(s), 6907416 fetch(es)


Table                                    Natural     Index    Update    Insert    Delete
********************************************************************************
CLIENTS                                               44
INCOM                                             159288
ORDERS                                              2275
TREAT                                 5877193
DOCTOR                                                 6
JPPAYMENTS                                          2655
CLAVANS                                              800
LOSECREDIT                                           119
```

Now let's look at the query

```
 select  d.*, c.histnum, Case when
(d.locktype = 1 and ? = 2) or (? = 1) then
'' else d.phones end phones_ext
from SPDolgCalc (?, ?, ?, ?, ?, ?, ?, ?) d
    left join clients c on (d.pcode =
c.pcode)
    where (d.DolgUE >= 0.01 OR d.DolgUE <= -
0.01) and  ((coalesce(d.hfamily, 0) = 0) or
(d.IsFamily = 0))
```

- **It does not contain table TREAT!**

How to identify problem in stored procedure?

- Stored procedures does not show correct plan (NATURAL in FB3)
- Stored procedures can be nested

```
SP1
SP2
SP3
SP4
SP10
SP3
SP6
```

Debugging stored procedure with Trace

- 1. Create trace config file c:\temp\mysp1.txt:

```
database
{
        enabled = true
        log_procedure_start = true
        Log_procedure_finish = true
        log_initfini = false
        time_threshold = 0
        max_sql_length = 65000
}
```

Debugging stored procedure with Trace

- 2. Run trace

- **fbtracemgr.exe -se localhost:service_mgr -start -conf "C:\temp\mysp1.conf" -user SYSDBA -pass masterkey > outputSP.txt**

- **3. Run stored procedure in dev tool or in the single-thread application**

Debugging stored procedure with Trace

- Result will be detailed execution of all nested stored procedures with times

```
SP1                1000ms
SP2              500ms
SP3        499ms
SP4        1ms
SP10               500ms
SP3        499ms
SP6        499ms
```

- It will be able to understand which nested procedure is a problem and concentrate on it.

TIPS AND TRICKS

Avoid repetitive reading-1

```
select field_1 from some_table where id = :p1 into var_1;
select field_1 from some_table where id = :p2 into var_2;
select field_1 from some_table where id = :p3 into var_3;
select field_1 from some_table where id = :p4 into var_4;
```

- Instead of 4 queries we can do the single

```
select
     max(iif(id=1,x, null))
    ,max(iif(id=2,x, null))
    ,max(iif(id=3,x, null))
    ,max(iif(id=4,x, null))
from test
where id in (1,2,3,4)
into var_1, var_2, var_3, var_4;
```

Avoid repetitive reading -2

```
CREATE TABLE INV (…, QTY INTEGER, STATUS VARCHAR(4));

SELECT
(SELECT sum(QTY) FROM INV WHERE STATUS='SOLD') SOLD,
(SELECT sum(QTY) FROM INV WHERE STATUS='CNC') CANCEL,
(SELECT sum(QTY) FROM INV WHERE STATUS='INPR') INPROC
FROM STATUSES;
```

This query reads INV 3 times.

Avoid repetitive reading-2

- Better do with EXECUTE BLOCK or STORED PROCEDURE – will be 1 read of INV table

```
FOR SELECT QTY, STATUS FROM INV
INTO :current_qty, :current_status
DO BEGIN
 If(current_status='SOLD') THEN
     SOLD=SOLD+current_qty;
 If(current_status='CNC') THEN
     CNC=CNC+current_qty;
 If(current_status='INPR') THEN
     INPROC=INPROC+current_qty;
END
```

Avoid DISTINCT (if not needed)

- DISTINCT never uses index for sorting!

```
select distinct e.job_code from employee e
==> plan natural

select e.job_code from employee e group by 1
==> PLAN (E ORDER RDB$FOREIGN9)
```

GROUP BY unknown things

- GROUP BY <field_with_DESC_index> does not use index - until Firebird 4 Beta 1(CORE-4529).

- GROUP BY <field_collate_unicode_ci> never uses index – (CORE-4787)

Avoid unnecessary sorting in stored procedures

```
create or alter procedure NEW_PROCEDURE
returns (SUMX double precision)
as
declare variable _amount double precision;
begin
for select T1.amount from Table1 t1 where ....
    order by id
    into :_amount
    do
    begin
     sumx=sumx+_amount
    end;
  suspend;
end
```

Don’t use COUNT() to check existence

```
SELECT * FROM T1
WHERE (select count(t2.id) from T2 where
T1.id=t2.fkid)>0
```

This query will count all records in T2.

Much faster with Exists():

```
WHERE exists (select t2.id from T2 where
T1.id=t2.fkid)
```

This query will read only 1 record

Don’t use LEFT JOIN when unnecessary

```
SELECT * FROM T1 LEFT JOIN T2 ON ()
WHERE T2.CONDITION
```

- In this case, condition applied to T2 means we can use INNER JOIN:


```
SELECT * FROM T1 LEFT JOIN T2 ON ()
WHERE T2.CONDITION
```

COMPUTED BY

- If you are using

```
UPPER(Field1) LIKE UPPER('BlaBla1%')
```

- don't' forget to create INDEX on expression

```
CREATE INDEX ix1 on T1 COMPUTED BY
(UPPER(Field1))
```

UPDATE or INSERT with complex logic

- Usually
- UPDATE OR INSERT

- Sometimes need to check additional things

```
update test set f01 = :val_for_f01 where id
= :val_for_pk;
if ( row_count  = 0 ) then
   insert into test (id, f01) values(
:val_for_pk, :val_for_f01 );
```

Trick to sort wide result sets-1

```
CREATE TABLE T1(I1 integer,
Name varchar(50),
Notes varchar(10000));

SELECT i1, name, Notes FROM t1
ORDER BY name

Full size of sorted record=
(SELECT fields+ORDER BY fields)
```

The query will be very slow due to wide result set.

Trick to sort wide result sets-2

```
with c (id, name) as
(
  select id, name from T1
 order by name
)
select
      c.id
     ,c.name
     ,x.notes from c
join T1 as x on x.id = c.id
```

Indices for MAX(), MIN()

- Indices in Firebird are uni-directional
- For MIN – ASC index
- For MAX – DESC index

Too deep indices

- Better don’t use index than use index with depth 4-5-6 ☺

IN -> EXISTS() or IN->JOINS

select … from T1 where T1.id IN (select T2.id FROM T2 WHERE T2.Condition)

SELECT… FROM T1 WHERE EXISTS(select T2.id FROM T2 WHERE T1.id=T2.ID and T2.Condition)

SELECT… FROM T1 JOIN T2 ON(T1.id=t2.id and t2.Condition)

NOT IN <> NOT EXISTS()!

- NOT IN is not equivalent to NOT EXISTS, due to different NULL handling
- If there are no NULLs, it is possible to switch NOT IN to NOT EXISTS

Runtime plan in Firebird 3

select * from employee
where (emp_no = :param) or (:param is null)
where (emp_no = :param) or (:param = 0)

Old plan in 2.5

PLAN (EMPLOYEE NATURAL)

New plan

PLAN (EMPLOYEE NATURAL, EMPLOYEE INDEX (RDB$PRIMARY7))

Firebird 3 selects plan in runtime

Select Expression
    -> Filter
        -> **Condition**
            -> **Table "EMPLOYEE" Full Scan**
            -> **Table "EMPLOYEE" Access By ID**
                -> Bitmap
                    -> Index "RDB$PRIMARY7" Unique Scan

select * from employee
where emp_no in (1, 2, 3)

PLAN (EMPLOYEE INDEX (RDB$PRIMARY7, RDB$PRIMARY7, RDB$PRIMARY7))

Select Expression
    -> Filter
        -> Table "EMPLOYEE" Access By ID
            -> **Bitmap Or**
                -> **Bitmap Or**
                    -> Bitmap
                        -> Index "RDB$PRIMARY7" Unique Scan
                    -> Bitmap
                        -> Index "RDB$PRIMARY7" Unique Scan
                -> Bitmap
                    -> Index "RDB$PRIMARY7" Unique Scan

How to optimize IN (1,2,3..)

If index is unique (PK, UK), even 500 searches will be fast

If index is non-unique, it is much slower

Field in (1,3,4)

- Uses index 3 times: 1 bitmap, 3 scans

Field+0 in (1,3,4)

- Disables index

Field+0 in (1,3,4) and (field between 1 and 4)

- Enables 1 Range Scan for Between!

Simple hints

- Force index use (not recommended)
  - Where field > 0
- Disable index with expression
  - where field+0 > 5
  - order by *expression or number*
  - group by *expression or number*

Sequence of conditions in Where

- Where A = 1 and B = 5
- Calculated from left to right
- How to check
  - select * from employee where 1=1 or 1/0=0 – no error
  - select * from employee where 1/0=0 or 1=1 – integer divide by zero

- where B in (select …) and A = 1
- is wrong, better use
- where A=1 and B in (select …)

- Easy calculations at the left, heavy at the right

SELECT in COMPUTED BY

- create table A(…
  fld **computed by (select** fld1 from stored_proc…),

- Plan will change for various columns
- Better use triggers

LIKE, STARTING WITH

- FIELD LIKE ‘a%’
  - Uses index
- FIELD LIKE ‘%a’
  - Not uses index
- FIELD LIKE ‘%a%’
  - Not uses index
- FIELD LIKE :param
  - Not uses index!

- FIELD STARTING WITH ‘a’
  - = FIELD LIKE ‘a%’

More optimization tips and tricks

- 45 Ways To Speed Up Firebird
  - https://ib-aid.com/en/articles/45-ways-to-speed-up-firebird-database/
- 23 More Ways To Speed Up Firebird
  - https://ib-aid.com/en/articles/23-more-ways-to-speed-up-firebird/

Thank you!

Questions? ak@ib-aid.com